
Ansible 2.2 Documentation

发布 *devel*

Ansible, Inc

三月 04, 2020

1	Ansible 简介	1
1.1	安装指引手册	2
1.2	Ansible 移植指南	14
1.3	用户指南	98
1.4	Ansible Community Guide	481
1.5	Developer Guide	544
1.6	Public Cloud Guides	714
1.7	Network Technology Guides	789
1.8	Virtualization and Containerization Guides	816
1.9	Network Automation Getting Started	856
1.10	Network Automation Advanced Topics	885
1.11	Network Automation Developer Guide	980
1.12	Galaxy User Guide	997
1.13	Galaxy Developer Guide	1012
1.14	Controlling how Ansible behaves: precedence rules	1018
1.15	YAML Syntax	1022
1.16	Python 3 Support	1027
1.17	Interpreter Discovery	1029
1.18	Release and maintenance	1030
1.19	Testing Strategies	1033
1.20	Frequently Asked Questions	1039
1.21	Glossary	1053
1.22	Ansible Reference: Module Utilities	1060
1.23	Special Variables	1060
1.24	Red Hat Ansible Tower	1063
1.25	Ansible Automation Hub	1063
1.26	Logging Ansible output	1064

1.27 Ansible Roadmap	1064
索引	1079

CHAPTER 1

Ansible 简介

Ansible 是一款 IT 自动化工具。主要应用场景有配置系统、软件部署、持续发布及不停服平滑滚动更新的高级任务编排。

Ansible 本身非常简单易用，同时注重安全和可靠性，以最小化变动为特色，使用 OpenSSH 实现数据传输（如果有需要的话也可以使用其它传输模式或者 pull 模式），其语言设计非常利于人类阅读，即使是针对不刚接触 Ansible 的新手来讲亦是如此。

我们坚信无论什么范围的环境，简单都是必须的，所以我们的设计尽可能满足各类型的繁忙人群：开发人员、系统管理员、发布工程师、IT 管理员等所有类型的人。同时，Ansible 适用于各种环境，小到几台多到成千上万台的企业实际环境都完全满足。

Ansible 不使用 C/S 架构管理节点，即没有 Agent 。这样的架构使得 Ansible 不会存在如何升级远程 Agent 管理进程或者因为没有安装 Agent 而无法管理系统。因为 OpenSSH 是非常流行的开源组件，安全问题也非常少。Ansible 的去中心化管理方式深受业内认可，即它只依赖 OS 的 KEY 认证访问远程主机。如需，Ansible 可以便捷接入 Kerberos, LDAP 或者其它认证系统。

该文档覆盖了 Ansible 所有版本的文档，可以通过左边栏索引跳转对应页面「别跳了，就翻译了一个版本，官方更新太快了」。官方并行管理多个版本 Ansible 及对应版本的文档，所以使用前请确认参考的是对应版本的文档。最新的功能，我们会在版本中标识新增的功能。

Ansible 主版本大概每年 3-4 个版本。核心功能应用的发布会很谨慎，每个版本都非常重视代码质量和语言简洁性。但是，社区贡献的新模块和组件发展的非常快，每个版本都会合入很多的新模块。

1.1 安装指引手册

Ansible 安装指引!

1.1.1 安装 Ansible

该页面展示如何在不同平台安装 Ansible。Ansible 不用安装客户端，通过 SSH 协议管理远程机器。一旦安装，Ansible 不需要数据库，也不需要后台保持运行。一旦安装（一台简单的笔记本也可以安装）完成，它可以管理整个集群。当 Ansible 管理远程机器时，它不需要软件保持安装状态或者运行状态。因此，当 Ansible 的升级很简单，只需要迁移到新版本即可。

- 准备工作
 - 管理机环境要求
 - 受管节点环境要求
- 指定安装具体的版本
- 安装 *Ansible on RHEL, CentOS, or Fedora*
- 安装 *Ansible on Ubuntu*
- 安装 *Ansible on Debian*
- 安装 *Ansible on Gentoo with portage*
- 安装 *Ansible on FreeBSD*
- 安装 *Ansible on macOS*
- 安装 *Ansible on Solaris*
- 安装 *Ansible on Arch Linux*
- 安装 *Ansible on Slackware Linux*
- 安装 *Ansible on Clear Linux*
- 安装 *Ansible with pip*
- 从源码运行 *Ansible (devel)*
- 安装指定版本的 *Ansible*
- *Ansible* 命令行补全功能
 - 安装 *argcomplete on RHEL, CentOS, or Fedora*
 - 安装 *argcomplete with apt*
 - 安装 *argcomplete with pip*

- *Configuring `argcomplete`*
 - * *Globally*
 - * *Per command*
- *`argcomplete` with `zsh` or `tcsh`*
- *Ansible on GitHub*

准备工作

准备一台管理机，该管理机可通过 SSH 连接到你所有的被管理机。

管理机环境要求

运行 Ansible 的服务器必须且只需要安装有 Python 2.7+ 或者 Python 3.5+。Red Hat, Debian, CentOS, macOS, 任一 BSD 系列的系统均可。但 ‘Windows’ 不能用于管理机。

选择管理机时，需要注意的时，网络条件越好越便于管理。比如：当你选择在云上使用 Ansible 时，那么管理机和管理节点都在云上最佳选择，连接外网的节点速度则会慢很多，也存在很大的安全风险。

注解： macOS 系统默认的文件句柄数比较小，如果你希望并发 15 个或者更多线程，你需要通过如下命令增加系统文件句柄打开上限。`sudo launchctl limit maxfiles unlimited`。不然，过多的线程数会提示报错：Too many open files 。

警告： 需要注意的是，部分模块或者组件在使用时需要额外安装插件。管理节点需要安装必要的插件后方可正常被管理。具体请参考对应的模块文档

受管节点环境要求

受管节点需要和外界正常通信，默认使用 SSH 协议。默认使用 SFTP 。如果 SFTP 无法使用，你可以在 `ansible.cfg` 中将其修改为 SCP 。同样，受管机需要有 Python 2.6+ 或 Python 3.5 以上的环境

注解：

- 如果受管机开启了 SELinux，你需要安装 `libselinux-python` ，不然 `copy/file/template` 等任何相关联的功能都无法使用。你可以使用 `yum module` 或 `dnf module` 在受管机安装软件。
- 默认情况下 Ansible 使用 `/usr/bin/python` 下的 Python 解释器运行命令。但部分 Linux 发行版可能只安装了 Python 3 解释器，可以从 `/usr/bin/python3` 找到。如果遇到类似如下的错误，表示你需要检查 Python 环境

```
"module_stdout": "/bin/sh: /usr/bin/python: No such file or directory\r\n"
```

你可以设置仓库文件对应亦是 `ansible_python_interpreter` (参考 *Inventory 使用进阶*) 指明解释器位置, 或者干脆安装一个 Python 2 的解释器。如果 Python 2 的解释器没有安装在指定目录 `/usr/bin/python`。那你依然需要修改仓库文件指定解释器的位置。 `ansible_python_interpreter`

- Ansible 的 raw module 模块和 script module 不依赖受管机的 Python 环境。因此, 从技术角度上讲, 我可以使用 Ansible 这两个模块 (raw module 和 script module) 编译安装 Python 环境。举例如下: 你想在 RHEL 系列系统上安装 Python 2 环境, 请参考如下命令:

```
$ ansible myhost --become -m raw -a "yum install -y python2"
```

指定安装具体的版本

具体安装哪个版本取决于你的需求。你可以选择如下的任何一种方式来安装 Ansible:

- 使用系统默认的包管理器安装 (for Red Hat Enterprise Linux (TM), CentOS, Fedora, Debian, or Ubuntu).
- Install with pip (Python 包管理器).
- 源码安装 devel 版本的 Ansible w 体验最新版本的功能

注解: 只有当你希望修改 Ansible 引擎或者尝试修改码源时, 你才会需要安装 devel 版本, 因为 devel 版本是非稳定版本, 变化非常快。

Ansible 每年发布 2-3 个新版本。得益于发布周期短, 小 BUGS 通常在下一个版本修复而不会在稳定分支上保留。主要 BUGS 如有需要会使用专门的维护分支, 当然, 这种情况并不多见。

安装 Ansible on RHEL, CentOS, or Fedora

On Fedora:

```
$ sudo dnf install ansible
```

On RHEL and CentOS:

```
$ sudo yum install ansible
```

RPMs for RHEL 7 and RHEL 8 参考 [Ansible Engine repository](#).

RHEL 8 开启 repository :


```
$ sudo subscription-manager repos --enable ansible-2.9-for-rhel-8-x86_64-rpms
```

RHEL 7 开启 repository :

```
$ sudo subscription-manager repos --enable rhel-7-server-ansible-2.9-rpms
```

RHEL, CentOS, and Fedora 的最新 RPM 版本获取方式: [EPEL](#) as well as [releases.ansible.com](#).

Ansible 2.4+ 可以管理包含 Python 2.6 或更高版本的早期操作系统。

你也可以编译自己的 RPM 包:

```
$ git clone https://github.com/ansible/ansible.git
$ cd ./ansible
$ make rpm
$ sudo rpm -Uvh ./rpm-build/ansible-*.noarch.rpm
```

安装 Ansible on Ubuntu

Ubuntu builds are available [in a PPA here](#).

配置 PPA 或者安装 Ansible:

```
$ sudo apt update
$ sudo apt install software-properties-common
$ sudo apt-add-repository --yes --update ppa:ansible/ansible
$ sudo apt install ansible
```

注解: 旧的 Ubuntu 发行版 “software-properties-common” 名字是 “python-software-properties” . 使用 `apt-get` 而不是 `apt` 。同时, 只有比较新的发行版才有 (i.e. 18.04, 18.10, etc.) `-u` or `--update` 参数。根据情况调整你的脚本。

Debian/Ubuntu 也可以从源码编译:

```
$ make deb
```

如果您希望从源头开始获得开发分支, 请参考接下来的介绍

安装 Ansible on Debian

Debian 用户可以使用和 Ubuntu PPA 一样的源。

增加如下行到 `/etc/apt/sources.list`:

```
deb http://ppa.launchpad.net/ansible/ansible/ubuntu trusty main
```

执行如下命令:

```
$ sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 93C4A3FD7BB9C367
$ sudo apt update
$ sudo apt install ansible
```

注解: 该方法已在 Debian Jessie 和 Stretch 中的 Trusty 来源中得到验证,但在早期版本中可能不受支持。旧版本中使用 `apt-get` 而不是 `apt`。

安装 Ansible on Gentoo with portage

```
$ emerge -av app-admin/ansible
```

要安装最新版本,可能需要先屏蔽 Ansible 软件包,然后再进行开发:

```
$ echo 'app-admin/ansible' >> /etc/portage/package.accept_keywords
```

安装 Ansible on FreeBSD

尽管 Ansible 可以工作在 Python 2 或 3 版本,但 FreeBSD 每个版本有不同的名字,安装方式如下:

```
$ sudo pkg install py27-ansible
```

or:

```
$ sudo pkg install py36-ansible
```

你也可以使用 ports 安装:

```
$ sudo make -C /usr/ports/sysutils/ansible install
```

同样可以指定版本安装,如 `ansible25`

旧版本的 FreeBSD 使用 `pkg` 管理方式 (具体看你使用什么包管理工具):

```
$ sudo pkg install ansible
```

安装 Ansible on macOS

Mac 安装 Ansible 推荐使用 pip

具体文档请参考[安装 Ansible with pip](#)。如果你的 macOS 系统是 10.12+，你最好升级到最新的 pip 版本，pip 必须作为模块在 macOS 运行，具体参考如上文档。

安装 Ansible on Solaris

参考 SysV package from OpenCSW.

```
# pkgadd -d http://get.opencsw.org/now
# /opt/csw/bin/pkgutil -i ansible
```

安装 Ansible on Arch Linux

通用仓库包含有 Ansible，直接安装即可

```
$ pacman -S ansible
```

AUR 的 [ansible-git](#). 拥有一个 PKGBUILD 功能，可以直接从 GitHub 拉取数据。

参考 ArchWiki: [Ansible](#)。

安装 Ansible on Slackware Linux

Ansible 编译脚本的 repository：[SlackBuilds.org](#)。编译安装参考：[sbopkg](#)。

使用 Ansible 和所有依赖创建队列

```
# sqg -p ansible
```

从创建的队列文件构建和安装软件包（如 sbopkg 提示是否应使用队列或软件包，输入 Q）：

```
# sbopkg -k -i ansible
```

安装 Ansible on Clear Linux

Linux 发行包版本的软件包中默认带有 Ansible 及其依赖包

```
$ sudo swupd bundle-add sysadmin-hostmgmt
```

使用 swupd 工具包升级软件

```
$ sudo swupd update
```

安装 Ansible with pip

Ansible 可以使用 Python 包管理器 pip 安装。但 macOS 因为 openssl 协议要求的原因, pip 和 *nix 的使用和其它系统会有一些区别, pip 以模块的方式运行。(英文原文: It should be noted that macOS requires a slightly different use of pip than *nix due to openssl requirements, therefore pip must be run as a module.) 如果 pip 事先没有安装, 使用如下命令安装

```
$ curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
$ python get-pip.py --user
```

安装 Ansible¹:

```
$ pip install --user ansible
```

macOS 系统不需要需要 sudo 或者额外安装其它补丁包, 只需要 pip 安装即可:

```
$ python -m pip install --user ansible
```

如果希望安装开发版本:

```
$ pip install --user git+https://github.com/ansible/ansible.git@devel
```

For macOS:

```
$ python -m pip install --user git+https://github.com/ansible/ansible.git@devel
```

如果你使用的是 macOS Mavericks (10.9), 编译的时候可能会有一些 warning 。建议额外声明如下编译变量:

```
$ CFLAGS=-Qunused-arguments CPPFLAGS=-Qunused-arguments pip install --user ansible
```

如果希望使用 paramiko 插件或者模块依赖 paramiko, 安装方式如下²:

```
$ pip install --user paramiko
```

For macOS:

```
$ python -m pip install --user paramiko
```

Ansible 也可以安装在 Python 虚拟环境管理器 virtualenv 指定的环境中

¹ 如果你在 macOS 上安装 “pycrypto” 有问题, 尝试指定 CC=clang sudo -E pip install pycrypto

² paramiko was included in Ansible's requirements.txt prior to 2.8.

```
$ python -m virtualenv ansible # Create a virtualenv if one does not already exist
$ source ansible/bin/activate # Activate the virtual environment
$ pip install ansible
```

如果想全局安装 Ansible

```
$ sudo python get-pip.py
$ sudo pip install ansible
```

注解： 需要注意的是使用 `sudo pip` 是全局安装的模式。由于 `pip` 无法与系统管理包器协调，所以如果使用 `sudo` 模式可能会改变系统环境，导致系统无法运行，macOS 系统尤其容易发生该问题。如果你非专门人士，对系统全局文件功能完全了解前，建议使用 `--user` 参数。

注解： 旧版本的 `pip` 默认网址是 <http://pypi.python.org/simple>，现在已经不再维护。安装 Ansible 前请升级 `pip` 到最新版本。

如果你已经安装的旧版本的 `pip`，升级文档可参考 <https://pip.pypa.io/en/stable/installing/#upgrading-pip>。‘_’。

从源码运行 Ansible (devel)

注解： 只有当你正在修改 Ansible 引擎或者尝试修改 Ansible 源码时，你才需要使用 `devel` 版本。`devel` 分支会随时改变，功能也不稳定。

从源码编译安装 Ansible 也很容易，不需要 `root` 权限，也不需要事先安装软件，更不需要保持后台运行或者设置数据库。

注解： 如果你希望使用 Ansible Tower 作为管理节点，不要使用源码编译安装。请使用 OS 管理工具（比如：`apt` or `yum` 或 `pip`）安装稳定版。

源码安装：

```
$ git clone https://github.com/ansible/ansible.git
$ cd ./ansible
```

“git”下载后 Ansible 后，设置环境变量：

Using Bash:

```
$ source ./hacking/env-setup
```

Using Fish:

```
$ source ./hacking/env-setup.fish
```

If you want to suppress spurious warnings/errors, use:

```
$ source ./hacking/env-setup -q
```

请保证已经事先安装了 Python 包管理工具 `pip`:

```
$ curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
$ python get-pip.py --user
```

使用如下命令安装依赖组件¹:

```
$ pip install --user -r ./requirements.txt
```

如果希望更改 Ansible checkout 的版本, 建议使用 `pull-with-rebase` 保留原始版本。

```
$ git pull --rebase
```

```
$ git pull --rebase #same as above
$ git submodule update --init --recursive
```

一旦开始运行 `env-setup` 脚本, 默认表示你使用的 `inventory` 仓库文件是 `/etc/ansible/hosts`。当然你也可以参考这篇文件指定 `inventory` 仓库 ([Inventory 使用进阶](#)):

```
$ echo "127.0.0.1" > ~/ansible_hosts
$ export ANSIBLE_INVENTORY=~/ansible_hosts
```

关于 `inventory` 从这里可以了解到更多内容[Inventory 使用进阶](#)。

执行第一条命令:

```
$ ansible all -m ping --ask-pass
```

也可以尝试 “`sudo make install`”。

安装指定版本的 Ansible

如果想打包 Ansible 或者自己构建一个本地的包但又不想 `git checkout`, 可以从这个页面下载指定的版本包安装 [Ansible downloads page](#).

这些包同样也放在 Ansible 的发行版仓库中 [git repository](#) 。

Ansible 命令行补全功能

Ansible 从 2.9 版本开始支持命令行补全功能，但需要安装 `argcomplete` 插件。`argcomplete` 完全支持 `bash`，部分功能支持 `zsh` `tcsh`。

你可以从 RedHat 的 EPEL 源直接安装 `python-argcomplete`，或者从其它发行版的标准 OS repo 库安装。更多安装配置信息请参考 [argcomplete documentation](#)。

安装 `argcomplete` on RHEL, CentOS, or Fedora

On Fedora:

```
$ sudo dnf install python-argcomplete
```

On RHEL and CentOS:

```
$ sudo yum install epel-release
$ sudo yum install python-argcomplete
```

安装 `argcomplete` with apt

```
$ sudo apt install python-argcomplete
```

安装 `argcomplete` with pip

```
$ pip install argcomplete
```

Configuring `argcomplete`

有 2 种方式配置 `argcomplete` 使 Ansible 支持 shell 命令补全：全局模式（`globally`）或者单个命令（`per command`）。

Globally

全局模式需要 Bash 的版本是 4.2

```
$ sudo activate-global-python-argcomplete
```

这条命令将生成 bash 补全文件到全局配置默认目录。可以使用 `--dest` 指定位置。

Per command

如果 bash 的版本不是 4.2，那必须独立声明注册每个脚本。

```
$ eval $(register-python-argcomplete ansible)
$ eval $(register-python-argcomplete ansible-config)
$ eval $(register-python-argcomplete ansible-console)
$ eval $(register-python-argcomplete ansible-doc)
$ eval $(register-python-argcomplete ansible-galaxy)
$ eval $(register-python-argcomplete ansible-inventory)
$ eval $(register-python-argcomplete ansible-playbook)
$ eval $(register-python-argcomplete ansible-pull)
$ eval $(register-python-argcomplete ansible-vault)
```

如果希望永久有效，如上命令需要写入到环境变量文件，`~/.profile` or `~/.bash_profile` .

argcomplete with zsh or tcsh

zsh 或者 tcsh 命令补全功能请参考 [argcomplete documentation](#).

Ansible on GitHub

Ansible 的 GitHub 地址 [GitHub project](#) 。我们可以在该地址提交 issue, bugs, 或者产品功能想法。

参见:

ad-hoc [命令操作指引](#) Examples of basic commands

Working With Playbooks Learning ansible' s configuration management language

How do I handle the package dependencies required by Ansible package dependencies during Ansible install

Ansible Installation related to FAQs

[Mailing List](#) Questions? Help? Ideas? Stop by the list on Google Groups

[irc.freenode.net](#) #ansible IRC chat channel

1.1.2 配置 Ansible

Topics

- 配置 *Ansible*
 - 配置文件
 - * 获取最新配置
 - 环境配置
 - 命令行选项

本页内容介绍如何配置 Ansible .

配置文件

主要的核心配置都在 (ansible.cfg)，其中大多数默认配置满足大部分用户的需求，参考文档中列出了搜索配置文件的路径 [reference documentation](#)。

获取最新配置

如果通过包管理工具安装的，那最新的 `ansible.cfg` 默认存放在 `/etc/ansible` 中，也有可能因为重复安装或升级，文件是以 `.rpmnew` 结尾

如果是通过 pip 或者源码包编译安装 Ansible，你需要手动创建或者覆盖原有配置。

案例参考：[example file is available on GitHub](#).

更多更全配置信息可参考：[configuration_settings](#). 从 2.4 版本开始，你可以使用 `ansible-config` 命令行列出可用选项和变量信息，并且可以检查当前配置。

更详细信息请参考 [ansible_configuration_settings](#).

环境配置

Ansible 配置同样支持使用系统变量。如果系统环境设置了，会覆盖 Ansible 的配置。

更详细信息请参考 [ansible_configuration_settings](#).

命令行选项

Ansible 并没有把所有的配置都展示在命令中，只是列出最有用或最通用的部分。在命令行指定的配置优先级比从配置文件中读取的优化级要高，也即会覆盖从配置文件中读取的环境变量。

更详细信息请参考 [ansible-playbook](#) and [ansible](#).

1.2 Ansible 移植指南

移植指南可以帮助您升级 Ansible，还有其配套的剧本，插件和其他部分。

请注意，列表并不完整。如果您有其它更好的信息，则可以通过单击右上角的“在 GitHub 上编辑”编辑或提交问题。

> # 本页内容重要程度较低，放在最后翻译

1.2.1 Ansible 2.10 Porting Guide

This section discusses the behavioral changes between Ansible 2.9 and Ansible 2.10.

It is intended to assist in updating your playbooks, plugins and other parts of your Ansible infrastructure so they will work with this version of Ansible.

We suggest you read this page along with [Ansible Changelog for 2.10](#) to understand what updates you may need to make.

This document is part of a collection on porting. The complete list of porting guides can be found at [porting guides](#).

Topics

- *Ansible 2.10 Porting Guide*
 - *Playbook*
 - *Command Line*
 - *Deprecated*
 - *Modules*
 - * *Modules removed*
 - * *Deprecation notices*
 - * *Noteworthy module changes*
 - *Plugins*
 - * *Lookup plugin names case-sensitivity*
 - * *Noteworthy plugin changes*
 - *Porting custom scripts*
 - *Networking*

Playbook

No notable changes

Command Line

No notable changes

Deprecated

- Windows Server 2008 and 2008 R2 will no longer be supported or tested in the next Ansible release, see *Are Server 2008, 2008 R2 and Windows 7 supported?*.
- The win_stat module has removed the deprecated `get_md55` option and `md5` return value.
- The win_psexec module has removed the deprecated `extra_opts` option.

Modules

Modules removed

The following modules no longer exist:

- letsencrypt use `acme_certificate` instead.

Deprecation notices

The following modules will be removed in Ansible 2.14. Please update your playbooks accordingly.

- `ldap_attr` use `ldap_attrs` instead.
- `vyos_static_route` use `vyos_static_routes` instead.

The following functionality will be removed in Ansible 2.14. Please update update your playbooks accordingly.

- The `openssl_csr` module's option `version` no longer supports values other than 1 (the current only standardized CSR version).
- `docker_container`: the `trust_image_content` option will be removed. It has always been ignored by the module.
- `iam_managed_policy`: the `fail_on_delete` option will be removed. It has always been ignored by the module.
- `s3_lifecycle`: the `requester_pays` option will be removed. It has always been ignored by the module.
- `s3_sync`: the `retries` option will be removed. It has always been ignored by the module.

- The return values `err` and `out` of `docker_stack` have been deprecated. Use `stdout` and `stderr` from now on instead.
- `cloudformation`: the `template_format` option will be removed. It has been ignored by the module since Ansible 2.3.
- `data_pipeline`: the `version` option will be removed. It has always been ignored by the module.
- `ec2_eip`: the `wait_timeout` option will be removed. It has had no effect since Ansible 2.3.
- `ec2_key`: the `wait` option will be removed. It has had no effect since Ansible 2.5.
- `ec2_key`: the `wait_timeout` option will be removed. It has had no effect since Ansible 2.5.
- `ec2_lc`: the `associate_public_ip_address` option will be removed. It has always been ignored by the module.
- `ec2_tag`: Support for `list` as a state has been deprecated. The `ec2_tag_info` can be used to fetch the tags on an EC2 resource.
- `iam_policy`: the `policy_document` option will be removed. To maintain the existing behavior use the `policy_json` option and read the file with the `lookup` plugin.
- `redfish_config`: the `bios_attribute_name` and `bios_attribute_value` options will be removed. To maintain the existing behavior use the `bios_attributes` option instead.
- `clc_aa_policy`: the `wait` parameter will be removed. It has always been ignored by the module.
- `redfish_config`, `redfish_command`: the behavior to select the first System, Manager, or Chassis resource to modify when multiple are present will be removed. Use the new `resource_id` option to specify target resource to modify.
- `win_domain_controller`: the `log_path` option will be removed. This was undocumented and only related to debugging information for module development.
- `win_package`: the `username` and `password` options will be removed. The same functionality can be done by using `become: yes` and `become_flags: logon_type=new_credentials logon_flags=netcredentials_only` on the task.
- `win_package`: the `ensure` alias for the `state` option will be removed. Please use `state` instead of `ensure`.
- `win_package`: the `productid` alias for the `product_id` option will be removed. Please use `product_id` instead of `productid`.

The following functionality will change in Ansible 2.14. Please update update your playbooks accordingly.

- The `docker_container` module has a new option, `container_default_behavior`, whose default value will change from `compatibility` to `no_defaults`. Set to an explicit value to avoid deprecation warnings.
- The `docker_container` module's `network_mode` option will be set by default to the name of the first network in `networks` if at least one network is given and `networks_cli_compatible` is `true`

(will be default from Ansible 2.12 on). Set to an explicit value to avoid deprecation warnings if you specify networks and set `networks_cli_compatible` to `true`. The current default (not specifying it) is equivalent to the value `default`.

- `ec2`: the `group` and `group_id` options will become mutually exclusive. Currently `group_id` is ignored if you pass both.
- `iam_policy`: the default value for the `skip_duplicates` option will change from `true` to `false`. To maintain the existing behavior explicitly set it to `true`.
- `iam_role`: the `purge_policies` option (also know as `purge_policy`) default value will change from `true` to `false`
- `elb_network_lb`: the default behaviour for the `state` option will change from `absent` to `present`. To maintain the existing behavior explicitly set state to `absent`.
- `vmware_tag_info`: the module will not return `tag_facts` since it does not return multiple tags with the same name and different category id. To maintain the existing behavior use `tag_info` which is a list of tag metadata.

The following modules will be removed in Ansible 2.14. Please update your playbooks accordingly.

- `vmware_dns_config` use `vmware_host_dns` instead.

Noteworthy module changes

- The `datacenter` option has been removed from `vmware_guest_find`
- The options `ip_address` and `subnet_mask` have been removed from `vmware_vmkernel`; use the sub-options `ip_address` and `subnet_mask` of the `network` option instead.
- Ansible modules created with `add_file_common_args=True` added a number of undocumented arguments which were mostly there to ease implementing certain action plugins. The undocumented arguments `src`, `follow`, `force`, `content`, `backup`, `remote_src`, `regexp`, `delimiter`, and `directory_mode` are now no longer added. Modules relying on these options to be added need to specify them by themselves.
- The `AWSRetry` decorator no longer catches `NotFound` exceptions by default. `NotFound` exceptions need to be explicitly added using `catch_extra_error_codes`. Some AWS modules may see an increase in transient failures due to AWS' s eventual consistency model.
- `vmware_datastore_maintenancemode` now returns `datastore_status` instead of Ansible internal key `results`.
- `vmware_host_kernel_manager` now returns `host_kernel_status` instead of Ansible internal key `results`.
- `vmware_host_ntp` now returns `host_ntp_status` instead of Ansible internal key `results`.

- `vmware_host_service_manager` now returns `host_service_status` instead of Ansible internal key `results`.
- `vmware_tag` now returns `tag_status` instead of Ansible internal key `results`.
- The deprecated `recurse` option in `pacman` module has been removed, you should use `extra_args=--recursive` instead.
- `vmware_guest_custom_attributes` module does not require VM name which was a required parameter for releases prior to Ansible 2.10.
- `zabbix_action` no longer requires `esc_period` and `event_source` arguments when `state=absent`.
- `zabbix_proxy` deprecates `interface` sub-options `type` and `main` when proxy type is set to passive via `status=passive`. Make sure these suboptions are removed from your playbook as they were never supported by Zabbix in the first place.
- `gitlab_user` no longer requires `name`, `email` and `password` arguments when `state=absent`.
- `win_pester` no longer runs all `*.ps1` file in the directory specified due to it executing potentially unknown scripts. It will follow the default behaviour of only running tests for files that are like `*.tests.ps1` which is built into Pester itself
- **`win_find` has been refactored to better match the behaviour of the `find` module. Here is what has changed**
 - When the directory specified by `paths` does not exist or is a file, it will no longer fail and will just warn the user
 - Junction points are no longer reported as `islnk`, use `isjunction` to properly report these files. This behaviour matches the `win_stat`
 - Directories no longer return a `size`, this matches the `stat` and `find` behaviour and has been removed due to the difficulties in correctly reporting the size of a directory
- `docker_container` no longer passes information on non-anonymous volumes or binds as `Volumes` to the Docker daemon. This increases compatibility with the `docker` CLI program. Note that if you specify `volumes: strict` in `comparisons`, this could cause existing containers created with `docker_container` from Ansible 2.9 or earlier to restart.
- `docker_container`'s support for port ranges was adjusted to be more compatible to the `docker` command line utility: a one-port container range combined with a multiple-port host range will no longer result in only the first host port be used, but the whole range being passed to Docker so that a free port in that range will be used.
- `purefb_fs` no longer supports the deprecated `nfs` option. This has been superceeded by `nfsv3`.
- `nxos_igmp_interface` no longer supports the deprecated `oif_prefix` and `oif_source` options. These have been superceeded by `oif_ps`.

- `aws_s3` can now delete versioned buckets even when they are not empty - set mode to delete to delete a versioned bucket and everything in it.
- The parameter `message` in `grafana_dashboard` module is renamed to `commit_message` since `message` is used by Ansible Core engine internally.
- The parameter `message` in `datadog_monitor` module is renamed to `notification_message` since `message` is used by Ansible Core engine internally.
- The parameter `message` in `bigpanda` module is renamed to `deployment_message` since `message` is used by Ansible Core engine internally.

Plugins

Lookup plugin names case-sensitivity

- Prior to Ansible 2.10 lookup plugin names passed in as an argument to the `lookup()` function were treated as case-insensitive as opposed to lookups invoked via `with_<lookup_name>`. 2.10 brings consistency to `lookup()` and `with_` to be both case-sensitive.

Noteworthy plugin changes

- The `hashi_vault` lookup plugin now returns the latest version when using the KV v2 secrets engine. Previously, it returned all versions of the secret which required additional steps to extract and filter the desired version.
- Some undocumented arguments from `FILE_COMMON_ARGUMENTS` have been removed; plugins using these, in particular action plugins, need to be adjusted. The undocumented arguments which were removed are `src`, `follow`, `force`, `content`, `backup`, `remote_src`, `regex`, `delimiter`, and `directory_mode`.

Porting custom scripts

No notable changes

Networking

No notable changes

1.2.2 Ansible 2.9 Porting Guide

This section discusses the behavioral changes between Ansible 2.8 and Ansible 2.9.

It is intended to assist in updating your playbooks, plugins and other parts of your Ansible infrastructure so they will work with this version of Ansible.

We suggest you read this page along with [Ansible Changelog for 2.9](#) to understand what updates you may need to make.

This document is part of a collection on porting. The complete list of porting guides can be found at [porting guides](#).

Topics

- *Ansible 2.9 Porting Guide*
 - *Playbook*
 - * *Inventory*
 - * *Loops*
 - *Command Line*
 - *Deprecated*
 - *Collection loader changes*
 - *Modules*
 - * *Renaming from `_facts` to `_info`*
 - * *Writing modules*
 - * *Modules removed*
 - * *Deprecation notices*
 - *Renamed modules*
 - * *Noteworthy module changes*
 - *Plugins*
 - * *Removed Lookup Plugins*
 - *Porting custom scripts*
 - *Networking*
 - * *Network resource modules*
 - * *Improved `gather_facts` support for network devices*
 - * *Top-level connection arguments removed in 2.9*

Playbook

Inventory

- `hash_behaviour` now affects inventory sources. If you have it set to `merge`, the data you get from inventory might change and you will have to update playbooks accordingly. If you're using the default setting (`overwrite`), you will see no changes. Inventory was ignoring this setting.

Loops

Ansible 2.9 handles “unsafe” data more robustly, ensuring that data marked “unsafe” is not templated. In previous versions, Ansible recursively marked all data returned by the direct use of `lookup()` as “unsafe”, but only marked structured data returned by indirect lookups using `with_X` style loops as “unsafe” if the returned elements were strings. Ansible 2.9 treats these two approaches consistently.

As a result, if you use `with_dict` to return keys with templatable values, your templates may no longer work as expected in Ansible 2.9.

To allow the old behavior, switch from using `with_X` to using `loop` with a filter as described at [Migrating from with_X to loop](#).

Command Line

- The location of the Galaxy token file has changed from `~/.ansible_galaxy` to `~/.ansible/galaxy_token`. You can configure both path and file name with the `galaxy_token_path` config.

Deprecated

No notable changes

Collection loader changes

The way to import a PowerShell or C# module util from a collection has changed in the Ansible 2.9 release. In Ansible 2.8 a util was imported with the following syntax:

```
#AnsibleRequires -CSharpUtil AnsibleCollections.namespace_name.collection_name.util_
↪ filename
#AnsibleRequires -PowerShell AnsibleCollections.namespace_name.collection_name.util_
↪ filename
```

In Ansible 2.9 this was changed to:

```
#AnsibleRequires -CSharpUtil ansible_collections.namespace_name.collection_name.plugins.  
↪module_utils.util_filename  
#AnsibleRequires -PowerShell ansible_collections.namespace_name.collection_name.plugins.  
↪module_utils.util_filename
```

The change in the collection import name also requires any C# util namespaces to be updated with the newer name format. This is more verbose but is designed to make sure we avoid plugin name conflicts across separate plugin types and to standardise how imports work in PowerShell with how Python modules work.

Modules

- The `win_get_url` and `win_uri` module now sends requests with a default `User-Agent` of `ansible-httpget`. This can be changed by using the `http_agent` key.
- The `apt` module now honors `update_cache=false` while installing its own dependency and skips the cache update. Explicitly setting `update_cache=true` or omitting the param `update_cache` will result in a cache update while installing its own dependency.

Renaming from `_facts` to `_info`

Ansible 2.9 renamed a lot of modules from `<something>_facts` to `<something>_info`, because the modules do not return *Ansible facts*. Ansible facts relate to a specific host. For example, the configuration of a network interface, the operating system on a unix server, and the list of packages installed on a Windows box are all Ansible facts. The renamed modules return values that are not unique to the host. For example, account information or region data for a cloud provider. Renaming these modules should provide more clarity about the types of return values each set of modules offers.

Writing modules

- Module and `module_utils` files can now use relative imports to include other `module_utils` files. This is useful for shortening long import lines, especially in collections.

Example of using a relative import in collections:

```
# File: ansible_collections/my_namespace/my_collection/plugins/modules/my_module.py  
# Old way to use an absolute import to import module_utils from the collection:  
from ansible_collections.my_namespace.my_collection.plugins.module_utils import my_  
↪util  
# New way using a relative import:  
from ..module_utils import my_util
```

Modules and module_utils shipped with Ansible can use relative imports as well but the savings are smaller:

```
# File: ansible/modules/system/ping.py
# Old way to use an absolute import to import module_utils from core:
from ansible.module_utils.basic import AnsibleModule
# New way using a relative import:
from ...module_utils.basic import AnsibleModule
```

Each single dot (.) represents one level of the tree (equivalent to ../ in filesystem relative links).

参见:

The Python Relative Import Docs go into more detail of how to write relative imports.

Modules removed

The following modules no longer exist:

- Apstra' s aos_* modules. See the new modules at <https://github.com/apstra>.
- ec2_ami_find use ec2_ami_facts instead.
- kubernetes use k8s instead.
- nxos_ip_interface use nxos_l3_interface instead.
- nxos_portchannel use nxos_linkagg instead.
- nxos_switchport use nxos_l2_interface instead.
- oc use k8s instead.
- panos_nat_policy use panos_nat_rule instead.
- panos_security_policy use panos_security_rule instead.
- vsphere_guest use vmware_guest instead.

Deprecation notices

The following modules will be removed in Ansible 2.13. Please update update your playbooks accordingly.

- cs_instance_facts use cs_instance_info instead.
- cs_zone_facts use cs_zone_info instead.
- digital_ocean_sshkey_facts use digital_ocean_sshkey_info instead.
- eos_interface use eos_interfaces instead.
- eos_l2_interface use eos_l2_interfaces instead.

- `eos_l3_interface` use `eos_l3_interfaces` instead.
- `eos_linkagg` use `eos_lag_interfaces` instead.
- `eos_lldp_interface` use `eos_lldp_interfaces` instead.
- `eos_vlan` use `eos_vlans` instead.
- `ios_interface` use `ios_interfaces` instead.
- `ios_l2_interface` use `ios_l2_interfaces` instead.
- `ios_l3_interface` use `ios_l3_interfaces` instead.
- `ios_vlan` use `ios_vlans` instead.
- `iosxr_interface` use `iosxr_interfaces` instead.
- `junos_interface` use `junos_interfaces` instead.
- `junos_l2_interface` use `junos_l2_interfaces` instead.
- `junos_l3_interface` use `junos_l3_interfaces` instead.
- `junos_linkagg` use `junos_lag_interfaces` instead.
- `junos_lldp` use `junos_lldp_global` instead.
- `junos_lldp_interface` use `junos_lldp_interfaces` instead.
- `junos_vlan` use `junos_vlans` instead.
- `lambda_facts` use `lambda_info` instead.
- `na_ontap_gather_facts` use `na_ontap_info` instead.
- `net_banner` use the platform-specific `[netos]_banner` modules instead.
- `net_interface` use the new platform-specific `[netos]_interfaces` modules instead.
- `net_l2_interface` use the new platform-specific `[netos]_l2_interfaces` modules instead.
- `net_l3_interface` use the new platform-specific `[netos]_l3_interfaces` modules instead.
- `net_linkagg` use the new platform-specific `[netos]_lag` modules instead.
- `net_lldp` use the new platform-specific `[netos]_lldp_global` modules instead.
- `net_lldp_interface` use the new platform-specific `[netos]_lldp_interfaces` modules instead.
- `net_logging` use the platform-specific `[netos]_logging` modules instead.
- `net_static_route` use the platform-specific `[netos]_static_route` modules instead.
- `net_system` use the platform-specific `[netos]_system` modules instead.
- `net_user` use the platform-specific `[netos]_user` modules instead.
- `net_vlan` use the new platform-specific `[netos]_vlans` modules instead.

- `net_vrf` use the platform-specific `[netos]_vrf` modules instead.
- `nginx_status_facts` use `nginx_status_info` instead.
- `nxos_interface` use `nxos_interfaces` instead.
- `nxos_l2_interface` use `nxos_l2_interfaces` instead.
- `nxos_l3_interface` use `nxos_l3_interfaces` instead.
- `nxos_linkagg` use `nxos_lag_interfaces` instead.
- `nxos_vlan` use `nxos_vlans` instead.
- `online_server_facts` use `online_server_info` instead.
- `online_user_facts` use `online_user_info` instead.
- `purefa_facts` use `purefa_info` instead.
- `purefb_facts` use `purefb_info` instead.
- `scaleway_image_facts` use `scaleway_image_info` instead.
- `scaleway_ip_facts` use `scaleway_ip_info` instead.
- `scaleway_organization_facts` use `scaleway_organization_info` instead.
- `scaleway_security_group_facts` use `scaleway_security_group_info` instead.
- `scaleway_server_facts` use `scaleway_server_info` instead.
- `scaleway_snapshot_facts` use `scaleway_snapshot_info` instead.
- `scaleway_volume_facts` use `scaleway_volume_info` instead.
- `vcenter_extension_facts` use `vcenter_extension_info` instead.
- `vmware_about_facts` use `vmware_about_info` instead.
- `vmware_category_facts` use `vmware_category_info` instead.
- `vmware_drs_group_facts` use `vmware_drs_group_info` instead.
- `vmware_drs_rule_facts` use `vmware_drs_rule_info` instead.
- `vmware_dvs_portgroup_facts` use `vmware_dvs_portgroup_info` instead.
- `vmware_guest_boot_facts` use `vmware_guest_boot_info` instead.
- `vmware_guest_customization_facts` use `vmware_guest_customization_info` instead.
- `vmware_guest_disk_facts` use `vmware_guest_disk_info` instead.
- `vmware_host_capability_facts` use `vmware_host_capability_info` instead.
- `vmware_host_config_facts` use `vmware_host_config_info` instead.
- `vmware_host_dns_facts` use `vmware_host_dns_info` instead.

- `vmware_host_feature_facts` use `vmware_host_feature_info` instead.
- `vmware_host_firewall_facts` use `vmware_host_firewall_info` instead.
- `vmware_host_ntp_facts` use `vmware_host_ntp_info` instead.
- `vmware_host_package_facts` use `vmware_host_package_info` instead.
- `vmware_host_service_facts` use `vmware_host_service_info` instead.
- `vmware_host_ssl_facts` use `vmware_host_ssl_info` instead.
- `vmware_host_vmhba_facts` use `vmware_host_vmhba_info` instead.
- `vmware_host_vmnic_facts` use `vmware_host_vmnic_info` instead.
- `vmware_local_role_facts` use `vmware_local_role_info` instead.
- `vmware_local_user_facts` use `vmware_local_user_info` instead.
- `vmware_portgroup_facts` use `vmware_portgroup_info` instead.
- `vmware_resource_pool_facts` use `vmware_resource_pool_info` instead.
- `vmware_target_canonical_facts` use `vmware_target_canonical_info` instead.
- `vmware_vmkernel_facts` use `vmware_vmkernel_info` instead.
- `vmware_vswitch_facts` use `vmware_vswitch_info` instead.
- `vultr_account_facts` use `vultr_account_info` instead.
- `vultr_block_storage_facts` use `vultr_block_storage_info` instead.
- `vultr_dns_domain_facts` use `vultr_dns_domain_info` instead.
- `vultr_firewall_group_facts` use `vultr_firewall_group_info` instead.
- `vultr_network_facts` use `vultr_network_info` instead.
- `vultr_os_facts` use `vultr_os_info` instead.
- `vultr_plan_facts` use `vultr_plan_info` instead.
- `vultr_region_facts` use `vultr_region_info` instead.
- `vultr_server_facts` use `vultr_server_info` instead.
- `vultr_ssh_key_facts` use `vultr_ssh_key_info` instead.
- `vultr_startup_script_facts` use `vultr_startup_script_info` instead.
- `vultr_user_facts` use `vultr_user_info` instead.
- `vyos_interface` use `vyos_interfaces` instead.
- `vyos_l3_interface` use `vyos_l3_interfaces` instead.
- `vyos_linkagg` use `vyos_lag_interfaces` instead.

- `vyos_lddp` use `vyos_lddp_global` instead.
- `vyos_lddp_interface` use `vyos_lddp_interfaces` instead.

The following functionality will be removed in Ansible 2.12. Please update update your playbooks accordingly.

- `vmware_cluster` DRS, HA and VSAN configuration; use `vmware_cluster_drs`, `vmware_cluster_ha` and `vmware_cluster_vsan` instead.

The following functionality will be removed in Ansible 2.13. Please update update your playbooks accordingly.

- `openssl_certificate` deprecates the `assertonly` provider. Please see the `openssl_certificate` documentation examples on how to replace the provider with the `openssl_certificate_info`, `openssl_csr_info`, `openssl_privatekey_info` and `assert` modules.

For the following modules, the PyOpenSSL-based backend `pyopenssl` has been deprecated and will be removed in Ansible 2.13:

- `get_certificate`
- `openssl_certificate`
- `openssl_certificate_info`
- `openssl_csr`
- `openssl_csr_info`
- `openssl_privatekey`
- `openssl_privatekey_info`
- `openssl_publickey`

Renamed modules

The following modules have been renamed. The old name is deprecated and will be removed in Ansible 2.13. Please update update your playbooks accordingly.

- The `ali_instance_facts` module was renamed to `ali_instance_info`.
- The `aws_acm_facts` module was renamed to `aws_acm_info`.
- The `aws_az_facts` module was renamed to `aws_az_info`.
- The `aws_caller_facts` module was renamed to `aws_caller_info`.
- The `aws_kms_facts` module was renamed to `aws_kms_info`.
- The `aws_region_facts` module was renamed to `aws_region_info`.

- The `aws_s3_bucket_facts` module was renamed to `aws_s3_bucket_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `aws_sgw_facts` module was renamed to `aws_sgw_info`.
- The `aws_waf_facts` module was renamed to `aws_waf_info`.
- The `azure_rm_aks_facts` module was renamed to `azure_rm_aks_info`.
- The `azure_rm_aksversion_facts` module was renamed to `azure_rm_aksversion_info`.
- The `azure_rm_applicationsecuritygroup_facts` module was renamed to `azure_rm_applicationsecuritygroup_info`.
- The `azure_rm_appserviceplan_facts` module was renamed to `azure_rm_appserviceplan_info`.
- The `azure_rm_automationaccount_facts` module was renamed to `azure_rm_automationaccount_info`.
- The `azure_rm_autoscale_facts` module was renamed to `azure_rm_autoscale_info`.
- The `azure_rm_availabilityset_facts` module was renamed to `azure_rm_availabilityset_info`.
- The `azure_rm_cdnendpoint_facts` module was renamed to `azure_rm_cdnendpoint_info`.
- The `azure_rm_cdnprofile_facts` module was renamed to `azure_rm_cdnprofile_info`.
- The `azure_rm_containerinstance_facts` module was renamed to `azure_rm_containerinstance_info`.
- The `azure_rm_containerregistry_facts` module was renamed to `azure_rm_containerregistry_info`.
- The `azure_rm_cosmosdbaccount_facts` module was renamed to `azure_rm_cosmosdbaccount_info`.
- The `azure_rm_deployment_facts` module was renamed to `azure_rm_deployment_info`.
- The `azure_rm_resourcegroup_facts` module was renamed to `azure_rm_resourcegroup_info`.
- The `bigip_device_facts` module was renamed to `bigip_device_info`.
- The `bigiq_device_facts` module was renamed to `bigiq_device_info`.
- The `cloudformation_facts` module was renamed to `cloudformation_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `cloudfront_facts` module was renamed to `cloudfront_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `cloudwatchlogs_log_group_facts` module was renamed to `cloudwatchlogs_log_group_info`.
- The `digital_ocean_account_facts` module was renamed to `digital_ocean_account_info`.
- The `digital_ocean_certificate_facts` module was renamed to `digital_ocean_certificate_info`.
- The `digital_ocean_domain_facts` module was renamed to `digital_ocean_domain_info`.
- The `digital_ocean_firewall_facts` module was renamed to `digital_ocean_firewall_info`.

- The `digital_ocean_floating_ip_facts` module was renamed to `digital_ocean_floating_ip_info`.
- The `digital_ocean_image_facts` module was renamed to `digital_ocean_image_info`.
- The `digital_ocean_load_balancer_facts` module was renamed to `digital_ocean_load_balancer_info`.
- The `digital_ocean_region_facts` module was renamed to `digital_ocean_region_info`.
- The `digital_ocean_size_facts` module was renamed to `digital_ocean_size_info`.
- The `digital_ocean_snapshot_facts` module was renamed to `digital_ocean_snapshot_info`.
- The `digital_ocean_tag_facts` module was renamed to `digital_ocean_tag_info`.
- The `digital_ocean_volume_facts` module was renamed to `digital_ocean_volume_info`.
- The `ec2_ami_facts` module was renamed to `ec2_ami_info`.
- The `ec2_asg_facts` module was renamed to `ec2_asg_info`.
- The `ec2_customer_gateway_facts` module was renamed to `ec2_customer_gateway_info`.
- The `ec2_eip_facts` module was renamed to `ec2_eip_info`.
- The `ec2_elb_facts` module was renamed to `ec2_elb_info`.
- The `ec2_eni_facts` module was renamed to `ec2_eni_info`.
- The `ec2_group_facts` module was renamed to `ec2_group_info`.
- The `ec2_instance_facts` module was renamed to `ec2_instance_info`.
- The `ec2_lc_facts` module was renamed to `ec2_lc_info`.
- The `ec2_placement_group_facts` module was renamed to `ec2_placement_group_info`.
- The `ec2_snapshot_facts` module was renamed to `ec2_snapshot_info`.
- The `ec2_vol_facts` module was renamed to `ec2_vol_info`.
- The `ec2_vpc_dhcp_option_facts` module was renamed to `ec2_vpc_dhcp_option_info`.
- The `ec2_vpc_endpoint_facts` module was renamed to `ec2_vpc_endpoint_info`.
- The `ec2_vpc_igw_facts` module was renamed to `ec2_vpc_igw_info`.
- The `ec2_vpc_nacl_facts` module was renamed to `ec2_vpc_nacl_info`.
- The `ec2_vpc_nat_gateway_facts` module was renamed to `ec2_vpc_nat_gateway_info`.
- The `ec2_vpc_net_facts` module was renamed to `ec2_vpc_net_info`.
- The `ec2_vpc_peering_facts` module was renamed to `ec2_vpc_peering_info`.
- The `ec2_vpc_route_table_facts` module was renamed to `ec2_vpc_route_table_info`.
- The `ec2_vpc_subnet_facts` module was renamed to `ec2_vpc_subnet_info`.

- The `ec2_vpc_vgw_facts` module was renamed to `ec2_vpc_vgw_info`.
- The `ec2_vpc_vpn_facts` module was renamed to `ec2_vpc_vpn_info`.
- The `ecs_service_facts` module was renamed to `ecs_service_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ecs_taskdefinition_facts` module was renamed to `ecs_taskdefinition_info`.
- The `efs_facts` module was renamed to `efs_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `elasticache_facts` module was renamed to `elasticache_info`.
- The `elb_application_lb_facts` module was renamed to `elb_application_lb_info`.
- The `elb_classic_lb_facts` module was renamed to `elb_classic_lb_info`.
- The `elb_target_facts` module was renamed to `elb_target_info`.
- The `elb_target_group_facts` module was renamed to `elb_target_group_info`.
- The `gcp_bigquery_dataset_facts` module was renamed to `gcp_bigquery_dataset_info`.
- The `gcp_bigquery_table_facts` module was renamed to `gcp_bigquery_table_info`.
- The `gcp_cloudbuild_trigger_facts` module was renamed to `gcp_cloudbuild_trigger_info`.
- The `gcp_compute_address_facts` module was renamed to `gcp_compute_address_info`.
- The `gcp_compute_backend_bucket_facts` module was renamed to `gcp_compute_backend_bucket_info`.
- The `gcp_compute_backend_service_facts` module was renamed to `gcp_compute_backend_service_info`.
- The `gcp_compute_disk_facts` module was renamed to `gcp_compute_disk_info`.
- The `gcp_compute_firewall_facts` module was renamed to `gcp_compute_firewall_info`.
- The `gcp_compute_forwarding_rule_facts` module was renamed to `gcp_compute_forwarding_rule_info`.
- The `gcp_compute_global_address_facts` module was renamed to `gcp_compute_global_address_info`.
- The `gcp_compute_global_forwarding_rule_facts` module was renamed to `gcp_compute_global_forwarding_rule_info`.
- The `gcp_compute_health_check_facts` module was renamed to `gcp_compute_health_check_info`.
- The `gcp_compute_http_health_check_facts` module was renamed to `gcp_compute_http_health_check_info`.
- The `gcp_compute_https_health_check_facts` module was renamed to `gcp_compute_https_health_check_info`.

- The `gcp_compute_image_facts` module was renamed to `gcp_compute_image_info`.
- The `gcp_compute_instance_facts` module was renamed to `gcp_compute_instance_info`.
- The `gcp_compute_instance_group_facts` module was renamed to `gcp_compute_instance_group_info`.
- The `gcp_compute_instance_group_manager_facts` module was renamed to `gcp_compute_instance_group_manager_info`.
- The `gcp_compute_instance_template_facts` module was renamed to `gcp_compute_instance_template_info`.
- The `gcp_compute_interconnect_attachment_facts` module was renamed to `gcp_compute_interconnect_attachment_info`.
- The `gcp_compute_network_facts` module was renamed to `gcp_compute_network_info`.
- The `gcp_compute_region_disk_facts` module was renamed to `gcp_compute_region_disk_info`.
- The `gcp_compute_route_facts` module was renamed to `gcp_compute_route_info`.
- The `gcp_compute_router_facts` module was renamed to `gcp_compute_router_info`.
- The `gcp_compute_ssl_certificate_facts` module was renamed to `gcp_compute_ssl_certificate_info`.
- The `gcp_compute_ssl_policy_facts` module was renamed to `gcp_compute_ssl_policy_info`.
- The `gcp_compute_subnetwork_facts` module was renamed to `gcp_compute_subnetwork_info`.
- The `gcp_compute_target_http_proxy_facts` module was renamed to `gcp_compute_target_http_proxy_info`.
- The `gcp_compute_target_https_proxy_facts` module was renamed to `gcp_compute_target_https_proxy_info`.
- The `gcp_compute_target_pool_facts` module was renamed to `gcp_compute_target_pool_info`.
- The `gcp_compute_target_ssl_proxy_facts` module was renamed to `gcp_compute_target_ssl_proxy_info`.
- The `gcp_compute_target_tcp_proxy_facts` module was renamed to `gcp_compute_target_tcp_proxy_info`.
- The `gcp_compute_target_vpn_gateway_facts` module was renamed to `gcp_compute_target_vpn_gateway_info`.
- The `gcp_compute_url_map_facts` module was renamed to `gcp_compute_url_map_info`.
- The `gcp_compute_vpn_tunnel_facts` module was renamed to `gcp_compute_vpn_tunnel_info`.
- The `gcp_container_cluster_facts` module was renamed to `gcp_container_cluster_info`.
- The `gcp_container_node_pool_facts` module was renamed to `gcp_container_node_pool_info`.

- The `gcp_dns_managed_zone_facts` module was renamed to `gcp_dns_managed_zone_info`.
- The `gcp_dns_resource_record_set_facts` module was renamed to `gcp_dns_resource_record_set_info`.
- The `gcp_iam_role_facts` module was renamed to `gcp_iam_role_info`.
- The `gcp_iam_service_account_facts` module was renamed to `gcp_iam_service_account_info`.
- The `gcp_pubsub_subscription_facts` module was renamed to `gcp_pubsub_subscription_info`.
- The `gcp_pubsub_topic_facts` module was renamed to `gcp_pubsub_topic_info`.
- The `gcp_redis_instance_facts` module was renamed to `gcp_redis_instance_info`.
- The `gcp_resourcemanager_project_facts` module was renamed to `gcp_resourcemanager_project_info`.
- The `gcp_sourcerepo_repository_facts` module was renamed to `gcp_sourcerepo_repository_info`.
- The `gcp_spanner_database_facts` module was renamed to `gcp_spanner_database_info`.
- The `gcp_spanner_instance_facts` module was renamed to `gcp_spanner_instance_info`.
- The `gcp_sql_database_facts` module was renamed to `gcp_sql_database_info`.
- The `gcp_sql_instance_facts` module was renamed to `gcp_sql_instance_info`.
- The `gcp_sql_user_facts` module was renamed to `gcp_sql_user_info`.
- The `gcp_tpu_node_facts` module was renamed to `gcp_tpu_node_info`.
- The `gcpubsub_facts` module was renamed to `gcpubsub_info`.
- The `github_webhook_facts` module was renamed to `github_webhook_info`.
- The `gluster_heal_facts` module was renamed to `gluster_heal_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `hcloud_datacenter_facts` module was renamed to `hcloud_datacenter_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `hcloud_floating_ip_facts` module was renamed to `hcloud_floating_ip_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `hcloud_image_facts` module was renamed to `hcloud_image_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `hcloud_location_facts` module was renamed to `hcloud_location_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `hcloud_server_facts` module was renamed to `hcloud_server_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.

- The `hcloud_server_type_facts` module was renamed to `hcloud_server_type_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `hcloud_ssh_key_facts` module was renamed to `hcloud_ssh_key_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `hcloud_volume_facts` module was renamed to `hcloud_volume_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `hpilo_facts` module was renamed to `hpilo_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `iam_mfa_device_facts` module was renamed to `iam_mfa_device_info`.
- The `iam_role_facts` module was renamed to `iam_role_info`.
- The `iam_server_certificate_facts` module was renamed to `iam_server_certificate_info`.
- The `idrac_redfish_facts` module was renamed to `idrac_redfish_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `intersight_facts` module was renamed to `intersight_info`.
- The `jenkins_job_facts` module was renamed to `jenkins_job_info`.
- The `k8s_facts` module was renamed to `k8s_info`.
- The `memset_memstore_facts` module was renamed to `memset_memstore_info`.
- The `memset_server_facts` module was renamed to `memset_server_info`.
- The `one_image_facts` module was renamed to `one_image_info`.
- The `onepassword_facts` module was renamed to `onepassword_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `oneview_datacenter_facts` module was renamed to `oneview_datacenter_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `oneview_enclosure_facts` module was renamed to `oneview_enclosure_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `oneview_ethernet_network_facts` module was renamed to `oneview_ethernet_network_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `oneview_fc_network_facts` module was renamed to `oneview_fc_network_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.

- The `oneview_fcoe_network_facts` module was renamed to `oneview_fcoe_network_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `oneview_logical_interconnect_group_facts` module was renamed to `oneview_logical_interconnect_group_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `oneview_network_set_facts` module was renamed to `oneview_network_set_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `oneview_san_manager_facts` module was renamed to `oneview_san_manager_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `os_flavor_facts` module was renamed to `os_flavor_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `os_image_facts` module was renamed to `os_image_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `os_keystone_domain_facts` module was renamed to `os_keystone_domain_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `os_networks_facts` module was renamed to `os_networks_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `os_port_facts` module was renamed to `os_port_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `os_project_facts` module was renamed to `os_project_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `os_server_facts` module was renamed to `os_server_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `os_subnets_facts` module was renamed to `os_subnets_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `os_user_facts` module was renamed to `os_user_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_affinity_label_facts` module was renamed to `ovirt_affinity_label_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_api_facts` module was renamed to `ovirt_api_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.

- The `ovirt_cluster_facts` module was renamed to `ovirt_cluster_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_datacenter_facts` module was renamed to `ovirt_datacenter_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_disk_facts` module was renamed to `ovirt_disk_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_event_facts` module was renamed to `ovirt_event_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_external_provider_facts` module was renamed to `ovirt_external_provider_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_group_facts` module was renamed to `ovirt_group_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_host_facts` module was renamed to `ovirt_host_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_host_storage_facts` module was renamed to `ovirt_host_storage_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_network_facts` module was renamed to `ovirt_network_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_nic_facts` module was renamed to `ovirt_nic_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_permission_facts` module was renamed to `ovirt_permission_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_quota_facts` module was renamed to `ovirt_quota_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_scheduling_policy_facts` module was renamed to `ovirt_scheduling_policy_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_snapshot_facts` module was renamed to `ovirt_snapshot_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_storage_domain_facts` module was renamed to `ovirt_storage_domain_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_storage_template_facts` module was renamed to `ovirt_storage_template_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values,

register a variable.

- The `ovirt_storage_vm_facts` module was renamed to `ovirt_storage_vm_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_tag_facts` module was renamed to `ovirt_tag_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_template_facts` module was renamed to `ovirt_template_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_user_facts` module was renamed to `ovirt_user_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_vm_facts` module was renamed to `ovirt_vm_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_vmpool_facts` module was renamed to `ovirt_vmpool_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `python_requirements_facts` module was renamed to `python_requirements_info`.
- The `rds_instance_facts` module was renamed to `rds_instance_info`.
- The `rds_snapshot_facts` module was renamed to `rds_snapshot_info`.
- The `redfish_facts` module was renamed to `redfish_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `redshift_facts` module was renamed to `redshift_info`.
- The `route53_facts` module was renamed to `route53_info`.
- The `smartos_image_facts` module was renamed to `smartos_image_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `vertica_facts` module was renamed to `vertica_info`. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `vmware_cluster_facts` module was renamed to `vmware_cluster_info`.
- The `vmware_datastore_facts` module was renamed to `vmware_datastore_info`.
- The `vmware_guest_facts` module was renamed to `vmware_guest_info`.
- The `vmware_guest_snapshot_facts` module was renamed to `vmware_guest_snapshot_info`.
- The `vmware_tag_facts` module was renamed to `vmware_tag_info`.
- The `vmware_vm_facts` module was renamed to `vmware_vm_info`.
- The `xenserver_guest_facts` module was renamed to `xenserver_guest_info`.
- The `zabbix_group_facts` module was renamed to `zabbix_group_info`.
- The `zabbix_host_facts` module was renamed to `zabbix_host_info`.

Noteworthy module changes

- `vmware_cluster` was refactored for easier maintenance/bugfixes. Use the three new, specialized modules to configure clusters. Configure DRS with `vmware_cluster_drs`, HA with `vmware_cluster_ha` and vSAN with `vmware_cluster_vsan`.
- `vmware_dvswitch` accepts `folder` parameter to place dvswitch in user defined folder. This option makes `datacenter` as an optional parameter.
- `vmware_datastore_cluster` accepts `folder` parameter to place datastore cluster in user defined folder. This option makes `datacenter` as an optional parameter.
- `mysql_db` returns new `db_list` parameter in addition to `db` parameter. This `db_list` parameter refers to list of database names. `db` parameter will be deprecated in version 2.13.
- `snow_record` and `snow_record_find` now takes environment variables for `instance`, `username` and `password` parameters. This change marks these parameters as optional.
- The deprecated `force` option in `win_firewall_rule` has been removed.
- `openssl_certificate`'s `ownca` provider creates authority key identifiers if not explicitly disabled with `ownca_create_authority_key_identifier: no`. This is only the case for the `cryptography` backend, which is selected by default if the `cryptography` library is available.
- `openssl_certificate`'s `ownca` and `selfsigned` providers create subject key identifiers if not explicitly disabled with `ownca_create_subject_key_identifier: never_create` resp. `selfsigned_create_subject_key_identifier: never_create`. If a subject key identifier is provided by the CSR, it is taken; if not, it is created from the public key. This is only the case for the `cryptography` backend, which is selected by default if the `cryptography` library is available.
- `openssh_keypair` now applies the same file permissions and ownership to both public and private keys (both get the same `mode`, `owner`, `group`, etc.). If you need to change permissions / ownership on one key, use the file to modify it after it is created.

Plugins

Removed Lookup Plugins

- `redis_kv` use `redis` instead.

Porting custom scripts

No notable changes

Networking

Network resource modules

Ansible 2.9 introduced the first batch of network resource modules. Sections of a network device's configuration can be thought of as a resource provided by that device. Network resource modules are intentionally scoped to configure a single resource and you can combine them as building blocks to configure complex network services. The older modules are deprecated in Ansible 2.9 and will be removed in Ansible 2.13. You should scan the list of deprecated modules above and replace them with the new network resource modules in your playbooks. See [Ansible Network Features in 2.9](#) for details.

Improved `gather_facts` support for network devices

In Ansible 2.9, the `gather_facts` keyword now supports gathering network device facts in standardized key/value pairs. You can feed these network facts into further tasks to manage the network device. You can also use the new `gather_network_resources` parameter with the network `*_facts` modules (such as `eos_facts`) to return just a subset of the device configuration. See [Gathering facts from network devices](#) for an example.

Top-level connection arguments removed in 2.9

Top-level connection arguments like `username`, `host`, and `password` are removed in version 2.9.

OLD In Ansible < 2.4

```
- name: example of using top-level options for connection properties
  ios_command:
    commands: show version
    host: "{{ inventory_hostname }}"
    username: cisco
    password: cisco
    authorize: yes
    auth_pass: cisco
```

Change your playbooks to the connection types `network_cli` and `netconf` using standard Ansible connection properties, and setting those properties in inventory by group. As you update your playbooks and inventory files, you can easily make the change to `become` for privilege escalation (on platforms that support it). For more information, see the [using become with network modules](#) guide and the [platform documentation](#).

1.2.3 Ansible 2.8 Porting Guide

This section discusses the behavioral changes between Ansible 2.7 and Ansible 2.8.

It is intended to assist in updating your playbooks, plugins and other parts of your Ansible infrastructure so they will work with this version of Ansible.

We suggest you read this page along with [Ansible Changelog for 2.8](#) to understand what updates you may need to make.

This document is part of a collection on porting. The complete list of porting guides can be found at [porting guides](#).

- *Playbook*
 - *Distribution Facts*
 - *Imports as handlers*
 - *Jinja Undefined values*
 - *Module option conversion to string*
 - *Command line facts*
 - *Bare variables in conditionals*
 - * *Updating your playbooks*
 - * *Double-interpolation*
 - * *Nested variables*
 - *Gathering Facts*
- *Python Interpreter Discovery*
 - *Retry File Creation default*
- *Command Line*
 - *Become Prompting*
- *Deprecated*
- *Modules*
 - *Modules removed*
 - *Deprecation notices*
 - *Noteworthy module changes*
- *Plugins*
- *Porting custom scripts*
 - *Display class*
- *Networking*

Playbook

Distribution Facts

The information returned for the `ansible_distribution_*` group of facts may have changed slightly. Ansible 2.8 uses a new backend library for information about distributions: [nir0s/distro](#). This library runs on Python-3.8 and fixes many bugs, including correcting release and version names.

The two facts used in playbooks most often, `ansible_distribution` and `ansible_distribution_major_version`, should not change. If you discover a change in these facts, please file a bug so we can address the difference. However, other facts like `ansible_distribution_release` and `ansible_distribution_version` may change as erroneous information gets corrected.

Imports as handlers

Beginning in version 2.8, a task cannot notify `import_tasks` or a static `include` that is specified in `handlers`.

The goal of a static import is to act as a pre-processor, where the import is replaced by the tasks defined within the imported file. When using an import, a task can notify any of the named tasks within the imported file, but not the name of the import itself.

To achieve the results of notifying a single name but running multiple handlers, utilize `include_tasks`, or listen *Handlers: running operations on change*.

Jinja Undefined values

Beginning in version 2.8, attempting to access an attribute of an Undefined value in Jinja will return another Undefined value, rather than throwing an error immediately. This means that you can now simply use a default with a value in a nested data structure when you don't know if the intermediate values are defined.

In Ansible 2.8:

```
{{ foo.bar.baz | default('DEFAULT') }}
```

In Ansible 2.7 and older:

```
{{ ((foo | default({})).bar | default({})).baz | default('DEFAULT') }}
```

or

```
{{ foo.bar.baz if (foo is defined and foo.bar is defined and foo.bar.baz is defined)
↪ else 'DEFAULT' }}
```

Module option conversion to string

Beginning in version 2.8, Ansible will warn if a module expects a string, but a non-string value is passed and automatically converted to a string. This highlights potential problems where, for example, a `yes` or `true` (parsed as truish boolean value) would be converted to the string `'True'`, or where a version number `1.10` (parsed as float value) would be converted to `'1.1'`. Such conversions can result in unexpected behavior depending on context.

This behavior can be changed to be an error or to be ignored by setting the `ANSIBLE_STRING_CONVERSION_ACTION` environment variable, or by setting the `string_conversion_action` configuration in the `defaults` section of `ansible.cfg`.

Command line facts

`cmdline` facts returned in system will be deprecated in favor of `proc_cmdline`. This change handles special case where Kernel command line parameter contains multiple values with the same key.

Bare variables in conditionals

In Ansible 2.7 and earlier, top-level variables sometimes treated boolean strings as if they were boolean values. This led to inconsistent behavior in conditional tests built on top-level variables defined as strings. Ansible 2.8 began changing this behavior. For example, if you set two conditions like this:

```
tasks:
- include_tasks: teardown.yml
  when: teardown

- include_tasks: provision.yml
  when: not teardown
```

based on a variable you define **as a string** (with quotation marks around it):

- In Ansible 2.7 and earlier, the two conditions above evaluated as `True` and `False` respectively if `teardown: 'true'`
- In Ansible 2.7 and earlier, both conditions evaluated as `False` if `teardown: 'false'`
- In Ansible 2.8 and later, you have the option of disabling conditional bare variables, so `when: teardown` always evaluates as `True` and `when: not teardown` always evaluates as `False` when `teardown` is a non-empty string (including `'true'` or `'false'`)

Ultimately, `when: 'string'` will always evaluate as `True` and `when: not 'string'` will always evaluate as `False`, as long as `'string'` is not empty, even if the value of `'string'` itself looks like a boolean. For users with playbooks that depend on the old behavior, we added a config setting that preserves it. You can

use the `ANSIBLE_CONDITIONAL_BARE_VARS` environment variable or `conditional_bare_variables` in the `defaults` section of `ansible.cfg` to select the behavior you want on your control node. The default setting is `true`, which preserves the old behavior. Set the config value or environment variable to `false` to start using the new option.

注解: In 2.10 the default setting for `conditional_bare_variables` will change to `false`. In 2.12 the old behavior will be deprecated.

Updating your playbooks

To prepare your playbooks for the new behavior, you must update your conditional statements so they accept only boolean values. For variables, you can use the `bool` filter to evaluate the string `'false'` as `False`:

```
vars:
    teardown: 'false'

tasks:
  - include_tasks: teardown.yml
    when: teardown | bool

  - include_tasks: provision.yml
    when: not teardown | bool
```

Alternatively, you can re-define your variables as boolean values (without quotation marks) instead of strings:

```
vars:
    teardown: false

tasks:
  - include_tasks: teardown.yml
    when: teardown

  - include_tasks: provision.yml
    when: not teardown
```

For dictionaries and lists, use the `length` filter to evaluate the presence of a dictionary or list as `True`:

```
- debug:
    when: my_list | length > 0
```

(下页继续)

(续上页)

```
- debug:
  when: my_dictionary | length > 0
```

Do not use the `bool` filter with lists or dictionaries. If you use `bool` with a list or dict, Ansible will always evaluate it as `False`.

Double-interpolation

The `conditional_bare_variables` setting also affects variables set based on other variables. The old behavior unexpectedly double-interpolated those variables. For example:

```
vars:
  double_interpolated: 'bare_variable'
  bare_variable: false

tasks:
  - debug:
    when: double_interpolated
```

- In Ansible 2.7 and earlier, `when: double_interpolated` evaluated to the value of `bare_variable`, in this case, `False`. If the variable `bare_variable` is undefined, the conditional fails.
- In Ansible 2.8 and later, with bare variables disabled, Ansible evaluates `double_interpolated` as the string `'bare_variable'`, which is `True`.

To double-interpolate variable values, use curly braces:

```
vars:
  double_interpolated: "{{ other_variable }}"
  other_variable: false
```

Nested variables

The `conditional_bare_variables` setting does not affect nested variables. Any string value assigned to a subkey is already respected and not treated as a boolean. If `complex_variable['subkey']` is a non-empty string, then `when: complex_variable['subkey']` is always `True` and `when: not complex_variable['subkey']` is always `False`. If you want a string subkey like `complex_variable['subkey']` to be evaluated as a boolean, you must use the `bool` filter.

Gathering Facts

In Ansible 2.8 the implicit “Gathering Facts” task in a play was changed to obey play tags. Previous to 2.8, the “Gathering Facts” task would ignore play tags and tags supplied from the command line and always run in a task.

The behavior change affects the following example play.

```
- name: Configure Webservers
  hosts: webserver
  tags:
    - webserver
  tasks:
    - name: Install nginx
      package:
        name: nginx
      tags:
        - nginx
```

In Ansible 2.8, if you supply `--tags nginx`, the implicit “Gathering Facts” task will be skipped, as the task now inherits the tag of `webserver` instead of `always`.

If no play level tags are set, the “Gathering Facts” task will be given a tag of `always` and will effectively match prior behavior.

You can achieve similar results to the pre-2.8 behavior, by using an explicit `gather_facts` task in your tasks list.

```
- name: Configure Webservers
  hosts: webserver
  gather_facts: false
  tags:
    - webserver
  tasks:
    - name: Gathering Facts
      gather_facts:
        tags:
          - always

    - name: Install nginx
      package:
        name: nginx
      tags:
        - nginx
```


Python Interpreter Discovery

In Ansible 2.7 and earlier, Ansible defaulted to `/usr/bin/python` as the setting for `ansible_python_interpreter`. If you ran Ansible against a system that installed Python with a different name or a different path, your playbooks would fail with `/usr/bin/python: bad interpreter: No such file or directory` unless you either set `ansible_python_interpreter` to the correct value for that system or added a Python interpreter and any necessary dependencies at `usr/bin/python`.

Starting in Ansible 2.8, Ansible searches for the correct path and executable name for Python on each target system, first in a lookup table of default Python interpreters for common distros, then in an ordered fallback list of possible Python interpreter names/paths.

It's risky to rely on a Python interpreter set from the fallback list, because the interpreter may change on future runs. If an interpreter from higher in the fallback list gets installed (for example, as a side-effect of installing other packages), your original interpreter and its dependencies will no longer be used. For this reason, Ansible warns you when it uses a Python interpreter discovered from the fallback list. If you see this warning, the best solution is to explicitly set `ansible_python_interpreter` to the path of the correct interpreter for those target systems.

You can still set `ansible_python_interpreter` to a specific path at any variable level (as a host variable, in vars files, in playbooks, etc.). If you prefer to use the Python interpreter discovery behavior, use one of the four new values for `ansible_python_interpreter` introduced in Ansible 2.8:

New value	Behavior
auto (future default)	If a Python interpreter is discovered, Ansible uses the discovered Python, even if <code>/usr/bin/python</code> is also present. Warns when using the fallback list.
auto_legacy (Ansible 2.8 default)	If a Python interpreter is discovered, and <code>/usr/bin/python</code> is absent, Ansible uses the discovered Python. Warns when using the fallback list. If a Python interpreter is discovered, and <code>/usr/bin/python</code> is present, Ansible uses <code>/usr/bin/python</code> and prints a deprecation warning about future default behavior. Warns when using the fallback list.
auto_legacy_bellows	Behaves like auto_legacy but suppresses the deprecation and fallback-list warnings.
auto_silent	Behaves like auto but suppresses the fallback-list warning.

In Ansible 2.12, Ansible will switch the default from **auto_legacy** to **auto**. The difference in behaviour is that **auto_legacy** uses `/usr/bin/python` if present and falls back to the discovered Python when it is not present. **auto** will always use the discovered Python, regardless of whether `/usr/bin/python` exists. The **auto_legacy** setting provides compatibility with previous versions of Ansible that always defaulted to `/usr/bin/python`.

If you installed Python and dependencies (**boto**, etc.) to `/usr/bin/python` as a workaround on distros with a different default Python interpreter (for example, Ubuntu 16.04+, RHEL8, Fedora 23+), you have two

options:

1. Move existing dependencies over to the default Python for each platform/distribution/version.
2. Use `auto_legacy`. This setting lets Ansible find and use the workaround Python on hosts that have it, while also finding the correct default Python on newer hosts. But remember, the default will change in 4 releases.

Retry File Creation default

In Ansible 2.8, `retry_files_enabled` now defaults to `False` instead of `True`. The behavior can be modified to previous version by editing the default `ansible.cfg` file and setting the value to `True`.

Command Line

Become Prompting

Beginning in version 2.8, by default Ansible will use the word `BECOME` to prompt you for a password for elevated privileges (`sudo` privileges on Unix systems or `enable` mode on network devices):

By default in Ansible 2.8:

```
ansible-playbook --become --ask-become-pass site.yml
BECOME password:
```

If you want the prompt to display the specific `become_method` you're using, instead of the agnostic value `BECOME`, set `AGNOSTIC_BECOME_PROMPT` to `False` in your Ansible configuration.

By default in Ansible 2.7, or with `AGNOSTIC_BECOME_PROMPT=False` in Ansible 2.8:

```
ansible-playbook --become --ask-become-pass site.yml
SUDO password:
```

Deprecated

- Setting the async directory using `ANSIBLE_ASYNC_DIR` as an task/play environment key is deprecated and will be removed in Ansible 2.12. You can achieve the same result by setting `ansible_async_dir` as a variable like:

```
- name: run task with custom async directory
  command: sleep 5
  async: 10
  vars:
    ansible_async_dir: /tmp/.ansible_async
```

- Plugin writers who need a `FactCache` object should be aware of two deprecations:
 1. The `FactCache` class has moved from `ansible.plugins.cache.FactCache` to `ansible.vars.fact_cache.FactCache`. This is because the `FactCache` is not part of the cache plugin API and cache plugin authors should not be subclassing it. `FactCache` is still available from its old location but will issue a deprecation warning when used from there. The old location will be removed in Ansible 2.12.
 2. The `FactCache.update()` method has been converted to follow the dict API. It now takes a dictionary as its sole argument and updates itself with the dictionary's items. The previous API where `update()` took a key and a value will now issue a deprecation warning and will be removed in 2.12. If you need the old behavior switch to `FactCache.first_order_merge()` instead.
- Supporting file-backed caching via `self.cache` is deprecated and will be removed in Ansible 2.12. If you maintain an inventory plugin, update it to use `self._cache` as a dictionary. For implementation details, see the *developer guide on inventory plugins*.
- Importing cache plugins directly is deprecated and will be removed in Ansible 2.12. Use the `plugin_loader` so direct options, environment variables, and other means of configuration can be reconciled using the config system rather than constants.

```
from ansible.plugins.loader import cache_loader
cache = cache_loader.get('redis', **kwargs)
```

Modules

Major changes in popular modules are detailed here

The exec wrapper that runs PowerShell modules has been changed to set `$ErrorActionPreference = "Stop"` globally. This may mean that custom modules can fail if they implicitly relied on this behavior. To get the old behavior back, add `$ErrorActionPreference = "Continue"` to the top of the module. This change was made to restore the old behavior of the EAP that was accidentally removed in a previous release and ensure that modules are more resilient to errors that may occur in execution.

Modules removed

The following modules no longer exist:

- `ec2_remote_facts`
- `azure`
- `cs_nic`
- `netcaler`
- `win_msi`

Deprecation notices

The following modules will be removed in Ansible 2.12. Please update your playbooks accordingly.

- `foreman` use [foreman-ansible-modules](#) instead.
- `katello` use [foreman-ansible-modules](#) instead.
- `github_hooks` use `github_webhook` and `github_webhook_facts` instead.
- `digital_ocean` use `digital_ocean_droplet` instead.
- `gce` use `gcp_compute_instance` instead.
- `gcspanner` use `gcp_spanner_instance` and `gcp_spanner_database` instead.
- `gcdn_record` use `gcp_dns_resource_record_set` instead.
- `gcdn_zone` use `gcp_dns_managed_zone` instead.
- `gcp_forwarding_rule` use `gcp_compute_global_forwarding_rule` or `gcp_compute_forwarding_rule` instead.
- `gcp_healthcheck` use `gcp_compute_health_check`, `gcp_compute_http_health_check`, or `gcp_compute_https_health_check` instead.
- `gcp_backend_service` use `gcp_compute_backend_service` instead.
- `gcp_target_proxy` use `gcp_compute_target_http_proxy` instead.
- `gcp_url_map` use `gcp_compute_url_map` instead.
- `panos` use the [Palo Alto Networks Ansible Galaxy role](#) instead.

Noteworthy module changes

- The `foreman` and `katello` modules have been deprecated in favor of a set of modules that are broken out per entity with better idempotency in mind.
- The `foreman` and `katello` modules replacement is officially part of the Foreman Community and supported there.
- The `tower_credential` module originally required the `ssh_key_data` to be the path to a `ssh_key_file`. In order to work like Tower/AWX, `ssh_key_data` now contains the content of the file. The previous behavior can be achieved with `lookup('file', '/path/to/file')`.
- The `win_scheduled_task` module deprecated support for specifying a trigger repetition as a list and this format will be removed in Ansible 2.12. Instead specify the repetition as a dictionary value.
- The `win_feature` module has removed the deprecated `restart_needed` return value, use the standardized `reboot_required` value instead.

- The `win_package` module has removed the deprecated `restart_required` and `exit_code` return value, use the standardized `reboot_required` and `rc` value instead.
- The `win_get_url` module has removed the deprecated `win_get_url` return dictionary, contained values are returned directly.
- The `win_get_url` module has removed the deprecated `skip_certificate_validation` option, use the standardized `validate_certs` option instead.
- The `vmware_local_role_facts` module now returns a list of dicts instead of a dict of dicts for role information.
- If `docker_network` or `docker_volume` were called with `diff: yes`, `check_mode: yes` or `debug: yes`, a return value called `diff` was returned of type `list`. To enable proper diff output, this was changed to type `dict`; the original `list` is returned as `diff.differences`.
- The `na_ontap_cluster_peer` module has replaced `source_intercluster_lif` and `dest_intercluster_lif` string options with `source_intercluster_lifs` and `dest_intercluster_lifs` list options
- The `modprobe` module now detects kernel builtins. Previously, attempting to remove (with `state: absent`) a builtin kernel module succeeded without any error message because `modprobe` did not detect the module as `present`. Now, `modprobe` will fail if a kernel module is builtin and `state: absent` (with an error message from the `modprobe` binary like `modprobe: ERROR: Module nfs is builtin.`), and it will succeed without reporting changed if `state: present`. Any playbooks that are using `changed_when: no` to mask this quirk can safely remove that workaround. To get the previous behavior when applying `state: absent` to a builtin kernel module, use `failed_when: false` or `ignore_errors: true` in your playbook.
- The `digital_ocean` module has been deprecated in favor of modules that do not require external dependencies. This allows for more flexibility and better module support.
- The `docker_container` module has deprecated the returned fact `docker_container`. The same value is available as the returned variable `container`. The returned fact will be removed in Ansible 2.12.
- The `docker_network` module has deprecated the returned fact `docker_container`. The same value is available as the returned variable `network`. The returned fact will be removed in Ansible 2.12.
- The `docker_volume` module has deprecated the returned fact `docker_container`. The same value is available as the returned variable `volume`. The returned fact will be removed in Ansible 2.12.
- The `docker_service` module was renamed to `docker_compose`.
- The renamed `docker_compose` module used to return one fact per service, named same as the service. A dictionary of these facts is returned as the regular return value `services`. The returned facts will be removed in Ansible 2.12.
- The `docker_swarm_service` module no longer sets a defaults for the following options:
 - `user`. Before, the default was `root`.

- `update_delay`. Before, the default was 10.
 - `update_parallelism`. Before, the default was 1.
- `vmware_vm_facts` used to return dict of dict with virtual machine's facts. Ansible 2.8 and onwards will return list of dict with virtual machine's facts. Please see module `vmware_vm_facts` documentation for example.
- `vmware_guest_snapshot` module used to return `results`. Since Ansible 2.8 and onwards `results` is a reserved keyword, it is replaced by `snapshot_results`. Please see module `vmware_guest_snapshots` documentation for example.
- The `panos` modules have been deprecated in favor of using the Palo Alto Networks [Ansible Galaxy role](#). Contributions to the role can be made [here](#).
- The `ipa_user` module originally always sent `password` to FreeIPA regardless of whether the password changed. Now the module only sends `password` if `update_password` is set to `always`, which is the default.
- The `win_psexec` has deprecated the undocumented `extra_opts` module option. This will be removed in Ansible 2.10.
- The `win_nssm` module has deprecated the following options in favor of using the `win_service` module to configure the service after installing it with `win_nssm`: * `dependencies`, use `dependencies` of `win_service` instead * `start_mode`, use `start_mode` of `win_service` instead * `user`, use `username` of `win_service` instead * `password`, use `password` of `win_service` instead These options will be removed in Ansible 2.12.
- The `win_nssm` module has also deprecated the `start`, `stop`, and `restart` values of the `status` option. You should use the `win_service` module to control the running state of the service. This will be removed in Ansible 2.12.
- The `status` module option for `win_nssm` has changed its default value to `present`. Before, the default was `start`. Consequently, the service is no longer started by default after creation with `win_nssm`, and you should use the `win_service` module to start it if needed.
- The `app_parameters` module option for `win_nssm` has been deprecated; use `argument` instead. This will be removed in Ansible 2.12.
- The `app_parameters_free_form` module option for `win_nssm` has been aliased to the new `arguments` option.
- The `win_dsc` module will now validate the input options for a DSC resource. In previous versions invalid options would be ignored but are now not.
- The `openssl_pkcs12` module will now regenerate the pkcs12 file if there are differences between the file on disk and the parameters passed to the module.

Plugins

- Ansible no longer defaults to the **paramiko** connection plugin when using macOS as the control node. Ansible will now use the **ssh** connection plugin by default on a macOS control node. Since **ssh** supports connection persistence between tasks and playbook runs, it performs better than **paramiko**. If you are using password authentication, you will need to install **sshpass** when using the **ssh** connection plugin. Or you can explicitly set the connection type to **paramiko** to maintain the pre-2.8 behavior on macOS.
- Connection plugins have been standardized to allow use of **ansible_<conn-type>_user** and **ansible_<conn-type>_password** variables. Variables such as **ansible_<conn-type>_pass** and **ansible_<conn-type>_username** are treated with lower priority than the standardized names and may be deprecated in the future. In general, the **ansible_user** and **ansible_password** vars should be used unless there is a reason to use the connection-specific variables.
- The **powershell** shell plugin now uses **async_dir** to define the async path for the results file and the default has changed to **%USERPROFILE%\\.ansible_async**. To control this path now, either set the **ansible_async_dir** variable or the **async_dir** value in the **powershell** section of the config ini.
- Order of enabled inventory plugins (**INVENTORY_ENABLED**) has been updated, **auto** is now before **yaml** and **ini**.
- The private **_options** attribute has been removed from the **CallbackBase** class of callback plugins. If you have a third-party callback plugin which needs to access the command line arguments, use code like the following instead of trying to use **self._options**:

```
from ansible import context
[...]
tags = context.CLIARGS['tags']
```

context.CLIARGS is a read-only dictionary so normal dictionary retrieval methods like **CLIARGS.get('tags')** and **CLIARGS['tags']** work as expected but you won't be able to modify the cli arguments at all.

- Play recap now counts **ignored** and **rescued** tasks as well as **ok**, **changed**, **unreachable**, **failed** and **skipped** tasks, thanks to two additional stat counters in the **default** callback plugin. Tasks that fail and have **ignore_errors: yes** set are listed as **ignored**. Tasks that fail and then execute a rescue section are listed as **rescued**. Note that **rescued** tasks are no longer counted as **failed** as in Ansible 2.7 (and earlier).
- **osx_say** callback plugin was renamed into **say**.
- Inventory plugins now support caching via cache plugins. To start using a cache plugin with your inventory see the section on caching in the *inventory guide*. To port a custom cache plugin to be compatible with inventory see *developer guide on cache plugins*.

Porting custom scripts

Display class

As of Ansible 2.8, the `Display` class is now a “singleton” . Instead of using `__main__.display` each file should import and instantiate `ansible.utils.display.Display` on its own.

OLD In Ansible 2.7 (and earlier) the following was used to access the `display` object:

```
try:
    from __main__ import display
except ImportError:
    from ansible.utils.display import Display
    display = Display()
```

NEW In Ansible 2.8 the following should be used:

```
from ansible.utils.display import Display
display = Display()
```

Networking

- The `eos_config`, `ios_config`, and `nxos_config` modules have removed the deprecated `save` and `force` parameters, use the `save_when` parameter to replicate their functionality.
- The `nxos_vrf_af` module has removed the `safi` parameter. This parameter was deprecated in Ansible 2.4 and has had no impact on the module since then.

1.2.4 Ansible 2.7 Porting Guide

This section discusses the behavioral changes between Ansible 2.6 and Ansible 2.7.

It is intended to assist in updating your playbooks, plugins and other parts of your Ansible infrastructure so they will work with this version of Ansible.

We suggest you read this page along with [Ansible Changelog for 2.7](#) to understand what updates you may need to make.

This document is part of a collection on porting. The complete list of porting guides can be found at [porting guides](#).

Topics

- [Ansible 2.7 Porting Guide](#)

- *Command Line*
- *Python Compatibility*
- *Playbook*
 - * *Role Precedence Fix during Role Loading*
 - * *include_role and import_role variable exposure*
 - * *include_tasks/import_tasks inline variables*
 - * *vars_prompt with unknown algorithms*
- *Deprecated*
 - * *Expedited Deprecation: Use of __file__ in AnsibleModule*
 - * *Using a loop on a package module via squash_actions*
- *Modules*
 - * *Modules removed*
 - * *Deprecation notices*
 - * *Noteworthy module changes*
- *Plugins*
- *Porting custom scripts*
- *Networking*

Command Line

If you specify `--tags` or `--skip-tags` multiple times on the command line, Ansible will merge the specified tags together. In previous versions of Ansible, you could set `merge_multiple_cli_tags` to `False` if you wanted to keep only the last-specified `--tags`. This config option existed for backwards compatibility. The overwriting behavior was deprecated in 2.3 and the default behavior was changed in 2.4. Ansible-2.7 removes the config option; multiple `--tags` are now always merged.

If you have a shell script that depends on setting `merge_multiple_cli_tags` to `False`, please upgrade your script so it only adds the `--tags` you actually want before upgrading to Ansible-2.7.

Python Compatibility

Ansible has dropped compatibility with Python-2.6 on the controller (The host where `/usr/bin/ansible` or `/usr/bin/ansible-playbook` is run). Modules shipped with Ansible can still be used to manage hosts which only have Python-2.6. You just need to have a host with Python-2.7 or Python-3.5 or greater to manage those hosts from.

One thing that this does affect is the ability to use `/usr/bin/ansible-pull` to manage a host which has Python-2.6. `ansible-pull` runs on the host being managed but it is a controller script, not a module so it will need an updated Python. Actively developed Linux distros which ship with Python-2.6 have some means to install newer Python versions (For instance, you can install Python-2.7 via an SCL on RHEL-6) but you may need to also install Python bindings for many common modules to work (For RHEL-6, for instance, selinux bindings and yum would have to be installed for the updated Python install).

The decision to drop Python-2.6 support on the controller was made because many dependent libraries are becoming unavailable there. In particular, python-cryptography is no longer available for Python-2.6 and the last release of pycrypto (the alternative to python-cryptography) has known security bugs which will never be fixed.

Playbook

Role Precedence Fix during Role Loading

Ansible 2.7 makes a small change to variable precedence when loading roles, resolving a bug, ensuring that role loading matches *variable precedence expectations*.

Before Ansible 2.7, when loading a role, the variables defined in the role's `vars/main.yml` and `defaults/main.yml` were not available when parsing the role's `tasks/main.yml` file. This prevented the role from utilizing these variables when being parsed. The problem manifested when `import_tasks` or `import_role` was used with a variable defined in the role's `vars` or `defaults`.

In Ansible 2.7, role `vars` and `defaults` are now parsed before `tasks/main.yml`. This can cause a change in behavior if the same variable is defined at the play level and the role level with different values, and leveraged in `import_tasks` or `import_role` to define the role or file to import.

include_role and import_role variable exposure

In Ansible 2.7 a new module argument named `public` was added to the `include_role` module that dictates whether or not the role's `defaults` and `vars` will be exposed outside of the role, allowing those variables to be used by later tasks. This value defaults to `public: False`, matching current behavior.

`import_role` does not support the `public` argument, and will unconditionally expose the role's `defaults` and `vars` to the rest of the playbook. This functionality brings `import_role` into closer alignment with roles listed within the `roles` header in a play.

There is an important difference in the way that `include_role` (dynamic) will expose the role's variables, as opposed to `import_role` (static). `import_role` is a pre-processor, and the `defaults` and `vars` are evaluated at playbook parsing, making the variables available to tasks and roles listed at any point in the play. `include_role` is a conditional task, and the `defaults` and `vars` are evaluated at execution time, making the variables available to tasks and roles listed *after* the `include_role` task.

include_tasks/import_tasks inline variables

As of Ansible 2.7, *include_tasks* and *import_tasks* can no longer accept inline variables. Instead of using inline variables, tasks should supply variables under the **vars** keyword.

OLD In Ansible 2.6 (and earlier) the following was valid syntax for specifying variables:

```
- include_tasks: include_me.yml variable=value
```

NEW In Ansible 2.7 the task should be changed to use the **vars** keyword:

```
- include_tasks: include_me.yml
  vars:
    variable: value
```

vars_prompt with unknown algorithms

`vars_prompt` now throws an error if the hash algorithm specified in `encrypt` is not supported by the controller. This increases the safety of `vars_prompt` as it previously returned `None` if the algorithm was unknown. Some modules, notably the `user` module, treated a password of `None` as a request not to set a password. If your playbook starts erroring because of this, change the hashing algorithm being used with this filter.

Deprecated

Expedited Deprecation: Use of `__file__` in `AnsibleModule`

注解: The use of the `__file__` variable is deprecated in Ansible 2.7 and **will be eliminated in Ansible 2.8**. This is much quicker than our usual 4-release deprecation cycle.

We are deprecating the use of the `__file__` variable to refer to the file containing the currently-running code. This common Python technique for finding a filesystem path does not always work (even in vanilla Python). Sometimes a Python module can be imported from a virtual location (like inside of a zip file). When this happens, the `__file__` variable will reference a virtual location pointing to inside of the zip file. This can cause problems if, for instance, the code was trying to use `__file__` to find the directory containing the python module to write some temporary information.

Before the introduction of `AnsiBallZ` in Ansible 2.1, using `__file__` worked in `AnsibleModule` sometimes, but any module that used it would fail when pipelining was turned on (because the module would be piped into the python interpreter's standard input, so `__file__` wouldn't contain a file path). `AnsiBallZ` unintentionally made using `__file__` work, by always creating a temporary file for `AnsibleModule` to reside in.

Ansible 2.8 will no longer create a temporary file for `AnsibleModule`; instead it will read the file out of a zip file. This change should speed up module execution, but it does mean that starting with Ansible 2.8, referencing `__file__` will always fail in `AnsibleModule`.

If you are the author of a third-party module which uses `__file__` with `AnsibleModule`, please update your module(s) now, while the use of `__file__` is deprecated but still available. The most common use of `__file__` is to find a directory to write a temporary file. In Ansible 2.5 and above, you can use the `tmpdir` attribute on an `AnsibleModule` instance instead, as shown in this code from the `apt` module:

```
-     tmpdir = os.path.dirname(__file__)
-     package = os.path.join(tmpdir, to_native(deb.rsplitt('/', 1)[1]))
+     package = os.path.join(module.tmpdir, to_native(deb.rsplitt('/', 1)[1]))
```

Using a loop on a package module via `squash_actions`

The use of `squash_actions` to invoke a package module, such as “yum”, to only invoke the module once is deprecated, and will be removed in Ansible 2.11.

Instead of relying on implicit squashing, tasks should instead supply the list directly to the `name`, `pkg` or `package` parameter of the module. This functionality has been supported in most modules since Ansible 2.3.

OLD In Ansible 2.6 (and earlier) the following task would invoke the “yum” module only 1 time to install multiple packages

```
- name: Install packages
  yum:
    name: "{{ item }}"
    state: present
  with_items: "{{ packages }}"
```

NEW In Ansible 2.7 it should be changed to look like this:

```
- name: Install packages
  yum:
    name: "{{ packages }}"
    state: present
```

Modules

Major changes in popular modules are detailed here

- The `DEFAULT_SYSLOG_FACILITY` configuration option tells Ansible modules to use a specific `syslog facility` when logging information on all managed machines. Due to a bug with older Ansible

versions, this setting did not affect machines using journald with the systemd Python bindings installed. On those machines, Ansible log messages were sent to `/var/log/messages`, even if you set `DEFAULT_SYSLOG_FACILITY`. Ansible 2.7 fixes this bug, routing all Ansible log messages according to the value set for `DEFAULT_SYSLOG_FACILITY`. If you have `DEFAULT_SYSLOG_FACILITY` configured, the location of remote logs on systems which use journald may change.

Modules removed

The following modules no longer exist:

Deprecation notices

The following modules will be removed in Ansible 2.11. Please update your playbooks accordingly.

- `na_cdot_aggregate` use `na_ontap_aggregate` instead.
- `na_cdot_license` use `na_ontap_license` instead.
- `na_cdot_lun` use `na_ontap_lun` instead.
- `na_cdot_qtree` use `na_ontap_qtree` instead.
- `na_cdot_svm` use `na_ontap_svm` instead.
- `na_cdot_user` use `na_ontap_user` instead.
- `na_cdot_user_role` use `na_ontap_user_role` instead.
- `na_cdot_volume` use `na_ontap_volume` instead.
- `sf_account_manager` use `na_elementsw_account` instead.
- `sf_check_connections` use `na_elementsw_check_connections` instead.
- `sf_snapshot_schedule_manager` use `na_elementsw_snapshot_schedule` instead.
- `sf_volume_access_group_manager` use `na_elementsw_access_group` instead.
- `sf_volume_manager` use `na_elementsw_volume` instead.

Noteworthy module changes

- Check mode is now supported in the `command` and `shell` modules. However, only when `creates` or `removes` is specified. If either of these are specified, the module will check for existence of the file and report the correct changed status, if they are not included the module will skip like it had done previously.
- The `win_chocolatey` module originally required the `proxy_username` and `proxy_password` to escape any double quotes in the value. This is no longer required and the escaping may cause further issues.

- The `win_uri` module has removed the deprecated option `use_basic_parsing`, since Ansible 2.5 this option did nothing
- The `win_scheduled_task` module has removed the following deprecated options:
 - `executable`, use `path` in an actions entry instead
 - `argument`, use `arguments` in an actions entry instead
 - `store_password`, set `logon_type: password` instead
 - `days_of_week`, use `monthlydow` in a triggers entry instead
 - `frequency`, use `type`, in a triggers entry instead
 - `time`, use `start_boundary` in a triggers entry instead
- The `interface_name` module option for `na_ontap_net_vlan` has been removed and should be removed from your playbooks
- The `win_disk_image` module has deprecated the return value `mount_path`, use `mount_paths[0]` instead. This will be removed in Ansible 2.11.
- `include_role` and `include_tasks` can now be used directly from `ansible` (ad hoc) and `ansible-console`:

```
#> ansible -m include_role -a 'name=myrole' all
```

- The `pip` module has added a dependency on `setuptools` to support version requirements, this requirement is for the Python interpreter that executes the module and not the Python interpreter that the module is managing.
- Prior to Ansible 2.7.10, the `replace` module did the opposite of what was intended when using the `before` and `after` options together. This now works properly but may require changes to tasks.

Plugins

- The `hash_password` filter now throws an error if the hash algorithm specified is not supported by the controller. This increases the safety of the filter as it previously returned `None` if the algorithm was unknown. Some modules, notably the `user` module, treated a password of `None` as a request not to set a password. If your playbook starts erroring because of this, change the hashing algorithm being used with this filter.

Porting custom scripts

No notable changes.

Networking

No notable changes.

1.2.5 Ansible 2.6 Porting Guide

This section discusses the behavioral changes between Ansible 2.5 and Ansible 2.6.

It is intended to assist in updating your playbooks, plugins and other parts of your Ansible infrastructure so they will work with this version of Ansible.

We suggest you read this page along with [Ansible Changelog for 2.6](#) to understand what updates you may need to make.

This document is part of a collection on porting. The complete list of porting guides can be found at [porting guides](#).

Topics

- *Ansible 2.6 Porting Guide*
 - *Playbook*
 - *Deprecated*
 - *Modules*
 - * *Modules removed*
 - * *Deprecation notices*
 - * *Noteworthy module changes*
 - *Plugins*
 - * *Deprecation notices*
 - * *Noteworthy plugin changes*
 - *Porting custom scripts*
 - *Networking*
 - *Dynamic inventory scripts*

Playbook

- The deprecated task option `always_run` has been removed, please use `check_mode: no` instead.

Deprecated

- In the `nxos_igmp_interface` module, `oif_prefix` and `oif_source` properties are deprecated. Use `ois_ps` parameter with a dictionary of prefix and source to values instead.

Modules

Major changes in popular modules are detailed here:

Modules removed

The following modules no longer exist:

Deprecation notices

The following modules will be removed in Ansible 2.10. Please update your playbooks accordingly.

- `k8s_raw` use `k8s` instead.
- `openshift_raw` use `k8s` instead.
- `openshift_scale` use `k8s_scale` instead.

Noteworthy module changes

- The `upgrade` module option for `win_chocolatey` has been removed; use `state: latest` instead.
- The `reboot` module option for `win_feature` has been removed; use the `win_reboot` action plugin instead.
- The `win_iis_webapppool` module no longer accepts a string for the `attributes` module option; use the free form dictionary value instead.
- The `name` module option for `win_package` has been removed; this is not used anywhere and should just be removed from your playbooks.
- The `win_regedit` module no longer automatically corrects the hive path `HCCC` to `HKCC`; use `HKCC` because this is the correct hive path.
- The `file` module now emits a deprecation warning when `src` is specified with a state other than `hard` or `link` as it is only supposed to be useful with those. This could have an effect on people who were depending on a buggy interaction between `src` and other state's to place files into a subdirectory. For instance:

```
$ ansible localhost -m file -a 'path=/var/lib src=/tmp/ state=directory'
```


Would create a directory named `/tmp/lib`. Instead of the above, simply spell out the entire destination path like this:

```
$ ansible localhost -m file -a 'path=/tmp/lib state=directory'
```

- The `k8s_raw` and `openshift_raw` modules have been aliased to the new `k8s` module.
- The `k8s` module supports all Kubernetes resources including those from Custom Resource Definitions and aggregated API servers. This includes all OpenShift resources.
- The `k8s` module will not accept resources where subkeys have been `snake_cased`. This was a workaround that was suggested with the `k8s_raw` and `openshift_raw` modules.
- The `k8s` module may not accept resources where the `api_version` has been changed to match the shortened version in the Kubernetes Python client. You should now specify the proper full Kubernetes `api_version` for a resource.
- The `k8s` module can now process multi-document YAML files if they are passed with the `src` parameter. It will process each document as a separate resource. Resources provided inline with the `resource_definition` parameter must still be a single document.
- The `k8s` module will not automatically change `Project` creation requests into `ProjectRequest` creation requests as the `openshift_raw` module did. You must now specify the `ProjectRequest` kind explicitly.
- The `k8s` module will not automatically remove secrets from the Ansible return values (and by extension the log). In order to prevent secret values in a task from being logged, specify the `no_log` parameter on the task block.
- The `k8s_scale` module now supports scalable OpenShift objects, such as `DeploymentConfig`.
- The `lineinfile` module was changed to show a warning when using an empty string as a regexp. Since an empty regexp matches every line in a file, it will replace the last line in a file rather than inserting. If this is the desired behavior, use `'^'` which will match every line and will not trigger the warning.
- Openstack modules are no longer using `shade` library. Instead `openstacksdk` is used. Since `openstacksdk` should be already present as a dependency to `shade` no additional actions are required.

Plugins

Deprecation notices

The following modules will be removed in Ansible 2.10. Please update your playbooks accordingly.

- `openshift` use `k8s` instead.

Noteworthy plugin changes

- The `k8s` lookup plugin now supports all Kubernetes resources including those from Custom Resource Definitions and aggregated API servers. This includes all OpenShift resources.
- The `k8s` lookup plugin may not accept resources where the `api_version` has been changed to match the shortened version in the Kubernetes Python client. You should now specify the proper full Kubernetes `api_version` for a resource.
- The `k8s` lookup plugin will no longer remove secrets from the Ansible return values (and by extension the log). In order to prevent secret values in a task from being logged, specify the `no_log` parameter on the task block.

Porting custom scripts

No notable changes.

Networking

No notable changes.

Dynamic inventory scripts

- `contrib/inventory/openstack.py` has been renamed to `contrib/inventory/openstack_inventory.py`. If you have used `openstack.py` as a name for your OpenStack dynamic inventory file, change it to `openstack_inventory.py`. Otherwise the file name will conflict with imports from `openstacksdk`.

1.2.6 Ansible 2.5 Porting Guide

This section discusses the behavioral changes between Ansible 2.4 and Ansible 2.5.

It is intended to assist in updating your playbooks, plugins and other parts of your Ansible infrastructure so they will work with this version of Ansible.

We suggest you read this page along with [Ansible Changelog for 2.5](#) to understand what updates you may need to make.

This document is part of a collection on porting. The complete list of porting guides can be found at [porting guides](#).

Topics

- [Ansible 2.5 Porting Guide](#)

- *Playbook*
 - * *Dynamic includes and attribute inheritance*
 - * *Fixed handling of keywords and inline variables*
 - * *Migrating from with_X to loop*
 - * *with_list*
 - * *with_items*
 - * *with_indexed_items*
 - * *with_flattened*
 - * *with_together*
 - * *with_dict*
 - * *with_sequence*
 - * *with_subelements*
 - * *with_nested/with_cartesian*
 - * *with_random_choice*
- *Deprecated*
 - * *Jinja tests used as filters*
 - * *Ansible fact namespacing*
- *Modules*
 - * *github_release*
 - * *Modules removed*
 - * *Deprecation notices*
 - * *Noteworthy module changes*
- *Plugins*
 - * *Inventory*
 - * *Shell*
 - * *Filter*
 - * *Lookup*
- *Porting custom scripts*
- *Network*
 - * *Expanding documentation*

- * *Top-level connection arguments will be removed in 2.9*
- * *Adding persistent connection types `network_cli` and `netconf`*
- * *Developers: Shared Module Utilities Moved*

Playbook

Dynamic includes and attribute inheritance

In Ansible version 2.4, the concept of dynamic includes (`include_tasks`) versus static imports (`import_tasks`) was introduced to clearly define the differences in how `include` works between dynamic and static includes.

All attributes applied to a dynamic `include_*` would only apply to the include itself, while attributes applied to a static `import_*` would be inherited by the tasks within.

This separation was only partially implemented in Ansible version 2.4. As of Ansible version 2.5, this work is complete and the separation now behaves as designed; attributes applied to an `include_*` task will not be inherited by the tasks within.

To achieve an outcome similar to how Ansible worked prior to version 2.5, playbooks should use an explicit application of the attribute on the needed tasks, or use blocks to apply the attribute to many tasks. Another option is to use a static `import_*` when possible instead of a dynamic task.

OLD In Ansible 2.4:

```
- include_tasks: "{{ ansible_distribution }}"
  tags:
    - distro_include
```

Included file:

```
- block:
  - debug:
      msg: "In included file"

  - apt:
      name: nginx
      state: latest
```

NEW In Ansible 2.5:

Including task:

```
- include_tasks: "{{ ansible_distribution }}.yaml"
  tags:
    - distro_include
```

Included file:

```
- block:
  - debug:
      msg: "In included file"

  - apt:
      name: nginx
      state: latest
  tags:
    - distro_include
```

The relevant change in those examples is, that in Ansible 2.5, the included file defines the tag `distro_include` again. The tag is not inherited automatically.

Fixed handling of keywords and inline variables

We made several fixes to how we handle keywords and ‘inline variables’, to avoid conflating the two. Unfortunately these changes mean you must specify whether *name* is a keyword or a variable when calling roles. If you have playbooks that look like this:

```
roles:
  - { role: myrole, name: Justin, othervar: othervalue, become: True}
```

You will run into errors because Ansible reads *name* in this context as a keyword. Beginning in 2.5, if you want to use a variable name that is also a keyword, you must explicitly declare it as a variable for the role:

```
roles:
  - { role: myrole, vars: {name: Justin, othervar: othervalue}, become: True}
```

For a full list of keywords see `playbook_keywords`.

Migrating from `with_X` to `loop`

With the release of Ansible 2.5, the recommended way to perform loops is the use the new `loop` keyword instead of `with_X` style loops.

In many cases, `loop` syntax is better expressed using filters instead of more complex use of `query` or `lookup`.

The following examples will show how to convert many common `with_` style loops to `loop` and filters.

`with_list`

`with_list` is directly replaced by `loop`.

```
- name: with_list
  debug:
    msg: "{{ item }}"
  with_list:
    - one
    - two

- name: with_list -> loop
  debug:
    msg: "{{ item }}"
  loop:
    - one
    - two
```

`with_items`

`with_items` is replaced by `loop` and the `flatten` filter.

```
- name: with_items
  debug:
    msg: "{{ item }}"
  with_items: "{{ items }}"

- name: with_items -> loop
  debug:
    msg: "{{ item }}"
  loop: "{{ items|flatten(levels=1) }}"
```

`with_indexed_items`

`with_indexed_items` is replaced by `loop`, the `flatten` filter and `loop_control.index_var`.

```
- name: with_indexed_items
  debug:
```

(下页继续)

(续上页)

```

    msg: "{{ item.0 }}" - "{{ item.1 }}"
with_indexed_items: "{{ items }}"

- name: with_indexed_items -> loop
  debug:
    msg: "{{ index }}" - "{{ item }}"
  loop: "{{ items|flatten(levels=1) }}"
  loop_control:
    index_var: index

```

with_flattened

with_flattened is replaced by loop and the flatten filter.

```

- name: with_flattened
  debug:
    msg: "{{ item }}"
  with_flattened: "{{ items }}"

- name: with_flattened -> loop
  debug:
    msg: "{{ item }}"
  loop: "{{ items|flatten }}"

```

with_together

with_together is replaced by loop and the zip filter.

```

- name: with_together
  debug:
    msg: "{{ item.0 }}" - "{{ item.1 }}"
  with_together:
    - "{{ list_one }}"
    - "{{ list_two }}"

- name: with_together -> loop
  debug:
    msg: "{{ item.0 }}" - "{{ item.1 }}"
  loop: "{{ list_one|zip(list_two)|list }}"

```

with_dict

with_dict can be substituted by loop and either the dictsort or dict2items filters.

```
- name: with_dict
  debug:
    msg: "{{ item.key }} - {{ item.value }}"
  with_dict: "{{ dictionary }}"

- name: with_dict -> loop (option 1)
  debug:
    msg: "{{ item.key }} - {{ item.value }}"
  loop: "{{ dictionary|dict2items }}"

- name: with_dict -> loop (option 2)
  debug:
    msg: "{{ item.0 }} - {{ item.1 }}"
  loop: "{{ dictionary|dictsort }}"
```

with_sequence

with_sequence is replaced by loop and the range function, and potentially the format filter.

```
- name: with_sequence
  debug:
    msg: "{{ item }}"
  with_sequence: start=0 end=4 stride=2 format=testuser%02x

- name: with_sequence -> loop
  debug:
    msg: "{{ 'testuser%02x' | format(item) }}"
    # range is exclusive of the end point
  loop: "{{ range(0, 4 + 1, 2)|list }}"
```

with_subelements

with_subelements is replaced by loop and the subelements filter.

```
- name: with_subelements
  debug:
```

(下页继续)

(续上页)

```

    msg: "{{ item.0.name }}" - "{{ item.1 }}"
  with_subelements:
    - "{{ users }}"
    - mysql.hosts

- name: with_subelements -> loop
  debug:
    msg: "{{ item.0.name }}" - "{{ item.1 }}"
  loop: "{{ users|subelements('mysql.hosts') }}"

```

with_nested/with_cartesian

with_nested and with_cartesian are replaced by loop and the product filter.

```

- name: with_nested
  debug:
    msg: "{{ item.0 }}" - "{{ item.1 }}"
  with_nested:
    - "{{ list_one }}"
    - "{{ list_two }}"

- name: with_nested -> loop
  debug:
    msg: "{{ item.0 }}" - "{{ item.1 }}"
  loop: "{{ list_one|product(list_two)|list }}"

```

with_random_choice

with_random_choice is replaced by just use of the random filter, without need of loop.

```

- name: with_random_choice
  debug:
    msg: "{{ item }}"
  with_random_choice: "{{ my_list }}"

- name: with_random_choice -> loop (No loop is needed here)
  debug:
    msg: "{{ my_list|random }}"
  tags: random

```

Deprecated

Jinja tests used as filters

Using Ansible-provided jinja tests as filters will be removed in Ansible 2.9.

Prior to Ansible 2.5, jinja tests included within Ansible were most often used as filters. The large difference in use is that filters are referenced as `variable | filter_name` while jinja tests are referenced as `variable is test_name`.

Jinja tests are used for comparisons, while filters are used for data manipulation and have different applications in jinja. This change is to help differentiate the concepts for a better understanding of jinja, and where each can be appropriately used.

As of Ansible 2.5, using an Ansible provided jinja test with filter syntax, will display a deprecation error.

OLD In Ansible 2.4 (and earlier) the use of an Ansible included jinja test would likely look like this:

```
when:
  - result | failed
  - not result | success
```

NEW In Ansible 2.5 it should be changed to look like this:

```
when:
  - result is failed
  - results is not successful
```

In addition to the deprecation warnings, many new tests have been introduced that are aliases of the old tests. These new tests make more sense grammatically with the jinja test syntax, such as the new **successful** test which aliases **success**.

```
when: result is successful
```

See [Tests](#) for more information.

Additionally, a script was created to assist in the conversion for tests using filter syntax to proper jinja test syntax. This script has been used to convert all of the Ansible integration tests to the correct format. There are a few limitations documented, and all changes made by this script should be evaluated for correctness before executing the modified playbooks. The script can be found at https://github.com/ansible/ansible/blob/devel/hacking/fix_test_syntax.py.

Ansible fact namespaces

Ansible facts, which have historically been written to names like `ansible_*` in the main facts namespace, have been placed in their own new namespace, `ansible_facts.*` For example, the fact `ansible_distribution`

is now best queried through the variable structure `ansible_facts.distribution`.

A new configuration variable, `inject_facts_as_vars`, has been added to `ansible.cfg`. Its default setting, `'True'`, keeps the 2.4 behavior of facts variables being set in the old `ansible_*` locations (while also writing them to the new namespace). This variable is expected to be set to `'False'` in a future release. When `inject_facts_as_vars` is set to `False`, you must refer to `ansible_facts` through the new `ansible_facts.*` namespace.

Modules

Major changes in popular modules are detailed here.

github_release

In Ansible versions 2.4 and older, after creating a GitHub release using the `create_release` state, the `github_release` module reported state as `skipped`. In Ansible version 2.5 and later, after creating a GitHub release using the `create_release` state, the `github_release` module now reports state as `changed`.

Modules removed

The following modules no longer exist:

- `nxos_mtu` use `nxos_system`'s `system_mtu` option or `nxos_interface` instead
- `cl_interface_policy` use `nclu` instead
- `cl_bridge` use `nclu` instead
- `cl_img_install` use `nclu` instead
- `cl_ports` use `nclu` instead
- `cl_license` use `nclu` instead
- `cl_interface` use `nclu` instead
- `cl_bond` use `nclu` instead
- `ec2_vpc` use `ec2_vpc_net` along with supporting modules `ec2_vpc_igw`, `ec2_vpc_route_table`, `ec2_vpc_subnet`, `ec2_vpc_dhcp_option`, `ec2_vpc_nat_gateway`, `ec2_vpc_nacl` instead.
- `ec2_ami_search` use `ec2_ami_facts` instead
- `docker` use `docker_container` and `docker_image` instead

注解: These modules may no longer have documentation in the current release. Please see the [Ansible 2.4 module documentation](#) if you need to know how they worked for porting your playbooks.

Deprecation notices

The following modules will be removed in Ansible 2.9. Please update your playbooks accordingly.

- Apstra's `aos_*` modules are deprecated as they do not work with AOS 2.1 or higher. See new modules at <https://github.com/apstra>.
- `nxos_ip_interface` use `nxos_l3_interface` instead.
- `nxos_portchannel` use `nxos_linkagg` instead.
- `nxos_switchport` use `nxos_l2_interface` instead.
- `panos_security_policy` use `panos_security_rule` instead.
- `panos_nat_policy` use `panos_nat_rule` instead.
- `vsphere_guest` use `vmware_guest` instead.

Noteworthy module changes

- The `stat` and `win_stat` modules have changed the default of the option `get_md5` from `true` to `false`.

This option will be removed starting with Ansible version 2.9. The options `get_checksum: True` and `checksum_algorithm: md5` can still be used if an MD5 checksum is desired.

- `osx_say` module was renamed into `say`.
- Several modules which could deal with symlinks had the default value of their `follow` option changed as part of a feature to [standardize the behavior of follow](#):
 - The `file` module changed from `follow=False` to `follow=True` because its purpose is to modify the attributes of a file and most systems do not allow attributes to be applied to symlinks, only to real files.
 - The `replace` module had its `follow` parameter removed because it inherently modifies the content of an existing file so it makes no sense to operate on the link itself.
 - The `blockinfile` module had its `follow` parameter removed because it inherently modifies the content of an existing file so it makes no sense to operate on the link itself.
 - In Ansible-2.5.3, the `template` module became more strict about its `src` file being proper utf-8. Previously, non-utf8 contents in a template module `src` file would result in a mangled output file (the non-utf8 characters would be replaced with a unicode replacement character). Now, on Python2, the module will error out with the message, “Template source files must be utf-8 encoded” . On Python3, the module will first attempt to pass the non-utf8 characters through verbatim and fail if that does not succeed.

Plugins

As a developer, you can now use ‘doc fragments’ for common configuration options on plugin types that support the new plugin configuration system.

Inventory

Inventory plugins have been fine tuned, and we have started to add some common features:

- The ability to use a cache plugin to avoid costly API/DB queries is disabled by default. If using inventory scripts, some may already support a cache, but it is outside of Ansible’s knowledge and control. Moving to the internal cache will allow you to use Ansible’s existing cache refresh/invalidation mechanisms.
- A new ‘auto’ plugin, enabled by default, that can automatically detect the correct plugin to use IF that plugin is using our ‘common YAML configuration format’. The previous `host_list`, `script`, `yaml` and `ini` plugins still work as they did, the auto plugin is now the last one we attempt to use. If you had customized the enabled plugins you should revise the setting to include the new auto plugin.

Shell

Shell plugins have been migrated to the new plugin configuration framework. It is now possible to customize more settings, and settings which were previously ‘global’ can now also be overridden using host specific variables.

For example, `system_temps` is a new setting that allows you to control what Ansible will consider a ‘system temporary dir’. This is used when escalating privileges for a non-administrative user. Previously this was hardcoded to `‘/tmp’`, which some systems cannot use for privilege escalation. This setting now defaults to `[‘/var/tmp’, ‘/tmp’]`.

Another new setting is `admin_users` which allows you to specify a list of users to be considered ‘administrators’. Previously this was hardcoded to `root`. It now it defaults to `[root, toor, admin]`. This information is used when choosing between your `remote_temp` and `system_temps` directory.

For a full list, check the shell plugin you are using, the default shell plugin is `sh`.

Those that had to work around the global configuration limitations can now migrate to a per host/group settings, but also note that the new defaults might conflict with existing usage if the assumptions don’t correlate to your environment.

Filter

The lookup plugin API now throws an error if a non-iterable value is returned from a plugin. Previously, numbers or other non-iterable types returned by a plugin were accepted without error or warning. This change was made because plugins should always return a list. Please note that plugins that return strings

and other non-list iterable values will not throw an error, but may cause unpredictable behavior. If you have a custom lookup plugin that does not return a list, you should modify it to wrap the return values in a list.

Lookup

A new option was added to lookup plugins globally named **error** which allows you to control how errors produced by the lookup are handled, before this option they were always fatal. Valid values for this option are **warn**, **ignore** and **strict**. See the [lookup](#) page for more details.

Porting custom scripts

No notable changes.

Network

Expanding documentation

We're expanding the network documentation. There's new content and a new Ansible Network landing page. We will continue to build the network-related documentation moving forward.

Top-level connection arguments will be removed in 2.9

Top-level connection arguments like **username**, **host**, and **password** are deprecated and will be removed in version 2.9.

OLD In Ansible < 2.4

```
- name: example of using top-level options for connection properties
  ios_command:
    commands: show version
    host: "{{ inventory_hostname }}"
    username: cisco
    password: cisco
    authorize: yes
    auth_pass: cisco
```

The deprecation warnings reflect this schedule. The task above, run in Ansible 2.5, will result in:

```
[DEPRECATION WARNING]: Param 'username' is deprecated. See the module docs for more
↪information. This feature will be removed in version
2.9. Deprecation warnings can be disabled by setting deprecation_warnings=False in
↪ansible.cfg.
```

(下页继续)

(续上页)

```
[DEPRECATION WARNING]: Param 'password' is deprecated. See the module docs for more
↪information. This feature will be removed in version
2.9. Deprecation warnings can be disabled by setting deprecation_warnings=False in
↪ansible.cfg.
[DEPRECATION WARNING]: Param 'host' is deprecated. See the module docs for more
↪information. This feature will be removed in version 2.9.
Deprecation warnings can be disabled by setting deprecation_warnings=False in ansible.
↪cfg.
```

We recommend using the new connection types `network_cli` and `netconf` (see below), using standard Ansible connection properties, and setting those properties in inventory by group. As you update your playbooks and inventory files, you can easily make the change to `become` for privilege escalation (on platforms that support it). For more information, see the [using become with network modules](#) guide and the [platform documentation](#).

Adding persistent connection types `network_cli` and `netconf`

Ansible 2.5 introduces two top-level persistent connection types, `network_cli` and `netconf`. With `connection: local`, each task passed the connection parameters, which had to be stored in your playbooks. With `network_cli` and `netconf` the playbook passes the connection parameters once, so you can pass them at the command line if you prefer. We recommend you use `network_cli` and `netconf` whenever possible. Note that eAPI and NX-API still require `local` connections with `provider` dictionaries. See the [platform documentation](#) for more information. Unless you need a `local` connection, update your playbooks to use `network_cli` or `netconf` and to specify your connection variables with standard Ansible connection variables:

OLD In Ansible 2.4

```
---
vars:
  cli:
    host: "{{ inventory_hostname }}"
    username: operator
    password: secret
    transport: cli

tasks:
- nxos_config:
  src: config.j2
  provider: "{{ cli }}"
```

(下页继续)

(续上页)

```
username: admin
password: admin
```

NEW In Ansible 2.5

```
[nxos:vars]
ansible_connection=network_cli
ansible_network_os=nxos
ansible_user=operator
ansible_password=secret
```

```
tasks:
- nxos_config:
    src: config.j2
```

Using a provider dictionary with either `network_cli` or `netconf` will result in a warning.

Developers: Shared Module Utilities Moved

Beginning with Ansible 2.5, shared module utilities for network modules moved to `ansible.module_utils.network`.

- Platform-independent utilities are found in `ansible.module_utils.network.common`
- Platform-specific utilities are found in `ansible.module_utils.network.{{ platform }}`

If your module uses shared module utilities, you must update all references. For example, change:

OLD In Ansible 2.4

```
from ansible.module_utils.vyos import get_config, load_config
```

NEW In Ansible 2.5

```
from ansible.module_utils.network.vyos.vyos import get_config, load_config
```

See the module utilities developer guide see *Using and Developing Module Utilities* for more information.

1.2.7 Ansible 2.4 Porting Guide

This section discusses the behavioral changes between Ansible 2.3 and Ansible 2.4.

It is intended to assist in updating your playbooks, plugins and other parts of your Ansible infrastructure so they will work with this version of Ansible.

We suggest you read this page along with [Ansible Changelog for 2.4](#) to understand what updates you may need to make.

This document is part of a collection on porting. The complete list of porting guides can be found at [porting guides](#).

Topics

- *Ansible 2.4 Porting Guide*
 - *Python version*
 - *Inventory*
 - * *Initial playbook relative group_vars and host_vars*
 - *Deprecated*
 - * *Specifying Inventory sources*
 - * *Use of multiple tags*
 - * *Other caveats*
 - *Modules*
 - * *Modules removed*
 - * *Deprecation notices*
 - * *Noteworthy module changes*
 - *Plugins*
 - * *Vars plugin changes*
 - * *Inventory plugins*
 - * *Callback plugins*
 - * *Template lookup plugin: Escaping Strings*
 - *Tests*
 - * *Tests succeeded/failed*
 - *Networking*
 - * *Persistent Connection*
 - *Configuration*

Python version

Ansible will not support Python 2.4 or 2.5 on the target hosts anymore. Going forward, Python 2.6+ will be required on targets, as already is the case on the controller.

Inventory

Inventory has been refactored to be implemented via plugins and now allows for multiple sources. This change is mostly transparent to users.

One exception is the `inventory_dir`, which is now a host variable; previously it could only have one value so it was set globally. This means you can no longer use it early in plays to determine `hosts:` or similar keywords. This also changes the behaviour of `add_hosts` and the implicit localhost; because they no longer automatically inherit the global value, they default to `None`. See the module documentation for more information.

The `inventory_file` remains mostly unchanged, as it was always host specific.

Since there is no longer a single inventory, the ‘implicit localhost’ doesn’t get either of these variables defined.

A bug was fixed with the inventory path/directory, which was defaulting to the current working directory. This caused `group_vars` and `host_vars` to be picked up from the current working directory instead of just adjacent to the playbook or inventory directory when a host list (comma separated host names) was provided as inventory.

Initial playbook relative `group_vars` and `host_vars`

In Ansible versions prior to 2.4, the inventory system would maintain the context of the initial playbook that was executed. This allowed successively included playbooks from other directories to inherit `group_vars` and `host_vars` placed relative to the top level playbook file.

Due to some behavioral inconsistencies, this functionality will not be included in the new inventory system starting with Ansible version 2.4.

Similar functionality can still be achieved by using `vars_files`, `include_vars`, or `group_vars` and `host_vars` placed relative to the inventory file.

Deprecated

Specifying Inventory sources

Use of `--inventory-file` on the command line is now deprecated. Use `--inventory` or `-i`. The associated ini configuration key, `hostfile`, and environment variable, `ANSIBLE_HOSTS`, are also deprecated. Replace them with the configuration key `inventory` and environment variable `ANSIBLE_INVENTORY`.

Use of multiple tags

Specifying `--tags` (or `--skip-tags`) multiple times on the command line currently leads to the last one overriding all the previous ones. This behavior is deprecated. In the future, if you specify `-tags` multiple times the tags will be merged together. From now on, using `--tags` multiple times on one command line will emit a deprecation warning. Setting the `merge_multiple_cli_tags` option to `True` in the `ansible.cfg` file will enable the new behavior.

In 2.4, the default has change to merge the tags. You can enable the old overwriting behavior via the `config` option.

In 2.5, multiple `--tags` options will be merged with no way to go back to the old behavior.

Other caveats

No major changes in this version.

Modules

Major changes in popular modules are detailed here

- The `win_shell` and `win_command` modules now properly preserve quoted arguments in the command-line. Tasks that attempted to work around the issue by adding extra quotes/escaping may need to be reworked to remove the superfluous escaping. See [Issue 23019](#) for additional detail.

Modules removed

The following modules no longer exist:

- None

Deprecation notices

The following modules will be removed in Ansible 2.8. Please update your playbooks accordingly.

- `azure`, use `azure_rm_virtualmachine`, which uses the new Resource Manager SDK.
- `win_msi`, use `win_package` instead

Noteworthy module changes

- The `win_get_url` module has the dictionary `win_get_url` in its results deprecated, its content is now also available directly in the resulting output, like other modules. This dictionary will be removed in Ansible 2.8.

- The `win_unzip` module no longer includes the dictionary `win_unzip` in its results; the contents are now included directly in the resulting output, like other modules.
- The `win_package` module return values `exit_code` and `restart_required` have been deprecated in favour of `rc` and `reboot_required` respectively. The deprecated return values will be removed in Ansible 2.6.

Plugins

A new way to configure and document plugins has been introduced. This does not require changes to existing setups but developers should start adapting to the new infrastructure now. More details will be available in the developer documentation for each plugin type.

Vars plugin changes

There have been many changes to the implementation of vars plugins, but both users and developers should not need to change anything to keep current setups working. Developers should consider changing their plugins take advantage of new features.

The most notable difference to users is that vars plugins now get invoked on demand instead of at inventory build time. This should make them more efficient for large inventories, especially when using a subset of the hosts.

注解:

- This also creates a difference with `group/host_vars` when using them adjacent to playbooks. Before, the ‘first’ playbook loaded determined the variables; now the ‘current’ playbook does. We are looking to fix this soon, since ‘all playbooks’ in the path should be considered for variable loading.
 - In 2.4.1 we added a toggle to allow you to control this behaviour, ‘top’ will be the pre 2.4, ‘bottom’ will use the current playbook hosting the task and ‘all’ will use them all from top to bottom.
-

Inventory plugins

Developers should start migrating from hardcoded inventory with dynamic inventory scripts to the new Inventory Plugins. The scripts will still work via the `script` inventory plugin but Ansible development efforts will now concentrate on writing plugins rather than enhancing existing scripts.

Both users and developers should look into the new plugins because they are intended to alleviate the need for many of the hacks and workarounds found in the dynamic inventory scripts.

Callback plugins

Users:

- Callbacks are now using the new configuration system. Users should not need to change anything as the old system still works, but you might see a deprecation notice if any callbacks used are not inheriting from the built in classes. Developers need to update them as stated below.

Developers:

- If your callback does not inherit from `CallbackBase` (directly or indirectly via another callback), it will still work, but issue a deprecation notice. To avoid this and ensure it works in the future change it to inherit from `CallbackBase` so it has the new options handling methods and properties. You can also implement the new options handling methods and properties but that won't automatically inherit changes added in the future. You can look at `CallbackBase` itself and/or `AnsiblePlugin` for details.
- Any callbacks inheriting from other callbacks might need to also be updated to contain the same documented options as the parent or the options won't be available. This is noted in the developer guide.

Template lookup plugin: Escaping Strings

Prior to Ansible 2.4, backslashes in strings passed to the template lookup plugin would be escaped automatically. In 2.4, users are responsible for escaping backslashes themselves. This change brings the template lookup plugin inline with the template module so that the same backslash escaping rules apply to both.

If you have a template lookup like this:

```
- debug:
  msg: '{{ lookup("template", "template.j2") }}
```

OLD In Ansible 2.3 (and earlier) `template.j2` would look like this:

```
{{ "name surname" | regex_replace("[^\\s]+\\s+(.*)", "\\1") }}
```

NEW In Ansible 2.4 it should be changed to look like this:

```
{{ "name surname" | regex_replace("[^\\s]+\\s+(.*)", "\\1") }}
```

Tests

Tests succeeded/failed

Prior to Ansible version 2.4, a task return code of `rc` would override a return code of `failed`. In version 2.4, both `rc` and `failed` are used to calculate the state of the task. Because of this, test plugins

succeeded/failed` have also been changed. This means that overriding a task failure with `failed_when: no` will result in `succeeded/failed` returning `True/False`. For example:

```
- command: /bin/false
  register: result
  failed_when: no

- debug:
  msg: 'This is printed on 2.3'
  when: result|failed

- debug:
  msg: 'This is printed on 2.4'
  when: result|succeeded

- debug:
  msg: 'This is always printed'
  when: result.rc != 0
```

As we can see from the example above, in Ansible 2.3 `succeeded/failed` only checked the value of `rc`.

Networking

There have been a number of changes to how Networking Modules operate.

Playbooks should still use `connection: local`.

Persistent Connection

The configuration variables `connection_retries` and `connect_interval` which were added in Ansible 2.3 are now deprecated. For Ansible 2.4 and later use `connection_retry_timeout`.

To control timeouts use `command_timeout` rather than the previous top level `timeout` variable under `[default]`

See *Ansible Network debug guide* for more information.

Configuration

The configuration system has had some major changes. Users should be unaffected except for the following:

- All relative paths defined are relative to the *ansible.cfg* file itself. Previously they varied by setting. The new behavior should be more predictable.

- A new macro `{{CWD}}` is available for paths, which will make paths relative to the ‘current working directory’, this is unsafe but some users really want to rely on this behaviour.

Developers that were working directly with the previous API should revisit their usage as some methods (for example, `get_config`) were kept for backwards compatibility but will warn users that the function has been deprecated.

The new configuration has been designed to minimize the need for code changes in core for new plugins. The plugins just need to document their settings and the configuration system will use the documentation to provide what they need. This is still a work in progress; currently only ‘callback’ and ‘connection’ plugins support this. More details will be added to the specific plugin developer guides.

1.2.8 Ansible 2.3 Porting Guide

This section discusses the behavioral changes between Ansible 2.2 and Ansible 2.3.

It is intended to assist in updating your playbooks, plugins and other parts of your Ansible infrastructure so they will work with this version of Ansible.

We suggest you read this page along with [Ansible Changelog for 2.3](#) to understand what updates you may need to make.

This document is part of a collection on porting. The complete list of porting guides can be found at [porting guides](#).

Topics

- *Ansible 2.3 Porting Guide*
 - *Playbook*
 - * *Restructured async to work with action plugins*
 - * *OpenBSD version facts*
 - * *Names Blocks*
 - * *Use of multiple tags*
 - * *Other caveats*
 - *Modules*
 - * *Modules removed*
 - * *Deprecation notices*
 - * *Noteworthy module changes*
 - *AWS lambda*

- *Mount*
- *Plugins*
- *Porting custom scripts*
- *Networking*
 - * *Deprecation of top-level connection arguments*
 - * *delegate_to vs ProxyCommand*

Playbook

Restructued async to work with action plugins

In Ansible 2.2 (and possibly earlier) the *async:* keyword could not be used in conjunction with the action plugins such as *service*. This limitation has been removed in Ansible 2.3

NEW In Ansible 2.3:

```
- name: Install nginx asynchronously
  service:
    name: nginx
    state: restarted
  async: 45
```

OpenBSD version facts

The *ansible_distribution_release* and *ansible_distribution_version* facts on OpenBSD hosts were reversed in Ansible 2.2 and earlier. This has been changed so that version has the numeric portion and release has the name of the release.

OLD In Ansible 2.2 (and earlier)

```
"ansible_distribution": "OpenBSD",
"ansible_distribution_release": "6.0",
"ansible_distribution_version": "release",
```

NEW In Ansible 2.3:

```
"ansible_distribution": "OpenBSD",
"ansible_distribution_release": "release",
"ansible_distribution_version": "6.0",
```


Names Blocks

Blocks can now have names, this allows you to avoid the ugly *# this block is for...* comments.

NEW In Ansible 2.3:

```
- name: Block test case
  hosts: localhost
  tasks:
    - name: Attempt to setup foo
      block:
        - debug: msg='I execute normally'
        - command: /bin/false
        - debug: msg='I never execute, cause ERROR!'
      rescue:
        - debug: msg='I caught an error'
        - command: /bin/false
        - debug: msg='I also never execute :-( '
      always:
        - debug: msg="this always executes"
```

Use of multiple tags

Specifying `--tags` (or `--skip-tags`) multiple times on the command line currently leads to the last specified tag overriding all the other specified tags. This behaviour is deprecated. In the future, if you specify `-tags` multiple times the tags will be merged together. From now on, using `--tags` multiple times on one command line will emit a deprecation warning. Setting the `merge_multiple_cli_tags` option to `True` in the `ansible.cfg` file will enable the new behaviour.

In 2.4, the default will be to merge the tags. You can enable the old overwriting behavior via the `config` option. In 2.5, multiple `--tags` options will be merged with no way to go back to the old behaviour.

Other caveats

Here are some rare cases that might be encountered when updating. These are mostly caused by the more stringent parser validation and the capture of errors that were previously ignored.

- Made `any_errors_fatal` inheritable from play to task and all other objects in between.

Modules

No major changes in this version.

Modules removed

No major changes in this version.

Deprecation notices

The following modules will be removed in Ansible 2.5. Please update your playbooks accordingly.

- `ec2_vpc`
- `cl_bond`
- `cl_bridge`
- `cl_img_install`
- `cl_interface`
- `cl_interface_policy`
- `cl_license`
- `cl_ports`
- `nxos_mtu` use `nxos_system` instead

注解: These modules may no longer have documentation in the current release. Please see the [Ansible 2.3 module documentation](#) if you need to know how they worked for porting your playbooks.

Noteworthy module changes

AWS lambda

Previously ignored changes that only affected one parameter. Existing deployments may have outstanding changes that this bug fix will apply.

Mount

Mount: Some fixes so bind mounts are not mounted each time the playbook runs.

Plugins

No major changes in this version.

Porting custom scripts

No major changes in this version.

Networking

There have been a number of changes to number of changes to how Networking Modules operate.

Playbooks should still use `connection: local`.

The following changes apply to:

- dellos6
- dellos9
- dellos10
- eos
- ios
- iosxr
- junos
- sros
- vyos

Deprecation of top-level connection arguments

OLD In Ansible 2.2:

```
- name: example of using top-level options for connection properties
  ios_command:
    commands: show version
    host: "{{ inventory_hostname }}"
    username: cisco
    password: cisco
    authorize: yes
    auth_pass: cisco
```

Will result in:

```
[WARNING]: argument username has been deprecated and will be removed in a future version
[WARNING]: argument host has been deprecated and will be removed in a future version
[WARNING]: argument password has been deprecated and will be removed in a future version
```

NEW In Ansible 2.3:

```
- name: Gather facts
  eos_facts:
    gather_subset: all
    provider:
      username: myuser
      password: "{{ networkpassword }}"
      transport: cli
      host: "{{ ansible_host }}"
```

delegate_to vs ProxyCommand

The new connection framework for Network Modules in Ansible 2.3 that uses `cli` transport no longer supports the use of the `delegate_to` directive. In order to use a bastion or intermediate jump host to connect to network devices over `cli` transport, network modules now support the use of `ProxyCommand`.

To use `ProxyCommand` configure the proxy settings in the Ansible inventory file to specify the proxy host via `ansible_ssh_common_args`.

For details on how to do this see the [network proxy guide](#).

1.2.9 Ansible 2.0 迁移指南

This section discusses the behavioral changes between Ansible 1.x and Ansible 2.0.

It is intended to assist in updating your playbooks, plugins and other parts of your Ansible infrastructure so they will work with this version of Ansible.

We suggest you read this page along with [Ansible Changelog for 2.0](#) to understand what updates you may need to make.

This document is part of a collection on porting. The complete list of porting guides can be found at [porting guides](#).

Topics

- [Ansible 2.0 迁移指南](#)
 - [Playbook](#)
 - * [Deprecated](#)
 - * [Other caveats](#)
 - [Porting plugins](#)

- * *Lookup plugins*
- * *Connection plugins*
- * *Action plugins*
- * *Callback plugins*
- * *Connection plugins*
- *Hybrid plugins*
 - * *Lookup plugins*
 - * *Connection plugins*
 - * *Action plugins*
 - * *Callback plugins*
 - * *Connection plugins*
- *Porting custom scripts*

Playbook

This section discusses any changes you may need to make to your playbooks.

```
# Syntax in 1.9.x
- debug:
    msg: "{{ 'test1_junk 1\\\\\\3' | regex_replace('(.*)_junk (.*)', '\\\\1 \\\\2') }}"

# Syntax in 2.0.x
- debug:
    msg: "{{ 'test1_junk 1\\\\3' | regex_replace('(.*)_junk (.*)', '\\1 \\\\2') }}"

# Output:
"msg": "test1 1\\\\3"
```

To make an escaped string that will work on all versions you have two options:

```
- debug: msg="{{ 'test1_junk 1\\\\3' | regex_replace('(.*)_junk (.*)', '\\1 \\\\2') }}"
```

uses key=value escaping which has not changed. The other option is to check for the ansible version:

```
"{{ (ansible_version|version_compare('2.0', 'ge'))|ternary( 'test1_junk 1\\\\3' | regex_
↪replace('(.*)_junk (.*)', '\\1 \\\\2') , 'test1_junk 1\\\\\\\\3' | regex_replace('(.*)_junk_
↪(.*)', '\\\\\\\\1 \\\\\\\2') ) }}"
```

- trailing newline When a string with a trailing newline was specified in the playbook via yaml dict

format, the trailing newline was stripped. When specified in key=value format, the trailing newlines were kept. In v2, both methods of specifying the string will keep the trailing newlines. If you relied on the trailing newline being stripped, you can change your playbook using the following as an example:

```
# Syntax in 1.9.x
vars:
  message: >
    Testing
    some things
tasks:
- debug:
  msg: "{{ message }}"

# Syntax in 2.0.x
vars:
  old_message: >
    Testing
    some things
  message: "{{ old_message[:-1] }}"
- debug:
  msg: "{{ message }}"

# Output
"msg": "Testing some things"
```

- Behavior of templating DOS-type text files changes with Ansible v2.

A bug in Ansible v1 causes DOS-type text files (using a carriage return and newline) to be templated to Unix-type text files (using only a newline). In Ansible v2 this long-standing bug was finally fixed and DOS-type text files are preserved correctly. This may be confusing when you expect your playbook to not show any differences when migrating to Ansible v2, while in fact you will see every DOS-type file being completely replaced (with what appears to be the exact same content).

- When specifying complex args as a variable, the variable must use the full jinja2 variable syntax (`('{{var_name}}')`) - bare variable names there are no longer accepted. In fact, even specifying args with variables has been deprecated, and will not be allowed in future versions:

```
---
- hosts: localhost
  connection: local
  gather_facts: false
  vars:
    my_dirs:
      - { path: /tmp/3a, state: directory, mode: 0755 }
```

(下页继续)

(续上页)

```

- { path: /tmp/3b, state: directory, mode: 0700 }
tasks:
- file:
  args: "{{item}}" # <- args here uses the full variable syntax
  with_items: "{{my_dirs}}"

```

- porting task includes
- More dynamic. Corner-case formats that were not supposed to work now do not, as expected.
- variables defined in the yaml dict format <https://github.com/ansible/ansible/issues/13324>
- templating (variables in playbooks and template lookups) has improved with regard to keeping the original instead of turning everything into a string. If you need the old behavior, quote the value to pass it around as a string.
- Empty variables and variables set to null in yaml are no longer converted to empty strings. They will retain the value of *None*. You can override the *null_representation* setting to an empty string in your config file by setting the `ANSIBLE_NULL_REPRESENTATION` environment variable.
- Extras callbacks must be whitelisted in `ansible.cfg`. Copying is no longer necessary but whitelisting in `ansible.cfg` must be completed.
- dnf module has been rewritten. Some minor changes in behavior may be observed.
- win_updates has been rewritten and works as expected now.
- from 2.0.1 onwards, the implicit setup task from `gather_facts` now correctly inherits everything from `play`, but this might cause issues for those setting *environment* at the play level and depending on *ansible_env* existing. Previously this was ignored but now might issue an ‘Undefined’ error.

Deprecated

While all items listed here will show a deprecation warning message, they still work as they did in 1.9.x. Please note that they will be removed in 2.2 (Ansible always waits two major releases to remove a deprecated feature).

- Bare variables in `with_` loops should instead use the `"{{ var }}"` syntax, which helps eliminate ambiguity.
- The `ansible-galaxy` text format requirements file. Users should use the YAML format for requirements instead.
- Undefined variables within a `with_` loop’s list currently do not interrupt the loop, but they do issue a warning; in the future, they will issue an error.
- Using dictionary variables to set all task parameters is unsafe and will be removed in a future version. For example:

```

- hosts: localhost
  gather_facts: no
  vars:
    debug_params:
      msg: "hello there"
  tasks:
    # These are both deprecated:
    - debug: "{{debug_params}}"
    - debug:
      args: "{{debug_params}}"

    # Use this instead:
    - debug:
      msg: "{{debug_params['msg'] }}"

```

- Host patterns should use a comma (,) or colon (:) instead of a semicolon (;) to separate hosts/groups in the pattern.
- Ranges specified in host patterns should use the [x:y] syntax, instead of [x-y].
- Playbooks using privilege escalation should always use “become*” options rather than the old su*/sudo* options.
- The “short form” for vars_prompt is no longer supported. For example:

```

vars_prompt:
  variable_name: "Prompt string"

```

- Specifying variables at the top level of a task include statement is no longer supported. For example:

```

- include_tasks: foo.yml
  a: 1

```

Should now be:

```

- include_tasks: foo.yml
  vars:
    a: 1

```

- Setting any_errors_fatal on a task is no longer supported. This should be set at the play level only.
- Bare variables in the *environment* dictionary (for plays/tasks/etc.) are no longer supported. Variables specified there should use the full variable syntax: ‘{{foo}}’.
- Tags (or any directive) should no longer be specified with other parameters in a task include. Instead, they should be specified as an option on the task. For example:


```
- include_tasks: foo.yml tags=a,b,c
```

Should be:

```
- include_tasks: foo.yml
  tags: [a, b, c]
```

- The `first_available_file` option on tasks has been deprecated. Users should use the `with_first_found` option or `lookup ('first_found' , ...)` plugin.

Other caveats

Here are some corner cases encountered when updating. These are mostly caused by the more stringent parser validation and the capture of errors that were previously ignored.

- Bad variable composition:

```
with_items: myvar_{{rest_of_name}}
```

This worked ‘by accident’ as the errors were retemplated and ended up resolving the variable, it was never intended as valid syntax and now properly returns an error, use the following instead.:

```
hostvars[inventory_hostname]['myvar_' + rest_of_name]
```

- Misspelled directives:

```
- task: dostuf
  becom: yes
```

The task always ran without using privilege escalation (for that you need *become*) but was also silently ignored so the play ‘ran’ even though it should not, now this is a parsing error.

- Duplicate directives:

```
- task: dostuf
  when: True
  when: False
```

The first *when* was ignored and only the 2nd one was used as the play ran w/o warning it was ignoring one of the directives, now this produces a parsing error.

- Conflating variables and directives:

```
- role: {name=rosy, port=435 }  
  
# in tasks/main.yml  
- wait_for: port={{port}}
```

The *port* variable is reserved as a play/task directive for overriding the connection port, in previous versions this got conflated with a variable named *port* and was usable later in the play, this created issues if a host tried to reconnect or was using a non caching connection. Now it will be correctly identified as a directive and the *port* variable will appear as undefined, this now forces the use of non conflicting names and removes ambiguity when adding settings and variables to a role invocation.

- Bare operations on *with_*:

```
with_items: var1 + var2
```

An issue with the ‘bare variable’ features, which was supposed only template a single variable without the need of braces (`{{ }}`), would in some versions of Ansible template full expressions. Now you need to use proper templating and braces for all expressions everywhere except conditionals (*when*):

```
with_items: "{{var1 + var2}}"
```

The bare feature itself is deprecated as an undefined variable is indistinguishable from a string which makes it difficult to display a proper error.

Porting plugins

In ansible-1.9.x, you would generally copy an existing plugin to create a new one. Simply implementing the methods and attributes that the caller of the plugin expected made it a plugin of that type. In ansible-2.0, most plugins are implemented by subclassing a base class for each plugin type. This way the custom plugin does not need to contain methods which are not customized.

Lookup plugins

- lookup plugins ; import version

Connection plugins

- connection plugins

Action plugins

- action plugins

Callback plugins

Although Ansible 2.0 provides a new callback API the old one continues to work for most callback plugins. However, if your callback plugin makes use of `self.playbook`, `self.play`, or `self.task` then you will have to store the values for these yourself as ansible no longer automatically populates the callback with them. Here's a short snippet that shows you how:

```
import os
from ansible.plugins.callback import CallbackBase

class CallbackModule(CallbackBase):
    def __init__(self):
        self.playbook = None
        self.playbook_name = None
        self.play = None
        self.task = None

    def v2_playbook_on_start(self, playbook):
        self.playbook = playbook
        self.playbook_name = os.path.basename(self.playbook._file_name)

    def v2_playbook_on_play_start(self, play):
        self.play = play

    def v2_playbook_on_task_start(self, task, is_conditional):
        self.task = task

    def v2_on_any(self, *args, **kwargs):
        self._display.display('%s: %s: %s' % (self.playbook_name,
        self.play.name, self.task))
```

Connection plugins

- connection plugins

Hybrid plugins

In specific cases you may want a plugin that supports both ansible-1.9.x *and* ansible-2.0. Much like porting plugins from v1 to v2, you need to understand how plugins work in each version and support both requirements.

Since the ansible-2.0 plugin system is more advanced, it is easier to adapt your plugin to provide similar pieces (subclasses, methods) for ansible-1.9.x as ansible-2.0 expects. This way your code will look a lot cleaner.

You may find the following tips useful:

- Check whether the ansible-2.0 class(es) are available and if they are missing (ansible-1.9.x) mimic them with the needed methods (e.g. `__init__`)
- When ansible-2.0 python modules are imported, and they fail (ansible-1.9.x), catch the `ImportError` exception and perform the equivalent imports for ansible-1.9.x. With possible translations (e.g. importing specific methods).
- Use the existence of these methods as a qualifier to what version of Ansible you are running. So rather than using version checks, you can do capability checks instead. (See examples below)
- Document for each if-then-else case for which specific version each block is needed. This will help others to understand how they have to adapt their plugins, but it will also help you to remove the older ansible-1.9.x support when it is deprecated.
- When doing plugin development, it is very useful to have the `warning()` method during development, but it is also important to emit warnings for deadends (cases that you expect should never be triggered) or corner cases (e.g. cases where you expect misconfigurations).
- It helps to look at other plugins in ansible-1.9.x and ansible-2.0 to understand how the API works and what modules, classes and methods are available.

Lookup plugins

As a simple example we are going to make a hybrid `fileglob` lookup plugin.

```
from __future__ import (absolute_import, division, print_function)
__metaclass__ = type

import os
import glob

try:
    # ansible-2.0
    from ansible.plugins.lookup import LookupBase
except ImportError:
    # ansible-1.9.x

    class LookupBase(object):
        def __init__(self, basedir=None, runner=None, **kwargs):
```

(下页继续)

(续上页)

```

        self.runner = runner
        self.basedir = self.runner.basedir

    def get_basedir(self, variables):
        return self.basedir

try:
    # ansible-1.9.x
    from ansible.utils import (listify_lookup_plugin_terms, path_dwim, warning)
except ImportError:
    # ansible-2.0
    from ansible.utils.display import Display
    warning = Display().warning

class LookupModule(LookupBase):

    # For ansible-1.9.x, we added inject=None as valid argument
    def run(self, terms, inject=None, variables=None, **kwargs):

        # ansible-2.0, but we made this work for ansible-1.9.x too !
        basedir = self.get_basedir(variables)

        # ansible-1.9.x
        if 'listify_lookup_plugin_terms' in globals():
            terms = listify_lookup_plugin_terms(terms, basedir, inject)

        ret = []
        for term in terms:
            term_file = os.path.basename(term)

            # For ansible-1.9.x, we imported path_dwim() from ansible.utils
            if 'path_dwim' in globals():
                # ansible-1.9.x
                dwimmed_path = path_dwim(basedir, os.path.dirname(term))
            else:
                # ansible-2.0
                dwimmed_path = self._loader.path_dwim_relative(basedir, 'files', os.path.
↳dirname(term))

            globbed = glob.glob(os.path.join(dwimmed_path, term_file))

```

(下页继续)

(续上页)

```
ret.extend(g for g in globbed if os.path.isfile(g))

return ret
```

注解: In the above example we did not use the `warning()` method as we had no direct use for it in the final version. However we left this code in so people can use this part during development/porting/use.

Connection plugins

- connection plugins

Action plugins

- action plugins

Callback plugins

- callback plugins

Connection plugins

- connection plugins

Porting custom scripts

Custom scripts that used the `ansible.runner.Runner` API in 1.x have to be ported in 2.x. Please refer to: *Python API*

1.3 用户指南

欢迎使用 Ansible User Guide!

该页内容包括命令行使用、配置 inventory、编写 playbooks.

1.3.1 Ansible 快速入门指南

官方录制了一段大概 13 分钟快速入门指南的视频: [quickstart video](#)。简介了 Ansible 是什么以及如何快速使用。同时, 也会介绍一些 Ansible 生态系统中的一些其它产品。

Enjo 的同时, 还有很多其它文档需要学习才能更好的使用 Ansible.

参见:

‘系统管理员上手指南’ <<https://www.redhat.com/en/blog/system-administrators-guide-getting-started-ansible>> 一步步带你上手 Ansible

‘系统管理员自动化安全’ <<https://opensource.com/downloads/ansible-quickstart>> ‘__’ 可以下载到本地的手册

Network Automation Getting Started 网络工程师首次使用 Ansible 指南

1.3.2 Ansible 概览

这部分内容对所有用户均有用。你需要了解 Ansible 的使用背景才能在不同场景下使用他的自动化功能。本页内容是你深入理解其它内容的基础。

- 管理机
- 受控节点
- *Inventory* 仓库
- *Modules* 模块
- *Tasks* 任务
- *Playbooks* 任务剧本

管理机

任何安装了 Ansible 的服务器, 你都可以使用 `ansible` 或 `ansible-playbook` 命令。任何安装了 Ansible 的机器都可以做为管理节点, 便携式计算机, 共享桌面和服务器都可以。你可以配置多个管理节点。唯一需要注意的是, 管理节点不支持 Windows 系统。

受控节点

Ansible 管理的服务器或者网络设备都称为受控节点。受控节点有时候也叫做 “hosts” (主机)。受控节点不需要安装 Ansible.

Inventory 仓库

Inventory 仓库是保存受控节点信息的列表, 因为有时候也叫 “hostfile”, 类似于系统的 hosts 文件。Inventory 仓库可以以 IP 的方式指定受控节点。Inventory 同样可以组织管理节点、新增、嵌套组等方式, 非常便于扩展。更多请参考[the Working with Inventory](#)

Modules 模块

Modules 模块是 Ansible 执行代码的最小单元。每个模块都是特殊用途, 从特殊类型的数据库用户管理, 到特殊类型的网络设备 VLAN 接口管理。你可以在通过执行单个任务调用一个模块, 也可以通过 playbook 同时调用执行钩具模块。在链接中查看 Ansible 总共包括了多少个模块。[:ref:‘模块列表 <modules_by_category>’](#)

Tasks 任务

Ansible 执行操作的最小单位。ad-hoc 更适合临时执行命令的执行场景。

Playbooks 任务剧本

Playbooks 是任务列表的组合, 通常会把常用的命令列表通过正确的语法写入到 playbook 中。Playbook 可以像普通 tasks 一样调用变量, 其使用 YAML 语法, 便于读、写、分享、理解。更多请参考[Intro to Playbooks](#).

1.3.3 入门

假设你已经阅读了[安装指南](#) 安装好了管理节点并且了解了 Ansible 是如何工作的, 那么你可以开始基本的 Ansible 入门操

- 从 inventory 仓库中选择要执行命令的对象
- 连接测试这些节点 (或者网络设备, 或者其它受控节点)connects to those machines (or network devices, or other managed nodes)。通常使用 SSH 的方式
- 复制一个或多个模块到远程计算机并尝试执行

Ansible 的实际功能更强大, 但开始之前你需要了解他的基础用法才能发掘其它更强大的功能, 比如配置、部署、编排等。本页内容会通过简单的 Inventory 配置和 ad-hoc 命令来演示其基本用法。inventory 参考:[inventory](#), 更充分了解 Ansible 点击[这里playbooks](#).

- 选择指定机器执行
 - 行动: 创建基础清单
 - 基础进阶
- 连接远程受控节点

- 行动：检查 *SSH* 连接
 - 基础进阶
- 复制和执行模块
 - 行动：Ansible 初体验
 - 基础进阶
- 接下来

选择指定机器执行

Ansible 是通过读取 Inventory 中的配置知道我们要对哪些机器变更。虽然你可以在命令行使用 `ad-hoc` 临时命令时指定 IP 地址的方式来控制要操作的对象，但如果想充分使用 Ansible 的灵活性和或扩展性，你必须掌握 Inventory 的配置

行动：创建基础清单

创建 `/etc/ansible/hosts` 并添加一些主机列表。使用 IP 地址或者主机名均可：

```
192.0.2.50
aserver.example.org
bserver.example.org
```

基础进阶

Inventory 不仅可以存放 IPs 和主机名，也可以创建别名，点击查看 [aliases](#), set variable values for a single host with *host vars*, or set variable values for multiple hosts with *group vars*.

连接远程受控节点

Ansible 和远程主机的通信是通过 *SSH protocol*。默认 Ansible 使用开源软件 OpenSSH 通过当前用户连接远程主机。

行动：检查 SSH 连接

确认通过相同的用户可以连接到所有的受控节点，有必要的話，你可以手动添加公钥到对应主机的“`authorized_keys`”。

基础进阶

有如下几种方式指定用户连接远程受控节点:

- 在命令行使用 `-u` 指定用户
- 在 Inventory 是指定连接用户
- 在配置文件中设置连接用户
- 设置环境变量

点击[Controlling how Ansible behaves: precedence rules](#) 查看优化级管理规则。connection 连接模块，查看更多请点击[Connection methods and details](#).

复制和执行模块

一旦建立连接后，Ansible 会将命令或者 playbook 剧本需要的模块传输到远程主机后，在远程主机上执行命令。

行动：Ansible 初体验

使用 ping 模块测试主机存活

```
$ ansible all -m ping
```

在所有节点上执行一条实时命令:

```
$ ansible all -a "/bin/echo hello"
```

你运行命令的每台主机应该有类似如下的输出:

```
aserver.example.org | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python"
  },
  "changed": false,
  "ping": "pong"
}
```

基础进阶

Ansible 默认使用 sftp 传输文件。如果受控节点不支持 SFTP，你可以根据文档[配置 Ansible](#) 切换成 SCP 模式。这些文件会临时存放在指定目录下，并在该目录执行这些文件。

如果需要超级权限或者特殊权限，类似 `sudo`，使用 `become` 参数指定：

```
# as bruce
$ ansible all -m ping -u bruce
# as bruce, sudoing to root (sudo is default method)
$ ansible all -m ping -u bruce --become
# as bruce, sudoing to batman
$ ansible all -m ping -u bruce --become --become-user batman
```

更新请参考 in *Understanding privilege escalation: become*.

恭喜！你已经使用 Ansible 打通了所有的主机的奇经八脉。您使用了一个基本清单文件和一个 ad-hoc 临时命令来操作 Ansible 连接到特定的远程节点，并在这个过程中复制模块文件然后执行它，最后返回输出。您已经拥有一个可以正常运行的 Ansible 基础架构了。

接下来

接下来，你可以了解更多关于 ad-hoc 的使用[ad-hoc 命令操作指引](#)，探索更多其它模块，更多请参考 *Working With Playbooks*。Ansible 不仅能运行命令，还包括强大的配置管理和部署功能。

参见：

Inventory 使用进阶 inventory 详解

ad-hoc 命令操作指引 基础命令示范案例

Working With Playbooks 深入学习 Ansible 配置管理

Mailing List Questions? Help? Ideas? Stop by the list on Google Groups

irc.freenode.net #ansible IRC chat channel

1.3.4 Inventory 使用进阶

Ansible 从 Inventory 读取列表或组，可同时并发操作这些受控节点或主机。一旦 inventory 被定义，你就可以使用正则匹配主机或者组来指定要运行的主机列表 *patterns*。

Inventory 主机清单存放在 `/etc/ansible/hosts`。你可以在命令行使用 `-i <path>` 指定特定的 inventory 清单。当然，你可以一次指定多个 inventory 清单，也可以使用 pull inventory 的动态获取或者从云主机获取。具体参考[动态 Inventory 清单配置](#)。

Ansible 从 2.4 开始，使用 *Inventory Plugins* 的方式使 inventory 清单更灵活

- *Inventory 基础: formats, hosts, and groups*
 - 默认组 (Default groups)

- 多主机组
- 增加主机段
- *Inventory* 变量定义
- 给单台主机设置变量: *host variables*
 - *Inventory aliases*
- 给多台主机设置变量: *group variables*
- 编排主机和组变量
- 变量合并
- 使用多个 *Inventory*
- 主机连接: *Inventory* 参数设置
 - *Non-SSH* 连接类型
- *Inventory* 设置示例
 - 示例: 一个环境一个 *Inventory* 清单
 - 示例: 按功能分组
 - 示例: 按位置分组

Inventory 基础: formats, hosts, and groups

Inventory 文件可以有多种格式, 取决于你使用什么插件, 最常用的格式是 YAML 和 INI。下面是救命:

```
mail.example.com

[webservers]
foo.example.com
bar.example.com

[dbservers]
one.example.com
two.example.com
three.example.com
```

括号中的标题是组名, 用于对主机进行分类, 用于确定什么时间、什么目的、相对哪些主机做什么事情
如下为 YAML 格式的示例:

```
all:
  hosts:
    mail.example.com:
  children:
    webservers:
      hosts:
        foo.example.com:
        bar.example.com:
    dbservers:
      hosts:
        one.example.com:
        two.example.com:
        three.example.com:
```

默认组 (Default groups)

默认有两个分组：all and ungrouped 。all 组顾名思义包括所有主机。ungrouped 则是 all 组之外所有主机。所有的主机要不属于 all 组，要不就属于 ungrouped 组。

尽管 all 和 ungrouped 始终存在，但它们以隐式的方式出现，而不出现在诸如 group_names 的组列表中。

多主机组

你可以把一台主机放在多个组中。比如：亚特兰大 (Atlanta) 数据中心中的生产 (prod) 的 webserver 服务器可能包含在 [prod] 和 [atlanta] 和 [webservers] 组中。你可以参考如下思路创建组：

- What - 是什么？一个应用，栈，还是微服务. (例如: database servers, web servers, etc).
- Where - 在哪里？数据中心？某个地区？本地 DNS? 还是一个存储 (例如, east, west).
- When - 何时？开发阶段、生产阶段? (例如: prod, test).

扩展以前的 YAML 清单，加入如上示例中 (what, when, and where) 的示例：

```
all:
  hosts:
    mail.example.com:
  children:
    webservers:
      hosts:
        foo.example.com:
        bar.example.com:
    dbservers:
```

(下页继续)

(续上页)

```
hosts:
  one.example.com:
  two.example.com:
  three.example.com:
east:
  hosts:
    foo.example.com:
    one.example.com:
    two.example.com:
west:
  hosts:
    bar.example.com:
    three.example.com:
prod:
  hosts:
    foo.example.com:
    one.example.com:
    two.example.com:
test:
  hosts:
    bar.example.com:
    three.example.com:
```

可以看到 `one.example.com` 同时存在 `dbservers`, `east`, and `prod` 组中您还可以使用嵌套组来简化此清单中的 `prod` and `test` 组, 优化后结果如下:

```
all:
  hosts:
    mail.example.com:
  children:
    webservers:
      hosts:
        foo.example.com:
        bar.example.com:
    dbservers:
      hosts:
        one.example.com:
        two.example.com:
        three.example.com:
  east:
```

(下页继续)

(续上页)

```
hosts:
  foo.example.com:
  one.example.com:
  two.example.com:
west:
  hosts:
    bar.example.com:
    three.example.com:
prod:
  children:
    east:
test:
  children:
    west:
```

你可以找到更多编排 inventory 的安全 *Inventory* 设置示例。

增加主机段

如果您有许多具有相似模式的主机，则可以将它们添加为一个范围，而不必分别列出每个主机名：

In INI:

```
[webservers]
www[01:50].example.com
```

In YAML:

```
...
webservers:
  hosts:
    www[01:50].example.com:
```

对于数字匹配 [0-9]，也支持字母正则 [a-z]：

```
[databases]
db-[a:f].example.com
```

Inventory 变量定义

你可以直接在 Inventory 清单中定义的 host 或 group 变量。刚开始的时候，你可以直接添加 host 或 group 到 Inventory 文件中。当你越加越多的时候，你可能会考虑将变量和 host group 分离成独立的文件。

给单台主机设置变量: host variables

在 Playbook 中的示例 INI 文件的写法:

```
[atlanta]
host1 http_port=80 maxRequestsPerChild=808
host2 http_port=303 maxRequestsPerChild=909
```

YAML:

```
atlanta:
  host1:
    http_port: 80
    maxRequestsPerChild: 808
  host2:
    http_port: 303
    maxRequestsPerChild: 909
```

比如给 host 添加非标准 SSH 端口, 把端口直接添加到主机名后, 心冒号分隔即可:

```
badwolf.example.com:5309
```

connection 连接远程主机的方式:

```
[targets]

localhost          ansible_connection=local
other1.example.com  ansible_connection=ssh      ansible_user=myuser
other2.example.com  ansible_connection=ssh      ansible_user=myotheruser
```

注解: 如果你在 SSH 配置文件中定义了非标端口, 使用 `openssh` 连接主机可以默认读取不用指定特定端口, 但 `paramiko` 就不能自动发现了。

Inventory aliases

在 Inventory 中定义别名:

In INI 中定义:

```
jumper ansible_port=5555 ansible_host=192.0.2.50
```

在 YAML 中定义:


```
...
hosts:
  jumper:
    ansible_port: 5555
    ansible_host: 192.0.2.50
```

在如上的示例中，执行 Ansible 对 “jumper” 主机执行命令时，会连接 192.0.2.50 的 5555 端口。这种方式仅适用于通过静态 IP 的主机，或者通过隧道连接的主机。

注解：INI 格式定义的 `key=value` 声明的位置不同所代表的含意也不同：

- 当在主机同行声明时，INI 值将会被解释了 Python 内置结构，(strings, numbers, tuples, lists, dicts, booleans, None)。在同一行中主机可以接受多个 `key=value` 参数，因此，他们需要一种方法来指示空格是值的一部分而不是分隔符。
- 当声明 `:vars` 段落声明时，INI 的值会被解析为字符串。比如 `var=FALSE` 的 ‘FALSE’ 就是字符串。和 host lines 定义变量的方式不同，`:vars` 段落只接受一个条目，所以 = 后面所有的内容都是变量的值。
- 如果 INI Inventory 的变量定义必须是一个确切的类型 (a string or a boolean value)，请在 task 中使用过滤器指定类型，请勿依赖 INI Inventory 中定义的类型。
- 建议使用 YAML 格式定义 Inventory 清单，YAML 格式可以很好的避免变量混淆的情况。YAML 插件可以保证变量的一致性和正确性。

通常来讲，使用这种方式定义系统策略变量并不完美。在主配置文件 Inventory 中设置变量只是一种速记的临时方式。可以参考[编排主机和组变量](#) 如果在 ‘host_vars’ 目录中定义使用独立的变量文件设置变量。

给多台主机设置变量: group variables

如果组中的所有主机共享一个变量值，则可以一次将该变量应用于整个组，INI 格式：

```
[atlanta]
host1
host2

[atlanta:vars]
ntp_server=ntp.atlanta.example.com
proxy=proxy.atlanta.example.com
```

In YAML:

```
atlanta:
  hosts:
```

(下页继续)

(续上页)

```
host1:
host2:
vars:
ntp_server: ntp.atlanta.example.com
proxy: proxy.atlanta.example.com
```

组变量是一次将变量同时应用于多个主机的便捷方法。但是,在执行之前,Ansible 始终将变量(包括 Inventory 清单变量)展平到主机级别。如果该主机是多个组的成员,则 Ansible 将从所有这些组中读取变量值。如果同一主机在不同的组中被赋予不同的变量值,则 Ansible 会根据内部规则来选择要使用的值。具体合并值的方式请参考[rules for merging](#)。

继承变量值: 嵌套组的组变量设置

您可以设置一个 children 的组变量,:children INI 格式或 children: YAML 格式,您可以分别使用 :vars or vars: 给组定义变量

INI 格式:

```
[atlanta]
host1
host2

[raleigh]
host2
host3

[southeast:children]
atlanta
raleigh

[southeast:vars]
some_server=foo.southeast.example.com
halon_system_timeout=30
self_destruct_countdown=60
escape_pods=2

[usa:children]
southeast
northeast
southwest
northwest
```

YAML 格式:

```
all:
  children:
    usa:
      children:
        southeast:
          children:
            atlanta:
              hosts:
                host1:
                host2:
            raleigh:
              hosts:
                host2:
                host3:
          vars:
            some_server: foo.southeast.example.com
            halon_system_timeout: 30
            self_destruct_countdown: 60
            escape_pods: 2
        northeast:
        northwest:
        southwest:
```

如果您需要存储列表或哈希数据，或者更喜欢将主机和组特定变量与清单文件分开，请参考[编排主机和组变量](#) 子组有几个要注意的属性：

- 子组成员默认属于父组成员
- 子组的变量比父组的变量优先级高，即值会覆盖父组的变量。
- 组可以有多个父组或孩子，但不能循环关系。
- 主机也可以隶属于多个组中，但是只有一个主机实例，可以合并多个组中的数据。

编排主机和组变量

尽管你可以将变量存储在 Inventory 主清单文件中，但是将变量存储在单独的主机变量和组变量文件中，可以帮助您更轻松的组织变量值。主机和组变量文件必须使用 YAML 语法。文件可以以 '.yaml'，'.yml'，'.json' 结尾，甚至没有扩展结尾。更多请参考 [YAML Syntax](#)。

Ansible 通过搜索对应的 Inventory 清单文件或 Playbook 剧本文件来加载主机和组变量文件。如果你的清单文件 `/etc/ansible/hosts` 主机 'foosball' 同时属于 'raleigh' 和 'webservers' 组，该主机使用的变量可能包含在如下这些路径：

```
/etc/ansible/group_vars/raleigh # can optionally end in '.yaml', '.yml', or '.json'
/etc/ansible/group_vars/webserver
/etc/ansible/host_vars/foosball
```

举例，如果您按数据中心分组主机，并且每个数据中心使用其自己的 NTP 服务器和数据库服务器，则可以创建一个名为 `/etc/ansible/group_vars/raleigh` 的文件来存储 `raleigh` 的变量组：

```
---
ntp_server: acme.example.org
database_server: storage.example.org
```

您可以在 `hosts` 或者 `groups` 后创建 *directories*。Ansible 将按顺读取这些目录中的所有文件。举例 ‘raleigh’ 组：

```
/etc/ansible/group_vars/raleigh/db_settings
/etc/ansible/group_vars/raleigh/cluster_settings
```

‘raleigh’ 组中所有主机的变量在这些文件中都有效。这种方式非常适合组织变量比较多的场景或者原来的文件非常大，更多组变量请参考 [Ansible Vault](#)。

你也可以新建 `group_vars/` 和 `host_vars/` 目录到 `playbook` 同级目录下。`ansible-playbook` 会默认查找当前工作目录。其它类似 `ansible`, `ansible-console` 等命令默认从查找 `group_vars/` and `host_vars/` 目录。如果希望其他命令从 `playbook` 目录中加载组和主机变量，则必须在命令行上提供 `--playbook-dir` 选项。

如果您同时从 `Playbook` 目录和 `Inventory` 清单目录中加载清单文件，则 `Playbook` 目录中的变量将覆盖在清单目录中设置的变量。

将 `Inventory` 清单文件和变量保存在 `git repo`（或其他版本控制系统）中是跟踪库存和主机变量更改的绝佳方法。

变量合并

默认情况下，在运行之前变量会合并/展开到特定目标主机。这种方式使 Ansible 始终专注于主机和任务，因此组的定义是完全依赖 `Inventory` 和 `host`。默认，Ansible 会按序覆盖重复变量，无论是组变量还是主机变量。（请参考 `DEFAULT_HASH_BEHAVIOUR`）。优先顺序是（从最低到最高）：

- all group (because it is the ‘parent’ of all other groups)
- parent group
- child group
- host

默认，Ansible 按字母顺序合并处于相同父/子级别的组，后加载的组将覆盖先前的组。在下面的示例中，`a_group` 将与 `b_group` 组合并，并且匹配的 `b_group` 组覆盖 `a_group`。

您可以通过设置组变量 `ansible_group_priority` 来更改此行为，以更改同一级别的组的合并顺序（在解决父/子顺序之后）。数字越大，合并的时间越晚，优先级越高。如果未设置，则此变量默认为“1”。示例：

```
a_group:
    testvar: a
    ansible_group_priority: 10
b_group:
    testvar: b
```

在这个示例中，如果两个组的优先级相同，则最终的结果是 `testvar == b`，但是如果我们设置 `a_group` 的优先级为 10，则最终结果为 `testvar == a`

注解： `ansible_group_priority` 只在 Inventory 中有效，在 `group_vars/` 无效，因为该变量用于加载 `group_vars` 来设置变量变量优先级。

使用多个 Inventory

你可以命令行提供多个 Inventory 选项或者配置 `ANSIBLE_INVENTORY` 的方式，同时使用多个 Inventory 源（目录，动态 Inventory 脚本或者 Inventory 插件提供的文件）。该功能针对相互独立的环境非常有帮助，比如你想同时对测试环境和生产环境执行某操作。

同时使用两个源的命令执行方式如下：

```
ansible-playbook get_logs.yml -i staging -i production
```

注意，如果变量冲突，冲突解决方案请参考[变量合并](#) and *Variable precedence: Where should I put a variable?*。变量合并顺序由 Inventory 输入顺序决定。如果 `[all:vars]` 在 staging inventory 定义为 `myvar = 1`，但是在 production inventory 定义为 `myvar = 2`，最终的结果为 `myvar = 2`。同样，结果会反转如果改变参数输入顺序 `-i production -i staging`。

合并目录下的所有 Inventory

您还可以合并组合目录下的多个 Inventory 清单和不同类型的 Inventory 来创建新清单。这对于组合静态和动态主机并将它们作为一个 Inventory 清单进行管理很有用。以下 Inventory 清单结合了清单插件源，动态清单脚本，和带有静态主机的文件：

```
inventory/
  openstack.yml      # configure inventory plugin to get hosts from Openstack cloud
  dynamic-inventory.py # add additional hosts with dynamic inventory script
  static-inventory    # add static hosts and groups
  group_vars/
    all.yml           # assign variables to all hosts
```

您可以像下面这样指定一个 Inventory 清单目录:

```
ansible-playbook example.yml -i inventory
```

如果存在与其他库存来源之间的变量冲突或组依赖关系, 则控制库存来源的合并顺序可能很有用。合并根据文件名按字母顺序合并, 因此可以通过在文件前添加数字前缀来控制结果:

```
inventory/
  01-openstack.yml          # configure inventory plugin to get hosts from Openstack
↳ cloud
  02-dynamic-inventory.py    # add additional hosts with dynamic inventory script
  03-static-inventory        # add static hosts
group_vars/
  all.yml                   # assign variables to all hosts
```

如果 01-openstack.yml 在 all 中声明 myvar = 1 , 02-dynamic-inventory.py 声明 myvar = 2, 03-static-inventory 声明 myvar = 3, playbook 最终会返回结果 myvar = 3.

关于 Inventory 插件和动态 Inventory 脚本的更多信息请参考[Inventory Plugins](#) and [动态 Inventory 清单配置](#).

主机连接: Inventory 参数设置

综上所述, 设置以下变量可控制 Ansible 与远程主机的交互方式。

connection 模式:

注解: Ansible does not expose a channel to allow communication between the user and the ssh process to accept a password manually to decrypt an ssh key when using the ssh connection plugin (which is the default). The use of `ssh-agent` is highly recommended.

ansible__connection 连接受控主机的方式. 填写 ansible 连接的插件名字, 如 `smart`, `ssh` or `paramiko`. 默认: `smart`. 下一节将介绍基于非 SSH 的类型。

常用的连接参数有如下这些:

ansible__host 要连接的主机名, 如果与您要给它提供的别名不同。

ansible__port 连接端口, 默认 22

ansible__user 连接远程主机的用户

ansible__password 认证密码 (切勿明文保存在文本中, 使用加密的方式保存。详见[Keep vaulted variables safely visible](#))

SSH 连接选项:

ansible_ssh_private_key_file 指定 ssh 私钥。如果没有私钥或者有多个私钥时有用

ansible_ssh_common_args 这个设置通常添加在默认命令行 **sftp**, **scp** and **ssh** 之后。当为一台主机或组配置 ProxyCommand 时有用。

ansible_sftp_extra_args 此设置始终附加在默认的 *sftp* 命令行中。

ansible_scp_extra_args 此设置始终附加在默认的 *scp* 命令行中。

ansible_ssh_extra_args This setting is always appended to the default **ssh** command line. 此设置始终附加在默认的 *ssh* 命令行中。

ansible_ssh_pipelining 设置是否使用 SSH 管道，可以在 *ansible.cfg* 设置

ansible_ssh_executable (added in version 2.2) 此设置将覆盖默认行为以使用系统 **ssh**。这样会覆盖 *ansible.cfg* 文件中的 *ssh_executable* 设置

提权设置 (更多请参考 *Ansible Privilege Escalation*):

ansible_become 等同 *ansible_sudo* or *ansible_su*, 允许强制特权升级

ansible_become_method 允许设置权限提升方法

ansible_become_user 等同 *ansible_sudo_user* or *ansible_su_user*, 允许设置通过特权升级成为的用户

ansible_become_password 等同 *ansible_sudo_password* or *ansible_su_password*, 允许您设置特权升级密码。(切勿存储明文密码，请务必使用加密的方式。详见 *Keep vaulted variables safely visible*)

ansible_become_exe 等同 to *ansible_sudo_exe* or *ansible_su_exe*, 允许您为所选的升级方法设置可执行文件

ansible_become_flags 等同 *ansible_sudo_flags* or *ansible_su_flags*, 允许您设置传递给所选升级方法的标志。也可以在中 *ansible.cfg* 全局设置 *sudo_flags*

远程主机环境变量选项:

ansible_shell_type 指定远程主机使用的 Shell。在使用该选项前一定要先将 *ansible_shell_executable* 设置为 non-Bourne (sh) 。默认命令使用 **sh**。设置 **csh** or **fish** 将会在远程主机上使用 **csh** **fish**，而非默认的 **sh**

ansible_python_interpreter 目标主机 python 目录。对于一台主机上有多个 Python 环境或者默认路径不是 */usr/bin/python* 的 *BSD 环境，或者 where */usr/bin/python* 的不是 2.X 系统的 Python 情况有用。我们不使用:command:*/usr/bin/env* 命令机制，因为这需要设置远程用户的路径，并且假定 python 可执行文件名为 *python*，其中可执行文件可能命名为像 *python2.6* 一样的程序。

ansible_*_interpreter 适用于 ruby or perl 等类型 *ansible_python_interpreter* 环境。这将替换运行模块在远程主机上的 shabang.

ansible_shell_executable 设置远程主机使用何种 shell，默认 */bin/sh*，会覆盖 *executable* in *ansible.cfg*。如果远程主机没有安装 */bin/sh*，则需要修改下了。(比如：*/bin/sh* 在远程主机没有安装或者无法 *sudo* 运行)

Ansible-INI 示例:

```
some_host      ansible_port=2222      ansible_user=manager
aws_host       ansible_ssh_private_key_file=/home/example/.ssh/aws.pem
freebsd_host   ansible_python_interpreter=/usr/local/bin/python
ruby_module_host ansible_ruby_interpreter=/usr/bin/ruby.1.9.3
```

Non-SSH 连接类型

如上一节所述, Ansible 通过 SSH 执行剧本, 但不限于此连接类型。

通过设置 `ansible_connection=<connector>` 选项来修改连接类型。如下 non-SSH 选项可用。

local

部署应用到管理机自身。

docker

该连接器使用本地 Docker 客户端将 Playbook 直接部署到 Docker 容器中。此连接器处理以下参数:

ansible_host 要连接的 Docker 容器名字

ansible_user 在容器内操作的用户名, 用户必须存在于容器内。

ansible_become 如果设置为 `true`, 则 **become_user** 将用于在容器内进行操作。**ansible_docker_extra_args**

可以是一个字符串, 该字符串由 Docker 可识别的命令选项成, 不限定命令任何其他参数。这个选项主要用于配置远程主机 Docker daemon 服务。

如下为立即部署容器的示例:

```
- name: create jenkins container
  docker_container:
    docker_host: myserver.net:4243
    name: my_jenkins
    image: jenkins

- name: add container to inventory
  add_host:
    name: my_jenkins
    ansible_connection: docker
    ansible_docker_extra_args: "--tlsverify --tlscacert=/path/to/ca.pem --tlscert=/path/
    to/client-cert.pem --tlskey=/path/to/client-key.pem -H=tcp://myserver.net:4243"
    ansible_user: jenkins
    changed_when: false
```

(下页继续)

(续上页)

```
- name: create directory for ssh keys
  delegate_to: my_jenkins
  file:
    path: "/var/jenkins_home/.ssh/jupyter"
    state: directory
```

可用插件及更多案例请参考: *Plugin List*.

注解: 如果你是从头开始阅读文档, 这个案例应该是你看到的第一个 ansible playbook 案例。这不是一个 Inventory 文件。Playbooks 将在后面的章节更详细的介绍。

Inventory 设置示例

点击[这里](#)查看 playbooks 清单剧本和 Ansible 其它组件示例 *Sample Ansible setup*

示例: 一个环境一个 Inventory 清单

如果你需要管控多套环境, 明智的做法是每套环境对应一个独立的 Inventory 配置。这种方式很难有误操作。比如, 你很难实际要修改 “staging” 的服务器却修改了 “test” 环境的。

对于上述示例, 你可以使用 `inventory_test` file:

```
[dbservers]
db01.test.example.com
db02.test.example.com

[appservers]
app01.test.example.com
app02.test.example.com
app03.test.example.com
```

该文件仅包含属于 “test” 的主机环境。在另外一个文件定义 “staging” 环境的服务器 `inventory_staging`:

```
[dbservers]
db01.staging.example.com
db02.staging.example.com

[appservers]
app01.staging.example.com
```

(下页继续)

(续上页)

```
app02.staging.example.com
app03.staging.example.com
```

使用 `site.yml` 对 “test” 环境的所有服务器执行变更，使用如下命令：

```
ansible-playbook -i inventory_test site.yml -l appservers
```

示例：按功能分组

在上一节中，您已经看到具有相同功能的主机分组在一个群集。这种方式可以方便我们在不影响 DB 服务器的前提下，在 Playbook 或 Role 中定义防火墙规则：

```
- hosts: dbservers
  tasks:
  - name: allow access from 10.0.0.1
    iptables:
      chain: INPUT
      jump: ACCEPT
      source: 10.0.0.1
```

示例：按位置分组

有些任务可能更在意服务器所在的位置。如 `db01.test.example.com` 和 `app01.test.example.com` 在 DC1，而 `db02.test.example.com` 位于 DC2：

```
[dc1]
db01.test.example.com
app01.test.example.com

[dc2]
db02.test.example.com
```

在实践中，您甚至可能需要混合所有这些设置，因为有一天可能需要更新特定数据中心中的所有节点，而另一天则不管应用程序服务器位置在哪里，所有服务器都更新。

参见：

Inventory Plugins 从动态或静态源中拉取 Inventory

动态 Inventory 清单配置 从动态源中拉取 Inventory，例如云厂商

ad-hoc 命令操作指引 基础命令示例

Working With Playbooks 学习 Ansible 的配置、部署和语法编排

Mailing List Questions? Help? Ideas? Stop by the list on Google Groups

irc.freenode.net #ansible IRC chat channel

1.3.5 动态 Inventory 清单配置

- *Inventory* 脚本示例: *Cobbler*
- *Inventory* 清单脚本示例: *AWS EC2*
- *Inventory* 脚本示例: *OpenStack*
 - *OpenStack inventory* 脚本使用
 - 隐式使用 *OpenStack Inventory* 脚本
 - 刷新缓存
- 其它 *inventory scripts*
- *Inventory* 目录和多个 *Inventory* 源的使用
- 动态组中的静态组

如果你的 Ansible Inventory 清单会随需求变化,主机会随业务需求更换或关闭,上面的这些需求,静态 Inventory `:ref:'inventory'` 则无法满足你的需求了。你可能需要同时使用多个来源的主机清单:云厂商,LDAP, **Cobbler** <<https://cobbler.github.io>>‘_, 或者企业 CMDB 系统。

Inventory 插件利用了 Ansible 最新的核心代码。我们建议使用插件而非脚本动态获取 Inventory 清单。你可以参考 *write your own plugin* 连接动态 Inventory 源。

当然了,如果你坚持的话依然可以使用 Inventory 脚本的方式调用。但是我们 Inventory 插件的实现确保了向后兼容性,更加灵活健壮。如下是如何使用 Inventory 脚本的示例。

如果你需要一个 GUI 页面来处理动态 Inventory, *Red Hat Ansible Tower* 可以同步你所有的动态 Inventory 源,并提供 web 接口和 REST api 查看结果,同时提供图形化界面的编辑器。tower 会把所有的 hosts 存储到数据库,你可以关联历史事件,你可以查看哪些主机运行了什么命令,遇到过什么故障。

Inventory 脚本示例: Cobbler

Ansible 和 Cobbler **Cobbler** 无缝衔接, Cobbler 是一套 Linux 服务器部署安装的自动化工具,创始人是 Michael DeHaan,现在由 James Cammarata 主导,而这两个人现在都在 Ansible 工作。

Cobbler 虽然主要用于启动安装 OS,并管理 DHCP 和 DNS,但 Cobbler 具有通用层,可以表示多个(甚至同时)配置管理系统的数据库,并充当“轻量级 CMDB”。

将 Ansible 的 inventory 清单关联至 Cobbler, 复制 [this script](#) 到 `/etc/ansible` 并且添加可执行权限 `chmod +x` . 每当使用 `-i` 选项 (e.g. `-i /etc/ansible/cobbler.py`) 运行“cobblerd”时, 都会都可以使用 Cobbler 的 XMLRPC API 与 Cobbler 通信。

为了让 Ansible 知道 Cobbler 服务器在哪里, 并且可以使用缓存功能提升性能, 需要添加 `cobbler.ini` 到 `/etc/ansible` 目录。

```
[cobbler]

# Set Cobbler's hostname or IP address
host = http://127.0.0.1/cobbler_api

# API calls to Cobbler can be slow. For this reason, we cache the results of an API
# call. Set this to the path you want cache files to be written to. Two files
# will be written to this directory:
#   - ansible-cobbler.cache
#   - ansible-cobbler.index

cache_path = /tmp

# The number of seconds a cache file is considered valid. After this many
# seconds, a new API call will be made, and the cache file will be updated.

cache_max_age = 900
```

首先直接运行 `/etc/ansible/cobbler.py` 脚本, 正常情况下你可以看到 JSON 格式的数据输出, 但也有可能因为还没有配置好而没有任何输出。

让我们探索下他的使用场景. Cobbler 中, 假设有一个类似如下的使用场景:

```
cobbler profile add --name=webserver --distro=CentOS6-x86_64
cobbler profile edit --name=webserver --mgmt-classes="webserver" --ksmeta="a=2 b=3"
cobbler system edit --name=foo --dns-name="foo.example.com" --mgmt-classes="atlanta" --
↪ksmeta="c=4"
cobbler system edit --name=bar --dns-name="bar.example.com" --mgmt-classes="atlanta" --
↪ksmeta="c=5"
```

在上面的示例中, ‘foo.example.com’ 主机即可以直接在 Ansible 中找到, 但在使用组 ‘webserver’ or ‘atlanta’ 时也可以被找到。由于 Ansible 使用 SSH, 因此他只能通过 ‘foo.example.com’ 找到对应关系, ‘foo’ 是不行的。类似的, “ansible foo” Ansible 也是找不到对应主机的。但是 “ansible ‘foo*’” 是可以的, 因为系统 DNS 的名字是以 ‘foo’ 开始的。

该脚本不仅提供主机和组信息。此外, 作为奖励, 当运行 “setup” 模块 (使用 playbook 剧本时会自动发生) 时, 变量 “a”, “b” 和 “c” 都将自动填充到模板中

```
# file: /srv/motd.j2
Welcome, I am templated with a value of a={{ a }}, b={{ b }}, and c={{ c }}
```

可以像这样执行:

```
ansible webserver -m setup
ansible webserver -m template -a "src=/tmp/motd.j2 dest=/etc/motd"
```

注解: webserver 和配置文件的变量来自 Cobbler，您仍可以像在 Ansible 中一样传递自己的变量，但是外部 Inventory 清单脚本中的变量将覆盖具有相同名称的任何变量。

因此，使用上面的模板 (motd.j2)，这将修改 ‘foo’ 系统的 /etc/motd:

```
Welcome, I am templated with a value of a=2, b=3, and c=4
```

And on system ‘bar’ (bar.example.com):

```
Welcome, I am templated with a value of a=2, b=3, and c=5
```

从技术上讲，尽管没有充分的理由这样做，但这也适用:

```
ansible webserver -m shell -a "echo {{ a }}"
```

换句话说讲，你可以在参数/命令使用这些变量。

Inventory 清单脚本示例: AWS EC2

如果您使用 Amazon Web Services EC2，则维护一份静态 Inventory 清单文件可能不是最佳方法，因为主机可能会随着时间的流逝而移动，或者由外部应用程序进行管理，甚至您可能正在使用 AWS 自动缩放。这些情况建议你使用 [EC2 external inventory](#)。

您可以通过如下两种方式之一使用此脚本。最简单的方法是使用 Ansible 的 `-i` 命令行选项，并在将脚本标记为可执行文件后指定脚本的路径:

```
ansible -i ec2.py -u ubuntu us-east-1d -m ping
```

第二种方法是复制脚本到 `/etc/ansible/hosts` 并 `chmod +x`。同时，你需要复制 `ec2.ini` file to `/etc/ansible/ec2.ini`。然后，您可以像往常一样运行 Ansible。

要成功的调用 AWS API，您必须配置 Boto (AWS 的 Python 接口)。您可以通过这里查看 [several ways](#)，但是最简单的方法是导出两个环境变量:

```
export AWS_ACCESS_KEY_ID='AK123'
export AWS_SECRET_ACCESS_KEY='abc123'
```

你运行脚本可以自测配置是否成功:

```
cd contrib/inventory
./ec2.py --list
```

片刻后, 你会看到 JSON 格式输出的所有区域的 EC2 Inventory 清单。

如果你使用 Boto 配置文件管理多个 AWS 账户, 则可以传递 `--profile PROFILE` 参数给 `ec2.py` 脚本。示例配置文件可能是:

```
[profile dev]
aws_access_key_id = <dev access key>
aws_secret_access_key = <dev secret key>

[profile prod]
aws_access_key_id = <prod access key>
aws_secret_access_key = <prod secret key>
```

运行 `ec2.py --profile prod` 获取 `prod` 账户的 Inventory, 但是 `ansible-playbook` 不支持该选项。你同样也可以使用 `AWS_PROFILE` 变量, 比如 `AWS_PROFILE=prod ansible-playbook -i ec2.py myplaybook.yml`

由于每个区域都需要自己独立的 API 调用, 因此, 如果你只希望调用部分区域, 则可以编辑 `ec2.ini` 文件, 注释掉不需要的区域。

`ec2.ini` 中还有其他配置选项, 包括缓存控制和目标变量。默认情况下, `ec2.ini` 的配置是为“所有 Amazon 云服务”生效。但是您可以注释掉所有不适用的功能。

```
[ec2]
...

# To exclude RDS instances from the inventory, uncomment and set to False.
rds = False

# To exclude ElastiCache instances from the inventory, uncomment and set to False.
elasticache = False
...
```

从本质上讲, Inventory 清单文件只是从名称到目标的映射。默认的 `ec2.ini` 设置为从外部配置 EC2 (例如从笔记本电脑) 运行 Ansible, 显然这不是管理 EC2 的最有效方法。

如果你从 EC2, 内部 DNS 和 IP 地址运行 Ansible, 这种方式比使用公共 DNS 更高效。这种情况下, 你可以修改实例的 `ec2.ini` 中的 `destination_variable` 为私有 DNS 名称。这种方式在私有网段的 VPC 中运

行 Ansible 的场景下非常重要，这种方式也是唯一的办法访问私有 IP 地址的实例。VPN 实例，`ec2.ini` 的配置项 `vpc_destination_variable` 提供了

如果从 EC2 内部运行 Ansible，则内部 DNS 名称和 IP 地址可能比公共 DNS 名称更有意义。在这种情况下，你可以将 `ec2.ini` 中的 `destination_variable` 为修改实例的私有 DNS 名称。在 VPC 内的私有子网中运行 Ansible 时，这尤其重要，在该子网中，访问实例的唯一方法是通过其私有 IP 地址。对于 VPC 实例，对于 VPC 实例，`ec2.ini` 中的“`vpc_destination_variable`”如何选择最合适您用例的案例 [boto.ec2.instance variable](#)。

EC2 外部 Inventory 清单提供了从多个组到实例的映射：

Global `ec2` 组中的所有实例

Instance ID These are groups of one since instance IDs are unique.

e.g. `i-00112233 i-a1b1c1d1`

Region AWS 一个区域中的所有实例 e.g. `us-east-1 us-west-2`

Availability Zone 一组可用性区域中所有实例

e.g. `us-east-1a us-east-1b`

Security Group

实例可以属于一个或多个安全组。为每个安全组创建一个组，组名由字母和数字组成，除字母和数字外的所有字符都转换为下划线 (`_`)。所有的案例组都以 `security_group_` 开头。当下，dashed (`-`) 也使用下划线 (`_`) 表示。您可以使用 `ec2.ini` 中的 `replace_dash_in_groups` 设置进行更改（此设置在多个版本中已更改，因此请检查“`ec2.ini`”以获取详细信息）。

e.g. `security_group_default security_group_webserver`
`security_group_Pete_s_Fancy_Group`

Tags 每个实例可以具有与其关联的各种键/值对，称为“标签”。虽然任何字符串都可以用来表示键值，但最常见的标准名是‘Name’，。每个键值对都是其实例组的名字，但是要将特殊字符转换为下划线，格式如 `tag_KEY_VALUE` e.g. `tag_Name_Web can be used as is tag_Name_redis-master-001 becomes tag_Name_redis_master_001 tag_aws_cloudformation_logical-id_WebServerGroup becomes tag_aws_cloudformation_logical_id_WebServerGroup`

当 Ansible 和特定服务器交互时，EC2 Inventory 脚本将再次调用 `--host HOST` 选项。这将在索引缓存中查找 HOST 以获取实例 ID，然后对 AWS 进行 API 调用以获取该特定实例的信息。然后，它将有关该实例的信息作为变量提供给您的 Playbook。每个变量都以“`ec2_`”为前缀。

- `ec2_architecture`
- `ec2_description`
- `ec2_dns_name`
- `ec2_id`
- `ec2_image_id`

- ec2_instance_type
- ec2_ip_address
- ec2_kernel
- ec2_key_name
- ec2_launch_time
- ec2_monitored
- ec2_ownerId
- ec2_placement
- ec2_platform
- ec2_previous_state
- ec2_private_dns_name
- ec2_private_ip_address
- ec2_public_dns_name
- ec2_ramdisk
- ec2_region
- ec2_root_device_name
- ec2_root_device_type
- ec2_security_group_ids
- ec2_security_group_names
- ec2_spot_instance_request_id
- ec2_state
- ec2_state_code
- ec2_state_reason
- ec2_status
- ec2_subnet_id
- ec2_tag_Name
- ec2_tenancy
- ec2_virtualization_type
- ec2_vpc_id

`ec2_security_group_ids` 和 `ec2_security_group_names` 都是逗号分隔的所有安全组。每个 EC2 的 tag 格式如 `ec2_tag_KEY`。

查看实例支持的完整变量列表，执行如下脚本：

```
cd contrib/inventory
./ec2.py --host ec2-12-12-12-12.compute-1.amazonaws.com
```

需要注意的 AWS Inventory 脚本会缓存 API 的调用结果，缓存配置可以在 `ec2.ini` 修改。如果想清除缓存，带上 `--refresh-cache` 参数：

```
./ec2.py --refresh-cache
```

Inventory 脚本示例: OpenStack

如果您使用 OpenStack 云，则无需手动维护 Inventory 清单文件，而可以使用 `openstack_inventory.py` 动态清单直接从 OpenStack 中获取有关您的计算实例的信息。

从这里下载最新的 OpenStack Inventory 脚本 [here](#)。

你可以显式（传参 `-i openstack_inventory.py`）或隐式（`/etc/ansible/hosts`）的使用 Inventory 脚本。

OpenStack inventory 脚本使用

下载最新的 OpenStack 动态 Inventory 脚本并赋予可执行权限：

```
wget https://raw.githubusercontent.com/ansible/ansible/devel/contrib/inventory/openstack_
↪inventory.py
chmod +x openstack_inventory.py
```

注解： 不修改名字为 `openstack.py`。这个名字会和 `openstacksdk` 导入冲突。

加载 OpenStack RC file:

```
source openstack.rc
```

注解： OpenStack RC 文件包含客户端工具与云厂商建立连接所需要的环境变量，例如身份验证 URL，用户名，密码和区域名。如何下载，创建或加载 OpenStack RC 文件，更多请参考 [Set environment variables using the OpenStack RC file](#)。

您可以通过运行一个简单的命令（`nova list`）并确保其不返回错误来确认文件已成功获得源文件。

注解: OpenStack 命令行客户端需要运行 `nova list` 命令。关于如何安装，更多信息请参考 [Install the OpenStack command-line clients](#)

使用如下命令测试动态 Inventory 脚本是否正常运行:

```
./openstack_inventory.py --list
```

片刻后，你会得到关于实例的 JSON 格式信息。

确认动态清单脚本按预期工作后，可以告诉 Ansible 将 `openstack_inventory.py` 脚本用作 Inventory 清单文件，如下所示：

```
ansible -i openstack_inventory.py all -m ping
```

隐式使用 OpenStack Inventory 脚本

下载最新的 OpenStack 动态 Inventory 脚本，赋予可执行权限，复制为 `/etc/ansible/hosts`:

```
wget https://raw.githubusercontent.com/ansible/ansible/devel/contrib/inventory/openstack_
↪inventory.py
chmod +x openstack_inventory.py
sudo cp openstack_inventory.py /etc/ansible/hosts
```

下载配置模板文件，按需修改并且复制为 `/etc/ansible/openstack.yml`:

```
wget https://raw.githubusercontent.com/ansible/ansible/devel/contrib/inventory/openstack.
↪yml
vi openstack.yml
sudo cp openstack.yml /etc/ansible/
```

使用如下命令测试 OpenStack 动态 Inventory 脚本是否正常工作:

```
/etc/ansible/hosts --list
```

片刻后，你会得到关于实例的 JSON 格式信息。

刷新缓存

需要注意的是，OpenStack 动态 Inventory 脚本会缓存 API 的重复调用。在执行 `openstack_inventory.py` 使用 `--refresh` 参数清除缓存:

```
./openstack_inventory.py --refresh --list
```

其它 inventory scripts

所有的脚本这里找 [contrib/inventory directory](#). 所有的 Inventory 脚本的通用用法都差不多, 你可以在这里查看 *write your own inventory script*.

Inventory 目录和多个 Inventory 源的使用

如果 Ansible 运行时 `-i` 指定给的位置是目录 (或在 `ansible.cfg` 中配置), 则 Ansible 可以同时使用多个清单资源。这样做时, 可以在的运行中同时混合使用动态和静态管理的 Inventory 资源。即时混合云!

在 Inventory 目录中, 可执行文件会被视为动态 Inventory 源, 其它文件绝大部分会被视为静态 Inventory 源。以如下后缀结尾的文件将被忽略:

```
~, .orig, .bak, .ini, .cfg, .retry, .pyc, .pyo
```

您可以通过在 `ansible.cfg` 中配置 `inventory_ignore_extensions` 列表, 或设置环境变量 `ANSIBLE_INVENTORY_IGNORE` 制定自己希望忽略的后缀。不论哪种情况, 该值都应是逗号分隔, 如上所示。

库存目录中的任何 `group_vars` 和 `host_vars` 子目录都将按预期方式进行解析, 从而使 Inventory 目录成为组织不同配置集的有力方法。更多请参考[使用多个 Inventory](#)。

动态组中的静态组

在静态 Inventory 文件中定义嵌套组时, 子组必须被事先定义, 不然 Ansible 会抛错。如果要定义动态子组的静态组, 请预先在静态 Inventory 文件中定义动态组为空。

```
[tag_Name_staging_foo]

[tag_Name_staging_bar]

[staging:children]
tag_Name_staging_foo
tag_Name_staging_bar
```

参见:

[Inventory 使用进阶](#) 静态 Inventory 文件介绍

[Mailing List](#) Questions? Help? Ideas? Stop by the list on Google Groups

[irc.freenode.net](#) #ansible IRC chat channel

1.3.6 Pattern: 正则匹配主机和组

当你执行 ad-hoc 临时命令或 playbook 时，你必须指定要被变更的节点或组。Pattern 可以让你更方便的指定特定主机或组。可以单独指定一台主机，一个 IP 地址，一个 Inventory 清单组，一个子网的组或者你 Inventory 中的所有主机。Pattern 具有高度的灵活性- 您可以排除或要求主机的子集，使用通配符或正则表达式等等。Ansible 在模式中包含的所有清单主机上执行。

- 使用 *patterns* 模式
- 通过 *patterns*
- *patterns* 局限性
- 高级 *Pattern* 选项
 - 在 *patterns* 使用变量
 - 在 *patterns* 使用组位置参数
 - 在 *patterns* 中使用正则
- *Patterns and ansible-playbook* 标志

使用 *patterns* 模式

执行临时命令或剧本的任何时候都可以使用模式。该模式是 *ad-hoc command* 中唯一没有标志的参数。它通常是第二个参数:

```
ansible <pattern> -m <module_name> -a "<module options>"
```

举例:

```
ansible webservers -m service -a "name=httpd state=restarted"
```

在 Playbook 中，pattern 是 `hosts:` 的值。

```
- name: <play_name>
  hosts: <pattern>
```

举例:

```
- name: restart webservers
  hosts: webservers
```

由于您通常想一次对多个主机运行命令或剧本，因此模式通常指代清单组由于你通常希望针对多台主机一次性运行一个命令或 playbook，patterns 通常关联是一组主机列表。ad-hoc 命令和 playbook 将对 `webservers`

中的所有主机做变更。

通过 patterns

下表展示了 patterns 模式的用法。

Description	Pattern(s)	Targets
All hosts	all (or *)	
One host	host1	
Multiple hosts	host1:host2 (or host1,host2)	
One group	webservers	
Multiple groups	webservers:dbservers	all hosts in webservers plus all hosts in dbservers
Excluding groups	webservers:!atlanta	all hosts in webservers except those in atlanta
Intersection of groups	webservers:&staging	any hosts in webservers that are also in staging

注解： 你可以使用 (,) 或 (:) 分隔多个主机。推荐使用逗号，因为在 IPv6 的场景下首先逗号

当你掌握基础 pattern 后，你就可以高级进阶混合使用他们了。示例：

```
webservers:dbservers:&staging:!phoenix
```

该 pattern 表示 ‘webservers’ and ‘dbservers’ 组的所有主机，和 ‘staging’ 组中的交集的所有主机，并且这些主机不在 ‘phoenix’ 组中。

你也可以使用正则表示的 FQDN 主机名或 IP 地址：

```
192.0.* *.example.com *.com
```

你也可以同时混合使用正则和 patterns：

```
one*.com:dbservers
```

patterns 局限性

Patterns 依赖于 Inventory。如果 host 或 group 不在 Inventory 清单仓库，则不能使用 Pattern。如果你使用的 Patterns IP 或主机名不存在。会引发如下报错：

```
[WARNING]: No inventory was parsed, only implicit localhost is available
[WARNING]: Could not match supplied host pattern, ignoring: *.not_in_inventory.com
```

Pattern 必须遵循 Inventory 语法规则。如果你定义了一台 host *alias*：

```
atlanta:
  host1:
    http_port: 80
    maxRequestsPerChild: 808
    host: 127.0.0.2
```

你必须在 Pattern 使用别名。在如上的案例中，你必须在 pattern 中使用 `host1`。如果你使用 IP 地址，会报错：

```
[WARNING]: Could not match supplied host pattern, ignoring: 127.0.0.2
```

高级 Pattern 选项

如上所述的通用 patterns 已经满足你的绝大部分需求，但 Ansible 也提供了几种其它的方式来定义你的目标主机和组。

在 patterns 使用变量

ansible-playbook 通过 `-e` 选项可以接受变量，在 pattern 中可以使用 `{{ 变量 }}`：

```
webservers: !{{ excluded }}:&{{ required }}
```

在 patterns 使用组位置参数

您可以根据主机在主机组中的位置定义主机或主机子集。如下示例：

```
[webservers]
cobweb
webbing
weber
```

你可以使用下标切割选择需要的主机或主机组：

```
webservers[0]      # == cobweb
webservers[-1]     # == weber
webservers[0:2]    # == webservers[0],webservers[1]
                  # == cobweb,webbing
webservers[1:]     # == webbing,weber
webservers[:3]     # == cobweb,webbing,weber
```

在 patterns 中使用正则

以 ~ 开始的模式将会被认定为正则表达式:

```
~(web|db).*\.example\.com
```

Patterns and ansible-playbook 标志

命令行的优先级比 playbook 高，通过指定命令行选项可以覆盖 playbook 中的定义。举例：你在 playbook 中定义 `hosts: all`，但在命令行中指定 `-i 127.0.0.2`，命令行会覆盖 playbook 中的定义。这种方式甚至在 Inventory 中没有定义目标主机都可行。同样，你可以使用 `--limit` 标识指定特定的目标主机：

```
ansible-playbook site.yml --limit datacenter2
```

最后，你可以使用 `--limit` 从一个文件中读取主机列表，使用时在文件名前加 `@`：

```
ansible-playbook site.yml --limit @retry_hosts.txt
```

如果 `RETRY_FILES_ENABLED` 设置 `True`，当 `ansible-playbook` 执行失败的主机记录到以 `retry` 结尾的文件中。这个文件在每次 `ansible-playbook` 执行结束后都会被覆盖。

```
ansible-playbook site.yml --limit @site.retry
```

更多关于 ad-hoc 和 playbook 的 pattern 信息请参考[ad-hoc 命令操作指引](#) and [Intro to Playbooks](#).

参见：

[ad-hoc 命令操作指引](#) 基础命令示例

[Working With Playbooks](#) 学习 Ansible 配置管理语言

[Mailing List](#) Questions? Help? Ideas? Stop by the list on Google Groups

[irc.freenode.net](#) #ansible IRC chat channel

1.3.7 ad-hoc 命令操作指引

An Ansible ad-hoc command uses the `/usr/bin/ansible` command-line tool to automate a single task on one or more managed nodes. Ad-hoc commands are quick and easy, but they are not reusable. So why learn about ad-hoc commands first? Ad-hoc commands demonstrate the simplicity and power of Ansible. The concepts you learn here will port over directly to the playbook language. Before reading and executing these examples, please read [Inventory 使用进阶](#).

- [Why use ad-hoc commands?](#)

- *Use cases for ad-hoc tasks*
 - *Rebooting servers*
 - *Managing files*
 - *Managing packages*
 - *Managing users and groups*
 - *Managing services*
 - *Gathering facts*

Why use ad-hoc commands?

Ad-hoc commands are great for tasks you repeat rarely. For example, if you want to power off all the machines in your lab for Christmas vacation, you could execute a quick one-liner in Ansible without writing a playbook. An ad-hoc command looks like this:

```
$ ansible [pattern] -m [module] -a "[module options]"
```

You can learn more about *patterns* and *modules* on other pages.

Use cases for ad-hoc tasks

Ad-hoc tasks can be used to reboot servers, copy files, manage packages and users, and much more. You can use any Ansible module in an ad-hoc task. Ad-hoc tasks, like playbooks, use a declarative model, calculating and executing the actions required to reach a specified final state. They achieve a form of idempotence by checking the current state before they begin and doing nothing unless the current state is different from the specified final state.

Rebooting servers

The default module for the **ansible** command-line utility is the **command** module. You can use an ad-hoc task to call the **command** module and reboot all web servers in Atlanta, 10 at a time. Before Ansible can do this, you must have all servers in Atlanta listed in a group called `[atlanta]` in your inventory, and you must have working SSH credentials for each machine in that group. To reboot all the servers in the `[atlanta]` group:

```
$ ansible atlanta -a "/sbin/reboot"
```

By default Ansible uses only 5 simultaneous processes. If you have more hosts than the value set for the fork count, Ansible will talk to them, but it will take a little longer. To reboot the `[atlanta]` servers with 10 parallel forks:


```
$ ansible atlanta -a "/sbin/reboot" -f 10
```

/usr/bin/ansible will default to running from your user account. To connect as a different user:

```
$ ansible atlanta -a "/sbin/reboot" -f 10 -u username
```

Rebooting probably requires privilege escalation. You can connect to the server as `username` and run the command as the `root` user by using the *become* keyword:

```
$ ansible atlanta -a "/sbin/reboot" -f 10 -u username --become [--ask-become-pass]
```

If you add `--ask-become-pass` or `-K`, Ansible prompts you for the password to use for privilege escalation (sudo/su/pfexec/doas/etc).

注解: The command module does not support extended shell syntax like piping and redirects (although shell variables will always work). If your command requires shell-specific syntax, use the *shell* module instead. Read more about the differences on the *Working With Modules* page.

So far all our examples have used the default ‘command’ module. To use a different module, pass `-m` for module name. For example, to use the shell module:

```
$ ansible raleigh -m shell -a 'echo $TERM'
```

When running any command with the Ansible *ad hoc* CLI (as opposed to *Playbooks*), pay particular attention to shell quoting rules, so the local shell retains the variable and passes it to Ansible. For example, using double rather than single quotes in the above example would evaluate the variable on the box you were on.

Managing files

An ad-hoc task can harness the power of Ansible and SCP to transfer many files to multiple machines in parallel. To transfer a file directly to all servers in the [atlanta] group:

```
$ ansible atlanta -m copy -a "src=/etc/hosts dest=/tmp/hosts"
```

If you plan to repeat a task like this, use the template module in a playbook.

The file module allows changing ownership and permissions on files. These same options can be passed directly to the copy module as well:

```
$ ansible webserver -m file -a "dest=/srv/foo/a.txt mode=600"
$ ansible webserver -m file -a "dest=/srv/foo/b.txt mode=600 owner=mdehaan group=mdehaan"
↪ "
```

The file module can also create directories, similar to `mkdir -p`:

```
$ ansible webservers -m file -a "dest=/path/to/c mode=755 owner=mdehaan group=mdehaan ↵  
↪state=directory"
```

As well as delete directories (recursively) and delete files:

```
$ ansible webservers -m file -a "dest=/path/to/c state=absent"
```

Managing packages

You might also use an ad-hoc task to install, update, or remove packages on managed nodes using a package management module like yum. To ensure a package is installed without updating it:

```
$ ansible webservers -m yum -a "name=acme state=present"
```

To ensure a specific version of a package is installed:

```
$ ansible webservers -m yum -a "name=acme-1.5 state=present"
```

To ensure a package is at the latest version:

```
$ ansible webservers -m yum -a "name=acme state=latest"
```

To ensure a package is not installed:

```
$ ansible webservers -m yum -a "name=acme state=absent"
```

Ansible has modules for managing packages under many platforms. If there is no module for your package manager, you can install packages using the command module or create a module for your package manager.

Managing users and groups

You can create, manage, and remove user accounts on your managed nodes with ad-hoc tasks:

```
$ ansible all -m user -a "name=foo password=<crypted password here>"  
  
$ ansible all -m user -a "name=foo state=absent"
```

See the user module documentation for details on all of the available options, including how to manipulate groups and group membership.

Managing services

Ensure a service is started on all web servers:

```
$ ansible web servers -m service -a "name=httpd state=started"
```

Alternatively, restart a service on all web servers:

```
$ ansible web servers -m service -a "name=httpd state=restarted"
```

Ensure a service is stopped:

```
$ ansible web servers -m service -a "name=httpd state=stopped"
```

Gathering facts

Facts represent discovered variables about a system. You can use facts to implement conditional execution of tasks but also just to get ad-hoc information about your systems. To see all facts:

```
$ ansible all -m setup
```

You can also filter this output to display only certain facts, see the setup module documentation for details.

Now that you understand the basic elements of Ansible execution, you are ready to learn to automate repetitive tasks using *Ansible Playbooks*.

参见:

配置 *Ansible* All about the Ansible config file

all_modules A list of available modules

Working With Playbooks Using Ansible for configuration management & deployment

Mailing List Questions? Help? Ideas? Stop by the list on Google Groups

irc.freenode.net #ansible IRC chat channel

1.3.8 Connection methods and details

This section shows you how to expand and refine the connection methods Ansible uses for your inventory.

ControlPersist and paramiko

By default, Ansible uses native OpenSSH, because it supports ControlPersist (a performance feature), Kerberos, and options in ~/.ssh/config such as Jump Host setup. If your control machine uses an older

version of OpenSSH that does not support ControlPersist, Ansible will fallback to a Python implementation of OpenSSH called ‘paramiko’ .

Setting a remote user

By default, Ansible connects to all remote devices with the user name you are using on the control node. If that user name does not exist on a remote device, you can set a different user name for the connection. If you just need to do some tasks as a different user, look at *Understanding privilege escalation: become*. You can set the connection user in a playbook:

```
---
- name: update webserver
  hosts: webservers
  remote_user: admin

  tasks:
  - name: thing to do first in this playbook
    . . .
```

as a host variable in inventory:

other1.example.com	ansible_connection=ssh	ansible_user=myuser
other2.example.com	ansible_connection=ssh	ansible_user=myotheruser

or as a group variable in inventory:

```
cloud:
  hosts:
    cloud1: my_backup.cloud.com
    cloud2: my_backup2.cloud.com
  vars:
    ansible_user: admin
```

Setting up SSH keys

By default, Ansible assumes you are using SSH keys to connect to remote machines. SSH keys are encouraged, but you can use password authentication if needed with the `--ask-pass` option. If you need to provide a password for *privilege escalation* (sudo, pbrun, etc.), use `--ask-become-pass`.

注解: Ansible does not expose a channel to allow communication between the user and the ssh process to accept a password manually to decrypt an ssh key when using the ssh connection plugin (which is the

default). The use of `ssh-agent` is highly recommended.

To set up SSH agent to avoid retyping passwords, you can do:

```
$ ssh-agent bash
$ ssh-add ~/.ssh/id_rsa
```

Depending on your setup, you may wish to use Ansible's `--private-key` command line option to specify a pem file instead. You can also add the private key file:

```
$ ssh-agent bash
$ ssh-add ~/.ssh/keypair.pem
```

Another way to add private key files without using `ssh-agent` is using `ansible_ssh_private_key_file` in an inventory file as explained here: [Inventory 使用进阶](#).

Running against localhost

You can run commands against the control node by using “localhost” or “127.0.0.1” for the server name:

```
$ ansible localhost -m ping -e 'ansible_python_interpreter="/usr/bin/env python"'
```

You can specify localhost explicitly by adding this to your inventory file:

```
localhost ansible_connection=local ansible_python_interpreter="/usr/bin/env python"
```

Managing host key checking

Ansible enables host key checking by default. Checking host keys guards against server spoofing and man-in-the-middle attacks, but it does require some maintenance.

If a host is reinstalled and has a different key in ‘known_hosts’, this will result in an error message until corrected. If a new host is not in ‘known_hosts’ your control node may prompt for confirmation of the key, which results in an interactive experience if using Ansible, from say, cron. You might not want this.

If you understand the implications and wish to disable this behavior, you can do so by editing `/etc/ansible/ansible.cfg` or `~/.ansible.cfg`:

```
[defaults]
host_key_checking = False
```

Alternatively this can be set by the `ANSIBLE_HOST_KEY_CHECKING` environment variable:

```
$ export ANSIBLE_HOST_KEY_CHECKING=False
```

Also note that host key checking in paramiko mode is reasonably slow, therefore switching to ‘ssh’ is also recommended when using this feature.

Other connection methods

Ansible can use a variety of connection methods beyond SSH. You can select any connection plugin, including managing things locally and managing chroot, lxc, and jail containers. A mode called ‘ansible-pull’ can also invert the system and have systems ‘phone home’ via scheduled git checkouts to pull configuration directives from a central repository.

1.3.9 Working with command line tools

Most users are familiar with *ansible* and *ansible-playbook*, but those are not the only utilities Ansible provides. Below is a complete list of Ansible utilities. Each page contains a description of the utility and a listing of supported parameters.

1.3.10 Working With Playbooks

Playbooks are Ansible’s configuration, deployment, and orchestration language. They can describe a policy you want your remote systems to enforce, or a set of steps in a general IT process.

If Ansible modules are the tools in your workshop, playbooks are your instruction manuals, and your inventory of hosts are your raw material.

At a basic level, playbooks can be used to manage configurations of and deployments to remote machines. At a more advanced level, they can sequence multi-tier rollouts involving rolling updates, and can delegate actions to other hosts, interacting with monitoring servers and load balancers along the way.

While there’s a lot of information here, there’s no need to learn everything at once. You can start small and pick up more features over time as you need them.

Playbooks are designed to be human-readable and are developed in a basic text language. There are multiple ways to organize playbooks and the files they include, and we’ll offer up some suggestions on that and making the most out of Ansible.

You should look at [Example Playbooks](#) while reading along with the playbook documentation. These illustrate best practices as well as how to put many of the various concepts together.

Intro to Playbooks

Ansible Playbooks offer a repeatable, re-usable, simple configuration management and multi-machine deployment system, one that is well suited to deploying complex applications. If you need to execute a task

with Ansible more than once, write a playbook and put it under source control. Then you can use the playbook to push out new configuration or confirm the configuration of remote systems. The playbooks in the [ansible-examples repository](#) illustrate many useful techniques. You may want to look at these in another tab as you read the documentation.

Playbooks can:

- declare configurations
- orchestrate steps of any manual ordered process, on multiple sets of machines, in a defined order
- launch tasks synchronously or *asynchronously*

- *Playbook syntax*
- *Playbook execution*
 - *Task execution*
 - *Desired state and ‘idempotency’*
 - *Running playbooks*
- *Handlers: running operations on change*
 - *Controlling when handlers run*
 - *Using variables with handlers*
- *Ansible-Pull*
- *Verifying playbooks*
 - *ansible-lint*

Playbook syntax

Playbooks are expressed in YAML format with a minimum of syntax. If you are not familiar with YAML, look at our overview of *YAML Syntax* and consider installing an add-on for your text editor (see *Other Tools And Programs*) to help you write clean YAML syntax in your playbooks.

A playbook is composed of one or more ‘plays’ in an ordered list. The terms ‘playbook’ and ‘play’ are sports analogies. Each play executes part of the overall goal of the playbook, running one or more tasks. Each task calls an Ansible module.

Playbook execution

A playbook runs in order from top to bottom. Within each play, tasks also run in order from top to bottom. Playbooks with multiple ‘plays’ can orchestrate multi-machine deployments, running one play on your

webservers, then another play on your database servers, then a third play on your network infrastructure, and so on. At a minimum, each play defines two things:

- the managed nodes to target, using a *pattern*
- at least one task to execute

In this example, the first play targets the web servers; the second play targets the database servers:

```
---
- name: update web servers
  hosts: webservers
  remote_user: root

  tasks:
  - name: ensure apache is at the latest version
    yum:
      name: httpd
      state: latest
  - name: write the apache config file
    template:
      src: /srv/httpd.j2
      dest: /etc/httpd.conf

- name: update db servers
  hosts: databases
  remote_user: root

  tasks:
  - name: ensure postgresql is at the latest version
    yum:
      name: postgresql
      state: latest
  - name: ensure that postgresql is started
    service:
      name: postgresql
      state: started
```

Your playbook can include more than just a hosts line and tasks. For example, the playbook above sets a `remote_user` for each play. This is the user account for the SSH connection. You can add other `playbook_keywords` at the playbook, play, or task level to influence how Ansible behaves. Playbook keywords can control the *connection plugin*, whether to use *privilege escalation*, how to handle errors, and more. To support a variety of environments, Ansible lets you set many of these parameters as command-line flags, in your Ansible configuration, or in your inventory. Learning the *precedence rules* for these sources of data will

help you as you expand your Ansible ecosystem.

Task execution

By default, Ansible executes each task in order, one at a time, against all machines matched by the host pattern. Each task executes a module with specific arguments. When a task has executed on all target machines, Ansible moves on to the next task. You can use *strategies* to change this default behavior. Within each play, Ansible applies the same task directives to all hosts. If a task fails on a host, Ansible takes that host out of the rotation for the rest of the playbook.

When you run a playbook, Ansible returns information about connections, the **name** lines of all your plays and tasks, whether each task has succeeded or failed on each machine, and whether each task has made a change on each machine. At the bottom of the playbook execution, Ansible provides a summary of the nodes that were targeted and how they performed. General failures and fatal “unreachable” communication attempts are kept separate in the counts.

Desired state and ‘idempotency’

Most Ansible modules check whether the desired final state has already been achieved, and exit without performing any actions if that state has been achieved, so that repeating the task does not change the final state. Modules that behave this way are often called ‘idempotent.’ Whether you run a playbook once, or multiple times, the outcome should be the same. However, not all playbooks and not all modules behave this way. If you are unsure, test your playbooks in a sandbox environment before running them multiple times in production.

Running playbooks

To run your playbook, use the `ansible-playbook` command:

```
ansible-playbook playbook.yml -f 10
```

Use the `--verbose` flag when running your playbook to see detailed output from successful modules as well as unsuccessful ones.

Handlers: running operations on change

Sometimes you want a task to run only when a change is made on a machine. For example, you may want to restart a service if a task updates the configuration of that service, but not if the configuration is unchanged. Ansible uses handlers to address this use case. Handlers are tasks that only run when notified. Each handler should have a globally unique name.

This playbook, `verify-apache.yml`, contains a single play with variables, the remote user, and a handler:

```
---
- name: verify apache installation
  hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
    - name: ensure apache is at the latest version
      yum:
        name: httpd
        state: latest
    - name: write the apache config file
      template:
        src: /srv/httpd.j2
        dest: /etc/httpd.conf
      notify:
        - restart apache
    - name: ensure apache is running
      service:
        name: httpd
        state: started
  handlers:
    - name: restart apache
      service:
        name: httpd
        state: restarted
```

In the example above, the second task notifies the handler. A single task can notify more than one handler:

```
- name: template configuration file
  template:
    src: template.j2
    dest: /etc/foo.conf
  notify:
    - restart memcached
    - restart apache
  handlers:
    - name: restart memcached
      service:
        name: memcached
```

(下页继续)

(续上页)

```
    state: restarted
- name: restart apache
  service:
    name: apache
    state: restarted
```

Controlling when handlers run

By default, handlers run after all the tasks in a particular play have been completed. This approach is efficient, because the handler only runs once, regardless of how many tasks notify it. For example, if multiple tasks update a configuration file and notify a handler to restart Apache, Ansible only bounces Apache once to avoid unnecessary restarts.

If you need handlers to run before the end of the play, add a task to flush them using the meta module, which executes Ansible actions:

```
tasks:
- shell: some tasks go here
- meta: flush_handlers
- shell: some other tasks
```

The `meta: flush_handlers` task triggers any handlers that have been notified at that point in the play.

Using variables with handlers

You may want your Ansible handlers to use variables. For example, if the name of a service varies slightly by distribution, you want your output to show the exact name of the restarted service for each target machine. Avoid placing variables in the name of the handler. Since handler names are templated early on, Ansible may not have a value available for a handler name like this:

```
handlers:
# this handler name may cause your play to fail!
- name: restart "{{ web_service_name }}"
```

If the variable used in the handler name is not available, the entire play fails. Changing that variable mid-play **will not** result in newly created handler.

Instead, place variables in the task parameters of your handler. You can load the values using `include_vars` like this:

```
tasks:
  - name: Set host variables based on distribution
    include_vars: "{{ ansible_facts.distribution }}.yaml"

handlers:
  - name: restart web service
    service:
      name: "{{ web_service_name | default('httpd') }}"
      state: restarted
```

Handlers can also “listen” to generic topics, and tasks can notify those topics as follows:

```
handlers:
  - name: restart memcached
    service:
      name: memcached
      state: restarted
    listen: "restart web services"
  - name: restart apache
    service:
      name: apache
      state: restarted
    listen: "restart web services"

tasks:
  - name: restart everything
    command: echo "this task will restart the web services"
    notify: "restart web services"
```

This use makes it much easier to trigger multiple handlers. It also decouples handlers from their names, making it easier to share handlers among playbooks and roles (especially when using 3rd party roles from a shared source like Galaxy).

注解:

- Handlers always run in the order they are defined, not in the order listed in the notify-statement. This is also the case for handlers using *listen*.
- Handler names and *listen* topics live in a global namespace.
- Handler names are templatable and *listen* topics are not.
- Use unique handler names. If you trigger more than one handler with the same name, the first one(s) get overwritten. Only the last one defined will run.

- You can notify a handler defined inside a static include.
 - You cannot notify a handler defined inside a dynamic include.
-

When using handlers within roles, note that:

- handlers notified within `pre_tasks`, `tasks`, and `post_tasks` sections are automatically flushed in the end of section where they were notified.
- handlers notified within `roles` section are automatically flushed in the end of `tasks` section, but before any `tasks` handlers.
- handlers are play scoped and as such can be used outside of the role they are defined in.

Ansible-Pull

Should you want to invert the architecture of Ansible, so that nodes check in to a central location, instead of pushing configuration out to them, you can.

The `ansible-pull` is a small script that will checkout a repo of configuration instructions from git, and then run `ansible-playbook` against that content.

Assuming you load balance your checkout location, `ansible-pull` scales essentially infinitely.

Run `ansible-pull --help` for details.

There's also a [clever playbook](#) available to configure `ansible-pull` via a crontab from push mode.

Verifying playbooks

You may want to verify your playbooks to catch syntax errors and other problems before you run them. The `ansible-playbook` command offers several options for verification, including `--check`, `--diff`, `--list-hosts`, `list-tasks`, and `--syntax-check`. The *Tools for Validating Playbooks* describes other tools for validating and testing playbooks.

ansible-lint

You can use `ansible-lint` for detailed, Ansible-specific feedback on your playbooks before you execute them. For example, if you run `ansible-lint` on the playbook called `verify-apache.yml` near the top of this page, you should get the following results:

```
$ ansible-lint verify-apache.yml
[403] Package installs should not use latest
verify-apache.yml:8
Task/Handler: ensure apache is at the latest version
```

The [ansible-lint default rules](#) page describes each error. For [403], the recommended fix is to change `state: latest` to `state: present` in the playbook.

参见:

[ansible-lint](#) Learn how to test Ansible Playbooks syntax

[YAML Syntax](#) Learn about YAML syntax

[Tips and tricks](#) Tips for managing playbooks in the real world

[all_modules](#) Learn about available modules

[Should you develop a module?](#) Learn to extend Ansible by writing your own modules

[Pattern: 正则匹配主机和组](#) Learn about how to select hosts

[GitHub examples directory](#) Complete end-to-end playbook examples

[Mailing List](#) Questions? Help? Ideas? Stop by the list on Google Groups

Tips and tricks

These tips and tricks have helped us optimize our Ansible usage, and we offer them here as suggestions. We hope they will help you organize content, write playbooks, maintain inventory, and execute Ansible. Ultimately, though, you should use Ansible in the way that makes most sense for your organization and your goals.

- *General tips*
 - *Keep it simple*
 - *Use version control*
- *Playbook tips*
 - *Use whitespace*
 - *Always name tasks*
 - *Always mention the state*
 - *Use comments*
- *Inventory tips*
 - *Use dynamic inventory with clouds*
 - *Group inventory by function*
 - *Separate production and staging inventory*
 - *Keep vaulted variables safely visible*

- *Execution tricks*
 - *Try it in staging first*
 - *Update in batches*
 - *Handling OS and distro differences*

General tips

These concepts apply to all Ansible activities and artifacts.

Keep it simple

Whenever you can, do things simply. Use advanced features only when necessary, and select the feature that best matches your use case. For example, you will probably not need `vars`, `vars_files`, `vars_prompt` and `--extra-vars` all at once, while also using an external inventory file. If something feels complicated, it probably is. Take the time to look for a simpler solution.

Use version control

Keep your playbooks, roles, inventory, and variables files in git or another version control system and make commits to the repository when you make changes. Version control gives you an audit trail describing when and why you changed the rules that automate your infrastructure.

Playbook tips

These tips help make playbooks and roles easier to read, maintain, and debug.

Use whitespace

Generous use of whitespace, for example, a blank line before each block or task, makes a playbook easy to scan.

Always name tasks

Task names are optional, but extremely useful. In its output, Ansible shows you the name of each task it runs. Choose names that describe what each task does and why.

Always mention the state

For many modules, the ‘state’ parameter is optional. Different modules have different default settings for ‘state’, and some modules support several ‘state’ settings. Explicitly setting ‘state=present’ or ‘state=absent’ makes playbooks and roles clearer.

Use comments

Even with task names and explicit state, sometimes a part of a playbook or role (or inventory/variable file) needs more explanation. Adding a comment (any line starting with ‘#’) helps others (and possibly yourself in future) understand what a play or task (or variable setting) does, how it does it, and why.

Inventory tips

These tips help keep your inventory well organized.

Use dynamic inventory with clouds

With cloud providers and other systems that maintain canonical lists of your infrastructure, use *dynamic inventory* to retrieve those lists instead of manually updating static inventory files. With cloud resources, you can use tags to differentiate production and staging environments.

Group inventory by function

A system can be in multiple groups. See *Inventory 使用进阶* and *Pattern: 正则匹配主机和组*. If you create groups named for the function of the nodes in the group, for example *webserver*s or *dbserver*s, your playbooks can target machines based on function. You can assign function-specific variables using the group variable system, and design Ansible roles to handle function-specific use cases. See *Roles*.

Separate production and staging inventory

You can keep your production environment separate from development, test, and staging environments by using separate inventory files or directories for each environment. This way you pick with -i what you are targeting. Keeping all your environments in one file can lead to surprises!

Keep vaulted variables safely visible

You should encrypt sensitive or secret variables with Ansible Vault. However, encrypting the variable names as well as the variable values makes it hard to find the source of the values. You can keep the names of your variables accessible (by `grep`, for example) without exposing any secrets by adding a layer of indirection:

1. Create a `group_vars/` subdirectory named after the group.
2. Inside this subdirectory, create two files named `vars` and `vault`.
3. In the `vars` file, define all of the variables needed, including any sensitive ones.
4. Copy all of the sensitive variables over to the `vault` file and prefix these variables with `vault_`.
5. Adjust the variables in the `vars` file to point to the matching `vault_` variables using jinja2 syntax:
`db_password: {{ vault_db_password }}`.
6. Encrypt the `vault` file to protect its contents.
7. Use the variable name from the `vars` file in your playbooks.

When running a playbook, Ansible finds the variables in the unencrypted file, which pulls the sensitive variable values from the encrypted file. There is no limit to the number of variable and vault files or their names.

Execution tricks

These tips apply to using Ansible, rather than to Ansible artifacts.

Try it in staging first

Testing changes in a staging environment before rolling them out in production is always a great idea. Your environments need not be the same size and you can use group variables to control the differences between those environments.

Update in batches

Use the ‘serial’ keyword to control how many machines you update at once in the batch. See *Delegation, Rolling Updates, and Local Actions*.

Handling OS and distro differences

Group variables files and the `group_by` module work together to help Ansible execute across a range of operating systems and distributions that require different settings, packages, and tools. The `group_by` module creates a dynamic group of hosts matching certain criteria. This group does not need to be defined in the inventory file. This approach lets you execute different tasks on different operating systems or distributions. For example:

```
---
```

(下页继续)

(续上页)

```

- name: talk to all hosts just so we can learn about them
  hosts: all
  tasks:
    - name: Classify hosts depending on their OS distribution
      group_by:
        key: os_{{ ansible_facts['distribution'] }}

# now just on the CentOS hosts...

- hosts: os_CentOS
  gather_facts: False
  tasks:
    - # tasks that only happen on CentOS go in this play

```

The first play categorizes all systems into dynamic groups based on the operating system name. Later plays can use these groups as patterns on the `hosts` line. You can also add group-specific settings in group vars files. All three names must match: the name created by the `group_by` task, the name of the pattern in subsequent plays, and the name of the group vars file. For example:

```

---
# file: group_vars/all
asdf: 10

---
# file: group_vars/os_CentOS.yml
asdf: 42

```

In this example, CentOS machines get the value of ‘42’ for `asdf`, but other machines get ‘10’. This can be used not only to set variables, but also to apply certain roles to only certain systems.

You can use the same setup with `include_vars` when you only need OS-specific variables, not tasks:

```

- hosts: all
  tasks:
    - name: Set OS distribution dependent variables
      include_vars: "os_{{ ansible_facts['distribution'] }}.yaml"
    - debug:
        var: asdf

```

This pulls in variables from the `group_vars/os_CentOS.yml` file.

参见:

YAML Syntax Learn about YAML syntax

Working With Playbooks Review the basic playbook features

all_modules Learn about available modules

Should you develop a module? Learn how to extend Ansible by writing your own modules

Pattern: 正则匹配主机和组 Learn about how to select hosts

GitHub examples directory Complete playbook files from the github project source

Mailing List Questions? Help? Ideas? Stop by the list on Google Groups

Re-using Ansible artifacts

You can write a simple playbook in one very large file, and most users learn the one-file approach first. However, breaking tasks up into different files is an excellent way to organize complex sets of tasks and reuse them. Smaller, more distributed artifacts let you re-use the same variables, tasks, and plays in multiple playbooks to address different use cases. You can use distributed artifacts across multiple parent playbooks or even multiple times within one playbook. For example, you might want to update your customer database as part of several different playbooks. If you put all the tasks related to updating your database in a tasks file, you can re-use them in many playbooks while only maintaining them in one place.

- *Creating re-usable files and roles*
- *Re-using playbooks*
- *Re-using files and roles*
 - *Includes: dynamic re-use*
 - *Imports: static re-use*
 - *Comparing includes and imports: dynamic vs. static*
- *Re-using tasks as handlers*
 - *Triggering included (dynamic) handlers*
 - *Triggering imported (static) handlers*

Creating re-usable files and roles

Ansible offers four distributed, re-usable artifacts: variables files, task files, playbooks, and roles.

- A variables file contains only variables.
- A task file contains only tasks.

- A playbook contains at least one play, and may contain variables, tasks, and other content. You can re-use tightly focused playbooks, but you can only re-use them statically, not dynamically.
- A role contains a set of related tasks, variables, defaults, handlers, and even modules or other plugins in a defined file-tree. Unlike variables files, task files, or playbooks, roles can be easily uploaded and shared via Ansible Galaxy. See *Roles* for details about creating and using roles.

2.4 新版功能.

Re-using playbooks

You can incorporate multiple playbooks into a master playbook. However, you can only use imports to re-use playbooks. For example:

```
- import_playbook: webservers.yml
- import_playbook: databases.yml
```

Importing incorporates playbooks in other playbooks statically. Ansible runs the plays and tasks in each imported playbook in the order they are listed, just as if they had been defined directly in the master playbook.

Re-using files and roles

Ansible offers two ways to re-use files and roles in a playbook: dynamic and static.

- For dynamic re-use, add an `include_*` task in the tasks section of a play:
 - `include_role`
 - `include_tasks`
 - `include_vars`
- For static re-use, add an `import_*` task in the tasks section of a play:
 - `import_role`
 - `import_tasks`

Task include and import statements can be used at arbitrary depth.

You can still use the bare *roles* keyword at the play level to incorporate a role in a playbook statically. However, the bare include keyword, once used for both task files and playbook-level includes, is now deprecated.

Includes: dynamic re-use

Including roles, tasks, or variables adds them to a playbook dynamically. Ansible processes included files and roles as they come up in a playbook, so included tasks can be affected by the results of earlier tasks

within the top-level playbook. Included roles and tasks are similar to handlers - they may or may not run, depending on the results of other tasks in the top-level playbook. The primary advantage of using `include_*` statements is looping. When a loop is used with an include, the included tasks or role will be executed once for each item in the loop.

You can pass variables into includes. See *Variable precedence: Where should I put a variable?* for more details on variable inheritance and precedence.

Imports: static re-use

Importing roles, tasks, or playbooks adds them to a playbook statically. Ansible pre-processes imported files and roles before it runs any tasks in a playbook, so imported content is never affected by other tasks within the top-level playbook.

You can pass variables to imports. You must pass variables if you want to run an imported file more than once in a playbook. For example:

```
tasks:
- import_tasks: wordpress.yml
  vars:
    wp_user: timmy
- import_tasks: wordpress.yml
  vars:
    wp_user: alice
- import_tasks: wordpress.yml
  vars:
    wp_user: bob
```

See *Variable precedence: Where should I put a variable?* for more details on variable inheritance and precedence.

Comparing includes and imports: dynamic vs. static

Each approach to re-using distributed Ansible artifacts has advantages and limitations. You may choose dynamic re-use for some playbooks and static re-use for others. Although you can use both dynamic and static re-use in a single playbook, it is best to select one approach per playbook. Mixing static and dynamic re-use can introduce difficult-to-diagnose bugs into your playbooks. This table summarizes the main differences so you can choose the best approach for each playbook you create.

	Include_*	Import_*
Type of re-use	Dynamic	Static
When processed	At runtime, when encountered	Pre-processed during playbook parsing
Task or play	All includes are tasks	<code>import_playbook</code> cannot be a task
Task options	Apply only to include task itself	Apply to all child tasks in import
Calling from loops	Executed once for each loop item	Cannot be used in a loop
Using <code>--list-tags</code>	Tags within includes not listed	All tags appear with <code>--list-tags</code>
Using <code>--list-tasks</code>	Tasks within includes not listed	All tasks appear with <code>--list-tasks</code>
Notifying handlers	Cannot trigger handlers within includes	Can trigger individual imported handlers
Using <code>--start-at-task</code>	Cannot start at tasks within includes	Can start at imported tasks
Using inventory variables	Can <code>include_*: {{ inventory_var }}</code>	Cannot <code>import_*: {{ inventory_var }}</code>
With playbooks	No <code>include_playbook</code>	Can import full playbooks
With variables files	Can include variables files	Use <code>vars_files:</code> to import variables

Re-using tasks as handlers

You can also use includes and imports in the *Handlers: running operations on change* section of a playbook. For instance, if you want to define how to restart Apache, you only have to do that once for all of your playbooks. You might make a `restarts.yml` file that looks like:

```
# restarts.yml
- name: restart apache
  service:
    name: apache
    state: restarted

- name: restart mysql
  service:
    name: mysql
    state: restarted
```

You can trigger handlers from either an import or an include, but the procedure is different for each method of re-use. If you include the file, you must notify the include itself, which triggers all the tasks in `restarts.yml`. If you import the file, you must notify the individual task(s) within `restarts.yml`. You can mix direct tasks and handlers with included or imported tasks and handlers.

Triggering included (dynamic) handlers

Includes are executed at run-time, so the name of the include exists during play execution, but the included tasks do not exist until the include itself is triggered. To use the `restart apache` task with dynamic re-use, refer to the name of the include itself. This approach triggers all tasks in the included file as handlers. For example, with the task file shown above:

```
- trigger an included (dynamic) handler
hosts: localhost
handlers:
  - name: restart services
    include_tasks: restarts.yml
tasks:
  - command: "true"
    notify: restart services
```

Triggering imported (static) handlers

Imports are processed before the play begins, so the name of the import no longer exists during play execution, but the names of the individual imported tasks do exist. To use the `restart apache` task with static re-use, refer to the name of each task or tasks within the imported file. For example, with the task file shown above:

```
- trigger an imported (static) handler
hosts: localhost
handlers:
  - name: restart services
    import_tasks: restarts.yml
tasks:
  - command: "true"
    notify: restart apache
  - command: "true"
    notify: restart mysql
```

参见:

[utilities__modules](#) Documentation of the `include*` and `import*` modules discussed here.

[Working With Playbooks](#) Review the basic Playbook language features

[Using Variables](#) All about variables in playbooks

[Conditionals](#) Conditionals in playbooks

[Loops](#) Loops in playbooks

Tips and tricks Various tips about managing playbooks in the real world

Galaxy User Guide How to share roles on galaxy, role management

GitHub Ansible examples Complete playbook files from the GitHub project source

Mailing List Questions? Help? Ideas? Stop by the list on Google Groups

Roles

Roles let you automatically load related vars_files, tasks, handlers, and other Ansible artifacts based on a known file structure. Once you group your content in roles, you can easily re-use them and share them with other users.

- *Role directory structure*
- *Storing and finding roles*
- *Using roles*
 - *Using roles at the play level*
 - *Including roles: dynamic re-use*
 - *Importing roles: static re-use*
- *Running a role multiple times in one playbook*
- *Using role dependencies*
 - *Running role dependencies multiple times*
- *Embedding modules and plugins in roles*
- *Sharing roles: Ansible Galaxy*

Role directory structure

An Ansible role has a defined directory structure with seven main standard directories. You must include at least one of these directories in each role. You can omit any directories the role does not use. For example:

```
# playbooks
site.yml
webservers.yml
fooservers.yml
roles/
    common/
```

(下页继续)

(续上页)

```
tasks/
handlers/
library/
files/
templates/
vars/
defaults/
meta/
webservers/
  tasks/
  defaults/
  meta/
```

Each directory within a role must contain a `main.yml` file with relevant content:

- `tasks/main.yml` - the main list of tasks that the role executes.
- `handlers/main.yml` - handlers, which may be used within or outside this role.
- `library/my_module.py` - modules, which may be used within this role (see *Embedding modules and plugins in roles* for more information).
- `defaults/main.yml` - default variables for the role (see *Using Variables* for more information). These variables have the lowest priority of any variables available, and can be easily overridden by any other variable, including inventory variables.
- `vars/main.yml` - other variables for the role (see *Using Variables* for more information).
- `files/main.yml` - files that the role deploys.
- `templates/main.yml` - templates that the role deploys.
- `meta/main.yml` - metadata for the role, including role dependencies.

You can add other YAML files in some directories. For example, you can place platform-specific tasks in separate files and refer to them in the `tasks/main.yml` file:

```
# roles/example/tasks/main.yml
- name: install the correct web server for RHEL
  import_tasks: redhat.yml
  when: ansible_facts['os_family']|lower == 'redhat'
- name: install the correct web server for debian
  import_tasks: debian.yml
  when: ansible_facts['os_family']|lower == 'debian'

# roles/example/tasks/redhat.yml
```

(下页继续)

(续上页)

```
- install web server
  yum:
    name: "httpd"
    state: present

# roles/example/tasks/debian.yml
- install web server
  apt:
    name: "apache2"
    state: present
```

Roles may also include modules and other plugin types in a directory called `library`. For more information, please refer to *Embedding modules and plugins in roles* below.

Storing and finding roles

By default, Ansible looks for roles in two locations:

- in a directory called `roles/`, relative to the playbook file
- in `/etc/ansible/roles`

If you store your roles in a different location, set the `roles_path` configuration option so Ansible can find your roles. Checking shared roles into a single location makes them easier to use in multiple playbooks. See [配置 Ansible](#) for details about managing settings in `ansible.cfg`.

Alternatively, you can call a role with a fully qualified path:

```
---
- hosts: webservers
  roles:
    - role: '/path/to/my/roles/common'
```

Using roles

You can use roles in three ways:

- at the play level with the `roles` option,
- at the tasks level with `include_role`, or
- at the tasks level with `import_role`

Using roles at the play level

The classic (original) way to use roles is with the `roles` option for a given play:

```
---
- hosts: webservers
  roles:
    - common
    - webservers
```

When you use the `roles` option at the play level, for each role ‘x’ :

- If `roles/x/tasks/main.yml` exists, Ansible adds the tasks in that file to the play.
- If `roles/x/handlers/main.yml` exists, Ansible adds the handlers in that file to the play.
- If `roles/x/vars/main.yml` exists, Ansible adds the variables in that file to the play.
- If `roles/x/defaults/main.yml` exists, Ansible adds the variables in that file to the play.
- If `roles/x/meta/main.yml` exists, Ansible adds any role dependencies in that file to the list of roles.
- Any `copy`, `script`, `template` or `include` tasks (in the role) can reference files in `roles/x/{files,templates,tasks}/` (dir depends on task) without having to path them relatively or absolutely.

When you use the `roles` option at the play level, Ansible treats the roles as static imports and processes them during playbook parsing. Ansible executes your playbook in this order:

- Any `pre_tasks` defined in the play.
- Any handlers triggered by `pre_tasks`.
- Each role listed in `roles:`, in the order listed. Any role dependencies defined in the roles `meta/main.yml` run first, subject to tag filtering and conditionals. See [Using role dependencies](#) for more details.
- Any `tasks` defined in the play.
- Any handlers triggered by the roles or tasks.
- Any `post_tasks` defined in the play.
- Any handlers triggered by `post_tasks`.

注解: If using tags with tasks in a role, be sure to also tag your `pre_tasks`, `post_tasks`, and role dependencies and pass those along as well, especially if the `pre/post` tasks and role dependencies are used for monitoring outage window control or load balancing. See [Tags](#) for details on adding and using tags.

You can pass other keywords to the `roles` option:

```
---
- hosts: webservers
  roles:
    - common
    - role: foo_app_instance
      vars:
        dir: '/opt/a'
        app_port: 5000
      tags: typeA
    - role: foo_app_instance
      vars:
        dir: '/opt/b'
        app_port: 5001
      tags: typeB
```

When you add a tag to the `role` option, Ansible applies the tag to ALL tasks within the role.

When using `vars:` within the `roles:` section of a playbook, the variables are added to the play variables, making them available to all tasks within the play before and after the role. This behavior can be changed by `DEFAULT_PRIVATE_ROLE_VARS`.

Including roles: dynamic re-use

You can re-use roles dynamically anywhere in the `tasks` section of a play using `include_role`. While roles added in a `roles` section run before any other tasks in a playbook, included roles run in the order they are defined. If there are other tasks before an `include_role` task, the other tasks will run first.

To include a role:

```
---
- hosts: webservers
  tasks:
    - debug:
        msg: "this task runs before the example role"
    - include_role:
        name: example
    - debug:
        msg: "this task runs after the example role"
```

You can pass other keywords, including variables and tags, when including roles:

```

---
- hosts: webservers
  tasks:
    - include_role:
        name: foo_app_instance
      vars:
        dir: '/opt/a'
        app_port: 5000
      tags: typeA
  ...

```

When you add a *tag* to an `include_role` task, Ansible applies the tag *only* to the include itself. This means you can pass `--tags` to run only selected tasks from the role, if those tasks themselves have the same tag as the include statement. See *Selectively running tagged tasks in re-usable files* for details.

You can conditionally include a role:

```

---
- hosts: webservers
  tasks:
    - include_role:
        name: some_role
      when: "ansible_facts['os_family'] == 'RedHat'"

```

Importing roles: static re-use

You can re-use roles statically anywhere in the `tasks` section of a play using `import_role`. The behavior is the same as using the `roles` keyword. For example:

```

---
- hosts: webservers
  tasks:
    - debug:
        msg: "before we run our role"
    - import_role:
        name: example
    - debug:
        msg: "after we ran our role"

```

You can pass other keywords, including variables and tags, when importing roles:

```
---
- hosts: webservers
  tasks:
    - import_role:
        name: foo_app_instance
      vars:
        dir: '/opt/a'
        app_port: 5000
    ...
```

When you add a tag to an `import_role` statement, Ansible applies the tag to *all* tasks within the role. See *Tag inheritance: adding tags to multiple tasks* for details.

Running a role multiple times in one playbook

Ansible only executes each role once, even if you define it multiple times, unless the parameters defined on the role are different for each definition. For example, Ansible only runs the role `foo` once in a play like this:

```
---
- hosts: webservers
  roles:
    - foo
    - bar
    - foo
```

You have two options to force Ansible to run a role more than once:

1. Pass different parameters in each role definition.
2. Add `allow_duplicates: true` to the `meta/main.yml` file for the role.

Example 1 - passing different parameters:

```
---
- hosts: webservers
  roles:
    - role: foo
      vars:
        message: "first"
    - { role: foo, vars: { message: "second" } }
```

In this example, because each role definition has different parameters, Ansible runs `foo` twice.

Example 2 - using `allow_duplicates: true`:

```
# playbook.yml
---
- hosts: webservers
  roles:
    - foo
    - foo

# roles/foo/meta/main.yml
---
allow_duplicates: true
```

In this example, Ansible runs `foo` twice because we have explicitly enabled it to do so.

Using role dependencies

Role dependencies let you automatically pull in other roles when using a role. Ansible does not execute role dependencies when you include or import a role. You must use the `roles` keyword if you want Ansible to execute role dependencies.

Role dependencies are stored in the `meta/main.yml` file within the role directory. This file should contain a list of roles and parameters to insert before the specified role. For example:

```
# roles/myapp/meta/main.yml
---
dependencies:
  - role: common
    vars:
      some_parameter: 3
  - role: apache
    vars:
      apache_port: 80
  - role: postgres
    vars:
      dbname: blarg
      other_parameter: 12
```

Ansible always executes role dependencies before the role that includes them. Ansible executes recursive role dependencies as well. If one role depends on a second role, and the second role depends on a third role, Ansible executes the third role, then the second role, then the first role.

Running role dependencies multiple times

Ansible treats duplicate role dependencies like duplicate roles listed under `roles::`. Ansible only executes role dependencies once, even if defined multiple times, unless the parameters defined on the role are different for each definition. If two roles in a playbook both list a third role as a dependency, Ansible only runs that role dependency once, unless you pass different parameters or use `allow_duplicates: true` in the dependent (third) role. See *Galaxy role dependencies* for more details.

For example, a role named `car` depends on a role named `wheel` as follows:

```
---
dependencies:
  - role: wheel
    vars:
      n: 1
  - role: wheel
    vars:
      n: 2
  - role: wheel
    vars:
      n: 3
  - role: wheel
    vars:
      n: 4
```

And the `wheel` role depends on two roles: `tire` and `brake`. The `meta/main.yml` for `wheel` would then contain the following:

```
---
dependencies:
  - role: tire
  - role: brake
```

And the `meta/main.yml` for `tire` and `brake` would contain the following:

```
---
allow_duplicates: true
```

The resulting order of execution would be as follows:

```
tire(n=1)
brake(n=1)
wheel(n=1)
```

(下页继续)

(续上页)

```
tire(n=2)
brake(n=2)
wheel(n=2)
...
car
```

To use `allow_duplicates: true` with role dependencies, you must specify it for the dependent role, not for the parent role. In the example above, `allow_duplicates: true` appears in the `meta/main.yml` of the `tire` and `brake` roles. The `wheel` role does not require `allow_duplicates: true`, because each instance defined by `car` uses different parameter values.

注解: See [Using Variables](#) for details on how Ansible chooses among variable values defined in different places (variable inheritance and scope).

Embedding modules and plugins in roles

If you write a custom module (see [Should you develop a module?](#)) or a plugin (see [Developing plugins](#)), you might wish to distribute it as part of a role. For example, if you write a module that helps configure your company's internal software, and you want other people in your organization to use this module, but you do not want to tell everyone how to configure their Ansible library path, you can include the module in your `internal_config` role.

Alongside the 'tasks' and 'handlers' structure of a role, add a directory named 'library'. In this 'library' directory, then include the module directly inside of it.

Assuming you had this:

```
roles/
  my_custom_modules/
    library/
      module1
      module2
```

The module will be usable in the role itself, as well as any roles that are called *after* this role, as follows:

```
---
- hosts: webservers
  roles:
    - my_custom_modules
```

(下页继续)

(续上页)

```
- some_other_role_using_my_custom_modules
- yet_another_role_using_my_custom_modules
```

If necessary, you can also embed a module in a role to modify a module in Ansible’s core distribution. For example, you can use the development version of a particular module before it is released in production releases by copying the module and embedding the copy in a role. Use this approach with caution, as API signatures may change in core components, and this workaround is not guaranteed to work.

The same mechanism can be used to embed and distribute plugins in a role, using the same schema. For example, for a filter plugin:

```
roles/
  my_custom_filter/
    filter_plugins
      filter1
      filter2
```

These filters can then be used in a Jinja template in any role called after ‘my_custom_filter’ .

Sharing roles: Ansible Galaxy

[Ansible Galaxy](#) is a free site for finding, downloading, rating, and reviewing all kinds of community-developed Ansible roles and can be a great way to get a jumpstart on your automation projects.

The client `ansible-galaxy` is included in Ansible. The Galaxy client allows you to download roles from Ansible Galaxy, and also provides an excellent default framework for creating your own roles.

Read the [Ansible Galaxy documentation](#) page for more information

参见:

[Galaxy User Guide](#) How to create new roles, share roles on Galaxy, role management

[YAML Syntax](#) Learn about YAML syntax

[Working With Playbooks](#) Review the basic Playbook language features

[Tips and tricks](#) Tips for managing playbooks in the real world

[Using Variables](#) Variables in playbooks

[Conditionals](#) Conditionals in playbooks

[Loops](#) Loops in playbooks

[Tags](#) Using tags to select or skip roles/tasks in long playbooks

all_modules List of available modules

Should you develop a module? Extending Ansible by writing your own modules

GitHub Ansible examples Complete playbook files from the GitHub project source

Mailing List Questions? Help? Ideas? Stop by the list on Google Groups

Using Variables

- *Creating valid variable names*
- *Defining variables in inventory*
- *Defining variables in a playbook*
- *Defining variables in included files and roles*
- *Using variables with Jinja2*
- *Transforming variables with Jinja2 filters*
- *Hey wait, a YAML gotcha*
- *Variables discovered from systems: Facts*
 - *Disabling facts*
 - *Local facts (facts.d)*
 - *Ansible version*
 - *Caching Facts*
- *Registering variables*
- *Accessing complex variable data*
- *Accessing information about other hosts with magic variables*
- *Defining variables in files*
- *Passing variables on the command line*
- *Variable precedence: Where should I put a variable?*
 - *Scoping variables*
 - *Examples of where to set a variable*
- *Using advanced variable syntax*

While automation exists to make it easier to make things repeatable, all systems are not exactly alike; some may require configuration that is slightly different from others. In some instances, the observed behavior or state of one system might influence how you configure other systems. For example, you might need to find out the IP address of a system and use it as a configuration value on another system.

Ansible uses *variables* to help deal with differences between systems.

To understand variables you'll also want to read *Conditionals* and *Loops*. Useful things like the **group_by** module and the **when** conditional can also be used with variables, and to help manage differences between systems.

The [ansible-examples github repository](#) contains many examples of how variables are used in Ansible.

Creating valid variable names

Before you start using variables, it's important to know what are valid variable names.

Variable names should be letters, numbers, and underscores. Variables should always start with a letter.

`foo_port` is a great variable. `foo5` is fine too.

`foo-port`, `foo port`, `foo.port` and `12` are not valid variable names.

Python keywords such as `async` and `lambda` are not valid variable names and thus must be avoided.

YAML also supports dictionaries which map keys to values. For instance:

```
foo:
  field1: one
  field2: two
```

You can then reference a specific field in the dictionary using either bracket notation or dot notation:

```
foo['field1']
foo.field1
```

These will both reference the same value (“one”). However, if you choose to use dot notation be aware that some keys can cause problems because they collide with attributes and methods of python dictionaries. You should use bracket notation instead of dot notation if you use keys which start and end with two underscores (Those are reserved for special meanings in python) or are any of the known public attributes:

`add`, `append`, `as_integer_ratio`, `bit_length`, `capitalize`, `center`, `clear`, `conjugate`, `copy`, `count`, `decode`, `denominator`, `difference`, `difference_update`, `discard`, `encode`, `endswith`, `expandtabs`, `extend`, `find`, `format`, `fromhex`, `fromkeys`, `get`, `has_key`, `hex`, `imag`, `index`, `insert`, `intersection`, `intersection_update`, `isalnum`, `isalpha`, `isdecimal`, `isdigit`, `isdisjoint`, `is_integer`, `islower`, `isnumeric`, `isspace`, `issubset`, `issuperset`, `istitle`, `isupper`, `items`, `iteritems`, `iterkeys`, `itervalues`, `join`, `keys`, `ljust`, `lower`, `lstrip`, `numerator`, `partition`, `pop`, `popitem`, `real`, `remove`, `replace`, `reverse`, `rfind`, `rindex`, `rjust`, `rpartition`, `rsplit`, `rstrip`, `setdefault`, `sort`, `split`, `splitlines`, `startswith`, `strip`, `swapcase`, `symmetric_difference`, `symmetric_difference_update`, `title`, `translate`, `union`, `update`, `upper`, `values`, `viewitems`, `viewkeys`, `viewvalues`, `zfill`.

Defining variables in inventory

Often you’ ll want to set variables for an individual host, or for a group of hosts in your inventory. For instance, machines in Boston may all use ‘boston.ntp.example.com’ as an NTP server. The *Inventory 使用进阶* page has details on setting 给单台主机设置变量: *host variables* and 给多台主机设置变量: *group variables* in inventory.

Defining variables in a playbook

You can define variables directly in a playbook:

```
- hosts: webservers
  vars:
    http_port: 80
```

This can be nice as it’ s right there when you are reading the playbook.

Defining variables in included files and roles

As described in *Roles*, variables can also be included in the playbook via include files, which may or may not be part of an Ansible Role. Usage of roles is preferred as it provides a nice organizational system.

Using variables with Jinja2

Once you’ ve defined variables, you can use them in your playbooks using the Jinja2 templating system. Here’ s a simple Jinja2 template:

```
My amp goes to {{ max_amp_value }}
```

This expression provides the most basic form of variable substitution.

You can use the same syntax in playbooks. For example:

```
template: src=foo.cfg.j2 dest={{ remote_install_path }}/foo.cfg
```

Here the variable defines the location of a file, which can vary from one system to another.

Inside a template you automatically have access to all variables that are in scope for a host. Actually it’ s more than that – you can also read variables about other hosts. We’ ll show how to do that in a bit.

注解: ansible allows Jinja2 loops and conditionals in templates, but in playbooks, we do not use them. Ansible playbooks are pure machine-parseable YAML. This is a rather important feature as it means it is

possible to code-generate pieces of files, or to have other ecosystem tools read Ansible files. Not everyone will need this but it can unlock possibilities.

参见:

Templating (Jinja2) More information about Jinja2 templating

Transforming variables with Jinja2 filters

Jinja2 filters let you transform the value of a variable within a template expression. For example, the `capitalize` filter capitalizes any value passed to it; the `to_yaml` and `to_json` filters change the format of your variable values. Jinja2 includes many [built-in filters](#) and Ansible supplies *many more filters*.

Hey wait, a YAML gotcha

YAML syntax requires that if you start a value with `{{ foo }}` you quote the whole line, since it wants to be sure you aren't trying to start a YAML dictionary. This is covered on the [YAML Syntax](#) documentation.

This won't work:

```
- hosts: app_servers
  vars:
    app_path: {{ base_path }}/22
```

Do it like this and you'll be fine:

```
- hosts: app_servers
  vars:
    app_path: "{{ base_path }}/22"
```

Variables discovered from systems: Facts

There are other places where variables can come from, but these are a type of variable that are discovered, not set by the user.

Facts are information derived from speaking with your remote systems. You can find a complete set under the `ansible_facts` variable, most facts are also 'injected' as top level variables preserving the `ansible_` prefix, but some are dropped due to conflicts. This can be disabled via the `INJECT_FACTS_AS_VARS` setting.

An example of this might be the IP address of the remote host, or what the operating system is.

To see what information is available, try the following in a play:

```
- debug: var=ansible_facts
```

To see the ‘raw’ information as gathered:

```
ansible hostname -m setup
```

This will return a large amount of variable data, which may look like this on Ansible 2.7:

```
{
  "ansible_all_ipv4_addresses": [
    "REDACTED IP ADDRESS"
  ],
  "ansible_all_ipv6_addresses": [
    "REDACTED IPV6 ADDRESS"
  ],
  "ansible_apparmor": {
    "status": "disabled"
  },
  "ansible_architecture": "x86_64",
  "ansible_bios_date": "11/28/2013",
  "ansible_bios_version": "4.1.5",
  "ansible_cmdline": {
    "BOOT_IMAGE": "/boot/vmlinuz-3.10.0-862.14.4.el7.x86_64",
    "console": "ttyS0,115200",
    "no_timer_check": true,
    "nofb": true,
    "nomodeset": true,
    "ro": true,
    "root": "LABEL=cloudimg-rootfs",
    "vga": "normal"
  },
  "ansible_date_time": {
    "date": "2018-10-25",
    "day": "25",
    "epoch": "1540469324",
    "hour": "12",
    "iso8601": "2018-10-25T12:08:44Z",
    "iso8601_basic": "20181025T120844109754",
    "iso8601_basic_short": "20181025T120844",
    "iso8601_micro": "2018-10-25T12:08:44.109968Z",
    "minute": "08",
```

(下页继续)

(续上页)

```

    "month": "10",
    "second": "44",
    "time": "12:08:44",
    "tz": "UTC",
    "tz_offset": "+0000",
    "weekday": "Thursday",
    "weekday_number": "4",
    "weeknumber": "43",
    "year": "2018"
  },
  "ansible_default_ipv4": {
    "address": "REDACTED",
    "alias": "eth0",
    "broadcast": "REDACTED",
    "gateway": "REDACTED",
    "interface": "eth0",
    "macaddress": "REDACTED",
    "mtu": 1500,
    "netmask": "255.255.255.0",
    "network": "REDACTED",
    "type": "ether"
  },
  "ansible_default_ipv6": {},
  "ansible_device_links": {
    "ids": {},
    "labels": {
      "xvda1": [
        "cloudimg-rootfs"
      ],
      "xvdd": [
        "config-2"
      ]
    }
  },
  "masters": {},
  "uuids": {
    "xvda1": [
      "cac81d61-d0f8-4b47-84aa-b48798239164"
    ],
    "xvdd": [
      "2018-10-25-12-05-57-00"
    ]
  }
}

```

(下页继续)

(续上页)

```

    ]
  }
},
"ansible_devices": {
  "xvda": {
    "holders": [],
    "host": "",
    "links": {
      "ids": [],
      "labels": [],
      "masters": [],
      "uuids": []
    },
    "model": null,
    "partitions": {
      "xvda1": {
        "holders": [],
        "links": {
          "ids": [],
          "labels": [
            "cloudimg-rootfs"
          ],
          "masters": [],
          "uuids": [
            "cac81d61-d0f8-4b47-84aa-b48798239164"
          ]
        },
        "sectors": "83883999",
        "sectorsize": 512,
        "size": "40.00 GB",
        "start": "2048",
        "uuid": "cac81d61-d0f8-4b47-84aa-b48798239164"
      }
    },
    "removable": "0",
    "rotational": "0",
    "sas_address": null,
    "sas_device_handle": null,
    "scheduler_mode": "deadline",
    "sectors": "83886080",

```

(下页继续)

(续上页)

```
    "sectorsize": "512",
    "size": "40.00 GB",
    "support_discard": "0",
    "vendor": null,
    "virtual": 1
  },
  "xvdd": {
    "holders": [],
    "host": "",
    "links": {
      "ids": [],
      "labels": [
        "config-2"
      ],
      "masters": [],
      "uuids": [
        "2018-10-25-12-05-57-00"
      ]
    },
    "model": null,
    "partitions": {},
    "removable": "0",
    "rotational": "0",
    "sas_address": null,
    "sas_device_handle": null,
    "scheduler_mode": "deadline",
    "sectors": "131072",
    "sectorsize": "512",
    "size": "64.00 MB",
    "support_discard": "0",
    "vendor": null,
    "virtual": 1
  },
  "xvde": {
    "holders": [],
    "host": "",
    "links": {
      "ids": [],
      "labels": [],
      "masters": [],
```

(下页继续)

(续上页)

```

        "uuids": []
    },
    "model": null,
    "partitions": {
        "xvde1": {
            "holders": [],
            "links": {
                "ids": [],
                "labels": [],
                "masters": [],
                "uuids": []
            },
            "sectors": "167770112",
            "sectorsize": 512,
            "size": "80.00 GB",
            "start": "2048",
            "uuid": null
        }
    },
    "removable": "0",
    "rotational": "0",
    "sas_address": null,
    "sas_device_handle": null,
    "scheduler_mode": "deadline",
    "sectors": "167772160",
    "sectorsize": "512",
    "size": "80.00 GB",
    "support_discard": "0",
    "vendor": null,
    "virtual": 1
}
},
"ansible_distribution": "CentOS",
"ansible_distribution_file_parsed": true,
"ansible_distribution_file_path": "/etc/redhat-release",
"ansible_distribution_file_variety": "RedHat",
"ansible_distribution_major_version": "7",
"ansible_distribution_release": "Core",
"ansible_distribution_version": "7.5.1804",
"ansible_dns": {

```

(下页继续)

(续上页)

```

    "nameservers": [
        "127.0.0.1"
    ]
},
"ansible_domain": "",
"ansible_effective_group_id": 1000,
"ansible_effective_user_id": 1000,
"ansible_env": {
    "HOME": "/home/zuul",
    "LANG": "en_US.UTF-8",
    "LESSOPEN": "||/usr/bin/lesspipe.sh %s",
    "LOGNAME": "zuul",
    "MAIL": "/var/mail/zuul",
    "PATH": "/usr/local/bin:/usr/bin",
    "PWD": "/home/zuul",
    "SELINUX_LEVEL_REQUESTED": "",
    "SELINUX_ROLE_REQUESTED": "",
    "SELINUX_USE_CURRENT_RANGE": "",
    "SHELL": "/bin/bash",
    "SHLVL": "2",
    "SSH_CLIENT": "REDACTED 55672 22",
    "SSH_CONNECTION": "REDACTED 55672 REDACTED 22",
    "USER": "zuul",
    "XDG_RUNTIME_DIR": "/run/user/1000",
    "XDG_SESSION_ID": "1",
    "_": "/usr/bin/python2"
},
"ansible_eth0": {
    "active": true,
    "device": "eth0",
    "ipv4": {
        "address": "REDACTED",
        "broadcast": "REDACTED",
        "netmask": "255.255.255.0",
        "network": "REDACTED"
    },
    "ipv6": [
        {
            "address": "REDACTED",
            "prefix": "64",

```

(下页继续)

(续上页)

```

        "scope": "link"
    }
],
"macaddress": "REDACTED",
"module": "xen_netfront",
"mtu": 1500,
"pciid": "vif-0",
"promisc": false,
"type": "ether"
},
"ansible_eth1": {
    "active": true,
    "device": "eth1",
    "ipv4": {
        "address": "REDACTED",
        "broadcast": "REDACTED",
        "netmask": "255.255.224.0",
        "network": "REDACTED"
    },
    "ipv6": [
        {
            "address": "REDACTED",
            "prefix": "64",
            "scope": "link"
        }
    ],
    "macaddress": "REDACTED",
    "module": "xen_netfront",
    "mtu": 1500,
    "pciid": "vif-1",
    "promisc": false,
    "type": "ether"
},
"ansible_fips": false,
"ansible_form_factor": "Other",
"ansible_fqdn": "centos-7-rax-dfw-0003427354",
"ansible_hostname": "centos-7-rax-dfw-0003427354",
"ansible_interfaces": [
    "lo",
    "eth1",

```

(下页继续)

(续上页)

```
    "eth0"
  ],
  "ansible_is_chroot": false,
  "ansible_kernel": "3.10.0-862.14.4.el7.x86_64",
  "ansible_lo": {
    "active": true,
    "device": "lo",
    "ipv4": {
      "address": "127.0.0.1",
      "broadcast": "host",
      "netmask": "255.0.0.0",
      "network": "127.0.0.0"
    },
    "ipv6": [
      {
        "address": "::1",
        "prefix": "128",
        "scope": "host"
      }
    ],
    "mtu": 65536,
    "promisc": false,
    "type": "loopback"
  },
  "ansible_local": {},
  "ansible_lsb": {
    "codename": "Core",
    "description": "CentOS Linux release 7.5.1804 (Core)",
    "id": "CentOS",
    "major_release": "7",
    "release": "7.5.1804"
  },
  "ansible_machine": "x86_64",
  "ansible_machine_id": "2db133253c984c82aef2fafc6f2bed",
  "ansible_memfree_mb": 7709,
  "ansible_memory_mb": {
    "nocache": {
      "free": 7804,
      "used": 173
    }
  },
```

(下页继续)

(续上页)

```

    "real": {
        "free": 7709,
        "total": 7977,
        "used": 268
    },
    "swap": {
        "cached": 0,
        "free": 0,
        "total": 0,
        "used": 0
    }
},
"ansible_memtotal_mb": 7977,
"ansible_mounts": [
    {
        "block_available": 7220998,
        "block_size": 4096,
        "block_total": 9817227,
        "block_used": 2596229,
        "device": "/dev/xvda1",
        "fstype": "ext4",
        "inode_available": 10052341,
        "inode_total": 10419200,
        "inode_used": 366859,
        "mount": "/",
        "options": "rw,seclabel,relatime,data=ordered",
        "size_available": 29577207808,
        "size_total": 40211361792,
        "uuid": "cac81d61-d0f8-4b47-84aa-b48798239164"
    },
    {
        "block_available": 0,
        "block_size": 2048,
        "block_total": 252,
        "block_used": 252,
        "device": "/dev/xvdd",
        "fstype": "iso9660",
        "inode_available": 0,
        "inode_total": 0,
        "inode_used": 0,

```

(下页继续)

(续上页)

```

        "mount": "/mnt/config",
        "options": "ro,relatime,mode=0700",
        "size_available": 0,
        "size_total": 516096,
        "uuid": "2018-10-25-12-05-57-00"
    }
],
"ansible_nodename": "centos-7-rax-dfw-0003427354",
"ansible_os_family": "RedHat",
"ansible_pkg_mgr": "yum",
"ansible_processor": [
    "0",
    "GenuineIntel",
    "Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz",
    "1",
    "GenuineIntel",
    "Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz",
    "2",
    "GenuineIntel",
    "Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz",
    "3",
    "GenuineIntel",
    "Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz",
    "4",
    "GenuineIntel",
    "Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz",
    "5",
    "GenuineIntel",
    "Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz",
    "6",
    "GenuineIntel",
    "Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz",
    "7",
    "GenuineIntel",
    "Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz"
],
"ansible_processor_cores": 8,
"ansible_processor_count": 8,
"ansible_processor_threads_per_core": 1,
"ansible_processor_vcpus": 8,

```

(下页继续)

(续上页)

```

"ansible_product_name": "HVM domU",
"ansible_product_serial": "REDACTED",
"ansible_product_uuid": "REDACTED",
"ansible_product_version": "4.1.5",
"ansible_python": {
    "executable": "/usr/bin/python2",
    "has_sslcontext": true,
    "type": "CPython",
    "version": {
        "major": 2,
        "micro": 5,
        "minor": 7,
        "releaselevel": "final",
        "serial": 0
    },
    "version_info": [
        2,
        7,
        5,
        "final",
        0
    ]
},
"ansible_python_version": "2.7.5",
"ansible_real_group_id": 1000,
"ansible_real_user_id": 1000,
"ansible_selinux": {
    "config_mode": "enforcing",
    "mode": "enforcing",
    "policyvers": 31,
    "status": "enabled",
    "type": "targeted"
},
"ansible_selinux_python_present": true,
"ansible_service_mgr": "systemd",
"ansible_ssh_host_key_ecdsa_public": "REDACTED KEY VALUE",
"ansible_ssh_host_key_ed25519_public": "REDACTED KEY VALUE",
"ansible_ssh_host_key_rsa_public": "REDACTED KEY VALUE",
"ansible_swapfree_mb": 0,
"ansible_swaptotal_mb": 0,

```

(下页继续)

(续上页)

```

"ansible_system": "Linux",
"ansible_system_capabilities": [
    ""
],
"ansible_system_capabilities_enforced": "True",
"ansible_system_vendor": "Xen",
"ansible_uptime_seconds": 151,
"ansible_user_dir": "/home/zuul",
"ansible_user_gecos": "",
"ansible_user_gid": 1000,
"ansible_user_id": "zuul",
"ansible_user_shell": "/bin/bash",
"ansible_user_uid": 1000,
"ansible_userspace_architecture": "x86_64",
"ansible_userspace_bits": "64",
"ansible_virtualization_role": "guest",
"ansible_virtualization_type": "xen",
"gather_subset": [
    "all"
],
"module_setup": true
}

```

In the above the model of the first disk may be referenced in a template or playbook as:

```
{{ ansible_facts['devices']['xvda']['model'] }}
```

Similarly, the hostname as the system reports it is:

```
{{ ansible_facts['nodename'] }}
```

Facts are frequently used in conditionals (see *Conditionals*) and also in templates.

Facts can be also used to create dynamic groups of hosts that match particular criteria, see the [Importing Modules](#) documentation on **group_by** for details, as well as in generalized conditional statements as discussed in the *Conditionals* chapter.

Disabling facts

If you know you don't need any fact data about your hosts, and know everything about your systems centrally, you can turn off fact gathering. This has advantages in scaling Ansible in push mode with very

large numbers of systems, mainly, or if you are using Ansible on experimental platforms. In any play, just do this:

```
- hosts: whatever
  gather_facts: no
```

Local facts (facts.d)

1.3 新版功能.

As discussed in the playbooks chapter, Ansible facts are a way of getting data about remote systems for use in playbook variables.

Usually these are discovered automatically by the `setup` module in Ansible. Users can also write custom facts modules, as described in the API guide. However, what if you want to have a simple way to provide system or user provided data for use in Ansible variables, without writing a fact module?

“Facts.d” is one mechanism for users to control some aspect of how their systems are managed.

注解: Perhaps “local facts” is a bit of a misnomer, it means “locally supplied user values” as opposed to “centrally supplied user values”, or what facts are – “locally dynamically determined values”.

If a remotely managed system has an `/etc/ansible/facts.d` directory, any files in this directory ending in `.fact`, can be JSON, INI, or executable files returning JSON, and these can supply local facts in Ansible. An alternate directory can be specified using the `fact_path` play keyword.

For example, assume `/etc/ansible/facts.d/preferences.fact` contains:

```
[general]
asdf=1
bar=2
```

This will produce a hash variable fact named `general` with `asdf` and `bar` as members. To validate this, run the following:

```
ansible <hostname> -m setup -a "filter=ansible_local"
```

And you will see the following fact added:

```
"ansible_local": {
  "preferences": {
    "general": {
      "asdf" : "1",
```

(下页继续)

(续上页)

```

        "bar" : "2"
    }
}

```

And this data can be accessed in a `template/playbook` as:

```
{{ ansible_local['preferences']['general']['asdf'] }}
```

The local namespace prevents any user supplied fact from overriding system facts or variables defined elsewhere in the playbook.

注解: The key part in the key=value pairs will be converted into lowercase inside the `ansible_local` variable. Using the example above, if the ini file contained `XYZ=3` in the `[general]` section, then you should expect to access it as: `{{ ansible_local['preferences']['general']['xyz'] }}` and not `{{ ansible_local['preferences']['general']['XYZ'] }}`. This is because Ansible uses Python's `ConfigParser` which passes all option names through the `optionxform` method and this method's default implementation converts option names to lower case.

If you have a playbook that is copying over a custom fact and then running it, making an explicit call to re-run the setup module can allow that fact to be used during that particular play. Otherwise, it will be available in the next play that gathers fact information. Here is an example of what that might look like:

```

- hosts: webserver
  tasks:
    - name: create directory for ansible custom facts
      file: state=directory recurse=yes path=/etc/ansible/facts.d
    - name: install custom ipmi fact
      copy: src=ipmi.fact dest=/etc/ansible/facts.d
    - name: re-read facts after adding custom fact
      setup: filter=ansible_local

```

In this pattern however, you could also write a fact module as well, and may wish to consider this as an option.

Ansible version

1.8 新版功能.

To adapt playbook behavior to specific version of ansible, a variable `ansible_version` is available, with the following structure:

```
"ansible_version": {
  "full": "2.0.0.2",
  "major": 2,
  "minor": 0,
  "revision": 0,
  "string": "2.0.0.2"
}
```

Caching Facts

1.8 新版功能.

As shown elsewhere in the docs, it is possible for one server to reference variables about another, like so:

```
{{ hostvars['asdf.example.com']['ansible_facts']['os_family'] }}
```

With “Fact Caching” disabled, in order to do this, Ansible must have already talked to ‘asdf.example.com’ in the current play, or another play up higher in the playbook. This is the default configuration of ansible.

To avoid this, Ansible 1.8 allows the ability to save facts between playbook runs, but this feature must be manually enabled. Why might this be useful?

With a very large infrastructure with thousands of hosts, fact caching could be configured to run nightly. Configuration of a small set of servers could run ad-hoc or periodically throughout the day. With fact caching enabled, it would not be necessary to “hit” all servers to reference variables and information about them.

With fact caching enabled, it is possible for machine in one group to reference variables about machines in the other group, despite the fact that they have not been communicated with in the current execution of /usr/bin/ansible-playbook.

To benefit from cached facts, you will want to change the **gathering** setting to **smart** or **explicit** or set **gather_facts** to **False** in most plays.

Currently, Ansible ships with two persistent cache plugins: redis and jsonfile.

To configure fact caching using redis, enable it in **ansible.cfg** as follows:

```
[defaults]
gathering = smart
fact_caching = redis
fact_caching_timeout = 86400
# seconds
```

To get redis up and running, perform the equivalent OS commands:

```
yum install redis
service redis start
pip install redis
```

Note that the Python redis library should be installed from pip, the version packaged in EPEL is too old for use by Ansible.

In current embodiments, this feature is in beta-level state and the Redis plugin does not support port or password configuration, this is expected to change in the near future.

To configure fact caching using jsonfile, enable it in `ansible.cfg` as follows:

```
[defaults]
gathering = smart
fact_caching = jsonfile
fact_caching_connection = /path/to/cachedir
fact_caching_timeout = 86400
# seconds
```

`fact_caching_connection` is a local filesystem path to a writeable directory (ansible will attempt to create the directory if one does not exist).

`fact_caching_timeout` is the number of seconds to cache the recorded facts.

Registering variables

Another major use of variables is running a command and registering the result of that command as a variable. When you execute a task and save the return value in a variable for use in later tasks, you create a registered variable. There are more examples of this in the *Conditionals* chapter.

For example:

```
- hosts: web_servers

  tasks:

    - shell: /usr/bin/foo
      register: foo_result
      ignore_errors: True

    - shell: /usr/bin/bar
      when: foo_result.rc == 5
```

Results will vary from module to module. Each module’s documentation includes a **RETURN** section describing that module’s return values. To see the values for a particular task, run your playbook with `-v`.

Registered variables are similar to facts, with a few key differences. Like facts, registered variables are host-level variables. However, registered variables are only stored in memory. (Ansible facts are backed by whatever cache plugin you have configured.) Registered variables are only valid on the host for the rest of the current playbook run. Finally, registered variables and facts have different *precedence levels*.

When you register a variable in a task with a loop, the registered variable contains a value for each item in the loop. The data structure placed in the variable during the loop will contain a **results** attribute, that is a list of all responses from the module. For a more in-depth example of how this works, see the *Loops* section on using register with a loop.

注解: If a task fails or is skipped, the variable still is registered with a failure or skipped status, the only way to avoid registering a variable is using tags.

Accessing complex variable data

We already described facts a little higher up in the documentation.

Some provided facts, like networking information, are made available as nested data structures. To access them a simple `{{ foo }}` is not sufficient, but it is still easy to do. Here’s how we get an IP address:

```
{{ ansible_facts["eth0"]["ipv4"]["address"] }}
```

OR alternatively:

```
{{ ansible_facts.eth0.ipv4.address }}
```

Similarly, this is how we access the first element of an array:

```
{{ foo[0] }}
```

Accessing information about other hosts with magic variables

Whether or not you define any variables, you can access information about your hosts with the *Special Variables* Ansible provides, including “magic” variables, facts, and connection variables. Magic variable names are reserved - do not set variables with these names. The variable **environment** is also reserved.

The most commonly used magic variables are **hostvars**, **groups**, **group_names**, and **inventory_hostname**.

hostvars lets you access variables for another host, including facts that have been gathered about that host. You can access host variables at any point in a playbook. Even if you haven’t connected to that host yet

in any play in the playbook or set of playbooks, you can still get the variables, but you will not be able to see the facts.

If your database server wants to use the value of a ‘fact’ from another node, or an inventory variable assigned to another node, it’s easy to do so within a template or even an action line:

```
{{ hostvars['test.example.com']['ansible_facts']['distribution'] }}
```

`groups` is a list of all the groups (and hosts) in the inventory. This can be used to enumerate all hosts within a group. For example:

```
{% for host in groups['app_servers'] %}
    # something that applies to all app servers.
{% endfor %}
```

A frequently used idiom is walking a group to find all IP addresses in that group.

```
{% for host in groups['app_servers'] %}
    {{ hostvars[host]['ansible_facts']['eth0']['ipv4']['address'] }}
{% endfor %}
```

You can use this idiom to point a frontend proxy server to all of the app servers, to set up the correct firewall rules between servers, etc. You need to make sure that the facts of those hosts have been populated before though, for example by running a play against them if the facts have not been cached recently (fact caching was added in Ansible 1.8).

`group_names` is a list (array) of all the groups the current host is in. This can be used in templates using Jinja2 syntax to make template source files that vary based on the group membership (or role) of the host:

```
{% if 'webserver' in group_names %}
    # some part of a configuration file that only applies to webservers
{% endif %}
```

`inventory_hostname` is the name of the hostname as configured in Ansible’s inventory host file. This can be useful when you’ve disabled fact-gathering, or you don’t want to rely on the discovered hostname `ansible_hostname`. If you have a long FQDN, you can use `inventory_hostname_short`, which contains the part up to the first period, without the rest of the domain.

Other useful magic variables refer to the current play or playbook, including:

2.2 新版功能.

`ansible_play_hosts` is the full list of all hosts still active in the current play.

2.2 新版功能.

`ansible_play_batch` is available as a list of hostnames that are in scope for the current ‘batch’ of the

play. The batch size is defined by `serial`, when not set it is equivalent to the whole play (making it the same as `ansible_play_hosts`).

2.3 新版功能.

`ansible_playbook_python` is the path to the python executable used to invoke the Ansible command line tool.

These vars may be useful for filling out templates with multiple hostnames or for injecting the list into the rules for a load balancer.

Also available, `inventory_dir` is the pathname of the directory holding Ansible's inventory host file, `inventory_file` is the pathname and the filename pointing to the Ansible's inventory host file.

`playbook_dir` contains the playbook base directory.

We then have `role_path` which will return the current role's pathname (since 1.8). This will only work inside a role.

And finally, `ansible_check_mode` (added in version 2.1), a boolean magic variable which will be set to `True` if you run Ansible with `--check`.

Defining variables in files

It's a great idea to keep your playbooks under source control, but you may wish to make the playbook source public while keeping certain important variables private. Similarly, sometimes you may just want to keep certain information in different files, away from the main playbook.

You can do this by using an external variables file, or files, just like this:

```
---

- hosts: all
  remote_user: root
  vars:
    favcolor: blue
  vars_files:
    - /vars/external_vars.yml

  tasks:

- name: this is just a placeholder
  command: /bin/echo foo
```

This removes the risk of sharing sensitive data with others when sharing your playbook source with them.

The contents of each variables file is a simple YAML dictionary, like this:

```
---
# in the above example, this would be vars/external_vars.yml
somevar: somevalue
password: magic
```

注解: It's also possible to keep per-host and per-group variables in very similar files, this is covered in [编排主机和组变量](#).

Passing variables on the command line

In addition to `vars_prompt` and `vars_files`, it is possible to set variables at the command line using the `--extra-vars` (or `-e`) argument. Variables can be defined using a single quoted string (containing one or more variables) using one of the formats below

key=value format:

```
ansible-playbook release.yml --extra-vars "version=1.23.45 other_variable=foo"
```

注解: Values passed in using the `key=value` syntax are interpreted as strings. Use the JSON format if you need to pass in anything that shouldn't be a string (Booleans, integers, floats, lists etc).

JSON string format:

```
ansible-playbook release.yml --extra-vars '{"version":"1.23.45","other_variable":"foo"}'
ansible-playbook arcade.yml --extra-vars '{"pacman":"mrs","ghosts":["inky","pinky","clyde
↪","sue"]}'
```

vars from a JSON or YAML file:

```
ansible-playbook release.yml --extra-vars "@some_file.json"
```

This is useful for, among other things, setting the hosts group or the user for the playbook.

Escaping quotes and other special characters:

Ensure you're escaping quotes appropriately for both your markup (e.g. JSON), and for the shell you're operating in.:

```

ansible-playbook arcade.yml --extra-vars '{"name": "Conan O'Brien"}'
ansible-playbook arcade.yml --extra-vars '{"name": "Conan O\\\'Brien"}'
ansible-playbook script.yml --extra-vars '{"dialog": "He said \\\\'I just can\'t get
↪ enough of those single and double-quotes"!\\""}'

```

In these cases, it's probably best to use a JSON or YAML file containing the variable definitions.

Variable precedence: Where should I put a variable?

A lot of folks may ask about how variables override another. Ultimately it's Ansible's philosophy that it's better you know where to put a variable, and then you have to think about it a lot less.

Avoid defining the variable “x” in 47 places and then ask the question “which x gets used”. Why? Because that's not Ansible's Zen philosophy of doing things.

There is only one Empire State Building. One Mona Lisa, etc. Figure out where to define a variable, and don't make it complicated.

However, let's go ahead and get precedence out of the way! It exists. It's a real thing, and you might have a use for it.

If multiple variables of the same name are defined in different places, they get overwritten in a certain order.

Here is the order of precedence from least to greatest (the last listed variables winning prioritization):

1. command line values (eg “-u user”)
2. role defaults (defined in role/defaults/main.yml)¹
3. inventory file or script group vars²
4. inventory group_vars/all³
5. playbook group_vars/all³
6. inventory group_vars/*³
7. playbook group_vars/*³
8. inventory file or script host vars²
9. inventory host_vars/*³
10. playbook host_vars/*³

¹ Tasks in each role will see their own role's defaults. Tasks defined outside of a role will see the last role's defaults.

² Variables defined in inventory file or provided by dynamic inventory.

³ Includes vars added by ‘vars plugins’ as well as host_vars and group_vars which are added by the default vars plugin shipped with Ansible.

11. host facts / cached set_facts⁴
12. play vars
13. play vars_prompt
14. play vars_files
15. role vars (defined in role/vars/main.yml)
16. block vars (only for tasks in block)
17. task vars (only for the task)
18. include_vars
19. set_facts / registered vars
20. role (and include_role) params
21. include params
22. extra vars (always win precedence)

Basically, anything that goes into “role defaults” (the defaults folder inside the role) is the most malleable and easily overridden. Anything in the vars directory of the role overrides previous versions of that variable in namespace. The idea here to follow is that the more explicit you get in scope, the more precedence it takes with command line `-e` extra vars always winning. Host and/or inventory variables can win over role defaults, but not explicit includes like the vars directory or an `include_vars` task.

注解: Within any section, redefining a var will overwrite the previous instance. If multiple groups have the same variable, the last one loaded wins. If you define a variable twice in a play’s `vars:` section, the second one wins.

注解: The previous describes the default config `hash_behaviour=replace`, switch to `merge` to only partially overwrite.

注解: Group loading follows parent/child relationships. Groups of the same ‘parent/child’ level are then merged following alphabetical order. This last one can be superseded by the user via `ansible_group_priority`, which defaults to 1 for all groups. This variable, `ansible_group_priority`, can only be set in the inventory source and not in `group_vars/` as the variable is used in the loading of `group_vars/`.

⁴ When created with set_facts’s cacheable option, variables will have the high precedence in the play, but will be the same as a host facts precedence when they come from the cache.

Another important thing to consider (for all versions) is that connection variables override config, command line and play/role/task specific options and keywords. See *Controlling how Ansible behaves: precedence rules* for more details. For example, if your inventory specifies `ansible_user: ramon` and you run:

```
ansible -u lola myhost
```

This will still connect as `ramon` because the value from the variable takes priority (in this case, the variable came from the inventory, but the same would be true no matter where the variable was defined).

For plays/tasks this is also true for `remote_user`. Assuming the same inventory config, the following play:

```
- hosts: myhost
  tasks:
    - command: I'll connect as ramon still
      remote_user: lola
```

will have the value of `remote_user` overwritten by `ansible_user` in the inventory.

This is done so host-specific settings can override the general settings. These variables are normally defined per host or group in inventory, but they behave like other variables.

If you want to override the remote user globally (even over inventory) you can use extra vars. For instance, if you run:

```
ansible... -e "ansible_user=maria" -u lola
```

the `lola` value is still ignored, but `ansible_user=maria` takes precedence over all other places where `ansible_user` (or `remote_user`) might be set.

A connection-specific version of a variable takes precedence over more generic versions. For example, `ansible_ssh_user` specified as a `group_var` would have a higher precedence than `ansible_user` specified as a `host_var`.

You can also override as a normal variable in a play:

```
- hosts: all
  vars:
    ansible_user: lola
  tasks:
    - command: I'll connect as lola!
```

Scoping variables

You can decide where to set a variable based on the scope you want that value to have. Ansible has three main scopes:

- Global: this is set by config, environment variables and the command line
- Play: each play and contained structures, vars entries (vars; vars_files; vars_prompt), role defaults and vars.
- Host: variables directly associated to a host, like inventory, include_vars, facts or registered task outputs

Examples of where to set a variable

Let's show some examples and where you would choose to put what based on the kind of control you might want over values.

First off, group variables are powerful.

Site-wide defaults should be defined as a `group_vars/all` setting. Group variables are generally placed alongside your inventory file. They can also be returned by a dynamic inventory script (see [动态 Inventory 清单配置](#)) or defined in things like *Red Hat Ansible Tower* from the UI or API:

```
---
# file: /etc/ansible/group_vars/all
# this is the site wide default
ntp_server: default-time.example.com
```

Regional information might be defined in a `group_vars/region` variable. If this group is a child of the `all` group (which it is, because all groups are), it will override the group that is higher up and more general:

```
---
# file: /etc/ansible/group_vars/boston
ntp_server: boston-time.example.com
```

If for some crazy reason we wanted to tell just a specific host to use a specific NTP server, it would then override the group variable!:

```
---
# file: /etc/ansible/host_vars/xyz.boston.example.com
ntp_server: override.example.com
```

So that covers inventory and what you would normally set there. It's a great place for things that deal with geography or behavior. Since groups are frequently the entity that maps roles onto hosts, it is sometimes a shortcut to set variables on the group instead of defining them on a role. You could go either way.

Remember: Child groups override parent groups, and hosts always override their groups.

Next up: learning about role variable precedence.

We'll pretty much assume you are using roles at this point. You should be using roles for sure. Roles are great. You are using roles aren't you? Hint hint.

If you are writing a redistributable role with reasonable defaults, put those in the `roles/x/defaults/main.yml` file. This means the role will bring along a default value but ANYTHING in Ansible will override it. See *Roles* for more info about this:

```
---
# file: roles/x/defaults/main.yml
# if not overridden in inventory or as a parameter, this is the value that will be used
http_port: 80
```

If you are writing a role and want to ensure the value in the role is absolutely used in that role, and is not going to be overridden by inventory, you should put it in `roles/x/vars/main.yml` like so, and inventory values cannot override it. `-e` however, still will:

```
---
# file: roles/x/vars/main.yml
# this will absolutely be used in this role
http_port: 80
```

This is one way to plug in constants about the role that are always true. If you are not sharing your role with others, app specific behaviors like ports is fine to put in here. But if you are sharing roles with others, putting variables in here might be bad. Nobody will be able to override them with inventory, but they still can by passing a parameter to the role.

Parameterized roles are useful.

If you are using a role and want to override a default, pass it as a parameter to the role like so:

```
roles:
  - role: apache
    vars:
      http_port: 8080
```

This makes it clear to the playbook reader that you've made a conscious choice to override some default in the role, or pass in some configuration that the role can't assume by itself. It also allows you to pass something site-specific that isn't really part of the role you are sharing with others.

This can often be used for things that might apply to some hosts multiple times. For example:

```
roles:
  - role: app_user
    vars:
      myname: Ian
```

(下页继续)

(续上页)

```
- role: app_user
  vars:
    myname: Terry
- role: app_user
  vars:
    myname: Graham
- role: app_user
  vars:
    myname: John
```

In this example, the same role was invoked multiple times. It's quite likely there was no default for `myname` supplied at all. Ansible can warn you when variables aren't defined – it's the default behavior in fact.

There are a few other things that go on with roles.

Generally speaking, variables set in one role are available to others. This means if you have a `roles/common/vars/main.yml` you can set variables in there and make use of them in other roles and elsewhere in your playbook:

```
roles:
  - role: common_settings
  - role: something
    vars:
      foo: 12
  - role: something_else
```

注解: There are some protections in place to avoid the need to namespace variables. In the above, variables defined in `common_settings` are most definitely available to 'something' and 'something_else' tasks, but if "something's" guaranteed to have `foo` set at 12, even if somewhere deep in common settings it set `foo` to 20.

So, that's precedence, explained in a more direct way. Don't worry about precedence, just think about if your role is defining a variable that is a default, or a "live" variable you definitely want to use. Inventory lies in precedence right in the middle, and if you want to forcibly override something, use `-e`.

If you found that a little hard to understand, take a look at the [ansible-examples](#) repo on GitHub for a bit more about how all of these things can work together.

Using advanced variable syntax

For information about advanced YAML syntax used to declare variables and have more control over the data placed in YAML files used by Ansible, see [Advanced Syntax](#).

参见:

[Intro to Playbooks](#) An introduction to playbooks

[Conditionals](#) Conditional statements in playbooks

[Filters](#) Jinja2 filters and their uses

[Loops](#) Looping in playbooks

[Roles](#) Playbook organization by roles

[Tips and tricks](#) Best practices in playbooks

[Special Variables](#) List of special variables

[User Mailing List](#) Have a question? Stop by the google group!

[irc.freenode.net](#) #ansible IRC chat channel

Templating (Jinja2)

Ansible uses Jinja2 templating to enable dynamic expressions and access to variables. Ansible includes a lot of specialized filters and tests for templating. You can use all the standard filters and tests included in Jinja2 as well. Ansible also offers a new plugin type: [Lookup Plugins](#).

All templating happens on the Ansible controller **before** the task is sent and executed on the target machine. This approach minimizes the package requirements on the target (jinja2 is only required on the controller). It also limits the amount of data Ansible passes to the target machine. Ansible parses templates on the controller and passes only the information needed for each task to the target machine, instead of passing all the data on the controller and parsing it on the target.

- [Get the current time](#)

Filters

Filters let you transform data inside template expressions. This page documents mainly Ansible-specific filters, but you can use any of the standard filters shipped with Jinja2 - see the list of [builtin filters](#) in the official Jinja2 template documentation. You can also use [Python methods](#) to manipulate variables. A few useful filters are typically added with each new Ansible release. The development documentation shows how to create custom Ansible filters as plugins, though we generally welcome new filters into the core code so everyone can use them.

Templating happens on the Ansible controller, **not** on the target host, so filters execute on the controller and manipulate data locally.

- *Handling undefined variables*
 - *Providing default values*
 - *Making variables optional*
 - *Defining mandatory values*
- *Defining different values for true/false/null*
- *Manipulating data types*
 - *Transforming dictionaries into lists*
 - *Transforming lists into dictionaries*
 - *Discovering the data type*
 - *Forcing the data type*
- *Controlling data formats: YAML and JSON*
- *Combining and selecting data*
 - *Combining items from multiple lists: zip and zip_longest*
 - *Combining objects and subelements*
 - *Combining hashes/dictionaries*
 - *Selecting values from arrays or hashtables*
 - *Combining lists*
 - * *permutations*
 - * *combinations*
 - * *products*
 - *Selecting JSON data: JSON queries*
- *Randomizing data*
 - *Random MAC addresses*
 - *Random items or numbers*
 - *Shuffling a list*
- *List filters*
- *Set theory filters*

- *Math filters*
- *Network filters*
 - *IP address filters*
 - *Network CLI filters*
 - *Network XML filters*
 - *Network VLAN filters*
- *Encryption filters*
- *Text filters*
 - *Adding comments to files*
 - *Splitting URLs*
 - *Searching strings with regular expressions*
 - *Working with filenames and pathnames*
- *String filters*
- *UUID filters*
- *Date and time filters*
- *Kubernetes filters*

Handling undefined variables

Filters can help you manage missing or undefined variables by providing defaults or making some variable optional. If you configure Ansible to ignore most undefined variables, you can mark some variables as requiring values with the `mandatory` filter.

Providing default values

You can provide default values for variables directly in your templates using the Jinja2 ‘default’ filter. This is often a better approach than failing if a variable is not defined:

```
{{ some_variable | default(5) }}
```

In the above example, if the variable ‘some_variable’ is not defined, Ansible uses the default value 5, rather than raising an “undefined variable” error and failing. If you are working within a role, you can also add a `defaults/main.yml` to define the default values for variables in your role.

Beginning in version 2.8, attempting to access an attribute of an Undefined value in Jinja will return another

Undefined value, rather than throwing an error immediately. This means that you can now simply use a default with a value in a nested data structure (i.e `{{ foo.bar.baz | default('DEFAULT') }}`) when you do not know if the intermediate values are defined.

If you want to use the default value when variables evaluate to false or an empty string you have to set the second parameter to `true`:

```
{{ lookup('env', 'MY_USER') | default('admin', true) }}
```

Making variables optional

In some cases, you want to make a variable optional. For example, if you want to use a system default for some items and control the value for others. To make a variable optional, set the default value to the special variable `omit`:

```
- name: touch files with an optional mode
  file:
    dest: "{{ item.path }}"
    state: touch
    mode: "{{ item.mode | default(omit) }}"
  loop:
    - path: /tmp/foo
    - path: /tmp/bar
    - path: /tmp/baz
      mode: "0444"
```

In this example, the default mode for the files `/tmp/foo` and `/tmp/bar` is determined by the `umask` of the system. Ansible does not send a value for `mode`. Only the third file, `/tmp/baz`, receives the `mode=0444` option.

注解: If you are “chaining” additional filters after the `default(omit)` filter, you should instead do something like this: `"{{ foo | default(None) | some_filter or omit }}"`. In this example, the default `None` (Python null) value will cause the later filters to fail, which will trigger the `or omit` portion of the logic. Using `omit` in this manner is very specific to the later filters you’re chaining though, so be prepared for some trial and error if you do this.

Defining mandatory values

If you configure Ansible to ignore undefined variables, you may want to define some values as mandatory. By default, Ansible fails if a variable in your playbook or command is undefined. You can configure Ansible to

allow undefined variables by setting `DEFAULT_UNDEFINED_VAR_BEHAVIOR` to `false`. In that case, you may want to require some variables to be defined. You can do with this with:

```
{{ variable | mandatory }}
```

The variable value will be used as is, but the template evaluation will raise an error if it is undefined.

Defining different values for true/false/null

You can create a test, then define one value to use when the test returns true and another when the test returns false (new in version 1.9):

```
{{ (name == "John") | ternary('Mr','Ms') }}
```

In addition, you can define a one value to use on true, one value on false and a third value on null (new in version 2.8):

```
{{ enabled | ternary('no shutdown', 'shutdown', omit) }}
```

Manipulating data types

Sometimes a variables file or registered variable contains a dictionary when your playbook needs a list. Sometimes you have a list when your template needs a dictionary. These filters help you transform these data types.

Transforming dictionaries into lists

2.6 新版功能.

To turn a dictionary into a list of items, suitable for looping, use *dict2items*:

```
{{ dict | dict2items }}
```

Which turns:

```
tags:
  Application: payment
  Environment: dev
```

into:

```
- key: Application
  value: payment
- key: Environment
  value: dev
```

2.8 新版功能.

`dict2items` accepts 2 keyword arguments, `key_name` and `value_name` that allow configuration of the names of the keys to use for the transformation:

```
{{ files | dict2items(key_name='file', value_name='path') }}
```

Which turns:

```
files:
  users: /etc/passwd
  groups: /etc/group
```

into:

```
- file: users
  path: /etc/passwd
- file: groups
  path: /etc/group
```

Transforming lists into dictionaries

2.7 新版功能.

This filter turns a list of dicts with 2 keys, into a dict, mapping the values of those keys into `key: value` pairs:

```
{{ tags | items2dict }}
```

Which turns:

```
tags:
- key: Application
  value: payment
- key: Environment
  value: dev
```

into:

```
Application: payment
Environment: dev
```

This is the reverse of the `dict2items` filter.

`items2dict` accepts 2 keyword arguments, `key_name` and `value_name` that allow configuration of the names of the keys to use for the transformation:

```
{{ tags | items2dict(key_name='key', value_name='value') }}
```

Discovering the data type

2.3 新版功能.

If you are unsure of the underlying Python type of a variable, you can use the `type_debug` filter to display it. This is useful in debugging when you need a particular type of variable:

```
{{ myvar | type_debug }}
```

Forcing the data type

You can cast values as certain types. For example, if you expect the input “True” from a *vars_prompt* and you want Ansible to recognize it as a Boolean value instead of a string:

```
- debug:
  msg: test
  when: some_string_value | bool
```

1.6 新版功能.

Controlling data formats: YAML and JSON

The following filters will take a data structure in a template and manipulate it or switch it from or to JSON or YAML format. These are occasionally useful for debugging:

```
{{ some_variable | to_json }}
{{ some_variable | to_yaml }}
```

For human readable output, you can use:

```
{{ some_variable | to_nice_json }}
{{ some_variable | to_nice_yaml }}
```

You can change the indentation of either format:

```
{{ some_variable | to_nice_json(indent=2) }}
{{ some_variable | to_nice_yaml(indent=8) }}
```

The `to_yaml` and `to_nice_yaml` filters use the [PyYAML library](#) which has a default 80 symbol string length limit. That causes unexpected line break after 80th symbol (if there is a space after 80th symbol) To avoid such behavior and generate long lines, use the `width` option. You must use a hardcoded number to define the width, instead of a construction like `float("inf")`, because the filter does not support proxying Python functions. For example:

```
{{ some_variable | to_yaml(indent=8, width=1337) }}
{{ some_variable | to_nice_yaml(indent=8, width=1337) }}
```

The filter does support passing through other YAML parameters. For a full list, see the [PyYAML documentation](#).

If you are reading in some already formatted data:

```
{{ some_variable | from_json }}
{{ some_variable | from_yaml }}
```

for example:

```
tasks:
  - shell: cat /some/path/to/file.json
    register: result

  - set_fact:
    myvar: "{{ result.stdout | from_json }}"
```

2.7 新版功能.

To parse multi-document YAML strings, the `from_yaml_all` filter is provided. The `from_yaml_all` filter will return a generator of parsed YAML documents.

for example:

```
tasks:
  - shell: cat /some/path/to/multidoc-file.yaml
    register: result
  - debug:
    msg: '{{ item }}'
    loop: '{{ result.stdout | from_yaml_all | list }}'
```


Combining and selecting data

These filters let you manipulate data from multiple sources and types and manage large data structures, giving you precise control over complex data.

Combining items from multiple lists: `zip` and `zip_longest`

2.3 新版功能.

To get a list combining the elements of other lists use `zip`:

```
- name: give me list combo of two lists
  debug:
    msg: "{{ [1,2,3,4,5] | zip(['a','b','c','d','e','f']) | list }}"

- name: give me shortest combo of two lists
  debug:
    msg: "{{ [1,2,3] | zip(['a','b','c','d','e','f']) | list }}"
```

To always exhaust all list use `zip_longest`:

```
- name: give me longest combo of three lists , fill with X
  debug:
    msg: "{{ [1,2,3] | zip_longest(['a','b','c','d','e','f'], [21, 22, 23], fillvalue='X
    ↪') | list }}"
```

Similarly to the output of the `items2dict` filter mentioned above, these filters can be used to construct a dict:

```
{{ dict(keys_list | zip(values_list)) }}
```

Which turns:

```
keys_list:
- one
- two
values_list:
- apple
- orange
```

into:

```
one: apple
two: orange
```

Combining objects and subelements

2.7 新版功能.

The `subelements` filter produces a product of an object and the subelement values of that object, similar to the `subelements` lookup. This lets you specify individual subelements to use in a template. For example, this expression:

```
{{ users | subelements('groups', skip_missing=True) }}
```

turns this data:

```
users:
- name: alice
  authorized:
  - /tmp/alice/onekey.pub
  - /tmp/alice/twokey.pub
  groups:
  - wheel
  - docker
- name: bob
  authorized:
  - /tmp/bob/id_rsa.pub
  groups:
  - docker
```

Into this data:

```
-
- name: alice
  groups:
  - wheel
  - docker
  authorized:
  - /tmp/alice/onekey.pub
  - /tmp/alice/twokey.pub
- wheel
-
```

(下页继续)

(续上页)

```

- name: alice
  groups:
    - wheel
    - docker
  authorized:
    - /tmp/alice/onekey.pub
    - /tmp/alice/twokey.pub
- docker
-
- name: bob
  authorized:
    - /tmp/bob/id_rsa.pub
  groups:
    - docker
- docker

```

You can use the transformed data with `loop` to iterate over the same subelement for multiple objects:

```

- name: Set authorized ssh key, extracting just that data from 'users'
  authorized_key:
    user: "{{ item.0.name }}"
    key: "{{ lookup('file', item.1) }}"
    loop: "{{ users | subelements('authorized') }}"

```

Combining hashes/dictionaries

2.0 新版功能.

The `combine` filter allows hashes to be merged. For example, the following would override keys in one hash:

```
{{ {'a':1, 'b':2} | combine({'b':3}) }}
```

The resulting hash would be:

```
{'a':1, 'b':3}
```

The filter can also take multiple arguments to merge:

```
{{ a | combine(b, c, d) }}
{{ [a, b, c, d] | combine }}
```

In this case, keys in `d` would override those in `c`, which would override those in `b`, and so on.

The filter also accepts two optional parameters: `recursive` and `list_merge`.

recursive Is a boolean, default to `False`. Should the `combine` recursively merge nested hashes. Note: It does **not** depend on the value of the `hash_behaviour` setting in `ansible.cfg`.

list_merge Is a string, its possible values are `replace` (default), `keep`, `append`, `prepend`, `append_rp` or `prepend_rp`. It modifies the behaviour of `combine` when the hashes to merge contain arrays/lists.

```
default:
  a:
    x: default
    y: default
  b: default
  c: default
patch:
  a:
    y: patch
    z: patch
  b: patch
```

If `recursive=False` (the default), nested hash aren't merged:

```
{{ default | combine(patch) }}
```

This would result in:

```
a:
  y: patch
  z: patch
b: patch
c: default
```

If `recursive=True`, recurse into nested hash and merge their keys:

```
{{ default | combine(patch, recursive=True) }}
```

This would result in:

```
a:
  x: default
  y: patch
  z: patch
b: patch
c: default
```

If `list_merge='replace'` (the default), arrays from the right hash will “replace” the ones in the left hash:

```
default:
  a:
    - default
patch:
  a:
    - patch
```

```
{{ default | combine(patch) }}
```

This would result in:

```
a:
  - patch
```

If `list_merge='keep'`, arrays from the left hash will be kept:

```
{{ default | combine(patch, list_merge='keep') }}
```

This would result in:

```
a:
  - default
```

If `list_merge='append'`, arrays from the right hash will be appended to the ones in the left hash:

```
{{ default | combine(patch, list_merge='append') }}
```

This would result in:

```
a:
  - default
  - patch
```

If `list_merge='prepend'`, arrays from the right hash will be prepended to the ones in the left hash:

```
{{ default | combine(patch, list_merge='prepend') }}
```

This would result in:

```
a:
  - patch
  - default
```

If `list_merge='append_rp'`, arrays from the right hash will be appended to the ones in the left hash. Elements of arrays in the left hash that are also in the corresponding array of the right hash will be removed (“rp” stands for “remove present”). Duplicate elements that aren’ t in both hashes are kept:

```
default:
```

```
  a:
```

```
    - 1
    - 1
    - 2
    - 3
```

```
patch:
```

```
  a:
```

```
    - 3
    - 4
    - 5
    - 5
```

```
{{ default | combine(patch, list_merge='append_rp') }}
```

This would result in:

```
a:
```

```
  - 1
  - 1
  - 2
  - 3
  - 4
  - 5
  - 5
```

If `list_merge='prepend_rp'`, the behavior is similar to the one for `append_rp`, but elements of arrays in the right hash are prepended:

```
{{ default | combine(patch, list_merge='prepend_rp') }}
```

This would result in:

```
a:
```

```
  - 3
  - 4
  - 5
  - 5
```

(下页继续)

(续上页)

```
- 1
- 1
- 2
```

recursive and list_merge can be used together:

```
default:
  a:
    a':
      x: default_value
      y: default_value
      list:
        - default_value
  b:
    - 1
    - 1
    - 2
    - 3
patch:
  a:
    a':
      y: patch_value
      z: patch_value
      list:
        - patch_value
  b:
    - 3
    - 4
    - 4
    - key: value
```

```
{{ default | combine(patch, recursive=True, list_merge='append_rp') }}
```

This would result in:

```
a:
  a':
    x: default_value
    y: patch_value
    z: patch_value
```

(下页继续)

(续上页)

```

list:
  - default_value
  - patch_value
b:
  - 1
  - 1
  - 2
  - 3
  - 4
  - 4
  - key: value

```

Selecting values from arrays or hashtables

2.1 新版功能.

The *extract* filter is used to map from a list of indices to a list of values from a container (hash or array):

```

{{ [0,2] | map('extract', ['x','y','z']) | list }}
{{ ['x','y'] | map('extract', {'x': 42, 'y': 31}) | list }}

```

The results of the above expressions would be:

```

['x', 'z']
[42, 31]

```

The filter can take another argument:

```

{{ groups['x'] | map('extract', hostvars, 'ec2_ip_address') | list }}

```

This takes the list of hosts in group 'x', looks them up in *hostvars*, and then looks up the *ec2_ip_address* of the result. The final result is a list of IP addresses for the hosts in group 'x'.

The third argument to the filter can also be a list, for a recursive lookup inside the container:

```

{{ ['a'] | map('extract', b, ['x','y']) | list }}

```

This would return a list containing the value of *b*['a']['x']['y'].

Combining lists

This set of filters returns a list of combined lists.

permutations

To get permutations of a list:

```
- name: give me largest permutations (order matters)
  debug:
    msg: "{{ [1,2,3,4,5] | permutations | list }}"

- name: give me permutations of sets of three
  debug:
    msg: "{{ [1,2,3,4,5] | permutations(3) | list }}"
```

combinations

Combinations always require a set size:

```
- name: give me combinations for sets of two
  debug:
    msg: "{{ [1,2,3,4,5] | combinations(2) | list }}"
```

Also see the *Combining items from multiple lists: zip and zip_longest*

products

The product filter returns the cartesian product of the input iterables.

This is roughly equivalent to nested for-loops in a generator expression.

For example:

```
- name: generate multiple hostnames
  debug:
    msg: "{{ ['foo', 'bar'] | product(['com']) | map('join', '.') | join(',') }}"
```

This would result in:

```
{ "msg": "foo.com,bar.com" }
```

Selecting JSON data: JSON queries

Sometimes you end up with a complex data structure in JSON format and you need to extract only a small set of data within it. The `json_query` filter lets you query a complex JSON structure and iterate over it

using a loop structure.

注解: This filter is built upon **jmespath**, and you can use the same syntax. For examples, see [jmespath examples](#).

Consider this data structure:

```
{
  "domain_definition": {
    "domain": {
      "cluster": [
        {
          "name": "cluster1"
        },
        {
          "name": "cluster2"
        }
      ],
      "server": [
        {
          "name": "server11",
          "cluster": "cluster1",
          "port": "8080"
        },
        {
          "name": "server12",
          "cluster": "cluster1",
          "port": "8090"
        },
        {
          "name": "server21",
          "cluster": "cluster2",
          "port": "9080"
        },
        {
          "name": "server22",
          "cluster": "cluster2",
          "port": "9090"
        }
      ]
    }
  },
}
```

(下页继续)

(续上页)

```

    "library": [
      {
        "name": "lib1",
        "target": "cluster1"
      },
      {
        "name": "lib2",
        "target": "cluster2"
      }
    ]
  }
}

```

To extract all clusters from this structure, you can use the following query:

```

- name: "Display all cluster names"
  debug:
    var: item
  loop: "{{ domain_definition | json_query('domain.cluster[*].name') }}"

```

Same thing for all server names:

```

- name: "Display all server names"
  debug:
    var: item
  loop: "{{ domain_definition | json_query('domain.server[*].name') }}"

```

This example shows ports from cluster1:

```

- name: "Display all ports from cluster1"
  debug:
    var: item
  loop: "{{ domain_definition | json_query(server_name_cluster1_query) }}"
  vars:
    server_name_cluster1_query: "domain.server[?cluster=='cluster1'].port"

```

注解: You can use a variable to make the query more readable.

Or, alternatively print out the ports in a comma separated string:

```
- name: "Display all ports from cluster1 as a string"
  debug:
    msg: "{{ domain_definition | json_query('domain.server[?cluster==`cluster1`].port') |
    ↪ join(', ') }}"
```

注解: Here, quoting literals using backticks avoids escaping quotes and maintains readability.

Or, using YAML single quote escaping:

```
- name: "Display all ports from cluster1"
  debug:
    var: item
    loop: "{{ domain_definition | json_query('domain.server[?cluster=='cluster1'].port') |
    ↪ }}"
```

注解: Escaping single quotes within single quotes in YAML is done by doubling the single quote.

In this example, we get a hash map with all ports and names of a cluster:

```
- name: "Display all server ports and names from cluster1"
  debug:
    var: item
    loop: "{{ domain_definition | json_query(server_name_cluster1_query) }}"
  vars:
    server_name_cluster1_query: "domain.server[?cluster=='cluster2'].{name: name, port:
    ↪ port}"
```

Randomizing data

When you need a randomly generated value, use one of these filters.

Random MAC addresses

2.6 新版功能.

This filter can be used to generate a random MAC address from a string prefix.

To get a random MAC address from a string prefix starting with '52:54:00' :

```
"{{ '52:54:00' | random_mac }}"
# => '52:54:00:ef:1c:03'
```

Note that if anything is wrong with the prefix string, the filter will issue an error.

2.9 新版功能.

As of Ansible version 2.9, you can also initialize the random number generator from a seed. This way, you can create random-but-idempotent MAC addresses:

```
"{{ '52:54:00' | random_mac(seed=inventory_hostname) }}"
```

Random items or numbers

This filter can be used similar to the default Jinja2 random filter (returning a random item from a sequence of items), but can also generate a random number based on a range.

To get a random item from a list:

```
"{{ ['a','b','c'] | random }}"
# => 'c'
```

To get a random number between 0 and a specified number:

```
"{{ 60 | random }} * * * * root /script/from/cron"
# => '21 * * * * root /script/from/cron'
```

Get a random number from 0 to 100 but in steps of 10:

```
{{ 101 | random(step=10) }}
# => 70
```

Get a random number from 1 to 100 but in steps of 10:

```
{{ 101 | random(1, 10) }}
# => 31
{{ 101 | random(start=1, step=10) }}
# => 51
```

It's also possible to initialize the random number generator from a seed. This way, you can create random-but-idempotent numbers:

```
"{{ 60 | random(seed=inventory_hostname) }} * * * * root /script/from/cron"
```

Shuffling a list

This filter will randomize an existing list, giving a different order every invocation.

To get a random list from an existing list:

```
{{ ['a','b','c'] | shuffle }}  
# => ['c','a','b']  
{{ ['a','b','c'] | shuffle }}  
# => ['b','c','a']
```

It's also possible to shuffle a list idempotent. All you need is a seed.:

```
{{ ['a','b','c'] | shuffle(seed=inventory_hostname) }}  
# => ['b','a','c']
```

The shuffle filter returns a list whenever possible. If you use it with a non 'listable' item, the filter does nothing.

List filters

These filters all operate on list variables.

To get the minimum value from list of numbers:

```
{{ list1 | min }}
```

To get the maximum value from a list of numbers:

```
{{ [3, 4, 2] | max }}
```

2.5 新版功能.

Flatten a list (same thing the *flatten* lookup does):

```
{{ [3, [4, 2] ] | flatten }}
```

Flatten only the first level of a list (akin to the *items* lookup):

```
{{ [3, [4, [2]] ] | flatten(levels=1) }}
```

Set theory filters

These functions return a unique set from sets or lists.

1.4 新版功能.

To get a unique set from a list:

```
{{ list1 | unique }}
```

To get a union of two lists:

```
{{ list1 | union(list2) }}
```

To get the intersection of 2 lists (unique list of all items in both):

```
{{ list1 | intersect(list2) }}
```

To get the difference of 2 lists (items in 1 that don't exist in 2):

```
{{ list1 | difference(list2) }}
```

To get the symmetric difference of 2 lists (items exclusive to each list):

```
{{ list1 | symmetric_difference(list2) }}
```

Math filters

1.9 新版功能.

Get the logarithm (default is e):

```
{{ myvar | log }}
```

Get the base 10 logarithm:

```
{{ myvar | log(10) }}
```

Give me the power of 2! (or 5):

```
{{ myvar | pow(2) }}
```

```
{{ myvar | pow(5) }}
```

Square root, or the 5th:

```
{{ myvar | root }}
```

```
{{ myvar | root(5) }}
```

Note that jinja2 already provides some like `abs()` and `round()`.

Network filters

These filters help you with common network tasks.

IP address filters

1.9 新版功能.

To test if a string is a valid IP address:

```
{{ myvar | ipaddr }}
```

You can also require a specific IP protocol version:

```
{{ myvar | ipv4 }}
{{ myvar | ipv6 }}
```

IP address filter can also be used to extract specific information from an IP address. For example, to get the IP address itself from a CIDR, you can use:

```
{{ '192.0.2.1/24' | ipaddr('address') }}
```

More information about `ipaddr` filter and complete usage guide can be found in `playbooks_filters_ipaddr`.

Network CLI filters

2.4 新版功能.

To convert the output of a network device CLI command into structured JSON output, use the `parse_cli` filter:

```
{{ output | parse_cli('path/to/spec') }}
```

The `parse_cli` filter will load the spec file and pass the command output through it, returning JSON output. The YAML spec file defines how to parse the CLI output.

The spec file should be valid formatted YAML. It defines how to parse the CLI output and return JSON data. Below is an example of a valid spec file that will parse the output from the `show vlan` command.

```
---
vars:
  vlan:
    vlan_id: "{{ item.vlan_id }}"
    name: "{{ item.name }}"
```

(下页继续)

(续上页)

```

    enabled: "{{{ item.state != 'act/lshut' }}}"
    state: "{{{ item.state }}}"

keys:
  vlans:
    value: "{{{ vlan }}}"
    items: "~(?P<vlan_id>\\d+)\\s+(?P<name>\\w+)\\s+(?P<state>active|act/lshut|suspended)
↪"
  state_static:
    value: present

```

The spec file above will return a JSON data structure that is a list of hashes with the parsed VLAN information.

The same command could be parsed into a hash by using the key and values directives. Here is an example of how to parse the output into a hash value using the same `show vlan` command.

```

---
vars:
  vlan:
    key: "{{{ item.vlan_id }}}"
    values:
      vlan_id: "{{{ item.vlan_id }}}"
      name: "{{{ item.name }}}"
      enabled: "{{{ item.state != 'act/lshut' }}}"
      state: "{{{ item.state }}}"

keys:
  vlans:
    value: "{{{ vlan }}}"
    items: "~(?P<vlan_id>\\d+)\\s+(?P<name>\\w+)\\s+(?P<state>active|act/lshut|suspended)
↪"
  state_static:
    value: present

```

Another common use case for parsing CLI commands is to break a large command into blocks that can be parsed. This can be done using the `start_block` and `end_block` directives to break the command into blocks that can be parsed.

```

---
vars:

```

(下页继续)

(续上页)

```

interface:
  name: "{{ item[0].match[0] }}"
  state: "{{ item[1].state }}"
  mode: "{{ item[2].match[0] }}"

keys:
  interfaces:
    value: "{{ interface }}"
    start_block: "^Ethernet.*$"
    end_block: "^$"
    items:
      - "(?P<name>Ethernet\\d\\d\\d*)"
      - "admin state is (?P<state>.+),"
      - "Port mode is (.+)"

```

The example above will parse the output of `show interface` into a list of hashes.

The network filters also support parsing the output of a CLI command using the TextFSM library. To parse the CLI output with TextFSM use the following filter:

```
{{ output.stdout[0] | parse_cli_textfsm('path/to/fsm') }}
```

Use of the TextFSM filter requires the TextFSM library to be installed.

Network XML filters

2.5 新版功能.

To convert the XML output of a network device command into structured JSON output, use the `parse_xml` filter:

```
{{ output | parse_xml('path/to/spec') }}
```

The `parse_xml` filter will load the spec file and pass the command output through formatted as JSON.

The spec file should be valid formatted YAML. It defines how to parse the XML output and return JSON data.

Below is an example of a valid spec file that will parse the output from the `show vlan | display xml` command.

```

---
vars:

```

(下页继续)

(续上页)

```

vlan:
  vlan_id: "{{ item.vlan_id }}"
  name: "{{ item.name }}"
  desc: "{{ item.desc }}"
  enabled: "{{ item.state.get('inactive') != 'inactive' }}"
  state: "% if item.state.get('inactive') == 'inactive' % inactive {% else %} active {
↪ % endif %}"

keys:
  vlans:
    value: "{{ vlan }}"
    top: configuration/vlans/vlan
    items:
      vlan_id: vlan-id
      name: name
      desc: description
      state: ".[@inactive='inactive']"

```

The spec file above will return a JSON data structure that is a list of hashes with the parsed VLAN information.

The same command could be parsed into a hash by using the key and values directives. Here is an example of how to parse the output into a hash value using the same `show vlan | display xml` command.

```

---
vars:
  vlan:
    key: "{{ item.vlan_id }}"
    values:
      vlan_id: "{{ item.vlan_id }}"
      name: "{{ item.name }}"
      desc: "{{ item.desc }}"
      enabled: "{{ item.state.get('inactive') != 'inactive' }}"
      state: "% if item.state.get('inactive') == 'inactive' % inactive {% else %}
↪ active {% endif %}"

keys:
  vlans:
    value: "{{ vlan }}"
    top: configuration/vlans/vlan
    items:

```

(下页继续)

(续上页)

```

vlan_id: vlan-id
name: name
desc: description
state: ".[@inactive='inactive']"

```

The value of `top` is the XPath relative to the XML root node. In the example XML output given below, the value of `top` is `configuration/vlans/vlan`, which is an XPath expression relative to the root node (`<rpc-reply>`). `configuration` in the value of `top` is the outer most container node, and `vlan` is the inner-most container node.

`items` is a dictionary of key-value pairs that map user-defined names to XPath expressions that select elements. The XPath expression is relative to the value of the XPath value contained in `top`. For example, the `vlan_id` in the spec file is a user defined name and its value `vlan-id` is the relative to the value of XPath in `top`

Attributes of XML tags can be extracted using XPath expressions. The value of `state` in the spec is an XPath expression used to get the attributes of the `vlan` tag in output XML.:

```

<rpc-reply>
  <configuration>
    <vlans>
      <vlan inactive="inactive">
        <name>vlan-1</name>
        <vlan-id>200</vlan-id>
        <description>This is vlan-1</description>
      </vlan>
    </vlans>
  </configuration>
</rpc-reply>

```

注解: For more information on supported XPath expressions, see <https://docs.python.org/2/library/xml.etree.elementtree.html#xpath-support>.

Network VLAN filters

2.8 新版功能.

Use the `vlan_parser` filter to manipulate an unsorted list of VLAN integers into a sorted string list of integers according to IOS-like VLAN list rules. This list has the following properties:

- Vlans are listed in ascending order.

- Three or more consecutive VLANs are listed with a dash.
- The first line of the list can be first_line_len characters long.
- Subsequent list lines can be other_line_len characters.

To sort a VLAN list:

```
{{ [3003, 3004, 3005, 100, 1688, 3002, 3999] | vlan_parser }}
```

This example renders the following sorted list:

```
['100,1688,3002-3005,3999']
```

Another example Jinja template:

```
{% set parsed_vlans = vlans | vlan_parser %}
switchport trunk allowed vlan {{ parsed_vlans[0] }}
{% for i in range (1, parsed_vlans | count) %}
switchport trunk allowed vlan add {{ parsed_vlans[i] }}
```

This allows for dynamic generation of VLAN lists on a Cisco IOS tagged interface. You can store an exhaustive raw list of the exact VLANs required for an interface and then compare that to the parsed IOS output that would actually be generated for the configuration.

Encryption filters

1.9 新版功能.

To get the sha1 hash of a string:

```
{{ 'test1' | hash('sha1') }}
```

To get the md5 hash of a string:

```
{{ 'test1' | hash('md5') }}
```

Get a string checksum:

```
{{ 'test2' | checksum }}
```

Other hashes (platform dependent):

```
{{ 'test2' | hash('blowfish') }}
```

To get a sha512 password hash (random salt):

```
{{ 'passwordsaresecret' | password_hash('sha512') }}
```

To get a sha256 password hash with a specific salt:

```
{{ 'secretpassword' | password_hash('sha256', 'mysecretsalt') }}
```

An idempotent method to generate unique hashes per system is to use a salt that is consistent between runs:

```
{{ 'secretpassword' | password_hash('sha512', 65534 | random(seed=inventory_hostname) |  
↪string) }}
```

Hash types available depend on the master system running ansible, 'hash' depends on hashlib password_hash depends on passlib (<https://passlib.readthedocs.io/en/stable/lib/passlib.hash.html>).

2.7 新版功能.

Some hash types allow providing a rounds parameter:

```
{{ 'secretpassword' | password_hash('sha256', 'mysecretsalt', rounds=10000) }}
```

Text filters

These filters work with strings and text.

Adding comments to files

The *comment* filter lets you turn text in a template into comments in a file, with a variety of comment styles. By default Ansible uses # to start a comment line and adds a blank comment line above and below your comment text. For example the following:

```
{{ "Plain style (default)" | comment }}
```

produces this output:

```
#  
# Plain style (default)  
#
```

Ansible offers styles for comments in C (//...), C block (/*...*/), Erlang (%...) and XML (<!--...-->):

```
{{ "C style" | comment('c') }}  
{{ "C block style" | comment('cblock') }}
```

(下页继续)

(续上页)

```
{{ "Erlang style" | comment('erlang') }}
{{ "XML style" | comment('xml') }}
```

You can define a custom comment character. This filter:

```
{{ "My Special Case" | comment(decoration="! ") }}
```

produces:

```
!
! My Special Case
!
```

You can fully customize the comment style:

```
{{ "Custom style" | comment('plain', prefix='#####\n', postfix='#\n#####\n   ###\n ↪
↪ #') }}
```

That creates the following output:

```
#####
#
# Custom style
#
#####
    ###
    #
```

The filter can also be applied to any Ansible variable. For example to make the output of the `ansible_managed` variable more readable, we can change the definition in the `ansible.cfg` file to this:

```
[defaults]

ansible_managed = This file is managed by Ansible.%n
    template: {file}
    date: %Y-%m-%d %H:%M:%S
    user: {uid}
    host: {host}
```

and then use the variable with the `comment` filter:

```
{{ ansible_managed | comment }}
```

which produces this output:

```
#
# This file is managed by Ansible.
#
# template: /home/ansible/env/dev/ansible_managed/roles/role1/templates/test.j2
# date: 2015-09-10 11:02:58
# user: ansible
# host: myhost
#
```

Splitting URLs

2.4 新版功能.

The `urlsplit` filter extracts the fragment, hostname, netloc, password, path, port, query, scheme, and username from an URL. With no arguments, returns a dictionary of all the fields:

```
{{ "http://user:password@www.acme.com:9000/dir/index.html?query=term#fragment" |
↳urlsplit('hostname') }}
# => 'www.acme.com'

{{ "http://user:password@www.acme.com:9000/dir/index.html?query=term#fragment" |
↳urlsplit('netloc') }}
# => 'user:password@www.acme.com:9000'

{{ "http://user:password@www.acme.com:9000/dir/index.html?query=term#fragment" |
↳urlsplit('username') }}
# => 'user'

{{ "http://user:password@www.acme.com:9000/dir/index.html?query=term#fragment" |
↳urlsplit('password') }}
# => 'password'

{{ "http://user:password@www.acme.com:9000/dir/index.html?query=term#fragment" |
↳urlsplit('path') }}
# => '/dir/index.html'

{{ "http://user:password@www.acme.com:9000/dir/index.html?query=term#fragment" |
↳urlsplit('port') }}
# => '9000'
```

(下页继续)

(续上页)

```

{{ "http://user:password@www.acme.com:9000/dir/index.html?query=term#fragment" |
↪urlsplit('scheme') }}
# => 'http'

{{ "http://user:password@www.acme.com:9000/dir/index.html?query=term#fragment" |
↪urlsplit('query') }}
# => 'query=term'

{{ "http://user:password@www.acme.com:9000/dir/index.html?query=term#fragment" |
↪urlsplit('fragment') }}
# => 'fragment'

{{ "http://user:password@www.acme.com:9000/dir/index.html?query=term#fragment" |
↪urlsplit }}
# =>
# {
#     "fragment": "fragment",
#     "hostname": "www.acme.com",
#     "netloc": "user:password@www.acme.com:9000",
#     "password": "password",
#     "path": "/dir/index.html",
#     "port": 9000,
#     "query": "query=term",
#     "scheme": "http",
#     "username": "user"
# }

```

Searching strings with regular expressions

To search a string with a regex, use the “`regex_search`” filter:

```

# search for "foo" in "foobar"
{{ 'foobar' | regex_search('(foo)') }}

# will return empty if it cannot find a match
{{ 'ansible' | regex_search('(foobar)') }}

# case insensitive search in multiline mode

```

(下页继续)

(续上页)

```
{{ 'foo\nBAR' | regex_search("^bar", multiline=True, ignorecase=True) }}
```

To search for all occurrences of regex matches, use the “`regex_findall`” filter:

```
# Return a list of all IPv4 addresses in the string
{{ 'Some DNS servers are 8.8.8.8 and 8.8.4.4' | regex_findall('\\b(?:[0-9]{1,3}\\\\.){3}[0-9]{1,3}\\b') }}
```

To replace text in a string with regex, use the “`regex_replace`” filter:

```
# convert "ansible" to "able"
{{ 'ansible' | regex_replace('^a.*i(.*)$', 'a\\1') }}

# convert "foobar" to "bar"
{{ 'foobar' | regex_replace('^f.*o(.*)$', '\\1') }}

# convert "localhost:80" to "localhost, 80" using named groups
{{ 'localhost:80' | regex_replace('^(?P<host>.+):(P<port>\\d+)$', '\\g<host>, \\g<port>') }}

# convert "localhost:80" to "localhost"
{{ 'localhost:80' | regex_replace(':80') }}

# change a multiline string
{{ var | regex_replace('^', '#CommentThis#', multiline=True) }}
```

注解: If you want to match the whole string and you are using `*` make sure to always wraparound your regular expression with the start/end anchors. For example `^(.*)$` will always match only one result, while `(.*)` on some Python versions will match the whole string and an empty string at the end, which means it will make two replacements:

```
# add "https://" prefix to each item in a list
GOOD:
{{ hosts | map('regex_replace', '^(.*)$', 'https://\\1') | list }}
{{ hosts | map('regex_replace', '(.)+', 'https://\\1') | list }}
{{ hosts | map('regex_replace', '^', 'https://') | list }}

BAD:
{{ hosts | map('regex_replace', '(.*)', 'https://\\1') | list }}
```

(下页继续)

(续上页)

```
# append ':80' to each item in a list
GOOD:
{{ hosts | map('regex_replace', '^(.*)$', '\\1:80') | list }}
{{ hosts | map('regex_replace', '(\\.+)', '\\1:80') | list }}
{{ hosts | map('regex_replace', '$', ':80') | list }}

BAD:
{{ hosts | map('regex_replace', '(.*)', '\\1:80') | list }}
```

注解: Prior to ansible 2.0, if “regex_replace” filter was used with variables inside YAML arguments (as opposed to simpler ‘key=value’ arguments), then you needed to escape backreferences (e.g. \\1) with 4 backslashes (\\\\) instead of 2 (\\).

2.0 新版功能.

To escape special characters within a standard Python regex, use the “regex_escape” filter (using the default re_type=’ python’ option):

```
# convert '~f.*o(.*)$' to '~f\\..*o(\\..*|)\\$'
{{ '~f.*o(.*)$' | regex_escape() }}
```

2.8 新版功能.

To escape special characters within a POSIX basic regex, use the “regex_escape” filter with the re_type=’ posix_basic’ option:

```
# convert '~f.*o(.*)$' to '~f\\..*o(\\..*|)\\$'
{{ '~f.*o(.*)$' | regex_escape('posix_basic') }}
```

Working with filenames and pathnames

To get the last name of a file path, like ‘foo.txt’ out of ‘/etc/asdf/foo.txt’ :

```
{{ path | basename }}
```

To get the last name of a windows style file path (new in version 2.0):

```
{{ path | win_basename }}
```

To separate the windows drive letter from the rest of a file path (new in version 2.0):

```
{{ path | win_splitdrive }}
```

To get only the windows drive letter:

```
{{ path | win_splitdrive | first }}
```

To get the rest of the path without the drive letter:

```
{{ path | win_splitdrive | last }}
```

To get the directory from a path:

```
{{ path | dirname }}
```

To get the directory from a windows path (new version 2.0):

```
{{ path | win_dirname }}
```

To expand a path containing a tilde (~) character (new in version 1.5):

```
{{ path | expanduser }}
```

To expand a path containing environment variables:

```
{{ path | expandvars }}
```

注解: *expandvars* expands local variables; using it on remote paths can lead to errors.

2.6 新版功能.

To get the real path of a link (new in version 1.8):

```
{{ path | realpath }}
```

To get the relative path of a link, from a start point (new in version 1.7):

```
{{ path | relpath('/etc') }}
```

To get the root and extension of a path or filename (new in version 2.0):

```
# with path == 'nginx.conf' the return would be ('nginx', '.conf')
{{ path | splitext }}
```

To join one or more path components:

```
{{ ('/etc', path, 'subdir', file) | path_join }}
```

2.10 新版功能.

String filters

To add quotes for shell usage:

```
- shell: echo {{ string_value | quote }}
```

To concatenate a list into a string:

```
{{ list | join(" ") }}
```

To work with Base64 encoded strings:

```
{{ encoded | b64decode }}
{{ decoded | string | b64encode }}
```

As of version 2.6, you can define the type of encoding to use, the default is `utf-8`:

```
{{ encoded | b64decode(encoding='utf-16-le') }}
{{ decoded | string | b64encode(encoding='utf-16-le') }}
```

注解: The `string` filter is only required for Python 2 and ensures that text to encode is a unicode string. Without that filter before `b64encode` the wrong value will be encoded.

2.6 新版功能.

UUID filters

To create a namespaced UUIDv5:

```
{{ string | to_uuid(namespace='11111111-2222-3333-4444-555555555555') }}
```

2.10 新版功能.

To create a namespaced UUIDv5 using the default Ansible namespace ‘361E6D51-FAEC-444A-9079-341386DA8E2E’ :

```
{{ string | to_uuid }}
```

1.9 新版功能.

To make use of one attribute from each item in a list of complex variables, use the Jinja2 `map` filter:

```
# get a comma-separated list of the mount points (e.g. "/,/mnt/stuff") on a host
{{ ansible_mounts | map(attribute='mount') | join(',') }}
```

Date and time filters

To get a date object from a string use the `to_datetime` filter:

```
# Get total amount of seconds between two dates. Default date format is %Y-%m-%d %H:%M:
↳ %S but you can pass your own format
{{ ((("2016-08-14 20:00:12" | to_datetime) - ("2015-12-25" | to_datetime('%Y-%m-%d'))).
↳ total_seconds() }}
```

```
# Get remaining seconds after delta has been calculated. NOTE: This does NOT convert
↳ years, days, hours, etc to seconds. For that, use total_seconds()
{{ ((("2016-08-14 20:00:12" | to_datetime) - ("2016-08-14 18:00:00" | to_datetime)).
↳ seconds )}
```

```
# This expression evaluates to "12" and not "132". Delta is 2 hours, 12 seconds
```

```
# get amount of days between two dates. This returns only number of days and discards
↳ remaining hours, minutes, and seconds
{{ ((("2016-08-14 20:00:12" | to_datetime) - ("2015-12-25" | to_datetime('%Y-%m-%d'))).
↳ days )}
```

2.4 新版功能.

To format a date using a string (like with the shell `date` command), use the “`strftime`” filter:

```
# Display year-month-day
{{ '%Y-%m-%d' | strftime }}
```

```
# Display hour:min:sec
{{ '%H:%M:%S' | strftime }}
```

```
# Use ansible_date_time.epoch fact
{{ '%Y-%m-%d %H:%M:%S' | strftime(ansible_date_time.epoch) }}
```

```
# Use arbitrary epoch value
{{ '%Y-%m-%d' | strftime(0) }}          # => 1970-01-01
{{ '%Y-%m-%d' | strftime(1441357287) }} # => 2015-09-04
```

注解: To get all string possibilities, check <https://docs.python.org/2/library/time.html#time.strftime>

Kubernetes filters

Use the “k8s_config_resource_name” filter to obtain the name of a Kubernetes ConfigMap or Secret, including its hash:

```
{{ configmap_resource_definition | k8s_config_resource_name }}
```

This can then be used to reference hashes in Pod specifications:

```
my_secret:
  kind: Secret
  name: my_secret_name

deployment_resource:
  kind: Deployment
  spec:
    template:
      spec:
        containers:
        - envFrom:
          - secretRef:
              name: {{ my_secret | k8s_config_resource_name }}
```

2.8 新版功能.

参见:

Intro to Playbooks An introduction to playbooks

Conditionals Conditional statements in playbooks

Using Variables All about variables

Loops Looping in playbooks

Roles Playbook organization by roles

Tips and tricks Best practices in playbooks

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

Tests

Tests in Jinja are a way of evaluating template expressions and returning True or False. Jinja ships with many of these. See [builtin tests](#) in the official Jinja template documentation.

The main difference between tests and filters are that Jinja tests are used for comparisons, whereas filters are used for data manipulation, and have different applications in jinja. Tests can also be used in list processing filters, like `map()` and `select()` to choose items in the list.

Like all templating, tests always execute on the Ansible controller, **not** on the target of a task, as they test local data.

In addition to those Jinja2 tests, Ansible supplies a few more and users can easily create their own.

- *Test syntax*
- *Testing strings*
- *Testing truthiness*
- *Comparing versions*
- *Set theory tests*
- *Testing if a list contains a value*
- *Testing if a list value is True*
- *Testing paths*
- *Testing size formats*
 - *Human readable*
 - *Human to bytes*
- *Testing task results*

Test syntax

Test syntax varies from filter syntax (`variable | filter`). Historically Ansible has registered tests as both jinja tests and jinja filters, allowing for them to be referenced using filter syntax.

As of Ansible 2.5, using a jinja test as a filter will generate a warning.

The syntax for using a jinja test is as follows:

```
variable is test_name
```

Such as:


```
result is failed
```

Testing strings

To match strings against a substring or a regular expression, use the `match`, `search` or `regex` filters:

```
vars:
  url: "http://example.com/users/foo/resources/bar"

tasks:
  - debug:
      msg: "matched pattern 1"
      when: url is match("http://example.com/users/*/resources/")

  - debug:
      msg: "matched pattern 2"
      when: url is search("/users/*/resources/.*")

  - debug:
      msg: "matched pattern 3"
      when: url is search("/users/")

  - debug:
      msg: "matched pattern 4"
      when: url is regex("example.com/\w+/foo")
```

`match` succeeds if it finds the pattern at the beginning of the string, while `search` succeeds if it finds the pattern anywhere within string. By default, `regex` works like `search`, but `regex` can be configured to perform other tests as well.

Testing truthiness

2.10 新版功能.

As of Ansible 2.10, you can now perform Python like `truthy` and `falsy` checks.

```
- debug:
    msg: "Truthy"
    when: value is truthy
    vars:
        value: "some string"
```

(下页继续)

(续上页)

```
- debug:
    msg: "Falsy"
  when: value is falsy
  vars:
    value: ""
```

Additionally, the `truthy` and `falsy` tests accept an optional parameter called `convert_bool` that will attempt to convert boolean indicators to actual booleans.

```
- debug:
    msg: "Truthy"
  when: value is truthy(convert_bool=True)
  vars:
    value: "yes"

- debug:
    msg: "Falsy"
  when: value is falsy(convert_bool=True)
  vars:
    value: "off"
```

Comparing versions

1.6 新版功能.

注解: In 2.5 `version_compare` was renamed to `version`

To compare a version number, such as checking if the `ansible_facts['distribution_version']` version is greater than or equal to '12.04', you can use the `version` test.

The `version` test can also be used to evaluate the `ansible_facts['distribution_version']`:

```
{{ ansible_facts['distribution_version'] is version('12.04', '>=') }}
```

If `ansible_facts['distribution_version']` is greater than or equal to 12.04, this test returns `True`, otherwise `False`.

The `version` test accepts the following operators:

```
<, lt, <=, le, >, gt, >=, ge, ==, =, eq, !=, <>, ne
```

This test also accepts a 3rd parameter, `strict` which defines if strict version parsing as defined by `distutils.version.StrictVersion` should be used. The default is `False` (using `distutils.version.LooseVersion`), `True` enables strict version parsing:

```
{{ sample_version_var is version('1.0', operator='lt', strict=True) }}
```

When using `version` in a playbook or role, don't use `{{ }}` as described in the [FAQ](#):

```
vars:
    my_version: 1.2.3

tasks:
    - debug:
        msg: "my_version is higher than 1.0.0"
        when: my_version is version('1.0.0', '>')
```

Set theory tests

2.1 新版功能.

注解: In 2.5 `issubset` and `issuperset` were renamed to `subset` and `superset`

To see if a list includes or is included by another list, you can use ‘subset’ and ‘superset’ :

```
vars:
    a: [1,2,3,4,5]
    b: [2,3]
tasks:
    - debug:
        msg: "A includes B"
        when: a is superset(b)

    - debug:
        msg: "B is included in A"
        when: b is subset(a)
```

Testing if a list contains a value

2.8 新版功能.

Ansible includes a `contains` test which operates similarly, but in reverse of the Jinja2 provided `in` test. The `contains` test is designed to work with the `select`, `reject`, `selectattr`, and `rejectattr` filters:

```
vars:
  lacp_groups:
    - master: lacp0
      network: 10.65.100.0/24
      gateway: 10.65.100.1
      dns4:
        - 10.65.100.10
        - 10.65.100.11
      interfaces:
        - em1
        - em2

    - master: lacp1
      network: 10.65.120.0/24
      gateway: 10.65.120.1
      dns4:
        - 10.65.100.10
        - 10.65.100.11
      interfaces:
        - em3
        - em4

tasks:
  - debug:
      msg: "{{ (lacp_groups|selectattr('interfaces', 'contains', 'em1')|first).master }}"
```

2.4 新版功能.

Testing if a list value is True

You can use *any* and *all* to check if any or all elements in a list are true or not:

```
vars:
  mylist:
    - 1
```

(下页继续)

(续上页)

```
- "{{ 3 == 3 }}"
- True
myotherlist:
- False
- True
tasks:

- debug:
    msg: "all are true!"
    when: mylist is all

- debug:
    msg: "at least one is true"
    when: myotherlist is any
```

Testing paths

注解: In 2.5 the following tests were renamed to remove the `is_` prefix

The following tests can provide information about a path on the controller:

```
- debug:
    msg: "path is a directory"
    when: mypath is directory

- debug:
    msg: "path is a file"
    when: mypath is file

- debug:
    msg: "path is a symlink"
    when: mypath is link

- debug:
    msg: "path already exists"
    when: mypath is exists

- debug:
```

(下页继续)

(续上页)

```

    msg: "path is {{ (mypath is abs)|ternary('absolute','relative')}}"

- debug:
    msg: "path is the same file as path2"
  when: mypath is same_file(path2)

- debug:
    msg: "path is a mount"
  when: mypath is mount

```

Testing size formats

The `human_readable` and `human_to_bytes` functions let you test your playbooks to make sure you are using the right size format in your tasks, and that you provide Byte format to computers and human-readable format to people.

Human readable

Asserts whether the given string is human readable or not.

For example:

```

- name: "Human Readable"
  assert:
    that:
      - '"1.00 Bytes" == 1|human_readable'
      - '"1.00 bits" == 1|human_readable(isbits=True)'
      - '"10.00 KB" == 10240|human_readable'
      - '"97.66 MB" == 102400000|human_readable'
      - '"0.10 GB" == 102400000|human_readable(unit="G")'
      - '"0.10 Gb" == 102400000|human_readable(isbits=True, unit="G")'

```

This would result in:

```
{ "changed": false, "msg": "All assertions passed" }
```

Human to bytes

Returns the given string in the Bytes format.

For example:

```
- name: "Human to Bytes"
  assert:
    that:
      - "{{'0'|human_to_bytes}}" == 0"
      - "{{'0.1'|human_to_bytes}}" == 0"
      - "{{'0.9'|human_to_bytes}}" == 1"
      - "{{'1'|human_to_bytes}}" == 1"
      - "{{'10.00 KB'|human_to_bytes}}" == 10240"
      - "{{'11 MB'|human_to_bytes}}" == 11534336"
      - "{{'1.1 GB'|human_to_bytes}}" == 1181116006"
      - "{{'10.00 Kb'|human_to_bytes(isbits=True)}}" == 10240"
```

This would result in:

```
{ "changed": false, "msg": "All assertions passed" }
```

Testing task results

The following tasks are illustrative of the tests meant to check the status of tasks:

```
tasks:

- shell: /usr/bin/foo
  register: result
  ignore_errors: True

- debug:
    msg: "it failed"
  when: result is failed

# in most cases you'll want a handler, but if you want to do something right now, this
↪ is nice

- debug:
    msg: "it changed"
  when: result is changed

- debug:
    msg: "it succeeded in Ansible >= 2.1"
  when: result is succeeded
```

(下页继续)

(续上页)

```
- debug:
    msg: "it succeeded"
    when: result is success

- debug:
    msg: "it was skipped"
    when: result is skipped
```

注解: From 2.1, you can also use success, failure, change, and skip so that the grammar matches, for those who need to be strict about it.

参见:

Intro to Playbooks An introduction to playbooks

Conditionals Conditional statements in playbooks

Using Variables All about variables

Loops Looping in playbooks

Roles Playbook organization by roles

Tips and tricks Best practices in playbooks

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

Lookups

Lookup plugins allow access to outside data sources. Like all templating, these plugins are evaluated on the Ansible control machine, and can include reading the filesystem as well as contacting external datastores and services. This data is then made available using the standard templating system in Ansible.

注解:

- Lookups occur on the local computer, not on the remote computer.
- They are executed within the directory containing the role or play, as opposed to local tasks which are executed with the directory of the executed script.
- You can pass wantlist=True to lookups to use in jinja2 template “for” loops.
- Lookups are an advanced feature. You should have a good working knowledge of Ansible plays before incorporating them.

警告: Some lookups pass arguments to a shell. When using variables from a remote/untrusted source, use the `/quote` filter to ensure safe usage.

Topics

- *Lookups*
 - *Lookups and loops*
 - *Lookups and variables*

Lookups and loops

lookup plugins are a way to query external data sources, such as shell commands or even key value stores.

Before Ansible 2.5, lookups were mostly used indirectly in `with_<lookup>` constructs for looping. Starting with Ansible version 2.5, lookups are used more explicitly as part of Jinja2 expressions fed into the `loop` keyword.

Lookups and variables

One way of using lookups is to populate variables. These macros are evaluated each time they are used in a task (or template):

```
vars:
  motd_value: "{{ lookup('file', '/etc/motd') }}"
tasks:
  - debug:
      msg: "motd value is {{ motd_value }}"
```

For more details and a complete list of lookup plugins available, please see *Working With Plugins*.

参见:

Working With Playbooks An introduction to playbooks

Conditionals Conditional statements in playbooks

Using Variables All about variables

Loops Looping in playbooks

User Mailing List Have a question? Stop by the google group!

[irc.freenode.net](#) #ansible IRC chat channel

Python3 in templates

Ansible uses Jinja2 to leverage Python data types and standard functions in templates and variables. You can use these data types and standard functions to perform a rich set of operations on your data. However, if you use templates, you must be aware of differences between Python versions.

These topics help you design templates that work on both Python2 and Python3. They might also help if you are upgrading from Python2 to Python3. Upgrading within Python2 or Python3 does not usually introduce changes that affect Jinja2 templates.

Dictionary views

In Python2, the `dict.keys()`, `dict.values()`, and `dict.items()` methods return a list. Jinja2 returns that to Ansible via a string representation that Ansible can turn back into a list.

In Python3, those methods return a [dictionary view](#) object. The string representation that Jinja2 returns for dictionary views cannot be parsed back into a list by Ansible. It is, however, easy to make this portable by using the `list` filter whenever using `dict.keys()`, `dict.values()`, or `dict.items()`:

```
vars:
  hosts:
    testhost1: 127.0.0.2
    testhost2: 127.0.0.3
tasks:
  - debug:
      msg: '{{ item }}'
      # Only works with Python 2
      #loop: "{{ hosts.keys() }}"
      # Works with both Python 2 and Python 3
      loop: "{{ hosts.keys() | list }}"
```

`dict.iteritems()`

Python2 dictionaries have `iterkeys()`, `itervalues()`, and `iteritems()` methods.

Python3 dictionaries do not have these methods. Use `dict.keys()`, `dict.values()`, and `dict.items()` to make your playbooks and templates compatible with both Python2 and Python3:

```
vars:
  hosts:
```

(下页继续)

(续上页)

```

testhost1: 127.0.0.2
testhost2: 127.0.0.3
tasks:
- debug:
    msg: '{{ item }}'
    # Only works with Python 2
    #loop: "{{ hosts.iteritems() }}"
    # Works with both Python 2 and Python 3
    loop: "{{ hosts.items() | list }}"

```

参见:

- The *Dictionary views* entry for information on why the `list` filter is necessary here.

Get the current time

2.8 新版功能.

The `now()` Jinja2 function retrieves a Python datetime object or a string representation for the current time.

The `now()` function supports 2 arguments:

utc Specify `True` to get the current time in UTC. Defaults to `False`.

fmt Accepts a `strftime` string that returns a formatted date time string.

参见:

Intro to Playbooks An introduction to playbooks

Conditionals Conditional statements in playbooks

Loops Looping in playbooks

Roles Playbook organization by roles

Tips and tricks Best practices in playbooks

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

Conditionals**Topics**

- *Conditionals*

- *The When Statement*
- *Loops and Conditionals*
- *Loading in Custom Facts*
- *Applying ‘when’ to roles, imports, and includes*
- *Conditional Imports*
- *Selecting Files And Templates Based On Variables*
- *Register Variables*
- *Commonly Used Facts*
 - * `ansible_facts[‘distribution’]`
 - * `ansible_facts[‘distribution_major_version’]`
 - * `ansible_facts[‘os_family’]`

Often the result of a play may depend on the value of a variable, fact (something learned about the remote system), or previous task result. In some cases, the values of variables may depend on other variables. Additional groups can be created to manage hosts based on whether the hosts match other criteria. This topic covers how conditionals are used in playbooks.

注解: There are many options to control execution flow in Ansible. More examples of supported conditionals can be located here: <https://jinja.palletsprojects.com/en/master/templates/#comparisons>.

The When Statement

Sometimes you will want to skip a particular step on a particular host. This could be something as simple as not installing a certain package if the operating system is a particular version, or it could be something like performing some cleanup steps if a filesystem is getting full.

This is easy to do in Ansible with the **when** clause, which contains a raw Jinja2 expression without double curly braces (see `group_by_module`).

注解: Jinja2 expressions are built up from comparisons, filters, tests, and logical combinations thereof. The below examples will give you an impression how to use them. However, for a more complete overview over all operators to use, please refer to the official [Jinja2 documentation](#).

It’s actually pretty simple:

```
tasks:
- name: "shut down Debian flavored systems"
  command: /sbin/shutdown -t now
  when: ansible_facts['os_family'] == "Debian"
  # note that all variables can be used directly in conditionals without double curly
  ↪braces
```

You can also use parentheses to group and logical operators to combine conditions:

```
tasks:
- name: "shut down CentOS 6 and Debian 7 systems"
  command: /sbin/shutdown -t now
  when: (ansible_facts['distribution'] == "CentOS" and ansible_facts['distribution_
  ↪major_version'] == "6") or
        (ansible_facts['distribution'] == "Debian" and ansible_facts['distribution_
  ↪major_version'] == "7")
```

Multiple conditions that all need to be true (that is, a logical **and**) can also be specified as a list:

```
tasks:
- name: "shut down CentOS 6 systems"
  command: /sbin/shutdown -t now
  when:
    - ansible_facts['distribution'] == "CentOS"
    - ansible_facts['distribution_major_version'] == "6"
```

A number of Jinja2 “tests” and “filters” can also be used in when statements, some of which are unique and provided by Ansible. Suppose we want to ignore the error of one statement and then decide to do something conditionally based on success or failure:

```
tasks:
- command: /bin/false
  register: result
  ignore_errors: True

- command: /bin/something
  when: result is failed

# Both `succeeded` and `success` both work. The former, however, is newer and uses the
  ↪correct tense, while the latter is mainly used in older versions of Ansible.

- command: /bin/something_else
  when: result is succeeded
```

(下页继续)

(续上页)

```
- command: /bin/still/something_else
  when: result is skipped
```

注解: both *success* and *succeeded* work (*similarly for fail/failed, etc*).

警告: You might expect a variable of a skipped task to be undefined and use *defined* or *default* to check that. **This is incorrect!** Even when a task is failed or skipped the variable is still registered with a failed or skipped status. See *Registering variables*.

To see what facts are available on a particular system, you can do the following in a playbook:

```
- debug: var=ansible_facts
```

Tip: Sometimes you'll get back a variable that's a string and you'll want to do a math operation comparison on it. You can do this like so:

```
tasks:
- shell: echo "only on Red Hat 6, derivatives, and later"
  when: ansible_facts['os_family'] == "RedHat" and ansible_facts['lsb']['major_release']|int >= 6
```

注解: the above example requires the `lsb_release` package on the target host in order to return the `lsb major_release` fact.

Variables defined in the playbooks or inventory can also be used, just make sure to apply the `|bool` filter to non-boolean variables (e.g., *string* variables with content like `yes`, `on`, `1`, `true`). An example may be the execution of a task based on a variable's boolean value:

```
vars:
  epic: true
  monumental: "yes"
```

Then a conditional execution might look like:

```
tasks:
- shell: echo "This certainly is epic!"
```

(下页继续)

(续上页)

```
when: epic or monumental|bool
```

or:

```
tasks:
  - shell: echo "This certainly isn't epic!"
    when: not epic
```

If a required variable has not been set, you can skip or fail using Jinja2's `defined` test. For example:

```
tasks:
  - shell: echo "I've got '{{ foo }}' and am not afraid to use it!"
    when: foo is defined

  - fail: msg="Bailing out. this play requires 'bar'"
    when: bar is undefined
```

This is especially useful in combination with the conditional import of vars files (see below). As the examples show, you don't need to use `{{ }}` to use variables inside conditionals, as these are already implied.

Loops and Conditionals

Combining `when` with loops (see *Loops*), be aware that the `when` statement is processed separately for each item. This is by design:

```
tasks:
  - command: echo {{ item }}
    loop: [ 0, 2, 4, 6, 8, 10 ]
    when: item > 5
```

If you need to skip the whole task depending on the loop variable being defined, used the `|default` filter to provide an empty iterator:

```
- command: echo {{ item }}
  loop: "{{ mylist|default([]) }}"
  when: item > 5
```

If using a dict in a loop:

```
- command: echo {{ item.key }}
  loop: "{{ query('dict', mydict|default({})) }}"
  when: item.value > 5
```

Loading in Custom Facts

It's also easy to provide your own facts if you want, which is covered in *Should you develop a module?*. To run them, just make a call to your own custom fact gathering module at the top of your list of tasks, and variables returned there will be accessible to future tasks:

```
tasks:
  - name: gather site specific fact data
    action: site_facts
  - command: /usr/bin/thingy
    when: my_custom_fact_just_retrieved_from_the_remote_system == '1234'
```

Applying 'when' to roles, imports, and includes

Note that if you have several tasks that all share the same conditional statement, you can affix the conditional to a task include statement as below. All the tasks get evaluated, but the conditional is applied to each and every task:

```
- import_tasks: tasks/sometasks.yml
  when: "'reticulating splines' in output"
```

注解: In versions prior to 2.0 this worked with task includes but not playbook includes. 2.0 allows it to work with both.

Or with a role:

```
- hosts: webservers
  roles:
    - role: debian_stock_config
      when: ansible_facts['os_family'] == 'Debian'
```

You will note a lot of **skipped** output by default in Ansible when using this approach on systems that don't match the criteria. In many cases the `group_by` module can be a more streamlined way to accomplish the same thing; see *Handling OS and distro differences*.

When a conditional is used with `include_*` tasks instead of imports, it is applied *only* to the include task itself and not to any other tasks within the included file(s). A common situation where this distinction is important is as follows:

```
# We wish to include a file to define a variable when it is not
# already defined
```

(下页继续)

(续上页)

```
# main.yml
- import_tasks: other_tasks.yml # note "import"
  when: x is not defined

# other_tasks.yml
- set_fact:
    x: foo
- debug:
    var: x
```

This expands at include time to the equivalent of:

```
- set_fact:
    x: foo
  when: x is not defined
- debug:
    var: x
  when: x is not defined
```

Thus if `x` is initially undefined, the `debug` task will be skipped. By using `include_tasks` instead of `import_tasks`, both tasks from `other_tasks.yml` will be executed as expected.

For more information on the differences between `include` v `import` see [Re-using Ansible artifacts](#).

Conditional Imports

注解: This is an advanced topic that is infrequently used.

Sometimes you will want to do certain things differently in a playbook based on certain criteria. Having one playbook that works on multiple platforms and OS versions is a good example.

As an example, the name of the Apache package may be different between CentOS and Debian, but it is easily handled with a minimum of syntax in an Ansible Playbook:

```
---
- hosts: all
  remote_user: root
  vars_files:
    - "vars/common.yml"
```

(下页继续)

(续上页)

```
- [ "vars/{{ ansible_facts['os_family'] }}.yaml", "vars/os_defaults.yaml" ]
tasks:
- name: make sure apache is started
  service: name={{ apache }} state=started
```

注解： The variable “ansible_facts[‘os_family’]” is being interpolated into the list of filenames being defined for vars_files.

As a reminder, the various YAML files contain just keys and values:

```
---
# for vars/RedHat.yaml
apache: httpd
somethingelse: 42
```

How does this work? For Red Hat operating systems (‘CentOS’ , for example), the first file Ansible tries to import is ‘vars/RedHat.yaml’ . If that file does not exist, Ansible attempts to load ‘vars/os_defaults.yaml’ . If no files in the list were found, an error is raised.

On Debian, Ansible first looks for ‘vars/Debian.yaml’ instead of ‘vars/RedHat.yaml’ , before falling back on ‘vars/os_defaults.yaml’ .

Ansible’ s approach to configuration – separating variables from tasks, keeping your playbooks from turning into arbitrary code with nested conditionals - results in more streamlined and auditable configuration rules because there are fewer decision points to track.

Selecting Files And Templates Based On Variables

注解： This is an advanced topic that is infrequently used. You can probably skip this section.

Sometimes a configuration file you want to copy, or a template you will use may depend on a variable. The following construct selects the first available file appropriate for the variables of a given host, which is often much cleaner than putting a lot of if conditionals in a template.

The following example shows how to template out a configuration file that was very different between, say, CentOS and Debian:

```
- name: template a file
  template:
```

(下页继续)

(续上页)

```

    src: "{{ item }}"
    dest: /etc/myapp/foo.conf
loop: "{{ query('first_found', { 'files': myfiles, 'paths': mypaths}) }}"
vars:
  myfiles:
    - "{{ansible_facts['distribution']}}.conf"
    - default.conf
  mypaths: ['search_location_one/somedir/', '/opt/other_location/somedir/']

```

Register Variables

Often in a playbook it may be useful to store the result of a given command in a variable and access it later. Use of the command module in this way can in many ways eliminate the need to write site specific facts, for instance, you could test for the existence of a particular program.

注解: Registration happens even when a task is skipped due to the conditional. This way you can query the variable for “is skipped” to know if task was attempted or not.

The **register** keyword decides what variable to save a result in. The resulting variables can be used in templates, action lines, or *when* statements. It looks like this (in an obviously trivial example):

```

- name: test play
  hosts: all

  tasks:
    - shell: cat /etc/motd
      register: motd_contents

    - shell: echo "motd contains the word hi"
      when: motd_contents.stdout.find('hi') != -1

```

As shown previously, the registered variable's string contents are accessible with the **stdout** value. The registered result can be used in the loop of a task if it is converted into a list (or already is a list) as shown below. **stdout_lines** is already available on the object as well though you could also call **home_dirs.stdout.split()** if you wanted, and could split by other fields:

```

- name: registered variable usage as a loop list
  hosts: all
  tasks:

```

(下页继续)

(续上页)

```
- name: retrieve the list of home directories
  command: ls /home
  register: home_dirs

- name: add home dirs to the backup spooler
  file:
    path: /mnt/bkspool/{{ item }}
    src: /home/{{ item }}
    state: link
  loop: "{{ home_dirs.stdout_lines }}"
  # same as loop: "{{ home_dirs.stdout.split() }}"
```

As shown previously, the registered variable's string contents are accessible with the `stdout` value. You may check the registered variable's string contents for emptiness:

```
- name: check registered variable for emptiness
  hosts: all

  tasks:

    - name: list contents of directory
      command: ls mydir
      register: contents

    - name: check contents for emptiness
      debug:
        msg: "Directory is empty"
      when: contents.stdout == ""
```

Commonly Used Facts

The following Facts are frequently used in Conditionals - see above for examples.

`ansible_facts['distribution']`

Possible values (sample, not complete list):

```
Alpine
Altlinux
Amazon
Archlinux
ClearLinux
Coreos
CentOS
Debian
Fedora
Gentoo
Mandriva
NA
OpenWrt
OracleLinux
RedHat
Slackware
SMGL
SUSE
Ubuntu
VMwareESX
```

ansible_facts['distribution_major_version']

This will be the major version of the operating system. For example, the value will be *16* for Ubuntu 16.04.

ansible_facts['os_family']

Possible values (sample, not complete list):

```
AIX
Alpine
Altlinux
Archlinux
Darwin
Debian
FreeBSD
Gentoo
HP-UX
Mandrake
RedHat
```

(下页继续)

(续上页)

SGML

Slackware

Solaris

Suse

Windows

参见:

Working With Playbooks An introduction to playbooks

Roles Playbook organization by roles

Tips and tricks Best practices in playbooks

Using Variables All about variables

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

Loops

Sometimes you want to repeat a task multiple times. In computer programming, this is called a loop. Common Ansible loops include changing ownership on several files and/or directories with the file module, creating multiple users with the user module, and repeating a polling step until a certain result is reached. Ansible offers two keywords for creating loops: `loop` and `with_<lookup>`.

注解:

- We added `loop` in Ansible 2.5. It is not yet a full replacement for `with_<lookup>`, but we recommend it for most use cases.
 - We have not deprecated the use of `with_<lookup>` - that syntax will still be valid for the foreseeable future.
 - We are looking to improve `loop` syntax - watch this page and the [changelog](#) for updates.
-

- *Comparing `loop` and `with_*`*
 - *Standard loops*
 - *Iterating over a simple list*
 - *Iterating over a list of hashes*
 - *Iterating over a dictionary*

- *Registering variables with a loop*
- *Complex loops*
 - *Iterating over nested lists*
 - *Retrying a task until a condition is met*
 - *Looping over inventory*
- *Ensuring list input for loop: query vs. lookup*
- *Adding controls to loops*
 - *Limiting loop output with label*
 - *Pausing within a loop*
 - *Tracking progress through a loop with index_var*
 - *Defining inner and outer variable names with loop_var*
 - *Extended loop variables*
 - *Accessing the name of your loop_var*
- *Migrating from with_X to loop*
 - *with_list*
 - *with_items*
 - *with_indexed_items*
 - *with_flattened*
 - *with_together*
 - *with_dict*
 - *with_sequence*
 - *with_subelements*
 - *with_nested/with_cartesian*
 - *with_random_choice*

Comparing loop and with_*

- The with_<lookup> keywords rely on *Lookup Plugins* - even items is a lookup.
- The loop keyword is equivalent to with_list, and is the best choice for simple loops.
- The loop keyword will not accept a string as input, see *Ensuring list input for loop: query vs. lookup*.

- Generally speaking, any use of `with_*` covered in *Migrating from with_X to loop* can be updated to use `loop`.
- Be careful when changing `with_items` to `loop`, as `with_items` performed implicit single-level flattening. You may need to use `flatten(1)` with `loop` to match the exact outcome. For example, to get the same output as:

```
with_items:
- 1
- [2,3]
- 4
```

you would need:

```
loop: "{{ [1, [2,3], 4] | flatten(1) }}"
```

- Any `with_*` statement that requires using `lookup` within a loop should not be converted to use the `loop` keyword. For example, instead of doing:

```
loop: "{{ lookup('fileglob', '*.txt', wantlist=True) }}"
```

it's cleaner to keep:

```
with_fileglob: '*.txt'
```

Standard loops

Iterating over a simple list

Repeated tasks can be written as standard loops over a simple list of strings. You can define the list directly in the task:

```
- name: add several users
  user:
    name: "{{ item }}"
    state: present
    groups: "wheel"
  loop:
    - testuser1
    - testuser2
```

You can define the list in a variables file, or in the `'vars'` section of your play, then refer to the name of the list in the task:


```
loop: "{{ somelist }}"
```

Either of these examples would be the equivalent of:

```
- name: add user testuser1
  user:
    name: "testuser1"
    state: present
    groups: "wheel"

- name: add user testuser2
  user:
    name: "testuser2"
    state: present
    groups: "wheel"
```

You can pass a list directly to a parameter for some plugins. Most of the packaging modules, like yum and apt, have this capability. When available, passing the list to a parameter is better than looping over the task. For example:

```
- name: optimal yum
  yum:
    name: "{{ list_of_packages }}"
    state: present

- name: non-optimal yum, slower and may cause issues with interdependencies
  yum:
    name: "{{ item }}"
    state: present
  loop: "{{ list_of_packages }}"
```

Check the module documentation to see if you can pass a list to any particular module's parameter(s).

Iterating over a list of hashes

If you have a list of hashes, you can reference subkeys in a loop. For example:

```
- name: add several users
  user:
    name: "{{ item.name }}"
    state: present
```

(下页继续)

(续上页)

```

groups: "{{ item.groups }}"
loop:
  - { name: 'testuser1', groups: 'wheel' }
  - { name: 'testuser2', groups: 'root' }

```

When combining *conditionals* with a loop, the `when:` statement is processed separately for each item. See *The When Statement* for examples.

Iterating over a dictionary

To loop over a dict, use the *dict2items*:

```

- name: Using dict2items
  debug:
    msg: "{{ item.key }} - {{ item.value }}"
  loop: "{{ tag_data | dict2items }}"
  vars:
    tag_data:
      Environment: dev
      Application: payment

```

Here, we are iterating over *tag_data* and printing the key and the value from it.

Registering variables with a loop

You can register the output of a loop as a variable. For example:

```

- shell: "echo {{ item }}"
  loop:
    - "one"
    - "two"
  register: echo

```

When you use `register` with a loop, the data structure placed in the variable will contain a **results** attribute that is a list of all responses from the module. This differs from the data structure returned when using `register` without a loop:

```

{
  "changed": true,
  "msg": "All items completed",

```

(下页继续)

(续上页)

```

"results": [
  {
    "changed": true,
    "cmd": "echo \"one\" ",
    "delta": "0:00:00.003110",
    "end": "2013-12-19 12:00:05.187153",
    "invocation": {
      "module_args": "echo \"one\"",
      "module_name": "shell"
    },
    "item": "one",
    "rc": 0,
    "start": "2013-12-19 12:00:05.184043",
    "stderr": "",
    "stdout": "one"
  },
  {
    "changed": true,
    "cmd": "echo \"two\" ",
    "delta": "0:00:00.002920",
    "end": "2013-12-19 12:00:05.245502",
    "invocation": {
      "module_args": "echo \"two\"",
      "module_name": "shell"
    },
    "item": "two",
    "rc": 0,
    "start": "2013-12-19 12:00:05.242582",
    "stderr": "",
    "stdout": "two"
  }
]
}

```

Subsequent loops over the registered variable to inspect the results may look like:

```

- name: Fail if return code is not 0
  fail:
    msg: "The command ({{ item.cmd }}) did not have a 0 return code"
  when: item.rc != 0

```

(下页继续)

(续上页)

```
loop: "{{ echo.results }}"
```

During iteration, the result of the current item will be placed in the variable:

```
- shell: echo "{{ item }}"
loop:
  - one
  - two
register: echo
changed_when: echo.stdout != "one"
```

Complex loops

Iterating over nested lists

You can use Jinja2 expressions to iterate over complex lists. For example, a loop can combine nested lists:

```
- name: give users access to multiple databases
mysql_user:
  name: "{{ item[0] }}"
  priv: "{{ item[1] }}.*:ALL"
  append_privs: yes
  password: "foo"
loop: "{{ ['alice', 'bob'] | product(['clientdb', 'employee', 'providerdb']) | list }}"
```

Retrying a task until a condition is met

1.4 新版功能.

You can use the `until` keyword to retry a task until a certain condition is met. Here's an example:

```
- shell: /usr/bin/foo
register: result
until: result.stdout.find("all systems go") != -1
retries: 5
delay: 10
```

This task runs up to 5 times with a delay of 10 seconds between each attempt. If the result of any attempt has “all systems go” in its stdout, the task succeeds. The default value for “retries” is 3 and “delay” is 5.

To see the results of individual retries, run the play with `-vv`.

When you run a task with `until` and register the result as a variable, the registered variable will include a key called “`attempts`”, which records the number of the retries for the task.

注解: You must set the `until` parameter if you want a task to retry. If `until` is not defined, the value for the `retries` parameter is forced to 1.

Looping over inventory

To loop over your inventory, or just a subset of it, you can use a regular loop with the `ansible_play_batch` or `groups` variables:

```
# show all the hosts in the inventory
- debug:
    msg: "{{ item }}"
    loop: "{{ groups['all'] }}"

# show all the hosts in the current play
- debug:
    msg: "{{ item }}"
    loop: "{{ ansible_play_batch }}"
```

There is also a specific lookup plugin `inventory_hostnames` that can be used like this:

```
# show all the hosts in the inventory
- debug:
    msg: "{{ item }}"
    loop: "{{ query('inventory_hostnames', 'all') }}"

# show all the hosts matching the pattern, ie all but the group www
- debug:
    msg: "{{ item }}"
    loop: "{{ query('inventory_hostnames', 'all:!www') }}"
```

More information on the patterns can be found in *Pattern: 正则匹配主机和组*.

Ensuring list input for loop: query vs. lookup

The `loop` keyword requires a list as input, but the `lookup` keyword returns a string of comma-separated values by default. Ansible 2.5 introduced a new Jinja2 function named `query` that always returns a list, offering a simpler interface and more predictable output from lookup plugins when using the `loop` keyword.

You can force lookup to return a list to loop by using `wantlist=True`, or you can use `query` instead.

These examples do the same thing:

```
loop: "{{ query('inventory_hostnames', 'all') }}"

loop: "{{ lookup('inventory_hostnames', 'all', wantlist=True) }}"
```

Adding controls to loops

2.1 新版功能.

The `loop_control` keyword lets you manage your loops in useful ways.

Limiting loop output with label

2.2 新版功能.

When looping over complex data structures, the console output of your task can be enormous. To limit the displayed output, use the `label` directive with `loop_control`:

```
- name: create servers
  digital_ocean:
    name: "{{ item.name }}"
    state: present
  loop:
    - name: server1
      disks: 3gb
      ram: 15Gb
      network:
        nic01: 100Gb
        nic02: 10Gb
        ...
  loop_control:
    label: "{{ item.name }}"
```

The output of this task will display just the `name` field for each `item` instead of the entire contents of the multi-line `{{ item }}` variable.

注解: This is for making console output more readable, not protecting sensitive data. If there is sensitive data in loop, set `no_log: yes` on the task to prevent disclosure.

Pausing within a loop

2.2 新版功能.

To control the time (in seconds) between the execution of each item in a task loop, use the `pause` directive with `loop_control`:

```
# main.yml
- name: create servers, pause 3s before creating next
  digital_ocean:
    name: "{{ item }}"
    state: present
  loop:
    - server1
    - server2
  loop_control:
    pause: 3
```

Tracking progress through a loop with `index_var`

2.5 新版功能.

To keep track of where you are in a loop, use the `index_var` directive with `loop_control`. This directive specifies a variable name to contain the current loop index:

```
- name: count our fruit
  debug:
    msg: "{{ item }} with index {{ my_idx }}"
  loop:
    - apple
    - banana
    - pear
  loop_control:
    index_var: my_idx
```

Defining inner and outer variable names with `loop_var`

2.1 新版功能.

You can nest two looping tasks using `include_tasks`. However, by default Ansible sets the loop variable `item` for each loop. This means the inner, nested loop will overwrite the value of `item` from the outer loop. You can specify the name of the variable for each loop using `loop_var` with `loop_control`:

```
# main.yml
- include_tasks: inner.yml
  loop:
    - 1
    - 2
    - 3
  loop_control:
    loop_var: outer_item

# inner.yml
- debug:
    msg: "outer item={{ outer_item }} inner item={{ item }}"
  loop:
    - a
    - b
    - c
```

注解: If Ansible detects that the current loop is using a variable which has already been defined, it will raise an error to fail the task.

Extended loop variables

2.8 新版功能.

As of Ansible 2.8 you can get extended loop information using the **extended** option to loop control. This option will expose the following information.

Variable	Description
<code>ansible_loop.allitems</code>	The list of all items in the loop
<code>ansible_loop.index</code>	The current iteration of the loop. (1 indexed)
<code>ansible_loop.index0</code>	The current iteration of the loop. (0 indexed)
<code>ansible_loop.revindex</code>	The number of iterations from the end of the loop (1 indexed)
<code>ansible_loop.revindex0</code>	The number of iterations from the end of the loop (0 indexed)
<code>ansible_loop.first</code>	True if first iteration
<code>ansible_loop.last</code>	True if last iteration
<code>ansible_loop.length</code>	The number of items in the loop
<code>ansible_loop.previtem</code>	The item from the previous iteration of the loop. Undefined during the first iteration.
<code>ansible_loop.nextitem</code>	The item from the following iteration of the loop. Undefined during the last iteration.

```
loop_control:
  extended: yes
```

Accessing the name of your loop_var

2.8 新版功能.

As of Ansible 2.8 you can get the name of the value provided to `loop_control.loop_var` using the `ansible_loop_var` variable

For role authors, writing roles that allow loops, instead of dictating the required `loop_var` value, you can gather the value via:

```
"{{ lookup('vars', ansible_loop_var) }}"
```

Migrating from with_X to loop

With the release of Ansible 2.5, the recommended way to perform loops is the use the new `loop` keyword instead of `with_X` style loops.

In many cases, `loop` syntax is better expressed using filters instead of more complex use of `query` or `lookup`.

The following examples will show how to convert many common `with_` style loops to `loop` and filters.

with_list

with_list is directly replaced by loop.

```
- name: with_list
  debug:
    msg: "{{ item }}"
  with_list:
    - one
    - two

- name: with_list -> loop
  debug:
    msg: "{{ item }}"
  loop:
    - one
    - two
```

with_items

with_items is replaced by loop and the flatten filter.

```
- name: with_items
  debug:
    msg: "{{ item }}"
  with_items: "{{ items }}"

- name: with_items -> loop
  debug:
    msg: "{{ item }}"
  loop: "{{ items|flatten(levels=1) }}"
```

with_indexed_items

with_indexed_items is replaced by loop, the flatten filter and loop_control.index_var.

```
- name: with_indexed_items
  debug:
    msg: "{{ item.0 }} - {{ item.1 }}"
  with_indexed_items: "{{ items }}"
```

(下页继续)

(续上页)

```
- name: with_indexed_items -> loop
  debug:
    msg: "{{ index }}" - "{{ item }}"
  loop: "{{ items|flatten(levels=1) }}"
  loop_control:
    index_var: index
```

with_flattened

with_flattened is replaced by loop and the flatten filter.

```
- name: with_flattened
  debug:
    msg: "{{ item }}"
  with_flattened: "{{ items }}"

- name: with_flattened -> loop
  debug:
    msg: "{{ item }}"
  loop: "{{ items|flatten }}"
```

with_together

with_together is replaced by loop and the zip filter.

```
- name: with_together
  debug:
    msg: "{{ item.0 }}" - "{{ item.1 }}"
  with_together:
    - "{{ list_one }}"
    - "{{ list_two }}"

- name: with_together -> loop
  debug:
    msg: "{{ item.0 }}" - "{{ item.1 }}"
  loop: "{{ list_one|zip(list_two)|list }}"
```

with_dict

with_dict can be substituted by loop and either the dictsort or dict2items filters.

```
- name: with_dict
  debug:
    msg: "{{ item.key }} - {{ item.value }}"
  with_dict: "{{ dictionary }}"

- name: with_dict -> loop (option 1)
  debug:
    msg: "{{ item.key }} - {{ item.value }}"
  loop: "{{ dictionary|dict2items }}"

- name: with_dict -> loop (option 2)
  debug:
    msg: "{{ item.0 }} - {{ item.1 }}"
  loop: "{{ dictionary|dictsort }}"
```

with_sequence

with_sequence is replaced by loop and the range function, and potentially the format filter.

```
- name: with_sequence
  debug:
    msg: "{{ item }}"
  with_sequence: start=0 end=4 stride=2 format=testuser%02x

- name: with_sequence -> loop
  debug:
    msg: "{{ 'testuser%02x' | format(item) }}"
    # range is exclusive of the end point
  loop: "{{ range(0, 4 + 1, 2)|list }}"
```

with_subelements

with_subelements is replaced by loop and the subelements filter.

```
- name: with_subelements
  debug:
```

(下页继续)

(续上页)

```

    msg: "{{ item.0.name }} - {{ item.1 }}"
  with_subelements:
    - "{{ users }}"
    - mysql.hosts

- name: with_subelements -> loop
  debug:
    msg: "{{ item.0.name }} - {{ item.1 }}"
  loop: "{{ users|subelements('mysql.hosts') }}"

```

with_nested/with_cartesian

with_nested and with_cartesian are replaced by loop and the product filter.

```

- name: with_nested
  debug:
    msg: "{{ item.0 }} - {{ item.1 }}"
  with_nested:
    - "{{ list_one }}"
    - "{{ list_two }}"

- name: with_nested -> loop
  debug:
    msg: "{{ item.0 }} - {{ item.1 }}"
  loop: "{{ list_one|product(list_two)|list }}"

```

with_random_choice

with_random_choice is replaced by just use of the random filter, without need of loop.

```

- name: with_random_choice
  debug:
    msg: "{{ item }}"
  with_random_choice: "{{ my_list }}"

- name: with_random_choice -> loop (No loop is needed here)
  debug:
    msg: "{{ my_list|random }}"
  tags: random

```

参见:

Intro to Playbooks An introduction to playbooks

Roles Playbook organization by roles

Tips and tricks Best practices in playbooks

Conditionals Conditional statements in playbooks

Using Variables All about variables

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

Blocks

Blocks create logical groups of tasks. Blocks also offer ways to handle task errors, similar to exception handling in many programming languages.

- *Grouping tasks with blocks*
- *Handling errors with blocks*

Grouping tasks with blocks

All tasks in a block inherit directives applied at the block level. Most of what you can apply to a single task (with the exception of loops) can be applied at the block level, so blocks make it much easier to set data or directives common to the tasks. The directive does not affect the block itself, it is only inherited by the tasks enclosed by a block. For example, a *when* statement is applied to the tasks within a block, not to the block itself.

列表 1: Block example with named tasks inside the block

```
tasks:
  - name: Install, configure, and start Apache
    block:
      - name: install httpd and memcached
        yum:
          name:
            - httpd
            - memcached
          state: present
      - name: apply the foo config template
```

(下页继续)

(续上页)

```

template:
  src: templates/src.j2
  dest: /etc/foo.conf
- name: start service bar and enable it
  service:
    name: bar
    state: started
    enabled: True
when: ansible_facts['distribution'] == 'CentOS'
become: true
become_user: root
ignore_errors: yes

```

In the example above, the ‘when’ condition will be evaluated before Ansible runs each of the three tasks in the block. All three tasks also inherit the privilege escalation directives, running as the root user. Finally, `ignore_errors: yes` ensures that Ansible continues to execute the playbook even if some of the tasks fail.

Names for tasks within blocks have been available since Ansible 2.3. We recommend using names in all tasks, within blocks or elsewhere, for better visibility into the tasks being executed when you run the playbook.

Handling errors with blocks

You can control how Ansible responds to task errors using blocks with `rescue` and `always` sections.

Rescue blocks specify tasks to run when an earlier task in a block fails. This approach is similar to exception handling in many programming languages. Ansible only runs rescue blocks after a task returns a ‘failed’ state. Bad task definitions and unreachable hosts will not trigger the rescue block.

列表 2: Block error handling example

```

tasks:
- name: Handle the error
  block:
    - debug:
        msg: 'I execute normally'
    - name: i force a failure
      command: /bin/false
    - debug:
        msg: 'I never execute, due to the above task failing, :-( '
  rescue:
    - debug:
        msg: 'I caught an error, can do stuff here to fix it, :-)'

```

You can also add an `always` section to a block. Tasks in the `always` section run no matter what the task status of the previous block is.

列表 3: Block with always section

```
- name: Always do X
  block:
    - debug:
        msg: 'I execute normally'
    - name: i force a failure
      command: /bin/false
    - debug:
        msg: 'I never execute :-( '
  always:
    - debug:
        msg: "This always executes, :-)"
```

Together, these elements offer complex error handling.

列表 4: Block with all sections

```
- name: Attempt and graceful roll back demo
  block:
    - debug:
        msg: 'I execute normally'
    - name: i force a failure
      command: /bin/false
    - debug:
        msg: 'I never execute, due to the above task failing, :-( '
  rescue:
    - debug:
        msg: 'I caught an error'
    - name: i force a failure in middle of recovery! >:-)
      command: /bin/false
    - debug:
        msg: 'I also never execute :-( '
  always:
    - debug:
        msg: "This always executes"
```

The tasks in the `block` execute normally. If any tasks in the block return `failed`, the `rescue` section executes tasks to recover from the error. The `always` section runs regardless of the results of the `block` and `rescue` sections.

If an error occurs in the block and the rescue task succeeds, Ansible reverts the failed status of the original task for the run and continues to run the play as if the original task had succeeded. The rescued task is considered successful, and does not trigger `max_fail_percentage` or `any_errors_fatal` configurations. However, Ansible still reports a failure in the playbook statistics.

You can use blocks with `flush_handlers` in a rescue task to ensure that all handlers run even if an error occurs:

列表 5: Block run handlers in error handling

```
tasks:
  - name: Attempt and graceful roll back demo
    block:
      - debug:
          msg: 'I execute normally'
        changed_when: yes
        notify: run me even after an error
      - command: /bin/false
    rescue:
      - name: make sure all handlers run
        meta: flush_handlers
handlers:
  - name: run me even after an error
    debug:
      msg: 'This handler runs even on error'
```

2.1 新版功能.

Ansible provides a couple of variables for tasks in the `rescue` portion of a block:

ansible_failed_task The task that returned ‘failed’ and triggered the rescue. For example, to get the name use `ansible_failed_task.name`.

ansible_failed_result The captured return result of the failed task that triggered the rescue. This would equate to having used this var in the `register` keyword.

参见:

Intro to Playbooks An introduction to playbooks

Roles Playbook organization by roles

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

Advanced Playbooks Features

Here are some playbook features that not everyone may need to learn, but can be quite useful for particular applications. Browsing these topics is recommended as you may find some useful tips here, but feel free to learn the basics of Ansible first and adopt these only if they seem relevant or useful to your environment.

Understanding privilege escalation: become

Ansible uses existing privilege escalation systems to execute tasks with root privileges or with another user's permissions. Because this feature allows you to 'become' another user, different from the user that logged into the machine (remote user), we call it **become**. The **become** keyword leverages existing privilege escalation tools like *sudo*, *su*, *pfexec*, *doas*, *pbrun*, *dzdo*, *ksu*, *runas*, *machinectl* and others.

- *Using become*
 - *Become directives*
 - *Become connection variables*
 - *Become command-line options*
- *Risks and limitations of become*
 - *Risks of becoming an unprivileged user*
 - *Not supported by all connection plugins*
 - *Only one method may be enabled per host*
 - *Privilege escalation must be general*
 - *May not access environment variables populated by pamd_systemd*
- *Become and network automation*
 - *Setting enable mode for all tasks*
 - * *Passwords for enable mode*
 - *authorize and auth_pass*
- *Become and Windows*
 - *Administrative rights*
 - *Local service accounts*
 - *Become without setting a password*
 - *Accounts without a password*
 - *Become flags for Windows*

Using become

You can control the use of **become** with play or task directives, connection variables, or at the command line. If you set privilege escalation properties in multiple ways, review the *general precedence rules* to understand which settings will be used.

A full list of all become plugins that are included in Ansible can be found in the *Plugin List*.

Become directives

You can set the directives that control **become** at the play or task level. You can override these by setting connection variables, which often differ from one host to another. These variables and directives are independent. For example, setting **become_user** does not set **become**.

become set to **yes** to activate privilege escalation.

become_user set to user with desired privileges —the user you *become*, NOT the user you login as. Does NOT imply **become: yes**, to allow it to be set at host level. Default value is **root**.

become_method (at play or task level) overrides the default method set in `ansible.cfg`, set to use any of the *Become Plugins*.

become_flags (at play or task level) permit the use of specific flags for the tasks or role. One common use is to change the user to nobody when the shell is set to no login. Added in Ansible 2.2.

For example, to manage a system service (which requires **root** privileges) when connected as a non-**root** user, you can use the default value of **become_user** (**root**):

```
- name: Ensure the httpd service is running
  service:
    name: httpd
    state: started
    become: yes
```

To run a command as the **apache** user:

```
- name: Run a command as the apache user
  command: somecommand
  become: yes
  become_user: apache
```

To do something as the **nobody** user when the shell is `nologin`:

```
- name: Run a command as nobody
  command: somecommand
  become: yes
  become_method: su
  become_user: nobody
  become_flags: '-s /bin/sh'
```

To specify a password for sudo, run `ansible-playbook` with `--ask-become-pass` (`-K` for short). If you run a playbook utilizing `become` and the playbook seems to hang, most likely it is stuck at the privilege escalation prompt. Stop it with `CTRL-c`, then execute the playbook with `-K` and the appropriate password.

Become connection variables

You can define different `become` options for each managed node or group. You can define these variables in inventory or use them as normal variables.

`ansible_become` equivalent of the `become` directive, decides if privilege escalation is used or not.

`ansible_become_method` which privilege escalation method should be used

`ansible_become_user` set the user you become through privilege escalation; does not imply `ansible_become: yes`

`ansible_become_password` set the privilege escalation password. See *Using Vault in playbooks* for details on how to avoid having secrets in plain text

For example, if you want to run all tasks as `root` on a server named `webserver`, but you can only connect as the `manager` user, you could use an inventory entry like this:

```
webserver ansible_user=manager ansible_become=yes
```

注解: The variables defined above are generic for all `become` plugins but plugin specific ones can also be set instead. Please see the documentation for each plugin for a list of all options the plugin has and how they can be defined. A full list of `become` plugins in Ansible can be found at *Become Plugins*.

Become command-line options

`--ask-become-pass, -K` ask for privilege escalation password; does not imply `become` will be used. Note that this password will be used for all hosts.

`--become, -b` run operations with `become` (no password implied)

--become-method=BECOME_METHOD privilege escalation method to use (default=sudo), valid choices: [sudo | su | pbrun | pfexec | doas | dzdo | ksu | runas | machinectl]

--become-user=BECOME_USER run operations as this user (default=root), does not imply `-become/-b`

Risks and limitations of become

Although privilege escalation is mostly intuitive, there are a few limitations on how it works. Users should be aware of these to avoid surprises.

Risks of becoming an unprivileged user

Ansible modules are executed on the remote machine by first substituting the parameters into the module file, then copying the file to the remote machine, and finally executing it there.

Everything is fine if the module file is executed without using `become`, when the `become_user` is root, or when the connection to the remote machine is made as root. In these cases Ansible creates the module file with permissions that only allow reading by the user and root, or only allow reading by the unprivileged user being switched to.

However, when both the connection user and the `become_user` are unprivileged, the module file is written as the user that Ansible connects as, but the file needs to be readable by the user Ansible is set to `become`. In this case, Ansible makes the module file world-readable for the duration of the Ansible module execution. Once the module is done executing, Ansible deletes the temporary file.

If any of the parameters passed to the module are sensitive in nature, and you do not trust the client machines, then this is a potential danger.

Ways to resolve this include:

- Use *pipelining*. When pipelining is enabled, Ansible does not save the module to a temporary file on the client. Instead it pipes the module to the remote python interpreter's stdin. Pipelining does not work for python modules involving file transfer (for example: copy, fetch, template), or for non-python modules.
- Install POSIX.1e filesystem acl support on the managed host. If the temporary directory on the remote host is mounted with POSIX acls enabled and the `setfacl` tool is in the remote `PATH` then Ansible will use POSIX acls to share the module file with the second unprivileged user instead of having to make the file readable by everyone.
- Avoid becoming an unprivileged user. Temporary files are protected by UNIX file permissions when you `become` root or do not use `become`. In Ansible 2.1 and above, UNIX file permissions are also secure if you make the connection to the managed machine as root and then use `become` to access an unprivileged account.

警告: Although the Solaris ZFS filesystem has filesystem ACLs, the ACLs are not POSIX.1e filesystem acls (they are NFSv4 ACLs instead). Ansible cannot use these ACLs to manage its temp file permissions so you may have to resort to `allow_world_readable_tmpfiles` if the remote machines use ZFS.

在 2.1 版更改.

Ansible makes it hard to unknowingly use `become` insecurely. Starting in Ansible 2.1, Ansible defaults to issuing an error if it cannot execute securely with `become`. If you cannot use pipelining or POSIX ACLs, you must connect as an unprivileged user, you must use `become` to execute as a different unprivileged user, and you decide that your managed nodes are secure enough for the modules you want to run there to be world readable, you can turn on `allow_world_readable_tmpfiles` in the `ansible.cfg` file. Setting `allow_world_readable_tmpfiles` will change this from an error into a warning and allow the task to run as it did prior to 2.1.

Not supported by all connection plugins

Privilege escalation methods must also be supported by the connection plugin used. Most connection plugins will warn if they do not support `become`. Some will just ignore it as they always run as root (jail, chroot, etc).

Only one method may be enabled per host

Methods cannot be chained. You cannot use `sudo /bin/su -` to become a user, you need to have privileges to run the command as that user in sudo or be able to su directly to it (the same for pbrun, pfexec or other supported methods).

Privilege escalation must be general

You cannot limit privilege escalation permissions to certain commands. Ansible does not always use a specific command to do something but runs modules (code) from a temporary file name which changes every time. If you have `‘/sbin/service’` or `‘/bin/chmod’` as the allowed commands this will fail with ansible as those paths won't match with the temporary file that Ansible creates to run the module. If you have security rules that constrain your sudo/pbrun/doas environment to running specific command paths only, use Ansible from a special account that does not have this constraint, or use *Red Hat Ansible Tower* to manage indirect access to SSH credentials.

May not access environment variables populated by pamd_systemd

For most Linux distributions using `systemd` as their init, the default methods used by `become` do not open a new “session”, in the sense of `systemd`. Because the `pam_systemd` module will not fully initialize a

new session, you might have surprises compared to a normal session opened through ssh: some environment variables set by `pam_systemd`, most notably `XDG_RUNTIME_DIR`, are not populated for the new user and instead inherited or just emptied.

This might cause trouble when trying to invoke systemd commands that depend on `XDG_RUNTIME_DIR` to access the bus:

```
$ echo $XDG_RUNTIME_DIR

$ systemctl --user status
Failed to connect to bus: Permission denied
```

To force `become` to open a new systemd session that goes through `pam_systemd`, you can use `become_method: machinectl`.

For more information, see [this systemd issue](#).

Become and network automation

As of version 2.6, Ansible supports `become` for privilege escalation (entering `enable` mode or privileged EXEC mode) on all Ansible-maintained platforms that support `enable` mode. Using `become` replaces the `authorize` and `auth_pass` options in a `provider` dictionary.

You must set the connection type to either `connection: network_cli` or `connection: httpapi` to use `become` for privilege escalation on network devices. Check the *Platform Options* and `network_modules` documentation for details.

You can use escalated privileges on only the specific tasks that need them, on an entire play, or on all plays. Adding `become: yes` and `become_method: enable` instructs Ansible to enter `enable` mode before executing the task, play, or playbook where those parameters are set.

If you see this error message, the task that generated it requires `enable` mode to succeed:

```
Invalid input (privileged mode required)
```

To set `enable` mode for a specific task, add `become` at the task level:

```
- name: Gather facts (eos)
  eos_facts:
    gather_subset:
      - "!hardware"
  become: yes
  become_method: enable
```

To set `enable` mode for all tasks in a single play, add `become` at the play level:

```
- hosts: eos-switches
  become: yes
  become_method: enable
  tasks:
    - name: Gather facts (eos)
      eos_facts:
        gather_subset:
          - "!hardware"
```

Setting enable mode for all tasks

Often you wish for all tasks in all plays to run using privilege mode, that is best achieved by using `group_vars:` `group__vars/eos.yml`

```
ansible_connection: network_cli
ansible_network_os: eos
ansible_user: myuser
ansible_become: yes
ansible_become_method: enable
```

Passwords for enable mode

If you need a password to enter `enable` mode, you can specify it in one of two ways:

- providing the `--ask-become-pass` command line option
- setting the `ansible_become_password` connection variable

警告: As a reminder passwords should never be stored in plain text. For information on encrypting your passwords and other secrets with Ansible Vault, see [Ansible Vault](#).

authorize and auth_pass

Ansible still supports `enable` mode with `connection: local` for legacy network playbooks. To enter `enable` mode with `connection: local`, use the module options `authorize` and `auth_pass`:

```
- hosts: eos-switches
  ansible_connection: local
  tasks:
```

(下页继续)

(续上页)

```
- name: Gather facts (eos)
  eos_facts:
    gather_subset:
      - "!hardware"
  provider:
    authorize: yes
    auth_pass: " {{ secret_auth_pass }}"
```

We recommend updating your playbooks to use `become` for network-device `enable` mode consistently. The use of `authorize` and of `provider` dictionaries will be deprecated in future. Check the *Platform Options* and `network_modules` documentation for details.

Become and Windows

Since Ansible 2.3, `become` can be used on Windows hosts through the `runas` method. Become on Windows uses the same inventory setup and invocation arguments as `become` on a non-Windows host, so the setup and variable names are the same as what is defined in this document.

While `become` can be used to assume the identity of another user, there are other uses for it with Windows hosts. One important use is to bypass some of the limitations that are imposed when running on WinRM, such as constrained network delegation or accessing forbidden system calls like the WUA API. You can use `become` with the same user as `ansible_user` to bypass these limitations and run commands that are not normally accessible in a WinRM session.

Administrative rights

Many tasks in Windows require administrative privileges to complete. When using the `runas` become method, Ansible will attempt to run the module with the full privileges that are available to the remote user. If it fails to elevate the user token, it will continue to use the limited token during execution.

A user must have the `SeDebugPrivilege` to run a become process with elevated privileges. This privilege is assigned to Administrators by default. If the debug privilege is not available, the become process will run with a limited set of privileges and groups.

To determine the type of token that Ansible was able to get, run the following task:

```
- win_whoami:
  become: yes
```

The output will look something similar to the below:

```
ok: [windows] => {
  "account": {
    "account_name": "vagrant-domain",
    "domain_name": "DOMAIN",
    "sid": "S-1-5-21-3088887838-4058132883-1884671576-1105",
    "type": "User"
  },
  "authentication_package": "Kerberos",
  "changed": false,
  "dns_domain_name": "DOMAIN.LOCAL",
  "groups": [
    {
      "account_name": "Administrators",
      "attributes": [
        "Mandatory",
        "Enabled by default",
        "Enabled",
        "Owner"
      ],
      "domain_name": "BUILTIN",
      "sid": "S-1-5-32-544",
      "type": "Alias"
    },
    {
      "account_name": "INTERACTIVE",
      "attributes": [
        "Mandatory",
        "Enabled by default",
        "Enabled"
      ],
      "domain_name": "NT AUTHORITY",
      "sid": "S-1-5-4",
      "type": "WellKnownGroup"
    }
  ],
  "impersonation_level": "SecurityAnonymous",
  "label": {
    "account_name": "High Mandatory Level",
    "domain_name": "Mandatory Label",
    "sid": "S-1-16-12288",
```

(下页继续)

(续上页)

```

    "type": "Label"
  },
  "login_domain": "DOMAIN",
  "login_time": "2018-11-18T20:35:01.9696884+00:00",
  "logon_id": 114196830,
  "logon_server": "DC01",
  "logon_type": "Interactive",
  "privileges": {
    "SeBackupPrivilege": "disabled",
    "SeChangeNotifyPrivilege": "enabled-by-default",
    "SeCreateGlobalPrivilege": "enabled-by-default",
    "SeCreatePagefilePrivilege": "disabled",
    "SeCreateSymbolicLinkPrivilege": "disabled",
    "SeDebugPrivilege": "enabled",
    "SeDelegateSessionUserImpersonatePrivilege": "disabled",
    "SeImpersonatePrivilege": "enabled-by-default",
    "SeIncreaseBasePriorityPrivilege": "disabled",
    "SeIncreaseQuotaPrivilege": "disabled",
    "SeIncreaseWorkingSetPrivilege": "disabled",
    "SeLoadDriverPrivilege": "disabled",
    "SeManageVolumePrivilege": "disabled",
    "SeProfileSingleProcessPrivilege": "disabled",
    "SeRemoteShutdownPrivilege": "disabled",
    "SeRestorePrivilege": "disabled",
    "SeSecurityPrivilege": "disabled",
    "SeShutdownPrivilege": "disabled",
    "SeSystemEnvironmentPrivilege": "disabled",
    "SeSystemProfilePrivilege": "disabled",
    "SeSystemtimePrivilege": "disabled",
    "SeTakeOwnershipPrivilege": "disabled",
    "SeTimeZonePrivilege": "disabled",
    "SeUndockPrivilege": "disabled"
  },
  "rights": [
    "SeNetworkLogonRight",
    "SeBatchLogonRight",
    "SeInteractiveLogonRight",
    "SeRemoteInteractiveLogonRight"
  ],
  "token_type": "TokenPrimary",

```

(下页继续)

(续上页)

```

"upn": "vagrant-domain@DOMAIN.LOCAL",
"user_flags": []
}

```

Under the `label` key, the `account_name` entry determines whether the user has Administrative rights. Here are the labels that can be returned and what they represent:

- **Medium:** Ansible failed to get an elevated token and ran under a limited token. Only a subset of the privileges assigned to user are available during the module execution and the user does not have administrative rights.
- **High:** An elevated token was used and all the privileges assigned to the user are available during the module execution.
- **System:** The NT AUTHORITY\System account is used and has the highest level of privileges available.

The output will also show the list of privileges that have been granted to the user. When the privilege value is `disabled`, the privilege is assigned to the logon token but has not been enabled. In most scenarios these privileges are automatically enabled when required.

If running on a version of Ansible that is older than 2.5 or the normal `runas` escalation process fails, an elevated token can be retrieved by:

- Set the `become_user` to `System` which has full control over the operating system.
- Grant `SeTcbPrivilege` to the user Ansible connects with on WinRM. `SeTcbPrivilege` is a high-level privilege that grants full control over the operating system. No user is given this privilege by default, and care should be taken if you grant this privilege to a user or group. For more information on this privilege, please see [Act as part of the operating system](#). You can use the below task to set this privilege on a Windows host:

```

- name: grant the ansible user the SeTcbPrivilege right
  win_user_right:
    name: SeTcbPrivilege
    users: '{{ansible_user}}'
    action: add

```

- Turn UAC off on the host and reboot before trying to become the user. UAC is a security protocol that is designed to run accounts with the `least privilege` principle. You can turn UAC off by running the following tasks:

```

- name: turn UAC off
  win_regedit:
    path: HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\policies\system
    name: EnableLUA

```

(下页继续)

(续上页)

```
data: 0
type: dword
state: present
register: uac_result

- name: reboot after disabling UAC
  win_reboot:
  when: uac_result is changed
```

注解: Granting the `SeTcbPrivilege` or turning UAC off can cause Windows security vulnerabilities and care should be given if these steps are taken.

Local service accounts

Prior to Ansible version 2.5, `become` only worked on Windows with a local or domain user account. Local service accounts like `System` or `NetworkService` could not be used as `become_user` in these older versions. This restriction has been lifted since the 2.5 release of Ansible. The three service accounts that can be set under `become_user` are:

- System
- NetworkService
- LocalService

Because local service accounts do not have passwords, the `ansible_become_password` parameter is not required and is ignored if specified.

Become without setting a password

As of Ansible 2.8, `become` can be used to become a Windows local or domain account without requiring a password for that account. For this method to work, the following requirements must be met:

- The connection user has the `SeDebugPrivilege` privilege assigned
- The connection user is part of the `BUILTIN\Administrators` group
- The `become_user` has either the `SeBatchLogonRight` or `SeNetworkLogonRight` user right

Using `become` without a password is achieved in one of two different methods:

- Duplicating an existing logon session's token if the account is already logged on
- Using S4U to generate a logon token that is valid on the remote host only

In the first scenario, the become process is spawned from another logon of that user account. This could be an existing RDP logon, console logon, but this is not guaranteed to occur all the time. This is similar to the `Run only when user is logged on` option for a Scheduled Task.

In the case where another logon of the become account does not exist, S4U is used to create a new logon and run the module through that. This is similar to the `Run whether user is logged on or not` with the `Do not store password` option for a Scheduled Task. In this scenario, the become process will not be able to access any network resources like a normal WinRM process.

To make a distinction between using become with no password and becoming an account that has no password make sure to keep `ansible_become_password` as undefined or set `ansible_become_password:.`

注解: Because there are no guarantees an existing token will exist for a user when Ansible runs, there's a high chance the become process will only have access to local resources. Use become with a password if the task needs to access network resources

Accounts without a password

警告: As a general security best practice, you should avoid allowing accounts without passwords.

Ansible can be used to become a Windows account that does not have a password (like the `Guest` account). To become an account without a password, set up the variables like normal but set `ansible_become_password: ''`.

Before become can work on an account like this, the local policy `Accounts: Limit local account use of blank passwords to console logon only` must be disabled. This can either be done through a Group Policy Object (GPO) or with this Ansible task:

```
- name: allow blank password on become
  win_regedit:
    path: HKLM:\SYSTEM\CurrentControlSet\Control\Lsa
    name: LimitBlankPasswordUse
    data: 0
    type: dword
    state: present
```

注解: This is only for accounts that do not have a password. You still need to set the account's password under `ansible_become_password` if the `become_user` has a password.

Become flags for Windows

Ansible 2.5 added the `become_flags` parameter to the `runas` become method. This parameter can be set using the `become_flags` task directive or set in Ansible's configuration using `ansible_become_flags`. The two valid values that are initially supported for this parameter are `logon_type` and `logon_flags`.

注解: These flags should only be set when becoming a normal user account, not a local service account like LocalSystem.

The key `logon_type` sets the type of logon operation to perform. The value can be set to one of the following:

- **interactive:** The default logon type. The process will be run under a context that is the same as when running a process locally. This bypasses all WinRM restrictions and is the recommended method to use.
- **batch:** Runs the process under a batch context that is similar to a scheduled task with a password set. This should bypass most WinRM restrictions and is useful if the `become_user` is not allowed to log on interactively.
- **new_credentials:** Runs under the same credentials as the calling user, but outbound connections are run under the context of the `become_user` and `become_password`, similar to `runas.exe /netonly`. The `logon_flags` flag should also be set to `netcredentials_only`. Use this flag if the process needs to access a network resource (like an SMB share) using a different set of credentials.
- **network:** Runs the process under a network context without any cached credentials. This results in the same type of logon session as running a normal WinRM process without credential delegation, and operates under the same restrictions.
- **network_cleartext:** Like the `network` logon type, but instead caches the credentials so it can access network resources. This is the same type of logon session as running a normal WinRM process with credential delegation.

For more information, see [dwLogonType](#).

The `logon_flags` key specifies how Windows will log the user on when creating the new process. The value can be set to none or multiple of the following:

- **with_profile:** The default logon flag set. The process will load the user's profile in the `HKEY_USERS` registry key to `HKEY_CURRENT_USER`.
- **netcredentials_only:** The process will use the same token as the caller but will use the `become_user` and `become_password` when accessing a remote resource. This is useful in inter-domain scenarios where there is no trust relationship, and should be used with the `new_credentials` logon_type.

By default `logon_flags=with_profile` is set, if the profile should not be loaded set `logon_flags=` or if the profile should be loaded with `netcredentials_only`, set `logon_flags=with_profile, netcredentials_only`.

For more information, see [dwLogonFlags](#).

Here are some examples of how to use `become_flags` with Windows tasks:

```
- name: copy a file from a fileshare with custom credentials
  win_copy:
    src: \\server\share\data\file.txt
    dest: C:\temp\file.txt
    remote_src: yes
  vars:
    ansible_become: yes
    ansible_become_method: runas
    ansible_become_user: DOMAIN\user
    ansible_become_password: Password01
    ansible_become_flags: logon_type=new_credentials logon_flags=netcredentials_only

- name: run a command under a batch logon
  win_whoami:
    become: yes
    become_flags: logon_type=batch

- name: run a command and not load the user profile
  win_whomai:
    become: yes
    become_flags: logon_flags=
```

Limitations of become on Windows

- Running a task with `async` and `become` on Windows Server 2008, 2008 R2 and Windows 7 only works when using Ansible 2.7 or newer.
- By default, the become user logs on with an interactive session, so it must have the right to do so on the Windows host. If it does not inherit the `SeAllowLogOnLocally` privilege or inherits the `SeDenyLogOnLocally` privilege, the become process will fail. Either add the privilege or set the `logon_type` flag to change the logon type used.
- Prior to Ansible version 2.3, become only worked when `ansible_winrm_transport` was either `basic` or `credssp`. This restriction has been lifted since the 2.4 release of Ansible for all hosts except Windows Server 2008 (non R2 version).
- The Secondary Logon service `seclogon` must be running to use `ansible_become_method: runas`

参见:

[Mailing List](#) Questions? Help? Ideas? Stop by the list on Google Groups

[#ansible](http://webchat.freenode.net) IRC chat channel

Asynchronous actions and polling

By default Ansible runs tasks synchronously, holding the connection to the remote node open until the action is completed. This means within a playbook, each task blocks the next task by default, meaning subsequent tasks will not run until the current task completes. This behavior can create challenges. For example, a task may take longer to complete than the SSH session allows for, causing a timeout. Or you may want a long-running process to execute in the background while you perform other tasks concurrently. Asynchronous mode lets you control how long-running tasks execute.

- *Asynchronous ad-hoc tasks*
- *Asynchronous playbook tasks*
 - *Avoid connection timeouts: `poll > 0`*
 - *Run tasks concurrently: `poll = 0`*

Asynchronous ad-hoc tasks

You can execute long-running operations in the background with *ad-hoc tasks*. For example, to execute `long_running_operation` asynchronously in the background, with a timeout (`-B`) of 3600 seconds, and without polling (`-P`):

```
$ ansible all -B 3600 -P 0 -a "/usr/bin/long_running_operation --do-stuff"
```

To check on the job status later, use the `async_status` module, passing it the job ID that was returned when you ran the original job in the background:

```
$ ansible web1.example.com -m async_status -a "jid=488359678239.2844"
```

Ansible can also check on the status of your long-running job automatically with polling. In most cases, Ansible will keep the connection to your remote node open between polls. To run for 30 minutes and poll for status every 60 seconds:

```
$ ansible all -B 1800 -P 60 -a "/usr/bin/long_running_operation --do-stuff"
```

Poll mode is smart so all jobs will be started before polling begins on any machine. Be sure to use a high enough `--forks` value if you want to get all of your jobs started very quickly. After the time limit (in seconds) runs out (`-B`), the process on the remote nodes will be terminated.

Asynchronous mode is best suited to long-running shell commands or software upgrades. Running the `copy` module asynchronously, for example, does not do a background file transfer.

Asynchronous playbook tasks

Playbooks also support asynchronous mode and polling, with a simplified syntax. You can use asynchronous mode in playbooks to avoid connection timeouts or to avoid blocking subsequent tasks. The behavior of asynchronous mode in a playbook depends on the value of *poll*.

Avoid connection timeouts: `poll > 0`

If you want to set a longer timeout limit for a certain task in your playbook, use `async` with `poll` set to a positive value. Ansible will still block the next task in your playbook, waiting until the async task either completes, fails or times out. However, the task will only time out if it exceeds the timeout limit you set with the `async` parameter.

To avoid timeouts on a task, specify its maximum runtime and how frequently you would like to poll for status:

```
---

- hosts: all
  remote_user: root

  tasks:

  - name: simulate long running op (15 sec), wait for up to 45 sec, poll every 5 sec
    command: /bin/sleep 15
    async: 45
    poll: 5
```

注解: The default poll value is set by the `DEFAULT_POLL_INTERVAL` setting. There is no default for the async time limit. If you leave off the ‘`async`’ keyword, the task runs synchronously, which is Ansible’s default.

注解: As of Ansible 2.3, `async` does not support check mode and will fail the task when run in check mode. See *Check Mode (“Dry Run”)* on how to skip a task in check mode.

Run tasks concurrently: `poll = 0`

If you want to run multiple tasks in a playbook concurrently, use `async` with `poll` set to 0. When you set `poll: 0`, Ansible starts the task and immediately moves on to the next task without waiting for a result.

Each async task runs until it either completes, fails or times out (runs longer than its `async` value). The playbook run ends without checking back on async tasks.

To run a playbook task asynchronously:

```
---

- hosts: all
  remote_user: root

  tasks:

  - name: simulate long running op, allow to run for 45 sec, fire and forget
    command: /bin/sleep 15
    async: 45
    poll: 0
```

注解: Do not specify a poll value of 0 with operations that require exclusive locks (such as yum transactions) if you expect to run other commands later in the playbook against those same resources.

注解: Using a higher value for `--forks` will result in kicking off asynchronous tasks even faster. This also increases the efficiency of polling.

If you need a synchronization point with an async task, you can register it to obtain its job ID and use the `async_status` module to observe it in a later task. For example:

```
- name: 'YUM - async task'
  yum:
    name: docker-io
    state: present
  async: 1000
  poll: 0
  register: yum_sleeper

- name: 'YUM - check on async task'
  async_status:
    jid: "{{ yum_sleeper.ansible_job_id }}"
  register: job_result
  until: job_result.finished
  retries: 30
```

注解: If the value of `async:` is not high enough, this will cause the “check on it later” task to fail because the temporary status file that the `async_status:` is looking for will not have been written or no longer exist

To run multiple asynchronous tasks while limiting the number of tasks running concurrently:

```
#####
# main.yml
#####
- name: Run items asynchronously in batch of two items
  vars:
    sleep_durations:
      - 1
      - 2
      - 3
      - 4
      - 5
    durations: "{{ item }}"
  include_tasks: execute_batch.yml
  loop: "{{ sleep_durations | batch(2) | list }}"

#####
# execute_batch.yml
#####
- name: Async sleeping for batched_items
  command: sleep {{ async_item }}
  async: 45
  poll: 0
  loop: "{{ durations }}"
  loop_control:
    loop_var: "async_item"
  register: async_results

- name: Check sync status
  async_status:
    jid: "{{ async_result_item.ansible_job_id }}"
  loop: "{{ async_results.results }}"
  loop_control:
    loop_var: "async_result_item"
  register: async_poll_results
  until: async_poll_results.finished
```

(下页继续)

(续上页)

`retries: 30`**参见:***Controlling playbook execution: strategies and more* Options for controlling playbook execution*Intro to Playbooks* An introduction to playbooks**User Mailing List** Have a question? Stop by the google group!**irc.freenode.net** #ansible IRC chat channel**Check Mode (“Dry Run”)**

1.1 新版功能.

Topics

- *Check Mode (“Dry Run”)*
 - *Enabling or disabling check mode for tasks*
 - *Information about check mode in variables*
 - *Showing Differences with `--diff`*

When `ansible-playbook` is executed with `--check` it will not make any changes on remote systems. Instead, any module instrumented to support ‘check mode’ (which contains most of the primary core modules, but it is not required that all modules do this) will report what changes they would have made rather than making them. Other modules that do not support check mode will also take no action, but just will not report what changes they might have made.

Check mode is just a simulation, and if you have steps that use conditionals that depend on the results of prior commands, it may be less useful for you. However it is great for one-node-at-time basic configuration management use cases.

Example:

```
ansible-playbook foo.yml --check
```

Enabling or disabling check mode for tasks

2.2 新版功能.

Sometimes you may want to modify the check mode behavior of individual tasks. This is done via the `check_mode` option, which can be added to tasks.

There are two options:

1. Force a task to **run in check mode**, even when the playbook is called **without** `--check`. This is called `check_mode: yes`.
2. Force a task to **run in normal mode** and make changes to the system, even when the playbook is called **with** `--check`. This is called `check_mode: no`.

注解: Prior to version 2.2 only the equivalent of `check_mode: no` existed. The notation for that was `always_run: yes`.

Instead of `yes/no` you can use a Jinja2 expression, just like the `when` clause.

Example:

```
tasks:
- name: this task will make changes to the system even in check mode
  command: /something/to/run --even-in-check-mode
  check_mode: no

- name: this task will always run under checkmode and not change the system
  lineinfile:
    line: "important config"
    dest: /path/to/myconfig.conf
    state: present
  check_mode: yes
```

Running single tasks with `check_mode: yes` can be useful to write tests for ansible modules, either to test the module itself or to the conditions under which a module would make changes. With `register` (see *Conditionals*) you can check the potential changes.

Information about check mode in variables

2.1 新版功能.

If you want to skip, or ignore errors on some tasks in check mode you can use a boolean magic variable `ansible_check_mode` which will be set to `True` during check mode.

Example:

```
tasks:

- name: this task will be skipped in check mode
  git:
```

(下页继续)

(续上页)

```

    repo: ssh://git@github.com/mylogin/hello.git
    dest: /home/mylogin/hello
    when: not ansible_check_mode

- name: this task will ignore errors in check mode
  git:
    repo: ssh://git@github.com/mylogin/hello.git
    dest: /home/mylogin/hello
    ignore_errors: "{{ ansible_check_mode }}"

```

Showing Differences with --diff

1.1 新版功能.

The `--diff` option to `ansible-playbook` works great with `--check` (detailed above) but can also be used by itself. When this flag is supplied and the module supports this, Ansible will report back the changes made or, if used with `--check`, the changes that would have been made. This is mostly used in modules that manipulate files (i.e. `template`) but other modules might also show ‘before and after’ information (i.e. `user`). Since the diff feature produces a large amount of output, it is best used when checking a single host at a time. For example:

```
ansible-playbook foo.yml --check --diff --limit foo.example.com
```

2.4 新版功能.

The `--diff` option can reveal sensitive information. This option can be disabled for tasks by specifying `diff: no`.

Example:

```

tasks:
- name: this task will not report a diff when the file changes
  template:
    src: secret.conf.j2
    dest: /etc/secret.conf
    owner: root
    group: root
    mode: '0600'
  diff: no

```

Playbook Debugger

Topics

- *Playbook Debugger*
 - *Using the debugger keyword*
 - * *On a task*
 - * *On a play*
 - *Configuration or environment variable*
 - *As a Strategy*
 - *Examples*
 - *Available Commands*
 - * *p(print) task/task_vars/host/result*
 - * *task.args[key] = value*
 - * *task_vars[key] = value*
 - * *u(pdate_task)*
 - * *r(edo)*
 - * *c(ontinue)*
 - * *q(uit)*
 - *Use with the free strategy*

Ansible includes a debugger as part of the strategy plugins. This debugger enables you to debug a task. You have access to all of the features of the debugger in the context of the task. You can then, for example, check or set the value of variables, update module arguments, and re-run the task with the new variables and arguments to help resolve the cause of the failure.

There are multiple ways to invoke the debugger.

Using the debugger keyword

2.5 新版功能.

The **debugger** keyword can be used on any block where you provide a **name** attribute, such as a play, role, block or task.

The **debugger** keyword accepts several values:

always Always invoke the debugger, regardless of the outcome

never Never invoke the debugger, regardless of the outcome

on_failed Only invoke the debugger if a task fails

on_unreachable Only invoke the debugger if the a host was unreachable

on_skipped Only invoke the debugger if the task is skipped

These options override any global configuration to enable or disable the debugger.

On a task

```
- name: Execute a command
  command: false
  debugger: on_failed
```

On a play

```
- name: Play
  hosts: all
  debugger: on_skipped
  tasks:
    - name: Execute a command
      command: true
      when: False
```

When provided at a generic level and a more specific level, the more specific wins:

```
- name: Play
  hosts: all
  debugger: never
  tasks:
    - name: Execute a command
      command: false
      debugger: on_failed
```

Configuration or environment variable

2.5 新版功能.

In ansible.cfg:

```
[defaults]
enable_task_debugger = True
```

As an environment variable:

```
ANSIBLE_ENABLE_TASK_DEBUGGER=True ansible-playbook -i hosts site.yml
```

When using this method, any failed or unreachable task will invoke the debugger, unless otherwise explicitly disabled.

As a Strategy

注解: This is a backwards compatible method, to match Ansible versions before 2.5, and may be removed in a future release

To use the `debug` strategy, change the `strategy` attribute like this:

```
- hosts: test
  strategy: debug
  tasks:
  ...
```

If you don't want change the code, you can define `ANSIBLE_STRATEGY=debug` environment variable in order to enable the debugger, or modify `ansible.cfg` such as:

```
[defaults]
strategy = debug
```

Examples

For example, run the playbook below:

```
- hosts: test
  debugger: on_failed
  gather_facts: no
  vars:
    var1: value1
  tasks:
    - name: wrong variable
      ping: data={{ wrong_var }}
```

The debugger is invoked since the *wrong_var* variable is undefined.

Let's change the module's arguments and run the task again

```
PLAY *****

TASK [wrong variable] *****
fatal: [192.0.2.10]: FAILED! => {"failed": true, "msg": "ERROR! 'wrong_var' is undefined
↪"}
Debugger invoked
[192.0.2.10] TASK: wrong variable (debug)> p result._result
{'failed': True,
 'msg': 'The task includes an option with an undefined variable. The error '
       '"was: 'wrong_var' is undefined\n"
       '\n'
       'The error appears to have been in '
       "'playbooks/debugger.yml': line 7, "
       'column 7, but may\n'
       'be elsewhere in the file depending on the exact syntax problem.\n'
       '\n'
       'The offending line appears to be:\n'
       '\n'
       '   tasks:\n'
       '     - name: wrong variable\n'
       '       ^ here\n'}
[192.0.2.10] TASK: wrong variable (debug)> p task.args
{'data': u'{{ wrong_var }}'}
[192.0.2.10] TASK: wrong variable (debug)> task.args['data'] = '{{ var1 }}'
[192.0.2.10] TASK: wrong variable (debug)> p task.args
{'data': '{{ var1 }}'}
[192.0.2.10] TASK: wrong variable (debug)> redo
ok: [192.0.2.10]

PLAY RECAP *****
192.0.2.10          : ok=1    changed=0    unreachable=0    failed=0
```

This time, the task runs successfully!

Available Commands

p(print) task/task_vars/host/result

Print values used to execute a module:

```
[192.0.2.10] TASK: install package (debug)> p task
TASK: install package
[192.0.2.10] TASK: install package (debug)> p task.args
{'u'name': u'{{ pkg_name }}'}
[192.0.2.10] TASK: install package (debug)> p task_vars
{'u'ansible_all_ipv4_addresses': [u'192.0.2.10'],
  u'ansible_architecture': u'x86_64',
  ...
}
[192.0.2.10] TASK: install package (debug)> p task_vars['pkg_name']
u'bash'
[192.0.2.10] TASK: install package (debug)> p host
192.0.2.10
[192.0.2.10] TASK: install package (debug)> p result._result
{'_ansible_no_log': False,
  'changed': False,
  u'failed': True,
  ...
  u'msg': u"No package matching 'not_exist' is available"}
```

task.args[key] = value

Update module' s argument.

If you run a playbook like this:

```
- hosts: test
  strategy: debug
  gather_facts: yes
  vars:
    pkg_name: not_exist
  tasks:
    - name: install package
      apt: name={{ pkg_name }}
```

Debugger is invoked due to wrong package name, so let' s fix the module' s args:

```
[192.0.2.10] TASK: install package (debug)> p task.args
{'u'name': u'{{ pkg_name }}'}
[192.0.2.10] TASK: install package (debug)> task.args['name'] = 'bash'
[192.0.2.10] TASK: install package (debug)> p task.args
{'u'name': 'bash'}
[192.0.2.10] TASK: install package (debug)> redo
```

Then the task runs again with new args.

task_vars[key] = value

Update task_vars.

Let's use the same playbook above, but fix task_vars instead of args:

```
[192.0.2.10] TASK: install package (debug)> p task_vars['pkg_name']
u'not_exist'
[192.0.2.10] TASK: install package (debug)> task_vars['pkg_name'] = 'bash'
[192.0.2.10] TASK: install package (debug)> p task_vars['pkg_name']
'bash'
[192.0.2.10] TASK: install package (debug)> update_task
[192.0.2.10] TASK: install package (debug)> redo
```

Then the task runs again with new task_vars.

注解: In 2.5 this was updated from vars to task_vars to not conflict with the vars() python function.

u(pdate_task)

2.8 新版功能.

This command re-creates the task from the original task data structure, and templates with updated task_vars

See the above documentation for *task_vars[key] = value* for an example of use.

r(edo)

Run the task again.

c(ontinue)

Just continue.

q(uit)

Quit from the debugger. The playbook execution is aborted.

Use with the free strategy

Using the debugger on the **free** strategy will cause no further tasks to be queued or executed while the debugger is active. Additionally, using **redo** on a task to schedule it for re-execution may cause the rescheduled task to execute after subsequent tasks listed in your playbook.

参见:

Intro to Playbooks An introduction to playbooks

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

Delegation, Rolling Updates, and Local Actions**Topics**

- *Delegation, Rolling Updates, and Local Actions*
 - *Rolling Update Batch Size*
 - *Maximum Failure Percentage*
 - *Delegation*
 - *Delegated facts*
 - *Run Once*
 - *Local Playbooks*
 - *Interrupt execution on any error*

Being designed for multi-tier deployments since the beginning, Ansible is great at doing things on one host on behalf of another, or doing local steps with reference to some remote hosts.

This in particular is very applicable when setting up continuous deployment infrastructure or zero downtime rolling updates, where you might be talking with load balancers or monitoring systems.

Additional features allow for tuning the orders in which things complete, and assigning a batch window size for how many machines to process at once during a rolling update.

This section covers all of these features. For examples of these items in use, [please see the ansible-examples repository](#). There are quite a few examples of zero-downtime update procedures for different kinds of applications.

You should also consult the module documentation section. Modules like `ec2_elb`, `nagios`, `bigip_pool`, and other `network_modules` dovetail neatly with the concepts mentioned here.

You'll also want to read up on [Roles](#), as the 'pre_task' and 'post_task' concepts are the places where you would typically call these modules.

Be aware that certain tasks are impossible to delegate, i.e. `include`, `add_host`, `debug`, etc as they always execute on the controller.

Rolling Update Batch Size

By default, Ansible will try to manage all of the machines referenced in a play in parallel. For a rolling update use case, you can define how many hosts Ansible should manage at a single time by using the `serial` keyword:

```
---
- name: test play
  hosts: webservers
  serial: 2
  gather_facts: False

  tasks:
    - name: task one
      command: hostname
    - name: task two
      command: hostname
```

In the above example, if we had 4 hosts in the group 'webservers', 2 would complete the play completely before moving on to the next 2 hosts:

```
PLAY [webservers] *****

TASK [task one] *****
changed: [web2]
changed: [web1]

TASK [task two] *****
```

(下页继续)

(续上页)

```

changed: [web1]
changed: [web2]

PLAY [webservers] *****

TASK [task one] *****
changed: [web3]
changed: [web4]

TASK [task two] *****
changed: [web3]
changed: [web4]

PLAY RECAP *****
web1      : ok=2    changed=2    unreachable=0    failed=0
web2      : ok=2    changed=2    unreachable=0    failed=0
web3      : ok=2    changed=2    unreachable=0    failed=0
web4      : ok=2    changed=2    unreachable=0    failed=0

```

The `serial` keyword can also be specified as a percentage, which will be applied to the total number of hosts in a play, in order to determine the number of hosts per pass:

```

---
- name: test play
  hosts: webservers
  serial: "30%"

```

If the number of hosts does not divide equally into the number of passes, the final pass will contain the remainder.

As of Ansible 2.2, the batch sizes can be specified as a list, as follows:

```

---
- name: test play
  hosts: webservers
  serial:
    - 1
    - 5
    - 10

```

In the above example, the first batch would contain a single host, the next would contain 5 hosts, and (if there are any hosts left), every following batch would contain 10 hosts until all available hosts are used.

It is also possible to list multiple batch sizes as percentages:

```
---
- name: test play
  hosts: webserver
  serial:
    - "10%"
    - "20%"
    - "100%"
```

You can also mix and match the values:

```
---
- name: test play
  hosts: webserver
  serial:
    - 1
    - 5
    - "20%"
```

注解: No matter how small the percentage, the number of hosts per pass will always be 1 or greater.

Maximum Failure Percentage

By default, Ansible will continue executing actions as long as there are hosts in the batch that have not yet failed. The batch size for a play is determined by the `serial` parameter. If `serial` is not set, then batch size is all the hosts specified in the `hosts:` field. In some situations, such as with the rolling updates described above, it may be desirable to abort the play when a certain threshold of failures have been reached. To achieve this, you can set a maximum failure percentage on a play as follows:

```
---
- hosts: webserver
  max_fail_percentage: 30
  serial: 10
```

In the above example, if more than 3 of the 10 servers in the group were to fail, the rest of the play would be aborted.

注解: The percentage set must be exceeded, not equaled. For example, if `serial` were set to 4 and you

wanted the task to abort when 2 of the systems failed, the percentage should be set at 49 rather than 50.

Delegation

This isn't actually rolling update specific but comes up frequently in those cases.

If you want to perform a task on one host with reference to other hosts, use the 'delegate_to' keyword on a task. This is ideal for placing nodes in a load balanced pool, or removing them. It is also very useful for controlling outage windows. Be aware that it does not make sense to delegate all tasks, debug, add_host, include, etc always get executed on the controller. Using this with the 'serial' keyword to control the number of hosts executing at one time is also a good idea:

```
---
- hosts: webservers
  serial: 5

  tasks:
    - name: take out of load balancer pool
      command: /usr/bin/take_out_of_pool {{ inventory_hostname }}
      delegate_to: 127.0.0.1

    - name: actual steps would go here
      yum:
        name: acme-web-stack
        state: latest

    - name: add back to load balancer pool
      command: /usr/bin/add_back_to_pool {{ inventory_hostname }}
      delegate_to: 127.0.0.1
```

These commands will run on 127.0.0.1, which is the machine running Ansible. There is also a shorthand syntax that you can use on a per-task basis: 'local_action'. Here is the same playbook as above, but using the shorthand syntax for delegating to 127.0.0.1:

```
---
# ...

tasks:
  - name: take out of load balancer pool
    local_action: command /usr/bin/take_out_of_pool {{ inventory_hostname }}
```

(下页继续)

(续上页)

```
# ...

- name: add back to load balancer pool
  local_action: command /usr/bin/add_back_to_pool {{ inventory_hostname }}
```

A common pattern is to use a local action to call ‘rsync’ to recursively copy files to the managed servers. Here is an example:

```
---
# ...

tasks:
- name: recursively copy files from management server to target
  local_action: command rsync -a /path/to/files {{ inventory_hostname }}:/path/to/
  ↪target/
```

Note that you must have passphrase-less SSH keys or an ssh-agent configured for this to work, otherwise rsync will need to ask for a passphrase.

In case you have to specify more arguments you can use the following syntax:

```
---
# ...

tasks:
- name: Send summary mail
  local_action:
    module: mail
    subject: "Summary Mail"
    to: "{{ mail_recipient }}"
    body: "{{ mail_body }}"
  run_once: True
```

The *ansible_host* variable (*ansible_ssh_host* in 1.x or specific to ssh/paramiko plugins) reflects the host a task is delegated to.

Delegated facts

By default, any fact gathered by a delegated task are assigned to the *inventory_hostname* (the current host) instead of the host which actually produced the facts (the delegated to host). The directive *delegate_facts* may be set to *True* to assign the task’s gathered facts to the delegated host instead of the current one.:

```
---
- hosts: app_servers

  tasks:
    - name: gather facts from db servers
      setup:
        delegate_to: "{{item}}"
        delegate_facts: True
        loop: "{{groups['dbservers'] }}"
```

The above will gather facts for the machines in the `dbservers` group and assign the facts to those machines and not to `app_servers`. This way you can lookup `hostvars['dbhost1']['ansible_default_ipv4']['address']` even though `dbservers` were not part of the play, or left out by using `-limit`.

Run Once

In some cases there may be a need to only run a task one time for a batch of hosts. This can be achieved by configuring “`run_once`” on a task:

```
---
# ...

tasks:

  # ...

  - command: /opt/application/upgrade_db.py
    run_once: true

# ...
```

This directive forces the task to attempt execution on the first host in the current batch and then applies all results and facts to all the hosts in the same batch.

This approach is similar to applying a conditional to a task such as:

```
- command: /opt/application/upgrade_db.py
  when: inventory_hostname == webservers[0]
```

But the results are applied to all the hosts.

Like most tasks, this can be optionally paired with “`delegate_to`” to specify an individual host to execute on:

```
- command: /opt/application/upgrade_db.py
  run_once: true
  delegate_to: web01.example.org
```

As always with delegation, the action will be executed on the delegated host, but the information is still that of the original host in the task.

注解: When used together with “serial”, tasks marked as “run_once” will be run on one host in *each* serial batch. If it’s crucial that the task is run only once regardless of “serial” mode, use **when:** `inventory_hostname == ansible_play_hosts_all[0]` construct.

注解: Any conditional (i.e *when:*) will use the variables of the ‘first host’ to decide if the task runs or not, no other hosts will be tested.

注解: If you want to avoid the default behaviour of setting the fact for all hosts, set `delegate_facts: True` for the specific task or block.

Local Playbooks

It may be useful to use a playbook locally, rather than by connecting over SSH. This can be useful for assuring the configuration of a system by putting a playbook in a crontab. This may also be used to run a playbook inside an OS installer, such as an Anaconda kickstart.

To run an entire playbook locally, just set the “hosts:” line to “hosts: 127.0.0.1” and then run the playbook like so:

```
ansible-playbook playbook.yml --connection=local
```

Alternatively, a local connection can be used in a single playbook play, even if other plays in the playbook use the default remote connection type:

```
---
- hosts: 127.0.0.1
  connection: local
```

注解: If you set the connection to local and there is no `ansible_python_interpreter` set, modules will run under `/usr/bin/python` and not under `{{ ansible_playbook_python }}`. Be sure to set `ansi-`

ble_python_interpreter: “{{ ansible_playbook_python }}” in host_vars/localhost.yml, for example. You can avoid this issue by using `local_action` or `delegate_to: localhost` instead.

Interrupt execution on any error

With the “`any_errors_fatal`” option, any failure on any host in a multi-host play will be treated as fatal and Ansible will exit as soon as all hosts in the current batch have finished the fatal task. Subsequent tasks and plays will not be executed. You can recover from what would be a fatal error by adding a rescue section to the block.

Sometimes “`serial`” execution is unsuitable; the number of hosts is unpredictable (because of dynamic inventory) and speed is crucial (simultaneous execution is required), but all tasks must be 100% successful to continue playbook execution.

For example, consider a service located in many datacenters with some load balancers to pass traffic from users to the service. There is a deploy playbook to upgrade service deb-packages. The playbook has the stages:

- disable traffic on load balancers (must be turned off simultaneously)
- gracefully stop the service
- upgrade software (this step includes tests and starting the service)
- enable traffic on the load balancers (which should be turned on simultaneously)

The service can’t be stopped with “alive” load balancers; they must be disabled first. Because of this, the second stage can’t be played if any server failed in the first stage.

For datacenter “A”, the playbook can be written this way:

```
---
- hosts: load_balancers_dc_a
  any_errors_fatal: True

  tasks:
    - name: 'shutting down datacenter [ A ]'
      command: /usr/bin/disable-dc

- hosts: frontends_dc_a

  tasks:
    - name: 'stopping service'
      command: /usr/bin/stop-software
    - name: 'updating software'
```

(下页继续)

(续上页)

```
    command: /usr/bin/upgrade-software

- hosts: load_balancers_dc_a

tasks:
  - name: 'Starting datacenter [ A ]'
    command: /usr/bin/enable-dc
```

In this example Ansible will start the software upgrade on the front ends only if all of the load balancers are successfully disabled.

参见:

Intro to Playbooks An introduction to playbooks

[Ansible Examples on GitHub](#) Many examples of full-stack deployments

[User Mailing List](#) Have a question? Stop by the google group!

[irc.freenode.net](#) #ansible IRC chat channel

Setting the Environment (and Working With Proxies)

1.1 新版功能.

The `environment` keyword allows you to set an environment variable for the action to be taken on the remote target. For example, it is quite possible that you may need to set a proxy for a task that does http requests. Or maybe a utility or script that are called may also need certain environment variables set to run properly.

Here is an example:

```
- hosts: all
  remote_user: root

tasks:

  - name: Install cobbler
    package:
      name: cobbler
      state: present
    environment:
      http_proxy: http://proxy.example.com:8080
```

注解:

environment: does not affect Ansible itself, **ONLY** the context of the specific task action and this does not i
Ansible's own configuration settings nor the execution of any other plugins, including lookups, filters,
and so on.

The environment can also be stored in a variable, and accessed like so:

```
- hosts: all
  remote_user: root

  # here we make a variable named "proxy_env" that is a dictionary
  vars:
    proxy_env:
      http_proxy: http://proxy.example.com:8080

  tasks:

    - name: Install cobbler
      package:
        name: cobbler
        state: present
        environment: "{{ proxy_env }}"
```

You can also use it at a play level:

```
- hosts: testhost

  roles:
    - php
    - nginx

  environment:
    http_proxy: http://proxy.example.com:8080
```

While just proxy settings were shown above, any number of settings can be supplied. The most logical place to define an environment hash might be a group_vars file, like so:

```
---
# file: group_vars/boston

ntp_server: ntp.bos.example.com
backup: bak.bos.example.com
proxy_env:
```

(下页继续)

(续上页)

```
http_proxy: http://proxy.bos.example.com:8080
https_proxy: http://proxy.bos.example.com:8080
```

Working With Language-Specific Version Managers

Some language-specific version managers (such as `rvm` and `nvm`) require environment variables be set while these tools are in use. When using these tools manually, they usually require sourcing some environment variables via a script or lines added to your shell configuration file. In Ansible, you can instead use the `environment` directive:

```
---
### A playbook demonstrating a common npm workflow:
# - Check for package.json in the application directory
# - If package.json exists:
#   * Run npm prune
#   * Run npm install

- hosts: application
  become: false

  vars:
    node_app_dir: /var/local/my_node_app

  environment:
    NVM_DIR: /var/local/nvm
    PATH: /var/local/nvm/versions/node/v4.2.1/bin:{{ ansible_env.PATH }}

  tasks:
    - name: check for package.json
      stat:
        path: '{{ node_app_dir }}/package.json'
      register: packagejson

    - name: npm prune
      command: npm prune
      args:
        chdir: '{{ node_app_dir }}'
      when: packagejson.stat.exists
```

(下页继续)

(续上页)

```
- name: npm install
  npm:
    path: '{{ node_app_dir }}'
    when: packagejson.stat.exists
```

注解: `ansible_env`: is normally populated by fact gathering (M(gather_facts)) and the value of the variables depends on the user that did the gathering action. If you change `remote_user`/`become_user` you might end up using the wrong values for those variables.

You might also want to simply specify the environment for a single task:

```
---
- name: install ruby 2.3.1
  command: rbenv install '{{ rbenv_ruby_version }}'
  args:
    creates: '{{ rbenv_root }}/versions/{{ rbenv_ruby_version }}/bin/ruby'
  vars:
    rbenv_root: /usr/local/rbenv
    rbenv_ruby_version: 2.3.1
  environment:
    CONFIGURE_OPTS: '--disable-install-doc'
    RBENV_ROOT: '{{ rbenv_root }}'
    PATH: '{{ rbenv_root }}/bin:{{ rbenv_root }}/shims:{{ rbenv_plugins }}/ruby-build/
    ↪bin:{{ ansible_env.PATH }}
```

参见:

Intro to Playbooks An introduction to playbooks

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

Error handling in playbooks

When Ansible receives a non-zero return code from a command or a failure from a module, by default it stops executing on that host and continues on other hosts. However, in some circumstances you may want different behavior. Sometimes a non-zero return code indicates success. Sometimes you want a failure on one host to stop execution on all hosts. Ansible provides tools and settings to handle these situations and help you get the behavior, output, and reporting you want.

- *Ignoring failed commands*
- *Ignoring unreachable host errors*
- *Resetting unreachable hosts*
- *Handlers and failure*
- *Defining failure*
- *Defining “changed”*
- *Ensuring success for command and shell*
- *Aborting a play on all hosts*
- *Controlling errors in blocks*

Ignoring failed commands

By default Ansible stops executing tasks on a host when a task fails on that host. You can use `ignore_errors` to continue on in spite of the failure:

```
- name: this will not count as a failure
  command: /bin/false
  ignore_errors: yes
```

The `ignore_errors` directive only works when the task is able to run and returns a value of ‘failed’. It will not make Ansible ignore undefined variable errors, connection failures, execution issues (for example, missing packages), or syntax errors.

Ignoring unreachable host errors

2.7 新版功能.

You may ignore task failure due to the host instance being ‘UNREACHABLE’ with the `ignore_unreachable` keyword. Note that task errors are what’s being ignored, not the unreachable host.

Here’s an example explaining the behavior for an unreachable host at the task level:

```
- name: this executes, fails, and the failure is ignored
  command: /bin/true
  ignore_unreachable: yes

- name: this executes, fails, and ends the play for this host
  command: /bin/true
```

And at the playbook level:

```
- hosts: all
  ignore_unreachable: yes
  tasks:
    - name: this executes, fails, and the failure is ignored
      command: /bin/true

    - name: this executes, fails, and ends the play for this host
      command: /bin/true
      ignore_unreachable: no
```

Resetting unreachable hosts

If Ansible cannot connect to a host, it marks that host as ‘UNREACHABLE’ and removes it from the list of active hosts for the run. You can use *meta: clear_host_errors* to reactivate all hosts, so subsequent tasks can try to reach them again.

Handlers and failure

Ansible runs *handlers* at the end of each play. If a task notifies a handler but another task fails later in the play, by default the handler does *not* run on that host, which may leave the host in an unexpected state. For example, a task could update a configuration file and notify a handler to restart some service. If a task later in the same play fails, the configuration file might be changed but the service will not be restarted.

You can change this behavior with the `--force-handlers` command-line option, by including `force_handlers: True` in a play, or by adding `force_handlers = True` to `ansible.cfg`. When handlers are forced, Ansible will run all notified handlers on all hosts, even hosts with failed tasks. (Note that certain errors could still prevent the handler from running, such as a host becoming unreachable.)

Defining failure

Ansible lets you define what “failure” means in each task using the `failed_when` conditional. As with all conditionals in Ansible, lists of multiple `failed_when` conditions are joined with an implicit `and`, meaning the task only fails when *all* conditions are met. If you want to trigger a failure when any of the conditions is met, you must define the conditions in a string with an explicit `or` operator.

You may check for failure by searching for a word or phrase in the output of a command:

```
- name: Fail task when the command error output prints FAILED
  command: /usr/bin/example-command -x -y -z
```

(下页继续)

(续上页)

```
register: command_result
failed_when: "'FAILED' in command_result.stderr"
```

or based on the return code:

```
- name: Fail task when both files are identical
  raw: diff foo/file1 bar/file2
  register: diff_cmd
  failed_when: diff_cmd.rc == 0 or diff_cmd.rc >= 2
```

You can also combine multiple conditions for failure. This task will fail if both conditions are true:

```
- name: Check if a file exists in temp and fail task if it does
  command: ls /tmp/this_should_not_be_here
  register: result
  failed_when:
    - result.rc == 0
    - '"No such" not in result.stdout'
```

If you want the task to fail when only one condition is satisfied, change the `failed_when` definition to:

```
failed_when: result.rc == 0 or "No such" not in result.stdout
```

If you have too many conditions to fit neatly into one line, you can split it into a multi-line yaml value with `>`:

```
- name: example of many failed_when conditions with OR
  shell: "./myBinary"
  register: ret
  failed_when: >
    ("No such file or directory" in ret.stdout) or
    (ret.stderr != '') or
    (ret.rc == 10)
```

Defining “changed”

Ansible lets you define when a particular task has “changed” a remote node using the `changed_when` conditional. This lets you determine, based on return codes or output, whether a change should be reported in Ansible statistics and whether a handler should be triggered or not. As with all conditionals in Ansible, lists of multiple `changed_when` conditions are joined with an implicit `and`, meaning the task only reports a change when *all* conditions are met. If you want to report a change when any of the conditions is met, you

must define the conditions in a string with an explicit `or` operator. For example:

```
tasks:

- shell: /usr/bin/billybass --mode="take me to the river"
  register: bass_result
  changed_when: "bass_result.rc != 2"

# this will never report 'changed' status
- shell: wall 'beep'
  changed_when: False
```

You can also combine multiple conditions to override “changed” result:

```
- command: /bin/fake_command
  register: result
  ignore_errors: True
  changed_when:
    - '"ERROR" in result.stderr'
    - result.rc == 2
```

See *Defining failure* for more conditional syntax examples.

Ensuring success for command and shell

The `command` and `shell` modules care about return codes, so if you have a command whose successful exit code is not zero, you may wish to do this:

```
tasks:

- name: run this command and ignore the result
  shell: /usr/bin/somecommand || /bin/true
```

Aborting a play on all hosts

Sometimes you want a failure on a single host to abort the entire play on all hosts. If you set `any_errors_fatal` and a task returns an error, Ansible lets all hosts in the current batch finish the fatal task and then stops executing the play on all hosts. You can set `any_errors_fatal` at the play or block level:

```
- hosts: somehosts
  any_errors_fatal: true
```

(下页继续)

(续上页)

```
roles:
  - myrole

- hosts: somehosts
  tasks:
    - block:
      - include_tasks: mytasks.yml
      any_errors_fatal: true
```

For finer-grained control, you can use `max_fail_percentage` to abort the run after a given percentage of hosts has failed.

Controlling errors in blocks

You can also use blocks to define responses to task errors. This approach is similar to exception handling in many programming languages. See *Handling errors with blocks* for details and examples.

参见:

Intro to Playbooks An introduction to playbooks

Tips and tricks Best practices in playbooks

Conditionals Conditional statements in playbooks

Using Variables All about variables

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

Advanced Syntax

The advanced YAML syntax examples on this page give you more control over the data placed in YAML files used by Ansible. You can find additional information about Python-specific YAML in the official [PyYAML Documentation](#).

- *Unsafe or raw strings*
- *YAML anchors and aliases: sharing variable values*

Unsafe or raw strings

When handling values returned by lookup plugins, Ansible uses a data type called **unsafe** to block templating. Marking data as unsafe prevents malicious users from abusing Jinja2 templates to execute arbitrary code on target machines. The Ansible implementation ensures that unsafe values are never templated. It is more comprehensive than escaping Jinja2 with `{% raw %} ... {% endraw %}` tags.

You can use the same **unsafe** data type in variables you define, to prevent templating errors and information disclosure. You can mark values supplied by *vars_prompts* as unsafe. You can also use **unsafe** in playbooks. The most common use cases include passwords that allow special characters like `{` or `%`, and JSON arguments that look like templates but should not be templated. For example:

```
---
mypassword: !unsafe 234%234{435lkj{{lkjsdf
```

In a playbook:

```
---
hosts: all
vars:
    my_unsafe_variable: !unsafe 'unsafe % value'
tasks:
    ...
```

For complex variables such as hashes or arrays, use **!unsafe** on the individual elements:

```
---
my_unsafe_array:
    - !unsafe 'unsafe element'
    - 'safe element'

my_unsafe_hash:
    unsafe_key: !unsafe 'unsafe value'
```

YAML anchors and aliases: sharing variable values

YAML [anchors and aliases](#) help you define, maintain, and use shared variable values in a flexible way. You define an anchor with `&`, then refer to it using an alias, denoted with `*`. Here's an example that sets three values with an anchor, uses two of those values with an alias, and overrides the third value:

```
---
...
```

(下页继续)

(续上页)

```
vars:
  app1:
    jvm: &jvm_opts
      opts: '-Xms1G -Xmx2G'
      port: 1000
      path: /usr/lib/app1
  app2:
    jvm:
      <<: *jvm_opts
      path: /usr/lib/app2
...
```

Here, `app1` and `app2` share the values for `opts` and `port` using the anchor `&jvm_opts` and the alias `*jvm_opts`. The value for `path` is merged by `<<` or [merge operator](#).

anchors and aliases also let you share complex sets of variable values, including nested variables. If you have one variable value that includes another variable value, you can define them separately:

```
vars:
  webapp_version: 1.0
  webapp_custom_name: ToDo_App-1.0
```

This is inefficient and, at scale, means more maintenance. To incorporate the version value in the name, you can use an anchor in `app_version` and an alias in `custom_name`:

```
vars:
  webapp:
    version: &my_version 1.0
    custom_name:
      - "ToDo_App"
      - *my_version
```

Now, you can re-use the value of `app_version` within the value of `custom_name` and use the output in a template:

```
---
- name: Using values nested inside dictionary
  hosts: localhost
  vars:
    webapp:
      version: &my_version 1.0
      custom_name:
```

(下页继续)

(续上页)

```
- "ToDo_App"
- *my_version

tasks:
- name: Using Anchor value
  debug:
    msg: My app is called "{{ webapp.custom_name | join('-') }}".
```

You’ve anchored the value of `version` with the `&my_version` anchor, and re-used it with the `*my_version` alias. Anchors and aliases let you access nested values inside dictionaries.

参见:

Using Variables All about variables

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

Working With Plugins

Plugins are pieces of code that augment Ansible’s core functionality. Ansible uses a plugin architecture to enable a rich, flexible and expandable feature set.

Ansible ships with a number of handy plugins, and you can easily write your own.

This section covers the various types of plugins that are included with Ansible:

Action Plugins

- *Enabling action plugins*
- *Using action plugins*
- *Plugin list*

Action plugins act in conjunction with *modules* to execute the actions required by playbook tasks. They usually execute automatically in the background doing prerequisite work before modules execute.

The ‘normal’ action plugin is used for modules that do not already have an action plugin.

Enabling action plugins

You can enable a custom action plugin by either dropping it into the `action_plugins` directory adjacent to your play, inside a role, or by putting it in one of the action plugin directory sources configured in `ansible.cfg`.

Using action plugins

Action plugin are executed by default when an associated module is used; no action is required.

Plugin list

You cannot list action plugins directly, they show up as their counterpart modules:

Use `ansible-doc -l` to see the list of available modules. Use `ansible-doc <name>` to see specific documentation and examples, this should note if the module has a corresponding action plugin.

参见:

Cache Plugins Ansible Cache plugins

Callback Plugins Ansible callback plugins

Connection Plugins Ansible connection plugins

Inventory Plugins Ansible inventory plugins

Shell Plugins Ansible Shell plugins

Strategy Plugins Ansible Strategy plugins

Vars Plugins Ansible Vars plugins

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

Become Plugins

- *Enabling Become Plugins*
- *Using Become Plugins*
- *Plugin List*

2.8 新版功能.

Become plugins work to ensure that Ansible can use certain privilege escalation systems when running the basic commands to work with the target machine as well as the modules required to execute the tasks specified in the play.

These utilities (`sudo`, `su`, `doas`, etc) generally let you ‘become’ another user to execute a command with the permissions of that user.

Enabling Become Plugins

The become plugins shipped with Ansible are already enabled. Custom plugins can be added by placing them into a `become_plugins` directory adjacent to your play, inside a role, or by placing them in one of the become plugin directory sources configured in `ansible.cfg`.

Using Become Plugins

In addition to the default configuration settings in `ansible_configuration_settings` or the `--become-method` command line option, you can use the `become_method` keyword in a play or, if you need to be ‘host specific’, the connection variable `ansible_become_method` to select the plugin to use.

You can further control the settings for each plugin via other configuration options detailed in the plugin themselves (linked below).

Plugin List

You can use `ansible-doc -t become -l` to see the list of available plugins. Use `ansible-doc -t become <plugin name>` to see specific documentation and examples.

参见:

Intro to Playbooks An introduction to playbooks

Inventory Plugins Ansible inventory plugins

Callback Plugins Ansible callback plugins

Filters Jinja2 filter plugins

Tests Jinja2 test plugins

Lookups Jinja2 lookup plugins

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

Cache Plugins

- *Enabling Fact Cache Plugins*
- *Enabling Inventory Cache Plugins*
- *Using Cache Plugins*
- *Plugin List*

Cache plugins allow Ansible to store gathered facts or inventory source data without the performance hit of retrieving them from source.

The default cache plugin is the memory plugin, which only caches the data for the current execution of Ansible. Other plugins with persistent storage are available to allow caching the data across runs. Some of these cache plugins write to files, others write to databases.

You can use different cache plugins for inventory and facts. If you enable inventory caching without setting an inventory-specific cache plugin, Ansible uses the fact cache plugin for both facts and inventory.

Enabling Fact Cache Plugins

Fact caching is always enabled. However, only one fact cache plugin can be active at a time. You can select the cache plugin to use for fact caching in the Ansible configuration, either with an environment variable:

```
export ANSIBLE_CACHE_PLUGIN=jsonfile
```

or in the `ansible.cfg` file:

```
[defaults]
fact_caching=redis
```

If the cache plugin is in a collection use the fully qualified name:

```
[defaults]
fact_caching = namespace.collection_name.cache_plugin_name
```

To enable a custom cache plugin, save it in a `cache_plugins` directory adjacent to your play, inside a role, or in one of the directory sources configured in `ansible.cfg`.

You also need to configure other settings specific to each plugin. Consult the individual plugin documentation or the Ansible configuration for more details.

Enabling Inventory Cache Plugins

Inventory caching is disabled by default. To cache inventory data, you must enable inventory caching and then select the specific cache plugin you want to use. Not all inventory plugins support caching, so check the documentation for the inventory plugin(s) you want to use. You can enable inventory caching with an environment variable:

```
export ANSIBLE_INVENTORY_CACHE=True
```

or in the `ansible.cfg` file:

```
[inventory]
cache=True
```

or if the inventory plugin accepts a YAML configuration source, in the configuration file:

```
# dev.aws_ec2.yaml
plugin: aws_ec2
cache: True
```

Only one inventory cache plugin can be active at a time. You can set it with an environment variable:

```
export ANSIBLE_INVENTORY_CACHE_PLUGIN=jsonfile
```

or in the ansible.cfg file:

```
[inventory]
cache_plugin=jsonfile
```

or if the inventory plugin accepts a YAML configuration source, in the configuration file:

```
# dev.aws_ec2.yaml
plugin: aws_ec2
cache_plugin: jsonfile
```

To cache inventory with a custom plugin in your plugin path, follow the *developer guide on cache plugins*.

You can use any cache plugin shipped with Ansible to cache inventory, but you cannot use a cache plugin inside a collection. If you enable caching for inventory plugins without selecting an inventory-specific cache plugin, Ansible falls back to caching inventory with the fact cache plugin you configured. Consult the individual inventory plugin documentation or the Ansible configuration for more details.

Using Cache Plugins

Cache plugins are used automatically once they are enabled.

Plugin List

You can use `ansible-doc -t cache -l` to see the list of available plugins. Use `ansible-doc -t cache <plugin name>` to see specific documentation and examples.

参见:

Action Plugins Ansible Action plugins

Callback Plugins Ansible callback plugins

Connection Plugins Ansible connection plugins

Inventory Plugins Ansible inventory plugins

Shell Plugins Ansible Shell plugins

Strategy Plugins Ansible Strategy plugins

Vars Plugins Ansible Vars plugins

User Mailing List Have a question? Stop by the google group!

webchat.freenode.net #ansible IRC chat channel

Callback Plugins

- *Example callback plugins*
- *Enabling callback plugins*
- *Setting a callback plugin for `ansible-playbook`*
- *Setting a callback plugin for `ad-hoc` commands*
- *Plugin list*

Callback plugins enable adding new behaviors to Ansible when responding to events. By default, callback plugins control most of the output you see when running the command line programs, but can also be used to add additional output, integrate with other tools and marshall the events to a storage backend.

Example callback plugins

The `log_plays` callback is an example of how to record playbook events to a log file, and the `mail` callback sends email on playbook failures.

The `say` callback responds with computer synthesized speech in relation to playbook events.

Enabling callback plugins

You can activate a custom callback by either dropping it into a `callback_plugins` directory adjacent to your play, inside a role, or by putting it in one of the callback directory sources configured in `ansible.cfg`.

Plugins are loaded in alphanumeric order. For example, a plugin implemented in a file named `1_first.py` would run before a plugin file named `2_second.py`.

Most callbacks shipped with Ansible are disabled by default and need to be whitelisted in your `ansible.cfg` file in order to function. For example:

```
#callback_whitelist = timer, mail, profile_roles, collection_namespace.collection_name.  
↪ custom_callback
```

Setting a callback plugin for ansible-playbook

You can only have one plugin be the main manager of your console output. If you want to replace the default, you should define `CALLBACK_TYPE = stdout` in the subclass and then configure the stdout plugin in `ansible.cfg`. For example:

```
stdout_callback = dense
```

or for my custom callback:

```
stdout_callback = mycallback
```

This only affects ansible-playbook by default.

Setting a callback plugin for ad-hoc commands

The ansible ad hoc command specifically uses a different callback plugin for stdout, so there is an extra setting in `ansible_configuration_settings` you need to add to use the stdout callback defined above:

```
[defaults]  
bin_ansible_callbacks=True
```

You can also set this as an environment variable:

```
export ANSIBLE_LOAD_CALLBACK_PLUGINS=1
```

Plugin list

You can use `ansible-doc -t callback -l` to see the list of available plugins. Use `ansible-doc -t callback <plugin name>` to see specific documents and examples.

参见:

Action Plugins Ansible Action plugins

Cache Plugins Ansible cache plugins

Connection Plugins Ansible connection plugins

Inventory Plugins Ansible inventory plugins

Shell Plugins Ansible Shell plugins

Strategy Plugins Ansible Strategy plugins

Vars Plugins Ansible Vars plugins

User Mailing List Have a question? Stop by the google group!

webchat.freenode.net #ansible IRC chat channel

Cliconf Plugins

- *Adding cliconf plugins*
- *Using cliconf plugins*
- *Plugin list*

Cliconf plugins are abstractions over the CLI interface to network devices. They provide a standard interface for Ansible to execute tasks on those network devices.

These plugins generally correspond one-to-one to network device platforms. The appropriate cliconf plugin will thus be automatically loaded based on the `ansible_network_os` variable.

Adding cliconf plugins

You can extend Ansible to support other network devices by dropping a custom plugin into the `cliconf_plugins` directory.

Using cliconf plugins

The cliconf plugin to use is determined automatically from the `ansible_network_os` variable. There should be no reason to override this functionality.

Most cliconf plugins can operate without configuration. A few have additional options that can be set to impact how tasks are translated into CLI commands.

Plugins are self-documenting. Each plugin should document its configuration options.

Plugin list

You can use `ansible-doc -t cliconf -l` to see the list of available plugins. Use `ansible-doc -t cliconf <plugin name>` to see detailed documentation and examples.

参见:

Ansible for Network Automation An overview of using Ansible to automate networking devices.

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible-network IRC chat channel

Connection Plugins

- *ssh plugins*
- *Adding connection plugins*
- *Using connection plugins*
- *Plugin List*

Connection plugins allow Ansible to connect to the target hosts so it can execute tasks on them. Ansible ships with many connection plugins, but only one can be used per host at a time.

By default, Ansible ships with several plugins. The most commonly used are the paramiko SSH, native ssh (just called ssh), and local connection types. All of these can be used in playbooks and with `/usr/bin/ansible` to decide how you want to talk to remote machines.

The basics of these connection types are covered in the *getting started* section.

ssh plugins

Because ssh is the default protocol used in system administration and the protocol most used in Ansible, ssh options are included in the command line tools. See `ansible-playbook` for more details.

Adding connection plugins

You can extend Ansible to support other transports (such as SNMP or message bus) by dropping a custom plugin into the `connection_plugins` directory.

Using connection plugins

You can set the connection plugin globally via configuration, at the command line (`-c, --connection`), as a keyword in your play, or by setting a *variable*, most often in your inventory. For example, for Windows machines you might want to set the winrm plugin as an inventory variable.

Most connection plugins can operate with minimal configuration. By default they use the inventory hostname and defaults to find the target host.

Plugins are self-documenting. Each plugin should document its configuration options. The following are connection variables common to most connection plugins:

ansible_host The name of the host to connect to, if different from the *inventory* hostname.

ansible_port The ssh port number, for ssh and paramiko_ssh it defaults to 22.

ansible_user The default user name to use for log in. Most plugins default to the ‘current user running Ansible’ .

Each plugin might also have a specific version of a variable that overrides the general version. For example, *ansible_ssh_host* for the ssh plugin.

Plugin List

You can use `ansible-doc -t connection -l` to see the list of available plugins. Use `ansible-doc -t connection <plugin name>` to see detailed documentation and examples.

参见:

Working with Playbooks An introduction to playbooks

Callback Plugins Ansible callback plugins

Filters Jinja2 filter plugins

Tests Jinja2 test plugins

Lookups Jinja2 lookup plugins

Vars Plugins Ansible vars plugins

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

Httpapi Plugins

- *Adding httpapi plugins*
- *Using httpapi plugins*
- *Plugin List*

Httpapi plugins tell Ansible how to interact with a remote device’ s HTTP-based API and execute tasks on the device.

Each plugin represents a particular dialect of API. Some are platform-specific (Arista eAPI, Cisco NXAPI), while others might be usable on a variety of platforms (RESTCONF).

Adding httpapi plugins

You can extend Ansible to support other APIs by dropping a custom plugin into the `httpapi_plugins` directory. See *Developing httpapi plugins* for details.

Using httpapi plugins

The httpapi plugin to use is determined automatically from the `ansible_network_os` variable.

Most httpapi plugins can operate without configuration. Additional options may be defined by each plugin.

Plugins are self-documenting. Each plugin should document its configuration options.

The following sample playbook shows the httpapi plugin for an Arista network device, assuming an inventory variable set as `ansible_network_os=eos` for the httpapi plugin to trigger off:

```
- hosts: leaf01
  connection: httpapi
  gather_facts: false
  tasks:

    - name: type a simple arista command
      eos_command:
        commands:
          - show version | json
      register: command_output

    - name: print command output to terminal window
      debug:
        var: command_output.stdout[0]["version"]
```

See the full working example at <https://github.com/network-automation/httpapi>.

Plugin List

You can use `ansible-doc -t httpapi -l` to see the list of available plugins. Use `ansible-doc -t httpapi <plugin name>` to see detailed documentation and examples.

参见:

Ansible for Network Automation An overview of using Ansible to automate networking devices.

Developing network modules How to develop network modules.

User Mailing List Have a question? Stop by the google group!

[irc.freenode.net](#) #ansible-network IRC chat channel

Inventory Plugins

- *Enabling inventory plugins*
- *Using inventory plugins*
- *Plugin List*

Inventory plugins allow users to point at data sources to compile the inventory of hosts that Ansible uses to target tasks, either via the `-i /path/to/file` and/or `-i 'host1, host2'` command line parameters or from other configuration sources.

Enabling inventory plugins

Most inventory plugins shipped with Ansible are disabled by default and need to be whitelisted in your `ansible.cfg` file in order to function. This is how the default whitelist looks in the config file that ships with Ansible:

```
[inventory]
enable_plugins = host_list, script, auto, yaml, ini, toml
```

This list also establishes the order in which each plugin tries to parse an inventory source. Any plugins left out of the list will not be considered, so you can ‘optimize’ your inventory loading by minimizing it to what you actually use. For example:

```
[inventory]
enable_plugins = advanced_host_list, constructed, yaml
```

The `auto` inventory plugin can be used to automatically determines which inventory plugin to use for a YAML configuration file. It can also be used for inventory plugins in a collection.

To whitelist specific inventory plugins in a collection you need to use the fully qualified name:

```
[inventory]
enable_plugins = namespace.collection_name.inventory_plugin_name
```

Using inventory plugins

The only requirement for using an inventory plugin after it is enabled is to provide an inventory source to parse. Ansible will try to use the list of enabled inventory plugins, in order, against each inventory source

provided. Once an inventory plugin succeeds at parsing a source, any remaining inventory plugins will be skipped for that source.

To start using an inventory plugin with a YAML configuration source, create a file with the accepted filename schema for the plugin in question, then add `plugin: plugin_name`. Each plugin documents any naming restrictions. For example, the `aws_ec2` inventory plugin has to end with `aws_ec2.(yaml|yml)`

```
# demo.aws_ec2.yml
plugin: aws_ec2
```

Or for the `openstack` plugin the file has to be called `clouds.yml` or `openstack.(yaml|yml)`:

```
# clouds.yml or openstack.(yaml|yml)
plugin: openstack
```

To use a plugin in a collection provide the fully qualified name:

```
plugin: namespace.collection_name.inventory_plugin_name
```

The `auto` inventory plugin is enabled by default and works by using the `plugin` field to indicate the plugin that should attempt to parse it. You can configure the whitelist/precedence of inventory plugins used to parse source using the `ansible.cfg` ['inventory'] `enable_plugins` list. After enabling the plugin and providing any required options, you can view the populated inventory with `ansible-inventory -i demo.aws_ec2.yml --graph`:

```
@all:
  |--@aws_ec2:
  |   |--ec2-12-345-678-901.compute-1.amazonaws.com
  |   |--ec2-98-765-432-10.compute-1.amazonaws.com
  |--@ungrouped:
```

If you are using an inventory plugin in a playbook-adjacent collection and want to test your setup with `ansible-inventory`, you will need to use the `--playbook-dir` flag.

You can set the default inventory path (via `inventory` in the `ansible.cfg` [defaults] section or the `ANSIBLE_INVENTORY` environment variable) to your inventory source(s). Now running `ansible-inventory --graph` should yield the same output as when you passed your YAML configuration source(s) directly. You can add custom inventory plugins to your plugin path to use in the same way.

Your inventory source might be a directory of inventory configuration files. The constructed inventory plugin only operates on those hosts already in inventory, so you may want the constructed inventory configuration parsed at a particular point (such as last). Ansible parses the directory recursively, alphabetically. You cannot configure the parsing approach, so name your files to make it work predictably. Inventory plugins that extend constructed features directly can work around that restriction by adding constructed options in addition to the inventory plugin options. Otherwise, you can use `-i` with multiple sources to impose a

specific order, e.g. `-i demo.aws_ec2.yml -i clouds.yml -i constructed.yml`.

You can create dynamic groups using host variables with the constructed `keyed_groups` option. The option `groups` can also be used to create groups and `compose` creates and modifies host variables. Here is an `aws_ec2` example utilizing constructed features:

```
# demo.aws_ec2.yml
plugin: aws_ec2
regions:
  - us-east-1
  - us-east-2
keyed_groups:
  # add hosts to tag_Name_value groups for each aws_ec2 host's tags.Name variable
  - key: tags.Name
    prefix: tag_Name_
    separator: ""
groups:
  # add hosts to the group development if any of the dictionary's keys or values is the
  ↪word 'devel'
  development: "'devel' in (tags|list)"
compose:
  # set the ansible_host variable to connect with the private IP address without
  ↪changing the hostname
  ansible_host: private_ip_address
```

Now the output of `ansible-inventory -i demo.aws_ec2.yml --graph`:

```
@all:
  |--@aws_ec2:
  |   |--ec2-12-345-678-901.compute-1.amazonaws.com
  |   |--ec2-98-765-432-10.compute-1.amazonaws.com
  |   |--...
  |--@development:
  |   |--ec2-12-345-678-901.compute-1.amazonaws.com
  |   |--ec2-98-765-432-10.compute-1.amazonaws.com
  |--@tag_Name_ECS_Instance:
  |   |--ec2-98-765-432-10.compute-1.amazonaws.com
  |--@tag_Name_Test_Server:
  |   |--ec2-12-345-678-901.compute-1.amazonaws.com
  |--@ungrouped
```

If a host does not have the variables in the configuration above (i.e. `tags.Name`, `tags`, `private_ip_address`), the host will not be added to groups other than those that the inventory plugin creates and the `ansible_host`

host variable will not be modified.

If an inventory plugin supports caching, you can enable and set caching options for an individual YAML configuration source or for multiple inventory sources using environment variables or Ansible configuration files. If you enable caching for an inventory plugin without providing inventory-specific caching options, the inventory plugin will use fact-caching options. Here is an example of enabling caching for an individual YAML configuration file:

```
# demo.aws_ec2.yml
plugin: aws_ec2
cache: yes
cache_plugin: jsonfile
cache_timeout: 7200
cache_connection: /tmp/aws_inventory
cache_prefix: aws_ec2
```

Here is an example of setting inventory caching with some fact caching defaults for the cache plugin used and the timeout in an `ansible.cfg` file:

```
[defaults]
fact_caching = jsonfile
fact_caching_connection = /tmp/ansible_facts
cache_timeout = 3600

[inventory]
cache = yes
cache_connection = /tmp/ansible_inventory
```

Besides cache plugins shipped with Ansible, cache plugins eligible for caching inventory can also reside in a custom cache plugin path. Cache plugins in collections are not supported yet for inventory.

Plugin List

You can use `ansible-doc -t inventory -l` to see the list of available plugins. Use `ansible-doc -t inventory <plugin name>` to see plugin-specific documentation and examples.

参见:

Intro to Playbooks An introduction to playbooks

Callback Plugins Ansible callback plugins

Connection Plugins Ansible connection plugins

Filters Jinja2 filter plugins

Tests Jinja2 test plugins

Lookups Jinja2 lookup plugins

Vars Plugins Ansible vars plugins

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

Lookup Plugins

- *Enabling lookup plugins*
- *Using lookup plugins*
- *Invoking lookup plugins with *query**
- *Plugin list*

Lookup plugins allow Ansible to access data from outside sources. This can include reading the filesystem in addition to contacting external datastores and services. Like all templating, these plugins are evaluated on the Ansible control machine, not on the target/remote.

The data returned by a lookup plugin is made available using the standard templating system in Ansible, and are typically used to load variables or templates with information from those systems.

Lookups are an Ansible-specific extension to the Jinja2 templating language.

注解:

- Lookups are executed with a working directory relative to the role or play, as opposed to local tasks, which are executed relative the executed script.
- Since Ansible version 1.9, you can pass `wantlist=True` to lookups to use in Jinja2 template “for” loops.
- Lookup plugins are an advanced feature; to best leverage them you should have a good working knowledge of how to use Ansible plays.

警告:

- Some lookups pass arguments to a shell. When using variables from a remote/untrusted source, use the `/quote` filter to ensure safe usage.

Enabling lookup plugins

You can activate a custom lookup by either dropping it into a `lookup_plugins` directory adjacent to your play, inside a role, or by putting it in one of the lookup directory sources configured in `ansible.cfg`.

Using lookup plugins

Lookup plugins can be used anywhere you can use templating in Ansible: in a play, in variables file, or in a Jinja2 template for the template module.

```
vars:
  file_contents: "{{lookup('file', 'path/to/file.txt')}}"
```

Lookups are an integral part of loops. Wherever you see `with_`, the part after the underscore is the name of a lookup. This is also the reason most lookups output lists and take lists as input; for example, `with_items` uses the `items` lookup:

```
tasks:
- name: count to 3
  debug: msg="{{item}}"
  with_items: [1, 2, 3]
```

You can combine lookups with *Filters*, *Tests* and even each other to do some complex data generation and manipulation. For example:

```
tasks:
- name: valid but useless and over complicated chained lookups and filters
  debug: msg="find the answer here:\n{{ lookup('url', 'https://google.com/search?q=' +
↪ item|urlencode)|join(' ') }}"
  with_nested:
    - "{{lookup('consul_kv', 'bcs/' + lookup('file', '/the/question') + ',
↪ host=localhost, port=2000')|shuffle}}"
    - "{{lookup('sequence', 'end=42 start=2 step=2')|map('log', 4)|list}}"
    - ['a', 'c', 'd', 'c']
```

2.6 新版功能.

You can now control how errors behave in all lookup plugins by setting `errors` to `ignore`, `warn`, or `strict`. The default setting is `strict`, which causes the task to fail. For example:

To ignore errors:

```
- name: file doesnt exist, but i dont care .. file plugin itself warns anyways ...
  debug: msg="{{ lookup('file', '/idontexist', errors='ignore') }}"
```

```
[WARNING]: Unable to find '/idontexist' in expected paths (use -vvvvv to see paths)

ok: [localhost] => {
    "msg": ""
}
```

To get a warning instead of a failure:

```
- name: file doesnt exist, let me know, but continue
  debug: msg="{{ lookup('file', '/idontexist', errors='warn') }}"
```

```
[WARNING]: Unable to find '/idontexist' in expected paths (use -vvvvv to see paths)

[WARNING]: An unhandled exception occurred while running the lookup plugin 'file'. Error
↳ was a <class 'ansible.errors.AnsibleError'>, original message: could not locate file
↳ in lookup: /idontexist

ok: [localhost] => {
    "msg": ""
}
```

Fatal error (the default):

```
- name: file doesnt exist, FAIL (this is the default)
  debug: msg="{{ lookup('file', '/idontexist', errors='strict') }}"
```

```
[WARNING]: Unable to find '/idontexist' in expected paths (use -vvvvv to see paths)

fatal: [localhost]: FAILED! => {"msg": "An unhandled exception occurred while running
↳ the lookup plugin 'file'. Error was a <class 'ansible.errors.AnsibleError'>, original
↳ message: could not locate file in lookup: /idontexist"}
```

Invoking lookup plugins with query

2.5 新版功能.

In Ansible 2.5, a new jinja2 function called `query` was added for invoking lookup plugins. The difference between `lookup` and `query` is largely that `query` will always return a list. The default behavior of `lookup` is to return a string of comma separated values. `lookup` can be explicitly configured to return a list using `wantlist=True`.

This was done primarily to provide an easier and more consistent interface for interacting with the new `loop` keyword, while maintaining backwards compatibility with other uses of `lookup`.

The following examples are equivalent:

```
lookup('dict', dict_variable, wantlist=True)

query('dict', dict_variable)
```

As demonstrated above the behavior of `wantlist=True` is implicit when using `query`.

Additionally, `q` was introduced as a shortform of `query`:

```
q('dict', dict_variable)
```

Plugin list

You can use `ansible-doc -t lookup -l` to see the list of available plugins. Use `ansible-doc -t lookup <plugin name>` to see specific documents and examples.

参见:

Intro to Playbooks An introduction to playbooks

Inventory Plugins Ansible inventory plugins

Callback Plugins Ansible callback plugins

Filters Jinja2 filter plugins

Tests Jinja2 test plugins

Lookups Jinja2 lookup plugins

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

Netconf Plugins

- *Adding netconf plugins*
- *Using netconf plugins*
- *Plugin list*

Netconf plugins are abstractions over the Netconf interface to network devices. They provide a standard interface for Ansible to execute tasks on those network devices.

These plugins generally correspond one-to-one to network device platforms. The appropriate netconf plugin will thus be automatically loaded based on the `ansible_network_os` variable. If the platform supports standard Netconf implementation as defined in the Netconf RFC specification the `default` netconf plugin will be used. In case if the platform supports propriety Netconf RPC' s in that case the interface can be defined in platform specific netconf plugin.

Adding netconf plugins

You can extend Ansible to support other network devices by dropping a custom plugin into the `netconf_plugins` directory.

Using netconf plugins

The netconf plugin to use is determined automatically from the `ansible_network_os` variable. There should be no reason to override this functionality.

Most netconf plugins can operate without configuration. A few have additional options that can be set to impact how tasks are translated into netconf commands. A ncclient device specific handler name can be set in the netconf plugin or else the value of `default` is used as per ncclient device handler.

Plugins are self-documenting. Each plugin should document its configuration options.

Plugin list

You can use `ansible-doc -t netconf -l` to see the list of available plugins. Use `ansible-doc -t netconf <plugin name>` to see detailed documentation and examples.

参见:

Ansible for Network Automation An overview of using Ansible to automate networking devices.

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible-network IRC chat channel

Shell Plugins

- *Enabling shell plugins*
- *Using shell plugins*

Shell plugins work to ensure that the basic commands Ansible runs are properly formatted to work with the target machine and allow the user to configure certain behaviors related to how Ansible executes tasks.

Enabling shell plugins

You can add a custom shell plugin by dropping it into a `shell_plugins` directory adjacent to your play, inside a role, or by putting it in one of the shell plugin directory sources configured in `ansible.cfg`.

警告: You should not alter which plugin is used unless you have a setup in which the default `/bin/sh` is not a POSIX compatible shell or is not available for execution.

Using shell plugins

In addition to the default configuration settings in `ansible_configuration_settings`, you can use the connection variable `ansible_shell_type` to select the plugin to use. In this case, you will also want to update the `ansible_shell_executable` to match.

You can further control the settings for each plugin via other configuration options detailed in the plugin themselves (linked below).

参见:

Intro to Playbooks An introduction to playbooks

Inventory Plugins Ansible inventory plugins

Callback Plugins Ansible callback plugins

Filters Jinja2 filter plugins

Tests Jinja2 test plugins

Lookups Jinja2 lookup plugins

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

Strategy Plugins

- *Enabling strategy plugins*
- *Using strategy plugins*
- *Plugin list*

Strategy plugins control the flow of play execution by handling task and host scheduling.

Enabling strategy plugins

All strategy plugins shipped with Ansible are enabled by default. You can enable a custom strategy plugin by putting it in one of the lookup directory sources configured in `ansible.cfg`.

Using strategy plugins

Only one strategy plugin can be used in a play, but you can use different ones for each play in a playbook or ansible run. The default is the linear plugin. You can change this default in Ansible configuration using an environment variable:

```
export ANSIBLE_STRATEGY=free
```

or in the `ansible.cfg` file:

```
[defaults]
strategy=linear
```

You can also specify the strategy plugin in the play via the `strategy` keyword in a play:

```
- hosts: all
  strategy: debug
  tasks:
    - copy: src=myhosts dest=/etc/hosts
      notify: restart_tomcat

    - package: name=tomcat state=present

  handlers:
    - name: restart_tomcat
      service: name=tomcat state=restarted
```

Plugin list

You can use `ansible-doc -t strategy -l` to see the list of available plugins. Use `ansible-doc -t strategy <plugin name>` to see plugin-specific specific documentation and examples.

参见:

Intro to Playbooks An introduction to playbooks

Inventory Plugins Ansible inventory plugins

Callback Plugins Ansible callback plugins

Filters Jinja2 filter plugins

Tests Jinja2 test plugins

Lookups Jinja2 lookup plugins

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

Vars Plugins

- *Enabling vars plugins*
- *Using vars plugins*
- *Plugin Lists*

Vars plugins inject additional variable data into Ansible runs that did not come from an inventory source, playbook, or command line. Playbook constructs like ‘host_vars’ and ‘group_vars’ work using vars plugins.

Vars plugins were partially implemented in Ansible 2.0 and rewritten to be fully implemented starting with Ansible 2.4.

The `host_group_vars` plugin shipped with Ansible enables reading variables from 给单台主机设置变量: *host variables* and 给多台主机设置变量: *group variables*.

Enabling vars plugins

You can activate a custom vars plugin by either dropping it into a `vars_plugins` directory adjacent to your play, inside a role, or by putting it in one of the directory sources configured in `ansible.cfg`.

Starting in Ansible 2.10, vars plugins can require whitelisting rather than running by default. To enable a plugin that requires whitelisting set `vars_plugins_enabled` in the `defaults` section of `ansible.cfg` or set the `ANSIBLE_VARS_ENABLED` environment variable to the list of vars plugins you want to execute. By default, the `host_group_vars` plugin shipped with Ansible is whitelisted.

Starting in Ansible 2.10, you can use vars plugins in collections. All vars plugins in collections require whitelisting and need to use the fully qualified collection name in the format `namespace.collection_name.vars_plugin_name`.

```
[defaults]
vars_plugins_enabled = host_group_vars,namespace.collection_name.vars_plugin_name
```


Using vars plugins

By default, vars plugins are used on demand automatically after they are enabled.

Starting in Ansible 2.10, vars plugins can be made to run at specific times. *ansible-inventory* does not use these settings, and always loads vars plugins.

The global setting `RUN_VARS_PLUGINS` can be set in `ansible.cfg` using `run_vars_plugins` in the `defaults` section or by the `ANSIBLE_RUN_VARS_PLUGINS` environment variable. The default option, `demand`, runs any enabled vars plugins relative to inventory sources whenever variables are demanded by tasks. You can use the option `start` to run any enabled vars plugins relative to inventory sources after importing that inventory source instead.

You can also control vars plugin execution on a per-plugin basis for vars plugins that support the `stage` option. To run the `host_group_vars` plugin after importing inventory you can add the following to `ansible.cfg`:

```
[vars_host_group_vars]
stage = inventory
```

Plugin Lists

You can use `ansible-doc -t vars -l` to see the list of available plugins. Use `ansible-doc -t vars <plugin name>` to see specific plugin-specific documentation and examples.

参见:

Action Plugins Ansible Action plugins

Cache Plugins Ansible Cache plugins

Callback Plugins Ansible callback plugins

Connection Plugins Ansible connection plugins

Inventory Plugins Ansible inventory plugins

Shell Plugins Ansible Shell plugins

Strategy Plugins Ansible Strategy plugins

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

Plugin Filter Configuration

Ansible 2.5 adds the ability for a site administrator to blacklist modules that they do not want to be available to Ansible. This is configured via a yaml configuration file (by default, `/etc/ansible/plugin_filters`).

yml). Use `plugin_filters_cfg` configuration in `defaults` section to change this configuration file path. The format of the file is:

```
---
filter_version: '1.0'
module_blacklist:
  # Deprecated
  - docker
  # We only allow pip, not easy_install
  - easy_install
```

The file contains two fields:

- a version so that it will be possible to update the format while keeping backwards compatibility in the future. The present version should be the string, "1.0"
- a list of modules to blacklist. Any module listed here will not be found by Ansible when it searches for a module to invoke for a task.

注解: The `stat` module is required for Ansible to run. So, please make sure you do not add this module in a blacklist modules list.

参见:

Intro to Playbooks An introduction to playbooks

ansible_configuration_settings Ansible configuration documentation and settings

Working with command line tools Ansible tools, description and options

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

Interactive input: prompts

If you want your playbook to prompt the user for certain input, add a `'vars_prompt'` section. Prompting the user for variables lets you avoid recording sensitive data like passwords. In addition to security, prompts support flexibility. For example, if you use one playbook across multiple software releases, you could prompt for the particular release version.

- *Encrypting values supplied by `vars_prompt`*
- *Allowing special characters in `vars_prompt` values*

Here is a most basic example:

```
---
- hosts: all
  vars_prompt:

    - name: username
      prompt: "What is your username?"
      private: no

    - name: password
      prompt: "What is your password?"

  tasks:

    - debug:
        msg: 'Logging in as {{ username }}'
```

The user input is hidden by default but it can be made visible by setting `private: no`.

注解: Prompts for individual `vars_prompt` variables will be skipped for any variable that is already defined through the command line `--extra-vars` option, or when running from a non-interactive session (such as cron or Ansible Tower). See *Passing variables on the command line*.

If you have a variable that changes infrequently, you can provide a default value that can be overridden:

```
vars_prompt:

- name: "release_version"
  prompt: "Product release version"
  default: "1.0"
```

Encrypting values supplied by vars_prompt

You can encrypt the entered value so you can use it, for instance, with the `user` module to define a password:

```
vars_prompt:

- name: "my_password2"
  prompt: "Enter password2"
```

(下页继续)

(续上页)

```
private: yes
encrypt: "sha512_crypt"
confirm: yes
salt_size: 7
```

If you have [Passlib](#) installed, you can use any crypt scheme the library supports:

- *des_crypt* - DES Crypt
- *bsdi_crypt* - BSDi Crypt
- *bigcrypt* - BigCrypt
- *crypt16* - Crypt16
- *md5_crypt* - MD5 Crypt
- *bcrypt* - BCrypt
- *sha1_crypt* - SHA-1 Crypt
- *sun_md5_crypt* - Sun MD5 Crypt
- *sha256_crypt* - SHA-256 Crypt
- *sha512_crypt* - SHA-512 Crypt
- *apr_md5_crypt* - Apache' s MD5-Crypt variant
- *phpass* - PHPass' Portable Hash
- *pbkdf2_digest* - Generic PBKDF2 Hashes
- *cta_pbkdf2_sha1* - Cryptacular' s PBKDF2 hash
- *dlitz_pbkdf2_sha1* - Dwayne Litzenberger' s PBKDF2 hash
- *scram* - SCRAM Hash
- *bsd_nthash* - FreeBSD' s MCF-compatible nthash encoding

The only parameters accepted are 'salt' or 'salt_size' . You can use your own salt by defining 'salt' , or have one generated automatically using 'salt_size' . By default Ansible generates a salt of size 8.

2.7 新版功能.

If you do not have Passlib installed, Ansible uses the [crypt](#) library as a fallback. Ansible supports at most four crypt schemes, depending on your platform at most the following crypt schemes are supported:

- *bcrypt* - BCrypt
- *md5_crypt* - MD5 Crypt
- *sha256_crypt* - SHA-256 Crypt

- *sha512_crypt* - SHA-512 Crypt

2.8 新版功能.

Allowing special characters in vars_prompt values

Some special characters, such as { and % can create templating errors. If you need to accept special characters, use the `unsafe` option:

```
vars_prompt:
- name: "my_password_with_weird_chars"
  prompt: "Enter password"
  unsafe: yes
  private: yes
```

参见:

Intro to Playbooks An introduction to playbooks

Conditionals Conditional statements in playbooks

Using Variables All about variables

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

Tags

If you have a large playbook, it may be useful to run only specific parts of it instead of running the entire playbook. You can do this with Ansible tags. Using tags to execute or skip selected tasks is a two-step process:

1. Add tags to your tasks, either individually or with tag inheritance from a block, play, role, or import
2. Select or skip tags when you run your playbook

- *Adding tags with the tags keyword*
 - *Adding tags to individual tasks*
 - *Adding tags to includes*
 - *Tag inheritance: adding tags to multiple tasks*
 - * *Adding tags to blocks*
 - * *Adding tags to plays*

- * *Adding tags to roles*
- * *Adding tags to imports*
- * *Tag inheritance for includes: blocks and the **apply** keyword*
- *Special tags: always and never*
- *Selecting or skipping tags when you run a playbook*
 - *Previewing the results of using tags*
 - *Selectively running tagged tasks in re-usable files*
 - *Configuring tags globally*

Adding tags with the tags keyword

You can add tags to a single task or include. You can also add tags to multiple tasks by defining them at the level of a block, play, role, or import. The keyword **tags** addresses all these use cases. The **tags** keyword always defines tags and adds them to tasks; it does not select or skip tasks for execution. You can only select or skip tasks based on tags at the command line when you run a playbook. See *Selecting or skipping tags when you run a playbook* for more details.

Adding tags to individual tasks

At the simplest level, you can apply one or more tags to an individual task. You can add tags to tasks in playbooks, in task files, or within a role. Here is an example that tags two tasks with different tags:

```
tasks:
- install the servers
  yum:
    name:
      - httpd
      - memcached
    state: present
  tags:
    - packages
    - webservers
- configure the service
  template:
    src: templates/src.j2
    dest: /etc/foo.conf
```

(下页继续)

(续上页)

```
tags:
- configuration
```

You can apply the same tag to more than one individual task. This example tags several tasks with the same tag, “ntp” :

```
---
# file: roles/common/tasks/main.yml

- name: be sure ntp is installed
  yum:
    name: ntp
    state: present
    tags: ntp

- name: be sure ntp is configured
  template:
    src: ntp.conf.j2
    dest: /etc/ntp.conf
  notify:
    - restart ntpd
  tags: ntp

- name: be sure ntpd is running and enabled
  service:
    name: ntpd
    state: started
    enabled: yes
  tags: ntp

- name: be sure file sharing is installed
  yum:
    name:
      - nfs-utils
      - nfs-util-lib
    state: present
  tags: filessharing
```

If you ran these four tasks in a playbook with `--tags ntp`, Ansible would run the three tasks tagged `ntp` and skip the one task that does not have that tag.

Adding tags to includes

You can apply tags to dynamic includes in a playbook. As with tags on an individual task, tags on an `include_*` task apply only to the include itself, not to any tasks within the included file or role. If you add `mytag` to a dynamic include, then run that playbook with `--tags mytag`, Ansible runs the include itself, runs any tasks within the included file or role tagged with `mytag`, and skips any tasks within the included file or role without that tag. See *Selectively running tagged tasks in re-usable files* for more details.

You add tags to includes the same way you add tags to any other task:

```
---
# file: roles/common/tasks/main.yml

- name: dynamic re-use of database tasks
  include_tasks: db.yml
  tags: db
```

You can add a tag only to the dynamic include of a role. In this example, the `foo` tag will *not* apply to tasks inside the `bar` role:

```
---
- hosts: webservers
  tasks:
    - include_role:
        name: bar
      tags:
        - foo
```

With plays, blocks, the `role` keyword, and static imports, Ansible applies tag inheritance, adding the tags you define to every task inside the play, block, role, or imported file. However, tag inheritance does *not* apply to dynamic re-use with `include_role` and `include_tasks`. With dynamic re-use (includes), the tags you define apply only to the include itself. If you need tag inheritance, use a static import. If you cannot use an import because the rest of your playbook uses includes, see *Tag inheritance for includes: blocks and the apply keyword* for ways to work around this behavior.

Tag inheritance: adding tags to multiple tasks

If you want to apply the same tag or tags to multiple tasks without adding a `tags` line to every task, you can define the tags at the level of your play or block, or when you add a role or import a file. Ansible applies the tags down the dependency chain to all child tasks. With roles and imports, Ansible appends the tags set by the `roles` section or import to any tags set on individual tasks or blocks within the role or imported file. This is called tag inheritance. Tag inheritance is convenient, because you do not have to tag every task. However, the tags still apply to the tasks individually.

Adding tags to blocks

If you want to apply a tag to many, but not all, of the tasks in your play, use a *block* and define the tags at that level. For example, we could edit the NTP example shown above to use a block:

```
# myrole/tasks/main.yml
tasks:
- block:
  tags: ntp
  - name: be sure ntp is installed
    yum:
      name: ntp
      state: present
  - name: be sure ntp is configured
    template:
      src: ntp.conf.j2
      dest: /etc/ntp.conf
    notify:
      - restart ntpd
  - name: be sure ntpd is running and enabled
    service:
      name: ntpd
      state: started
      enabled: yes

- name: be sure file sharing is installed
  yum:
    name:
      - nfs-utils
      - nfs-util-lib
    state: present
  tags: filesharing
```

Adding tags to plays

If all the tasks in a play should get the same tag, you can add the tag at the level of the play. For example, if you had a play with only the NTP tasks, you could tag the entire play:

```
- hosts: all
  tags: ntp
  tasks:
```

(下页继续)

```
- name: be sure ntp is installed
  yum:
    name: ntp
    state: present

- name: be sure ntp is configured
  template:
    src: ntp.conf.j2
    dest: /etc/ntp.conf
  notify:
    - restart ntpd

- name: be sure ntpd is running and enabled
  service:
    name: ntpd
    state: started
    enabled: yes

- hosts: fileserver
  tags: filesharing
  tasks:
    ...
```

Adding tags to roles

There are three ways to add tags to roles:

1. Add the same tag or tags to all tasks in the role by setting tags under **roles**. See examples in this section.
2. Add the same tag or tags to all tasks in the role by setting tags on a static **import_role** in your playbook. See examples in *Adding tags to imports*.
3. Add a tag or tags to individual tasks or blocks within the role itself. This is the only approach that allows you to select or skip some tasks within the role. To select or skip tasks within the role, you must have tags set on individual tasks or blocks, use the dynamic **include_role** in your playbook, and add the same tag or tags to the include. When you use this approach, and then run your playbook with **--tags foo**, Ansible runs the include itself plus any tasks in the role that also have the tag **foo**. See *Adding tags to includes* for details.

When you incorporate a role in your playbook statically with the **roles** keyword, Ansible adds any tags you define to all the tasks in the role. For example:

```
roles:
  - role: webserver
    vars:
      port: 5000
      tags: [ web, foo ]
```

or:

```
---
- hosts: webservers
  roles:
    - role: foo
      tags:
        - bar
        - baz

# using YAML shorthand, this is equivalent to:
# - { role: foo, tags: ["bar", "baz"] }
```

Adding tags to imports

You can also apply a tag or tags to all the tasks imported by the static `import_role` and `import_tasks` statements:

```
---
- hosts: webservers
  tasks:
    - import_role:
        name: foo
        tags:
          - bar
          - baz

    - import_tasks: foo.yml
      tags: [ web, foo ]
```

Tag inheritance for includes: blocks and the apply keyword

By default, Ansible does not apply *tag inheritance* to dynamic re-use with `include_role` and `include_tasks`. If you add tags to an include, they apply only to the include itself, not to any tasks

in the included file or role. This allows you to execute selected tasks within a role or task file - see *Selectively running tagged tasks in re-usable files* when you run your playbook.

If you want tag inheritance, you probably want to use imports. However, using both includes and imports in a single playbook can lead to difficult-to-diagnose bugs. For this reason, if your playbook uses `include_*` to re-use roles or tasks, and you need tag inheritance on one include, Ansible offers two workarounds. You can use the `apply` keyword:

```
- name: applies the db tag to the include and to all tasks in db.yml
  include_tasks:
    file: db.yml
    # adds 'db' tag to tasks within db.yml
    apply:
      tags: db
    # adds 'db' tag to this 'include_tasks' itself
  tags: db
```

Or you can use a block:

```
- block:
  - include_tasks: db.yml
  tags: db
```

Special tags: always and never

Ansible reserves two tag names for special behavior: `always` and `never`. If you assign the `always` tag to a task or play, Ansible will always run that task or play, unless you specifically skip it (`--skip-tags always`).

For example:

```
tasks:
- debug:
    msg: "Always runs"
    tags:
    - always

- debug:
    msg: "runs when you use tag1"
    tags:
    - tag1
```

警告:

- Fact gathering is tagged with ‘always’ by default. It is only skipped if you apply a tag and then use a different tag in `--tags` or the same tag in `--skip-tags`.

2.5 新版功能.

If you assign the **never** tag to a task or play, Ansible will skip that task or play unless you specifically request it (`--tags never`).

For example:

```
tasks:
- Rarely-used debug task
  debug: msg="{{ showmevar }}"
  tags: [ never, debug ]
```

The rarely-used debug task in the example above only runs when you specifically request the **debug** or **never** tags.

Selecting or skipping tags when you run a playbook

Once you have added tags to your tasks, includes, blocks, plays, roles, and imports, you can selectively execute or skip tasks based on their tags when you run `ansible-playbook`. Ansible runs or skips all tasks with tags that match the tags you pass at the command line. If you have added a tag at the block or play level, with **roles**, or with an import, that tag applies to every task within the block, play, role, or imported role or file. If you have a role with lots of tags and you want to call subsets of the role at different times, either *use it with dynamic includes*, or split the role into multiple roles.

`ansible-playbook` offers five tag-related command-line options:

- `--tags all` - run all tasks, ignore tags (default behavior)
- `--tags [tag1, tag2]` - run only tasks with the tags **tag1** and **tag2**
- `--skip-tags [tag3, tag4]` - run all tasks except those with the tags **tag3** and **tag4**
- `--tags tagged` - run only tasks with at least one tag
- `--tags untagged` - run only tasks with no tags

For example, to run only tasks and blocks tagged **configuration** and **packages** in a very long playbook:

```
ansible-playbook example.yml --tags "configuration,packages"
```

To run all tasks except those tagged **packages**:

```
ansible-playbook example.yml --skip-tags "packages"
```

Previewing the results of using tags

When you run a role or playbook, you might not know or remember which tasks have which tags, or which tags exist at all. Ansible offers two command-line flags for `ansible-playbook` that help you manage tagged playbooks:

- `--list-tags` - generate a list of available tags
- `--list-tasks` - when used with `--tags tagname` or `--skip-tags tagname`, generate a preview of tagged tasks

For example, if you do not know whether the tag for configuration tasks is `config` or `conf` in a playbook, role, or tasks file, you can display all available tags without running any tasks:

```
ansible-playbook example.yml --list-tags
```

If you do not know which tasks have the tags `configuration` and `packages`, you can pass those tags and add `--list-tasks`. Ansible lists the tasks but does not execute any of them.

```
ansible-playbook example.yml --tags "configuration,packages" --list-tasks
```

These command-line flags have one limitation: they cannot show tags or tasks within dynamically included files or roles. See *Comparing includes and imports: dynamic vs. static* for more information on differences between static imports and dynamic includes.

Selectively running tagged tasks in re-usable files

If you have a role or a tasks file with tags defined at the task or block level, you can selectively run or skip those tagged tasks in a playbook if you use a dynamic include instead of a static import. You must use the same tag on the included tasks and on the include statement itself. For example you might create a file with some tagged and some untagged tasks:

```
# mixed.yml
tasks:
- name: task with no tags
  debug:
    msg: this task has no tags

- name: tagged task
  debug:
```

(下页继续)

(续上页)

```
    msg: this task is tagged with mytag
    tags: mytag

- block:
  - name: First block task with mytag
    ...
  - name: Second block task with mytag
    ...
  tags:
  - mytag
```

And you might include the tasks file above in a playbook:

```
# myplaybook.yml
- hosts: all
  tasks:
  - include_tasks:
      name: mixed.yml
      tags: mytag
```

When you run the playbook with `ansible-playbook -i hosts myplaybook.yml --tags "mytag"`, Ansible skips the task with no tags, runs the tagged individual task, and runs the two tasks in the block.

Configuring tags globally

If you run or skip certain tags by default, you can use the `TAGS_RUN` and `TAGS_SKIP` options in Ansible configuration to set those defaults.

参见:

Intro to Playbooks An introduction to playbooks

Roles Playbook organization by roles

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

Using Vault in playbooks

Topics

- *Using Vault in playbooks*
 - *Running a Playbook With Vault*
 - *Multiple Vault Passwords*
 - *Vault Password Client Scripts*
 - *Single Encrypted Variable*
 - *Using `encrypt_string`*

The “Vault” is a feature of Ansible that allows you to keep sensitive data such as passwords or keys protected at rest, rather than as plaintext in playbooks or roles. These vaults can then be distributed or placed in source control.

There are 2 types of vaulted content and each has their own uses and limitations:

Vaulted files

- The full file is encrypted in the vault, this can contain Ansible variables or any other type of content.
- It will always be decrypted when loaded or referenced, Ansible cannot know if it needs the content unless it decrypts it.
- It can be used for inventory, anything that loads variables (i.e `vars_files`, `group_vars`, `host_vars`, `include_vars`, etc) and some actions that deal with files (i.e `M(copy)`, `M(assemble)`, `M(script)`, etc).

Single encrypted variable

- Only specific variables are encrypted inside a normal ‘variable file’ .
- Does not work for other content, only variables.
- Decrypted on demand, so you can have vaulted variables with different vault secrets and only provide those needed.
- You can mix vaulted and non vaulted variables in the same file, even inline in a play or role.

警告:

- Vault ONLY protects data ‘at rest’ . Once decrypted, play and plugin authors are responsible for avoiding any secret disclosure, see [no_log](#) for details on hiding output.

To enable this feature, a command line tool, `ansible-vault` is used to edit files, and a command line flag `--ask-vault-pass`, `--vault-password-file` or `--vault-id` is used. You can also modify your `ansible.cfg` file to specify the location of a password file or configure Ansible to always prompt for the password.

These options require no command line flag usage.

For best practices advice, refer to *Keep vaulted variables safely visible*.

Running a Playbook With Vault

To run a playbook that contains vault-encrypted data files, you must provide the vault password.

To specify the vault-password interactively:

```
ansible-playbook site.yml --ask-vault-pass
```

This prompt will then be used to decrypt (in memory only) any vault encrypted files that are accessed.

Alternatively, passwords can be specified with a file or a script (the script version will require Ansible 1.7 or later). When using this flag, ensure permissions on the file are such that no one else can access your key and do not add your key to source control:

```
ansible-playbook site.yml --vault-password-file ~/.vault_pass.txt

ansible-playbook site.yml --vault-password-file ~/.vault_pass.py
```

The password should be a string stored as a single line in the file.

If you are using a script instead of a flat file, ensure that it is marked as executable, and that the password is printed to standard output. If your script needs to prompt for data, prompts can be sent to standard error.

注解: You can also set `ANSIBLE_VAULT_PASSWORD_FILE` environment variable, e.g. `ANSIBLE_VAULT_PASSWORD_FILE=~/.vault_pass.txt` and Ansible will automatically search for the password in that file.

This is something you may wish to do if using Ansible from a continuous integration system like Jenkins.

The `--vault-password-file` option can also be used with the `ansible-pull` command if you wish, though this would require distributing the keys to your nodes, so understand the implications – vault is more intended for push mode.

Multiple Vault Passwords

Ansible 2.4 and later support the concept of multiple vaults that are encrypted with different passwords. Different vaults can be given a label to distinguish them (generally values like dev, prod etc.).

The `--ask-vault-pass` and `--vault-password-file` options can be used as long as only a single password is needed for any given run.

Alternatively the `--vault-id` option can be used to provide the password and indicate which vault label it's for. This can be clearer when multiple vaults are used within a single inventory. For example:

To be prompted for the 'dev' password:

```
ansible-playbook site.yml --vault-id dev@prompt
```

To get the 'dev' password from a file or script:

```
ansible-playbook site.yml --vault-id dev@~/.vault_pass.txt
```

```
ansible-playbook site.yml --vault-id dev@~/.vault_pass.py
```

If multiple vault passwords are required for a single run, `--vault-id` must be used as it can be specified multiple times to provide the multiple passwords. For example:

To read the 'dev' password from a file and prompt for the 'prod' password:

```
ansible-playbook site.yml --vault-id dev@~/.vault_pass.txt --vault-id prod@prompt
```

The `--ask-vault-pass` or `--vault-password-file` options can be used to specify one of the passwords, but it's generally cleaner to avoid mixing these with `--vault-id`.

注解: By default the vault label (dev, prod etc.) is just a hint. Ansible will try to decrypt each vault with every provided password.

Setting the config option `DEFAULT_VAULT_ID_MATCH` will change this behavior so that each password is only used to decrypt data that was encrypted with the same label. See [Labelling Vaults](#) for more details.

Vault Password Client Scripts

Ansible 2.5 and later support using a single executable script to get different passwords depending on the vault label. These client scripts must have a file name that ends with `-client`. For example:

To get the dev password from the system keyring using the `contrib/vault/vault-keyring-client.py` script:

```
ansible-playbook --vault-id dev@contrib/vault/vault-keyring-client.py
```

See [Vault Password Client Scripts](#) for a complete explanation of this topic.

Single Encrypted Variable

As of version 2.3, Ansible can now use a vaulted variable that lives in an otherwise 'clear text' YAML file:

```

notsecret: myvalue
mysecret: !vault |
    $ANSIBLE_VAULT;1.1;AES256
    ␣
    ↳66386439653236336462626566653063336164663966303231363934653561363964363833313662
    ␣
    ↳6431626536303530376336343832656537303632313433360a626438346336353331386135323734
    ␣
    ↳62656361653630373231613662633962316233633936396165386439616533353965373339616234
    ␣
    ↳3430613539666330390a313736323265656432366236633330313963326365653937323833366536
    34623731376664623134383463316265643436343438623266623965636363326136
other_plain_text: othervalue

```

To create a vaulted variable, use the `ansible-vault encrypt_string` command. See *Using encrypt_string* for details.

This vaulted variable will be decrypted with the supplied vault secret and used as a normal variable. The `ansible-vault` command line supports stdin and stdout for encrypting data on the fly, which can be used from your favorite editor to create these vaulted variables; you just have to be sure to add the `!vault` tag so both Ansible and YAML are aware of the need to decrypt. The `|` is also required, as vault encryption results in a multi-line string.

注解: Inline vaults ONLY work on variables, you cannot use directly on a task's options.

Using encrypt_string

This command will output a string in the above format ready to be included in a YAML file. The string to encrypt can be provided via stdin, command line arguments, or via an interactive prompt.

See *Use encrypt_string to create encrypted variables to embed in yaml*.

Start and Step

This shows a few alternative ways to run playbooks. These modes are very useful for testing new plays or debugging.

Start-at-task

If you want to start executing your playbook at a particular task, you can do so with the `--start-at-task` option:

```
ansible-playbook playbook.yml --start-at-task="install packages"
```

The above will start executing your playbook at a task named “install packages” .

Step

Playbooks can also be executed interactively with `--step`:

```
ansible-playbook playbook.yml --step
```

This will cause ansible to stop on each task, and ask if it should execute that task. Say you had a task called “configure ssh” , the playbook run will stop and ask:

```
Perform task: configure ssh (y/n/c):
```

Answering “y” will execute the task, answering “n” will skip the task, and answering “c” will continue executing all the remaining tasks without asking.

Module defaults

If you find yourself calling the same module repeatedly with the same arguments, it can be useful to define default arguments for that particular module using the `module_defaults` attribute.

Here is a basic example:

```
- hosts: localhost
  module_defaults:
    file:
      owner: root
      group: root
      mode: 0755
  tasks:
    - file:
        state: touch
        path: /tmp/file1
    - file:
        state: touch
```

(下页继续)

(续上页)

```

    path: /tmp/file2
- file:
    state: touch
    path: /tmp/file3

```

The `module_defaults` attribute can be used at the play, block, and task level. Any module arguments explicitly specified in a task will override any established default for that module argument:

```

- block:
  - debug:
    msg: "a different message"
  module_defaults:
    debug:
      msg: "a default message"

```

It's also possible to remove any previously established defaults for a module by specifying an empty dict:

```

- file:
  state: touch
  path: /tmp/file1
  module_defaults:
    file: {}

```

注解: Any module defaults set at the play level (and block/task level when using `include_role` or `import_role`) will apply to any roles used, which may cause unexpected behavior in the role.

Here are some more realistic use cases for this feature.

Interacting with an API that requires auth:

```

- hosts: localhost
  module_defaults:
    uri:
      force_basic_auth: true
      user: some_user
      password: some_password
  tasks:
    - uri:
      url: http://some.api.host/v1/whatever1
    - uri:

```

(下页继续)

(续上页)

```

    url: http://some.api.host/v1/whatever2
- uri:
    url: http://some.api.host/v1/whatever3

```

Setting a default AWS region for specific EC2-related modules:

```

- hosts: localhost
  vars:
    my_region: us-west-2
  module_defaults:
    ec2:
      region: '{{ my_region }}'
    ec2_instance_info:
      region: '{{ my_region }}'
    ec2_vpc_net_info:
      region: '{{ my_region }}'

```

Module defaults groups

2.7 新版功能.

Ansible 2.7 adds a preview-status feature to group together modules that share common sets of parameters. This makes it easier to author playbooks making heavy use of API-based modules such as cloud modules.

Group	Purpose	Ansible Version
aws	Amazon Web Services	2.7
azure	Azure	2.7
gcp	Google Cloud Platform	2.7
k8s	Kubernetes	2.8
os	OpenStack	2.8
acme	ACME	2.10
docker*	Docker	2.10
ovirt	oVirt	2.10
vmware	VMware	2.10

- The `docker_stack` module is not included in the `docker` defaults group.

Use the groups with `module_defaults` by prefixing the group name with `group/` - e.g. `group/aws`.

In a playbook, you can set module defaults for whole groups of modules, such as setting a common AWS region.

```
# example_play.yml
- hosts: localhost
  module_defaults:
    group/aws:
      region: us-west-2
  tasks:
    - aws_s3_bucket_info:
      # now the region is shared between both info modules
    - ec2_ami_info:
      filters:
        name: 'RHEL*7.5*'
```

Controlling playbook execution: strategies and more

By default, Ansible runs each task on all hosts affected by a play before starting the next task on any host, using 5 forks. If you want to change this default behavior, you can use a different strategy plugin, change the number of forks, or apply one of several play-level keywords like `serial`.

- *Selecting a strategy*
- *Setting the number of forks*
- *Using keywords to control execution*

Selecting a strategy

The default behavior described above is the linear strategy. Ansible offers other strategies, including the debug strategy (see also *Playbook Debugger*) and the free strategy, which allows each host to run until the end of the play as fast as it can:

```
- hosts: all
  strategy: free
  tasks:
    ...
```

You can select a different strategy for each play as shown above, or set your preferred strategy globally in `ansible.cfg`, under the `defaults` stanza:

```
[defaults]
strategy = free
```

All strategies are implemented as *strategy plugins*. Please review the documentation for each strategy plugin for details on how it works.

Setting the number of forks

If you have the processing power available and want to use more forks, you can set the number in `ansible.cfg`:

```
[defaults]
forks = 30
```

or pass it on the command line: `ansible-playbook -f 30 my_playbook.yml`.

Using keywords to control execution

Several play-level keyword also affect play execution. The most common one is `serial`, which sets a number, a percentage, or a list of numbers of hosts you want to manage at a time. Setting `serial` with any strategy directs Ansible to ‘batch’ the hosts, completing the play on the specified number or percentage of hosts before starting the next ‘batch’. This is especially useful for *rolling updates*.

The `throttle` keyword also affects execution and can be set at the block and task level. This keyword limits the number of workers up to the maximum set with the forks setting or `serial`. Use `throttle` to restrict tasks that may be CPU-intensive or interact with a rate-limiting API:

```
tasks:
- command: /path/to/cpu_intensive_command
  throttle: 1
```

The `order` keyword controls the order in which hosts are run. Possible values for order are:

inventory: (default) The order provided in the inventory

reverse_inventory: The reverse of the order provided by the inventory

sorted: Sorted alphabetically sorted by name

reverse_sorted: Sorted by name in reverse alphabetical order

shuffle: Randomly ordered on each run

Other keywords that affect play execution include `ignore_errors`, `ignore_unreachable`, and `any_errors_fatal`. Please note that these keywords are not strategies. They are play-level directives or options.

参见:

Intro to Playbooks An introduction to playbooks

Roles Playbook organization by roles

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

Playbook Example: Continuous Delivery and Rolling Upgrades

- *What is continuous delivery?*
- *Site deployment*
- *Reusable content: roles*
- *Configuration: group variables*
- *The rolling upgrade*
- *Managing other load balancers*
- *Continuous delivery end-to-end*

What is continuous delivery?

Continuous delivery (CD) means frequently delivering updates to your software application.

The idea is that by updating more often, you do not have to wait for a specific timed period, and your organization gets better at the process of responding to change.

Some Ansible users are deploying updates to their end users on an hourly or even more frequent basis – sometimes every time there is an approved code change. To achieve this, you need tools to be able to quickly apply those updates in a zero-downtime way.

This document describes in detail how to achieve this goal, using one of Ansible’s most complete example playbooks as a template: `lamp_haproxy`. This example uses a lot of Ansible features: roles, templates, and group variables, and it also comes with an orchestration playbook that can do zero-downtime rolling upgrades of the web application stack.

注解: [Click here for the latest playbooks for this example.](#)

The playbooks deploy Apache, PHP, MySQL, Nagios, and HAProxy to a CentOS-based set of servers.

We’re not going to cover how to run these playbooks here. Read the included README in the github project along with the example for that information. Instead, we’re going to take a close look at every part of the playbook and describe what it does.

Site deployment

Let's start with `site.yml`. This is our site-wide deployment playbook. It can be used to initially deploy the site, as well as push updates to all of the servers:

```
---
# This playbook deploys the whole application stack in this site.

# Apply common configuration to all hosts
- hosts: all

  roles:
    - common

# Configure and deploy database servers.
- hosts: dbservers

  roles:
    - db

# Configure and deploy the web servers. Note that we include two roles
# here, the 'base-apache' role which simply sets up Apache, and 'web'
# which includes our example web application.

- hosts: webservers

  roles:
    - base-apache
    - web

# Configure and deploy the load balancer(s).
- hosts: lbservers

  roles:
    - haproxy

# Configure and deploy the Nagios monitoring node(s).
- hosts: monitoring

  roles:
    - base-apache
```

(下页继续)

(续上页)

```
- nagios
```

注解: If you're not familiar with terms like playbooks and plays, you should review [Working With Playbooks](#).

In this playbook we have 5 plays. The first one targets **all** hosts and applies the **common** role to all of the hosts. This is for site-wide things like yum repository configuration, firewall configuration, and anything else that needs to apply to all of the servers.

The next four plays run against specific host groups and apply specific roles to those servers. Along with the roles for Nagios monitoring, the database, and the web application, we've implemented a **base-apache** role that installs and configures a basic Apache setup. This is used by both the sample web application and the Nagios hosts.

Reusable content: roles

By now you should have a bit of understanding about roles and how they work in Ansible. Roles are a way to organize content: tasks, handlers, templates, and files, into reusable components.

This example has six roles: **common**, **base-apache**, **db**, **haproxy**, **nagios**, and **web**. How you organize your roles is up to you and your application, but most sites will have one or more common roles that are applied to all systems, and then a series of application-specific roles that install and configure particular parts of the site.

Roles can have variables and dependencies, and you can pass in parameters to roles to modify their behavior. You can read more about roles in the [Roles](#) section.

Configuration: group variables

Group variables are variables that are applied to groups of servers. They can be used in templates and in playbooks to customize behavior and to provide easily-changed settings and parameters. They are stored in a directory called **group_vars** in the same location as your inventory. Here is `lamp_haproxy's group_vars/all` file. As you might expect, these variables are applied to all of the machines in your inventory:

```
---
httpd_port: 80
ntpserver: 192.0.2.23
```

This is a YAML file, and you can create lists and dictionaries for more complex variable structures. In this case, we are just setting two variables, one for the port for the web server, and one for the NTP server that our machines should use for time synchronization.

Here's another group variables file. This is `group_vars/dbservers` which applies to the hosts in the `dbservers` group:

```
---
mysqlservice: mysqld
mysql_port: 3306
dbuser: root
dbname: foodb
upassword: usersecret
```

If you look in the example, there are group variables for the `webservers` group and the `lbserver`s group, similarly.

These variables are used in a variety of places. You can use them in playbooks, like this, in `roles/db/tasks/main.yml`:

```
- name: Create Application Database
  mysql_db:
    name: "{{ dbname }}"
    state: present

- name: Create Application DB User
  mysql_user:
    name: "{{ dbuser }}"
    password: "{{ upassword }}"
    priv: "*.*:ALL"
    host: '%'
    state: present
```

You can also use these variables in templates, like this, in `roles/common/templates/ntp.conf.j2`:

```
driftfile /var/lib/ntp/drift

restrict 127.0.0.1
restrict -6 ::1

server {{ ntpserver }}

includefile /etc/ntp/crypto/pw

keys /etc/ntp/keys
```

You can see that the variable substitution syntax of `{{` and `}}` is the same for both templates and variables.

The syntax inside the curly braces is Jinja2, and you can do all sorts of operations and apply different filters to the data inside. In templates, you can also use for loops and if statements to handle more complex situations, like this, in `roles/common/templates/iptables.j2`:

```
{% if inventory_hostname in groups['dbservers'] %}
-A INPUT -p tcp --dport 3306 -j ACCEPT
{% endif %}
```

This is testing to see if the inventory name of the machine we're currently operating on (`inventory_hostname`) exists in the inventory group `dbservers`. If so, that machine will get an iptables ACCEPT line for port 3306.

Here's another example, from the same template:

```
{% for host in groups['monitoring'] %}
-A INPUT -p tcp -s {{ hostvars[host].ansible_default_ipv4.address }} --dport 5666 -j
→ACCEPT
{% endfor %}
```

This loops over all of the hosts in the group called `monitoring`, and adds an ACCEPT line for each monitoring hosts' default IPv4 address to the current machine's iptables configuration, so that Nagios can monitor those hosts.

You can learn a lot more about Jinja2 and its capabilities [here](#), and you can read more about Ansible variables in general in the *Using Variables* section.

The rolling upgrade

Now you have a fully-deployed site with web servers, a load balancer, and monitoring. How do you update it? This is where Ansible's orchestration features come into play. While some applications use the term 'orchestration' to mean basic ordering or command-blasting, Ansible refers to orchestration as 'conducting machines like an orchestra', and has a pretty sophisticated engine for it.

Ansible has the capability to do operations on multi-tier applications in a coordinated way, making it easy to orchestrate a sophisticated zero-downtime rolling upgrade of our web application. This is implemented in a separate playbook, called `rolling_update.yml`.

Looking at the playbook, you can see it is made up of two plays. The first play is very simple and looks like this:

```
- hosts: monitoring
  tasks: []
```

What's going on here, and why are there no tasks? You might know that Ansible gathers "facts" from the servers before operating upon them. These facts are useful for all sorts of things: networking information,

OS/distribution versions, etc. In our case, we need to know something about all of the monitoring servers in our environment before we perform the update, so this simple play forces a fact-gathering step on our monitoring servers. You will see this pattern sometimes, and it's a useful trick to know.

The next part is the update play. The first part looks like this:

```
- hosts: webservers
  user: root
  serial: 1
```

This is just a normal play definition, operating on the **webservers** group. The **serial** keyword tells Ansible how many servers to operate on at once. If it's not specified, Ansible will parallelize these operations up to the default “forks” limit specified in the configuration file. But for a zero-downtime rolling upgrade, you may not want to operate on that many hosts at once. If you had just a handful of webservers, you may want to set **serial** to 1, for one host at a time. If you have 100, maybe you could set **serial** to 10, for ten at a time.

Here is the next part of the update play:

```
pre_tasks:
- name: disable nagios alerts for this host webserver service
  nagios:
    action: disable_alerts
    host: "{{ inventory_hostname }}"
    services: webserver
  delegate_to: "{{ item }}"
  loop: "{{ groups.monitoring }}"

- name: disable the server in haproxy
  shell: echo "disable server myaplb/{{ inventory_hostname }}" | socat stdio /var/lib/
↪haproxy/stats
  delegate_to: "{{ item }}"
  loop: "{{ groups.lbservers }}"
```

注解:

- The **serial** keyword forces the play to be executed in ‘batches’. Each batch counts as a full play with a subselection of hosts. This has some consequences on play behavior. For example, if all hosts in a batch fails, the play fails, which in turn fails the entire run. You should consider this when combining with **max_fail_percentage**.

The **pre_tasks** keyword just lets you list tasks to run before the roles are called. This will make more sense in a minute. If you look at the names of these tasks, you can see that we are disabling Nagios alerts and

then removing the webserver that we are currently updating from the HAProxy load balancing pool.

The `delegate_to` and `loop` arguments, used together, cause Ansible to loop over each monitoring server and load balancer, and perform that operation (delegate that operation) on the monitoring or load balancing server, “on behalf” of the webserver. In programming terms, the outer loop is the list of web servers, and the inner loop is the list of monitoring servers.

Note that the HAProxy step looks a little complicated. We’re using HAProxy in this example because it’s freely available, though if you have (for instance) an F5 or Netscaler in your infrastructure (or maybe you have an AWS Elastic IP setup?), you can use modules included in core Ansible to communicate with them instead. You might also wish to use other monitoring modules instead of nagios, but this just shows the main goal of the ‘pre tasks’ section – take the server out of monitoring, and take it out of rotation.

The next step simply re-applies the proper roles to the web servers. This will cause any configuration management declarations in `web` and `base-apache` roles to be applied to the web servers, including an update of the web application code itself. We don’t have to do it this way—we could instead just purely update the web application, but this is a good example of how roles can be used to reuse tasks:

```
roles:
- common
- base-apache
- web
```

Finally, in the `post_tasks` section, we reverse the changes to the Nagios configuration and put the web server back in the load balancing pool:

```
post_tasks:
- name: Enable the server in haproxy
  shell: echo "enable server myapplb/{{ inventory_hostname }}" | socat stdio /var/lib/
↪haproxy/stats
  delegate_to: "{{ item }}"
  loop: "{{ groups.lbservers }}"

- name: re-enable nagios alerts
  nagios:
    action: enable_alerts
    host: "{{ inventory_hostname }}"
    services: webserver
  delegate_to: "{{ item }}"
  loop: "{{ groups.monitoring }}"
```

Again, if you were using a Netscaler or F5 or Elastic Load Balancer, you would just substitute in the appropriate modules instead.

Managing other load balancers

In this example, we use the simple HAProxy load balancer to front-end the web servers. It's easy to configure and easy to manage. As we have mentioned, Ansible has built-in support for a variety of other load balancers like Citrix NetScaler, F5 BigIP, Amazon Elastic Load Balancers, and more. See the *Working With Modules* documentation for more information.

For other load balancers, you may need to send shell commands to them (like we do for HAProxy above), or call an API, if your load balancer exposes one. For the load balancers for which Ansible has modules, you may want to run them as a `local_action` if they contact an API. You can read more about local actions in the *Delegation, Rolling Updates, and Local Actions* section. Should you develop anything interesting for some hardware where there is not a core module, it might make for a good module for core inclusion!

Continuous delivery end-to-end

Now that you have an automated way to deploy updates to your application, how do you tie it all together? A lot of organizations use a continuous integration tool like [Jenkins](#) or [Atlassian Bamboo](#) to tie the development, test, release, and deploy steps together. You may also want to use a tool like [Gerrit](#) to add a code review step to commits to either the application code itself, or to your Ansible playbooks, or both.

Depending on your environment, you might be deploying continuously to a test environment, running an integration test battery against that environment, and then deploying automatically into production. Or you could keep it simple and just use the rolling-update for on-demand deployment into test or production specifically. This is all up to you.

For integration with Continuous Integration systems, you can easily trigger playbook runs using the `ansible-playbook` command line tool, or, if you're using [Red Hat Ansible Tower](#), the `tower-cli` or the built-in REST API. (The `tower-cli` command `'joblaunch'` will spawn a remote job over the REST API and is pretty slick).

This should give you a good idea of how to structure a multi-tier application with Ansible, and orchestrate operations upon that app, with the eventual goal of continuous delivery to your customers. You could extend the idea of the rolling upgrade to lots of different parts of the app; maybe add front-end web servers along with application servers, for instance, or replace the SQL database with something like MongoDB or Riak. Ansible gives you the capability to easily manage complicated environments and automate common operations.

参见:

[lamp_haproxy example](#) The `lamp_haproxy` example discussed here.

[Working With Playbooks](#) An introduction to playbooks

[Roles](#) An introduction to playbook roles

[Using Variables](#) An introduction to Ansible variables

Ansible.com: Continuous Delivery An introduction to Continuous Delivery with Ansible

1.3.11 Ansible Vault

Topics

- *Ansible Vault*
 - *What Can Be Encrypted With Vault*
 - * *File-level encryption*
 - * *Variable-level encryption*
 - *Vault IDs and Multiple Vault Passwords*
 - *Creating Encrypted Files*
 - *Editing Encrypted Files*
 - *Rekeying Encrypted Files*
 - *Encrypting Unencrypted Files*
 - *Decrypting Encrypted Files*
 - *Viewing Encrypted Files*
 - *Use encrypt_string to create encrypted variables to embed in yaml*
 - *Providing Vault Passwords*
 - * *Labelling Vaults*
 - * *Multiple Vault Passwords*
 - *Vault Password Client Scripts*
 - *Speeding Up Vault Operations*
 - *Vault Format*
 - *Vault Payload Format 1.1 - 1.2*

Ansible Vault is a feature of ansible that allows you to keep sensitive data such as passwords or keys in encrypted files, rather than as plaintext in playbooks or roles. These vault files can then be distributed or placed in source control.

To enable this feature, a command line tool - `ansible-vault` - is used to edit files, and a command line flag (`--ask-vault-pass`, `--vault-password-file` or `--vault-id`) is used. Alternately, you may specify the location of a password file or command Ansible to always prompt for the password in your `ansible.cfg` file. These options require no command line flag usage.

For best practices advice, refer to *Keep vaulted variables safely visible*.

What Can Be Encrypted With Vault

File-level encryption

Ansible Vault can encrypt any structured data file used by Ansible.

This can include “group_vars/” or “host_vars/” inventory variables, variables loaded by “include_vars” or “vars_files” , or variable files passed on the ansible-playbook command line with `-e @file.yml` or `-e @file.json`. Role variables and defaults are also included.

Ansible tasks, handlers, and so on are also data so these can be encrypted with vault as well. To hide the names of variables that you’ re using, you can encrypt the task files in their entirety.

Ansible Vault can also encrypt arbitrary files, even binary files. If a vault-encrypted file is given as the `src` argument to the `copy`, `template`, `unarchive`, `script` or `assemble` modules, the file will be placed at the destination on the target host decrypted (assuming a valid vault password is supplied when running the play).

注解: The advantages of file-level encryption are that it is easy to use and that password rotation is straightforward with *rekeying*. The drawback is that the contents of files are no longer easy to access and read. This may be problematic if it is a list of tasks (when encrypting a variables file, *best practice* is to keep references to these variables in a non-encrypted file).

Variable-level encryption

Ansible also supports encrypting single values inside a YAML file, using the `!vault` tag to let YAML and Ansible know it uses special processing. This feature is covered in more detail *below*.

注解: The advantage of variable-level encryption is that files are still easily legible even if they mix plaintext and encrypted variables. The drawback is that password rotation is not as simple as with file-level encryption: the `rekey` command does not work with this method.

Vault IDs and Multiple Vault Passwords

A vault ID is an identifier for one or more vault secrets; Ansible supports multiple vault passwords.

Vault IDs provide labels to distinguish between individual vault passwords.

To use vault IDs, you must provide an ID *label* of your choosing and a *source* to obtain its password (either `prompt` or a file path):

```
--vault-id label@source
```

This switch is available for all Ansible commands that can interact with vaults: `ansible-vault`, `ansible-playbook`, etc.

Vault-encrypted content can specify which vault ID it was encrypted with.

For example, a playbook can now include a vars file encrypted with a ‘dev’ vault ID and a ‘prod’ vault ID.

Creating Encrypted Files

To create a new encrypted data file, run the following command:

```
ansible-vault create foo.yml
```

First you will be prompted for a password. After providing a password, the tool will launch whatever editor you have defined with `$EDITOR`, and defaults to `vi`. Once you are done with the editor session, the file will be saved as encrypted data.

The default cipher is AES (which is shared-secret based).

To create a new encrypted data file with the Vault ID ‘password1’ assigned to it and be prompted for the password, run:

```
ansible-vault create --vault-id password1@prompt foo.yml
```

Editing Encrypted Files

To edit an encrypted file in place, use the `ansible-vault edit` command. This command will decrypt the file to a temporary file and allow you to edit the file, saving it back when done and removing the temporary file:

```
ansible-vault edit foo.yml
```

To edit a file encrypted with the ‘vault2’ password file and assigned the ‘pass2’ vault ID:

```
ansible-vault edit --vault-id pass2@vault2 foo.yml
```

Rekeying Encrypted Files

Should you wish to change your password on a vault-encrypted file or files, you can do so with the `rekey` command:

```
ansible-vault rekey foo.yml bar.yml baz.yml
```

This command can rekey multiple data files at once and will ask for the original password and also the new password.

To rekey files encrypted with the ‘preprod2’ vault ID and the ‘ppold’ file and be prompted for the new password:

```
ansible-vault rekey --vault-id preprod2@ppold --new-vault-id preprod2@prompt foo.yml bar.  
↪.yml baz.yml
```

A different ID could have been set for the rekeyed files by passing it to `--new-vault-id`.

Encrypting Unencrypted Files

If you have existing files that you wish to encrypt, use the `ansible-vault encrypt` command. This command can operate on multiple files at once:

```
ansible-vault encrypt foo.yml bar.yml baz.yml
```

To encrypt existing files with the ‘project’ ID and be prompted for the password:

```
ansible-vault encrypt --vault-id project@prompt foo.yml bar.yml baz.yml
```

注解: It is technically possible to separately encrypt files or strings with the *same* vault ID but *different* passwords, if different password files or prompted passwords are provided each time. This could be desirable if you use vault IDs as references to classes of passwords (rather than a single password) and you always know which specific password or file to use in context. However this may be an unnecessarily complex use-case. If two files are encrypted with the same vault ID but different passwords by accident, you can use the *rekey* command to fix the issue.

Decrypting Encrypted Files

If you have existing files that you no longer want to keep encrypted, you can permanently decrypt them by running the `ansible-vault decrypt` command. This command will save them unencrypted to the disk, so be sure you do not want `ansible-vault edit` instead:

```
ansible-vault decrypt foo.yml bar.yml baz.yml
```

Viewing Encrypted Files

If you want to view the contents of an encrypted file without editing it, you can use the `ansible-vault view` command:

```
ansible-vault view foo.yml bar.yml baz.yml
```

Use `encrypt_string` to create encrypted variables to embed in yaml

The `ansible-vault encrypt_string` command will encrypt and format a provided string into a format that can be included in ansible-playbook YAML files.

To encrypt a string provided as a cli arg:

```
ansible-vault encrypt_string --vault-password-file a_password_file 'foobar' --name 'the_
↪secret'
```

Result:

```
the_secret: !vault |
    $ANSIBLE_VAULT;1.1;AES256
    62313365396662343061393464336163383764373764613633653634306231386433626436623361
    6134333665353966363534333632666535333761666131620a663537646436643839616531643561
    63396265333966386166373632626539326166353965363262633030333630313338646335303630
    3438626666666137650a353638643435666633633964366338633066623234616432373231333331
    6564
```

To use a vault-id label for ‘dev’ vault-id:

```
ansible-vault encrypt_string --vault-id dev@a_password_file 'foooodev' --name 'the_dev_
↪secret'
```

Result:

```
the_dev_secret: !vault |
    $ANSIBLE_VAULT;1.2;AES256;dev
    ↪
    ↪30613233633461343837653833666333643061636561303338373661313838333565653635353162
    ↪
    ↪3263363434623733343538653462613064333634333464660a663633623939393439316636633863
    ↪
    ↪61636237636537333938306331383339353265363239643939666639386530626330633337633833
    ↪
    ↪6664656334373166630a363736393262666465663432613932613036303963343263623137386239(下页继续)
```

(续上页)

6330

To encrypt a string read from stdin and name it 'db_password' :

```
echo -n 'letmein' | ansible-vault encrypt_string --vault-id dev@a_password_file --stdin-
↪name 'db_password'
```

警告: This method leaves the string in your shell history. Do not use it outside of testing.

Result:

```
Reading plaintext input from stdin. (ctrl-d to end input)
db_password: !vault |
           $ANSIBLE_VAULT;1.2;AES256;dev
           ↪
           ↪61323931353866666336306139373937316366366138656131323863373866376666353364373761
           ↪
           ↪3539633234313836346435323766306164626134376564330a373530313635343535343133316133
           ↪
           ↪36643666306434616266376434363239346433643238336464643566386135356334303736353136
           ↪
           ↪6565633133366366360a326566323363363936613664616364623437336130623133343530333739
           3039
```

To be prompted for a string to encrypt, encrypt it, and give it the name 'new_user_password' :

```
ansible-vault encrypt_string --vault-id dev@a_password_file --stdin-name 'new_user_
↪password'
```

Output:

```
Reading plaintext input from stdin. (ctrl-d to end input)
```

User enters 'hunter2' and hits ctrl-d.

警告: Do not press Enter after supplying the string. That will add a newline to the encrypted value.

Result:

```

new_user_password: !vault |
    $ANSIBLE_VAULT;1.2;AES256;dev
    37636561366636643464376336303466613062633537323632306566653533383833366462366662
    6565353063303065303831323539656138653863353230620a653638643639333133306331336365
    62373737623337616130386137373461306535383538373162316263386165376131623631323434
    3866363862363335620a376466656164383032633338306162326639643635663936623939666238
    3161

```

See also *Single Encrypted Variable*

After you added the encrypted value to a var file (vars.yml), you can see the original value using the debug module.

```

ansible localhost -m debug -a var="new_user_password" -e "@vars.yml" --ask-vault-pass
Vault password:

localhost | SUCCESS => {
    "new_user_password": "hunter2"
}

```

Providing Vault Passwords

When all data is encrypted using a single password the `--ask-vault-pass` or `--vault-password-file` cli options should be used.

For example, to use a password store in the text file `/path/to/my/vault-password-file`:

```
ansible-playbook --vault-password-file /path/to/my/vault-password-file site.yml
```

To prompt for a password:

```
ansible-playbook --ask-vault-pass site.yml
```

To get the password from a vault password executable script `my-vault-password.py`:

```
ansible-playbook --vault-password-file my-vault-password.py
```

The config option `DEFAULT_VAULT_PASSWORD_FILE` can be used to specify a vault password file so that the `--vault-password-file` cli option does not have to be specified every time.

Labelling Vaults

Since Ansible 2.4 the `--vault-id` can be used to indicate which vault ID ('dev' , 'prod' , 'cloud' , etc) a password is for as well as how to source the password (prompt, a file path, etc).

By default the vault-id label is only a hint, any values encrypted with the password will be decrypted. The config option `DEFAULT_VAULT_ID_MATCH` can be set to require the vault id to match the vault ID used when the value was encrypted. This can reduce errors when different values are encrypted with different passwords.

For example, to use a password file `dev-password` for the vault-id 'dev' :

```
ansible-playbook --vault-id dev@dev-password site.yml
```

To prompt for the password for the 'dev' vault ID:

```
ansible-playbook --vault-id dev@prompt site.yml
```

To get the 'dev' vault ID password from an executable script `my-vault-password.py`:

```
ansible-playbook --vault-id dev@my-vault-password.py
```

The config option `DEFAULT_VAULT_IDENTITY_LIST` can be used to specify a default vault ID and password source so that the `--vault-id` cli option does not have to be specified every time.

The `--vault-id` option can also be used without specifying a vault-id. This behaviour is equivalent to `--ask-vault-pass` or `--vault-password-file` so is rarely used.

For example, to use a password file `dev-password`:

```
ansible-playbook --vault-id dev-password site.yml
```

To prompt for the password:

```
ansible-playbook --vault-id @prompt site.yml
```

To get the password from an executable script `my-vault-password.py`:

```
ansible-playbook --vault-id my-vault-password.py
```

注解: Prior to Ansible 2.4, the `--vault-id` option is not supported so `--ask-vault-pass` or `--vault-password-file` must be used.

Multiple Vault Passwords

Ansible 2.4 and later support using multiple vault passwords, `--vault-id` can be provided multiple times. For example, to use a ‘dev’ password read from a file and to be prompted for the ‘prod’ password:

```
ansible-playbook --vault-id dev@dev-password --vault-id prod@prompt site.yml
```

By default the vault ID labels (dev, prod etc.) are only hints, Ansible will attempt to decrypt vault content with each password. The password with the same label as the encrypted data will be tried first, after that each vault secret will be tried in the order they were provided on the command line.

Where the encrypted data doesn't have a label, or the label doesn't match any of the provided labels, the passwords will be tried in the order they are specified.

In the above case, the ‘dev’ password will be tried first, then the ‘prod’ password for cases where Ansible doesn't know which vault ID is used to encrypt something.

To add a vault ID label to the encrypted data use the `--vault-id` option with a label when encrypting the data.

The `DEFAULT_VAULT_ID_MATCH` config option can be set so that Ansible will only use the password with the same label as the encrypted data. This is more efficient and may be more predictable when multiple passwords are used.

The config option `DEFAULT_VAULT_IDENTITY_LIST` can have multiple values which is equivalent to multiple `--vault-id` cli options.

The `--vault-id` can be used in lieu of the `--vault-password-file` or `--ask-vault-pass` options, or it can be used in combination with them.

When using `ansible-vault` commands that encrypt content (`ansible-vault encrypt`, `ansible-vault encrypt_string`, etc) only one vault-id can be used.

Vault Password Client Scripts

When implementing a script to obtain a vault password it may be convenient to know which vault ID label was requested. For example a script loading passwords from a secret manager may want to use the vault ID label to pick either the ‘dev’ or ‘prod’ password.

Since Ansible 2.5 this is supported through the use of Client Scripts. A Client Script is an executable script with a name ending in `-client`. Client Scripts are used to obtain vault passwords in the same way as any other executable script. For example:

```
ansible-playbook --vault-id dev@contrib/vault/vault-keyring-client.py
```

The difference is in the implementation of the script. Client Scripts are executed with a `--vault-id` option so they know which vault ID label was requested. So the above Ansible execution results in the below execution of the Client Script:

```
contrib/vault/vault-keyring-client.py --vault-id dev
```

`contrib/vault/vault-keyring-client.py` is an example of Client Script that loads passwords from the system keyring.

Speeding Up Vault Operations

If you have many encrypted files, decrypting them at startup may cause a perceptible delay. To speed this up, install the cryptography package:

```
pip install cryptography
```

Vault Format

A vault encrypted file is a UTF-8 encoded txt file.

The file format includes a newline terminated header.

For example:

```
$ANSIBLE_VAULT;1.1;AES256
```

or:

```
$ANSIBLE_VAULT;1.2;AES256;vault-id-label
```

The header contains the vault format id, the vault format version, the vault cipher, and a vault-id label (with format version 1.2), separated by semi-colons ‘;’

The first field `$ANSIBLE_VAULT` is the format id. Currently `$ANSIBLE_VAULT` is the only valid file format id. This is used to identify files that are vault encrypted (via `vault.is_encrypted_file()`).

The second field (1.X) is the vault format version. All supported versions of ansible will currently default to ‘1.1’ or ‘1.2’ if a labeled vault-id is supplied.

The ‘1.0’ format is supported for reading only (and will be converted automatically to the ‘1.1’ format on write). The format version is currently used as an exact string compare only (version numbers are not currently ‘compared’).

The third field (AES256) identifies the cipher algorithm used to encrypt the data. Currently, the only supported cipher is ‘AES256’ . [vault format 1.0 used ‘AES’ , but current code always uses ‘AES256’]

The fourth field (`vault-id-label`) identifies the vault-id label used to encrypt the data. For example using a vault-id of `dev@prompt` results in a vault-id-label of ‘dev’ being used.

Note: In the future, the header could change. Anything after the vault id and version can be considered to depend on the vault format version. This includes the cipher id, and any additional fields that could be after that.

The rest of the content of the file is the ‘vaulttext’. The vaulttext is a text armored version of the encrypted ciphertext. Each line will be 80 characters wide, except for the last line which may be shorter.

Vault Payload Format 1.1 - 1.2

The vaulttext is a concatenation of the ciphertext and a SHA256 digest with the result ‘hexlified’.

‘hexlify’ refers to the `hexlify()` method of the Python Standard Library’s `binascii` module.

`hexlify()`’ed result of:

- `hexlify()`’ed string of the salt, followed by a newline (`0x0a`)
- `hexlify()`’ed string of the crypted HMAC, followed by a newline. The HMAC is:
 - a [RFC2104](#) style HMAC
 - * inputs are:
 - The AES256 encrypted ciphertext
 - A PBKDF2 key. This key, the cipher key, and the cipher IV are generated from:
 - the salt, in bytes
 - 10000 iterations
 - SHA256() algorithm
 - the first 32 bytes are the cipher key
 - the second 32 bytes are the HMAC key
 - remaining 16 bytes are the cipher IV
- `hexlify()`’ed string of the ciphertext. The ciphertext is:
- AES256 encrypted data. The data is encrypted using:
 - AES-CTR stream cipher
 - cipher key
 - IV
 - a 128 bit counter block seeded from an integer IV
 - the plaintext

- * the original plaintext
- * padding up to the AES256 blocksize. (The data used for padding is based on [RFC5652](#))

1.3.12 Sample Ansible setup

You have learned about playbooks, inventory, roles, and variables. This section pulls all those elements together, outlining a sample setup for automating a web service. You can find more example playbooks illustrating these patterns in our [ansible-examples repository](#). (NOTE: These may not use all of the features in the latest release, but are still an excellent reference!).

The sample setup organizes playbooks, roles, inventory, and variables files by function, with tags at the play and task level for greater granularity and control. This is a powerful and flexible approach, but there are other ways to organize Ansible content. Your usage of Ansible should fit your needs, not ours, so feel free to modify this approach and organize your content as you see fit.

- *Sample directory layout*
- *Alternative directory layout*
- *Sample group and host variables*
- *Sample playbooks organized by function*
- *Sample task and handler files in a function-based role*
- *What the sample setup enables*
- *Organizing for deployment or configuration*
- *Using local Ansible modules*

Sample directory layout

This layout organizes most tasks in roles, with a single inventory file for each environment and a few playbooks in the top-level directory:

```
production          # inventory file for production servers
staging             # inventory file for staging environment

group_vars/
  group1.yml         # here we assign variables to particular groups
  group2.yml
host_vars/
  hostname1.yml      # here we assign variables to particular systems
```

(下页继续)

(续上页)

```

hostname2.yml

library/                # if any custom modules, put them here (optional)
module_utils/           # if any custom module_utils to support modules, put them here
↳ (optional)
filter_plugins/         # if any custom filter plugins, put them here (optional)

site.yml                # master playbook
webservers.yml          # playbook for webserver tier
dbservers.yml           # playbook for dbserver tier
tasks/                  # task files included from playbooks
    webservers-extra.yml # <-- avoids confusing playbook with task files

roles/
    common/              # this hierarchy represents a "role"
        tasks/           #
            main.yml      # <-- tasks file can include smaller files if warranted
        handlers/        #
            main.yml      # <-- handlers file
        templates/       # <-- files for use with the template resource
            ntp.conf.j2   # <----- templates end in .j2
        files/           #
            bar.txt       # <-- files for use with the copy resource
            foo.sh        # <-- script files for use with the script resource
        vars/            #
            main.yml      # <-- variables associated with this role
        defaults/        #
            main.yml      # <-- default lower priority variables for this role
        meta/            #
            main.yml      # <-- role dependencies
        library/         # roles can also include custom modules
        module_utils/    # roles can also include custom module_utils
        lookup_plugins/  # or other types of plugins, like lookup in this case

    webtier/             # same kind of structure as "common" was above, done for the
↳ webtier role
        monitoring/      # ""
        fooapp/          # ""

```

Alternative directory layout

Alternatively you can put each inventory file with its `group_vars`/`host_vars` in a separate directory. This is particularly useful if your `group_vars`/`host_vars` don't have that much in common in different environments. The layout could look something like this:

```
inventories/
  production/
    hosts                # inventory file for production servers
    group_vars/
      group1.yml         # here we assign variables to particular groups
      group2.yml
    host_vars/
      hostname1.yml      # here we assign variables to particular systems
      hostname2.yml

  staging/
    hosts                # inventory file for staging environment
    group_vars/
      group1.yml         # here we assign variables to particular groups
      group2.yml
    host_vars/
      stagehost1.yml     # here we assign variables to particular systems
      stagehost2.yml

library/
module_utils/
filter_plugins/

site.yml
webservers.yml
dbservers.yml

roles/
  common/
  webtier/
  monitoring/
  fooapp/
```

This layout gives you more flexibility for larger environments, as well as a total separation of inventory variables between different environments. However, this approach is harder to maintain, because there are more files. For more information on organizing group and host variables, see [编排主机和组变量](#).

Sample group and host variables

These sample group and host variables files record the variable values that apply to each machine or group of machines. For instance, the data center in Atlanta has its own NTP servers, so when setting up `ntp.conf`, we should use them:

```
---
# file: group_vars/atlanta
ntp: ntp-atlanta.example.com
backup: backup-atlanta.example.com
```

Similarly, the webserver have some configuration that does not apply to the database servers:

```
---
# file: group_vars/webserver
apacheMaxRequestsPerChild: 3000
apacheMaxClients: 900
```

Default values, or values that are universally true, belong in a file called `group_vars/all`:

```
---
# file: group_vars/all
ntp: ntp-boston.example.com
backup: backup-boston.example.com
```

If necessary, you can define specific hardware variance in systems in a `host_vars` file:

```
---
# file: host_vars/db-bos-1.example.com
foo_agent_port: 86
bar_agent_port: 99
```

Again, if you are using *dynamic inventory*, Ansible creates many dynamic groups automatically. So a tag like “`class:webserver`” would load in variables from the file “`group_vars/ec2_tag_class_webserver`” automatically.

Sample playbooks organized by function

With this setup, a single playbook can define all the infrastructure. The `site.yml` playbook imports two other playbooks, one for the webserver and one for the database servers:

```
---
# file: site.yml
```

(下页继续)

(续上页)

```
- import_playbook: webservers.yml
- import_playbook: dbservers.yml
```

The webservers.yml file, also at the top level, maps the configuration of the webservers group to the roles related to the webservers group:

```
---
# file: webservers.yml
- hosts: webservers
  roles:
    - common
    - webtier
```

With this setup, you can configure your whole infrastructure by “running” site.yml, or run a subset by running webservers.yml. This is analogous to the Ansible “--limit” parameter but a little more explicit:

```
ansible-playbook site.yml --limit webservers
ansible-playbook webservers.yml
```

Sample task and handler files in a function-based role

Ansible loads any file called main.yml in a role sub-directory. This sample tasks/main.yml file is simple - it sets up NTP, but it could do more if we wanted:

```
---
# file: roles/common/tasks/main.yml

- name: be sure ntp is installed
  yum:
    name: ntp
    state: present
  tags: ntp

- name: be sure ntp is configured
  template:
    src: ntp.conf.j2
    dest: /etc/ntp.conf
  notify:
    - restart ntpd
  tags: ntp
```

(下页继续)

(续上页)

```
- name: be sure ntpd is running and enabled
  service:
    name: ntpd
    state: started
    enabled: yes
  tags: ntp
```

Here is an example handlers file. As a review, handlers are only fired when certain tasks report changes, and are run at the end of each play:

```
---
# file: roles/common/handlers/main.yml
- name: restart ntpd
  service:
    name: ntpd
    state: restarted
```

See [Roles](#) for more information.

What the sample setup enables

The basic organizational structure described above enables a lot of different automation options. To reconfigure your entire infrastructure:

```
ansible-playbook -i production site.yml
```

To reconfigure NTP on everything:

```
ansible-playbook -i production site.yml --tags ntp
```

To reconfigure only the webserver:

```
ansible-playbook -i production webserver.yml
```

To reconfigure only the webserver in Boston:

```
ansible-playbook -i production webserver.yml --limit boston
```

To reconfigure only the first 10 webserver in Boston, and then the next 10:

```
ansible-playbook -i production webservers.yml --limit boston[0:9]
ansible-playbook -i production webservers.yml --limit boston[10:19]
```

The sample setup also supports basic ad-hoc commands:

```
ansible boston -i production -m ping
ansible boston -i production -m command -a '/sbin/reboot'
```

To discover what tasks would run or what hostnames would be affected by a particular Ansible command:

```
# confirm what task names would be run if I ran this command and said "just ntp tasks"
ansible-playbook -i production webservers.yml --tags ntp --list-tasks

# confirm what hostnames might be communicated with if I said "limit to boston"
ansible-playbook -i production webservers.yml --limit boston --list-hosts
```

Organizing for deployment or configuration

The sample setup models a typical configuration topology. When doing multi-tier deployments, there are going to be some additional playbooks that hop between tiers to roll out an application. In this case, ‘site.yml’ may be augmented by playbooks like ‘deploy_exampledotcom.yml’ but the general concepts still apply. Ansible allows you to deploy and configure using the same tool, so you would likely reuse groups and keep the OS configuration in separate playbooks or roles from the app deployment.

Consider “playbooks” as a sports metaphor – you can have one set of plays to use against all your infrastructure and situational plays that you use at different times and for different purposes.

Using local Ansible modules

If a playbook has a `./library` directory relative to its YAML file, this directory can be used to add Ansible modules that will automatically be in the Ansible module path. This is a great way to keep modules that go with a playbook together. This is shown in the directory structure example at the start of this section.

参见:

[YAML Syntax](#) Learn about YAML syntax

[Working With Playbooks](#) Review the basic playbook features

all_modules Learn about available modules

[Should you develop a module?](#) Learn how to extend Ansible by writing your own modules

[Pattern: 正则匹配主机和组](#) Learn about how to select hosts

[GitHub examples directory](#) Complete playbook files from the github project source

[Mailing List](#) Questions? Help? Ideas? Stop by the list on Google Groups

1.3.13 Working With Modules

Introduction to modules

Modules (also referred to as “task plugins” or “library plugins”) are discrete units of code that can be used from the command line or in a playbook task. Ansible executes each module, usually on the remote target node, and collects return values.

You can execute modules from the command line:

```
ansible webserver -m service -a "name=httpd state=started"
ansible webserver -m ping
ansible webserver -m command -a "/sbin/reboot -t now"
```

Each module supports taking arguments. Nearly all modules take **key=value** arguments, space delimited. Some modules take no arguments, and the command/shell modules simply take the string of the command you want to run.

From playbooks, Ansible modules are executed in a very similar way:

```
- name: reboot the servers
  action: command /sbin/reboot -t now
```

Which can be abbreviated to:

```
- name: reboot the servers
  command: /sbin/reboot -t now
```

Another way to pass arguments to a module is using YAML syntax also called ‘complex args’

```
- name: restart webserver
  service:
    name: httpd
    state: restarted
```

All modules return JSON format data. This means modules can be written in any programming language. Modules should be idempotent, and should avoid making any changes if they detect that the current state matches the desired final state. When used in an Ansible playbook, modules can trigger ‘change events’ in the form of notifying ‘handlers’ to run additional tasks.

Documentation for each module can be accessed from the command line with the `ansible-doc` tool:

```
ansible-doc yum
```

For a list of all available modules, see the Module Docs, or run the following at a command prompt:

```
ansible-doc -l
```

参见:

ad-hoc 命令操作指引 Examples of using modules in /usr/bin/ansible

Working With Playbooks Examples of using modules with /usr/bin/ansible-playbook

Should you develop a module? How to write your own modules

Python API Examples of using modules with the Python API

Mailing List Questions? Help? Ideas? Stop by the list on Google Groups

irc.freenode.net #ansible IRC chat channel

Return Values

Topics

- *Return Values*

- *Common*

- * *backup_file*
 - * *changed*
 - * *diff*
 - * *failed*
 - * *invocation*
 - * *msg*
 - * *rc*
 - * *results*
 - * *skipped*
 - * *stderr*
 - * *stderr_lines*
 - * *stdout*
 - * *stdout_lines*

```

– Internal use

* ansible_facts

* exception

* warnings

* deprecations

```

Ansible modules normally return a data structure that can be registered into a variable, or seen directly when output by the *ansible* program. Each module can optionally document its own unique return values (visible through *ansible-doc* and on the *main docsite*).

This document covers return values common to all modules.

注解: Some of these keys might be set by Ansible itself once it processes the module's return information.

Common

backup_file

For those modules that implement *backup=no/yes* when manipulating files, a path to the backup file created.

changed

A boolean indicating if the task had to make changes.

diff

Information on differences between the previous and current state. Often a dictionary with entries **before** and **after**, which will then be formatted by the callback plugin to a diff view.

failed

A boolean that indicates if the task was failed or not.

invocation

Information on how the module was invoked.

msg

A string with a generic message relayed to the user.

rc

Some modules execute command line utilities or are geared for executing commands directly (raw, shell, command, etc), this field contains ‘return code’ of these utilities.

results

If this key exists, it indicates that a loop was present for the task and that it contains a list of the normal module ‘result’ per item.

skipped

A boolean that indicates if the task was skipped or not

stderr

Some modules execute command line utilities or are geared for executing commands directly (raw, shell, command, etc), this field contains the error output of these utilities.

stderr_lines

When *stderr* is returned we also always provide this field which is a list of strings, one item per line from the original.

stdout

Some modules execute command line utilities or are geared for executing commands directly (raw, shell, command, etc). This field contains the normal output of these utilities.

stdout_lines

When *stdout* is returned, Ansible always provides a list of strings, each containing one item per line from the original output.

Internal use

These keys can be added by modules but will be removed from registered variables; they are ‘consumed’ by Ansible itself.

ansible_facts

This key should contain a dictionary which will be appended to the facts assigned to the host. These will be directly accessible and don’t require using a registered variable.

exception

This key can contain traceback information caused by an exception in a module. It will only be displayed on high verbosity (-vvv).

warnings

This key contains a list of strings that will be presented to the user.

deprecations

This key contains a list of dictionaries that will be presented to the user. Keys of the dictionaries are *msg* and *version*, values are string, value for the *version* key can be an empty string.

参见:

all_modules Learn about available modules

GitHub modules directory Browse source of core and extras modules

Mailing List Development mailing list

irc.freenode.net #ansible IRC chat channel

Module Maintenance & Support

- *Maintenance*
 - *Core*
 - *Network*
 - *Certified*

- *Community*
- *Issue Reporting*
- *Support*

Maintenance

To clarify who maintains each included module, adding features and fixing bugs, each included module now has associated metadata that provides information about maintenance.

Core

Core Maintained modules are maintained by the Ansible Engineering Team. These modules are integral to the basic foundations of the Ansible distribution.

Network

Network Maintained modules are maintained by the Ansible Network Team. Please note there are additional networking modules that are categorized as Certified or Community not maintained by Ansible.

Certified

Certified modules are maintained by Ansible Partners.

Community

Community Maintained modules are submitted and maintained by the Ansible community. These modules are not maintained by Ansible, and are included as a convenience.

Issue Reporting

If you believe you have found a bug in a module and are already running the latest stable or development version of Ansible, first look at the [issue tracker in the Ansible repo](#) to see if an issue has already been filed. If not, please file one.

Should you have a question rather than a bug report, inquiries are welcome on the [ansible-project Google group](#) or on Ansible's “#ansible” channel, located on [irc.freenode.net](#).

For development-oriented topics, use the [ansible-devel Google group](#) or Ansible's #ansible and #ansible-devel channels, located on [irc.freenode.net](#). You should also read the *Community Guide*, *Testing Ansible*, and the *Developer Guide*.

The modules are hosted on GitHub in a subdirectory of the [Ansible](#) repo.

NOTE: If you have a Red Hat Ansible Automation product subscription, please follow the standard issue reporting process via the [Red Hat Customer Portal](#).

Support

For more information on how included Ansible modules are supported by Red Hat, please refer to the following [knowledge base article](#) as well as other resources on the [Red Hat Customer Portal](#).

参见:

Module index A complete list of all available modules.

ad-hoc 命令操作指引 Examples of using modules in `/usr/bin/ansible`

Working With Playbooks Examples of using modules with `/usr/bin/ansible-playbook`

Should you develop a module? How to write your own modules

List of Ansible Certified Modules High level list of Ansible certified modules from Partners

Mailing List Questions? Help? Ideas? Stop by the list on Google Groups

irc.freenode.net #ansible IRC chat channel

Ansible ships with a number of modules (called the ‘module library’) that can be executed directly on remote hosts or through *Playbooks*.

Users can also write their own modules. These modules can control system resources, like services, packages, or files (anything really), or handle executing system commands.

参见:

ad-hoc 命令操作指引 Examples of using modules in `/usr/bin/ansible`

Intro to Playbooks Introduction to using modules with `/usr/bin/ansible-playbook`

Ansible module development: getting started How to write your own modules

Python API Examples of using modules with the Python API

Interpreter Discovery Configuring the right Python interpreter on target hosts

Mailing List Questions? Help? Ideas? Stop by the list on Google Groups

irc.freenode.net #ansible IRC chat channel

1.3.14 Ansible and BSD

Managing BSD machines is different from managing Linux/Unix machines. If you have managed nodes running BSD, review these topics.

- *Connecting to BSD nodes*
- *Bootstrapping BSD*
- *Setting the Python interpreter*
- *Which modules are available?*
- *Using BSD as the control node*
- *BSD facts*
- *BSD efforts and contributions*

Connecting to BSD nodes

Ansible connects to managed nodes using OpenSSH by default. This works on BSD if you use SSH keys for authentication. However, if you use SSH passwords for authentication, Ansible relies on `sshpass`. Most versions of `sshpass` do not deal well with BSD login prompts, so when using SSH passwords against BSD machines, use `paramiko` to connect instead of OpenSSH. You can do this in `ansible.cfg` globally or you can set it as an inventory/group/host variable. For example:

```
[freebsd]
mybsdhost1 ansible_connection=paramiko
```

Bootstrapping BSD

Ansible is agentless by default, however, it requires Python on managed nodes. Only the `raw` module will operate without Python. Although this module can be used to bootstrap Ansible and install Python on BSD variants (see below), it is very limited and the use of Python is required to make full use of Ansible's features.

The following example installs Python 2.7 which includes the `json` library required for full functionality of Ansible. On your control machine you can execute the following for most versions of FreeBSD:

```
ansible -m raw -a "pkg install -y python27" mybsdhost1
```

Or for OpenBSD:

```
ansible -m raw -a "pkg_add python%3.7"
```

Once this is done you can now use other Ansible modules apart from the `raw` module.

注解: This example demonstrated using `pkg` on FreeBSD and `pkg_add` on OpenBSD, however you should

be able to substitute the appropriate package tool for your BSD; the package name may also differ. Refer to the package list or documentation of the BSD variant you are using for the exact Python package name you intend to install.

Setting the Python interpreter

To support a variety of Unix/Linux operating systems and distributions, Ansible cannot always rely on the existing environment or `env` variables to locate the correct Python binary. By default, modules point at `/usr/bin/python` as this is the most common location. On BSD variants, this path may differ, so it is advised to inform Ansible of the binary's location, through the `ansible_python_interpreter` inventory variable. For example:

```
[freebsd:vars]
ansible_python_interpreter=/usr/local/bin/python2.7
[openbsd:vars]
ansible_python_interpreter=/usr/local/bin/python3.7
```

If you use additional plugins beyond those bundled with Ansible, you can set similar variables for `bash`, `perl` or `ruby`, depending on how the plugin is written. For example:

```
[freebsd:vars]
ansible_python_interpreter=/usr/local/bin/python
ansible_perl_interpreter=/usr/bin/perl5
```

Which modules are available?

The majority of the core Ansible modules are written for a combination of Linux/Unix machines and other generic services, so most should function well on the BSDs with the obvious exception of those that are aimed at Linux-only technologies (such as LVG).

Using BSD as the control node

Using BSD as the control machine is as simple as installing the Ansible package for your BSD variant or by following the `pip` or 'from source' instructions.

BSD facts

Ansible gathers facts from the BSDs in a similar manner to Linux machines, but since the data, names and structures can vary for network, disks and other devices, one should expect the output to be slightly different yet still familiar to a BSD administrator.

BSD efforts and contributions

BSD support is important to us at Ansible. Even though the majority of our contributors use and target Linux we have an active BSD community and strive to be as BSD-friendly as possible. Please feel free to report any issues or incompatibilities you discover with BSD; pull requests with an included fix are also welcome!

参见:

ad-hoc 命令操作指引 Examples of basic commands

Working With Playbooks Learning ansible' s configuration management language

Should you develop a module? How to write modules

Mailing List Questions? Help? Ideas? Stop by the list on Google Groups

irc.freenode.net #ansible IRC chat channel

1.3.15 Windows Guides

The following sections provide information on managing Windows hosts with Ansible.

Because Windows is a non-POSIX-compliant operating system, there are differences between how Ansible interacts with them and the way Windows works. These guides will highlight some of the differences between Linux/Unix hosts and hosts running Windows.

Setting up a Windows Host

This document discusses the setup that is required before Ansible can communicate with a Microsoft Windows host.

- *Host Requirements*
 - *Upgrading PowerShell and .NET Framework*
 - *WinRM Memory Hotfix*
- *WinRM Setup*
 - *WinRM Listener*
 - * *Setup WinRM Listener*
 - * *Delete WinRM Listener*
 - *WinRM Service Options*
 - *Common WinRM Issues*

- * *HTTP 401/Credentials Rejected*
- * *HTTP 500 Error*
- * *Timeout Errors*
- * *Connection Refused Errors*
- *Windows SSH Setup*
 - *Installing Win32-OpenSSH*
 - *Configuring the Win32-OpenSSH shell*
 - *Win32-OpenSSH Authentication*
 - *Configuring Ansible for SSH on Windows*
 - *Known issues with SSH on Windows*

Host Requirements

For Ansible to communicate to a Windows host and use Windows modules, the Windows host must meet these requirements:

- Ansible can generally manage Windows versions under current and extended support from Microsoft. Ansible can manage desktop OSs including Windows 7, 8.1, and 10, and server OSs including Windows Server 2008, 2008 R2, 2012, 2012 R2, 2016, and 2019.
- Ansible requires PowerShell 3.0 or newer and at least .NET 4.0 to be installed on the Windows host.
- A WinRM listener should be created and activated. More details for this can be found below.

注解: While these are the base requirements for Ansible connectivity, some Ansible modules have additional requirements, such as a newer OS or PowerShell version. Please consult the module's documentation page to determine whether a host meets those requirements.

Upgrading PowerShell and .NET Framework

Ansible requires PowerShell version 3.0 and .NET Framework 4.0 or newer to function on older operating systems like Server 2008 and Windows 7. The base image does not meet this requirement. You can use the [Upgrade-PowerShell.ps1](#) script to update these.

This is an example of how to run this script from PowerShell:

```
$url = "https://raw.githubusercontent.com/jborean93/ansible-windows/master/scripts/
↳Upgrade-PowerShell.ps1"
$file = "$env:temp\Upgrade-PowerShell.ps1"
$username = "Administrator"
$password = "Password"

(New-Object -TypeName System.Net.WebClient).DownloadFile($url, $file)
Set-ExecutionPolicy -ExecutionPolicy Unrestricted -Force

# Version can be 3.0, 4.0 or 5.1
&$file -Version 5.1 -Username $username -Password $password -Verbose
```

Once completed, you will need to remove auto logon and set the execution policy back to the default of **Restricted**. You can do this with the following PowerShell commands:

```
# This isn't needed but is a good security practice to complete
Set-ExecutionPolicy -ExecutionPolicy Restricted -Force

$reg_winlogon_path = "HKLM:\Software\Microsoft\Windows NT\CurrentVersion\Winlogon"
Set-ItemProperty -Path $reg_winlogon_path -Name AutoAdminLogon -Value 0
Remove-ItemProperty -Path $reg_winlogon_path -Name DefaultUserName -ErrorAction_
↳SilentlyContinue
Remove-ItemProperty -Path $reg_winlogon_path -Name DefaultPassword -ErrorAction_
↳SilentlyContinue
```

The script works by checking to see what programs need to be installed (such as .NET Framework 4.5.2) and what PowerShell version is required. If a reboot is required and the **username** and **password** parameters are set, the script will automatically reboot and logon when it comes back up from the reboot. The script will continue until no more actions are required and the PowerShell version matches the target version. If the **username** and **password** parameters are not set, the script will prompt the user to manually reboot and logon when required. When the user is next logged in, the script will continue where it left off and the process continues until no more actions are required.

注解: If running on Server 2008, then SP2 must be installed. If running on Server 2008 R2 or Windows 7, then SP1 must be installed.

注解: Windows Server 2008 can only install PowerShell 3.0; specifying a newer version will result in the script failing.

注解: The `username` and `password` parameters are stored in plain text in the registry. Make sure the cleanup commands are run after the script finishes to ensure no credentials are still stored on the host.

WinRM Memory Hotfix

When running on PowerShell v3.0, there is a bug with the WinRM service that limits the amount of memory available to WinRM. Without this hotfix installed, Ansible will fail to execute certain commands on the Windows host. These hotfixes should be installed as part of the system bootstrapping or imaging process. The script `Install-WMF3Hotfix.ps1` can be used to install the hotfix on affected hosts.

The following PowerShell command will install the hotfix:

```
$url = "https://raw.githubusercontent.com/jborean93/ansible-windows/master/scripts/
↳ Install-WMF3Hotfix.ps1"
$file = "$env:temp\Install-WMF3Hotfix.ps1"

(New-Object -TypeName System.Net.WebClient).DownloadFile($url, $file)
powershell.exe -ExecutionPolicy Bypass -File $file -Verbose
```

For more details, please refer to the [Hotfix document](#) from Microsoft.

WinRM Setup

Once Powershell has been upgraded to at least version 3.0, the final step is for the WinRM service to be configured so that Ansible can connect to it. There are two main components of the WinRM service that governs how Ansible can interface with the Windows host: the `listener` and the `service` configuration settings.

Details about each component can be read below, but the script `ConfigureRemotingForAnsible.ps1` can be used to set up the basics. This script sets up both HTTP and HTTPS listeners with a self-signed certificate and enables the `Basic` authentication option on the service.

To use this script, run the following in PowerShell:

```
$url = "https://raw.githubusercontent.com/ansible/ansible/devel/examples/scripts/
↳ ConfigureRemotingForAnsible.ps1"
$file = "$env:temp\ConfigureRemotingForAnsible.ps1"

(New-Object -TypeName System.Net.WebClient).DownloadFile($url, $file)

powershell.exe -ExecutionPolicy Bypass -File $file
```

There are different switches and parameters (like `-EnableCredSSP` and `-ForceNewSSLCert`) that can be set alongside this script. The documentation for these options are located at the top of the script itself.

注解: The `ConfigureRemotingForAnsible.ps1` script is intended for training and development purposes only and should not be used in a production environment, since it enables settings (like `Basic` authentication) that can be inherently insecure.

WinRM Listener

The WinRM services listens for requests on one or more ports. Each of these ports must have a listener created and configured.

To view the current listeners that are running on the WinRM service, run the following command:

```
winrm enumerate winrm/config/Listener
```

This will output something like:

```
Listener
  Address = *
  Transport = HTTP
  Port = 5985
  Hostname
  Enabled = true
  URLPrefix = wsman
  CertificateThumbprint
  ListeningOn = 10.0.2.15, 127.0.0.1, 192.168.56.155, ::1, fe80::5efe:10.0.2.15%6,
↵ fe80::5efe:192.168.56.155%8, fe80::
ffff:ffff:ffff%2, fe80::203d:7d97:c2ed:ec78%3, fe80::e8ea:d765:2c69:7756%7

Listener
  Address = *
  Transport = HTTPS
  Port = 5986
  Hostname = SERVER2016
  Enabled = true
  URLPrefix = wsman
  CertificateThumbprint = E6CDAA82EEAF2ECE8546E05DB7F3E01AA47D76CE
  ListeningOn = 10.0.2.15, 127.0.0.1, 192.168.56.155, ::1, fe80::5efe:10.0.2.15%6,
↵ fe80::5efe:192.168.56.155%8, fe80::
ffff:ffff:ffff%2, fe80::203d:7d97:c2ed:ec78%3, fe80::e8ea:d765:2c69:7756%7
```


In the example above there are two listeners activated; one is listening on port 5985 over HTTP and the other is listening on port 5986 over HTTPS. Some of the key options that are useful to understand are:

- **Transport:** Whether the listener is run over HTTP or HTTPS, it is recommended to use a listener over HTTPS as the data is encrypted without any further changes required.
- **Port:** The port the listener runs on, by default it is 5985 for HTTP and 5986 for HTTPS. This port can be changed to whatever is required and corresponds to the host var `ansible_port`.
- **URLPrefix:** The URL prefix to listen on, by default it is `wsman`. If this is changed, the host var `ansible_winrm_path` must be set to the same value.
- **CertificateThumbprint:** If running over an HTTPS listener, this is the thumbprint of the certificate in the Windows Certificate Store that is used in the connection. To get the details of the certificate itself, run this command with the relevant certificate thumbprint in PowerShell:

```
$thumbprint = "E6CDAA82EEAF2ECE8546E05DB7F3E01AA47D76CE"
Get-ChildItem -Path cert:\LocalMachine\My -Recurse | Where-Object { $_.Thumbprint -
↪eq $thumbprint } | Select-Object *
```

Setup WinRM Listener

There are three ways to set up a WinRM listener:

- Using `winrm quickconfig` for HTTP or `winrm quickconfig -transport:https` for HTTPS. This is the easiest option to use when running outside of a domain environment and a simple listener is required. Unlike the other options, this process also has the added benefit of opening up the Firewall for the ports required and starts the WinRM service.
- Using Group Policy Objects. This is the best way to create a listener when the host is a member of a domain because the configuration is done automatically without any user input. For more information on group policy objects, see the [Group Policy Objects documentation](#).
- Using PowerShell to create the listener with a specific configuration. This can be done by running the following PowerShell commands:

```
$selector_set = @{
    Address = "*"
    Transport = "HTTPS"
}
$value_set = @{
    CertificateThumbprint = "E6CDAA82EEAF2ECE8546E05DB7F3E01AA47D76CE"
}

New-WSManInstance -ResourceURI "winrm/config/Listener" -SelectorSet $selector_set -
↪ValueSet $value_set
```

(下页继续)

(续上页)

To see the other options with this PowerShell cmdlet, see [New-WSManInstance](#).

注解: When creating an HTTPS listener, an existing certificate needs to be created and stored in the LocalMachine\My certificate store. Without a certificate being present in this store, most commands will fail.

Delete WinRM Listener

To remove a WinRM listener:

```
# Remove all listeners
Remove-Item -Path WSMan:\localhost\Listener\* -Recurse -Force

# Only remove listeners that are run over HTTPS
Get-ChildItem -Path WSMan:\localhost\Listener | Where-Object { $_.Keys -contains
  ↳ "Transport=HTTPS" } | Remove-Item -Recurse -Force
```

注解: The Keys object is an array of strings, so it can contain different values. By default it contains a key for Transport= and Address= which correspond to the values from winrm enumerate winrm/config/Listeners.

WinRM Service Options

There are a number of options that can be set to control the behavior of the WinRM service component, including authentication options and memory settings.

To get an output of the current service configuration options, run the following command:

```
winrm get winrm/config/Service
winrm get winrm/config/Winrs
```

This will output something like:

```
Service
  RootSDDL = O:NSG:BAD:P(A;;;GA;;;BA)(A;;;GR;;;IU)S:P(AU;FA;GA;;;WD)(AU;SA;GXGW;;;WD)
  MaxConcurrentOperations = 4294967295
  MaxConcurrentOperationsPerUser = 1500
```

(下页继续)

(续上页)

```

EnumerationTimeoutms = 240000
MaxConnections = 300
MaxPacketRetrievalTimeSeconds = 120
AllowUnencrypted = false
Auth
    Basic = true
    Kerberos = true
    Negotiate = true
    Certificate = true
    CredSSP = true
    CbtHardeningLevel = Relaxed
DefaultPorts
    HTTP = 5985
    HTTPS = 5986
IPv4Filter = *
IPv6Filter = *
EnableCompatibilityHttpListener = false
EnableCompatibilityHttpsListener = false
CertificateThumbprint
AllowRemoteAccess = true

Winrs
    AllowRemoteShellAccess = true
    IdleTimeout = 7200000
    MaxConcurrentUsers = 2147483647
    MaxShellRunTime = 2147483647
    MaxProcessesPerShell = 2147483647
    MaxMemoryPerShellMB = 2147483647
    MaxShellsPerUser = 2147483647

```

While many of these options should rarely be changed, a few can easily impact the operations over WinRM and are useful to understand. Some of the important options are:

- **Service\AllowUnencrypted:** This option defines whether WinRM will allow traffic that is run over HTTP without message encryption. Message level encryption is only possible when `ansible_winrm_transport` is `ntlm`, `kerberos` or `credssp`. By default this is `false` and should only be set to `true` when debugging WinRM messages.
- **Service\Auth*:** These flags define what authentication options are allowed with the WinRM service. By default, `Negotiate` (NTLM) and `Kerberos` are enabled.
- **Service\Auth\CbtHardeningLevel:** Specifies whether channel binding tokens are not verified (`None`),

verified but not required (Relaxed), or verified and required (Strict). CBT is only used when connecting with NTLM or Kerberos over HTTPS.

- **Service\CertificateThumbprint:** This is the thumbprint of the certificate used to encrypt the TLS channel used with CredSSP authentication. By default this is empty; a self-signed certificate is generated when the WinRM service starts and is used in the TLS process.
- **Winrs\MaxShellRunTime:** This is the maximum time, in milliseconds, that a remote command is allowed to execute.
- **Winrs\MaxMemoryPerShellMB:** This is the maximum amount of memory allocated per shell, including the shell's child processes.

To modify a setting under the **Service** key in PowerShell:

```
# substitute {path} with the path to the option after winrm/config/Service
Set-Item -Path WSMAN:\localhost\Service\{path} -Value "value here"

# for example, to change Service\Auth\CbtHardeningLevel run
Set-Item -Path WSMAN:\localhost\Service\Auth\CbtHardeningLevel -Value Strict
```

To modify a setting under the **Winrs** key in PowerShell:

```
# Substitute {path} with the path to the option after winrm/config/Winrs
Set-Item -Path WSMAN:\localhost\Shell\{path} -Value "value here"

# For example, to change Winrs\MaxShellRunTime run
Set-Item -Path WSMAN:\localhost\Shell\MaxShellRunTime -Value 2147483647
```

注解: If running in a domain environment, some of these options are set by GPO and cannot be changed on the host itself. When a key has been configured with GPO, it contains the text `[Source="GPO"]` next to the value.

Common WinRM Issues

Because WinRM has a wide range of configuration options, it can be difficult to setup and configure. Because of this complexity, issues that are shown by Ansible could in fact be issues with the host setup instead.

One easy way to determine whether a problem is a host issue is to run the following command from another Windows host to connect to the target Windows host:

```
# Test out HTTP
winrs -r:http://server:5985/wsman -u:Username -p:Password ipconfig
```

(下页继续)

(续上页)

```
# Test out HTTPS (will fail if the cert is not verifiable)
winrs -r:https://server:5986/wsman -u:Username -p:Password -ssl ipconfig

# Test out HTTPS, ignoring certificate verification
$username = "Username"
$password = ConvertTo-SecureString -String "Password" -AsPlainText -Force
$cred = New-Object -TypeName System.Management.Automation.PSCredential -ArgumentList
↪ $username, $password

$session_option = New-PSSessionOption -SkipCACheck -SkipCNCheck -SkipRevocationCheck
Invoke-Command -ComputerName server -UseSSL -ScriptBlock { ipconfig } -Credential $cred -
↪ SessionOption $session_option
```

If this fails, the issue is probably related to the WinRM setup. If it works, the issue may not be related to the WinRM setup; please continue reading for more troubleshooting suggestions.

HTTP 401/Credentials Rejected

A HTTP 401 error indicates the authentication process failed during the initial connection. Some things to check for this are:

- Verify that the credentials are correct and set properly in your inventory with `ansible_user` and `ansible_password`
- Ensure that the user is a member of the local Administrators group or has been explicitly granted access (a connection test with the `winrs` command can be used to rule this out).
- Make sure that the authentication option set by `ansible_winrm_transport` is enabled under `Service\Auth*`
- If running over HTTP and not HTTPS, use `ntlm`, `kerberos` or `credssp` with `ansible_winrm_message_encryption: auto` to enable message encryption. If using another authentication option or if the installed `pywinrm` version cannot be upgraded, the `Service\AllowUnencrypted` can be set to `true` but this is only recommended for troubleshooting
- Ensure the downstream packages `pywinrm`, `requests-ntlm`, `requests-kerberos`, and/or `requests-credssp` are up to date using `pip`.
- If using Kerberos authentication, ensure that `Service\Auth\CbtHardeningLevel` is not set to `Strict`.
- When using Basic or Certificate authentication, make sure that the user is a local account and not a domain account. Domain accounts do not work with Basic and Certificate authentication.

HTTP 500 Error

These indicate an error has occurred with the WinRM service. Some things to check for include:

- Verify that the number of current open shells has not exceeded either `WinRsMaxShellsPerUser` or any of the other Winrs quotas haven't been exceeded.

Timeout Errors

These usually indicate an error with the network connection where Ansible is unable to reach the host. Some things to check for include:

- Make sure the firewall is not set to block the configured WinRM listener ports
- Ensure that a WinRM listener is enabled on the port and path set by the host vars
- Ensure that the `winrm` service is running on the Windows host and configured for automatic start

Connection Refused Errors

These usually indicate an error when trying to communicate with the WinRM service on the host. Some things to check for:

- Ensure that the WinRM service is up and running on the host. Use `(Get-Service -Name winrm).Status` to get the status of the service.
- Check that the host firewall is allowing traffic over the WinRM port. By default this is 5985 for HTTP and 5986 for HTTPS.

Sometimes an installer may restart the WinRM or HTTP service and cause this error. The best way to deal with this is to use `win_psexec` from another Windows host.

Windows SSH Setup

Ansible 2.8 has added an experimental SSH connection for Windows managed nodes.

警告: Use this feature at your own risk! Using SSH with Windows is experimental, the implementation may make backwards incompatible changes in feature releases. The server side components can be unreliable depending on the version that is installed.

Installing Win32-OpenSSH

The first step to using SSH with Windows is to install the [Win32-OpenSSH](#) service on the Windows host. Microsoft offers a way to install Win32-OpenSSH through a Windows capability but currently the version

that is installed through this process is too old to work with Ansible. To install Win32-OpenSSH for use with Ansible, select one of these three installation options:

- Manually install the service, following the [install instructions](#) from Microsoft.
- Use `win_chocolatey` to install the service:

```
- name: install the Win32-OpenSSH service
  win_chocolatey:
    name: openssh
    package_params: /SSHServerFeature
    state: present
```

- Use an existing Ansible Galaxy role like `jborean93.win_openssh`:

```
# Make sure the role has been downloaded first
ansible-galaxy install jborean93.win_openssh

# main.yml
- name: install Win32-OpenSSH service
  hosts: windows
  gather_facts: no
  roles:
    - role: jborean93.win_openssh
      opt_openssh_setup_service: True
```

注解: Win32-OpenSSH is still a beta product and is constantly being updated to include new features and bugfixes. If you are using SSH as a connection option for Windows, it is highly recommend you install the latest release from one of the 3 methods above.

Configuring the Win32-OpenSSH shell

By default Win32-OpenSSH will use `cmd.exe` as a shell. To configure a different shell, use an Ansible task to define the registry setting:

```
- name: set the default shell to PowerShell
  win_regedit:
    path: HKLM:\SOFTWARE\OpenSSH
    name: DefaultShell
    data: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
    type: string
```

(下页继续)

(续上页)

```
state: present

# Or revert the settings back to the default, cmd
- name: set the default shell to cmd
  win_regedit:
    path: HKLM:\SOFTWARE\OpenSSH
    name: DefaultShell
    state: absent
```

Win32-OpenSSH Authentication

Win32-OpenSSH authentication with Windows is similar to SSH authentication on Unix/Linux hosts. You can use a plaintext password or SSH public key authentication, add public keys to an `authorized_key` file in the `.ssh` folder of the user's profile directory, and configure the service using the `sshd_config` file used by the SSH service as you would on a Unix/Linux host.

When using SSH key authentication with Ansible, the remote session won't have access to the user's credentials and will fail when attempting to access a network resource. This is also known as the double-hop or credential delegation issue. There are two ways to work around this issue:

- Use plaintext password auth by setting `ansible_password`
- Use `become` on the task with the credentials of the user that needs access to the remote resource

Configuring Ansible for SSH on Windows

To configure Ansible to use SSH for Windows hosts, you must set two connection variables:

- set `ansible_connection` to `ssh`
- set `ansible_shell_type` to `cmd` or `powershell`

The `ansible_shell_type` variable should reflect the `DefaultShell` configured on the Windows host. Set to `cmd` for the default shell or set to `powershell` if the `DefaultShell` has been changed to PowerShell.

Known issues with SSH on Windows

Using SSH with Windows is experimental, and we expect to uncover more issues. Here are the known ones:

- Win32-OpenSSH versions older than `v7.9.0.0p1-Beta` do not work when `powershell` is the shell type
- While SCP should work, SFTP is the recommended SSH file transfer mechanism to use when copying or fetching a file

参见:

Intro to Playbooks An introduction to playbooks

Tips and tricks Best practices advice

List of Windows Modules Windows specific module list, all implemented in PowerShell

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

Windows Remote Management

Unlike Linux/Unix hosts, which use SSH by default, Windows hosts are configured with WinRM. This topic covers how to configure and use WinRM with Ansible.

Topics

- *What is WinRM?*
- *Authentication Options*
 - *Basic*
 - *Certificate*
 - * *Generate a Certificate*
 - * *Import a Certificate to the Certificate Store*
 - * *Mapping a Certificate to an Account*
 - *NTLM*
 - *Kerberos*
 - * *Installing the Kerberos Library*
 - * *Configuring Host Kerberos*
 - * *Automatic Kerberos Ticket Management*
 - * *Manual Kerberos Ticket Management*
 - * *Troubleshooting Kerberos*
 - *CredSSP*
 - * *Installing CredSSP Library*
 - * *CredSSP and TLS 1.2*
 - * *Set CredSSP Certificate*
- *Non-Administrator Accounts*

- *WinRM Encryption*
- *Inventory Options*
- *IPv6 Addresses*
- *HTTPS Certificate Validation*
- *TLS 1.2 Support*
- *Limitations*

What is WinRM?

WinRM is a management protocol used by Windows to remotely communicate with another server. It is a SOAP-based protocol that communicates over HTTP/HTTPS, and is included in all recent Windows operating systems. Since Windows Server 2012, WinRM has been enabled by default, but in most cases extra configuration is required to use WinRM with Ansible.

Ansible uses the `pywinrm` package to communicate with Windows servers over WinRM. It is not installed by default with the Ansible package, but can be installed by running the following:

```
pip install "pywinrm>=0.3.0"
```

注解: on distributions with multiple python versions, use `pip2` or `pip2.x`, where `x` matches the python minor version Ansible is running under.

警告: Using the `winrm` or `psrp` connection plugins in Ansible on MacOS in the latest releases typically fail. This is a known problem that occurs deep within the Python stack and cannot be changed by Ansible. The only workaround today is to set the environment variable `no_proxy=*` and avoid using Kerberos auth.

Authentication Options

When connecting to a Windows host, there are several different options that can be used when authenticating with an account. The authentication type may be set on inventory hosts or groups with the `ansible_winrm_transport` variable.

The following matrix is a high level overview of the options:

Option	Local Accounts	Active Directory Accounts	Credential Delegation	HTTP Encryption
Basic	Yes	No	No	No
Certificate	Yes	No	No	No
Kerberos	No	Yes	Yes	Yes
NTLM	Yes	Yes	No	Yes
CredSSP	Yes	Yes	Yes	Yes

Basic

Basic authentication is one of the simplest authentication options to use, but is also the most insecure. This is because the username and password are simply base64 encoded, and if a secure channel is not in use (eg, HTTPS) then it can be decoded by anyone. Basic authentication can only be used for local accounts (not domain accounts).

The following example shows host vars configured for basic authentication:

```
ansible_user: LocalUsername
ansible_password: Password
ansible_connection: winrm
ansible_winrm_transport: basic
```

Basic authentication is not enabled by default on a Windows host but can be enabled by running the following in PowerShell:

```
Set-Item -Path WSMan:\localhost\Service\Auth\Basic -Value $true
```

Certificate

Certificate authentication uses certificates as keys similar to SSH key pairs, but the file format and key generation process is different.

The following example shows host vars configured for certificate authentication:

```
ansible_connection: winrm
ansible_winrm_cert_pem: /path/to/certificate/public/key.pem
ansible_winrm_cert_key_pem: /path/to/certificate/private/key.pem
ansible_winrm_transport: certificate
```

Certificate authentication is not enabled by default on a Windows host but can be enabled by running the following in PowerShell:

```
Set-Item -Path WSMAN:\localhost\Service\Auth\Certificate -Value $true
```

注解: Encrypted private keys cannot be used as the urllib3 library that is used by Ansible for WinRM does not support this functionality.

Generate a Certificate

A certificate must be generated before it can be mapped to a local user. This can be done using one of the following methods:

- OpenSSL
- PowerShell, using the `New-SelfSignedCertificate` cmdlet
- Active Directory Certificate Services

Active Directory Certificate Services is beyond of scope in this documentation but may be the best option to use when running in a domain environment. For more information, see the [Active Directory Certificate Services documentation](#).

注解: Using the PowerShell cmdlet `New-SelfSignedCertificate` to generate a certificate for authentication only works when being generated from a Windows 10 or Windows Server 2012 R2 host or later. OpenSSL is still required to extract the private key from the PFX certificate to a PEM file for Ansible to use.

To generate a certificate with OpenSSL:

```
# Set the name of the local user that will have the key mapped to
USERNAME="username"

cat > openssl.conf << EOL
distinguished_name = req_distinguished_name
[req_distinguished_name]
[v3_req_client]
extendedKeyUsage = clientAuth
subjectAltName = otherName:1.3.6.1.4.1.311.20.2.3;UTF8:$USERNAME@localhost
EOL

export OPENSSL_CONF=openssl.conf
openssl req -x509 -nodes -days 3650 -newkey rsa:2048 -out cert.pem -outform PEM -keyout ↵
↵cert_key.pem -subj "/CN=$USERNAME" -extensions v3_req_client
```

(下页继续)

(续上页)

```
rm openssl.conf
```

To generate a certificate with `New-SelfSignedCertificate`:

```
# Set the name of the local user that will have the key mapped
$username = "username"
$output_path = "C:\temp"

# Instead of generating a file, the cert will be added to the personal
# LocalComputer folder in the certificate store
$cert = New-SelfSignedCertificate -Type Custom `
    -Subject "CN=$username" `
    -TextExtension @("2.5.29.37={text}1.3.6.1.5.5.7.3.2","2.5.29.17={text}upn=
↪$username@localhost") `
    -KeyUsage DigitalSignature,KeyEncipherment `
    -KeyAlgorithm RSA `
    -KeyLength 2048

# Export the public key
$pem_output = @()
$pem_output += "-----BEGIN CERTIFICATE-----"
$pem_output += [System.Convert]::ToBase64String($cert.RawData) -replace ".{64}", "$&\n"
$pem_output += "-----END CERTIFICATE-----"
[System.IO.File]::WriteAllLines("$output_path\cert.pem", $pem_output)

# Export the private key in a PFX file
[System.IO.File]::WriteAllBytes("$output_path\cert.pfx", $cert.Export("Pfx"))
```

注解: To convert the PFX file to a private key that `pywinrm` can use, run the following command with OpenSSL `openssl pkcs12 -in cert.pfx -nocerts -nodes -out cert_key.pem -passin pass: -passout pass:`

Import a Certificate to the Certificate Store

Once a certificate has been generated, the issuing certificate needs to be imported into the **Trusted Root Certificate Authorities** of the **LocalMachine** store, and the client certificate public key must be present in the **Trusted People** folder of the **LocalMachine** store. For this example, both the issuing certificate and public key are the same.

Following example shows how to import the issuing certificate:

```
$cert = New-Object -TypeName System.Security.Cryptography.X509Certificates.  
↪X509Certificate2  
$cert.Import("cert.pem")  
  
$store_name = [System.Security.Cryptography.X509Certificates.StoreName]::Root  
$store_location = [System.Security.Cryptography.X509Certificates.  
↪StoreLocation]::LocalMachine  
$store = New-Object -TypeName System.Security.Cryptography.X509Certificates.X509Store -  
↪ArgumentList $store_name, $store_location  
$store.Open("MaxAllowed")  
$store.Add($cert)  
$store.Close()
```

注解: If using ADCS to generate the certificate, then the issuing certificate will already be imported and this step can be skipped.

The code to import the client certificate public key is:

```
$cert = New-Object -TypeName System.Security.Cryptography.X509Certificates.  
↪X509Certificate2  
$cert.Import("cert.pem")  
  
$store_name = [System.Security.Cryptography.X509Certificates.StoreName]::TrustedPeople  
$store_location = [System.Security.Cryptography.X509Certificates.  
↪StoreLocation]::LocalMachine  
$store = New-Object -TypeName System.Security.Cryptography.X509Certificates.X509Store -  
↪ArgumentList $store_name, $store_location  
$store.Open("MaxAllowed")  
$store.Add($cert)  
$store.Close()
```

Mapping a Certificate to an Account

Once the certificate has been imported, map it to the local user account:

```
$username = "username"  
$password = ConvertTo-SecureString -String "password" -AsPlainText -Force  
$credential = New-Object -TypeName System.Management.Automation.PSCredential -  
↪ArgumentList $username, $password
```

(下页继续)

(续上页)

```
# This is the issuer thumbprint which in the case of a self generated cert
# is the public key thumbprint, additional logic may be required for other
# scenarios
$thumbprint = (Get-ChildItem -Path cert:\LocalMachine\root | Where-Object { $_.Subject -
↪eq "CN=$username" }).Thumbprint

New-Item -Path WSMAN:\localhost\ClientCertificate `
    -Subject "$username@localhost" `
    -URI * `
    -Issuer $thumbprint `
    -Credential $credential `
    -Force
```

Once this is complete, the hostvar `ansible_winrm_cert_pem` should be set to the path of the public key and the `ansible_winrm_cert_key_pem` variable should be set to the path of the private key.

NTLM

NTLM is an older authentication mechanism used by Microsoft that can support both local and domain accounts. NTLM is enabled by default on the WinRM service, so no setup is required before using it.

NTLM is the easiest authentication protocol to use and is more secure than **Basic** authentication. If running in a domain environment, **Kerberos** should be used instead of NTLM.

Kerberos has several advantages over using NTLM:

- NTLM is an older protocol and does not support newer encryption protocols.
- NTLM is slower to authenticate because it requires more round trips to the host in the authentication stage.
- Unlike Kerberos, NTLM does not allow credential delegation.

This example shows host variables configured to use NTLM authentication:

```
ansible_user: LocalUsername
ansible_password: Password
ansible_connection: winrm
ansible_winrm_transport: ntlm
```

Kerberos

Kerberos is the recommended authentication option to use when running in a domain environment. Kerberos supports features like credential delegation and message encryption over HTTP and is one of the more secure options that is available through WinRM.

Kerberos requires some additional setup work on the Ansible host before it can be used properly.

The following example shows host vars configured for Kerberos authentication:

```
ansible_user: username@MY.DOMAIN.COM
ansible_password: Password
ansible_connection: winrm
ansible_winrm_transport: kerberos
```

As of Ansible version 2.3, the Kerberos ticket will be created based on `ansible_user` and `ansible_password`. If running on an older version of Ansible or when `ansible_winrm_kinit_mode` is `manual`, a Kerberos ticket must already be obtained. See below for more details.

There are some extra host variables that can be set:

```
ansible_winrm_kinit_mode: managed/manual (manual means Ansible will not obtain a ticket)
ansible_winrm_kinit_cmd: the kinit binary to use to obtain a Kerberos ticket (default to kinit)
ansible_winrm_service: overrides the SPN prefix that is used, the default is `HTTP` and should rarely ever need changing
ansible_winrm_kerberos_delegation: allows the credentials to traverse multiple hops
ansible_winrm_kerberos_hostname_override: the hostname to be used for the kerberos exchange
```

Installing the Kerberos Library

Some system dependencies that must be installed prior to using Kerberos. The script below lists the dependencies based on the distro:

```
# Via Yum (RHEL/Centos/Fedora)
yum -y install python-devel krb5-devel krb5-libs krb5-workstation

# Via Apt (Ubuntu)
sudo apt-get install python-dev libkrb5-dev krb5-user

# Via Portage (Gentoo)
emerge -av app-crypt/mit-krb5
```

(下页继续)

(续上页)

```
emerge -av dev-python/setuptools

# Via Pkg (FreeBSD)
sudo pkg install security/krb5

# Via OpenCSW (Solaris)
pkgadd -d http://get.opencsw.org/now
/opt/csw/bin/pkgutil -U
/opt/csw/bin/pkgutil -y -i libkrb5_3

# Via Pacman (Arch Linux)
pacman -S krb5
```

Once the dependencies have been installed, the `python-kerberos` wrapper can be install using `pip`:

```
pip install pywinrm[kerberos]
```

注解: While Ansible has supported Kerberos auth through `pywinrm` for some time, optional features or more secure options may only be available in newer versions of the `pywinrm` and/or `pykerberos` libraries. It is recommended you upgrade each version to the latest available to resolve any warnings or errors. This can be done through tools like `pip` or a system package manager like `dnf`, `yum`, `apt` but the package names and versions available may differ between tools.

Configuring Host Kerberos

Once the dependencies have been installed, Kerberos needs to be configured so that it can communicate with a domain. This configuration is done through the `/etc/krb5.conf` file, which is installed with the packages in the script above.

To configure Kerberos, in the section that starts with:

```
[realms]
```

Add the full domain name and the fully qualified domain names of the primary and secondary Active Directory domain controllers. It should look something like this:

```
[realms]
  MY.DOMAIN.COM = {
    kdc = domain-controller1.my.domain.com
```

(下页继续)

(续上页)

```
kdc = domain-controller2.my.domain.com
}
```

In the section that starts with:

```
[domain_realm]
```

Add a line like the following for each domain that Ansible needs access for:

```
[domain_realm]
.my.domain.com = MY.DOMAIN.COM
```

You can configure other settings in this file such as the default domain. See [krb5.conf](#) for more details.

Automatic Kerberos Ticket Management

Ansible version 2.3 and later defaults to automatically managing Kerberos tickets when both `ansible_user` and `ansible_password` are specified for a host. In this process, a new ticket is created in a temporary credential cache for each host. This is done before each task executes to minimize the chance of ticket expiration. The temporary credential caches are deleted after each task completes and will not interfere with the default credential cache.

To disable automatic ticket management, set `ansible_winrm_kinit_mode=manual` via the inventory.

Automatic ticket management requires a standard `kinit` binary on the control host system path. To specify a different location or binary name, set the `ansible_winrm_kinit_cmd` hostvar to the fully qualified path to a MIT `krb5` `kinit`-compatible binary.

Manual Kerberos Ticket Management

To manually manage Kerberos tickets, the `kinit` binary is used. To obtain a new ticket the following command is used:

```
kinit username@MY.DOMAIN.COM
```

注解: The domain must match the configured Kerberos realm exactly, and must be in upper case.

To see what tickets (if any) have been acquired, use the following command:

```
klist
```

To destroy all the tickets that have been acquired, use the following command:

```
kdestroy
```

Troubleshooting Kerberos

Kerberos is reliant on a properly-configured environment to work. To troubleshoot Kerberos issues, ensure that:

- The hostname set for the Windows host is the FQDN and not an IP address.
- The forward and reverse DNS lookups are working properly in the domain. To test this, ping the windows host by name and then use the ip address returned with `nslookup`. The same name should be returned when using `nslookup` on the IP address.
- The Ansible host's clock is synchronized with the domain controller. Kerberos is time sensitive, and a little clock drift can cause the ticket generation process to fail.
- Ensure that the fully qualified domain name for the domain is configured in the `krb5.conf` file. To check this, run:

```
kinit -C username@MY.DOMAIN.COM  
klist
```

If the domain name returned by `klist` is different from the one requested, an alias is being used. The `krb5.conf` file needs to be updated so that the fully qualified domain name is used and not an alias.

- If the default kerberos tooling has been replaced or modified (some IdM solutions may do this), this may cause issues when installing or upgrading the Python Kerberos library. As of the time of this writing, this library is called `pykerberos` and is known to work with both MIT and Heimdal Kerberos libraries. To resolve `pykerberos` installation issues, ensure the system dependencies for Kerberos have been met (see: [Installing the Kerberos Library](#)), remove any custom Kerberos tooling paths from the `PATH` environment variable, and retry the installation of Python Kerberos library package.

CredSSP

CredSSP authentication is a newer authentication protocol that allows credential delegation. This is achieved by encrypting the username and password after authentication has succeeded and sending that to the server using the CredSSP protocol.

Because the username and password are sent to the server to be used for double hop authentication, ensure that the hosts that the Windows host communicates with are not compromised and are trusted.

CredSSP can be used for both local and domain accounts and also supports message encryption over HTTP.

To use CredSSP authentication, the host vars are configured like so:

```
ansible_user: Username
ansible_password: Password
ansible_connection: winrm
ansible_winrm_transport: credssp
```

There are some extra host variables that can be set as shown below:

```
ansible_winrm_credssp_disable_tlsv1_2: when true, will not use TLS 1.2 in the CredSSP
↪auth process
```

CredSSP authentication is not enabled by default on a Windows host, but can be enabled by running the following in PowerShell:

```
Enable-WSManCredSSP -Role Server -Force
```

Installing CredSSP Library

The `requests-credssp` wrapper can be installed using `pip`:

```
pip install pywinrm[credssp]
```

CredSSP and TLS 1.2

By default the `requests-credssp` library is configured to authenticate over the TLS 1.2 protocol. TLS 1.2 is installed and enabled by default for Windows Server 2012 and Windows 8 and more recent releases.

There are two ways that older hosts can be used with CredSSP:

- Install and enable a hotfix to enable TLS 1.2 support (recommended for Server 2008 R2 and Windows 7).
- Set `ansible_winrm_credssp_disable_tlsv1_2=True` in the inventory to run over TLS 1.0. This is the only option when connecting to Windows Server 2008, which has no way of supporting TLS 1.2

See *[TLS 1.2 Support](#)* for more information on how to enable TLS 1.2 on the Windows host.

Set CredSSP Certificate

CredSSP works by encrypting the credentials through the TLS protocol and uses a self-signed certificate by default. The `CertificateThumbprint` option under the WinRM service configuration can be used to specify the thumbprint of another certificate.

注解: This certificate configuration is independent of the WinRM listener certificate. With CredSSP, message transport still occurs over the WinRM listener, but the TLS-encrypted messages inside the channel use the service-level certificate.

To explicitly set the certificate to use for CredSSP:

```
# Note the value $certificate_thumbprint will be different in each
# situation, this needs to be set based on the cert that is used.
$certificate_thumbprint = "7C8DCBD5427AFEE6560F4AF524E325915F51172C"

# Set the thumbprint value
Set-Item -Path WSMan:\localhost\Service\CertificateThumbprint -Value $certificate_
↪thumbprint
```

Non-Administrator Accounts

WinRM is configured by default to only allow connections from accounts in the local **Administrators** group. This can be changed by running:

```
winrm configSDDL default
```

This will display an ACL editor, where new users or groups may be added. To run commands over WinRM, users and groups must have at least the **Read** and **Execute** permissions enabled.

While non-administrative accounts can be used with WinRM, most typical server administration tasks require some level of administrative access, so the utility is usually limited.

WinRM Encryption

By default WinRM will fail to work when running over an unencrypted channel. The WinRM protocol considers the channel to be encrypted if using TLS over HTTP (HTTPS) or using message level encryption. Using WinRM with TLS is the recommended option as it works with all authentication options, but requires a certificate to be created and used on the WinRM listener.

The `ConfigureRemotingForAnsible.ps1` creates a self-signed certificate and creates the listener with that certificate. If in a domain environment, AD CS can also create a certificate for the host that is issued by the domain itself.

If using HTTPS is not an option, then HTTP can be used when the authentication option is **NTLM**, **Kerberos** or **CredSSP**. These protocols will encrypt the WinRM payload with their own encryption method before sending it to the server. The message-level encryption is not used when running over HTTPS because the

encryption uses the more secure TLS protocol instead. If both transport and message encryption is required, set `ansible_winrm_message_encryption=always` in the host vars.

注解: Message encryption over HTTP requires `pywinrm>=0.3.0`.

A last resort is to disable the encryption requirement on the Windows host. This should only be used for development and debugging purposes, as anything sent from Ansible can be viewed, manipulated and also the remote session can completely be taken over by anyone on the same network. To disable the encryption requirement:

```
Set-Item -Path WSMan:\localhost\Service\AllowUnencrypted -Value $true
```

注解: Do not disable the encryption check unless it is absolutely required. Doing so could allow sensitive information like credentials and files to be intercepted by others on the network.

Inventory Options

Ansible's Windows support relies on a few standard variables to indicate the username, password, and connection type of the remote hosts. These variables are most easily set up in the inventory, but can be set on the `host_vars/` `group_vars` level.

When setting up the inventory, the following variables are required:

```
# It is suggested that these be encrypted with ansible-vault:
# ansible-vault edit group_vars/windows.yml
ansible_connection: winrm

# May also be passed on the command-line via --user
ansible_user: Administrator

# May also be supplied at runtime with --ask-pass
ansible_password: SecretPasswordGoesHere
```

Using the variables above, Ansible will connect to the Windows host with Basic authentication through HTTPS. If `ansible_user` has a UPN value like `username@MY.DOMAIN.COM` then the authentication option will automatically attempt to use Kerberos unless `ansible_winrm_transport` has been set to something other than `kerberos`.

The following custom inventory variables are also supported for additional configuration of WinRM connections:

- **ansible_port**: The port WinRM will run over, HTTPS is 5986 which is the default while HTTP is 5985
- **ansible_winrm_scheme**: Specify the connection scheme (`http` or `https`) to use for the WinRM connection. Ansible uses `https` by default unless **ansible_port** is 5985
- **ansible_winrm_path**: Specify an alternate path to the WinRM endpoint, Ansible uses `/wsman` by default
- **ansible_winrm_realm**: Specify the realm to use for Kerberos authentication. If **ansible_user** contains `@`, Ansible will use the part of the username after `@` by default
- **ansible_winrm_transport**: Specify one or more authentication transport options as a comma-separated list. By default, Ansible will use `kerberos`, `basic` if the `kerberos` module is installed and a realm is defined, otherwise it will be `plaintext`
- **ansible_winrm_server_cert_validation**: Specify the server certificate validation mode (`ignore` or `validate`). Ansible defaults to `validate` on Python 2.7.9 and higher, which will result in certificate validation errors against the Windows self-signed certificates. Unless verifiable certificates have been configured on the WinRM listeners, this should be set to `ignore`
- **ansible_winrm_operation_timeout_sec**: Increase the default timeout for WinRM operations, Ansible uses 20 by default
- **ansible_winrm_read_timeout_sec**: Increase the WinRM read timeout, Ansible uses 30 by default. Useful if there are intermittent network issues and read timeout errors keep occurring
- **ansible_winrm_message_encryption**: Specify the message encryption operation (`auto`, `always`, `never`) to use, Ansible uses `auto` by default. `auto` means message encryption is only used when **ansible_winrm_scheme** is `http` and **ansible_winrm_transport** supports message encryption. `always` means message encryption will always be used and `never` means message encryption will never be used
- **ansible_winrm_ca_trust_path**: Used to specify a different cacert container than the one used in the `certifi` module. See the HTTPS Certificate Validation section for more details.
- **ansible_winrm_send_cbt**: When using `ntlm` or `kerberos` over HTTPS, the authentication library will try to send channel binding tokens to mitigate against man in the middle attacks. This flag controls whether these bindings will be sent or not (default: `yes`).
- **ansible_winrm_***: Any additional keyword arguments supported by `winrm.Protocol` may be provided in place of `*`

In addition, there are also specific variables that need to be set for each authentication option. See the section on authentication above for more information.

注解: Ansible 2.0 has deprecated the “ssh” from `ansible_ssh_user`, `ansible_ssh_pass`, `ansible_ssh_host`, and `ansible_ssh_port` to become `ansible_user`, `ansible_password`, `ansible_host`, and `ansible_port`. If using a version of Ansible prior to 2.0, the older style (`ansible_ssh_*`) should be

used instead. The shorter variables are ignored, without warning, in older versions of Ansible.

注解: `ansible_winrm_message_encryption` is different from transport encryption done over TLS. The WinRM payload is still encrypted with TLS when run over HTTPS, even if `ansible_winrm_message_encryption=never`.

IPv6 Addresses

IPv6 addresses can be used instead of IPv4 addresses or hostnames. This option is normally set in an inventory. Ansible will attempt to parse the address using the `ipaddress` package and pass to `pywinrm` correctly.

When defining a host using an IPv6 address, just add the IPv6 address as you would an IPv4 address or hostname:

```
[windows-server]
2001:db8::1

[windows-server:vars]
ansible_user=username
ansible_password=password
ansible_connection=winrm
```

注解: The `ipaddress` library is only included by default in Python 3.x. To use IPv6 addresses in Python 2.7, make sure to run `pip install ipaddress` which installs a backported package.

HTTPS Certificate Validation

As part of the TLS protocol, the certificate is validated to ensure the host matches the subject and the client trusts the issuer of the server certificate. When using a self-signed certificate or setting `ansible_winrm_server_cert_validation: ignore` these security mechanisms are bypassed. While self signed certificates will always need the `ignore` flag, certificates that have been issued from a certificate authority can still be validated.

One of the more common ways of setting up a HTTPS listener in a domain environment is to use Active Directory Certificate Service (AD CS). AD CS is used to generate signed certificates from a Certificate Signing Request (CSR). If the WinRM HTTPS listener is using a certificate that has been signed by another authority, like AD CS, then Ansible can be set up to trust that issuer as part of the TLS handshake.

To get Ansible to trust a Certificate Authority (CA) like AD CS, the issuer certificate of the CA can be exported as a PEM encoded certificate. This certificate can then be copied locally to the Ansible controller and used as a source of certificate validation, otherwise known as a CA chain.

The CA chain can contain a single or multiple issuer certificates and each entry is contained on a new line. To then use the custom CA chain as part of the validation process, set `ansible_winrm_ca_trust_path` to the path of the file. If this variable is not set, the default CA chain is used instead which is located in the install path of the Python package `certifi`.

注解: Each HTTP call is done by the Python requests library which does not use the systems built-in certificate store as a trust authority. Certificate validation will fail if the server's certificate issuer is only added to the system's truststore.

TLS 1.2 Support

As WinRM runs over the HTTP protocol, using HTTPS means that the TLS protocol is used to encrypt the WinRM messages. TLS will automatically attempt to negotiate the best protocol and cipher suite that is available to both the client and the server. If a match cannot be found then Ansible will error out with a message similar to:

```
HTTPSConnectionPool(host='server', port=5986): Max retries exceeded with url: /wsman
↳ (Caused by SSLError(SSLError(1, '[SSL: UNSUPPORTED_PROTOCOL] unsupported protocol (_
↳ ssl.c:1056)')))
```

Commonly this is when the Windows host has not been configured to support TLS v1.2 but it could also mean the Ansible controller has an older OpenSSL version installed.

Windows 8 and Windows Server 2012 come with TLS v1.2 installed and enabled by default but older hosts, like Server 2008 R2 and Windows 7, have to be enabled manually.

注解: There is a bug with the TLS 1.2 patch for Server 2008 which will stop Ansible from connecting to the Windows host. This means that Server 2008 cannot be configured to use TLS 1.2. Server 2008 R2 and Windows 7 are not affected by this issue and can use TLS 1.2.

To verify what protocol the Windows host supports, you can run the following command on the Ansible controller:

```
openssl s_client -connect <hostname>:5986
```

The output will contain information about the TLS session and the `Protocol` line will display the version that was negotiated:

```

New, TLSv1/SSLv3, Cipher is ECDHE-RSA-AES256-SHA
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
    Protocol   : TLSv1
    Cipher     : ECDHE-RSA-AES256-SHA
    Session-ID: 962A00001C95D2A601BE1CCFA7831B85A7EEE897AECDBF3D9ECD4A3BE4F6AC9B
    Session-ID-ctx:
    Master-Key: ....
    Start Time: 1552976474
    Timeout    : 7200 (sec)
    Verify return code: 21 (unable to verify the first certificate)
---

New, TLSv1/SSLv3, Cipher is ECDHE-RSA-AES256-GCM-SHA384
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
    Protocol   : TLSv1.2
    Cipher     : ECDHE-RSA-AES256-GCM-SHA384
    Session-ID: AE16000050DA9FD44D03BB8839B64449805D9E43DBD670346D3D9E05D1AEEA84
    Session-ID-ctx:
    Master-Key: ....
    Start Time: 1552976538
    Timeout    : 7200 (sec)
    Verify return code: 21 (unable to verify the first certificate)

```

If the host is returning TLSv1 then it should be configured so that TLS v1.2 is enable. You can do this by running the following PowerShell script:

```

Function Enable-TLS12 {
    param(
        [ValidateSet("Server", "Client")]
        [String]$Component = "Server"
    )
}

```

(下页继续)

(续上页)

```

    $protocols_path =
    ↳ 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols'
        New-Item -Path "$protocols_path\TLS 1.2\$Component" -Force
        New-ItemProperty -Path "$protocols_path\TLS 1.2\$Component" -Name Enabled -Value 1 -
    ↳ Type DWORD -Force
        New-ItemProperty -Path "$protocols_path\TLS 1.2\$Component" -Name DisabledByDefault -
    ↳ Value 0 -Type DWORD -Force
    }

Enable-TLS12 -Component Server

# Not required but highly recommended to enable the Client side TLS 1.2 components
Enable-TLS12 -Component Client

Restart-Computer

```

The below Ansible tasks can also be used to enable TLS v1.2:

```

- name: enable TLSv1.2 support
  win_regedit:
    path:
    ↳ HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols\TLS 1.2\{{
    ↳ item.type }}
    name: '{{ item.property }}'
    data: '{{ item.value }}'
    type: dword
    state: present
  register: enable_tls12
  loop:
  - type: Server
    property: Enabled
    value: 1
  - type: Server
    property: DisabledByDefault
    value: 0
  - type: Client
    property: Enabled
    value: 1
  - type: Client

```

(下页继续)

(续上页)

```
property: DisabledByDefault
value: 0

- name: reboot if TLS config was applied
  win_reboot:
  when: enable_tls12 is changed
```

There are other ways to configure the TLS protocols as well as the cipher suites that are offered by the Windows host. One tool that can give you a GUI to manage these settings is [IIS Crypto](#) from Nartac Software.

Limitations

Due to the design of the WinRM protocol , there are a few limitations when using WinRM that can cause issues when creating playbooks for Ansible. These include:

- Credentials are not delegated for most authentication types, which causes authentication errors when accessing network resources or installing certain programs.
- Many calls to the Windows Update API are blocked when running over WinRM.
- Some programs fail to install with WinRM due to no credential delegation or because they access forbidden Windows API like WUA over WinRM.
- Commands under WinRM are done under a non-interactive session, which can prevent certain commands or executables from running.
- You cannot run a process that interacts with DPAPI, which is used by some installers (like Microsoft SQL Server).

Some of these limitations can be mitigated by doing one of the following:

- Set `ansible_winrm_transport` to `credssp` or `kerberos` (with `ansible_winrm_kerberos_delegation=true`) to bypass the double hop issue and access network resources
- Use `become` to bypass all WinRM restrictions and run a command as it would locally. Unlike using an authentication transport like `credssp`, this will also remove the non-interactive restriction and API restrictions like WUA and DPAPI
- Use a scheduled task to run a command which can be created with the `win_scheduled_task` module. Like `become`, this bypasses all WinRM restrictions but can only run a command and not modules.

参见:

Intro to Playbooks An introduction to playbooks

Tips and tricks Best practices advice

List of Windows Modules Windows specific module list, all implemented in PowerShell

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

Using Ansible and Windows

When using Ansible to manage Windows, many of the syntax and rules that apply for Unix/Linux hosts also apply to Windows, but there are still some differences when it comes to components like path separators and OS-specific tasks. This document covers details specific to using Ansible for Windows.

Topics

- *Use Cases*
 - *Installing Software*
 - *Installing Updates*
 - *Set Up Users and Groups*
 - * *Local*
 - * *Domain*
 - *Running Commands*
 - * *Choosing Command or Shell*
 - * *Argument Rules*
 - *Creating and Running a Scheduled Task*
- *Path Formatting for Windows*
 - *YAML Style*
 - *Legacy key=value Style*
- *Limitations*
- *Developing Windows Modules*

Use Cases

Ansible can be used to orchestrate a multitude of tasks on Windows servers. Below are some examples and info about common tasks.

Installing Software

There are three main ways that Ansible can be used to install software:

- Using the `win_chocolatey` module. This sources the program data from the default public [Chocolatey](#) repository. Internal repositories can be used instead by setting the `source` option.
- Using the `win_package` module. This installs software using an MSI or .exe installer from a local/network path or URL.
- Using the `win_command` or `win_shell` module to run an installer manually.

The `win_chocolatey` module is recommended since it has the most complete logic for checking to see if a package has already been installed and is up-to-date.

Below are some examples of using all three options to install 7-Zip:

```
# Install/uninstall with chocolatey
- name: Ensure 7-Zip is installed via Chocolatey
  win_chocolatey:
    name: 7zip
    state: present

- name: Ensure 7-Zip is not installed via Chocolatey
  win_chocolatey:
    name: 7zip
    state: absent

# Install/uninstall with win_package
- name: Download the 7-Zip package
  win_get_url:
    url: https://www.7-zip.org/a/7z1701-x64.msi
    dest: C:\temp\7z.msi

- name: Ensure 7-Zip is installed via win_package
  win_package:
    path: C:\temp\7z.msi
    state: present

- name: Ensure 7-Zip is not installed via win_package
  win_package:
    path: C:\temp\7z.msi
    state: absent
```

(下页继续)

(续上页)

```
# Install/uninstall with win_command
- name: Download the 7-Zip package
  win_get_url:
    url: https://www.7-zip.org/a/7z1701-x64.msi
    dest: C:\temp\7z.msi

- name: Check if 7-Zip is already installed
  win_reg_stat:
    name: HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\{23170F69-40C1-2702-
    ↪1701-000001000000}
    register: 7zip_installed

- name: Ensure 7-Zip is installed via win_command
  win_command: C:\Windows\System32\msiexec.exe /i C:\temp\7z.msi /qn /norestart
  when: 7zip_installed.exists == false

- name: Ensure 7-Zip is uninstalled via win_command
  win_command: C:\Windows\System32\msiexec.exe /x {23170F69-40C1-2702-1701-000001000000}
    ↪/qn /norestart
  when: 7zip_installed.exists == true
```

Some installers like Microsoft Office or SQL Server require credential delegation or access to components restricted by WinRM. The best method to bypass these issues is to use `become` with the task. With `become`, Ansible will run the installer as if it were run interactively on the host.

注解: Many installers do not properly pass back error information over WinRM. In these cases, if the install has been verified to work locally the recommended method is to use `become`.

注解: Some installers restart the WinRM or HTTP services, or cause them to become temporarily unavailable, making Ansible assume the system is unreachable.

Installing Updates

The `win_updates` and `win_hotfix` modules can be used to install updates or hotfixes on a host. The module `win_updates` is used to install multiple updates by category, while `win_hotfix` can be used to install a single update or hotfix file that has been downloaded locally.

注解: The `win_hotfix` module has a requirement that the DISM PowerShell cmdlets are present. These cmdlets were only added by default on Windows Server 2012 and newer and must be installed on older Windows hosts.

The following example shows how `win_updates` can be used:

```
- name: Install all critical and security updates
  win_updates:
    category_names:
      - CriticalUpdates
      - SecurityUpdates
    state: installed
    register: update_result

- name: Reboot host if required
  win_reboot:
    when: update_result.reboot_required
```

The following example show how `win_hotfix` can be used to install a single update or hotfix:

```
- name: Download KB3172729 for Server 2012 R2
  win_get_url:
    url: http://download.windowsupdate.com/d/msdownload/update/software/secu/2016/07/
    ↪ windows8.1-kb3172729-x64_e8003822a7ef4705cbb65623b72fd3cec73fe222.msu
    dest: C:\temp\KB3172729.msu

- name: Install hotfix
  win_hotfix:
    hotfix_kb: KB3172729
    source: C:\temp\KB3172729.msu
    state: present
    register: hotfix_result

- name: Reboot host if required
  win_reboot:
    when: hotfix_result.reboot_required
```

Set Up Users and Groups

Ansible can be used to create Windows users and groups both locally and on a domain.

Local

The modules `win_user`, `win_group` and `win_group_membership` manage Windows users, groups and group memberships locally.

The following is an example of creating local accounts and groups that can access a folder on the same host:

```
- name: Create local group to contain new users
  win_group:
    name: LocalGroup
    description: Allow access to C:\Development folder

- name: Create local user
  win_user:
    name: '{{ item.name }}'
    password: '{{ item.password }}'
    groups: LocalGroup
    update_password: no
    password_never_expires: yes
  loop:
    - name: User1
      password: Password1
    - name: User2
      password: Password2

- name: Create Development folder
  win_file:
    path: C:\Development
    state: directory

- name: Set ACL of Development folder
  win_acl:
    path: C:\Development
    rights: FullControl
    state: present
    type: allow
    user: LocalGroup

- name: Remove parent inheritance of Development folder
  win_acl_inheritance:
    path: C:\Development
    reorganize: yes
```

(下页继续)

(续上页)

```
state: absent
```

Domain

The modules `win_domain_user` and `win_domain_group` manages users and groups in a domain. The below is an example of ensuring a batch of domain users are created:

```
- name: Ensure each account is created
  win_domain_user:
    name: '{{ item.name }}'
    upn: '{{ item.name }}@MY.DOMAIN.COM'
    password: '{{ item.password }}'
    password_never_expires: no
    groups:
      - Test User
      - Application
    company: Ansible
    update_password: on_create
  loop:
    - name: Test User
      password: Password
    - name: Admin User
      password: SuperSecretPass01
    - name: Dev User
      password: '@fvr3IbFBujSRh!3hBg%wgFucD8^x8W5'
```

Running Commands

In cases where there is no appropriate module available for a task, a command or script can be run using the `win_shell`, `win_command`, `raw`, and `script` modules.

The `raw` module simply executes a Powershell command remotely. Since `raw` has none of the wrappers that Ansible typically uses, `become`, `async` and environment variables do not work.

The `script` module executes a script from the Ansible controller on one or more Windows hosts. Like `raw`, `script` currently does not support `become`, `async`, or environment variables.

The `win_command` module is used to execute a command which is either an executable or batch file, while the `win_shell` module is used to execute commands within a shell.

Choosing Command or Shell

The `win_shell` and `win_command` modules can both be used to execute a command or commands. The `win_shell` module is run within a shell-like process like `PowerShell` or `cmd`, so it has access to shell operators like `<`, `>`, `|`, `;`, `&&`, and `||`. Multi-lined commands can also be run in `win_shell`.

The `win_command` module simply runs a process outside of a shell. It can still run a shell command like `mkdir` or `New-Item` by passing the shell commands to a shell executable like `cmd.exe` or `PowerShell.exe`.

Here are some examples of using `win_command` and `win_shell`:

```
- name: Run a command under PowerShell
  win_shell: Get-Service -Name service | Stop-Service

- name: Run a command under cmd
  win_shell: mkdir C:\temp
  args:
    executable: cmd.exe

- name: Run a multiple shell commands
  win_shell: |
    New-Item -Path C:\temp -ItemType Directory
    Remove-Item -Path C:\temp -Force -Recurse
    $path_info = Get-Item -Path C:\temp
    $path_info.FullName

- name: Run an executable using win_command
  win_command: whoami.exe

- name: Run a cmd command
  win_command: cmd.exe /c mkdir C:\temp

- name: Run a vbs script
  win_command: cscript.exe script.vbs
```

注解: Some commands like `mkdir`, `del`, and `copy` only exist in the CMD shell. To run them with `win_command` they must be prefixed with `cmd.exe /c`.

Argument Rules

When running a command through `win_command`, the standard Windows argument rules apply:

- Each argument is delimited by a white space, which can either be a space or a tab.
- An argument can be surrounded by double quotes ". Anything inside these quotes is interpreted as a single argument even if it contains whitespace.
- A double quote preceded by a backslash \ is interpreted as just a double quote " and not as an argument delimiter.
- Backslashes are interpreted literally unless it immediately precedes double quotes; for example \ == \ and \" == "
- If an even number of backslashes is followed by a double quote, one backslash is used in the argument for every pair, and the double quote is used as a string delimiter for the argument.
- If an odd number of backslashes is followed by a double quote, one backslash is used in the argument for every pair, and the double quote is escaped and made a literal double quote in the argument.

With those rules in mind, here are some examples of quoting:

```
- win_command: C:\temp\executable.exe argument1 "argument 2" "C:\path\with space"
↪ "double \"quoted\""
```

```
argv[0] = C:\temp\executable.exe
argv[1] = argument1
argv[2] = argument 2
argv[3] = C:\path\with space
argv[4] = double "quoted"
```

```
- win_command: '"C:\Program Files\Program\program.exe" "escaped \\" backslash" unquoted-
↪ end-backslash\'
```

```
argv[0] = C:\Program Files\Program\program.exe
argv[1] = escaped \" backslash
argv[2] = unquoted-end-backslash\
```

```
# Due to YAML and Ansible parsing '\"' must be written as '{% raw %}\{ endraw %}'
- win_command: C:\temp\executable.exe C:\no\space\path "arg with end \ before end quote{
↪ % raw %}\{ endraw %}"
```

```
argv[0] = C:\temp\executable.exe
argv[1] = C:\no\space\path
argv[2] = arg with end \ before end quote\"
```

For more information, see [escaping arguments](#).

Creating and Running a Scheduled Task

WinRM has some restrictions in place that cause errors when running certain commands. One way to bypass these restrictions is to run a command through a scheduled task. A scheduled task is a Windows component that provides the ability to run an executable on a schedule and under a different account.

Ansible version 2.5 added modules that make it easier to work with scheduled tasks in Windows. The following is an example of running a script as a scheduled task that deletes itself after running:

```
- name: Create scheduled task to run a process
  win_scheduled_task:
    name: adhoc-task
    username: SYSTEM
    actions:
      - path: PowerShell.exe
        arguments: |
          Start-Sleep -Seconds 30 # This isn't required, just here as a demonstration
          New-Item -Path C:\temp\test -ItemType Directory
      # Remove this action if the task shouldn't be deleted on completion
      - path: cmd.exe
        arguments: /c schtasks.exe /Delete /TN "adhoc-task" /F
    triggers:
      - type: registration

- name: Wait for the scheduled task to complete
  win_scheduled_task_stat:
    name: adhoc-task
    register: task_stat
    until: (task_stat.state is defined and task_stat.state.status != "TASK_STATE_RUNNING")
    or (task_stat.task_exists == False)
    retries: 12
    delay: 10
```

注解: The modules used in the above example were updated/added in Ansible version 2.5.

Path Formatting for Windows

Windows differs from a traditional POSIX operating system in many ways. One of the major changes is the shift from / as the path separator to \. This can cause major issues with how playbooks are written, since \ is often used as an escape character on POSIX systems.

Ansible allows two different styles of syntax; each deals with path separators for Windows differently:

YAML Style

When using the YAML syntax for tasks, the rules are well-defined by the YAML standard:

- When using a normal string (without quotes), YAML will not consider the backslash an escape character.
- When using single quotes `'`, YAML will not consider the backslash an escape character.
- When using double quotes `"`, the backslash is considered an escape character and needs to be escaped with another backslash.

注解: You should only quote strings when it is absolutely necessary or required by YAML, and then use single quotes.

The YAML specification considers the following [escape sequences](#):

- `\0`, `\\`, `\"`, `_`, `\a`, `\b`, `\e`, `\f`, `\n`, `\r`, `\t`, `\v`, `\L`, `\N` and `\P` – Single character escape
- `<TAB>`, `<SPACE>`, `<NBSP>`, `<LNSP>`, `<PSP>` – Special characters
- `\x..` – 2-digit hex escape
- `\u....` – 4-digit hex escape
- `\U.....` – 8-digit hex escape

Here are some examples on how to write Windows paths:

```
# GOOD
tempdir: C:\Windows\Temp

# WORKS
tempdir: 'C:\Windows\Temp'
tempdir: "C:\\Windows\\Temp"

# BAD, BUT SOMETIMES WORKS
tempdir: C:\\Windows\\Temp
tempdir: 'C:\\Windows\\Temp'
tempdir: C:/Windows/Temp
```

This is an example which will fail:

```
# FAILS
tempdir: "C:\Windows\Temp"
```

This example shows the use of single quotes when they are required:

```
---
- name: Copy tomcat config
  win_copy:
    src: log4j.xml
    dest: '{{tc_home}}\lib\log4j.xml'
```

Legacy key=value Style

The legacy `key=value` syntax is used on the command line for ad-hoc commands, or inside playbooks. The use of this style is discouraged within playbooks because backslash characters need to be escaped, making playbooks harder to read. The legacy syntax depends on the specific implementation in Ansible, and quoting (both single and double) does not have any effect on how it is parsed by Ansible.

The Ansible `key=value` parser `parse_kv()` considers the following escape sequences:

- `\`, `'`, `"`, `\a`, `\b`, `\f`, `\n`, `\r`, `\t` and `\v` – Single character escape
- `\x..` – 2-digit hex escape
- `\u....` – 4-digit hex escape
- `\U.....` – 8-digit hex escape
- `\N{...}` – Unicode character by name

This means that the backslash is an escape character for some sequences, and it is usually safer to escape a backslash when in this form.

Here are some examples of using Windows paths with the `key=value` style:

```
# GOOD
tempdir=C:\\Windows\\Temp

# WORKS
tempdir='C:\\Windows\\Temp'
tempdir="C:\\Windows\\Temp"

# BAD, BUT SOMETIMES WORKS
tempdir=C:\Windows\Temp
tempdir='C:\Windows\Temp'
```

(下页继续)

(续上页)

```
tempdir="C:\Windows\Temp"
tempdir=C:/Windows/Temp

# FAILS
tempdir=C:\Windows\temp
tempdir='C:\Windows\temp'
tempdir="C:\Windows\temp"
```

The failing examples don't fail outright but will substitute `\t` with the `<TAB>` character resulting in `tempdir` being `C:\Windows<TAB>emp`.

Limitations

Some things you cannot do with Ansible and Windows are:

- Upgrade PowerShell
- Interact with the WinRM listeners

Because WinRM is reliant on the services being online and running during normal operations, you cannot upgrade PowerShell or interact with WinRM listeners with Ansible. Both of these actions will cause the connection to fail. This can technically be avoided by using `async` or a scheduled task, but those methods are fragile if the process it runs breaks the underlying connection Ansible uses, and are best left to the bootstrapping process or before an image is created.

Developing Windows Modules

Because Ansible modules for Windows are written in PowerShell, the development guides for Windows modules differ substantially from those for standard standard modules. Please see [Windows module development walkthrough](#) for more information.

参见:

[Intro to Playbooks](#) An introduction to playbooks

[Tips and tricks](#) Best practices advice

List of Windows Modules Windows specific module list, all implemented in PowerShell

User Mailing List Have a question? Stop by the google group!

[irc.freenode.net](#) #ansible IRC chat channel

Desired State Configuration

Topics

- *What is Desired State Configuration?*
- *Host Requirements*
- *Why Use DSC?*
- *How to Use DSC?*
 - *Property Types*
 - * *PSCredential*
 - * *CimInstance Type*
 - * *HashTable Type*
 - * *Arrays*
 - * *DateTime*
 - *Run As Another User*
- *Custom DSC Resources*
 - *Finding Custom DSC Resources*
 - *Installing a Custom Resource*
- *Examples*
 - *Extract a zip file*
 - *Create a directory*
 - *Interact with Azure*
 - *Setup IIS Website*

What is Desired State Configuration?

Desired State Configuration, or DSC, is a tool built into PowerShell that can be used to define a Windows host setup through code. The overall purpose of DSC is the same as Ansible, it is just executed in a different manner. Since Ansible 2.4, the `win_dsc` module has been added and can be used to leverage existing DSC resources when interacting with a Windows host.

More details on DSC can be viewed at [DSC Overview](#).

Host Requirements

To use the `win_dsc` module, a Windows host must have PowerShell v5.0 or newer installed. All supported hosts, except for Windows Server 2008 (non R2) can be upgraded to PowerShell v5.

Once the PowerShell requirements have been met, using DSC is as simple as creating a task with the `win_dsc` module.

Why Use DSC?

DSC and Ansible modules have a common goal which is to define and ensure the state of a resource. Because of this, resources like the DSC [File resource](#) and Ansible `win_file` can be used to achieve the same result. Deciding which to use depends on the scenario.

Reasons for using an Ansible module over a DSC resource:

- The host does not support PowerShell v5.0, or it cannot easily be upgraded
- The DSC resource does not offer a feature present in an Ansible module. For example `win_regedit` can manage the `REG_NONE` property type, while the DSC `Registry` resource cannot
- DSC resources have limited check mode support, while some Ansible modules have better checks
- DSC resources do not support diff mode, while some Ansible modules do
- Custom resources require further installation steps to be run on the host beforehand, while Ansible modules are built-in to Ansible
- There are bugs in a DSC resource where an Ansible module works

Reasons for using a DSC resource over an Ansible module:

- The Ansible module does not support a feature present in a DSC resource
- There is no Ansible module available
- There are bugs in an existing Ansible module

In the end, it doesn't matter whether the task is performed with DSC or an Ansible module; what matters is that the task is performed correctly and the playbooks are still readable. If you have more experience with DSC over Ansible and it does the job, just use DSC for that task.

How to Use DSC?

The `win_dsc` module takes in a free-form of options so that it changes according to the resource it is managing. A list of built in resources can be found at [resources](#).

Using the [Registry](#) resource as an example, this is the DSC definition as documented by Microsoft:

```

Registry [string] #ResourceName
{
    Key = [string]
    ValueName = [string]
    [ Ensure = [string] { Enable | Disable } ]
    [ Force = [bool] ]
    [ Hex = [bool] ]
    [ DependsOn = [string[]] ]
    [ ValueData = [string[]] ]
    [ ValueType = [string] { Binary | Dword | ExpandString | MultiString | Qword |
↳String } ]
}

```

When defining the task, `resource_name` must be set to the DSC resource being used - in this case the `resource_name` should be set to `Registry`. The `module_version` can refer to a specific version of the DSC resource installed; if left blank it will default to the latest version. The other options are parameters that are used to define the resource, such as `Key` and `ValueName`. While the options in the task are not case sensitive, keeping the case as-is is recommended because it makes it easier to distinguish DSC resource options from Ansible's `win_dsc` options.

This is what the Ansible task version of the above DSC Registry resource would look like:

```

- name: Use win_dsc module with the Registry DSC resource
  win_dsc:
    resource_name: Registry
    Ensure: Present
    Key: HKEY_LOCAL_MACHINE\SOFTWARE\ExampleKey
    ValueName: TestValue
    ValueData: TestData

```

Starting in Ansible 2.8, the `win_dsc` module automatically validates the input options from Ansible with the DSC definition. This means Ansible will fail if the option name is incorrect, a mandatory option is not set, or the value is not a valid choice. When running Ansible with a verbosity level of 3 or more (`-vvv`), the return value will contain the possible invocation options based on the `resource_name` specified. Here is an example of the invocation output for the above `Registry` task:

```

changed: [2016] => {
  "changed": true,
  "invocation": {
    "module_args": {
      "DependsOn": null,
      "Ensure": "Present",

```

(下页继续)

(续上页)

```

        "Force": null,
        "Hex": null,
        "Key": "HKEY_LOCAL_MACHINE\\SOFTWARE\\ExampleKey",
        "PsDscRunAsCredential_password": null,
        "PsDscRunAsCredential_username": null,
        "ValueData": [
            "TestData"
        ],
        "ValueName": "TestValue",
        "ValueType": null,
        "module_version": "latest",
        "resource_name": "Registry"
    }
},
"module_version": "1.1",
"reboot_required": false,
"verbose_set": [
    "Perform operation 'Invoke CimMethod' with following parameters, 'methodName' =
↪ResourceSet,'className' = MSFT_DSCLocalConfigurationManager,'namespaceName' = root/
↪Microsoft/Windows/DesiredStateConfiguration'.",
    "An LCM method call arrived from computer SERVER2016 with user sid S-1-5-21-
↪3088887838-4058132883-1884671576-1105.",
    "[SERVER2016]: LCM: [ Start Set      ] [[Registry]DirectResourceAccess]",
    "[SERVER2016]:                               [[Registry]DirectResourceAccess] (SET)
↪Create registry key 'HKLM:\\SOFTWARE\\ExampleKey'",
    "[SERVER2016]:                               [[Registry]DirectResourceAccess] (SET)
↪Set registry key value 'HKLM:\\SOFTWARE\\ExampleKey\\TestValue' to 'TestData' of type
↪'String'",
    "[SERVER2016]: LCM: [ End   Set      ] [[Registry]DirectResourceAccess] in 0.
↪1930 seconds.",
    "[SERVER2016]: LCM: [ End   Set      ]      in 0.2720 seconds.",
    "Operation 'Invoke CimMethod' complete.",
    "Time taken for configuration job to complete is 0.402 seconds"
],
"verbose_test": [
    "Perform operation 'Invoke CimMethod' with following parameters, 'methodName' =
↪ResourceTest,'className' = MSFT_DSCLocalConfigurationManager,'namespaceName' = root/
↪Microsoft/Windows/DesiredStateConfiguration'.",
    "An LCM method call arrived from computer SERVER2016 with user sid S-1-5-21-
↪3088887838-4058132883-1884671576-1105.",

```

(下页继续)

(续上页)

```

        "[SERVER2016]: LCM:  [ Start  Test      ]  [[Registry]DirectResourceAccess]",
        "[SERVER2016]:                               [[Registry]DirectResourceAccess]␣
↪Registry key 'HKLM:\\SOFTWARE\\ExampleKey' does not exist",
        "[SERVER2016]: LCM:  [ End    Test      ]  [[Registry]DirectResourceAccess] False␣
↪in 0.2510 seconds.",
        "[SERVER2016]: LCM:  [ End    Set       ]      in 0.3310 seconds.",
        "Operation 'Invoke CimMethod' complete.",
        "Time taken for configuration job to complete is 0.475 seconds"
    ]
}

```

The `invocation.module_args` key shows the actual values that were set as well as other possible values that were not set. Unfortunately this will not show the default value for a DSC property, only what was set from the Ansible task. Any `*_password` option will be masked in the output for security reasons, if there are any other sensitive module options, set `no_log: True` on the task to stop all task output from being logged.

Property Types

Each DSC resource property has a type that is associated with it. Ansible will try to convert the defined options to the correct type during execution. For simple types like `[string]` and `[bool]` this is a simple operation, but complex types like `[PSCredential]` or arrays (like `[string[]]`) this require certain rules.

PSCredential

A `[PSCredential]` object is used to store credentials in a secure way, but Ansible has no way to serialize this over JSON. To set a DSC `PSCredential` property, the definition of that parameter should have two entries that are suffixed with `_username` and `_password` for the username and password respectively. For example:

```

PsDscRunAsCredential_username: '{{ ansible_user }}'
PsDscRunAsCredential_password: '{{ ansible_password }}'

SourceCredential_username: AdminUser
SourceCredential_password: PasswordForAdminUser

```

注解: On versions of Ansible older than 2.8, you should set `no_log: yes` on the task definition in Ansible to ensure any credentials used are not stored in any log file or console output.

A [PSCredential] is defined with EmbeddedInstance("MSFT_Credential") in a DSC resource MOF definition.

CimInstance Type

A [CimInstance] object is used by DSC to store a dictionary object based on a custom class defined by that resource. Defining a value that takes in a [CimInstance] in YAML is the same as defining a dictionary in YAML. For example, to define a [CimInstance] value in Ansible:

```
# [CimInstance]AuthenticationInfo == MSFT_xWebAuthenticationInformation
AuthenticationInfo:
  Anonymous: no
  Basic: yes
  Digest: no
  Windows: yes
```

In the above example, the CIM instance is a representation of the class `MSFT_xWebAuthenticationInformation`. This class accepts four boolean variables, `Anonymous`, `Basic`, `Digest`, and `Windows`. The keys to use in a [CimInstance] depend on the class it represents. Please read through the documentation of the resource to determine the keys that can be used and the types of each key value. The class definition is typically located in the `<resource name>.schema.mof`.

HashTable Type

A [HashTable] object is also a dictionary but does not have a strict set of keys that can/need to be defined. Like a [CimInstance], define it like a normal dictionary value in YAML. A [HashTable] is defined with EmbeddedInstance("MSFT_KeyValuePair") in a DSC resource MOF definition.

Arrays

Simple type arrays like [string[]] or [UInt32[]] are defined as a list or as a comma separated string which are then cast to their type. Using a list is recommended because the values are not manually parsed by the `win_dsc` module before being passed to the DSC engine. For example, to define a simple type array in Ansible:

```
# [string[]]
ValueData: entry1, entry2, entry3
ValueData:
- entry1
- entry2
- entry3
```

(下页继续)

(续上页)

```
# [UInt32[]]
ReturnCode: 0,3010
ReturnCode:
- 0
- 3010
```

Complex type arrays like `[CimInstance[]]` (array of dicts), can be defined like this example:

```
# [CimInstance[]]BindingInfo == MSFT_xWebBindingInformation
BindingInfo:
- Protocol: https
  Port: 443
  CertificateStoreName: My
  CertificateThumbprint: C676A89018C4D5902353545343634F35E6B3A659
  HostName: DSCTest
  IPAddress: '*'
  SSLFlags: 1
- Protocol: http
  Port: 80
  IPAddress: '*'
```

The above example, is an array with two values of the class `MSFT_xWebBindingInformation`. When defining a `[CimInstance[]]`, be sure to read the resource documentation to find out what keys to use in the definition.

DateTime

A `[DateTime]` object is a `DateTime` string representing the date and time in the [ISO 8601](#) date time format. The value for a `[DateTime]` field should be quoted in YAML to ensure the string is properly serialized to the Windows host. Here is an example of how to define a `[DateTime]` value in Ansible:

```
# As UTC-0 (No timezone)
DateTime: '2019-02-22T13:57:31.2311892+00:00'

# As UTC+4
DateTime: '2019-02-22T17:57:31.2311892+04:00'

# As UTC-4
DateTime: '2019-02-22T09:57:31.2311892-04:00'
```

All the values above are equal to a UTC date time of February 22nd 2019 at 1:57pm with 31 seconds and

2311892 milliseconds.

Run As Another User

By default, DSC runs each resource as the SYSTEM account and not the account that Ansible use to run the module. This means that resources that are dynamically loaded based on a user profile, like the HKEY_CURRENT_USER registry hive, will be loaded under the SYSTEM profile. The parameter `PsDscRunAsCredential` is a parameter that can be set for every DSC resource force the DSC engine to run under a different account. As `PsDscRunAsCredential` has a type of `PSCredential`, it is defined with the `_username` and `_password` suffix.

Using the Registry resource type as an example, this is how to define a task to access the HKEY_CURRENT_USER hive of the Ansible user:

```
- name: Use win_dsc with PsDscRunAsCredential to run as a different user
  win_dsc:
    resource_name: Registry
    Ensure: Present
    Key: HKEY_CURRENT_USER\ExampleKey
    ValueName: TestValue
    ValueData: TestData
    PsDscRunAsCredential_username: '{{ ansible_user }}'
    PsDscRunAsCredential_password: '{{ ansible_password }}'
  no_log: yes
```

Custom DSC Resources

DSC resources are not limited to the built-in options from Microsoft. Custom modules can be installed to manage other resources that are not usually available.

Finding Custom DSC Resources

You can use the [PSGallery](#) to find custom resources, along with documentation on how to install them on a Windows host.

The `Find-DscResource` cmdlet can also be used to find custom resources. For example:

```
# Find all DSC resources in the configured repositories
Find-DscResource

# Find all DSC resources that relate to SQL
Find-DscResource -ModuleName "*sql*"
```

注解: DSC resources developed by Microsoft that start with **x**, means the resource is experimental and comes with no support.

Installing a Custom Resource

There are three ways that a DSC resource can be installed on a host:

- Manually with the `Install-Module` cmdlet
- Using the `win_psmodule` Ansible module
- Saving the module manually and copying it another host

This is an example of installing the `xWebAdministration` resources using `win_psmodule`:

```
- name: Install xWebAdministration DSC resource
  win_psmodule:
    name: xWebAdministration
    state: present
```

Once installed, the `win_dsc` module will be able to use the resource by referencing it with the `resource_name` option.

The first two methods above only work when the host has access to the internet. When a host does not have internet access, the module must first be installed using the methods above on another host with internet access and then copied across. To save a module to a local filepath, the following PowerShell cmdlet can be run:

```
Save-Module -Name xWebAdministration -Path C:\temp
```

This will create a folder called `xWebAdministration` in `C:\temp` which can be copied to any host. For PowerShell to see this offline resource, it must be copied to a directory set in the `PSModulePath` environment variable. In most cases the path `C:\Program Files\WindowsPowerShell\Module` is set through this variable, but the `win_path` module can be used to add different paths.

Examples

Extract a zip file

```
- name: Extract a zip file
  win_dsc:
    resource_name: Archive
```

(下页继续)

(续上页)

```
Destination: C:\temp\output
Path: C:\temp\zip.zip
Ensure: Present
```

Create a directory

```
- name: Create file with some text
win_dsc:
  resource_name: File
  DestinationPath: C:\temp\file
  Contents: |
    Hello
    World
  Ensure: Present
  Type: File

- name: Create directory that is hidden is set with the System attribute
win_dsc:
  resource_name: File
  DestinationPath: C:\temp\hidden-directory
  Attributes: Hidden,System
  Ensure: Present
  Type: Directory
```

Interact with Azure

```
- name: Install xAzure DSC resources
win_psmodule:
  name: xAzure
  state: present

- name: Create virtual machine in Azure
win_dsc:
  resource_name: xAzureVM
  ImageName: a699494373c04fc0bc8f2bb1389d6106__Windows-Server-2012-R2-201409.01-en.us-
↪127GB.vhd
  Name: DSCHOST01
  ServiceName: ServiceName
```

(下页继续)

(续上页)

```

StorageAccountName: StorageAccountName
InstanceSize: Medium
Windows: yes
Ensure: Present
Credential_username: '{{ ansible_user }}'
Credential_password: '{{ ansible_password }}'

```

Setup IIS Website

```

- name: Install xWebAdministration module
  win_psmodule:
    name: xWebAdministration
    state: present

- name: Install IIS features that are required
  win_dsc:
    resource_name: WindowsFeature
    Name: '{{ item }}'
    Ensure: Present
  loop:
    - Web-Server
    - Web-Asp-Net45

- name: Setup web content
  win_dsc:
    resource_name: File
    DestinationPath: C:\inetpub\IISSite\index.html
    Type: File
    Contents: |
      <html>
      <head><title>IIS Site</title></head>
      <body>This is the body</body>
      </html>
    Ensure: present

- name: Create new website
  win_dsc:
    resource_name: xWebsite
    Name: NewIISSite

```

(下页继续)

(续上页)

```
State: Started
PhysicalPath: C:\inetpub\IISSite\index.html
BindingInfo:
- Protocol: https
  Port: 8443
  CertificateStoreName: My
  CertificateThumbprint: C676A89018C4D5902353545343634F35E6B3A659
  HostName: DSCTest
  IPAddress: '*'
  SSLFlags: 1
- Protocol: http
  Port: 8080
  IPAddress: '*'
AuthenticationInfo:
  Anonymous: no
  Basic: yes
  Digest: no
  Windows: yes
```

参见:

Intro to Playbooks An introduction to playbooks

Tips and tricks Best practices advice

List of Windows Modules Windows specific module list, all implemented in PowerShell

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

Windows performance

This document offers some performance optimizations you might like to apply to your Windows hosts to speed them up specifically in the context of using Ansible with them, and generally.

Optimise PowerShell performance to reduce Ansible task overhead

To speed up the startup of PowerShell by around 10x, run the following PowerShell snippet in an Administrator session. Expect it to take tens of seconds.

注解: If native images have already been created by the ngen task or service, you will observe no difference

in performance (but this snippet will at that point execute faster than otherwise).

```
function Optimize-PowershellAssemblies {
    # NGEN powershell assembly, improves startup time of powershell by 10x
    $old_path = $env:path
    try {
        $env:path = [Runtime.InteropServices.RuntimeEnvironment]::GetRuntimeDirectory()
        [AppDomain]::CurrentDomain.GetAssemblies() | % {
            if (!$_.location) {continue}
            $Name = Split-Path $_.location -leaf
            if ($Name.StartsWith("Microsoft.PowerShell.")) {
                Write-Progress -Activity "Native Image Installation" -Status "$name"
                ngen install $_.location | % {"`t$_"}
            }
        }
    } finally {
        $env:path = $old_path
    }
}
Optimize-PowershellAssemblies
```

PowerShell is used by every Windows Ansible module. This optimisation reduces the time PowerShell takes to start up, removing that overhead from every invocation.

This snippet uses the native image generator, ngen to pre-emptively create native images for the assemblies that PowerShell relies on.

Fix high-CPU-on-boot for VMs/cloud instances

If you are creating golden images to spawn instances from, you can avoid a disruptive high CPU task near startup via processing the ngen queue within your golden image creation, if you know the CPU types won't change between golden image build process and runtime.

Place the following near the end of your playbook, bearing in mind the factors that can cause native images to be invalidated (see [MSDN](#)).

```
- name: generate native .NET images for CPU
  win_dotnet_ngen:
```

Windows Frequently Asked Questions

Here are some commonly asked questions in regards to Ansible and Windows and their answers.

注解: This document covers questions about managing Microsoft Windows servers with Ansible. For questions about Ansible Core, please see the *general FAQ page*.

Does Ansible work with Windows XP or Server 2003?

Ansible does not work with Windows XP or Server 2003 hosts. Ansible does work with these Windows operating system versions:

- Windows Server 2008 ¹
- Windows Server 2008 R2 ¹
- Windows Server 2012
- Windows Server 2012 R2
- Windows Server 2016
- Windows Server 2019
- Windows 7 ¹
- Windows 8.1
- Windows 10

1 - See the *Server 2008 FAQ* entry for more details.

Ansible also has minimum PowerShell version requirements - please see *Setting up a Windows Host* for the latest information.

Are Server 2008, 2008 R2 and Windows 7 supported?

Microsoft ended Extended Support for these versions of Windows on January 14th, 2020, and Ansible deprecated official support in the 2.10 release. No new feature development will occur targeting these operating systems, and automated testing has ceased. However, existing modules and features will likely continue to work, and simple pull requests to resolve issues with these Windows versions may be accepted.

Can I manage Windows Nano Server with Ansible?

Ansible does not currently work with Windows Nano Server, since it does not have access to the full .NET Framework that is used by the majority of the modules and internal components.

Can Ansible run on Windows?

No, Ansible can only manage Windows hosts. Ansible cannot run on a Windows host natively, though it can run under the Windows Subsystem for Linux (WSL).

注解: The Windows Subsystem for Linux is not supported by Ansible and should not be used for production systems.

To install Ansible on WSL, the following commands can be run in the bash terminal:

```
sudo apt-get update
sudo apt-get install python-pip git libffi-dev libssl-dev -y
pip install --user ansible pywinrm
```

To run Ansible from source instead of a release on the WSL, simply uninstall the pip installed version and then clone the git repo.

```
pip uninstall ansible -y
git clone https://github.com/ansible/ansible.git
source ansible/hacking/env-setup

# To enable Ansible on login, run the following
echo ". ~/.ansible/hacking/env-setup -q" >> ~/.bashrc
```

Can I use SSH keys to authenticate to Windows hosts?

You cannot use SSH keys with the WinRM or PSRP connection plugins. These connection plugins use X509 certificates for authentication instead of the SSH key pairs that SSH uses.

The way X509 certificates are generated and mapped to a user is different from the SSH implementation; consult the *Windows Remote Management* documentation for more information.

Ansible 2.8 has added an experimental option to use the SSH connection plugin, which uses SSH keys for authentication, for Windows servers. See *this question* for more information.

Why can I run a command locally that does not work under Ansible?

Ansible executes commands through WinRM. These processes are different from running a command locally in these ways:

- Unless using an authentication option like CredSSP or Kerberos with credential delegation, the WinRM process does not have the ability to delegate the user's credentials to a network resource, causing

Access is Denied errors.

- All processes run under WinRM are in a non-interactive session. Applications that require an interactive session will not work.
- When running through WinRM, Windows restricts access to internal Windows APIs like the Windows Update API and DPAPI, which some installers and programs rely on.

Some ways to bypass these restrictions are to:

- Use `become`, which runs a command as it would when run locally. This will bypass most WinRM restrictions, as Windows is unaware the process is running under WinRM when `become` is used. See the *Understanding privilege escalation: become* documentation for more information.
- Use a scheduled task, which can be created with `win_scheduled_task`. Like `become`, it will bypass all WinRM restrictions, but it can only be used to run commands, not modules.
- Use `win_psexec` to run a command on the host. PSEXec does not use WinRM and so will bypass any of the restrictions.
- To access network resources without any of these workarounds, you can use CredSSP or Kerberos with credential delegation enabled.

See *Understanding privilege escalation: become* more info on how to use `become`. The limitations section at *Windows Remote Management* has more details around WinRM limitations.

This program won't install on Windows with Ansible

See *this question* for more information about WinRM limitations.

What Windows modules are available?

Most of the Ansible modules in Ansible Core are written for a combination of Linux/Unix machines and arbitrary web services. These modules are written in Python and most of them do not work on Windows.

Because of this, there are dedicated Windows modules that are written in PowerShell and are meant to be run on Windows hosts. A list of these modules can be found [here](#).

In addition, the following Ansible Core modules/action-plugins work with Windows:

- `add_host`
- `assert`
- `async_status`
- `debug`
- `fail`
- `fetch`

- `group_by`
- `include`
- `include_role`
- `include_vars`
- `meta`
- `pause`
- `raw`
- `script`
- `set_fact`
- `set_stats`
- `setup`
- `slurp`
- `template` (also: `win_template`)
- `wait_for_connection`

Can I run Python modules on Windows hosts?

No, the WinRM connection protocol is set to use PowerShell modules, so Python modules will not work. A way to bypass this issue to use `delegate_to: localhost` to run a Python module on the Ansible controller. This is useful if during a playbook, an external service needs to be contacted and there is no equivalent Windows module available.

Can I connect to Windows hosts over SSH?

Ansible 2.8 has added an experimental option to use the SSH connection plugin to manage Windows hosts. To connect to Windows hosts over SSH, you must install and configure the [Win32-OpenSSH](#) fork that is in development with Microsoft on the Windows host(s). While most of the basics should work with SSH, Win32-OpenSSH is rapidly changing, with new features added and bugs fixed in every release. It is highly recommend you [install](#) the latest release of Win32-OpenSSH from the [GitHub Releases](#) page when using it with Ansible on Windows hosts.

To use SSH as the connection to a Windows host, set the following variables in the inventory:

```
ansible_connection=ssh

# Set either cmd or powershell not both
```

(下页继续)

(续上页)

```
ansible_shell_type=cmd
# ansible_shell_type=powershell
```

The value for `ansible_shell_type` should either be `cmd` or `powershell`. Use `cmd` if the `DefaultShell` has not been configured on the SSH service and `powershell` if that has been set as the `DefaultShell`.

Why is connecting to a Windows host via SSH failing?

Unless you are using `Win32-OpenSSH` as described above, you must connect to Windows hosts using *Windows Remote Management*. If your Ansible output indicates that SSH was used, either you did not set the connection vars properly or the host is not inheriting them correctly.

Make sure `ansible_connection: winrm` is set in the inventory for the Windows host(s).

Why are my credentials being rejected?

This can be due to a myriad of reasons unrelated to incorrect credentials.

See HTTP 401/Credentials Rejected at *Setting up a Windows Host* for a more detailed guide of this could mean.

Why am I getting an error SSL CERTIFICATE_VERIFY_FAILED?

When the Ansible controller is running on Python 2.7.9+ or an older version of Python that has backported `SSLContext` (like Python 2.7.5 on RHEL 7), the controller will attempt to validate the certificate WinRM is using for an HTTPS connection. If the certificate cannot be validated (such as in the case of a self signed cert), it will fail the verification process.

To ignore certificate validation, add `ansible_winrm_server_cert_validation: ignore` to inventory for the Windows host.

参见:

Windows Guides The Windows documentation index

Intro to Playbooks An introduction to playbooks

Tips and tricks Best practices advice

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

1.3.16 Using collections

Collections are a distribution format for Ansible content that can include playbooks, roles, modules, and plugins. You can install and use collections through [Ansible Galaxy](#).

- For details on how to *develop* collections see *Developing collections*.
- For the current development status of Collections and FAQ see [Ansible Collections Community Guide](#).

- *Installing collections*
 - *Installing collections with `ansible-galaxy`*
 - *Installing an older version of a collection*
 - *Install multiple collections with a requirements file*
 - *Downloading a collection for offline use*
 - *Configuring the `ansible-galaxy` client*
- *Listing collections*
- *Verifying collections*
 - *Verifying collections with `ansible-galaxy`*
- *Using collections in a Playbook*
- *Simplifying module names with the `collections` keyword*
 - *Using `collections` in roles*
 - *Using `collections` in playbooks*

Installing collections

Installing collections with `ansible-galaxy`

By default, `ansible-galaxy collection install` uses <https://galaxy.ansible.com> as the Galaxy server (as listed in the `ansible.cfg` file under `galaxy_server`). You do not need any further configuration.

See *Configuring the `ansible-galaxy` client* if you are using any other Galaxy server, such as Red Hat Automation Hub.

To install a collection hosted in Galaxy:

```
ansible-galaxy collection install my_namespace.my_collection
```

You can also directly use the tarball from your build:

```
ansible-galaxy collection install my_namespace-my_collection-1.0.0.tar.gz -p ./
↳ collections
```

注解: The install command automatically appends the path `ansible_collections` to the one specified with the `-p` option unless the parent directory is already in a folder called `ansible_collections`.

When using the `-p` option to specify the install path, use one of the values configured in `COLLECTIONS_PATHS`, as this is where Ansible itself will expect to find collections. If you don't specify a path, `ansible-galaxy collection install` installs the collection to the first path defined in `COLLECTIONS_PATHS`, which by default is `~/.ansible/collections`

You can also keep a collection adjacent to the current playbook, under a `collections/ansible_collections/` directory structure.

```
./
  play.yml
  collections/
    ansible_collections/
      my_namespace/
        my_collection/<collection structure lives here>
```

See *Collection structure* for details on the collection directory structure.

Installing an older version of a collection

You can only have one version of a collection installed at a time. By default `ansible-galaxy` installs the latest available version. If you want to install a specific version, you can add a version range identifier. For example, to install the 1.0.0-beta.1 version of the collection:

```
ansible-galaxy collection install my_namespace.my_collection:==1.0.0-beta.1
```

You can specify multiple range identifiers separated by `,`. Use single quotes so the shell passes the entire command, including `>`, `!`, and other operators, along. For example, to install the most recent version that is greater than or equal to 1.0.0 and less than 2.0.0:

```
ansible-galaxy collection install 'my_namespace.my_collection:>=1.0.0,<2.0.0'
```

Ansible will always install the most recent version that meets the range identifiers you specify. You can use the following range identifiers:

- `*`: The most recent version. This is the default.
- `!=`: Not equal to the version specified.

- ==: Exactly the version specified.
- >=: Greater than or equal to the version specified.
- >: Greater than the version specified.
- <=: Less than or equal to the version specified.
- <: Less than the version specified.

注解: By default `ansible-galaxy` ignores pre-release versions. To install a pre-release version, you must use the `==` range identifier to require it explicitly.

Install multiple collections with a requirements file

You can also setup a `requirements.yml` file to install multiple collections in one command. This file is a YAML file in the format:

```
---
collections:
  # With just the collection name
  - my_namespace.my_collection

  # With the collection name, version, and source options
  - name: my_namespace.my_other_collection
    version: 'version range identifiers (default: `*`)'
    source: 'The Galaxy URL to pull the collection from (default: `--api-server` from
↪cmdline)'
```

The `version` key can take in the same range identifier format documented above.

Roles can also be specified and placed under the `roles` key. The values follow the same format as a requirements file used in older Ansible releases.

```
---
roles:
  # Install a role from Ansible Galaxy.
  - name: geerlingguy.java
    version: 1.9.6

collections:
  # Install a collection from Ansible Galaxy.
  - name: geerlingguy.php_roles
```

(下页继续)

```
version: 0.9.3
source: https://galaxy.ansible.com
```

注解: While both roles and collections can be specified in one requirements file, they need to be installed separately. The `ansible-galaxy role install -r requirements.yml` will only install roles and `ansible-galaxy collection install -r requirements.yml -p ./` will only install collections.

Downloading a collection for offline use

To download the collection tarball from Galaxy for offline use:

1. Navigate to the collection page.
2. Click on *Download tarball*.

You may also need to manually download any dependent collections.

Configuring the ansible-galaxy client

By default, `ansible-galaxy` uses <https://galaxy.ansible.com> as the Galaxy server (as listed in the `ansible.cfg` file under `galaxy_server`).

You can use either option below to configure `ansible-galaxy` collection to use other servers (such as Red Hat Automation Hub or a custom Galaxy server):

- Set the `server_list` in the `galaxy_server_list` configuration option in `ansible_configuration_settings_locations`.
- Use the `--server` command line argument to limit to an individual server.

To configure a Galaxy server list in `ansible.cfg`:

1. Add the `server_list` option under the `[galaxy]` section to one or more server names.
2. Create a new section for each server name.
3. Set the `url` option for each server name.
4. Optionally, set the API token for each server name. See *API token* for details.

注解: The `url` option for each server name must end with a forward slash `/`. If you do not set the API token in your Galaxy server list, use the `--api-key` argument to pass in the token to the `ansible-galaxy collection publish` command.

For Automation Hub, you additionally need to:

1. Set the `auth_url` option for each server name.
2. Set the API token for each server name. Go to <https://cloud.redhat.com/ansible/automation-hub/token/> and click *:Get API token* from the version dropdown to copy your API token.

The following example shows how to configure multiple servers:

```
[galaxy]
server_list = automation_hub, my_org_hub, release_galaxy, test_galaxy

[galaxy_server.automation_hub]
url=https://cloud.redhat.com/api/automation-hub/
auth_url=https://sso.redhat.com/auth/realms/redhat-external/protocol/openid-connect/token
token=my_ah_token

[galaxy_server.my_org_hub]
url=https://automation.my_org/
username=my_user
password=my_pass

[galaxy_server.release_galaxy]
url=https://galaxy.ansible.com/
token=my_token

[galaxy_server.test_galaxy]
url=https://galaxy-dev.ansible.com/
token=my_test_token
```

注解: You can use the `--server` command line argument to select an explicit Galaxy server in the `server_list` and the value of this argument should match the name of the server. To use a server not in the server list, set the value to the URL to access that server (all servers in the server list will be ignored). Also you cannot use the `--api-key` argument for any of the predefined servers. You can only use the `api_key` argument if you did not define a server list or if you specify a URL in the `--server` argument.

Galaxy server list configuration options

The `galaxy_server_list` option is a list of server identifiers in a prioritized order. When searching for a collection, the install process will search in that order, for example, `automation_hub` first, then `my_org_hub`, `release_galaxy`, and finally `test_galaxy` until the collection is found. The actual Galaxy instance is then defined under the section `[galaxy_server.{{ id }}]` where `{{ id }}` is the server identifier defined in the list. This section can then define the following keys:

- **url**: The URL of the Galaxy instance to connect to. Required.
- **token**: An API token key to use for authentication against the Galaxy instance. Mutually exclusive with **username**.
- **username**: The username to use for basic authentication against the Galaxy instance. Mutually exclusive with **token**.
- **password**: The password to use, in conjunction with **username**, for basic authentication.
- **auth_url**: The URL of a Keycloak server 'token_endpoint' if using SSO authentication (for example, Automation Hub). Mutually exclusive with **username**. Requires **token**.

As well as defining these server options in the `ansible.cfg` file, you can also define them as environment variables. The environment variable is in the form `ANSIBLE_GALAXY_SERVER_{{ id }}_{{ key }}` where `{{ id }}` is the upper case form of the server identifier and `{{ key }}` is the key to define. For example I can define `token` for `release_galaxy` by setting `ANSIBLE_GALAXY_SERVER_RELEASE_GALAXY_TOKEN=secret_token`.

For operations that use only one Galaxy server (for example, the `publish`, `info`, or `install` commands). the `ansible-galaxy` collection command uses the first entry in the `server_list`, unless you pass in an explicit server with the `--server` argument.

注解: Once a collection is found, any of its requirements are only searched within the same Galaxy instance as the parent collection. The install process will not search for a collection requirement in a different Galaxy instance.

Listing collections

To list installed collections, run `ansible-galaxy collection list`. This shows all of the installed collections found in the configured collections search paths. The path where the collections are located are displayed as well as version information. If no version information is available, a `*` is displayed for the version number.

```
# /home/astark/.ansible/collections/ansible_collections
Collection          Version
-----
cisco.aci            0.0.5
cisco.mso            0.0.4
sandwiches.ham       *
splunk.enterprise_security 0.0.5

# /usr/share/ansible/collections/ansible_collections
Collection          Version
```

(下页继续)

(续上页)

```

-----
fortinet.fortios  1.0.6
pureport.pureport 0.0.8
sensu.sensu_go    1.3.0

```

Run with `-vvv` to display more detailed information.

To list a specific collection, pass a valid fully qualified collection name (FQCN) to the command `ansible-galaxy collection list`. All instances of the collection will be listed.

```

> ansible-galaxy collection list fortinet.fortios

# /home/astark/.ansible/collections/ansible_collections
Collection      Version
-----
fortinet.fortios 1.0.1

# /usr/share/ansible/collections/ansible_collections
Collection      Version
-----
fortinet.fortios 1.0.6

```

To search other paths for collections, use the `-p` option. Specify multiple search paths by separating them with a `:`. The list of paths specified on the command line will be added to the beginning of the configured collections search paths.

```

> ansible-galaxy collection list -p '/opt/ansible/collections:/etc/ansible/collections'

# /opt/ansible/collections/ansible_collections
Collection      Version
-----
sandwiches.club 1.7.2

# /etc/ansible/collections/ansible_collections
Collection      Version
-----
sandwiches.pbj  1.2.0

# /home/astark/.ansible/collections/ansible_collections
Collection      Version
-----

```

(下页继续)

(续上页)

```

cisco.aci          0.0.5
cisco.mso          0.0.4
fortinet.fortios   1.0.1
sandwiches.ham     *
splunk.enterprise_security 0.0.5

# /usr/share/ansible/collections/ansible_collections
Collection          Version
-----
fortinet.fortios    1.0.6
pureport.pureport   0.0.8
sensu.sensu_go      1.3.0

```

Verifying collections

Verifying collections with ansible-galaxy

Once installed, you can verify that the content of the installed collection matches the content of the collection on the server. This feature expects that the collection is installed in one of the configured collection paths and that the collection exists on one of the configured galaxy servers.

```
ansible-galaxy collection verify my_namespace.my_collection
```

The output of the `ansible-galaxy collection verify` command is quiet if it is successful. If a collection has been modified, the altered files are listed under the collection name.

```

ansible-galaxy collection verify my_namespace.my_collection
Collection my_namespace.my_collection contains modified content in the following files:
my_namespace.my_collection
    plugins/inventory/my_inventory.py
    plugins/modules/my_module.py

```

You can use the `-vvv` flag to display additional information, such as the version and path of the installed collection, the URL of the remote collection used for validation, and successful verification output.

```

ansible-galaxy collection verify my_namespace.my_collection -vvv
...
Verifying 'my_namespace.my_collection:1.0.0'.
Installed collection found at '/path/to/ansible_collections/my_namespace/my_collection/'
Remote collection found at 'https://galaxy.ansible.com/download/my_namespace-my_
↳collection-1.0.0.tar.gz'

```

(下页继续)

(续上页)

```
Successfully verified that checksums for 'my_namespace.my_collection:1.0.0' match the
↳remote collection
```

If you have a pre-release or non-latest version of a collection installed you should include the specific version to verify. If the version is omitted, the installed collection is verified against the latest version available on the server.

```
ansible-galaxy collection verify my_namespace.my_collection:1.0.0
```

In addition to the `namespace.collection_name:version` format, you can provide the collections to verify in a `requirements.yml` file. Dependencies listed in `requirements.yml` are not included in the verify process and should be verified separately.

```
ansible-galaxy collection verify -r requirements.yml
```

Verifying against `tar.gz` files is not supported. If your `requirements.yml` contains paths to tar files or URLs for installation, you can use the `--ignore-errors` flag to ensure that all collections using the `namespace.name` format in the file are processed.

Using collections in a Playbook

Once installed, you can reference a collection content by its fully qualified collection name (FQCN):

```
- hosts: all
  tasks:
    - my_namespace.my_collection.mymodule:
      option1: value
```

This works for roles or any type of plugin distributed within the collection:

```
- hosts: all
  tasks:
    - import_role:
      name: my_namespace.my_collection.role1

    - my_namespace.mycollection.mymodule:
      option1: value

    - debug:
      msg: '{{ lookup("my_namespace.my_collection.lookup1", 'param1') | my_namespace.my_
↳collection.filter1 }}'
```

Simplifying module names with the `collections` keyword

The `collections` keyword lets you define a list of collections that your role or playbook should search for unqualified module and action names. So you can use the `collections` keyword, then simply refer to modules and action plugins by their short-form names throughout that role or playbook.

警告: If your playbook uses both the `collections` keyword and one or more roles, the roles do not inherit the collections set by the playbook. See below for details.

Using collections in roles

Within a role, you can control which collections Ansible searches for the tasks inside the role using the `collections` keyword in the role's `meta/main.yml`. Ansible will use the collections list defined inside the role even if the playbook that calls the role defines different collections in a separate `collections` keyword entry. Roles defined inside a collection always implicitly search their own collection first, so you don't need to use the `collections` keyword to access modules, actions, or other roles contained in the same collection.

```
# myrole/meta/main.yml
collections:
  - my_namespace.first_collection
  - my_namespace.second_collection
  - other_namespace.other_collection
```

Using collections in playbooks

In a playbook, you can control the collections Ansible searches for modules and action plugins to execute. However, any roles you call in your playbook define their own collections search order; they do not inherit the calling playbook's settings. This is true even if the role does not define its own `collections` keyword.

```
- hosts: all
  collections:
    - my_namespace.my_collection
  tasks:
    - import_role:
        name: role1

    - mymodule:
        option1: value
```

(下页继续)

(续上页)

```
- debug:
    msg: '{{ lookup("my_namespace.my_collection.lookup1", 'param1')| my_namespace.my_
↪collection.filter1 }}'
```

The `collections` keyword merely creates an ordered ‘search path’ for non-namespaced plugin and role references. It does not install content or otherwise change Ansible’s behavior around the loading of plugins or roles. Note that an FQCN is still required for non-action or module plugins (e.g., lookups, filters, tests).

参见:

[*Developing collections*](#) Develop or modify a collection.

[`collections__galaxy__meta`](#) Understand the collections metadata structure.

[Mailing List](#) The development mailing list

[irc.freenode.net](#) #ansible IRC chat channel

1.4 Ansible Community Guide

Welcome to the Ansible Community Guide!

The purpose of this guide is to teach you everything you need to know about being a contributing member of the Ansible community. All types of contributions are welcome, and necessary to Ansible’s continued success.

This page outlines the most common situations and questions that bring readers to this section. If you prefer a traditional table of contents, there’s one at the bottom of the page.

1.4.1 Getting started

- I’m new to the community. Where can I find the Ansible *Community Code of Conduct*?
- I’d like to know what I’m agreeing to when I contribute to Ansible. Does Ansible have a *Contributors License Agreement*?
- I’d like to contribute but I’m not sure how. Are there *easy ways to contribute*?
- I want to talk to other Ansible users. How do I find an *Ansible Meetup* near me?
- I have a question. Which *Ansible email lists and IRC channels* will help me find answers?
- I want to learn more about Ansible. What can I do?
 - [Read books](#).
 - [Get certified](#).
 - [Attend events](#).

- Review getting started guides.
- Watch videos - includes Ansible Automates, AnsibleFest & webinar recordings.
- I'd like updates about new Ansible versions. How are new releases announced?
- I want to use the current release. How do I know which *releases are current*?

1.4.2 Going deeper

- I think Ansible is broken. How do I *report a bug*?
- I need functionality that Ansible doesn't offer. How do I *request a feature*?
- I'm waiting for a particular feature. How do I see what's *planned for future Ansible Releases*?
- I have a specific Ansible interest or expertise (for example, VMware, Linode, and so on.). How do I get involved in a *working group*?
- I'd like to participate in conversations about features and fixes. How do I review GitHub issues and pull requests?
- I found a typo or another problem on docs.ansible.com. How can I *improve the documentation*?

1.4.3 Working with the Ansible repo

- I want to code my first changes to Ansible. How do I *set up my Python development environment*?
- I'd like to get more efficient as a developer. How can I find *editors, linters, and other tools* that will support my Ansible development efforts?
- I want my PR to meet Ansible's guidelines. Where can I find guidance on *coding in Ansible*?
- I want to learn more about Ansible roadmaps, releases, and projects. How do I find information on *the development cycle*?
- I'd like to connect Ansible to a new API or other resource. How do I *contribute a group of related modules*?
- My pull request is marked `needs_rebase`. How do I *rebase my PR*?
- I'm using an older version of Ansible and want a bug fixed in my version that's already been fixed on the `devel` branch. How do I *backport a bugfix PR*?
- I have an open pull request with a failing test. How do I learn about Ansible's *testing (CI) process*?
- I'm ready to step up as a module maintainer. What are the *guidelines for maintainers*?
- A module I maintain is obsolete. How do I *deprecate a module*?

1.4.4 Traditional Table of Contents

If you prefer to read the entire Community Guide, here' s a list of the pages in order:

Community Code of Conduct

Topics

- *Community Code of Conduct*
 - *Anti-harassment policy*
 - *Policy violations*

Every community can be strengthened by a diverse variety of viewpoints, insights, opinions, skillsets, and skill levels. However, with diversity comes the potential for disagreement and miscommunication. The purpose of this Code of Conduct is to ensure that disagreements and differences of opinion are conducted respectfully and on their own merits, without personal attacks or other behavior that might create an unsafe or unwelcoming environment.

These policies are not designed to be a comprehensive set of Things You Cannot Do. We ask that you treat your fellow community members with respect and courtesy, and in general, Don' t Be A Jerk. This Code of Conduct is meant to be followed in spirit as much as in letter and is not exhaustive.

All Ansible events and participants therein are governed by this Code of Conduct and anti-harassment policy. We expect organizers to enforce these guidelines throughout all events, and we expect attendees, speakers, sponsors, and volunteers to help ensure a safe environment for our whole community. Specifically, this Code of Conduct covers participation in all Ansible-related forums and mailing lists, code and documentation contributions, public IRC channels, private correspondence, and public meetings.

Ansible community members are...

Considerate

Contributions of every kind have far-ranging consequences. Just as your work depends on the work of others, decisions you make surrounding your contributions to the Ansible community will affect your fellow community members. You are strongly encouraged to take those consequences into account while making decisions.

Patient

Asynchronous communication can come with its own frustrations, even in the most responsive of communities. Please remember that our community is largely built on volunteered time, and that questions, contributions, and requests for support may take some time to receive a response. Repeated “bumps” or “reminders” in rapid succession are not good displays of patience. Additionally, it is considered poor manners

to ping a specific person with general questions. Pose your question to the community as a whole, and wait patiently for a response.

Respectful

Every community inevitably has disagreements, but remember that it is possible to disagree respectfully and courteously. Disagreements are never an excuse for rudeness, hostility, threatening behavior, abuse (verbal or physical), or personal attacks.

Kind

Everyone should feel welcome in the Ansible community, regardless of their background. Please be courteous, respectful and polite to fellow community members. Do not make or post offensive comments related to skill level, gender, gender identity or expression, sexual orientation, disability, physical appearance, body size, race, or religion. Sexualized images or imagery, real or implied violence, intimidation, oppression, stalking, sustained disruption of activities, publishing the personal information of others without explicit permission to do so, unwanted physical contact, and unwelcome sexual attention are all strictly prohibited. Additionally, you are encouraged not to make assumptions about the background or identity of your fellow community members.

Inquisitive

The only stupid question is the one that does not get asked. We encourage our users to ask early and ask often. Rather than asking whether you can ask a question (the answer is always yes!), instead, simply ask your question. You are encouraged to provide as many specifics as possible. Code snippets in the form of Gists or other paste site links are almost always needed in order to get the most helpful answers. Refrain from pasting multiple lines of code directly into the IRC channels - instead use gist.github.com or another paste site to provide code snippets.

Helpful

The Ansible community is committed to being a welcoming environment for all users, regardless of skill level. We were all beginners once upon a time, and our community cannot grow without an environment where new users feel safe and comfortable asking questions. It can become frustrating to answer the same questions repeatedly; however, community members are expected to remain courteous and helpful to all users equally, regardless of skill or knowledge level. Avoid providing responses that prioritize snideness and snark over useful information. At the same time, everyone is expected to read the provided documentation thoroughly. We are happy to answer questions, provide strategic guidance, and suggest effective workflows, but we are not here to do your job for you.

Anti-harassment policy

Harassment includes (but is not limited to) all of the following behaviors:

- Offensive comments related to gender (including gender expression and identity), age, sexual orientation, disability, physical appearance, body size, race, and religion

- Derogatory terminology including words commonly known to be slurs
- Posting sexualized images or imagery in public spaces
- Deliberate intimidation
- Stalking
- Posting others' personal information without explicit permission
- Sustained disruption of talks or other events
- Inappropriate physical contact
- Unwelcome sexual attention

Participants asked to stop any harassing behavior are expected to comply immediately. Sponsors are also subject to the anti-harassment policy. In particular, sponsors should not use sexualized images, activities, or other material. Meetup organizing staff and other volunteer organizers should not use sexualized attire or otherwise create a sexualized environment at community events.

In addition to the behaviors outlined above, continuing to behave a certain way after you have been asked to stop also constitutes harassment, even if that behavior is not specifically outlined in this policy. It is considerate and respectful to stop doing something after you have been asked to stop, and all community members are expected to comply with such requests immediately.

Policy violations

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting codeof-conduct@ansible.com, to any channel operator in the community IRC channels, or to the local organizers of an event. Meetup organizers are encouraged to prominently display points of contact for reporting unacceptable behavior at local events.

If a participant engages in harassing behavior, the meetup organizers may take any action they deem appropriate. These actions may include but are not limited to warning the offender, expelling the offender from the event, and barring the offender from future community events.

Organizers will be happy to help participants contact security or local law enforcement, provide escorts to an alternate location, or otherwise assist those experiencing harassment to feel safe for the duration of the meetup. We value the safety and well-being of our community members and want everyone to feel welcome at our events, both online and offline.

We expect all participants, organizers, speakers, and attendees to follow these policies at all of our event venues and event-related social events.

The Ansible Community Code of Conduct is licensed under the Creative Commons Attribution-Share Alike 3.0 license. Our Code of Conduct was adapted from Codes of Conduct of other open source projects, including:

- Contributor Covenant

- Elastic
- The Fedora Project
- OpenStack
- Puppet Labs
- Ubuntu

How can I help?

Topics

- *How can I help?*
 - *Become a power user*
 - *Ask and answer questions online*
 - *Review, fix, and maintain the documentation*
 - *Participate in your local meetup*
 - *File and verify issues*
 - *Review and submit pull requests*
 - *Become a module maintainer*
 - *Join a working group*
 - *Teach Ansible to others*
 - *Social media*

Thanks for being interested in helping the Ansible project!

There are many ways to help the Ansible project…but first, please read and understand the *Community Code of Conduct*.

Become a power user

A great way to help the Ansible project is to become a power user:

- Use Ansible everywhere you can
- Take tutorials and classes
- Read the *official documentation*
- Study some of the [many excellent books](#) about Ansible

- [Get certified](#).

When you become a power user, your ability and opportunities to help the Ansible project in other ways will multiply quickly.

Ask and answer questions online

There are many forums online where Ansible users ask and answer questions. Reach out and communicate with your fellow Ansible users.

You can find the official *[Ansible communication channels](#)*.

Review, fix, and maintain the documentation

Typos are everywhere, even in the Ansible documentation. We work hard to keep the documentation up-to-date, but you may also find outdated examples. We offer easy ways to *[report and/or fix documentation errors](#)*.

Participate in your local meetup

There are Ansible meetups [all over the world](#). Join your local meetup. Attend regularly. Ask good questions. Volunteer to give a presentation about how you use Ansible.

If there isn't a meetup near you, we'll be happy to help you [start one](#).

File and verify issues

All software has bugs, and Ansible is no exception. When you find a bug, you can help tremendously by *[telling us about it](#)*.

If you should discover that the bug you're trying to file already exists in an issue, you can help by verifying the behavior of the reported bug with a comment in that issue, or by reporting any additional information.

Review and submit pull requests

As you become more familiar with how Ansible works, you may be able to fix issues or develop new features yourself. If you think you've got a solution to a bug you've found in Ansible, or if you've got a new feature that you've written and would like to share with millions of Ansible users, read all about the *[Ansible development process](#)* to learn how to get your code accepted into Ansible.

Another good way to help is to review pull requests that other Ansible users have submitted. The Ansible community keeps a full list of [open pull requests by file](#), so if there's a particular module or plug-in that

particularly interests you, you can easily keep track of all the relevant new pull requests and provide testing or feedback.

Become a module maintainer

Once you've learned about the development process and have contributed code to a particular module, we encourage you to become a maintainer of that module. There are hundreds of different modules in Ansible, and the vast majority of them are written and maintained entirely by members of the Ansible community.

To learn more about the responsibilities of being an Ansible module maintainer, please read our [module maintainer guidelines](#).

Join a working group

Working groups are a way for Ansible community members to self-organize around particular topics of interest. We have working groups around various topics. To join or create a working group, please read the [Ansible Working Groups](#).

Teach Ansible to others

We're working on a standardized Ansible workshop called [Lightbulb](#) that can provide a good hands-on introduction to Ansible usage and concepts.

Social media

If you like Ansible and just want to spread the good word, feel free to share on your social media platform of choice, and let us know by using [@ansible](#) or [#ansible](#). We'll be looking for you.

Reporting Bugs And Requesting Features

Topics

- [Reporting Bugs And Requesting Features](#)
 - [Reporting a bug](#)
 - [Requesting a feature](#)

Reporting a bug

Ansible practices responsible disclosure - if this is a security-related bug, email security@ansible.com instead of filing a ticket or posting to any public groups, and you will receive a prompt response.

Ansible bugs should be reported to github.com/ansible/ansible/issues after signing up for a free GitHub account. Before reporting a bug, please use the bug/issue search to see if the issue has already been reported.

Knowing your Ansible version and the exact commands you are running, and what you expect, saves time and helps us help everyone with their issues more quickly. For that reason, we provide an issue template; please fill it out as completely and as accurately as possible.

Do not use the issue tracker for “how do I do this” type questions. These are great candidates for IRC or the mailing list instead where things are likely to be more of a discussion.

To be respectful of reviewers’ time and allow us to help everyone efficiently, please provide minimal well-reduced and well-commented examples rather than sharing your entire production playbook. Include playbook snippets and output where possible.

When sharing YAML in playbooks, formatting can be preserved by using [code blocks](#).

For multiple-file content, we encourage use of gist.github.com. Online pastebin content can expire, so it’s nice to have things around for a longer term if they are referenced in a ticket.

If you are not sure if something is a bug yet, you are welcome to ask about something on the [mailing list or IRC first](#).

As we are a very high volume project, if you determine that you do have a bug, please be sure to open the issue yourself to ensure we have a record of it. Don’t rely on someone else in the community to file the bug report for you.

Requesting a feature

The best way to get a feature into Ansible is to [submit a pull request](#).

Contributing to the Ansible Documentation

Ansible has a lot of documentation and a small team of writers. Community support helps us keep up with new features, fixes, and changes.

Improving the documentation is an easy way to make your first contribution to the Ansible project. You don’t have to be a programmer, since our documentation is written in YAML (module documentation) or [reStructuredText](#) (rST). If you’re using Ansible, you already use YAML in your playbooks. And rST is mostly just text. You don’t even need git experience, if you use the [Edit on GitHub](#) option.

If you find a typo, a broken example, a missing topic, or any other error or omission on this documentation website, let us know. Here are some ways to support Ansible documentation:

- *Editing docs directly on GitHub*
- *Reviewing open PRs and issues*
- *Opening a new issue and/or PR*
- *Verifying your documentation PR*
 - *Setting up your environment to build documentation locally*
 - *Testing the documentation locally*
 - *Building the documentation locally*
 - * *Building a single rST page*
 - * *Building all the rST pages*
 - * *Building module docs and rST pages*
 - * *Building rST files with `sphinx-build`*
 - * *Running the final tests*
- *Joining the documentation working group*

Editing docs directly on GitHub

For typos and other quick fixes, you can edit the documentation right from the site. Look at the top right corner of this page. That **Edit on GitHub** link is available on every page in the documentation. If you have a GitHub account, you can submit a quick and easy pull request this way.

To submit a documentation PR from docs.ansible.com with **Edit on GitHub**:

1. Click on **Edit on GitHub**.
2. If you don't already have a fork of the ansible repo on your GitHub account, you'll be prompted to create one.
3. Fix the typo, update the example, or make whatever other change you have in mind.
4. Enter a commit message in the first rectangle under the heading **Propose file change** at the bottom of the GitHub page. The more specific, the better. For example, "fixes typo in my_module description". You can put more detail in the second rectangle if you like. Leave the `+label: docsite_pr` there.
5. Submit the suggested change by clicking on the green "Propose file change" button. GitHub will handle branching and committing for you, and open a page with the heading "Comparing Changes".
6. Click on **Create pull request** to open the PR template.
7. Fill out the PR template, including as much detail as appropriate for your change. You can change the title of your PR if you like (by default it's the same as your commit message). In the **Issue Type**

section, delete all lines except the `Docs Pull Request` line.

8. Submit your change by clicking on `Create pull request` button.
9. Be patient while Ansibot, our automated script, adds labels, pings the docs maintainers, and kicks off a CI testing run.
10. Keep an eye on your PR - the docs team may ask you for changes.

Reviewing open PRs and issues

You can also contribute by reviewing open documentation [issues](#) and [PRs](#). To add a helpful review, please:

- Include a comment - “looks good to me” only helps if we know why.
- For issues, reproduce the problem.
- For PRs, test the change.

Opening a new issue and/or PR

If the problem you’ve noticed is too complex to fix with the `Edit on GitHub` option, and no open issue or PR already documents the problem, please open an issue and/or a PR on the `ansible/ansible` repo.

A great documentation GitHub issue or PR includes:

- a specific title
- a detailed description of the problem (even for a PR - it’s hard to evaluate a suggested change unless we know what problem it’s meant to solve)
- links to other information (related issues/PRs, external documentation, pages on `docs.ansible.com`, and so on.)

Verifying your documentation PR

If you make multiple changes to the documentation, or add more than a line to it, before you open a pull request, please:

1. Check that your text follows our [Ansible style guide](#).
2. Test your changes for rST errors.
3. Build the page, and preferably the entire documentation site, locally.

Setting up your environment to build documentation locally

To build documentation locally, ensure you have a working [development environment](#).

To work with documentation on your local machine, you need to have python-3.5 or greater and the following packages installed:

- gcc
- jinja2
- libyaml
- Pygments >= 2.4.0
- pyparsing
- PyYAML
- rstcheck
- six
- sphinx
- sphinx-notfound-page
- straight.plugin

These required packages are listed in two `requirements.txt` files to make installation easier:

```
pip install --user -r requirements.txt
pip install --user -r docs/docsite/requirements.txt
```

You can drop `--user` if you have set up a virtual environment (venv/virtenv).

注解: On macOS with Xcode, you may need to install `six` and `pyparsing` with `--ignore-installed` to get versions that work with `sphinx`.

Testing the documentation locally

To test an individual file for rST errors:

```
rstcheck changed_file.rst
```

Building the documentation locally

Building the documentation is the best way to check for errors and review your changes. Once `rstcheck` runs with no errors, navigate to `ansible/docs/docsite` and then build the page(s) you want to review.

Building a single rST page

To build a single rST file with the make utility:

```
make htmlesingle rst=path/to/your_file.rst
```

For example:

```
make htmlesingle rst=community/documentation_contributions.rst
```

This process compiles all the links but provides minimal log output. If you're writing a new page or want more detailed log output, refer to the instructions on *Building rST files with sphinx-build*

注解: `make htmlesingle` adds `rst/` to the beginning of the path you provide in `rst=`, so you can't type the filename with autocomplete. Here are the error messages you will see if you get this wrong:

- If you run `make htmlesingle` from the `docs/docsite/rst/` directory: `make: *** No rule to make target `htmlesingle'. Stop.`
 - If you run `make htmlesingle` from the `docs/docsite/` directory with the full path to your rST document: `sphinx-build: error: cannot find files ['rst/rst/community/documentation_contributions.rst'].`
-

Building all the rST pages

To build all the rST files without any module documentation:

```
MODULES=None make webdocs
```

Building module docs and rST pages

To build documentation for a few modules plus all the rST files, use a comma-separated list:

```
MODULES=one_module,another_module make webdocs
```

To build all the module documentation plus all the rST files:

```
make webdocs
```

Building rST files with sphinx-build

Advanced users can build one or more rST files with the sphinx utility directly. `sphinx-build` returns misleading `undefined label` warnings if you only build a single page, because it does not create internal links. However, `sphinx-build` returns more extensive syntax feedback, including warnings about indentation errors and `x-string without end-string` warnings. This can be useful, especially if you're creating a new page from scratch. To build a page or pages with `sphinx-build`:

```
sphinx-build [options] sourcedir outdir [filenames...]
```

You can specify filenames, or `-a` for all files, or omit both to compile only new/changed files.

For example:

```
sphinx-build -b html -c rst/ rst/dev_guide/ _build/html/dev_guide/ rst/dev_guide/  
→developing_modules_documenting.rst
```

Running the final tests

When you submit a documentation pull request, automated tests are run. Those same tests can be run locally. To do so, navigate to the repository's top directory and run:

```
make clean &&  
bin/ansible-test sanity --test docs-build &&  
bin/ansible-test sanity --test rstcheck
```

Unfortunately, leftover rST-files from previous document-generating can occasionally confuse these tests. It is therefore safest to run them on a clean copy of the repository, which is the purpose of `make clean`. If you type these three lines one at a time and manually check the success of each, you do not need the `&&`.

Joining the documentation working group

The Documentation Working Group is just getting started, please visit the [community repo](#) for more information.

参见:

More about testing module documentation

More about documenting modules

Communicating

- *Code of Conduct*
- *Mailing list information*
- *IRC channels*
 - *General channels*
 - *Working groups*
 - *Regional and Language-specific channels*
 - *IRC meetings*
- *Ansible Tower support questions*

Code of Conduct

Please read and understand the *Community Code of Conduct*.

Mailing list information

Ansible has several mailing lists. Your first post to the mailing list will be moderated (to reduce spam), so please allow up to a day or so for your first post to appear.

- [Ansible Announce list](#) is a read-only list that shares information about new releases of Ansible, and also rare infrequent event information, such as announcements about an upcoming AnsibleFest, which is our official conference series. Worth subscribing to!
- [Ansible AWX List](#) is for [Ansible AWX](#) the upstream version of [Red Hat Ansible Tower](#)
- [Ansible Container List](#) is for users and developers of the Ansible Container project.
- [Ansible Development List](#) is for learning how to develop on Ansible, asking about prospective feature design, or discussions about extending ansible or features in progress.
- [Ansible Lockdown List](#) is for all things related to Ansible Lockdown projects, including DISA STIG automation and CIS Benchmarks.
- [Ansible Outreach List](#) help with promoting Ansible and [Ansible Meetups](#)
- [Ansible Project List](#) is for sharing Ansible tips, answering questions, and general user discussion.
- [Molecule List](#) is designed to aid with the development and testing of Ansible roles with Molecule.

To subscribe to a group from a non-Google account, you can send an email to the subscription address requesting the subscription. For example: `ansible-devel+subscribe@googlegroups.com`

IRC channels

Ansible has several IRC channels on [Freenode](#).

Our IRC channels may require you to register your nickname. If you receive an error when you connect, see [Freenode's Nickname Registration guide](#) for instructions.

General channels

- **#ansible** - For general use questions and support.
- **#ansible-devel** - For discussions on developer topics and code related to features or bugs.
- **#ansible-meeting** - For public community meetings. We will generally announce these on one or more of the above mailing lists. See the [meeting schedule and agenda page](#)

Working groups

Many of our community [Working Groups](#) meet on Freenode IRC channels. If you want to get involved in a working group, join the channel where it meets or comment on the agenda.

- [Amazon \(AWS\) Working Group](#) - **#ansible-aws**
- [Ansible Lockdown Working Group | gh/ansible/ansible-lockdown](#) - **#ansible-lockdown**- Security playbooks/roles
- [AWX Working Group](#) - **#ansible-awx** - Upstream for Ansible Tower
- [Azure Working Group](#) - **#ansible-azure**
- [Community Working Group](#) - **#ansible-community** - Including Meetups
- [Container Working Group](#) - **#ansible-container**
- [Contributor Experience Working Group](#) - **#ansible-community**
- [Docker Working Group](#) - **#ansible-devel**
- [Documentation Working Group](#)- **#ansible-docs**
- [Galaxy Working Group](#) - **#ansible-galaxy**
- [JBoss Working Group](#) - **#ansible-jboss**
- [Lightbulb Training](#) - **#ansible-lightbulb** - Ansible training
- [Linode Working Group](#) - **#ansible-linode**
- [Molecule Working Group | molecule.io](#) - **#ansible-molecule** - testing platform for Ansible playbooks and roles
- [Network Working Group](#) - **#ansible-network**

- Remote Management Working Group - [#ansible-devel](#)
- Testing Working Group - [#ansible-devel](#)
- VMware Working Group - [#ansible-vmware](#)
- Windows Working Group - [#ansible-windows](#)

Want to form a new Working Group?

Regional and Language-specific channels

- [#ansible-es](#) - Channel for Spanish speaking Ansible community.
- [#ansibleeu](#) - Channel for the European Ansible Community.
- [#ansible-fr](#) - Channel for French speaking Ansible community.
- [#ansiblerzh](#) - Channel for Zurich/Swiss Ansible community.

IRC meetings

The Ansible community holds regular IRC meetings on various topics, and anyone who is interested is invited to participate. For more information about Ansible meetings, consult the [meeting schedule and agenda page](#).

Ansible Tower support questions

Red Hat Ansible Tower is a UI, Server, and REST endpoint for Ansible. The Red Hat Ansible Automation subscription contains support for Ansible, Ansible Tower, Ansible Automation for Networking, and more.

If you have a question about Ansible Tower, visit [Red Hat support](#) rather than using the IRC channel or the general project mailing list.

The Ansible Development Cycle

The Ansible development cycle happens on two levels. At a macro level, the team plans releases and tracks progress with roadmaps and projects. At a micro level, each PR has its own lifecycle.

- *Macro development: roadmaps, releases, and projects*
- *Micro development: the lifecycle of a PR*
 - *Automated PR review: ansibullbot*
 - * *Ansibot workflow*
 - * *PR labels*

- *Workflow labels*
- *Information labels*
- *Special Labels*
- *Human PR review*
- *Making your PR merge-worthy*
 - *Changelogs*
 - * *Creating a changelog fragment*
- *Backporting merged PRs*

Macro development: roadmaps, releases, and projects

If you want to follow the conversation about what features will be added to Ansible for upcoming releases and what bugs are being fixed, you can watch these resources:

- the *Ansible Roadmap*
- the *Ansible Release Schedule*
- various GitHub projects - for example:
 - the 2.10 release project
 - the network bugs project
 - the core documentation project

Micro development: the lifecycle of a PR

Ansible accepts code through **pull requests** (“PRs” for short). GitHub provides a great overview of [how the pull request process works](#) in general. The ultimate goal of any pull request is to get merged and become part of Ansible Core. Here’ s an overview of the PR lifecycle:

- Contributor opens a PR
- Ansibot reviews the PR
- Ansibot assigns labels
- Ansibot pings maintainers
- Shippable runs the test suite
- Developers, maintainers, community review the PR
- Contributor addresses any feedback from reviewers

- Developers, maintainers, community re-review
- PR merged or closed

Automated PR review: **ansibullbot**

Because Ansible receives many pull requests, and because we love automating things, we've automated several steps of the process of reviewing and merging pull requests with a tool called Ansibullbot, or Ansibot for short.

Ansibullbot serves many functions:

- Responds quickly to PR submitters to thank them for submitting their PR
- Identifies the community maintainer responsible for reviewing PRs for any files affected
- Tracks the current status of PRs
- Pings responsible parties to remind them of any PR actions for which they may be responsible
- Provides maintainers with the ability to move PRs through the workflow
- Identifies PRs abandoned by their submitters so that we can close them
- Identifies modules abandoned by their maintainers so that we can find new maintainers

Ansibot workflow

Ansibullbot runs continuously. You can generally expect to see changes to your issue or pull request within thirty minutes. Ansibullbot examines every open pull request in the repositories, and enforces state roughly according to the following workflow:

- If a pull request has no workflow labels, it's considered **new**. Files in the pull request are identified, and the maintainers of those files are pinged by the bot, along with instructions on how to review the pull request. (Note: sometimes we strip labels from a pull request to “reboot” this process.)
- If the module maintainer is not `$team_ansible`, the pull request then goes into the **community_review** state.
- If the module maintainer is `$team_ansible`, the pull request then goes into the **core_review** state (and probably sits for a while).
- If the pull request is in **community_review** and has received comments from the maintainer:
 - If the maintainer says `shipit`, the pull request is labeled **shipit**, whereupon the Core team assesses it for final merge.
 - If the maintainer says `needs_info`, the pull request is labeled **needs_info** and the submitter is asked for more info.

- If the maintainer says **needs_revision**, the pull request is labeled **needs_revision** and the submitter is asked to fix some things.
- If the submitter says **ready_for_review**, the pull request is put back into **community_review** or **core_review** and the maintainer is notified that the pull request is ready to be reviewed again.
- If the pull request is labeled **needs_revision** or **needs_info** and the submitter has not responded lately:
 - The submitter is first politely pinged after two weeks, pinged again after two more weeks and labeled **pending_action**, and the issue or pull request will be closed two weeks after that.
 - If the submitter responds at all, the clock is reset.
- If the pull request is labeled **community_review** and the reviewer has not responded lately:
 - The reviewer is first politely pinged after two weeks, pinged again after two more weeks and labeled **pending_action**, and then may be reassigned to **\$team_ansible** or labeled **core_review**, or often the submitter of the pull request is asked to step up as a maintainer.
- If Shippable tests fail, or if the code is not able to be merged, the pull request is automatically put into **needs_revision** along with a message to the submitter explaining why.

There are corner cases and frequent refinements, but this is the workflow in general.

PR labels

There are two types of PR Labels generally: **workflow** labels and **information** labels.

Workflow labels

- **community_review**: Pull requests for modules that are currently awaiting review by their maintainers in the Ansible community.
- **core_review**: Pull requests for modules that are currently awaiting review by their maintainers on the Ansible Core team.
- **needs_info**: Waiting on info from the submitter.
- **needs_rebase**: Waiting on the submitter to rebase.
- **needs_revision**: Waiting on the submitter to make changes.
- **shipit**: Waiting for final review by the core team for potential merge.

Information labels

- **backport**: this is applied automatically if the PR is requested against any branch that is not devel. The bot immediately assigns the labels backport and **core_review**.

- **bugfix_pull_request**: applied by the bot based on the templated description of the PR.
- **cloud**: applied by the bot based on the paths of the modified files.
- **docs_pull_request**: applied by the bot based on the templated description of the PR.
- **easyfix**: applied manually, inconsistently used but sometimes useful.
- **feature_pull_request**: applied by the bot based on the templated description of the PR.
- **networking**: applied by the bot based on the paths of the modified files.
- **owner_pr**: largely deprecated. Formerly workflow, now informational. Originally, PRs submitted by the maintainer would automatically go to **shipit** based on this label. If the submitter is also a maintainer, we notify the other maintainers and still require one of the maintainers (including the submitter) to give a **shipit**.
- **pending_action**: applied by the bot to PRs that are not moving. Reviewed every couple of weeks by the community team, who tries to figure out the appropriate action (closure, asking for new maintainers, and so on).

Special Labels

- **new_plugin**: this is for new modules or plugins that are not yet in Ansible.

Note: *new_plugin* kicks off a completely separate process, and frankly it doesn't work very well at present. We're working our best to improve this process.

Human PR review

After Ansibot reviews the PR and applies labels, the PR is ready for human review. The most likely reviewers for any PR are the maintainers for the module that PR modifies.

Each module has at least one assigned *maintainer*, listed in the `BOTMETA.yml` file.

The maintainer's job is to review PRs that affect that module and decide whether they should be merged (**shipit**) or revised (**needs_revision**). We'd like to have at least one community maintainer for every module. If a module has no community maintainers assigned, the maintainer is listed as **\$team_ansible**.

Once a human applies the **shipit** label, the *committers* decide whether the PR is ready to be merged. Not every PR that gets the **shipit** label is actually ready to be merged, but the better our reviewers are, and the better our guidelines are, the more likely it will be that a PR that reaches **shipit** will be mergeable.

Making your PR merge-worthy

We don't merge every PR. Here are some tips for making your PR useful, attractive, and merge-worthy.

Changelogs

Changelogs help users and developers keep up with changes to Ansible. Ansible builds a changelog for each release from fragments. You **must** add a changelog fragment to any PR that changes functionality or fixes a bug. You don't have to add a changelog fragment for PRs that add new modules and plugins, because our tooling does that for you automatically.

We build short summary changelogs for minor releases as well as for major releases. If you backport a bugfix, include a changelog fragment with the backport PR.

Creating a changelog fragment

A basic changelog fragment is a `.yaml` file placed in the `changelogs/fragments/` directory. Each file contains a yaml dict with keys like `bugfixes` or `major_changes` followed by a list of changelog entries of bugfixes or features. Each changelog entry is rst embedded inside of the yaml file which means that certain constructs would need to be escaped so they can be interpreted by rst and not by yaml (or escaped for both yaml and rst if that's your desire). Each PR **must** use a new fragment file rather than adding to an existing one, so we can trace the change back to the PR that introduced it.

To create a changelog entry, create a new file with a unique name in the `changelogs/fragments/` directory. The file name should include the PR number and a description of the change. It must end with the file extension `.yaml`. For example: `40696-user-backup-shadow-file.yaml`

A single changelog fragment may contain multiple sections but most will only contain one section. The toplevel keys (`bugfixes`, `major_changes`, and so on) are defined in the [config file](#) for our release note tool. Here are the valid sections and a description of each:

major_changes Major changes to Ansible itself. Generally does not include module or plugin changes.

minor_changes Minor changes to Ansible, modules, or plugins. This includes new features, new parameters added to modules, or behavior changes to existing parameters.

deprecated_features Features that have been deprecated and are scheduled for removal in a future release.

removed_features Features that were previously deprecated and are now removed.

bugfixes Fixes that resolve issues. If there is a specific issue related to this bugfix, add a link in the changelog entry.

known_issues Known issues that are currently not fixed or will not be fixed.

Most changelog entries will be `bugfixes` or `minor_changes`. When writing a changelog entry that pertains to a particular module, start the entry with `- [module name] -` and include a link to the related issue if one exists.

Here are some examples:

bugfixes:

- win_updates - fixed issue where running win_updates on async fails without any error

minor_changes:

- lineinfile - add warning when using an empty regexp (<https://github.com/ansible/ansible/issues/29443>)

bugfixes:

- copy module - The copy module was attempting to change the mode of files for remote_src=True even if mode was not set as a parameter. This failed on filesystems which do not have permission bits.

You can find more example changelog fragments in the [changelog directory](#) for the 2.6 release. You can also find documentation of the format, including hints on embedding rst in the yaml, in the [reno documentation](#).

Once you've written the changelog fragment for your PR, commit the file and include it with the pull request.

Backporting merged PRs

All Ansible PRs must be merged to the `devel` branch first. After a pull request has been accepted and merged to the `devel` branch, the following instructions will help you create a pull request to backport the change to a previous stable branch.

We do **not** backport features.

注解: These instructions assume that:

- `stable-2.9` is the targeted release branch for the backport
- <https://github.com/ansible/ansible.git> is configured as a `git remote` named `upstream`. If you do not use a `git remote` named `upstream`, adjust the instructions accordingly.
- <https://github.com/<yourgithubaccount>/ansible.git> is configured as a `git remote` named `origin`. If you do not use a `git remote` named `origin`, adjust the instructions accordingly.

1. Prepare your `devel`, `stable`, and feature branches:

```
git fetch upstream
git checkout -b backport/2.9/[PR_NUMBER_FROM_DEVEL] upstream/stable-2.9
```

2. Cherry pick the relevant commit SHA from the `devel` branch into your feature branch, handling merge conflicts as necessary:

```
git cherry-pick -x [SHA_FROM_DEVEL]
```

3. Add a *changelog fragment* for the change, and commit it.
4. Push your feature branch to your fork on GitHub:

```
git push origin backport/2.9/[PR_NUMBER_FROM_DEVEL]
```

5. Submit the pull request for backport/2.9/[PR_NUMBER_FROM_DEVEL] against the stable-2.9 branch
6. The Release Manager will decide whether to merge the backport PR before the next minor release. There isn't any need to follow up. Just ensure that the automated tests (CI) are green.

注解: The choice to use backport/2.9/[PR_NUMBER_FROM_DEVEL] as the name for the feature branch is somewhat arbitrary, but conveys meaning about the purpose of that branch. It is not required to use this format, but it can be helpful, especially when making multiple backport PRs for multiple stable branches.

注解: If you prefer, you can use CPython's cherry-picker tool (`pip install --user 'cherry-picker >= 1.3.2'`) to backport commits from devel to stable branches in Ansible. Take a look at the [cherry-picker documentation](#) for details on installing, configuring, and using it.

Contributors License Agreement

By contributing you agree that these contributions are your own (or approved by your employer) and you grant a full, complete, irrevocable copyright license to all users and developers of the project, present and future, pursuant to the license of the project.

Triage Process

The issue and PR triage processes are driven by the [Ansibot](#). Whenever an issue or PR is filed, the Ansibot examines the issue to ensure that all relevant data is present, and handles the routing of the issue as it works its way to eventual completion.

For details on how Ansibot manages the triage process, please consult the [Ansibot Issue Guide](#).

Other Tools And Programs

- *Popular Editors*

- *Atom*
- *Emacs*
- *PyCharm*
- *Sublime*
- *Visual Studio Code*
- *vim*
- *Development Tools*
 - *Finding related issues and PRs*
- *Tools for Validating Playbooks*
- *Other Tools*

The Ansible community uses a range of tools for working with the Ansible project. This is a list of some of the most popular of these tools.

If you know of any other tools that should be added, this list can be updated by clicking “Edit on GitHub” on the top right of this page.

Popular Editors

Atom

An open-source, free GUI text editor created and maintained by GitHub. You can keep track of git project changes, commit from the GUI, and see what branch you are on. You can customize the themes for different colors and install syntax highlighting packages for different languages. You can install Atom on Linux, macOS and Windows. Useful Atom plugins include:

- `language-yaml` - YAML highlighting for Atom (built-in).
- `linter-js-yaml` - parses your YAML files in Atom through js-yaml.

Emacs

A free, open-source text editor and IDE that supports auto-indentation, syntax highlighting and built in terminal shell(among other things).

- `yaml-mode` - YAML highlighting and syntax checking.
- `jinja2-mode` - Jinja2 highlighting and syntax checking.
- `magit-mode` - Git porcelain within Emacs.

PyCharm

A full IDE (integrated development environment) for Python software development. It ships with everything you need to write python scripts and complete software, including support for YAML syntax highlighting. It's a little overkill for writing roles/playbooks, but it can be a very useful tool if you write modules and submit code for Ansible. Can be used to debug the Ansible engine.

Sublime

A closed-source, subscription GUI text editor. You can customize the GUI with themes and install packages for language highlighting and other refinements. You can install Sublime on Linux, macOS and Windows. Useful Sublime plugins include:

- [GitGutter](#) - shows information about files in a git repository.
- [SideBarEnhancements](#) - provides enhancements to the operations on Sidebar of Files and Folders.
- [Sublime Linter](#) - a code-linting framework for Sublime Text 3.
- [Pretty YAML](#) - prettifies YAML for Sublime Text 2 and 3.
- [Yamllint](#) - a Sublime wrapper around yamllint.

Visual Studio Code

An open-source, free GUI text editor created and maintained by Microsoft. Useful Visual Studio Code plugins include:

- [YAML Support by Red Hat](#) - provides YAML support through yamllanguage-server with built-in Kubernetes and Kedge syntax support.
- [Ansible Syntax Highlighting Extension](#) - YAML & Jinja2 support.
- [Visual Studio Code extension for Ansible](#) - provides autocompletion, syntax highlighting.

vim

An open-source, free command-line text editor. Useful vim plugins include:

- [Ansible vim](#) - vim syntax plugin for Ansible 2.x, it supports YAML playbooks, Jinja2 templates, and Ansible's hosts files.

Development Tools

Finding related issues and PRs

There are various ways to find existing issues and pull requests (PRs)

- [PR by File](#) - shows a current list of all open pull requests by individual file. An essential tool for Ansible module maintainers.
- [jctanner's Ansible Tools](#) - miscellaneous collection of useful helper scripts for Ansible development.

Tools for Validating Playbooks

- [Ansible Lint](#) - the official, highly configurable best-practices linter for Ansible playbooks, by Ansible.
- [Ansible Review](#) - an extension of Ansible Lint designed for code review.
- [Molecule](#) is a testing framework for Ansible plays and roles, by Ansible
- [yamllint](#) is a command-line utility to check syntax validity including key repetition and indentation issues.

Other Tools

- [Ansible cmdb](#) - takes the output of Ansible's fact gathering and converts it into a static HTML overview page containing system configuration information.
- [Ansible Inventory Grapher](#) - visually displays inventory inheritance hierarchies and at what level a variable is defined in inventory.
- [Ansible Playbook Grapher](#) - A command line tool to create a graph representing your Ansible playbook tasks and roles.
- [Ansible Shell](#) - an interactive shell for Ansible with built-in tab completion for all the modules.
- [Ansible Silo](#) - a self-contained Ansible environment by Docker.
- [Ansigenome](#) - a command line tool designed to help you manage your Ansible roles.
- [ARA](#) - records Ansible playbook runs and makes the recorded data available and intuitive for users and systems by integrating with Ansible as a callback plugin.
- [Awesome Ansible](#) - a collaboratively curated list of awesome Ansible resources.
- [AWX](#) - provides a web-based user interface, REST API, and task engine built on top of Ansible. AWX is the upstream project for Red Hat Ansible Tower, part of the Red Hat Ansible Automation subscription.
- [Mitogen for Ansible](#) - uses the [Mitogen](#) library to execute Ansible playbooks in a more efficient way (decreases the execution time).

- [nanvault](#) - a standalone tool to encrypt and decrypt files in the Ansible Vault format, featuring UNIX-style composability.
- [OpsTools-ansible](#) - uses Ansible to configure an environment that provides the support of [OpsTools](#), namely centralized logging and analysis, availability monitoring, and performance monitoring.
- [TD4A](#) - a template designer for automation. TD4A is a visual design aid for building and testing jinja2 templates. It will combine data in yaml format with a jinja2 template and render the output.

Ansible style guide

Welcome to the Ansible style guide! To create clear, concise, consistent, useful materials on docs.ansible.com, follow these guidelines:

- *Linguistic guidelines*
 - *Stylistic cheat-sheet*
 - *Header case*
 - *Avoid using Latin phrases*
- *reStructuredText guidelines*
 - *Header notation*
 - *Internal navigation*
 - * *Adding anchors*
 - * *Adding internal links*
 - * *Adding links to modules and plugins*
 - * *Adding local TOCs*
- *More resources*

Linguistic guidelines

We want the Ansible documentation to be:

- clear
- direct
- conversational
- easy to translate

We want reading the docs to feel like having an experienced, friendly colleague explain how Ansible works.

Stylistic cheat-sheet

This cheat-sheet illustrates a few rules that help achieve the “Ansible tone” :

Rule	Good example	Bad example
Use active voice	You can run a task by	A task can be run by
Use the present tense	This command creates a	This command will create a
Address the reader	As you expand your inventory	When the number of managed nodes grows
Use standard English	Return to this page	Hop back to this page
Use American English	The color of the output	The colour of the output

Header case

Headers should be written in sentence case. For example, this section’ s title is **Header case**, not **Header Case** or **HEADER CASE**.

Avoid using Latin phrases

Latin words and phrases like **e.g.** or **etc.** are easily understood by English speakers. They may be harder to understand for others and are also tricky for automated translation.

Use the following English terms in place of Latin terms or abbreviations:

Latin	English
i.e	in other words
e.g.	for example
etc	and so on
via	by/ through
vs./versus	rather than/against

reStructuredText guidelines

The Ansible documentation is written in reStructuredText and processed by Sphinx. We follow these technical or mechanical guidelines on all rST pages:

Header notation

Section headers in reStructuredText can use a variety of notations. Sphinx will ‘learn on the fly’ when creating a hierarchy of headers. To make our documents easy to read and to edit, we follow a standard set of header notations. We use:

- ### with overline, for parts:

```
#####
Developer guide
#####
```

- *** with overline, for chapters:

```
*****
Ansible style guide
*****
```

- === for sections:

```
Mechanical guidelines
=====
```

- --- for subsections:

```
Internal navigation
-----
```

- ^^^ for sub-subsections:

```
Adding anchors
^^^^^^^^^^^^^^
```

- "" for paragraphs:

```
Paragraph that needs a title
""""""""""
```

Internal navigation

[Anchors](#) (also called [labels](#)) and [links](#) work together to help users find related content. Local tables of contents also help users navigate quickly to the information they need. All internal links should use the `:ref:` syntax. Every page should have at least one anchor to support internal `:ref:` links. Long pages, or pages with multiple levels of headers, can also include a local TOC.

Adding anchors

- Include at least one anchor on every page

- Place the main anchor above the main header
- If the file has a unique title, use that for the main page anchor:

```
.. _unique_page::
```

- You may also add anchors elsewhere on the page

Adding internal links

- All internal links must use `:ref:` syntax. These links both point to the anchor defined above:

```
:ref:`unique_page`  
:ref:`this page <unique_page>`
```

The second example adds custom text for the link.

Adding links to modules and plugins

- Module links use the module name followed by `_module` for the anchor.
- Plugin links use the plugin name followed by the plugin type. For example, `enable become plugin`).

```
:ref:`this module <this_module>`  
:ref:`that connection plugin <that_connection>`
```

Adding local TOCs

The page you're reading includes a [local TOC](#). If you include a local TOC:

- place it below, not above, the main heading and (optionally) introductory text
- use the `:local:` directive so the page's main header is not included
- do not include a title

The syntax is:

```
.. contents::  
   :local:
```

More resources

These pages offer more help with grammatical, stylistic, and technical rules for documentation.

Basic rules

- *Use standard American English*
- *Write for a global audience*
- *Follow naming conventions*
- *Use clear sentence structure*
- *Avoid verbosity*
- *Highlight menu items and commands*

Use standard American English

Ansible uses Standard American English. Watch for common words that are spelled differently in American English (color vs colour, organize vs organise, etc.).

Write for a global audience

Everything you say should be understandable by people of different backgrounds and cultures. Avoid idioms and regionalism and maintain a neutral tone that cannot be misinterpreted. Avoid attempts at humor.

Follow naming conventions

Always follow naming conventions and trademarks.

Use clear sentence structure

Clear sentence structure means:

- Start with the important information first.
- Avoid padding/adding extra words that make the sentence harder to understand.
- Keep it short - Longer sentences are harder to understand.

Some examples of improving sentences:

Bad: The unwise walking about upon the area near the cliff edge may result in a dangerous fall and therefore it is recommended that one remains a safe distance to maintain personal safety.

Better: Danger! Stay away from the cliff.

Bad: Furthermore, large volumes of water are also required for the process of extraction.

Better: Extraction also requires large volumes of water.

Avoid verbosity

Write short, succinct sentences. Avoid terms like:

- “...as has been said before,”
- “..each and every,”
- “...point in time,”
- “...in order to,”

Highlight menu items and commands

When documenting menus or commands, it helps to **bold** what is important.

For menu procedures, bold the menu names, button names, etc to help the user find them on the GUI:

1. On the **File** menu, click **Open**.
2. Type a name in the **User Name** field.
3. In the **Open** dialog box, click **Save**.
4. On the toolbar, click the **Open File** icon.

For code or command snippets, use the RST `code-block` directive:

```
.. code-block:: bash

ssh my_vyos_user@vyos.example.net
show config
```

Voice Style

The essence of the Ansible writing style is short sentences that flow naturally together. Mix up sentence structures. Vary sentence subjects. Address the reader directly. Ask a question. And when the reader adjusts to the pace of shorter sentences, write a longer one.

- Write how real people speak...
- ...but try to avoid slang and colloquialisms that might not translate well into other languages.
- Say big things with small words.
- Be direct. Tell the reader exactly what you want them to do.
- Be honest.

- Short sentences show confidence.
- Grammar rules are meant to be bent, but only if the reader knows you are doing this.
- Choose words with fewer syllables for faster reading and better understanding.
- Think of copy as one-on-one conversations rather than as a speech. It's more difficult to ignore someone who is speaking to you directly.
- When possible, start task-oriented sentences (those that direct a user to do something) with action words. For example: Find software...Contact support...Install the media... and so forth.

Active Voice

Use the active voice (“Start Linuxconf by typing...”) rather than passive (“Linuxconf can be started by typing...”) whenever possible. Active voice makes for more lively, interesting reading. Also avoid future tense (or using the term “will”) whenever possible. For example, future tense (“The screen will display...”) does not read as well as an active voice (“The screen displays”). Remember, the users you are writing for most often refer to the documentation while they are using the system, not after or in advance of using the system.

Trademark Usage

Why is it important to use the TM, SM, and ® for our registered marks?

Before a trademark is registered with the United States Patent and Trademark Office it is appropriate to use the TM or SM symbol depending whether the product is for goods or services. It is important to use the TM or SM as it is notification to the public that Ansible claims rights to the mark even though it has not yet been registered.

Once the trademark is registered, it is appropriate to use the symbol in place of the TM or SM. The symbol designation must be used in conjunction with the trademark if Ansible is to fully protect its rights. If we don't protect these marks, we run the risk of losing them in the way of Aspirin or Trampoline or Escalator.

General Rules:

Trademarks should be used on 1st references on a page or within a section.

Use Red Hat® Ansible Tower® or Ansible®, on first reference when referring to products.

Use “Ansible” alone as the company name, as in “Ansible announced quarterly results,” which is not marked.

Also add the trademark disclaimer. * When using Ansible trademarks in the body of written text, you should use the following credit line in a prominent place, usually a footnote.

For Registered Trademarks: - [Name of Trademark] is a registered trademark of Red Hat, Inc. in the United States and other countries.

For Unregistered Trademarks (TMs/SMs): - [Name of Trademark] is a trademark of Red Hat, Inc. in the United States and other countries.

For registered and unregistered trademarks: - [Name of Trademark] is a registered trademark and [Name of Trademark] is a trademark of Red Hat, Inc. in the United States and other countries.

Guidelines for the proper use of trademarks:

Always distinguish trademarks from surround text with at least initial capital letters or in all capital letters.

Always use proper trademark form and spelling.

Never use a trademark as a noun. Always use a trademark as an adjective modifying the noun.

Correct: Red Hat® Ansible Tower® system performance is incredible.

Incorrect: Ansible' s performance is incredible.

Never use a trademark as a verb. Trademarks are products or services, never actions.

Correct: “Orchestrate your entire network using Red Hat® Ansible Tower®.”

Incorrect: “Ansible your entire network.”

Never modify a trademark to a plural form. Instead, change the generic word from the singular to the plural.

Correct: “Corporate demand for Red Hat® Ansible Tower® configuration software is surging.”

Incorrect: “Corporate demand for Ansible is surging.”

Never modify a trademark from its possessive form, or make a trademark possessive. Always use it in the form it has been registered.

Never translate a trademark into another language.

Never use trademarks to coin new words or names.

Never use trademarks to create a play on words.

Never alter a trademark in any way including through unapproved fonts or visual identifiers.

Never abbreviate or use any Ansible trademarks as an acronym.

The importance of Ansible trademarks

The Ansible trademark and the “A” logo in a shaded circle are our most valuable assets. The value of these trademarks encompass the Ansible Brand. Effective trademark use is more than just a name, it defines the level of quality the customer will receive and it ties a product or service to a corporate image. A trademark

may serve as the basis for many of our everyday decisions and choices. The Ansible Brand is about how we treat customers and each other. In order to continue to build a stronger more valuable Brand we must use it in a clear and consistent manner.

The mark consists of the letter “A” in a shaded circle. As of 5/11/15, this was a pending trademark (registration in process).

Common Ansible Trademarks

- Ansible®
- Ansible Tower®

Other Common Trademarks and Resource Sites:

- Linux is a registered trademark of Linus Torvalds.
- UNIX® is a registered trademark of The Open Group.
- Microsoft, Windows, Vista, XP, and NT are registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. <https://www.microsoft.com/en-us/legal/intellectualproperty/trademarks/en-us.aspx>
- Apple, Mac, Mac OS, Macintosh, Pages and TrueType are either registered trademarks or trademarks of Apple Computer, Inc. in the United States and/or other countries. <https://www.apple.com/legal/intellectual-property/trademark/appletmlist.html>
- Adobe, Acrobat, GoLive, InDesign, Illustrator, PostScript , PhotoShop and the OpenType logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries. <https://www.adobe.com/legal/permissions/trademarks.html>
- Macromedia and Macromedia Flash are trademarks of Macromedia, Inc. <https://www.adobe.com/legal/permissions/trademarks.html>
- IBM is a registered trademark of International Business Machines Corporation. <https://www.ibm.com/legal/us/en/copytrade.shtml>
- Celeron, Celeron Inside, Centrino, Centrino logo, Core Inside, Intel Core, Intel Inside, Intel Inside logo, Itanium, Itanium Inside, Pentium, Pentium Inside, VTune, Xeon, and Xeon Inside are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries. <https://www.intel.com/content/www/us/en/legal/trademarks.html>

Grammar and Punctuation

Common Styles and Usage, and Common Mistakes

Ansible

- Write “Ansible.” Not “Ansible, Inc.” or “AnsibleWorks” The only exceptions to this rule are when we’ re writing legal or financial statements.
- Never use the logotype by itself in body text. Always keep the same font you are using the rest of the sentence.
- A company is singular in the US. In other words, Ansible is an “it,” not a “they.”

Capitalization

If it’ s not a real product, service, or department at Ansible, don’ t capitalize it. Not even if it seems important. Capitalize only the first letter of the first word in headlines.

Colon

A colon is generally used before a list or series: - The Triangle Area consists of three cities: Raleigh, Durham, and Chapel Hill.

But not if the list is a complement or object of an element in the sentence: - Before going on vacation, be sure to (1) set the alarm, (2) cancel the newspaper, and (3) ask a neighbor to collect your mail.

Use a colon after “as follows” and “the following” if the related list comes immediately after: wedge The steps for changing directories are as follows:

1. Open a terminal.
2. Type cd...

Use a colon to introduce a bullet list (or dash, or icon/symbol of your choice):

In the Properties dialog box, you’ ll find the following entries:

- Connection name
- Count
- Cost per item

Commas

Use serial commas, the comma before the “and” in a series of three or more items:

- “Item 1, item 2, and item 3.”

It’ s easier to read that way and helps avoid confusion. The primary exception to this you will see is in PR, where it is traditional not to use serial commas because it is often the style of journalists.

Commas are always important, considering the vast difference in meanings of the following two statements.

- Let' s eat, Grandma
- Let' s eat Grandma.

Correct punctuation could save Grandma' s life.

If that does not convince you, maybe this will:



Contractions

Do not use contractions in Ansible documents.

Em dashes

When possible, use em-dashes with no space on either side. When full em-dashes aren't available, use double-dashes with no spaces on either side—like this.

A pair of em dashes can be used in place of commas to enhance readability. Note, however, that dashes are always more emphatic than commas.

A pair of em dashes can replace a pair of parentheses. Dashes are considered less formal than parentheses; they are also more intrusive. If you want to draw attention to the parenthetical content, use dashes. If you want to include the parenthetical content more subtly, use parentheses.

注解: When dashes are used in place of parentheses, surrounding punctuation should be omitted. Compare the following examples.

```
Upon discovering the errors (all 124 of them), the publisher immediately recalled the
↳books.
```

```
Upon discovering the errors—all 124 of them—the publisher immediately recalled the
↳books.
```

When used in place of parentheses at the end of a sentence, only a single dash is used.

```
After three weeks on set, the cast was fed up with his direction (or, rather, lack of
↳direction).
```

```
After three weeks on set, the cast was fed up with his direction—or, rather, lack of
↳direction.
```

Exclamation points (!)

Do not use them at the end of sentences. An exclamation point can be used when referring to a command, such as the bang (!) command.

Gender References

Do not use gender-specific pronouns in documentation. It is far less awkward to read a sentence that uses “they” and “their” rather than “he/she” and “his/hers.”

It is fine to use “you” when giving instructions and “the user,” “new users,” etc. in more general explanations.

Never use “one” in place of “you” when writing technical documentation. Using “one” is far too formal.

Never use “we” when writing. “We” aren’ t doing anything on the user side. Ansible’ s products are doing the work as requested by the user.

Hyphen

The hyphen’ s primary function is the formation of certain compound terms. Do not use a hyphen unless it serves a purpose. If a compound adjective cannot be misread or, as with many psychological terms, its meaning is established, a hyphen is not necessary.

Use hyphens to avoid ambiguity or confusion:

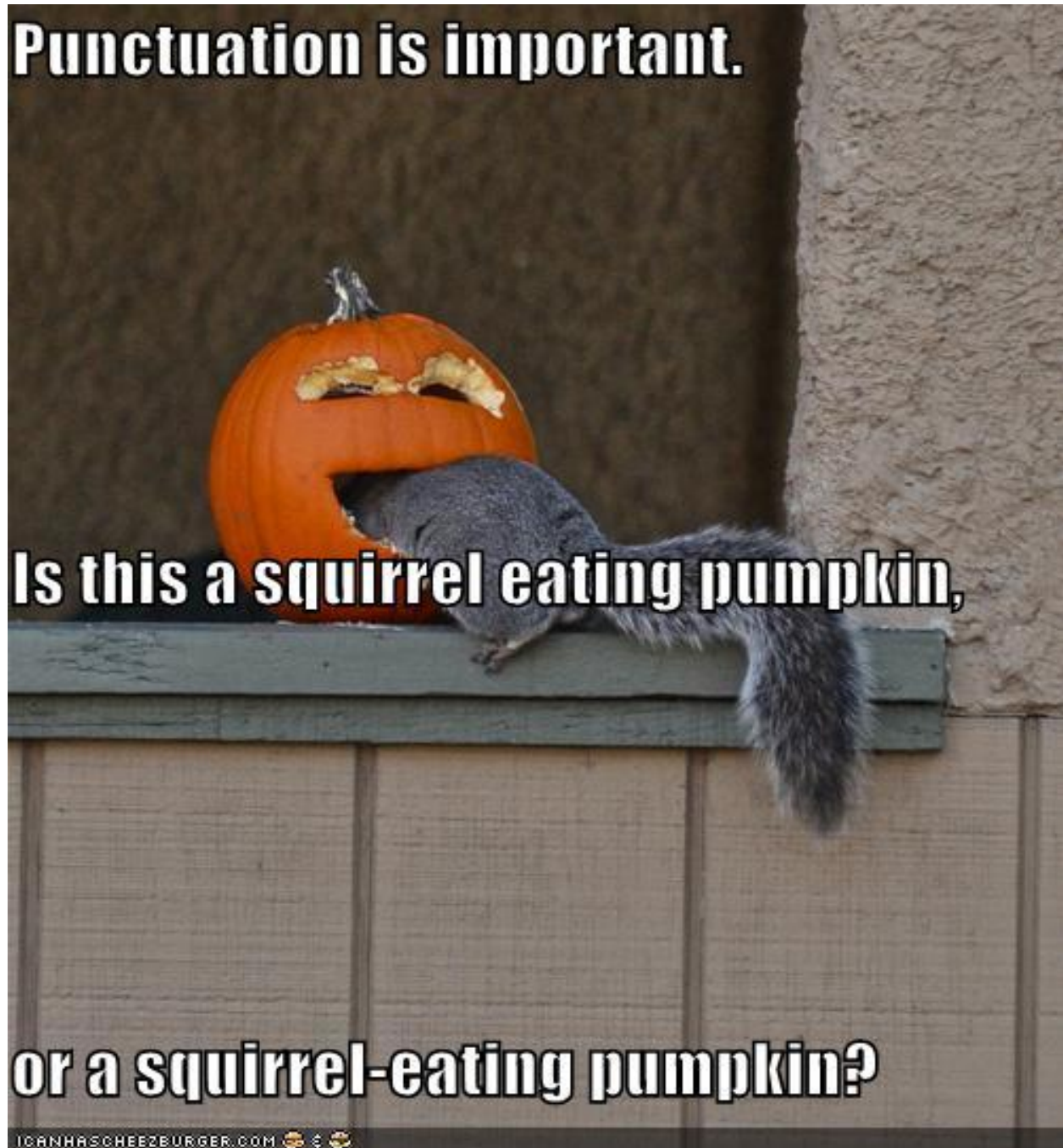
```
a little-used car
a little used-car

cross complaint
cross-complaint

high-school girl
high schoolgirl

fine-tooth comb (most people do not comb their teeth)

third-world war
third world war
```



In professionally printed material (particularly books, magazines, and newspapers), the hyphen is used to divide words between the end of one line and the beginning of the next. This allows for an evenly aligned right margin without highly variable (and distracting) word spacing.

Lists

Keep the structure of bulleted lists equivalent and consistent. If one bullet is a verb phrase, they should all be verb phrases. If one is a complete sentence, they should all be complete sentences, etc.

Capitalize the first word of each bullet. Unless it is obvious that it is just a list of items, such as a list of items like: * computer * monitor * keyboard * mouse

When the bulleted list appears within the context of other copy, (unless it's a straight list like the previous example) add periods, even if the bullets are sentence fragments. Part of the reason behind this is that each bullet is said to complete the original sentence.

In some cases where the bullets are appearing independently, such as in a poster or a homepage promotion, they do not need periods.

When giving instructional steps, use numbered lists instead of bulleted lists.

Months and States

Abbreviate months and states according to AP. Months are only abbreviated if they are used in conjunction with a day. Example: “The President visited in January 1999.” or “The President visited Jan. 12.”

Months: Jan., Feb., March, April, May, June, July, Aug., Sept., Nov., Dec.

States: Ala., Ariz., Ark., Calif., Colo., Conn., Del., Fla., Ga., Ill., Ind., Kan., Ky., La., Md., Mass., Mich., Minn., Miss., Mo., Mont., Neb., Nev., NH, NJ, NM, NY, NC, ND, Okla., Ore., Pa., RI, SC, SD, Tenn., Vt., Va., Wash., W.Va., Wis., Wyo.

Numbers

Numbers between one and nine are written out. 10 and above are numerals. The exception to this is writing “4 million” or “4 GB.” It's also acceptable to use numerals in tables and charts.

Phone Numbers

Phone number style: 1 (919) 555-0123 x002 and 1 888-GOTTEXT

Quotations (Using Quotation Marks and Writing Quotes)

“Place the punctuation inside the quotes,” the editor said.

Except in rare instances, use only “said” or “says” because anything else just gets in the way of the quote itself, and also tends to editorialize.

Place the name first right after the quote: “I like to write first-person because I like to become the character I'm writing,” Wally Lamb said.

Not: “I like to write first-person because I like to become the character I'm writing,” said Wally Lamb.

Semicolon

Use a semicolon to separate items in a series if the items contain commas:

- Everyday I have coffee, toast, and fruit for breakfast; a salad for lunch; and a peanut butter sandwich, cookies, ice cream, and chocolate cake for dinner.

Use a semicolon before a conjunctive adverb (however, therefore, otherwise, namely, for example, etc.): - I think; therefore, I am.

Spacing after sentences

Use only a single space after a sentence.

Time

- Time of day is written as “4 p.m.”

Spelling - Word Usage - Common Words and Phrases to Use and Avoid

Acronyms

Always uppercase. An acronym is a word formed from the initial letters of a name, such as ROM for Read-only memory, SaaS for Software as a Service, or by combining initial letters or part of a series of words, such as LILO for LInux LOader.

Spell out the acronym before using it in alone text, such as “The Embedded DevKit (EDK)…”

Applications

When used as a proper name, use the capitalization of the product, such as GNUPro, Source-Navigator, and Ansible Tower. When used as a command, use lowercase as appropriate, such as “To start GCC, type gcc.”

注解: “vi” is always lowercase.

As

This is often used to mean “because” , but has other connotations, for example, parallel or simultaneous actions. If you mean “because” , say “because” .

Asks for

Use “requests” instead.

Assure/Ensure/Insure

Assure implies a sort of mental comfort. As in “I assured my husband that I would eventually bring home beer.”

Ensure means “to make sure.”

Insure relates to monetary insurance.

Back up

This is a verb. You “back up” files; you do not “backup” files.

Backup

This is a noun. You create “backup” files; you do not create “back up” files.

Backward

Correct. Avoid using backwards unless you are stating that something has “backwards compatibility.”

Backwards compatibility

Correct as is.

By way of

Use “using” instead.

Can/May

Use “can” to describe actions or conditions that are possible. Use “may” only to describe situations where permission is being given. If either “can,” “could,” or “may” apply, use “can” because it’s less tentative.

CD or cd

When referring to a compact disk, use CD, such as “Insert the CD into the CD-ROM drive.” When referring to the change directory command, use cd.

CD-ROM

Correct. Do not use “cdrom,” “CD-Rom,” “CDROM,” “cd-rom” or any other variation. When referring to the drive, use CD-ROM drive, such as “Insert the CD into the CD-ROM drive.” The plural is “CD-ROMs.”

Command line

Correct. Do not use “command-line” or “commandline.”

Use to describes where to place options for a command, but not where to type the command. Use “shell prompt” instead to describe where to type commands. The line on the display screen where a command is expected. Generally, the command line is the line that contains the most recently displayed command prompt.

Daylight saving time (DST)

Correct. Do not use daylight savings time. Daylight Saving Time (DST) is often misspelled “Daylight Savings” , with an “s” at the end. Other common variations are “Summer Time” and “Daylight-Saving Time” . (<https://www.timeanddate.com/time/dst/daylight-savings-time.html>)

Download

Correct. Do not use “down load” or “down-load.”

e.g.

Spell it out: “For example.”

Failover

When used as a noun, a failover is a backup operation that automatically switches to a standby database, server or network if the primary system fails or is temporarily shut down for servicing. Failover is an important fault tolerance function of mission-critical systems that rely on constant accessibility. Failover automatically and transparently to the user redirects requests from the failed or down system to the backup system that mimics the operations of the primary system.

Fail over

When used as a verb, fail over is two words since there can be different tenses such as failed over.

Fewer

Fewer is used with plural nouns. Think things you could count. Time, money, distance, and weight are often listed as exceptions to the traditional “can you count it” rule, often thought of as singular amounts (the work will take less than 5 hours, for example).

File name

Correct. Do not use “filename.”

File system

Correct. Do not use “filesystem.” The system that an operating system or program uses to organize and keep track of files. For example, a hierarchical file system is one that uses directories to organize files into a tree structure. Although the operating system provides its own file management system, you can buy separate file management systems. These systems interact smoothly with the operating system but provide more features, such as improved backup procedures and stricter file protection.

For instance

For example,” instead.

For further/additional/whatever information

Use “For more information”

For this reason

Use “therefore” .

Forward

Correct. Avoid using “forwards.”

Gigabyte (GB)

2 to the 30th power (1,073,741,824) bytes. One gigabyte is equal to 1,024 megabytes. Gigabyte is often abbreviated as G or GB.

Got

Avoid. Use “must” instead.

High-availability

Correct. Do not use “high availability.”

Highly available

Correct. Do not use highly-available.”

Hostname

Correct. Do not use host name.

i.e.

Spell it out: “That is.”

Installer

Avoid. Use “installation program” instead.

It’ s and its

“It’ s” is a contraction for “it is;” use “it is” instead of “it’ s.” Use “its” as a possessive pronoun (for example, “the store is known for its low prices”).

Less

Less is used with singular nouns. For example “View less details” wouldn’ t be correct but “View less detail” works. Use fewer when you have plural nouns (things you can count).

Linux

Correct. Do not use “LINUX” or “linux” unless referring to a command, such as “To start Linux, type linux.” Linux is a registered trademark of Linus Torvalds.

Login

A noun used to refer to the login prompt, such as “At the login prompt, enter your username.”

Log in

A verb used to refer to the act of logging in. Do not use “login,” “loggin,” “logon,” and other variants. For example, “When starting your computer, you are requested to log in…”

Log on

To make a computer system or network recognize you so that you can begin a computer session. Most personal computers have no log-on procedure – you just turn the machine on and begin working. For larger systems and networks, however, you usually need to enter a username and password before the computer system will allow you to execute programs.

Lots of

Use “Several” or something equivalent instead.

Make sure

This means “be careful to remember, attend to, or find out something.” For example, “…make sure that the rhedk group is listed in the output.” Try to use verify or ensure instead.

Manual/man page

Correct. Two words. Do not use “manpage”

MB

- (1) When spelled MB, short for megabyte (1,000,000 or 1,048,576 bytes, depending on the context).
- (2) When spelled Mb, short for megabit.

MBps

Short for megabytes per second, a measure of data transfer speed. Mass storage devices are generally measured in MBps.

MySQL

Common open source database server and client package. Do not use “MYSQL” or “mySQL.”

Need to

Avoid. Use “must” instead.

Read-only

Correct. Use when referring to the access permissions of files or directories.

Real time/real-time

Depends. If used as a noun, it is the actual time during which something takes place. For example, “The computer may partly analyze the data in real time (as it comes in) – R. H. March.” If used as an adjective, “real-time” is appropriate. For example, “XEmacs is a self-documenting, customizable, extensible, real-time display editor.”

Refer to

Use to indicate a reference (within a manual or website) or a cross-reference (to another manual or documentation source).

See

Don’ t use. Use “Refer to” instead.

Since

This is often used to mean “because” , but “since” has connotations of time, so be careful. If you mean “because” , say “because” .

Tells

Use “Instructs” instead.

That/which

“That” introduces a restrictive clause—a clause that must be there for the sentence to make sense. A restrictive clause often defines the noun or phrase preceding it. “Which” introduces a non-restrictive, parenthetical clause—a clause that could be omitted without affecting the meaning of the sentence. For example: The car was travelling at a speed that would endanger lives. The car, which was traveling at a speed that would endanger lives, swerved onto the sidewalk. Use “who” or “whom,” rather than “that” or “which,” when referring to a person.

Then/than

“Then” refers to a time in the past or the next step in a sequence. “Than” is used for comparisons.

Then

is used for time.

*First I stole a panda bear, then
we drank malt liquor together.*

↑
The sequence of
actions indicates time:
first stealing the panda,
and then drinking.

Than

is used for comparison.

*I'm much better at holding my
liquor than a panda bear.*

↑
This is comparing a
panda's drinking ability
with your own, so you
should use “than.”



Third-party

Correct. Do not use “third party” .

Troubleshoot

Correct. Do not use “trouble shoot” or “trouble-shoot.” To isolate the source of a problem and fix it. In the case of computer systems, the term troubleshoot is usually used when the problem is suspected to be hardware -related. If the problem is known to be in software, the term debug is more commonly used.

UK

Correct as is, no periods.

UNIX®

Correct. Do not use “Unix” or “unix.” UNIX® is a registered trademark of The Open Group.

Unset

Don’ t use. Use Clear.

US

Correct as is, no periods.

User

When referring to the reader, use “you” instead of “user.” For example, “The user must…” is incorrect. Use “You must…” instead. If referring to more than one user, calling the collection “users” is acceptable, such as “Other users may wish to access your database.”

Username

Correct. Do not use “user name.”

View

When using as a reference (“View the documentation available online.”), do not use View. Use “Refer to” instead.

Within

Don't use to refer to a file that exists in a directory. Use "In" .

World Wide Web

Correct. Capitalize each word. Abbreviate as "WWW" or "Web."

Webpage

Correct. Do not use "web page" or "Web page."

Web server

Correct. Do not use "webserver" . For example, "The Apache HTTP Server is the default Web server..."

Website

Correct. Do not use "web site" or "Web site." For example, "The Ansible website contains ..."

Who/whom

Use the pronoun "who" as a subject. Use the pronoun "whom" as a direct object, an indirect object, or the object of a preposition. For example: Who owns this? To whom does this belong?

Will

Do not use future tense unless it is absolutely necessary. For instance, do not use the sentence, "The next section will describe the process in more detail." Instead, use the sentence, "The next section describes the process in more detail."

Wish

Use "need" instead of "desire" and "wish." Use "want" when the reader's actions are optional (that is, they may not "need" something but may still "want" something).

x86

Correct. Do not capitalize the "x."

x86_64

Do not use. Do not use “Hammer” . Always use “AMD64 and Intel® EM64T” when referring to this architecture.

You

Correct. Do not use “I,” “he,” or “she.”

You may

Try to avoid using this. For example, “you may” can be eliminated from this sentence “You may double-click on the desktop…”

Writing documentation so search can find it

One of the keys to writing good documentation is to make it findable. Readers use a combination of internal site search and external search engines such as Google or duckduckgo.

To ensure Ansible documentation is findable, you should:

1. Use headings that clearly reflect what you are documenting.
2. Use numbered lists for procedures or high-level steps where possible.
3. Avoid linking to github blobs where possible.

Using clear headings in documentation

We all use simple English when we want to find something. For example, the title of this page could have been any one of the following:

- Search optimization
- Findable documentation
- Writing for findability

What we are really trying to describe is - how do I write documentation so search engines can find my content? That simple phrase is what drove the title of this section. When you are creating your headings for documentation, spend some time to think about what you would type in a search box to find it, or more importantly, how someone less familiar with Ansible would try to find that information. Your heading should be the answer to that question.

One word of caution - you do want to limit the size of your headings. A full heading such as *How do I write documentation so search engines can find my content?* is too long. Search engines would truncate anything over 50 - 60 characters. Long headings would also wrap on smaller devices such as a smart phone.

Using numbered lists for zero position snippets

Google can optimize the search results by adding a [feature snippet](#) at the top of the search results. This snippet provides a small window into the documentation on that first search result that adds more detail than the rest of the search results, and can occasionally answer the reader's questions right there, or at least verify that the linked page is what the reader is looking for.

Google returns the feature snippet in the form of numbered steps. Where possible, you should add a numbered list near the top of your documentation page, where appropriate. The steps can be the exact procedure a reader would follow, or could be a high level introduction to the documentation topic, such as the numbered list at the top of this page.

Problems with github blobs on search results

Search engines do not typically return github blobs in search results, at least not in higher ranked positions. While it is possible and sometimes necessary to link to github blobs from documentation, the better approach would be to copy that information into an .rst page in Ansible documentation.

Other search hints

While it may not be possible to adapt your documentation to all search optimizations, keep the following in mind as you write your documentation:

- **Search engines don't parse beyond the '#'** in an html page. So for example, all the subheadings on this page are appended to the main page URL. As such, when I search for 'Using number lists for zero position snippets', the search result would be a link to the top of this page, not a link directly to the subheading I searched for. Using [local TOCs](#) helps alleviate this problem as the reader can scan for the header at top of the page and click to the section they are looking for. For critical documentation, consider creating a new page that can be a direct search result page.
- **Make your first few sentences clearly describe your page topic.** Search engines return not just the URL, but a short description of the information at the URL. For Ansible documentation, we do not have description metadata embedded on each page. Instead, the search engines return the first couple of sentences (140 characters) on the page. That makes your first sentence or two very important to the reader who is searching for something in Ansible.

Resources

- Follow the style of the *Ansible Documentation*
- Ask for advice on IRC, on the `#ansible-devel` Freenode channel
- Review these online style guides:
 - AP Stylebook
 - Chicago Manual of Style
 - Strunk and White's Elements of Style

参见:

Contributing to the Ansible Documentation How to contribute to the Ansible documentation

Testing the documentation locally How to build the Ansible documentation

[#ansible-docs](https://irc.freenode.net) IRC chat channel

Committers Guidelines

These are the guidelines for people with commit privileges on the Ansible GitHub repository. Committers are essentially acting as members of the Ansible Core team, although not necessarily as employees of Ansible and Red Hat. Please read the guidelines before you commit.

These guidelines apply to everyone. At the same time, this ISN'T a process document. So just use good judgment. You've been given commit access because we trust your judgment.

That said, use the trust wisely.

If you abuse the trust and break components and builds, and so on, the trust level falls and you may be asked not to commit or you may lose your commit privileges.

Features, high-level design, and roadmap

As a core team member, you are an integral part of the team that develops the *roadmap*. Please be engaged, and push for the features and fixes that you want to see. Also keep in mind that Red Hat, as a company, will commit to certain features, fixes, APIs, and so on. for various releases. Red Hat, the company, and the Ansible team must get these committed features (and so on.) completed and released as scheduled. Obligations to users, the community, and customers must come first. Because of these commitments, a feature you want to develop yourself may not get into a release if it impacts a lot of other parts within Ansible.

Any other new features and changes to high level design should go through the proposal process (TBD), to ensure the community and core team have had a chance to review the idea and approve it. The core team has sole responsibility for merging new features based on proposals.

Our workflow on GitHub

As a committer, you may already know this, but our workflow forms a lot of our team policies. Please ensure you're aware of the following workflow steps:

- Fork the repository upon which you want to do some work to your own personal repository
- Work on the specific branch upon which you need to commit
- Create a Pull Request back to the Ansible repository and tag the people you would like to review; assign someone as the primary “owner” of your request
- Adjust code as necessary based on the Comments provided
- Ask someone on the Core Team to do a final review and merge

Addendum to workflow for committers:

The Core Team is aware that this can be a difficult process at times. Sometimes, the team breaks the rules by making direct commits or merging their own PRs. This section is a set of guidelines. If you're changing a comma in a doc, or making a very minor change, you can use your best judgement. This is another trust thing. The process is critical for any major change, but for little things or getting something done quickly, use your best judgement and make sure people on the team are aware of your work.

Roles on Core

- Core committers: Fine to do PRs for most things, but we should have a timebox. Hanging PRs may merge on the judgement of these devs.
- *Module maintainers*: Module maintainers own specific modules and have indirect commit access through the current module PR mechanisms.

General rules

Individuals with direct commit access to ansible/ansible are entrusted with powers that allow them to do a broad variety of things—probably more than we can write down. Rather than rules, treat these as general *guidelines*, individuals with this power are expected to use their best judgement.

- Don't
 - Commit directly.
 - Merge your own PRs. Someone else should have a chance to review and approve the PR merge. If you are a Core Committer, you have a small amount of leeway here for very minor changes.
 - Forget about alternate environments. Consider the alternatives—yes, people have bad environments, but they are the ones who need us the most.

- Drag your community team members down. Always discuss the technical merits, but you should never address the person’s limitations (you can later go for beers and call them idiots, but not in IRC/GitHub/and so on.).
- Forget about the maintenance burden. Some things are really cool to have, but they might not be worth shoehorning in if the maintenance burden is too great.
- Break playbooks. Always keep backwards compatibility in mind.
- Forget to keep it simple. Complexity breeds all kinds of problems.
- Do
 - Squash, avoid merges whenever possible, use GitHub’s squash commits or cherry pick if needed (bisect thanks you).
 - Be active. Committers who have no activity on the project (through merges, triage, commits, and so on.) will have their permissions suspended.
 - Consider backwards compatibility (goes back to “don’t break existing playbooks”).
 - Write tests. PRs with tests are looked at with more priority than PRs without tests that should have them included. While not all changes require tests, be sure to add them for bug fixes or functionality changes.
 - Discuss with other committers, specially when you are unsure of something.
 - Document! If your PR is a new feature or a change to behavior, make sure you’ve updated all associated documentation or have notified the right people to do so. It also helps to add the version of Core against which this documentation is compatible (to avoid confusion with stable versus devel docs, for backwards compatibility, and so on.).
 - Consider scope, sometimes a fix can be generalized
 - Keep it simple, then things are maintainable, debuggable and intelligible.

Committers are expected to continue to follow the same community and contribution guidelines followed by the rest of the Ansible community.

People

Individuals who’ve been asked to become a part of this group have generally been contributing in significant ways to the Ansible community for some time. Should they agree, they are requested to add their names and GitHub IDs to this file, in the section below, through a pull request. Doing so indicates that these individuals agree to act in the ways that their fellow committers trust that they will act.

Name	GitHub ID	IRC Nick	Other
James Cammarata	jimi-c	jimi	

下页继续

表 1 – 续上页

Name	GitHub ID	IRC Nick	Other
Brian Coca	bcoca	bcoca	
Matt Davis	nitzmahone	nitzmahone	
Toshio Kuratomi	abadger	abadger1999	
Jason McKerr	mckerrj	newtMcKerr	
Robyn Bergeron	robynbergeron	rbergeron	
Greg DeKoenigsberg	gregdek	gregdek	
Monty Taylor	emonty	mordred	
Matt Martz	sivel	sivel	
Nate Case	qalthos	Qalthos	
James Tanner	jctanner	jtanner	
Peter Sprygada	privateip	privateip	
Abhijit Menon-Sen	amenonsen	crab	
Michael Scherer	mscherer	misc	
René Moser	resmo	resmo	
David Shrewsbury	Shrews	Shrews	
Sandra Wills	docschick	docschick	
Graham Mainwaring	ghjm		
Chris Houseknecht	chouseknecht		
Trond Hindenes	trondhindenes		
Jon Hawkesworth	jhawkesworth	jhawkesworth	
Will Thames	willthames	willthames	
Adrian Likins	alikins	alikins	
Dag Wieers	dagwieers	dagwieers	dag@wieers.com
Tim Rupp	caphrim007	caphrim007	
Sloane Hertel	s-hertel	shertel	
Sam Doran	samdoran	samdoran	
Matt Clay	mattclay	mattclay	
Martin Krizek	mkrizek	mkrizek	
Ganesh Nalawade	ganeshrn	ganeshrn	
Trishna Guha	trishnaguha	trishnag	
Andrew Gaffney	agaffney	agaffney	
Jordan Borean	jborean93	jborean93	
Abhijeet Kasurde	Akasurde	akasurde	
Adam Miller	maxamillion	maxamillion	
Sviatoslav Sydorenko	webknjaz	webknjaz	
Alicia Cozine	acozine	acozine	
Sandra McCann	samccann	samccann	

下页继续

表 1 – 续上页

Name	GitHub ID	IRC Nick	Other
Felix Fontein	felixfontein	felixfontein	felix@fontein.de

Module Maintainer Guidelines

Topics

- *Module Maintainer Guidelines*
 - *Maintainer responsibilities*
 - *Pull requests, issues, and workflow*
 - * *Pull requests*
 - * *Issues*
 - * *PR workflow*
 - * *Maintainers (BOTMETA.yml)*
 - *Adding and removing maintainers*
 - *Tools and other Resources*

Thank you for being a maintainer of part of Ansible’ s codebase. This guide provides module maintainers an overview of their responsibilities, resources for additional information, and links to helpful tools.

In addition to the information below, module maintainers should be familiar with:

- *General Ansible community development practices*
- Documentation on *module development*

Maintainer responsibilities

When you contribute a new module to the [ansible/ansible](#) repository, you become the maintainer for that module once it has been merged. Maintainership empowers you with the authority to accept, reject, or request revisions to pull requests on your module – but as they say, “with great power comes great responsibility.”

Maintainers of Ansible modules are expected to provide feedback, responses, or actions on pull requests or issues to the module(s) they maintain in a reasonably timely manner.

It is also recommended that you occasionally revisit the [contribution guidelines](#), as they are continually refined. Occasionally, you may be requested to update your module to move it closer to the general accepted standard requirements. We hope for this to be infrequent, and will always be a request with a fair amount of lead time (ie: not by tomorrow!).

Finally, following the [ansible-devel](#) mailing list can be a great way to participate in the broader Ansible community, and a place where you can influence the overall direction, quality, and goals of Ansible and its modules. If you're not on this relatively low-volume list, please join us here: <https://groups.google.com/forum/#!forum/ansible-devel>

The Ansible community hopes that you will find that maintaining your module is as rewarding for you as having the module is for the wider community.

Pull requests, issues, and workflow

Pull requests

Module pull requests are located in the [main Ansible repository](#).

Because of the high volume of pull requests, notification of PRs to specific modules are routed by an automated bot to the appropriate maintainer for handling. It is recommended that you set an appropriate notification process to receive notifications which mention your GitHub ID.

Issues

Issues for modules, including bug reports, documentation bug reports, and feature requests, are tracked in the [ansible repository](#).

Issues for modules are routed to their maintainers by an automated process. This process is still being refined, and currently depends upon the issue creator to provide adequate details (specifically, providing the proper module name) in order to route it correctly. If you are a maintainer of a specific module, it is recommended that you periodically search module issues for issues which mention your module's name (or some variation on that name), as well as setting an appropriate notification process for receiving notification of mentions of your GitHub ID.

PR workflow

Automated routing of pull requests is handled by a tool called [Ansibot](#).

Being moderately familiar with how the workflow behind the bot operates can be helpful to you, and – should things go awry – your feedback can be helpful to the folks that continually help Ansibullbot to evolve.

A detailed explanation of the PR workflow can be seen in the *[The Ansible Development Cycle](#)*.

Maintainers (BOTMETA.yml)

The full list of maintainers is located in [BOTMETA.yml](#).

Adding and removing maintainers

Communities change over time, and no one maintains a module forever. If you'd like to propose an additional maintainer for your module, please submit a PR to `BOTMETA.yml` with the GitHub username of the new maintainer.

If you'd like to step down as a maintainer, please submit a PR to the `BOTMETA.yml` removing your GitHub ID from the module in question. If that would leave the module with no maintainers, put “ansible” as the maintainer. This will indicate that the module is temporarily without a maintainer, and the Ansible community team will search for a new maintainer.

Tools and other Resources

- PRs in flight, organized by directory
- Ansibullbot
- *The Ansible Development Cycle*

Release Manager Guidelines

Topics

- *Release Manager Guidelines*
 - *Pre-releases: what and why*
 - * *Beta releases*
 - * *Release candidates*
 - *Ansible release process*

The release manager's purpose is to ensure a smooth release. To achieve that goal, they need to coordinate between:

- Developers with commit privileges on the [Ansible GitHub repository](#)
- Contributors without commit privileges
- The community
- Ansible documentation team
- Ansible Tower team

Pre-releases: what and why

Pre-releases exist to draw testers. They give people who don't feel comfortable running from source control a means to get an early version of the code to test and give us feedback. To ensure we get good feedback about a release, we need to make sure all major changes in a release are put into a pre-release. Testers must be given time to test those changes before the final release. Ideally we want there to be sufficient time between pre-releases for people to install and test one version for a span of time. Then they can spend more time using the new code than installing the latest version.

The right length of time for a tester is probably around two weeks. However, for our three-to-four month development cycle to work, we compress this down to one week; any less runs the risk of people spending more time installing the code instead of running it. However, if there's a time crunch (with a release date that cannot slip), it is better to release with new changes than to hold back those changes to give people time to test between. People cannot test what is not released, so we have to get those tarballs out there even if people feel they have to install more frequently.

Beta releases

In a beta release, we know there are still bugs. We will continue to accept fixes for these. Although we review these fixes, sometimes they can be invasive or potentially destabilize other areas of the code.

During the beta, we will no longer accept feature submissions.

Release candidates

In a release candidate, we've fixed all known blockers. Any remaining bugfixes are ones that we are willing to leave out of the release. At this point we need user testing to determine if there are any other blocker bugs lurking.

Blocker bugs generally are those that cause significant problems for users. Regressions are more likely to be considered blockers because they will break present users' usage of Ansible.

The Release Manager will cherry-pick fixes for new release blockers. The release manager will also choose whether to accept bugfixes for isolated areas of the code or defer those to the next minor release. By themselves, non-blocker bugs will not trigger a new release; they will only make it into the next major release if blocker bugs require that a new release be made.

The last RC should be as close to the final as possible. The following things may be changed:

- Version numbers are changed automatically and will differ as the pre-release tags are removed from the versions.
- Tests and docs/docsite/ can differ if really needed as they do not break runtime. However, the release manager may still reject them as they have the potential to cause breakage that will be visible during the release process.

注解: We want to specifically emphasize that code (in `bin/`, `lib/ansible/`, and `setup.py`) must be the same unless there are extraordinary extenuating circumstances. If there are extenuating circumstances, the Release Manager is responsible for notifying groups (like the Tower Team) which would want to test the code.

Ansible release process

The release process is kept in a [separate document](#) so that it can be easily updated during a release. If you need access to edit this, please ask one of the current release managers to add you.

GitHub Admins

Topics

- *GitHub Admins*
 - *Adding and removing committers*
 - *Changing branch permissions for releases*

GitHub Admins have more permissions on GitHub than normal contributors or even committers. There are a few responsibilities that come with that increased power.

Adding and removing committers

The Ansible Team will periodically review who is actively contributing to Ansible to grant or revoke contributors' ability to commit on their own. GitHub Admins are the people who have the power to actually manage the GitHub permissions.

Changing branch permissions for releases

When we make releases we make people go through a [Release Manager Guidelines](#) to push commits to that branch. The GitHub admins are responsible for setting the branch so only the Release Manager can commit to the branch when the release process reaches that stage and later opening the branch once the release has been made. The Release manager will let the GitHub Admin know when this needs to be done.

参见:

The [GitHub Admin Process Docs](#) for instructions on how to change branch permissions.

1.5 Developer Guide

Welcome to the Ansible Developer Guide!

Who should use this guide?

If you want to extend Ansible by using a custom module or plugin locally, creating a module or plugin, adding functionality to an existing module, or expanding test coverage, this guide is for you. We've included detailed information for developers on how to test and document modules, as well as the prerequisites for getting your module or plugin accepted into the main Ansible repository.

Find the task that best describes what you want to do:

- I'm looking for a way to address a use case:
 - I want to *add a custom plugin or module locally*.
 - I want to figure out if *developing a module is the right approach* for my use case.
 - I want to *develop a collection*.
- I've read the info above, and I'm sure I want to develop a module:
 - What do I need to know before I start coding?
 - I want to *set up my Python development environment*.
 - I want to *get started writing a module*.
 - **I want to write a specific kind of module:**
 - * a *network module*
 - * a *Windows module*.
 - * an *Amazon module*.
 - * an *OpenStack module*.
 - * an *oVirt/RHV module*.
 - * a *VMware module*.
 - I want to *write a series of related modules* that integrate Ansible with a new product (for example, a database, cloud provider, network platform, etc.).
- I want to refine my code:
 - I want to *debug my module code*.
 - I want to *add tests*.
 - I want to *document my module*.
 - I want to *document my set of modules for a network platform*.
 - I want to follow *conventions and tips for clean, usable module code*.

- I want to *make sure my code runs on Python 2 and Python 3*.
- I want to work on other development projects:
 - I want to *write a plugin*.
 - I want to *connect Ansible to a new source of inventory*.
 - I want to *deprecate an outdated module*.
- I want to contribute back to the Ansible project:
 - I want to *understand how to contribute to Ansible*.
 - I want to *contribute my module or plugin*.
 - I want to *understand the license agreement* for contributions to Ansible.

If you prefer to read the entire guide, here's a list of the pages in order.

1.5.1 Adding modules and plugins locally

- *Modules and plugins: what's the difference?*
- *Adding a module locally*
- *Adding a plugin locally*

The easiest, quickest, and most popular way to extend Ansible is to copy or write a module or a plugin for local use. You can store local modules and plugins on your Ansible control node for use within your team or organization. You can also share a local plugin or module by embedding it in a role and publishing it on Ansible Galaxy. If you've been using roles off Galaxy, you may have been using local modules and plugins without even realizing it. If you're using a local module or plugin that already exists, this page is all you need.

Extending Ansible with local modules and plugins offers lots of shortcuts:

- You can copy other people's modules and plugins.
- If you're writing a new module, you can choose any programming language you like.
- You don't have to clone the main Ansible repo.
- You don't have to open a pull request.
- You don't have to add tests (though we recommend that you do!).

To save a local module or plugin so Ansible can find and use it, drop the module or plugin in the correct “magic” directory. For local modules, use the name of the file as the module name: for example, if the module file is `~/.ansible/plugins/modules/local_users.py`, use `local_users` as the module name.

Modules and plugins: what's the difference?

If you're looking to add local functionality to Ansible, you may be wondering whether you need a module or a plugin. Here's a quick overview of the differences:

- Modules are reusable, standalone scripts that can be used by the Ansible API, the **ansible** command, or the **ansible-playbook** command. Modules provide a defined interface, accepting arguments and returning information to Ansible by printing a JSON string to stdout before exiting. Modules execute on the target system (usually that means on a remote system) in separate processes.
- *Plugins* augment Ansible's core functionality and execute on the control node within the `/usr/bin/ansible` process. Plugins offer options and extensions for the core features of Ansible - transforming data, logging output, connecting to inventory, and more.

Adding a module locally

Ansible automatically loads all executable files found in certain directories as modules, so you can create or add a local module in any of these locations:

- any directory added to the `ANSIBLE_LIBRARY` environment variable (`$ANSIBLE_LIBRARY` takes a colon-separated list like `$PATH`)
- `~/.ansible/plugins/modules/`
- `/usr/share/ansible/plugins/modules/`

Once you save your module file in one of these locations, Ansible will load it and you can use it in any local task, playbook, or role.

To confirm that `my_custom_module` is available:

- type `ansible-doc -t module my_custom_module`. You should see the documentation for that module.

To use a local module only in certain playbooks:

- store it in a sub-directory called `library` in the directory that contains the playbook(s)

To use a local module only in a single role:

- store it in a sub-directory called `library` within that role

Adding a plugin locally

Ansible loads plugins automatically too, loading each type of plugin separately from a directory named for the type of plugin. Here's the full list of plugin directory names:

- `action_plugins*`
- `cache_plugins`

- `callback_plugins`
- `connection_plugins`
- `filter_plugins*`
- `inventory_plugins`
- `lookup_plugins`
- `shell_plugins`
- `strategy_plugins`
- `test_plugins*`
- `vars_plugins`

To load your local plugins automatically, create or add them in any of these locations:

- any directory added to the relevant `ANSIBLE_plugin_type_PLUGINS` environment variable (these variables, such as `$ANSIBLE_INVENTORY_PLUGINS` and `$ANSIBLE_VARS_PLUGINS` take colon-separated lists like `$PATH`)
- the directory named for the correct `plugin_type` within `~/.ansible/plugins/` - for example, `~/.ansible/plugins/callback`
- the directory named for the correct `plugin_type` within `/usr/share/ansible/plugins/` - for example, `/usr/share/ansible/plugins/action`

Once your plugin file is in one of these locations, Ansible will load it and you can use it in a any local module, task, playbook, or role. Alternatively, you can edit your `ansible.cfg` file to add directories that contain local plugins - see `ansible_configuration_settings` for details.

To confirm that `plugins/plugin_type/my_custom_plugin` is available:

- type `ansible-doc -t <plugin_type> my_custom_lookup_plugin`. For example, `ansible-doc -t lookup my_custom_lookup_plugin`. You should see the documentation for that plugin. This works for all plugin types except the ones marked with `*` in the list above - see `ansible-doc` for more details.

To use your local plugin only in certain playbooks:

- store it in a sub-directory for the correct `plugin_type` (for example, `callback_plugins` or `inventory_plugins`) in the directory that contains the playbook(s)

To use your local plugin only in a single role:

- store it in a sub-directory for the correct `plugin_type` (for example, `cache_plugins` or `strategy_plugins`) within that role

When shipped as part of a role, the plugin will be available as soon as the role is called in the play.

1.5.2 Should you develop a module?

Developing Ansible modules is easy, but often it isn't necessary. Before you start writing a new module, ask:

1. Does a similar module already exist?

An existing module may cover the functionality you want. Ansible Core includes thousands of modules. Search our list of existing modules to see if there's a module that does what you need.

2. Does a Pull Request already exist?

An existing Pull Request may cover the functionality you want. If someone else has already started developing a similar module, you can review and test it. There are a few ways to find open module Pull Requests:

- [GitHub new module PRs](#)
- [All updates to modules](#)
- [New module PRs listed by directory search for `lib/ansible/modules/`](#)

If you find an existing PR that looks like it addresses your needs, please provide feedback on the PR. Community feedback speeds up the review and merge process.

3. Should you use or develop an action plugin instead?

An action plugin may be the best way to get the functionality you want. Action plugins run on the control node instead of on the managed node, and their functionality is available to all modules. For more information about developing plugins, read the [developing plugins page](#).

4. Should you use a role instead?

A combination of existing modules may cover the functionality you want. You can write a role for this type of use case. Check out the [roles documentation](#).

5. Should you write multiple modules instead of one module?

The functionality you want may be too large for a single module. If you want to connect Ansible to a new cloud provider, database, or network platform, you may need to [develop a related group of modules](#).

- Modules should have a concise and well defined functionality. Basically, follow the UNIX philosophy of doing one thing well.
- Modules should not require that a user know all the underlying options of an API/tool to be used. For instance, if the legal values for a required module parameter cannot be documented, that's a sign that the module would be rejected.
- Modules should typically encompass much of the logic for interacting with a resource. A lightweight wrapper around an API that does not contain much logic would likely cause users to offload too much logic into a playbook, and for this reason the module would be rejected. Instead try creating multiple modules for interacting with smaller individual pieces of the API.

If your use case isn't covered by an existing module, an open PR, an action plugin, or a role, and you don't need to create multiple modules, then you're ready to start developing a new module. Choose from the topics below for next steps:

- I want to *get started on a new module*.
- I want to review *tips and conventions for developing good modules*.
- I want to *write a Windows module*.
- I want *an overview of Ansible's architecture*.
- I want to *document my module*.
- I want to *contribute my module back to Ansible Core*.
- I want to *add unit and integration tests to my module*.
- I want to *add Python 3 support to my module*.
- I want to *write multiple modules*.

参见:

all_modules Learn about available modules

GitHub modules directory Browse module source code

Mailing List Development mailing list

irc.freenode.net #ansible IRC chat channel

1.5.3 Ansible module development: getting started

A module is a reusable, standalone script that Ansible runs on your behalf, either locally or remotely. Modules interact with your local machine, an API, or a remote system to perform specific tasks like changing a database password or spinning up a cloud instance. Each module can be used by the Ansible API, or by the **ansible** or **ansible-playbook** programs. A module provides a defined interface, accepting arguments and returning information to Ansible by printing a JSON string to stdout before exiting. Ansible ships with thousands of modules, and you can easily write your own. If you're writing a module for local use, you can choose any programming language and follow your own rules. This tutorial illustrates how to get started developing an Ansible module in Python.

Topics

- *Environment setup*
 - *Prerequisites via apt (Ubuntu)*
 - *Common environment setup*

- *Starting a new module*
- *Exercising your module code*
 - *Exercising module code locally*
 - *Exercising module code in a playbook*
- *Testing basics*
 - *Sanity tests*
 - *Unit tests*
- *Contributing back to Ansible*
- *Communication and development support*
- *Credit*

Environment setup

Prerequisites via apt (Ubuntu)

Due to dependencies (for example ansible -> paramiko -> pynacl -> libffi):

```
sudo apt update
sudo apt install build-essential libssl-dev libffi-dev python-dev
```

Common environment setup

1. Clone the Ansible repository: `$ git clone https://github.com/ansible/ansible.git`
2. Change directory into the repository root dir: `$ cd ansible`
3. Create a virtual environment: `$ python3 -m venv venv` (or for Python 2 `$ virtualenv venv`. Note, this requires you to install the virtualenv package: `$ pip install virtualenv`)
4. Activate the virtual environment: `$. venv/bin/activate`
5. Install development requirements: `$ pip install -r requirements.txt`
6. Run the environment setup script for each new dev shell process: `$. hacking/env-setup`

注解: After the initial setup above, every time you are ready to start developing Ansible you should be able to just run the following from the root of the Ansible repo: `$. venv/bin/activate && . hacking/env-setup`

Starting a new module

To create a new module:

1. Navigate to the correct directory for your new module: `$ cd lib/ansible/modules/cloud/azure/`
2. Create your new module file: `$ touch my_test.py`
3. Paste the content below into your new module file. It includes the *required Ansible format and documentation* and some example code.
4. Modify and extend the code to do what you want your new module to do. See the *programming tips* and *Python 3 compatibility* pages for pointers on writing clean, concise module code.

```
#!/usr/bin/python

# Copyright: (c) 2018, Terry Jones <terry.jones@example.org>
# GNU General Public License v3.0+ (see COPYING or https://www.gnu.org/licenses/gpl-3.0.
# →txt)

ANSIBLE_METADATA = {
    'metadata_version': '1.1',
    'status': ['preview'],
    'supported_by': 'community'
}

DOCUMENTATION = '''
---
module: my_test

short_description: This is my test module

version_added: "2.4"

description:
    - "This is my longer description explaining my test module"

options:
    name:
        description:
            - This is the message to send to the test module
        required: true
    new:
        description:
```

(下页继续)

(续上页)

```
        - Control to demo if the result of this module is changed or not
        required: false

extends_documentation_fragment:
    - azure

author:
    - Your Name (@yourhandle)
'''

EXAMPLES = '''
# Pass in a message
- name: Test with a message
  my_test:
    name: hello world

# pass in a message and have changed true
- name: Test with a message and changed output
  my_test:
    name: hello world
    new: true

# fail the module
- name: Test failure of the module
  my_test:
    name: fail me
'''

RETURN = '''
original_message:
    description: The original name param that was passed in
    type: str
    returned: always
message:
    description: The output message that the test module generates
    type: str
    returned: always
'''

from ansible.module_utils.basic import AnsibleModule
```

(下页继续)

(续上页)

```
def run_module():
    # define available arguments/parameters a user can pass to the module
    module_args = dict(
        name=dict(type='str', required=True),
        new=dict(type='bool', required=False, default=False)
    )

    # seed the result dict in the object
    # we primarily care about changed and state
    # changed is if this module effectively modified the target
    # state will include any data that you want your module to pass back
    # for consumption, for example, in a subsequent task
    result = dict(
        changed=False,
        original_message='',
        message=''
    )

    # the AnsibleModule object will be our abstraction working with Ansible
    # this includes instantiation, a couple of common attr would be the
    # args/params passed to the execution, as well as if the module
    # supports check mode
    module = AnsibleModule(
        argument_spec=module_args,
        supports_check_mode=True
    )

    # if the user is working with this module in only check mode we do not
    # want to make any changes to the environment, just return the current
    # state with no modifications
    if module.check_mode:
        module.exit_json(**result)

    # manipulate or modify the state as needed (this is going to be the
    # part where your module will do what it needs to do)
    result['original_message'] = module.params['name']
    result['message'] = 'goodbye'

    # use whatever logic you need to determine whether or not this module
```

(下页继续)

```
# made any modifications to your target
if module.params['new']:
    result['changed'] = True

# during the execution of the module, if there is an exception or a
# conditional state that effectively causes a failure, run
# AnsibleModule.fail_json() to pass in the message and the result
if module.params['name'] == 'fail me':
    module.fail_json(msg='You requested this to fail', **result)

# in the event of a successful module execution, you will want to
# simple AnsibleModule.exit_json(), passing the key/value results
module.exit_json(**result)

def main():
    run_module()

if __name__ == '__main__':
    main()
```

Exercising your module code

Once you've modified the sample code above to do what you want, you can try out your module. Our *debugging tips* will help if you run into bugs as you exercise your module code.

Exercising module code locally

If your module does not need to target a remote host, you can quickly and easily exercise your code locally like this:

- Create an arguments file, a basic JSON config file that passes parameters to your module so you can run it. Name the arguments file `/tmp/args.json` and add the following content:

```
{
  "ANSIBLE_MODULE_ARGS": {
    "name": "hello",
    "new": true
  }
}
```

- If you are using a virtual environment (highly recommended for development) activate it: `$. venv/bin/activate`
- Setup the environment for development: `$. hacking/env-setup`
- Run your test module locally and directly: `$ python -m ansible.modules.cloud.azure.my_test /tmp/args.json`

This should return output like this:

```
{
  "changed": true,
  "state": {
    "original_message": "hello",
    "new_message": "goodbye"
  },
  "invocation": {
    "module_args": {
      "name": "hello",
      "new": true
    }
  }
}
```

Exercising module code in a playbook

The next step in testing your new module is to consume it with an Ansible playbook.

- Create a playbook in any directory: `$ touch testmod.yml`
- Add the following to the new playbook file:

```
- name: test my new module
  hosts: localhost
  tasks:
    - name: run the new module
      my_test:
        name: 'hello'
        new: true
        register: testout
    - name: dump test output
      debug:
        msg: '{{ testout }}
```

- Run the playbook and analyze the output: `$ ansible-playbook ./testmod.yml`

Testing basics

These two examples will get you started with testing your module code. Please review our [testing](#) section for more detailed information, including instructions for testing module documentation, adding integration tests, and more.

Sanity tests

You can run through Ansible's sanity checks in a container:

```
$ ansible-test sanity -v --docker --python 2.7 MODULE_NAME
```

Note that this example requires Docker to be installed and running. If you'd rather not use a container for this, you can choose to use `--venv` instead of `--docker`.

Unit tests

You can add unit tests for your module in `./test/units/modules`. You must first setup your testing environment. In this example, we're using Python 3.5.

- Install the requirements (outside of your virtual environment): `$ pip3 install -r ./test/lib/ansible_test/_data/requirements/units.txt`
- To run all tests do the following: `$ ansible-test units --python 3.5` (you must run `. hacking/env-setup` prior to this)

注解: Ansible uses pytest for unit testing.

To run pytest against a single test module, you can do the following (provide the path to the test module appropriately):

```
$ pytest -r a --cov=. --cov-report=html --fulltrace --color yes test/units/modules/.../
test/my_test.py
```

Contributing back to Ansible

If you would like to contribute to the main Ansible repository by adding a new feature or fixing a bug, [create a fork](#) of the Ansible repository and develop against a new feature branch using the `devel` branch as a starting point. When you have a good working code change, you can submit a pull request to the Ansible repository by selecting your feature branch as a source and the Ansible `devel` branch as a target.

If you want to contribute your module back to the upstream Ansible repo, review our [submission checklist](#), [programming tips](#), and [strategy for maintaining Python 2 and Python 3 compatibility](#), as well as information about [testing](#) before you open a pull request. The *Community Guide* covers how to open a pull request and what happens next.

Communication and development support

Join the IRC channel `#ansible-devel` on freenode for discussions surrounding Ansible development.

For questions and discussions pertaining to using the Ansible product, use the `#ansible` channel.

Credit

Thank you to Thomas Stringer ([@trstringer](#)) for contributing source material for this topic.

1.5.4 Contributing your module to Ansible

If you want to contribute a module to Ansible, you must meet our objective and subjective requirements. Please read the details below, and also review our *[tips for module development](#)*.

Modules accepted into the [main project repo](#) ship with every Ansible installation. However, contributing to the main project isn't the only way to distribute a module - you can embed modules in roles on Galaxy or simply share copies of your module code for *[local use](#)*.

Contributing to Ansible: objective requirements

To contribute a module to Ansible, you must:

- write your module in either Python or Powershell for Windows
- use the `AnsibleModule` common code
- support Python 2.6 and Python 3.5 - if your module cannot support Python 2.6, explain the required minimum Python version and rationale in the requirements section in `DOCUMENTATION`
- use proper *[Python 3 syntax](#)*
- follow [PEP 8](#) Python style conventions - see `testing__pep8` for more information
- license your module under the GPL license (GPLv3 or later)
- understand the *[license agreement](#)*, which applies to all contributions
- conform to Ansible's *[formatting and documentation](#)* standards
- include comprehensive *[tests](#)* for your module
- minimize module dependencies
- support *[check_mode](#)* if possible
- ensure your code is readable
- if a module is named `<something>_facts`, it should be because its main purpose is returning `ansible_facts`. Do not name modules that do not do this with `_facts`. Only use `ansible_facts` for information that is specific to the host machine, for example network interfaces and their configuration, which operating system and which programs are installed.
- Modules that query/return general information (and not `ansible_facts`) should be named `_info`. General information is non-host specific information, for example information on online/cloud services (you can access different accounts for the same online service from the same host), or information on VMs and containers accessible from the machine.

Please make sure your module meets these requirements before you submit your PR/proposal. If you have questions, reach out via [Ansible's IRC chat channel](#) or the [Ansible development mailing list](#).

Contributing to Ansible: subjective requirements

If your module meets our objective requirements, we'll review your code to see if we think it's clear, concise, secure, and maintainable. We'll consider whether your module provides a good user experience, helpful error messages, reasonable defaults, and more. This process is subjective, and we can't list exact standards for acceptance. For the best chance of getting your module accepted into the Ansible repo, follow our *tips for module development*.

Other checklists

- *Tips for module development.*
- *Amazon development checklist.*
- *Windows development checklist.*

1.5.5 Conventions, tips, and pitfalls

Topics

- *Scoping your module(s)*
- *Designing module interfaces*
- *General guidelines & tips*
- *Functions and Methods*
- *Python tips*
- *Importing and using shared code*
- *Handling module failures*
- *Handling exceptions (bugs) gracefully*
- *Creating correct and informative module output*
- *Following Ansible conventions*
- *Module Security*

As you design and develop modules, follow these basic conventions and tips for clean, usable code:

Scoping your module(s)

Especially if you want to contribute your module(s) back to Ansible Core, make sure each module includes enough logic and functionality, but not too much. If you're finding these guidelines tricky, consider *whether you really need to write a module* at all.

- Each module should have a concise and well-defined functionality. Basically, follow the UNIX philosophy of doing one thing well.
- Do not add `get`, `list` or `info` state options to an existing module - create a new `_info` or `_facts` module.
- Modules should not require that a user know all the underlying options of an API/tool to be used. For instance, if the legal values for a required module parameter cannot be documented, the module does not belong in Ansible Core.
- Modules should encompass much of the logic for interacting with a resource. A lightweight wrapper around a complex API forces users to offload too much logic into their playbooks. If you want to connect Ansible to a complex API, *create multiple modules* that interact with smaller individual pieces of the API.
- Avoid creating a module that does the work of other modules; this leads to code duplication and divergence, and makes things less uniform, unpredictable and harder to maintain. Modules should be the building blocks. If you are asking 'how can I have a module execute other modules' ...you want to write a role.

Designing module interfaces

- If your module is addressing an object, the parameter for that object should be called `name` whenever possible, or accept `name` as an alias.
- Modules accepting boolean status should accept `yes`, `no`, `true`, `false`, or anything else a user may likely throw at them. The AnsibleModule common code supports this with `type='bool'`.
- Avoid `action/command`, they are imperative and not declarative, there are other ways to express the same thing.

General guidelines & tips

- Each module should be self-contained in one file, so it can be auto-transferred by Ansible.
- Module name MUST use underscores instead of hyphens or spaces as a word separator. Using hyphens and spaces will prevent Ansible from importing your module.
- Always use the `hacking/test-module.py` script when developing modules - it will warn you about common pitfalls.

- If you have a local module that returns facts specific to your installations, a good name for this module is `site_facts`.
- Eliminate or minimize dependencies. If your module has dependencies, document them at the top of the module file and raise JSON error messages when dependency import fails.
- Don't write to files directly; use a temporary file and then use the `atomic_move` function from `ansible.module_utils.basic` to move the updated temporary file into place. This prevents data corruption and ensures that the correct context for the file is kept.
- Avoid creating caches. Ansible is designed without a central server or authority, so you cannot guarantee it will not run with different permissions, options or locations. If you need a central authority, have it on top of Ansible (for example, using bastion/cm/ci server or tower); do not try to build it into modules.
- If you package your module(s) in an RPM, install the modules on the control machine in `/usr/share/ansible`. Packaging modules in RPMs is optional.

Functions and Methods

- Each function should be concise and should describe a meaningful amount of work.
- “Don't repeat yourself” is generally a good philosophy.
- Function names should use underscores: `my_function_name`.
- Each function's name should describes what it does.
- Each function should have a docstring.
- If your code is too nested, that's usually a sign the loop body could benefit from being a function. Parts of our existing code are not the best examples of this at times.

Python tips

- When fetching URLs, use `fetch_url` or `open_url` from `ansible.module_utils.urls`. Do not use `urllib2`, which does not natively verify TLS certificates and so is insecure for https.
- Include a `main` function that wraps the normal execution.
- Call your `main` function from a conditional so you can import it into unit tests - for example:

```
if __name__ == '__main__':  
    main()
```

Importing and using shared code

- Use shared code whenever possible - don't reinvent the wheel. Ansible offers the `AnsibleModule` common Python code, plus *utilities* for many common use cases and patterns. You can also create documentation fragments for docs that apply to multiple modules.
- Import `ansible.module_utils` code in the same place as you import other libraries.
- Do NOT use wildcards (*) for importing other python modules; instead, list the function(s) you are importing (for example, `from some.other_python_module.basic import otherFunction`).
- Import custom packages in `try/except`, capture any import errors, and handle them with `fail_json()` in `main()`. For example:

```
import traceback

from ansible.basic import missing_required_lib

LIB_IMP_ERR = None
try:
    import foo
    HAS_LIB = True
except:
    HAS_LIB = False
    LIB_IMP_ERR = traceback.format_exc()
```

Then in `main()`, just after the `argspec`, do

```
if not HAS_LIB:
    module.fail_json(msg=missing_required_lib("foo"),
                    exception=LIB_IMP_ERR)
```

And document the dependency in the `requirements` section of your module's *DOCUMENTATION block*.

Handling module failures

When your module fails, help users understand what went wrong. If you are using the `AnsibleModule` common Python code, the `failed` element will be included for you automatically when you call `fail_json`. For polite module failure behavior:

- Include a key of `failed` along with a string explanation in `msg`. If you don't do this, Ansible will use standard return codes: 0=success and non-zero=failure.
- Don't raise a traceback (`stacktrace`). Ansible can deal with `stacktraces` and automatically converts anything unparseable into a failed result, but raising a `stacktrace` on module failure is not user-friendly.

- Do not use `sys.exit()`. Use `fail_json()` from the module object.

Handling exceptions (bugs) gracefully

- Validate upfront—fail fast and return useful and clear error messages.
- Use defensive programming—use a simple design for your module, handle errors gracefully, and avoid direct stacktraces.
- Fail predictably—if we must fail, do it in a way that is the most expected. Either mimic the underlying tool or the general way the system works.
- Give out a useful message on what you were doing and add exception messages to that.
- Avoid catchall exceptions, they are not very useful unless the underlying API gives very good error messages pertaining the attempted action.

Creating correct and informative module output

Modules must output valid JSON only. Follow these guidelines for creating correct, useful module output:

- Make your top-level return type a hash (dictionary).
- Nest complex return values within the top-level hash.
- Incorporate any lists or simple scalar values within the top-level return hash.
- Do not send module output to standard error, because the system will merge standard out with standard error and prevent the JSON from parsing.
- Capture standard error and return it as a variable in the JSON on standard out. This is how the `command` module is implemented.
- Never do `print("some status message")` in a module, because it will not produce valid JSON output.
- Always return useful data, even when there is no change.
- Be consistent about returns (some modules are too random), unless it is detrimental to the state/action.
- Make returns reusable—most of the time you don't want to read it, but you do want to process it and re-purpose it.
- Return diff if in diff mode. This is not required for all modules, as it won't make sense for certain ones, but please include it when applicable.
- Enable your return values to be serialized as JSON with Python's standard [JSON encoder and decoder](#) library. Basic python types (strings, int, dicts, lists, etc) are serializable.
- Do not return an object via `exit_json()`. Instead, convert the fields you need from the object into the fields of a dictionary and return the dictionary.

- Results from many hosts will be aggregated at once, so your module should return only relevant output. Returning the entire contents of a log file is generally bad form.

If a module returns stderr or otherwise fails to produce valid JSON, the actual output will still be shown in Ansible, but the command will not succeed.

Following Ansible conventions

Ansible conventions offer a predictable user interface across all modules, playbooks, and roles. To follow Ansible conventions in your module development:

- Use consistent names across modules (yes, we have many legacy deviations - don't make the problem worse!).
- Use consistent parameters (arguments) within your module(s).
- Do not use 'message' or 'syslog_facility' as a parameter name, as this is used internally by Ansible.
- Normalize parameters with other modules - if Ansible and the API your module connects to use different names for the same parameter, add aliases to your parameters so the user can choose which names to use in tasks and playbooks.
- Return facts from `*_facts` modules in the `ansible_facts` field of the *result dictionary* so other modules can access them.
- Implement `check_mode` in all `*_info` and `*_facts` modules. Playbooks which conditionalize based on fact information will only conditionalize correctly in `check_mode` if the facts are returned in `check_mode`. Usually you can add `supports_check_mode=True` when instantiating `AnsibleModule`.
- Use module-specific environment variables. For example, if you use the helpers in `module_utils.api` for basic authentication with `module_utils.urls.fetch_url()` and you fall back on environment variables for default values, use a module-specific environment variable like `API_<MODULENAME>_USERNAME` to avoid conflict between modules.
- Keep module options simple and focused - if you're loading a lot of choices/states on an existing option, consider adding a new, simple option instead.
- Keep options small when possible. Passing a large data structure to an option might save us a few tasks, but it adds a complex requirement that we cannot easily validate before passing on to the module.
- If you want to pass complex data to an option, write an expert module that allows this, along with several smaller modules that provide a more 'atomic' operation against the underlying APIs and services. Complex operations require complex data. Let the user choose whether to reflect that complexity in tasks and plays or in vars files.
- Implement declarative operations (not CRUD) so the user can ignore existing state and focus on final state. For example, use `started/stopped`, `present/absent`.

- Strive for a consistent final state (aka idempotency). If running your module twice in a row against the same system would result in two different states, see if you can redesign or rewrite to achieve consistent final state. If you can't, document the behavior and the reasons for it.
- Provide consistent return values within the standard Ansible return structure, even if NA/None are used for keys normally returned under other options.
- Follow additional guidelines that apply to families of modules if applicable. For example, AWS modules should follow the [Amazon development checklist](#).

Module Security

- Avoid passing user input from the shell.
- Always check return codes.
- You must always use `module.run_command`, not `subprocess` or `Popen` or `os.system`.
- Avoid using the shell unless absolutely necessary.
- If you must use the shell, you must pass `use_unsafe_shell=True` to `module.run_command`.
- If any variables in your module can come from user input with `use_unsafe_shell=True`, you must wrap them with `pipes.quote(x)`.
- When fetching URLs, use `fetch_url` or `open_url` from `ansible.module_utils.urls`. Do not use `urllib2`, which does not natively verify TLS certificates and so is insecure for https.

1.5.6 Ansible and Python 3

Topics

- *Minimum version of Python 3.x and Python 2.x*
- *Developing Ansible code that supports Python 2 and Python 3*
 - *Understanding strings in Python 2 and Python 3*
 - * *Controller string strategy: the Unicode Sandwich*
 - *Unicode Sandwich common borders: places to convert bytes to text in controller code*
 - * *Reading and writing to files*
 - * *Filesystem interaction*
 - * *Interacting with other programs*
 - * *Module string strategy: Native String*
 - * *Module_utils string strategy: hybrid*

- *Tips, tricks, and idioms for Python 2/Python 3 compatibility*
 - * *Use forward-compatibility boilerplate*
 - * *Prefix byte strings with `b_`*
 - * *Import Ansible's bundled Python `six` library*
 - * *Handle exceptions with `as`*
 - * *Update octal numbers*
- *String formatting for controller code*
 - * *Use `str.format()` for Python 2.6 compatibility*
 - * *Use percent format with byte strings*
- *Testing modules on Python 3*

Ansible maintains a single code base that runs on both Python 2 and Python 3 because we want Ansible to be able to manage a wide variety of machines. Contributors to Ansible should be aware of the tips in this document so that they can write code that will run on the same versions of Python as the rest of Ansible.

To ensure that your code runs on Python 3 as well as on Python 2, learn the tips and tricks and idioms described here. Most of these considerations apply to all three types of Ansible code:

1. controller-side code - code that runs on the machine where you invoke `/usr/bin/ansible`
2. modules - the code which Ansible transmits to and invokes on the managed machine.
3. shared `module_utils` code - the common code that's used by modules to perform tasks and sometimes used by controller-side code as well

However, the three types of code do not use the same string strategy. If you're developing a module or some `module_utils` code, be sure to read the section on string strategy carefully.

Minimum version of Python 3.x and Python 2.x

On the controller we support Python 3.5 or greater and Python 2.7 or greater. Module-side, we support Python 3.5 or greater and Python 2.6 or greater.

Python 3.5 was chosen as a minimum because it is the earliest Python 3 version adopted as the default Python by a Long Term Support (LTS) Linux distribution (in this case, Ubuntu-16.04). Previous LTS Linux distributions shipped with a Python 2 version which users can rely upon instead of the Python 3 version.

For Python 2, the default is for modules to run on at least Python 2.6. This allows users with older distributions that are stuck on Python 2.6 to manage their machines. Modules are allowed to drop support for Python 2.6 when one of their dependent libraries requires a higher version of Python. This is not an invitation to add unnecessary dependent libraries in order to force your module to be usable only with

a newer version of Python; instead it is an acknowledgment that some libraries (for instance, boto3 and docker-py) will only function with a newer version of Python.

注解: Python 2.4 Module-side Support:

Support for Python 2.4 and Python 2.5 was dropped in Ansible-2.4. RHEL-5 (and its rebuilds like CentOS-5) were supported until April of 2017. Ansible-2.3 was released in April of 2017 and was the last Ansible release to support Python 2.4 on the module-side.

Developing Ansible code that supports Python 2 and Python 3

The best place to start learning about writing code that supports both Python 2 and Python 3 is [Lennart Regebro's book: Porting to Python 3](#). The book describes several strategies for porting to Python 3. The one we're using is [to support Python 2 and Python 3 from a single code base](#)

Understanding strings in Python 2 and Python 3

Python 2 and Python 3 handle strings differently, so when you write code that supports Python 3 you must decide what string model to use. Strings can be an array of bytes (like in C) or they can be an array of text. Text is what we think of as letters, digits, numbers, other printable symbols, and a small number of unprintable “symbols” (control codes).

In Python 2, the two types for these (`str` for bytes and `unicode` for text) are often used interchangeably. When dealing only with ASCII characters, the strings can be combined, compared, and converted from one type to another automatically. When non-ASCII characters are introduced, Python 2 starts throwing exceptions due to not knowing what encoding the non-ASCII characters should be in.

Python 3 changes this behavior by making the separation between bytes (`bytes`) and text (`str`) more strict. Python 3 will throw an exception when trying to combine and compare the two types. The programmer has to explicitly convert from one type to the other to mix values from each.

In Python 3 it's immediately apparent to the programmer when code is mixing the byte and text types inappropriately, whereas in Python 2, code that mixes those types may work until a user causes an exception by entering non-ASCII input. Python 3 forces programmers to proactively define a strategy for working with strings in their program so that they don't mix text and byte strings unintentionally.

Ansible uses different strategies for working with strings in controller-side code, in `:ref: modules <module_string_strategy>`, and in `module_utils` code.

Controller string strategy: the Unicode Sandwich

In controller-side code we use a strategy known as the Unicode Sandwich (named after Python 2's `unicode` text type). For Unicode Sandwich we know that at the border of our code and the outside world (for example,

file and network IO, environment variables, and some library calls) we are going to receive bytes. We need to transform these bytes into text and use that throughout the internal portions of our code. When we have to send those strings back out to the outside world we first convert the text back into bytes. To visualize this, imagine a ‘sandwich’ consisting of a top and bottom layer of bytes, a layer of conversion between, and all text type in the center.

Unicode Sandwich common borders: places to convert bytes to text in controller code

This is a partial list of places where we have to convert to and from bytes when using the Unicode Sandwich string strategy. It’s not exhaustive but it gives you an idea of where to watch for problems.

Reading and writing to files

In Python 2, reading from files yields bytes. In Python 3, it can yield text. To make code that’s portable to both we don’t make use of Python 3’s ability to yield text but instead do the conversion explicitly ourselves. For example:

```
from ansible.module_utils._text import to_text

with open('filename-with-utf8-data.txt', 'rb') as my_file:
    b_data = my_file.read()
    try:
        data = to_text(b_data, errors='surrogate_or_strict')
    except UnicodeError:
        # Handle the exception gracefully -- usually by displaying a good
        # user-centric error message that can be traced back to this piece
        # of code.
        pass
```

注解: Much of Ansible assumes that all encoded text is UTF-8. At some point, if there is demand for other encodings we may change that, but for now it is safe to assume that bytes are UTF-8.

Writing to files is the opposite process:

```
from ansible.module_utils._text import to_bytes

with open('filename.txt', 'wb') as my_file:
    my_file.write(to_bytes(some_text_string))
```

Note that we don’t have to catch `UnicodeError` here because we’re transforming to UTF-8 and all text strings in Python can be transformed back to UTF-8.

Filesystem interaction

Dealing with filenames often involves dropping back to bytes because on UNIX-like systems filenames are bytes. On Python 2, if we pass a text string to these functions, the text string will be converted to a byte string inside of the function and a traceback will occur if non-ASCII characters are present. In Python 3, a traceback will only occur if the text string can't be decoded in the current locale, but it's still good to be explicit and have code which works on both versions:

```
import os.path

from ansible.module_utils._text import to_bytes

filename = u'/var/tmp/くらとみ.txt'
f = open(to_bytes(filename), 'wb')
mtime = os.path.getmtime(to_bytes(filename))
b_filename = os.path.expandvars(to_bytes(filename))
if os.path.exists(to_bytes(filename)):
    pass
```

When you are only manipulating a filename as a string without talking to the filesystem (or a C library which talks to the filesystem) you can often get away without converting to bytes:

```
import os.path

os.path.join(u'/var/tmp/café', u'くらとみ')
os.path.split(u'/var/tmp/café/くらとみ')
```

On the other hand, if the code needs to manipulate the filename and also talk to the filesystem, it can be more convenient to transform to bytes right away and manipulate in bytes.

警告: Make sure all variables passed to a function are the same type. If you're working with something like `os.path.join()` which takes multiple strings and uses them in combination, you need to make sure that all the types are the same (either all bytes or all text). Mixing bytes and text will cause tracebacks.

Interacting with other programs

Interacting with other programs goes through the operating system and C libraries and operates on things that the UNIX kernel defines. These interfaces are all byte-oriented so the Python interface is byte oriented as well. On both Python 2 and Python 3, byte strings should be given to Python's subprocess library and byte strings should be expected back from it.

One of the main places in Ansible's controller code that we interact with other programs is the connection plugins' `exec_command` methods. These methods transform any text strings they receive in the command (and arguments to the command) to execute into bytes and return stdout and stderr as byte strings. Higher level functions (like action plugins' `_low_level_execute_command`) transform the output into text strings.

Module string strategy: Native String

In modules we use a strategy known as Native Strings. This makes things easier on the community members who maintain so many of Ansible's modules, by not breaking backwards compatibility by mandating that all strings inside of modules are text and converting between text and bytes at the borders.

Native strings refer to the type that Python uses when you specify a bare string literal:

```
"This is a native string"
```

In Python 2, these are byte strings. In Python 3 these are text strings. Modules should be coded to expect bytes on Python 2 and text on Python 3.

Module_utils string strategy: hybrid

In `module_utils` code we use a hybrid string strategy. Although Ansible's `module_utils` code is largely like module code, some pieces of it are used by the controller as well. So it needs to be compatible with modules and with the controller's assumptions, particularly the string strategy. The `module_utils` code attempts to accept native strings as input to its functions and emit native strings as their output.

In `module_utils` code:

- Functions **must** accept string parameters as either text strings or byte strings.
- Functions may return either the same type of string as they were given or the native string type for the Python version they are run on.
- Functions that return strings **must** document whether they return strings of the same type as they were given or native strings.

Module-utils functions are therefore often very defensive in nature. They convert their string parameters into text (using `ansible.module_utils._text.to_text`) at the beginning of the function, do their work, and then convert the return values into the native string type (using `ansible.module_utils._text.to_native`) or back to the string type that their parameters received.

Tips, tricks, and idioms for Python 2/Python 3 compatibility

Use forward-compatibility boilerplate

Use the following boilerplate code at the top of all python files to make certain constructs act the same way on Python 2 and Python 3:

```
# Make coding more python3-ish
from __future__ import (absolute_import, division, print_function)
__metaclass__ = type
```

`__metaclass__ = type` makes all classes defined in the file into new-style classes without explicitly inheriting from `object`.

The `__future__` imports do the following:

absolute_import Makes imports look in `sys.path` for the modules being imported, skipping the directory in which the module doing the importing lives. If the code wants to use the directory in which the module doing the importing, there's a new dot notation to do so.

division Makes division of integers always return a float. If you need to find the quotient use `x // y` instead of `x / y`.

print_function Changes `print` from a keyword into a function.

参见:

- [PEP 0328: Absolute Imports](#)
- [PEP 0238: Division](#)
- [PEP 3105: Print function](#)

Prefix byte strings with b_

Since mixing text and bytes types leads to tracebacks we want to be clear about what variables hold text and what variables hold bytes. We do this by prefixing any variable holding bytes with `b_`. For instance:

```
filename = u'/var/tmp/café.txt'
b_filename = to_bytes(filename)
with open(b_filename) as f:
    data = f.read()
```

We do not prefix the text strings instead because we only operate on byte strings at the borders, so there are fewer variables that need bytes than text.

Import Ansible's bundled Python `six` library

The third-party Python `six` library exists to help projects create code that runs on both Python 2 and Python 3. Ansible includes a version of the library in `module_utils` so that other modules can use it without requiring that it is installed on the remote system. To make use of it, import it like this:

```
from ansible.module_utils import six
```

注解: Ansible can also use a system copy of `six`

Ansible will use a system copy of `six` if the system copy is a later version than the one Ansible bundles.

Handle exceptions with `as`

In order for code to function on Python 2.6+ and Python 3, use the new exception-catching syntax which uses the `as` keyword:

```
try:
    a = 2/0
except ValueError as e:
    module.fail_json(msg="Tried to divide by zero: %s" % e)
```

Do **not** use the following syntax as it will fail on every version of Python 3:

```
try:
    a = 2/0
except ValueError, e:
    module.fail_json(msg="Tried to divide by zero: %s" % e)
```

Update octal numbers

In Python 2.x, octal literals could be specified as `0755`. In Python 3, octals must be specified as `0o755`.

String formatting for controller code

Use `str.format()` for Python 2.6 compatibility

Starting in Python 2.6, strings gained a method called `format()` to put strings together. However, one commonly used feature of `format()` wasn't added until Python 2.7, so you need to remember not to use it in Ansible code:

```
# Does not work in Python 2.6!
new_string = "Dear {}, Welcome to {}".format(username, location)

# Use this instead
new_string = "Dear {0}, Welcome to {1}".format(username, location)
```

Both of the format strings above map positional arguments of the `format()` method into the string. However, the first version doesn't work in Python 2.6. Always remember to put numbers into the placeholders so the code is compatible with Python 2.6.

参见:

Python documentation on [format strings](#)

Use percent format with byte strings

In Python 3.x, byte strings do not have a `format()` method. However, it does have support for the older, percent-formatting.

```
b_command_line = b'ansible-playbook --become-user %s -K %s' % (user, playbook_file)
```

注解: Percent formatting added in Python 3.5

Percent formatting of byte strings was added back into Python 3 in 3.5. This isn't a problem for us because Python 3.5 is our minimum version. However, if you happen to be testing Ansible code with Python 3.4 or earlier, you will find that the byte string formatting here won't work. Upgrade to Python 3.5 to test.

参见:

Python documentation on [percent formatting](#)

Testing modules on Python 3

Ansible modules are slightly harder to code to support Python 3 than normal code from other projects. A lot of mocking has to go into unit testing an Ansible module, so it's harder to test that your changes have fixed everything or to make sure that later commits haven't regressed the Python 3 support. Review our [testing](#) pages for more information.

1.5.7 Debugging modules

Debugging (local)

To break into a module running on `localhost` and step through with the debugger:

- Set a breakpoint in the module: `import pdb; pdb.set_trace()`
- Run the module on the local machine: `$ python -m pdb ./my_new_test_module.py ./args.json`

Example

```
echo '{ "msg" : "hello" }' | python ./my_new_test_module.py
```

Debugging (remote)

To debug a module running on a remote target (i.e. not `localhost`):

1. On your controller machine (running Ansible) set `ANSIBLE_KEEP_REMOTE_FILES=1` to tell Ansible to retain the modules it sends to the remote machine instead of removing them after you playbook runs.
2. Run your playbook targeting the remote machine and specify `-vvvv` (verbose) to display the remote location Ansible is using for the modules (among many other things).
3. Take note of the directory Ansible used to store modules on the remote host. This directory is usually under the home directory of your `ansible_user`, in the form `~/.ansible/tmp/ansible-tmp-....`
4. SSH into the remote target after the playbook runs.
5. Navigate to the directory you noted in step 3.
6. Extract the module you want to debug from the zipped file that Ansible sent to the remote host: `$ python AnsiballZ_my_test_module.py explode`. Ansible will expand the module into `./debug-dir`. You can optionally run the zipped file by specifying `python AnsiballZ_my_test_module.py`.
7. Navigate to the debug directory: `$ cd debug-dir`.
8. Modify or set a breakpoint in `__main__.py`.
9. Ensure that the unzipped module is executable: `$ chmod 755 __main__.py`.
10. Run the unzipped module directly, passing the `args` file that contains the params that were originally passed: `$./__main__.py args`. This approach is good for reproducing behavior as well as modifying the parameters for debugging.

Debugging AnsibleModule-based modules

小技巧: If you're using the `hacking/test-module.py` script then most of this is taken care of for you. If you need to do some debugging of the module on the remote machine that the module will actually run on

or when the module is used in a playbook then you may need to use this information instead of relying on `test-module.py`.

Starting with Ansible 2.1, AnsibleModule-based modules are put together as a zip file consisting of the module file and the various python module boilerplate inside of a wrapper script instead of as a single file with all of the code concatenated together. Without some help, this can be harder to debug as the file needs to be extracted from the wrapper in order to see what's actually going on in the module. Luckily the wrapper script provides some helper methods to do just that.

If you are using Ansible with the `ANSIBLE_KEEP_REMOTE_FILES` environment variables to keep the remote module file, here's a sample of how your debugging session will start:

```
$ ANSIBLE_KEEP_REMOTE_FILES=1 ansible localhost -m ping -a 'data=debugging_session' -vvv
<127.0.0.1> ESTABLISH LOCAL CONNECTION FOR USER: badger
<127.0.0.1> EXEC /bin/sh -c '( umask 77 && mkdir -p "` echo $HOME/.ansible/tmp/ansible-
↳ tmp-1461434734.35-235318071810595 `" && echo "` echo $HOME/.ansible/tmp/ansible-tmp-
↳ 1461434734.35-235318071810595 `" )'
<127.0.0.1> PUT /var/tmp/tmpjdbJ1w TO /home/badger/.ansible/tmp/ansible-tmp-1461434734.
↳ 35-235318071810595/ping
<127.0.0.1> EXEC /bin/sh -c 'LANG=en_US.UTF-8 LC_ALL=en_US.UTF-8 LC_MESSAGES=en_US.UTF-8
↳ /usr/bin/python /home/badger/.ansible/tmp/ansible-tmp-1461434734.35-235318071810595/
↳ ping'
localhost | SUCCESS => {
    "changed": false,
    "invocation": {
        "module_args": {
            "data": "debugging_session"
        },
        "module_name": "ping"
    },
    "ping": "debugging_session"
}
```

Setting `ANSIBLE_KEEP_REMOTE_FILES` to 1 tells Ansible to keep the remote module files instead of deleting them after the module finishes executing. Giving Ansible the `-vvv` option makes Ansible more verbose. That way it prints the file name of the temporary module file for you to see.

If you want to examine the wrapper file you can. It will show a small python script with a large, base64 encoded string. The string contains the module that is going to be executed. Run the wrapper's `explode` command to turn the string into some python files that you can work with:

```
$ python /home/badger/.ansible/tmp/ansible-tmp-1461434734.35-235318071810595/ping explode
```

(下页继续)

(续上页)

```
Module expanded into:
/home/badger/.ansible/tmp/ansible-tmp-1461434734.35-235318071810595/debug_dir
```

When you look into the `debug_dir` you'll see a directory structure like this:

```
ansible_module_ping.py
args
ansible
  __init__.py
  module_utils
    basic.py
    __init__.py
```

- `ansible_module_ping.py` is the code for the module itself. The name is based on the name of the module with a prefix so that we don't clash with any other python module names. You can modify this code to see what effect it would have on your module.
- The `args` file contains a JSON string. The string is a dictionary containing the module arguments and other variables that Ansible passes into the module to change its behaviour. If you want to modify the parameters that are passed to the module, this is the file to do it in.
- The `ansible` directory contains code from `ansible.module_utils` that is used by the module. Ansible includes files for any `ansible.module_utils` imports in the module but not any files from any other module. So if your module uses `ansible.module_utils.url` Ansible will include it for you, but if your module includes `requests` then you'll have to make sure that the python `requests` library is installed on the system before running the module. You can modify files in this directory if you suspect that the module is having a problem in some of this boilerplate code rather than in the module code you have written.

Once you edit the code or arguments in the exploded tree you need some way to run it. There's a separate wrapper subcommand for this:

```
$ python /home/badger/.ansible/tmp/ansible-tmp-1461434734.35-235318071810595/ping execute
{"invocation": {"module_args": {"data": "debugging_session"}}, "changed": false, "ping":
↪ "debugging_session"}
```

This subcommand takes care of setting the `PYTHONPATH` to use the exploded `debug_dir/ansible/module_utils` directory and invoking the script using the arguments in the `args` file. You can continue to run it like this until you understand the problem. Then you can copy it back into your real module file and test that the real module works via `ansible` or `ansible-playbook`.

注解: The wrapper provides one more subcommand, `excommunicate`. This subcommand is very similar to `execute` in that it invokes the exploded module on the arguments in the `args`. The way it does this is

different, however. `excommunicate` imports the `main` function from the module and then calls that. This makes `excommunicate` execute the module in the wrapper's process. This may be useful for running the module under some graphical debuggers but it is very different from the way the module is executed by Ansible itself. Some modules may not work with `excommunicate` or may behave differently than when used with Ansible normally. Those are not bugs in the module; they're limitations of `excommunicate`. Use at your own risk.

1.5.8 Module format and documentation

If you want to contribute your module to Ansible, you must write your module in Python and follow the standard format described below. (Unless you're writing a Windows module, in which case the [Windows guidelines](#) apply.) In addition to following this format, you should review our [submission checklist](#), [programming tips](#), and [strategy for maintaining Python 2 and Python 3 compatibility](#), as well as information about [testing](#) before you open a pull request.

Every Ansible module written in Python must begin with seven standard sections in a particular order, followed by the code. The sections in order are:

- *Python shebang & UTF-8 coding*
- *Copyright and license*
- *ANSIBLE_METADATA block*
- *DOCUMENTATION block*
- *EXAMPLES block*
- *RETURN block*
- *Python imports*
- *Testing module documentation*

注解: Why don't the imports go first?

Keen Python programmers may notice that contrary to PEP 8's advice we don't put `imports` at the top of the file. This is because the `ANSIBLE_METADATA` through `RETURN` sections are not used by the module code itself; they are essentially extra docstrings for the file. The imports are placed after these special variables for the same reason as PEP 8 puts the imports after the introductory comments and docstrings. This keeps the active parts of the code together and the pieces which are purely informational apart. The decision to exclude E402 is based on readability (which is what PEP 8 is about). Documentation strings in a module are much more similar to module level docstrings, than code, and are never utilized by the module itself. Placing the imports below this documentation and closer to the code, consolidates and groups all related

code in a congruent manner to improve readability, debugging and understanding.

警告: Copy old modules with care!

Some older modules in Ansible Core have `imports` at the bottom of the file, `Copyright` notices with the full GPL prefix, and/or `ANSIBLE_METADATA` fields in the wrong order. These are legacy files that need updating - do not copy them into new modules. Over time we're updating and correcting older modules. Please follow the guidelines on this page!

Python shebang & UTF-8 coding

Every Ansible module must begin with `#!/usr/bin/python` - this “shebang” allows `ansible_python_interpreter` to work. This is immediately followed by `# -*- coding: utf-8 -*-` to clarify that the file is UTF-8 encoded.

Copyright and license

After the shebang and UTF-8 coding, there should be a `copyright` line with the original copyright holder and a license declaration. The license declaration should be **ONLY** one line, not the full GPL prefix.:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

# Copyright: (c) 2018, Terry Jones <terry.jones@example.org>
# GNU General Public License v3.0+ (see COPYING or https://www.gnu.org/licenses/gpl-3.0.
↪txt)
```

Major additions to the module (for instance, rewrites) may add additional copyright lines. Any legal review will include the source control history, so an exhaustive copyright header is not necessary. When adding a second copyright line for a significant feature or rewrite, add the newer line above the older one:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

# Copyright: (c) 2017, [New Contributor(s)]
# Copyright: (c) 2015, [Original Contributor(s)]
# GNU General Public License v3.0+ (see COPYING or https://www.gnu.org/licenses/gpl-3.0.
↪txt)
```

ANSIBLE_METADATA block

After the shebang, the UTF-8 coding, the copyright, and the license, your module file should contain an `ANSIBLE_METADATA` section. This section provides information about the module for use by other tools. For new modules, the following block can be simply added into your module:

```
ANSIBLE_METADATA = {'metadata_version': '1.1',
                    'status': ['preview'],
                    'supported_by': 'community'}
```

警告:

- `metadata_version` is the version of the `ANSIBLE_METADATA` schema, *not* the version of the module.
- Promoting a module's `status` or `supported_by` status should only be done by members of the Ansible Core Team.

Ansible metadata fields

metadata_version An “X.Y”formatted string. X and Y are integers which define the metadata format version. Modules shipped with Ansible are tied to an Ansible release, so we will only ship with a single version of the metadata. We'll increment Y if we add fields or legal values to an existing field. We'll increment X if we remove fields or values or change the type or meaning of a field. Current `metadata_version` is “1.1”

supported_by Who supports the module. Default value is `community`. For information on what the support level values entail, please see [Modules Support](#). Values are:

- core
- network
- certified
- community
- curated (*deprecated value - modules in this category should be core or certified instead*)

status List of strings describing how stable the module is likely to be. See also [The lifecycle of an Ansible module](#). The default value is a single element list [“preview”]. The following strings are valid statuses and have the following meanings:

stableinterface The module's options (the parameters or arguments it accepts) are stable. Every effort will be made not to remove options or change their meaning. **Not** a rating of the module's code quality.

preview The module is in tech preview. It may be unstable, the options may change, or it may require libraries or web services that are themselves subject to incompatible changes.

deprecated The module is deprecated and will be removed in a future release.

removed The module is not present in the release. A stub is kept so that documentation can be built. The documentation helps users port from the removed module to new modules.

DOCUMENTATION block

After the shebang, the UTF-8 coding, the copyright line, the license, and the `ANSIBLE_METADATA` section comes the `DOCUMENTATION` block. Ansible's online module documentation is generated from the `DOCUMENTATION` blocks in each module's source code. The `DOCUMENTATION` block must be valid YAML. You may find it easier to start writing your `DOCUMENTATION` string in an *editor with YAML syntax highlighting* before you include it in your Python file. You can start by copying our [example documentation string](#) into your module file and modifying it. If you run into syntax issues in your YAML, you can validate it on the [YAML Lint](#) website.

Module documentation should briefly and accurately define what each module and option does, and how it

- Descriptions should always start with a capital letter and end with a full stop. Consistency always helps.
- Verify that arguments in doc and module spec dict are identical.
- For password / secret arguments `no_log=True` should be set.
- For arguments that seem to contain sensitive information but **do not** contain secrets, such as “password_length” , set `no_log=False` to disable the warning message.
- If an option is only sometimes required, describe the conditions. For example, “Required when I(state=present).”
- If your module allows `check_mode`, reflect this fact in the documentation.

Each documentation field is described below. Before committing your module documentation, please test it at the command line and as HTML:

- As long as your module file is *available locally*, you can use `ansible-doc -t module my_module_name` to view your module documentation at the command line. Any parsing errors will be obvious - you can view details by adding `-vvv` to the command.
- You should also test the HTML output of your module documentation.

Documentation fields

All fields in the `DOCUMENTATION` block are lower-case. All fields are required unless specified otherwise:

`module`

- The name of the module.
- Must be the same as the filename, without the `.py` extension.

`short_description`

- A short description which is displayed on the `all_modules` page and `ansible-doc -l`.
- The `short_description` is displayed by `ansible-doc -l` without any category grouping, so it needs enough detail to explain the module's purpose without the context of the directory structure in which it lives.
- Unlike `description:`, `short_description` should not have a trailing period/full stop.

`description`

- A detailed description (generally two or more sentences).
- Must be written in full sentences, i.e. with capital letters and periods/full stops.
- Shouldn't mention the module name.
- Make use of multiple entries rather than using one long paragraph.
- Don't quote complete values unless it is required by YAML.

`version_added`

- The version of Ansible when the module was added.
- This is a string, and not a float, i.e. `version_added: '2.1'`

`author`

- Name of the module author in the form `First Last (@GitHubID)`.
- Use a multi-line list if there is more than one author.
- Don't use quotes as it should not be required by YAML.

`deprecated`

- Marks modules that will be removed in future releases. See also *The lifecycle of an Ansible module*.

`options`

- Options are often called *parameters* or *arguments*. Because the documentation field is called *options*, we will use that term.

- If the module has no options (for example, it's a `_facts` module), all you need is one line: `options: {}`.
- If your module has options (in other words, accepts arguments), each option should be documented thoroughly. For each module option, include:

option-name

- Declarative operation (not CRUD), to focus on the final state, for example *online:*, rather than *is_online:*.
- The name of the option should be consistent with the rest of the module, as well as other modules in the same category.
- When in doubt, look for other modules to find option names that are used for the same purpose, we like to offer consistency to our users.

description

- Detailed explanation of what this option does. It should be written in full sentences.
- The first entry is a description of the option itself; subsequent entries detail its use, dependencies, or format of possible values.
- Should not list the possible values (that's what `choices:` is for, though it should explain what the values do if they aren't obvious).
- If an option is only sometimes required, describe the conditions. For example, "Required when I(state=present)."
- Mutually exclusive options must be documented as the final sentence on each of the options.

required

- Only needed if `true`.
- If missing, we assume the option is not required.

default

- If `required` is false/missing, `default` may be specified (assumed 'null' if missing).
- Ensure that the default value in the docs matches the default value in the code.
- The default field must not be listed as part of the description, unless it requires additional information or conditions.
- If the option is a boolean value, you can use any of the boolean values recognized by Ansible: (such as true/false or yes/no). Choose the one

that reads better in the context of the option.

choices

- List of option values.
- Should be absent if empty.

type

- Specifies the data type that option accepts, must match the `argspec`.
- If an argument is `type='bool'`, this field should be set to `type: bool` and no `choices` should be specified.
- If an argument is `type='list'`, `elements` should be specified.

elements

- Specifies the data type for list elements in case `type='list'`.

aliases

- List of optional name aliases.
- Generally not needed.

version__added

- Only needed if this option was extended after initial Ansible release, i.e. this is greater than the top level *version__added* field.
- This is a string, and not a float, i.e. `version__added: '2.3'`.

suboptions

- If this option takes a dict or list of dicts, you can define the structure here.
- See `azure_rm_securitygroup_module`, `azure_rm_azurefirewall_module` and `os_ironic_node_module` for examples.

requirements

- List of requirements (if applicable).
- Include minimum versions.

seealso

- A list of references to other modules, documentation or Internet resources
- A reference can be one of the following formats:

```

seealso:

# Reference by module name
- module: aci_tenant

# Reference by module name, including description
- module: aci_tenant
  description: ACI module to create tenants on a Cisco ACI fabric.

# Reference by rST documentation anchor
- ref: aci_guide
  description: Detailed information on how to manage your ACI_
  ↪ infrastructure using Ansible.

# Reference by Internet resource
- name: APIC Management Information Model reference
  description: Complete reference of the APIC object model.
  link: https://developer.cisco.com/docs/apic-mim-ref/

```

notes

- Details of any important information that doesn't fit in one of the above sections.
- For example, whether `check_mode` is or is not supported.

Linking within module documentation

You can link from your module documentation to other module docs, other resources on docs.ansible.com, and resources elsewhere on the internet. The correct formats for these links are:

- L() for Links with a heading. For example: See L(IOS Platform Options guide,../network/user_guide/platform_ios.html).
- U() for URLs. For example: See U(<https://www.ansible.com/products/tower>) for an overview.
- I() for option names. For example: Required if I(state=present).
- C() for files and option values. For example: If not set the environment variable C(ACME_PASSWORD) will be used.
- M() for module names. For example: See also M(win_copy) or M(win_template).

注解: For modules in a collection, you can only use L() and M() for content within that collection. Use

U() to refer to content in a different collection.

注解:

- To refer a group of modules, use `C(...)`, e.g. Refer to the `C(win_*)` modules.
 - Because it stands out better, using `seealso` is preferred for general references over the use of notes or adding links to the description.
-

Documentation fragments

If you're writing multiple related modules, they may share common documentation, such as authentication details, file mode settings, `notes:` or `seealso:` entries. Rather than duplicate that information in each module's `DOCUMENTATION` block, you can save it once as a `doc_fragment` plugin and use it in each module's documentation. In Ansible, shared documentation fragments are contained in a `ModuleDocFragment` class in `lib/ansible/plugins/doc_fragments/`. To include a documentation fragment, add `extends_documentation_fragment: FRAGMENT_NAME` in your module's documentation.

Modules should only use items from a doc fragment if the module will implement all of the interface documented there in a manner that behaves the same as the existing modules which import that fragment. The goal is that items imported from the doc fragment will behave identically when used in another module that imports the doc fragment.

By default, only the `DOCUMENTATION` property from a doc fragment is inserted into the module documentation. It is possible to define additional properties in the doc fragment in order to import only certain parts of a doc fragment or mix and match as appropriate. If a property is defined in both the doc fragment and the module, the module value overrides the doc fragment.

Here is an example doc fragment named `example_fragment.py`:

```
class ModuleDocFragment(object):
    # Standard documentation
    DOCUMENTATION = r'''
    options:
        # options here
    '''

    # Additional section
    OTHER = r'''
    options:
        # other options here
    '''
```

To insert the contents of OTHER in a module:

```
extends_documentation_fragment: example_fragment.other
```

Or use both :

```
extends_documentation_fragment:
- example_fragment
- example_fragment.other
```

2.8 新版功能.

Since Ansible 2.8, you can have user-supplied doc_fragments by using a `doc_fragments` directory adjacent to play or role, just like any other plugin.

For example, all AWS modules should include:

```
extends_documentation_fragment:
- aws
- ec2
```

Using documentation fragments in collections describes how to incorporate documentation fragments in a collection.

EXAMPLES block

After the shebang, the UTF-8 coding, the copyright line, the license, the `ANSIBLE_METADATA` section, and the `DOCUMENTATION` block comes the `EXAMPLES` block. Here you show users how your module works with real-world examples in multi-line plain-text YAML format. The best examples are ready for the user to copy and paste into a playbook. Review and update your examples with every change to your module.

Per playbook best practices, each example should include a `name:` line:

```
EXAMPLES = r'''
- name: Ensure foo is installed
  modulename:
    name: foo
    state: present
'''
```

The `name:` line should be capitalized and not include a trailing dot.

If your examples use boolean options, use yes/no values. Since the documentation generates boolean values as yes/no, having the examples use these values as well makes the module documentation more consistent.

If your module returns facts that are often needed, an example of how to use them can be helpful.

RETURN block

After the shebang, the UTF-8 coding, the copyright line, the license, the `ANSIBLE_METADATA` section, `DOCUMENTATION` and `EXAMPLES` blocks comes the `RETURN` block. This section documents the information the module returns for use by other modules.

If your module doesn't return anything (apart from the standard returns), this section of your module should read: `RETURN = r''' # '''` Otherwise, for each value returned, provide the following fields. All fields are required unless specified otherwise.

return name Name of the returned field.

description Detailed description of what this value represents. Capitalized and with trailing dot.

returned When this value is returned, such as `always`, `changed` or `success`. This is a string and can contain any human-readable content.

type Data type.

elements If `type='list'`, specifies the data type of the list's elements.

sample One or more examples.

version_added Only needed if this return was extended after initial Ansible release, i.e. this is greater than the top level `version_added` field. This is a string, and not a float, i.e. `version_added: '2.3'`.

contains Optional. To describe nested return values, set `type: complex`, `type: dict`, or `type: list/elements: dict` and repeat the elements above for each sub-field.

Here are two example `RETURN` sections, one with three simple fields and one with a complex nested field:

```
RETURN = r'''
dest:
    description: Destination file/path.
    returned: success
    type: str
    sample: /path/to/file.txt
src:
    description: Source file used for the copy on the target machine.
    returned: changed
    type: str
    sample: /home/httpd/.ansible/tmp/ansible-tmp-1423796390.97-147729857856000/source
md5sum:
    description: MD5 checksum of the file after running copy.
```

(下页继续)

(续上页)

```

    returned: when supported
    type: str
    sample: 2a5aeccc61dc98c4d780b14b330e3282
'''

RETURN = r'''
packages:
    description: Information about package requirements
    returned: success
    type: complex
    contains:
        missing:
            description: Packages that are missing from the system
            returned: success
            type: list
            sample:
                - libmysqlclient-dev
                - libxml2-dev
        badversion:
            description: Packages that are installed but at bad versions.
            returned: success
            type: list
            sample:
                - package: libxml2-dev
                  version: 2.9.4+dfsg1-2
                  constraint: ">= 3.0"
'''

```

Python imports

After the shebang, the UTF-8 coding, the copyright line, the license, and the sections for `ANSIBLE_METADATA`, `DOCUMENTATION`, `EXAMPLES`, and `RETURN`, you can finally add the python imports. All modules must use Python imports in the form:

```
from module_utils.basic import AnsibleModule
```

The use of “wildcard” imports such as `from module_utils.basic import *` is no longer allowed.

Testing module documentation

To test Ansible documentation locally please follow instruction.

1.5.9 Windows module development walkthrough

In this section, we will walk through developing, testing, and debugging an Ansible Windows module.

Because Windows modules are written in Powershell and need to be run on a Windows host, this guide differs from the usual development walkthrough guide.

What's covered in this section:

- *Windows environment setup*
- *Create a Windows server in a VM*
- *Create an Ansible inventory*
- *Provisioning the environment*
- *Windows new module development*
- *Windows module utilities*
- *Windows playbook module testing*
- *Windows debugging*
- *Windows unit testing*
- *Windows integration testing*
- *Windows communication and development support*

Windows environment setup

Unlike Python module development which can be run on the host that runs Ansible, Windows modules need to be written and tested for Windows hosts. While evaluation editions of Windows can be downloaded from Microsoft, these images are usually not ready to be used by Ansible without further modification. The easiest way to set up a Windows host so that it is ready to be used by Ansible is to set up a virtual machine using Vagrant. Vagrant can be used to download existing OS images called *boxes* that are then deployed to a hypervisor like VirtualBox. These boxes can either be created and stored offline or they can be downloaded from a central repository called Vagrant Cloud.

This guide will use the Vagrant boxes created by the [packer-windoze](#) repository which have also been uploaded to [Vagrant Cloud](#). To find out more info on how these images are created, please go to the GitHub repo and look at the README file.

Before you can get started, the following programs must be installed (please consult the Vagrant and VirtualBox documentation for installation instructions):

- Vagrant
- VirtualBox

Create a Windows server in a VM

To create a single Windows Server 2016 instance, run the following:

```
vagrant init jborean93/WindowsServer2016
vagrant up
```

This will download the Vagrant box from Vagrant Cloud and add it to the local boxes on your host and then start up that instance in VirtualBox. When starting for the first time, the Windows VM will run through the sysprep process and then create a HTTP and HTTPS WinRM listener automatically. Vagrant will finish its process once the listeners are online, after which the VM can be used by Ansible.

Create an Ansible inventory

The following Ansible inventory file can be used to connect to the newly created Windows VM:

```
[windows]
WindowsServer  ansible_host=127.0.0.1

[windows:vars]
ansible_user=vagrant
ansible_password=vagrant
ansible_port=55986
ansible_connection=winrm
ansible_winrm_transport=ntlm
ansible_winrm_server_cert_validation=ignore
```

注解: The port 55986 is automatically forwarded by Vagrant to the Windows host that was created, if this conflicts with an existing local port then Vagrant will automatically use another one at random and display show that in the output.

The OS that is created is based on the image set. The following images can be used:

- jborean93/WindowsServer2008-x86
- jborean93/WindowsServer2008-x64

- [jborean93/WindowsServer2008R2](#)
- [jborean93/WindowsServer2012](#)
- [jborean93/WindowsServer2012R2](#)
- [jborean93/WindowsServer2016](#)

When the host is online, it can be accessible by RDP on 127.0.0.1:3389 but the port may differ depending if there was a conflict. To get rid of the host, run `vagrant destroy --force` and Vagrant will automatically remove the VM and any other files associated with that VM.

While this is useful when testing modules on a single Windows instance, these hosts won't work without modification with domain based modules. The Vagrantfile at [ansible-windows](#) can be used to create a test domain environment to be used in Ansible. This repo contains three files which are used by both Ansible and Vagrant to create multiple Windows hosts in a domain environment. These files are:

- **Vagrantfile:** The Vagrant file that reads the inventory setup of `inventory.yml` and provisions the hosts that are required
- **inventory.yml:** Contains the hosts that are required and other connection information such as IP addresses and forwarded ports
- **main.yml:** Ansible playbook called by Vagrant to provision the domain controller and join the child hosts to the domain

By default, these files will create the following environment:

- A single domain controller running on Windows Server 2016
- Five child hosts for each major Windows Server version joined to that domain
- A domain with the DNS name `domain.local`
- A local administrator account on each host with the username `vagrant` and password `vagrant`
- A domain admin account `vagrant-domain@domain.local` with the password `VagrantPass1`

The domain name and accounts can be modified by changing the variables `domain_*` in the `inventory.yml` file if it is required. The inventory file can also be modified to provision more or less servers by changing the hosts that are defined under the `domain_children` key. The host variable `ansible_host` is the private IP that will be assigned to the VirtualBox host only network adapter while `vagrant_box` is the box that will be used to create the VM.

Provisioning the environment

To provision the environment as is, run the following:

```
git clone https://github.com/jborean93/ansible-windows.git
cd vagrant
vagrant up
```

注解: Vagrant provisions each host sequentially so this can take some time to complete. If any errors occur during the Ansible phase of setting up the domain, run `vagrant provision` to rerun just that step.

Unlike setting up a single Windows instance with Vagrant, these hosts can also be accessed using the IP address directly as well as through the forwarded ports. It is easier to access it over the host only network adapter as the normal protocol ports are used, e.g. RDP is still over 3389. In cases where the host cannot be resolved using the host only network IP, the following protocols can be access over 127.0.0.1 using these forwarded ports:

- RDP: 295xx
- SSH: 296xx
- WinRM HTTP: 297xx
- WinRM HTTPS: 298xx
- SMB: 299xx

Replace xx with the entry number in the inventory file where the domain controller started with 00 and is incremented from there. For example, in the default `inventory.yml` file, WinRM over HTTPS for SERVER2012R2 is forwarded over port 29804 as it's the fourth entry in `domain_children`.

注解: While an SSH server is available on all Windows hosts but Server 2008 (non R2), it is not a support connection for Ansible managing Windows hosts and should not be used with Ansible.

Windows new module development

When creating a new module there are a few things to keep in mind:

- Module code is in Powershell (.ps1) files while the documentation is contained in Python (.py) files of the same name
- Avoid using `Write-Host/Debug/Verbose/Error` in the module and add what needs to be returned to the `$module.Result` variable
- To fail a module, call `$module.FailJson("failure message here")`, an Exception or ErrorRecord can be set to the second argument for a more descriptive error message
- You can pass in the exception or ErrorRecord as a second argument to `FailJson("failure", $_)` to get a more detailed output
- Most new modules require check mode and integration tests before they are merged into the main Ansible codebase

- Avoid using try/catch statements over a large code block, rather use them for individual calls so the error message can be more descriptive
- Try and catch specific exceptions when using try/catch statements
- Avoid using PSCustomObjects unless necessary
- Look for common functions in `./lib/ansible/module_utils/powershell/` and use the code there instead of duplicating work. These can be imported by adding the line `#Requires -Module *` where `*` is the filename to import, and will be automatically included with the module code sent to the Windows target when run via Ansible
- As well as PowerShell module utils, C# module utils are stored in `./lib/ansible/module_utils/csharp/` and are automatically imported in a module execution if the line `#AnsibleRequires -CSharpUtil *` is present
- C# and PowerShell module utils achieve the same goal but C# allows a developer to implement low level tasks, such as calling the Win32 API, and can be faster in some cases
- Ensure the code runs under Powershell v3 and higher on Windows Server 2008 and higher; if higher minimum Powershell or OS versions are required, ensure the documentation reflects this clearly
- Ansible runs modules under strictmode version 2.0. Be sure to test with that enabled by putting `Set-StrictMode -Version 2.0` at the top of your dev script
- Favor native Powershell cmdlets over executable calls if possible
- Use the full cmdlet name instead of aliases, e.g. `Remove-Item` over `rm`
- Use named parameters with cmdlets, e.g. `Remove-Item -Path C:\temp` over `Remove-Item C:\temp`

A very basic powershell module `win_environment` is included below. It demonstrates how to implement check-mode and diff-support, and also shows a warning to the user when a specific condition is met.

A slightly more advanced module is `win_uri` which additionally shows how to use different parameter types (bool, str, int, list, dict, path) and a selection of choices for parameters, how to fail a module and how to handle exceptions.

As part of the new `AnsibleModule` wrapper, the input parameters are defined and validated based on an argument spec. The following options can be set at the root level of the argument spec:

- **mutually_exclusive:** A list of lists, where the inner list contains module options that cannot be set together
- **no_log:** Stops the module from emitting any logs to the Windows Event log
- **options:** A dictionary where the key is the module option and the value is the spec for that option
- **required_by:** A dictionary where the option(s) specified by the value must be set if the option specified by the key is also set
- **required_if:** A list of lists where the inner list contains 3 or 4 elements;

- The first element is the module option to check the value against
- The second element is the value of the option specified by the first element, if matched then the required if check is run
- The third element is a list of required module options when the above is matched
- An optional fourth element is a boolean that states whether all module options in the third elements are required (default: `$false`) or only one (`$true`)
- **required_one_of**: A list of lists, where the inner list contains module options where at least one must be set
- **required_together**: A list of lists, where the inner list contains module options that must be set together
- **supports_check_mode**: Whether the module supports check mode, by default this is `$false`

The actual input options for a module are set within the `options` value as a dictionary. The keys of this dictionary are the module option names while the values are the spec of that module option. Each spec can have the following options set:

- **aliases**: A list of aliases for the module option
- **choices**: A list of valid values for the module option, if `type=list` then each list value is validated against the choices and not the list itself
- **default**: The default value for the module option if not set
- **deprecated_aliases**: A list of hashtables that define aliases that are deprecated and the versions they will be removed in. Each entry must contain the keys `name` and `version`
- **elements**: When `type=list`, this sets the type of each list value, the values are the same as `type`
- **no_log**: Will sanitise the input value before being returned in the `module_invocation` return value
- **removed_in_version**: States when a deprecated module option is to be removed, a warning is displayed to the end user if set
- **required**: Will fail when the module option is not set
- **type**: The type of the module option, if not set then it defaults to `str`. The valid types are;
 - `bool`: A boolean value
 - `dict`: A dictionary value, if the input is a JSON or key=value string then it is converted to dictionary
 - `float`: A float or [Single](#) value
 - `int`: An Int32 value
 - `json`: A string where the value is converted to a JSON string if the input is a dictionary

- **list**: A list of values, **elements=<type>** can convert the individual list value types if set. If **elements=dict** then **options** is defined, the values will be validated against the argument spec. When the input is a string then the string is split by `,` and any whitespace is trimmed
- **path**: A string where values like `%TEMP%` are expanded based on environment values. If the input value starts with `\\?\` then no expansion is run
- **raw**: No conversions occur on the value passed in by Ansible
- **sid**: Will convert Windows security identifier values or Windows account names to a `SecurityIdentifier` value
- **str**: The value is converted to a string

When **type=dict**, or **type=list** and **elements=dict**, the following keys can also be set for that module option:

- **apply_defaults**: The value is based on the **options** spec defaults for that key if `True` and null if `False`. Only valid when the module option is not defined by the user and **type=dict**.
- **mutually_exclusive**: Same as the root level **mutually_exclusive** but validated against the values in the sub dict
- **options**: Same as the root level **options** but contains the valid options for the sub option
- **required_if**: Same as the root level **required_if** but validated against the values in the sub dict
- **required_by**: Same as the root level **required_by** but validated against the values in the sub dict
- **required_together**: Same as the root level **required_together** but validated against the values in the sub dict
- **required_one_of**: Same as the root level **required_one_of** but validated against the values in the sub dict

A module type can also be a delegate function that converts the value to whatever is required by the module option. For example the following snippet shows how to create a custom type that creates a `UInt64` value:

```
$spec = @{
    uint64_type = @{ type = [Func[[Object], [UInt64]]]{ [System.UInt64]::Parse($args[0]) } }
}
$uint64_type = $module.Params.uint64_type
```

When in doubt, look at some of the other core modules and see how things have been implemented there.

Sometimes there are multiple ways that Windows offers to complete a task; this is the order to favor when writing modules:

- Native Powershell cmdlets like `Remove-Item -Path C:\temp -Recurse`
- .NET classes like `[System.IO.Path]::GetRandomFileName()`

- WMI objects through the `New-CimInstance` cmdlet
- COM objects through `New-Object -ComObject` cmdlet
- Calls to native executables like `Secedit.exe`

PowerShell modules support a small subset of the `#Requires` options built into PowerShell as well as some Ansible-specific requirements specified by `#AnsibleRequires`. These statements can be placed at any point in the script, but are most commonly near the top. They are used to make it easier to state the requirements of the module without writing any of the checks. Each `requires` statement must be on its own line, but there can be multiple `requires` statements in one script.

These are the checks that can be used within Ansible modules:

- `#Requires -Module Ansible.ModuleUtils.<module_util>`: Added in Ansible 2.4, specifies a `module_util` to load in for the module execution.
- `#Requires -Version x.y`: Added in Ansible 2.5, specifies the version of PowerShell that is required by the module. The module will fail if this requirement is not met.
- `#AnsibleRequires -OSVersion x.y`: Added in Ansible 2.5, specifies the OS build version that is required by the module and will fail if this requirement is not met. The actual OS version is derived from `[Environment]::OSVersion.Version`.
- `#AnsibleRequires -Become`: Added in Ansible 2.5, forces the exec runner to run the module with `become`, which is primarily used to bypass WinRM restrictions. If `ansible_become_user` is not specified then the `SYSTEM` account is used instead.
- `#AnsibleRequires -CSharpUtil Ansible.<module_util>`: Added in Ansible 2.8, specifies a C# `module_util` to load in for the module execution.

C# module utils can reference other C# utils by adding the line `using Ansible.<module_util>;` to the top of the script with all the other `using` statements.

Windows module utilities

Like Python modules, PowerShell modules also provide a number of module utilities that provide helper functions within PowerShell. These `module_utils` can be imported by adding the following line to a PowerShell module:

```
#Requires -Module Ansible.ModuleUtils.Legacy
```

This will import the `module_util` at `./lib/ansible/module_utils/powershell/Ansible.ModuleUtils.Legacy.psm1` and enable calling all of its functions. As of Ansible 2.8, Windows module utils can also be written in C# and stored at `lib/ansible/module_utils/csharp`. These `module_utils` can be imported by adding the following line to a PowerShell module:

```
#AnsibleRequires -CSharpUtil Ansible.Basic
```

This will import the module_util at `./lib/ansible/module_utils/csharp/Ansible.Basic.cs` and automatically load the types in the executing process. C# module utils can reference each other and be loaded together by adding the following line to the using statements at the top of the util:

```
using Ansible.Become;
```

There are special comments that can be set in a C# file for controlling the compilation parameters. The following comments can be added to the script;

- `//AssemblyReference -Name <assembly dll> [-CLR [Core|Framework]]`: The assembly DLL to reference during compilation, the optional `-CLR` flag can also be used to state whether to reference when running under .NET Core, Framework, or both (if omitted)
- `//NoWarn -Name <error id> [-CLR [Core|Framework]]`: A compiler warning ID to ignore when compiling the code, the optional `-CLR` works the same as above. A list of warnings can be found at [Compiler errors](#)

As well as this, the following pre-processor symbols are defined;

- `CORECLR`: This symbol is present when PowerShell is running through .NET Core
- `WINDOWS`: This symbol is present when PowerShell is running on Windows
- `UNIX`: This symbol is present when PowerShell is running on Unix

A combination of these flags help to make a module util interoperable on both .NET Framework and .NET Core, here is an example of them in action:

```
#if CORECLR
using Newtonsoft.Json;
#else
using System.Web.Script.Serialization;
#endif

//AssemblyReference -Name Newtonsoft.Json.dll -CLR Core
//AssemblyReference -Name System.Web.Extensions.dll -CLR Framework

// Ignore error CS1702 for all .NET types
//NoWarn -Name CS1702

// Ignore error CS1956 only for .NET Framework
//NoWarn -Name CS1956 -CLR Framework
```

The following is a list of module_utils that are packaged with Ansible and a general description of what

they do:

- **ArgvParser**: Utility used to convert a list of arguments to an escaped string compliant with the Windows argument parsing rules.
- **CamelConversion**: Utility used to convert camelCase strings/lists/dicts to snake_case.
- **CommandUtil**: Utility used to execute a Windows process and return the stdout/stderr and rc as separate objects.
- **FileUtil**: Utility that expands on the `Get-ChildItem` and `Test-Path` to work with special files like `C:\pagefile.sys`.
- **Legacy**: General definitions and helper utilities for Ansible module.
- **LinkUtil**: Utility to create, remove, and get information about symbolic links, junction points and hard inks.
- **SID**: Utilities used to convert a user or group to a Windows SID and vice versa.

For more details on any specific module utility and their requirements, please see the [Ansible module utilities source code](#).

PowerShell module utilities can be stored outside of the standard Ansible distribution for use with custom modules. Custom module_utils are placed in a folder called `module_utils` located in the root folder of the playbook or role directory.

C# module utilities can also be stored outside of the standard Ansible distribution for use with custom modules. Like PowerShell utils, these are stored in a folder called `module_utils` and the filename must end in the extension `.cs`, start with `Ansible.` and be named after the namespace defined in the util.

The below example is a role structure that contains two PowerShell custom module_utils called `Ansible.ModuleUtils.ModuleUtil1`, `Ansible.ModuleUtils.ModuleUtil2`, and a C# util containing the namespace `Ansible.CustomUtil`:

```
meta/
  main.yml
defaults/
  main.yml
module_utils/
  Ansible.ModuleUtils.ModuleUtil1.psm1
  Ansible.ModuleUtils.ModuleUtil2.psm1
  Ansible.CustomUtil.cs
tasks/
  main.yml
```

Each PowerShell module_util must contain at least one function that has been exported with `Export-ModuleMember` at the end of the file. For example

```
Export-ModuleMember -Function Invoke-CustomUtil, Get-CustomInfo
```

Windows playbook module testing

You can test a module with an Ansible playbook. For example:

- Create a playbook in any directory `touch testmodule.yml`.
- Create an inventory file in the same directory `touch hosts`.
- Populate the inventory file with the variables required to connect to a Windows host(s).
- Add the following to the new playbook file:

```
---
- name: test out windows module
  hosts: windows
  tasks:
    - name: test out module
      win_module:
        name: test name
```

- Run the playbook `ansible-playbook -i hosts testmodule.yml`

This can be useful for seeing how Ansible runs with the new module end to end. Other possible ways to test the module are shown below.

Windows debugging

Debugging a module currently can only be done on a Windows host. This can be useful when developing a new module or implementing bug fixes. These are some steps that need to be followed to set this up:

- Copy the module script to the Windows server
- Copy the folders `./lib/ansible/module_utils/powershell` and `./lib/ansible/module_utils/csharp` to the same directory as the script above
- Add an extra `#` to the start of any `#Requires -Module` lines in the module code, this is only required for any lines starting with `#Requires -Module`
- Add the following to the start of the module script that was copied to the server:

```
# Set $ErrorActionPreference to what's set during Ansible execution
$ErrorActionPreference = "Stop"

# Set the first argument as the path to a JSON file that contains the module args
```

(下页继续)

(续上页)

```

$args = @("$($pwd.Path)\args.json")

# Or instead of an args file, set $complex_args to the pre-processed module args
$complex_args = @{
    _ansible_check_mode = $false
    _ansible_diff = $false
    path = "C:\temp"
    state = "present"
}

# Import any C# utils referenced with '#AnsibleRequires -CSharpUtil' or 'using Ansible.;
# The $_csharp_utils entries should be the context of the C# util files and not the path
Import-Module -Name "$($pwd.Path)\powershell\Ansible.ModuleUtils.AddType.psm1"
$_csharp_utils = @(
    [System.IO.File]::ReadAllText("$($pwd.Path)\csharp\Ansible.Basic.cs")
)
Add-CSharpType -References $_csharp_utils -IncludeDebugInfo

# Import any PowerShell modules referenced with '#Requires -Module`
Import-Module -Name "$($pwd.Path)\powershell\Ansible.ModuleUtils.Legacy.psm1"

# End of the setup code and start of the module code
#!/powershell

```

You can add more args to `$complex_args` as required by the module or define the module options through a JSON file with the structure:

```

{
  "ANSIBLE_MODULE_ARGS": {
    "_ansible_check_mode": false,
    "_ansible_diff": false,
    "path": "C:\\temp",
    "state": "present"
  }
}

```

There are multiple IDEs that can be used to debug a Powershell script, two of the most popular ones are

- Powershell ISE
- Visual Studio Code

To be able to view the arguments as passed by Ansible to the module follow these steps.

- Prefix the Ansible command with `ANSIBLE_KEEP_REMOTE_FILES=1` to specify that Ansible should keep the exec files on the server.
- Log onto the Windows server using the same user account that Ansible used to execute the module.
- Navigate to `%TEMP%\...` It should contain a folder starting with `ansible-tmp-`.
- Inside this folder, open the PowerShell script for the module.
- In this script is a raw JSON script under `$json_raw` which contains the module arguments under `module_args`. These args can be assigned manually to the `$complex_args` variable that is defined on your debug script or put in the `args.json` file.

Windows unit testing

Currently there is no mechanism to run unit tests for Powershell modules under Ansible CI.

Windows integration testing

Integration tests for Ansible modules are typically written as Ansible roles. These test roles are located in `./test/integration/targets`. You must first set up your testing environment, and configure a test inventory for Ansible to connect to.

In this example we will set up a test inventory to connect to two hosts and run the integration tests for `win_stat`:

- Run the command `source ./hacking/env-setup` to prepare environment.
- Create a copy of `./test/integration/inventory.winrm.template` and name it `inventory.winrm`.
- Fill in entries under `[windows]` and set the required variables that are needed to connect to the host.
- *Install the required Python modules* to support WinRM and a configured authentication method.
- To execute the integration tests, run `ansible-test windows-integration win_stat`; you can replace `win_stat` with the role you wish to test.

This will execute all the tests currently defined for that role. You can set the verbosity level using the `-v` argument just as you would with `ansible-playbook`.

When developing tests for a new module, it is recommended to test a scenario once in check mode and twice not in check mode. This ensures that check mode does not make any changes but reports a change, as well as that the second run is idempotent and does not report changes. For example:

```
- name: remove a file (check mode)
  win_file:
    path: C:\temp
    state: absent
```

(下页继续)

(续上页)

```

register: remove_file_check
check_mode: yes

- name: get result of remove a file (check mode)
  win_command: powershell.exe "if (Test-Path -Path 'C:\temp') { 'true' } else { 'false' }
  ↪"
  register: remove_file_actual_check

- name: assert remove a file (check mode)
  assert:
    that:
      - remove_file_check is changed
      - remove_file_actual_check.stdout == 'true\r\n'

- name: remove a file
  win_file:
    path: C:\temp
    state: absent
  register: remove_file

- name: get result of remove a file
  win_command: powershell.exe "if (Test-Path -Path 'C:\temp') { 'true' } else { 'false' }
  ↪"
  register: remove_file_actual

- name: assert remove a file
  assert:
    that:
      - remove_file is changed
      - remove_file_actual.stdout == 'false\r\n'

- name: remove a file (idempotent)
  win_file:
    path: C:\temp
    state: absent
  register: remove_file_again

- name: assert remove a file (idempotent)
  assert:
    that:

```

(下页继续)

(续上页)

- not remove_file_again is changed

Windows communication and development support

Join the IRC channel `#ansible-devel` or `#ansible-windows` on freenode for discussions about Ansible development for Windows.

For questions and discussions pertaining to using the Ansible product, use the `#ansible` channel.

1.5.10 Developing Cisco ACI modules

This is a brief walk-through of how to create new Cisco ACI modules for Ansible.

For more information about Cisco ACI, look at the *Cisco ACI user guide*.

What's covered in this section:

- *Introduction*
- *ACI module structure*
 - *Importing objects from Python libraries*
 - *Defining the argument spec*
 - *Using the AnsibleModule object*
 - *Mapping variable definition*
 - *Using the ACIModule object*
 - * *Constructing URLs*
 - * *Getting the existing configuration*
 - * *When state is present*
 - * *When state is absent*
 - * *Exiting the module*
- *Testing ACI library functions*
 - *Testing for sanity checks*
 - *Testing ACI integration tests*
 - *Testing for test coverage*

Introduction

Ansible already ships with a large collection of Cisco ACI modules, however the ACI object model is huge and covering all possible functionality would easily cover more than 1500 individual modules.

If you are in need of specific functionality, you have 2 options:

- Learn the ACI object model and use the low-level APIC REST API using the `aci_rest` module
- Write your own dedicated modules, which is actually quite easy

参见:

ACI Fundamentals: ACI Policy Model A good introduction to the ACI object model.

APIC Management Information Model reference Complete reference of the APIC object model.

APIC REST API Configuration Guide Detailed guide on how the APIC REST API is designed and used, incl. many examples.

So let's look at how a typical ACI module is built up.

ACI module structure

Importing objects from Python libraries

The following imports are standard across ACI modules:

```
from ansible.module_utils.aci import ACIModule, aci_argument_spec
from ansible.module_utils.basic import AnsibleModule
```

Defining the argument spec

The first line adds the standard connection parameters to the module. After that, the next section will update the `argument_spec` dictionary with module-specific parameters. The module-specific parameters should include:

- the `object_id` (usually the name)
- the configurable properties of the object
- the parent object IDs (all parents up to the root)
- only child classes that are a 1-to-1 relationship (1-to-many/many-to-many require their own module to properly manage)
- the state
 - `state: absent` to ensure object does not exist

- state: present to ensure the object and configs exist; this is also the default
- state: query to retrieve information about objects in the class

```
def main():
    argument_spec = aci_argument_spec()
    argument_spec.update(
        object_id=dict(type='str', aliases=['name']),
        object_prop1=dict(type='str'),
        object_prop2=dict(type='str', choices=['choice1', 'choice2', 'choice3']),
        object_prop3=dict(type='int'),
        parent_id=dict(type='str'),
        child_object_id=dict(type='str'),
        child_object_prop=dict(type='str'),
        state=dict(type='str', default='present', choices=['absent', 'present', 'query
↪']),
    )
```

提示: Do not provide default values for configuration arguments. Default values could cause unintended changes to the object.

Using the AnsibleModule object

The following section creates an AnsibleModule instance. The module should support check-mode, so we pass the `argument_spec` and `supports_check_mode` arguments. Since these modules support querying the APIC for all objects of the module's class, the object/parent IDs should only be required if `state: absent` or `state: present`.

```
module = AnsibleModule(
    argument_spec=argument_spec,
    supports_check_mode=True,
    required_if=[
        ['state', 'absent', ['object_id', 'parent_id']],
        ['state', 'present', ['object_id', 'parent_id']],
    ],
)
```


Mapping variable definition

Once the AnsibleModule object has been initiated, the necessary parameter values should be extracted from `params` and any data validation should be done. Usually the only params that need to be extracted are those related to the ACI object configuration and its child configuration. If you have integer objects that you would like to validate, then the validation should be done here, and the `ACIModule.payload()` method will handle the string conversion.

```
object_id = object_id
object_prop1 = module.params['object_prop1']
object_prop2 = module.params['object_prop2']
object_prop3 = module.params['object_prop3']
if object_prop3 is not None and object_prop3 not in range(x, y):
    module.fail_json(msg='Valid object_prop3 values are between x and (y-1)')
child_object_id = module.params[' child_objec_id']
child_object_prop = module.params['child_object_prop']
state = module.params['state']
```

Using the ACIModule object

The ACIModule class handles most of the logic for the ACI modules. The ACIModule extends functionality to the AnsibleModule object, so the module instance must be passed into the class instantiation.

```
aci = ACIModule(module)
```

The ACIModule has six main methods that are used by the modules:

- `construct_url`
- `get_existing`
- `payload`
- `get_diff`
- `post_config`
- `delete_config`

The first two methods are used regardless of what value is passed to the `state` parameter.

Constructing URLs

The `construct_url()` method is used to dynamically build the appropriate URL to interact with the object, and the appropriate filter string that should be appended to the URL to filter the results.

- When the **state** is not **query**, the URL is the base URL to access the APIC plus the distinguished name to access the object. The filter string will restrict the returned data to just the configuration data.
- When **state** is **query**, the URL and filter string used depends on what parameters are passed to the object. This method handles the complexity so that it is easier to add new modules and so that all modules are consistent in what type of data is returned.

注解: Our design goal is to take all ID parameters that have values, and return the most specific data possible. If you do not supply any ID parameters to the task, then all objects of the class will be returned. If your task does consist of ID parameters set, then the data for the specific object is returned. If a partial set of ID parameters are passed, then the module will use the IDs that are passed to build the URL and filter strings appropriately.

The `construct_url()` method takes 2 required arguments:

- **self** - passed automatically with the class instance
- **root_class** - A dictionary consisting of `aci_class`, `aci_rn`, `target_filter`, and `module_object` keys
 - **aci_class**: The name of the class used by the APIC, e.g. `fvTenant`
 - **aci_rn**: The relative name of the object, e.g. `tn-ACME`
 - **target_filter**: A dictionary with key-value pairs that make up the query string for selecting a subset of entries, e.g. `{'name': 'ACME'}`
 - **module_object**: The particular object for this class, e.g. `ACME`

Example:

```
aci.construct_url(  
    root_class=dict(  
        aci_class='fvTenant',  
        aci_rn='tn-{0}'.format(tenant),  
        target_filter={'name': tenant},  
        module_object=tenant,  
    ),  
)
```

Some modules, like `aci_tenant`, are the root class and so they would not need to pass any additional arguments to the method.

The `construct_url()` method takes 4 optional arguments, the first three imitate the root class as described above, but are for child objects:

- subclass_1 - A dictionary consisting of `aci_class`, `aci_rn`, `target_filter`, and `module_object` keys
 - Example: Application Profile Class (AP)
- subclass_2 - A dictionary consisting of `aci_class`, `aci_rn`, `target_filter`, and `module_object` keys
 - Example: End Point Group (EPG)
- subclass_3 - A dictionary consisting of `aci_class`, `aci_rn`, `target_filter`, and `module_object` keys
 - Example: Binding a Contract to an EPG
- child_classes - The list of APIC names for the child classes supported by the modules.
 - This is a list, even if it is a list of one
 - These are the unfriendly names used by the APIC
 - These are used to limit the returned `child_classes` when possible
 - Example: `child_classes=['fvRsBDSsubnetToProfile', 'fvRsNdPfxPol']`

注解: Sometimes the APIC will require special characters ([,], and -) or will use object metadata in the name (“vlanns” for VLAN pools); the module should handle adding special characters or joining of multiple parameters in order to keep expected inputs simple.

Getting the existing configuration

Once the URL and filter string have been built, the module is ready to retrieve the existing configuration for the object:

- **state: present** retrieves the configuration to use as a comparison against what was entered in the task. All values that are different than the existing values will be updated.
- **state: absent** uses the existing configuration to see if the item exists and needs to be deleted.
- **state: query** uses this to perform the query for the task and report back the existing data.

```
aci.get_existing()
```

When state is present

When **state: present**, the module needs to perform a diff against the existing configuration and the task entries. If any value needs to be updated, then the module will make a POST request with only the items that need to be updated. Some modules have children that are in a 1-to-1 relationship with another object; for these cases, the module can be used to manage the child objects.

Building the ACI payload

The `aci.payload()` method is used to build a dictionary of the proposed object configuration. All parameters that were not provided a value in the task will be removed from the dictionary (both for the object and its children). Any parameter that does have a value will be converted to a string and added to the final dictionary object that will be used for comparison against the existing configuration.

The `aci.payload()` method takes two required arguments and 1 optional argument, depending on if the module manages child objects.

- `aci_class` is the APIC name for the object's class, e.g. `aci_class='fvBD'`
- `class_config` is the appropriate dictionary to be used as the payload for the POST request
 - The keys should match the names used by the APIC.
 - The values should be the corresponding value in `module.params`; these are the variables defined above
- `child_configs` is optional, and is a list of child config dictionaries.
 - The child configs include the full child object dictionary, not just the attributes configuration portion.
 - The configuration portion is built the same way as the object.

```
aci.payload(  
    aci_class=aci_class,  
    class_config=dict(  
        name=bd,  
        descr=description,  
        type=bd_type,  
    ),  
    child_configs=[  
        dict(  
            fvRsCtx=dict(  
                attributes=dict(  
                    tnFvCtxName=vrf  
                ),  
            ),  
        ),  
    ],  
)
```

Performing the request

The `get_diff()` method is used to perform the diff, and takes only one required argument, `aci_class`. Example: `aci.get_diff(aci_class='fvBD')`

The `post_config()` method is used to make the POST request to the APIC if needed. This method doesn't take any arguments and handles check mode. Example: `aci.post_config()`

Example code

```
if state == 'present':
    aci.payload(
        aci_class='<object APIC class>',
        class_config=dict(
            name=object_id,
            prop1=object_prop1,
            prop2=object_prop2,
            prop3=object_prop3,
        ),
        child_configs=[
            dict(
                '<child APIC class>'=dict(
                    attributes=dict(
                        child_key=child_object_id,
                        child_prop=child_object_prop
                    ),
                ),
            ),
        ],
    )

    aci.get_diff(aci_class='<object APIC class>')

    aci.post_config()
```

When state is absent

If the task sets the state to absent, then the `delete_config()` method is all that is needed. This method does not take any arguments, and handles check mode.

```
elif state == 'absent':
    aci.delete_config()
```

Exiting the module

To have the module exit, call the ACIModule method `exit_json()`. This method automatically takes care of returning the common return values for you.

```
aci.exit_json()

if __name__ == '__main__':
    main()
```

Testing ACI library functions

You can test your `construct_url()` and `payload()` arguments without accessing APIC hardware by using the following python script:

```
#!/usr/bin/python
import json
from ansible.module_utils.network.aci.aci import ACIModule

# Just another class mimicing a bare AnsibleModule class for construct_url() and
↳ payload() methods
class AltModule():
    params = dict(
        host='dummy',
        port=123,
        protocol='https',
        state='present',
        output_level='debug',
    )

# A sub-class of ACIModule to overload __init__ (we don't need to log into APIC)
class AltACIModule(ACIModule):
    def __init__(self):
        self.result = dict(changed=False)
        self.module = AltModule()
        self.params = self.module.params
```

(下页继续)

(续上页)

```

# Instantiate our version of the ACI module
aci = AltACIModule()

# Define the variables you need below
aep = 'AEP'
aep_domain = 'uni/phys-DOMAIN'

# Below test the construct_url() arguments to see if it produced correct results
aci.construct_url(
    root_class=dict(
        aci_class='infraAttEntityP',
        aci_rn='infra/attentp-{}'.format(aep),
        target_filter={'name': aep},
        module_object=aep,
    ),
    subclass_1=dict(
        aci_class='infraRsDomP',
        aci_rn='rsdomP-{}'.format(aep_domain),
        target_filter={'tDn': aep_domain},
        module_object=aep_domain,
    ),
)

# Below test the payload arguments to see if it produced correct results
aci.payload(
    aci_class='infraRsDomP',
    class_config=dict(tDn=aep_domain),
)

# Print the URL and proposed payload
print 'URL:', json.dumps(aci.url, indent=4)
print 'PAYLOAD:', json.dumps(aci.proposed, indent=4)

```

This will result in:

```

URL: "https://dummy/api/mo/uni/infra/attentp-AEP/rsdomP-[phys-DOMAIN].json"
PAYLOAD: {
    "infraRsDomP": {
        "attributes": {
            "tDn": "phys-DOMAIN"
        }
    }
}

```

(下页继续)

(续上页)

```
    }  
  }  
}
```

Testing for sanity checks

You can run from your fork something like:

```
$ ansible-test sanity --python 2.7 lib/ansible/modules/network/aci/aci_tenant.py
```

参见:

testing_sanity Information on how to build sanity tests.

Testing ACI integration tests

You can run this:

```
$ ansible-test network-integration --continue-on-error --allow-unsupported --diff -v aci_  
↪tenant
```

注解: You may need to add `--python 2.7` or `--python 3.6` in order to use the correct python version for performing tests.

You may want to edit the used inventory at *test/integration/inventory.networking* and add something like:

```
[aci:vars]  
aci_hostname=my-apic-1  
aci_username=admin  
aci_password=my-password  
aci_use_ssl=yes  
aci_use_proxy=no  
  
[aci]  
localhost ansible_ssh_host=127.0.0.1 ansible_connection=local
```

参见:

testing_integration Information on how to build integration tests.

Testing for test coverage

You can run this:

```
$ ansible-test network-integration --python 2.7 --allow-unsupported --coverage aci_tenant
$ ansible-test coverage report
```

1.5.11 Guidelines for Ansible Amazon AWS module development

The Ansible AWS modules and these guidelines are maintained by the Ansible AWS Working Group. For further information see the [AWS working group community page](#). If you are planning to contribute AWS modules to Ansible then getting in touch with the working group will be a good way to start, especially because a similar module may already be under development.

- *Maintaining existing modules*
 - *Fixing bugs*
 - *Adding new features*
 - *Migrating to boto3*
 - *Porting code to AnsibleAWSModule*
- *Creating new AWS modules*
 - *Use boto3 and AnsibleAWSModule*
 - *Naming your module*
 - *Importing botocore and boto3*
 - *Supporting Module Defaults*
- *Connecting to AWS*
 - *Common Documentation Fragments for Connection Parameters*
- *Handling exceptions*
 - *Using is_boto3_error_code*
 - *Using fail_json_aws()*
 - *using fail_json() and avoiding ansible.module_utils.aws.core*
- *API throttling (rate limiting) and pagination*
- *Returning Values*
- *Dealing with IAM JSON policy*

- *Dealing with tags*
- *Helper functions*
 - *camel_dict_to_snake_dict*
 - *snake_dict_to_camel_dict*
 - *ansible_dict_to_boto3_filter_list*
 - *boto_exception*
 - *boto3_tag_list_to_aws_dict*
 - *ansible_dict_to_boto3_tag_list*
 - *get_ec2_security_group_ids_from_names*
 - *compare_policies*
 - *compare_aws_tags*
- *Integration Tests for AWS Modules*
 - *AWS Credentials for Integration Tests*
 - *AWS Permissions for Integration Tests*
 - * *Troubleshooting IAM policies*

Maintaining existing modules

Fixing bugs

Bug fixes to code that relies on boto will still be accepted. When possible, the code should be ported to use boto3.

Adding new features

Try to keep backward compatibility with relatively recent versions of boto3. That means that if you want to implement some functionality that uses a new feature of boto3, it should only fail if that feature actually needs to be run, with a message stating the missing feature and minimum required version of boto3.

Use feature testing (e.g. `hasattr('boto3.module', 'shiny_new_method')`) to check whether boto3 supports a feature rather than version checking. For example, from the `ec2` module:

```
if boto_supports_profile_name_arg(ec2):
    params['instance_profile_name'] = instance_profile_name
else:
```

(下页继续)

(续上页)

```

    if instance_profile_name is not None:
        module.fail_json(msg="instance_profile_name parameter requires boto version 2.5.
↪0 or higher")

```

Migrating to boto3

Prior to Ansible 2.0, modules were written in either boto3 or boto. We are still porting some modules to boto3. Modules that still require boto should be ported to use boto3 rather than using both libraries (boto and boto3). We would like to remove the boto dependency from all modules.

Porting code to AnsibleAWSModule

Some old AWS modules use the generic `AnsibleModule` as a base rather than the more efficient `AnsibleAWSModule`. To port an old module to `AnsibleAWSModule`, change:

```

from ansible.module_utils.basic import AnsibleModule
...
module = AnsibleModule(...)

```

to:

```

from ansible.module_utils.aws.core import AnsibleAWSModule
...
module = AnsibleAWSModule(...)

```

Few other changes are required. `AnsibleAWSModule` does not inherit methods from `AnsibleModule` by default, but most useful methods are included. If you do find an issue, please raise a bug report.

When porting, keep in mind that `AnsibleAWSModule` also will add the default `ec2` argument spec by default. In port modules, you should see common arguments specified with:

```

def main():
    argument_spec = ec2_argument_spec()
    argument_spec.update(dict(
        state=dict(default='present', choices=['present', 'absent', 'enabled', 'disabled
↪']),
        name=dict(default='default'),
        # ... and so on ...
    ))
    module = AnsibleModule(argument_spec=argument_spec, supports_check_mode=True,)

```

These can be replaced with:

```
def main():
    argument_spec = dict(
        state=dict(default='present', choices=['present', 'absent', 'enabled', 'disabled',
↪']),
        name=dict(default='default'),
        # ... and so on ...
    )
    module = AnsibleAWSModule(argument_spec=argument_spec, supports_check_mode=True,)
```

Creating new AWS modules

Use boto3 and AnsibleAWSModule

All new AWS modules must use boto3 and `AnsibleAWSModule`.

`AnsibleAWSModule` greatly simplifies exception handling and library management, reducing the amount of boilerplate code. If you cannot use `AnsibleAWSModule` as a base, you must document the reason and request an exception to this rule.

Naming your module

Base the name of the module on the part of AWS that you actually use. (A good rule of thumb is to take whatever module you use with boto as a starting point). Don't further abbreviate names - if something is a well known abbreviation of a major component of AWS (for example, VPC or ELB), that's fine, but don't create new ones independently.

Unless the name of your service is quite unique, please consider using `aws_` as a prefix. For example `aws_lambda`.

Importing botocore and boto3

The `ansible.module_utils.ec2` module and `ansible.module_utils.core.aws` modules both automatically import boto3 and botocore. If boto3 is missing from the system then the variable `HAS_BOTO3` will be set to false. Normally, this means that modules don't need to import boto3 directly. There is no need to check `HAS_BOTO3` when using `AnsibleAWSModule` as the module does that check:

```
from ansible.module_utils.aws.core import AnsibleAWSModule
try:
    import botocore
```

(下页继续)

(续上页)

```
except ImportError:
    pass # handled by AnsibleAWSModule
```

or:

```
from ansible.module_utils.basic import AnsibleModule
from ansible.module_utils.ec2 import HAS_BOTO3
try:
    import botocore
except ImportError:
    pass # handled by imported HAS_BOTO3

def main():

    if not HAS_BOTO3:
        module.fail_json(msg='boto3 and botocore are required for this module')
```

Supporting Module Defaults

The existing AWS modules support using `module_defaults` for common authentication parameters. To do the same for your new module, add an entry for it in `lib/ansible/config/module_defaults.yml`. These entries take the form of:

```
aws_module_name:
- aws
```

Connecting to AWS

`AnsibleAWSModule` provides the `resource` and `client` helper methods for obtaining boto3 connections. These handle some of the more esoteric connection options, such as security tokens and boto profiles.

If using the basic `AnsibleModule` then you should use `get_aws_connection_info` and then `boto3_conn` to connect to AWS as these handle the same range of connection options.

These helpers also for missing profiles or a region not set when it needs to be, so you don't have to.

An example of connecting to ec2 is shown below. Note that unlike boto there is no `NoAuthHandlerFound` exception handling like in boto. Instead, an `AuthFailure` exception will be thrown when you use the connection. To ensure that authorization, parameter validation and permissions errors are all caught, you should catch `ClientError` and `BotoCoreError` exceptions with every boto3 connection call. See exception handling:

```
module.client('ec2')
```

or for the higher level ec2 resource:

```
module.resource('ec2')
```

An example of the older style connection used for modules based on AnsibleModule rather than AnsibleAWSModule:

```
region, ec2_url, aws_connect_params = get_aws_connection_info(module, boto3=True)
connection = boto3_conn(module, conn_type='client', resource='ec2', region=region,
↪endpoint=ec2_url, **aws_connect_params)
```

```
region, ec2_url, aws_connect_params = get_aws_connection_info(module, boto3=True)
connection = boto3_conn(module, conn_type='client', resource='ec2', region=region,
↪endpoint=ec2_url, **aws_connect_params)
```

Common Documentation Fragments for Connection Parameters

There are two *common documentation fragments* that should be included into almost all AWS modules:

- **aws** - contains the common boto connection parameters
- **ec2** - contains the common region parameter required for many AWS modules

These fragments should be used rather than re-documenting these properties to ensure consistency and that the more esoteric connection options are documented. For example:

```
DOCUMENTATION = '''
module: my_module
# some lines omitted here
requirements: [ 'botocore', 'boto3' ]
extends_documentation_fragment:
    - aws
    - ec2
'''
```

Handling exceptions

You should wrap any boto3 or botocore call in a try block. If an exception is thrown, then there are a number of possibilities for handling it.

- Catch the general `ClientError` or look for a specific error code with `is_boto3_error_code`.
- Use `aws_module.fail_json_aws()` to report the module failure in a standard way
- Retry using `AWSRetry`
- Use `fail_json()` to report the failure without using `ansible.module_utils.aws.core`
- Do something custom in the case where you know how to handle the exception

For more information on botocore exception handling see the [botocore error documentation](#).

Using `is_boto3_error_code`

To use `ansible.module_utils.aws.core.is_boto3_error_code` to catch a single AWS error code, call it in place of `ClientError` in your except clauses. In this case, *only* the `InvalidGroup.NotFound` error code will be caught here, and any other error will be raised for handling elsewhere in the program.

```
try:
    info = connection.describe_security_groups(**kwargs)
except is_boto3_error_code('InvalidGroup.NotFound'):
    pass
do_something(info)  # do something with the info that was successfully returned
```

Using `fail_json_aws()`

In the `AnsibleAWSModule` there is a special method, `module.fail_json_aws()` for nice reporting of exceptions. Call this on your exception and it will report the error together with a traceback for use in Ansible verbose mode.

You should use the `AnsibleAWSModule` for all new modules, unless not possible. If adding significant amounts of exception handling to existing modules, we recommend migrating the module to use `AnsibleAWSModule` (there are very few changes required to do this)

```
from ansible.module_utils.aws.core import AnsibleAWSModule

# Set up module parameters
# module params code here

# Connect to AWS
# connection code here

# Make a call to AWS
```

(下页继续)

(续上页)

```

name = module.params.get['name']
try:
    result = connection.describe_frooble(FroobleName=name)
except (botocore.exceptions.BotoCoreError, botocore.exceptions.ClientError) as e:
    module.fail_json_aws(e, msg="Couldn't obtain frooble %s" % name)

```

Note that it should normally be acceptable to catch all normal exceptions here, however if you expect anything other than botocore exceptions you should test everything works as expected.

If you need to perform an action based on the error boto3 returned, use the error code.

```

# Make a call to AWS
name = module.params.get['name']
try:
    result = connection.describe_frooble(FroobleName=name)
except botocore.exceptions.ClientError as e:
    if e.response['Error']['Code'] == 'FroobleNotFound':
        workaround_failure() # This is an error that we can work around
    else:
        module.fail_json_aws(e, msg="Couldn't obtain frooble %s" % name)
except botocore.exceptions.BotoCoreError as e:
    module.fail_json_aws(e, msg="Couldn't obtain frooble %s" % name)

```

using fail_json() and avoiding ansible.module_utils.aws.core

Boto3 provides lots of useful information when an exception is thrown so pass this to the user along with the message.

```

from ansible.module_utils.ec2 import HAS_BOTO3
try:
    import botocore
except ImportError:
    pass # caught by imported HAS_BOTO3

# Connect to AWS
# connection code here

# Make a call to AWS
name = module.params.get['name']
try:

```

(下页继续)

(续上页)

```

    result = connection.describe_frooble(FroobleName=name)
except boto3.exceptions.ClientError as e:
    module.fail_json(msg="Couldn't obtain frooble %s: %s" % (name, str(e)),
                     exception=traceback.format_exc(),
                     **camel_dict_to_snake_dict(e.response))

```

Note: we use `str(e)` rather than `e.message` as the latter doesn't work with python3

If you need to perform an action based on the error boto3 returned, use the error code.

```

# Make a call to AWS
name = module.params.get['name']
try:
    result = connection.describe_frooble(FroobleName=name)
except boto3.exceptions.ClientError as e:
    if e.response['Error']['Code'] == 'FroobleNotFound':
        workaround_failure() # This is an error that we can work around
    else:
        module.fail_json(msg="Couldn't obtain frooble %s: %s" % (name, str(e)),
                         exception=traceback.format_exc(),
                         **camel_dict_to_snake_dict(e.response))
except boto3.exceptions.BotoCoreError as e:
    module.fail_json_aws(e, msg="Couldn't obtain frooble %s" % name)

```

API throttling (rate limiting) and pagination

For methods that return a lot of results, boto3 often provides [paginators](#). If the method you're calling has `NextToken` or `Marker` parameters, you should probably check whether a paginator exists (the top of each boto3 service reference page has a link to [Paginators](#), if the service has any). To use paginators, obtain a paginator object, call `paginator.paginate` with the appropriate arguments and then call `build_full_result`.

Any time that you are calling the AWS API a lot, you may experience API throttling, and there is an `AWSRetry` decorator that can be used to ensure backoff. Because exception handling could interfere with the retry working properly (as `AWSRetry` needs to catch throttling exceptions to work correctly), you'd need to provide a backoff function and then put exception handling around the backoff function.

You can use `exponential_backoff` or `jittered_backoff` strategies - see the `cloud module_utils` (`/lib/ansible/module_utils/cloud.py`) and [AWS Architecture blog](#) for more details.

The combination of these two approaches is then:

```

@AWSRetry.exponential_backoff(retries=5, delay=5)
def describe_some_resource_with_backoff(client, **kwargs):
    paginator = client.get_paginator('describe_some_resource')
    return paginator.paginate(**kwargs).build_full_result()['SomeResource']

def describe_some_resource(client, module):
    filters = ansible_dict_to_boto3_filter_list(module.params['filters'])
    try:
        return describe_some_resource_with_backoff(client, Filters=filters)
    except botocore.exceptions.ClientError as e:
        module.fail_json_aws(e, msg="Could not describe some resource")

```

If the underlying `describe_some_resources` API call throws a `ResourceNotFound` exception, `AWSRetry` takes this as a cue to retry until it's not thrown (this is so that when creating a resource, we can just retry until it exists).

To handle authorization failures or parameter validation errors in `describe_some_resource_with_backoff`, where we just want to return `None` if the resource doesn't exist and not retry, we need:

```

@AWSRetry.exponential_backoff(retries=5, delay=5)
def describe_some_resource_with_backoff(client, **kwargs):
    try:
        return client.describe_some_resource(ResourceName=kwargs['name'])['Resources']
    except botocore.exceptions.ClientError as e:
        if e.response['Error']['Code'] == 'ResourceNotFound':
            return None
        else:
            raise
    except BotoCoreError as e:
        raise

def describe_some_resource(client, module):
    name = module.params.get['name']
    try:
        return describe_some_resource_with_backoff(client, name=name)
    except (botocore.exceptions.BotoCoreError, botocore.exceptions.ClientError) as e:
        module.fail_json_aws(e, msg="Could not describe resource %s" % name)

```

To make use of `AWSRetry` easier, it can now be wrapped around a client returned by `AnsibleAWSModule`. any call from a client. To add retries to a client, create a client:

```
module.client('ec2', retry_decorator=AWSRetry.jittered_backoff(retries=10))
```

Any calls from that client can be made to use the decorator passed at call-time using the `aws_retry` argument. By default, no retries are used.

```
ec2 = module.client('ec2', retry_decorator=AWSRetry.jittered_backoff(retries=10))
ec2.describe_instances(InstanceIds=['i-123456789'], aws_retry=True)

# equivalent with normal AWSRetry
@AWSRetry.jittered_backoff(retries=10)
def describe_instances(client, **kwargs):
    return ec2.describe_instances(**kwargs)

describe_instances(module.client('ec2'), InstanceIds=['i-123456789'])
```

The call will be retried the specified number of times, so the calling functions don't need to be wrapped in the backoff decorator.

You can also use customization for `retries`, `delay` and `max_delay` parameters used by `AWSRetry.jittered_backoff` API using module params. You can take a look at the `cloudformation` <cloudformation_module> module for example.

To make all Amazon modules uniform, prefix the module param with `backoff_`, so `retries` becomes `backoff_retries` and likewise with `backoff_delay` and `backoff_max_delay`.

Returning Values

When you make a call using boto3, you will probably get back some useful information that you should return in the module. As well as information related to the call itself, you will also have some response metadata. It is OK to return this to the user as well as they may find it useful.

Boto3 returns all values CamelCased. Ansible follows Python standards for variable names and uses snake_case. There is a helper function in `module_utils/ec2.py` called `camel_dict_to_snake_dict` that allows you to easily convert the boto3 response to snake_case.

You should use this helper function and avoid changing the names of values returned by Boto3. E.g. if boto3 returns a value called 'SecretAccessKey' do not change it to 'AccessKey'.

```
# Make a call to AWS
result = connection.aws_call()

# Return the result to the user
module.exit_json(changed=True, **camel_dict_to_snake_dict(result))
```

Dealing with IAM JSON policy

If your module accepts IAM JSON policies then set the type to `'json'` in the module spec. For example:

```
argument_spec.update(
    dict(
        policy=dict(required=False, default=None, type='json'),
    )
)
```

Note that AWS is unlikely to return the policy in the same order that it was submitted. Therefore, use the `compare_policies` helper function which handles this variance.

`compare_policies` takes two dictionaries, recursively sorts and makes them hashable for comparison and returns True if they are different.

```
from ansible.module_utils.ec2 import compare_policies

import json

# some lines skipped here

# Get the policy from AWS
current_policy = json.loads(aws_object.get_policy())
user_policy = json.loads(module.params.get('policy'))

# Compare the user submitted policy to the current policy ignoring order
if compare_policies(user_policy, current_policy):
    # Update the policy
    aws_object.set_policy(user_policy)
else:
    # Nothing to do
    pass
```

Dealing with tags

AWS has a concept of resource tags. Usually the boto3 API has separate calls for tagging and untagging a resource. For example, the ec2 API has a `create_tags` and `delete_tags` call.

It is common practice in Ansible AWS modules to have a `purge_tags` parameter that defaults to true.

The `purge_tags` parameter means that existing tags will be deleted if they are not specified by the Ansible task.

There is a helper function *compare_aws_tags* to ease dealing with tags. It can compare two dicts and return the tags to set and the tags to delete. See the Helper function section below for more detail.

Helper functions

Along with the connection functions in Ansible `ec2.py` `module_utils`, there are some other useful functions detailed below.

camel_dict_to_snake_dict

`boto3` returns results in a dict. The keys of the dict are in CamelCase format. In keeping with Ansible format, this function will convert the keys to snake_case.

`camel_dict_to_snake_dict` takes an optional parameter called `ignore_list` which is a list of keys not to convert (this is usually useful for the `tags` dict, whose child keys should remain with case preserved)

Another optional parameter is `reversible`. By default, `HTTPEndpoint` is converted to `http_endpoint`, which would then be converted by `snake_dict_to_camel_dict` to `HttpEndpoint`. Passing `reversible=True` converts `HttpEndpoint` to `h_t_t_p_endpoint` which converts back to `HTTPEndpoint`.

snake_dict_to_camel_dict

`snake_dict_to_camel_dict` converts snake cased keys to camel case. By default, because it was first introduced for ECS purposes, this converts to dromedaryCase. An optional parameter called *capitalize_first*, which defaults to *False*, can be used to convert to CamelCase.

ansible_dict_to_boto3_filter_list

Converts a an Ansible list of filters to a boto3 friendly list of dicts. This is useful for any boto3 *__facts* modules.

boto_exception

Pass an exception returned from boto or boto3, and this function will consistently get the message from the exception.

Deprecated: use *AnsibleAWSModule*'s *fail_json_aws* instead.

boto3_tag_list_to_aws_dict

Converts a boto3 tag list to an Ansible dict. Boto3 returns tags as a list of dicts containing keys called 'Key' and 'Value' by default. This key names can be overridden when calling the function. For example,

if you have already camel_cased your list of tags you may want to pass lowercase key names instead i.e. 'key' and 'value' .

This function converts the list in to a single dict where the dict key is the tag key and the dict value is the tag value.

ansible_dict_to_boto3_tag_list

Opposite of above. Converts an Ansible dict to a boto3 tag list of dicts. You can again override the key names used if 'Key' and 'Value' is not suitable.

get_ec2_security_group_ids_from_names

Pass this function a list of security group names or combination of security group names and IDs and this function will return a list of IDs. You should also pass the VPC ID if known because security group names are not necessarily unique across VPCs.

compare_policies

Pass two dicts of policies to check if there are any meaningful differences and returns true if there are. This recursively sorts the dicts and makes them hashable before comparison.

This method should be used any time policies are being compared so that a change in order doesn't result in unnecessary changes.

compare_aws_tags

Pass two dicts of tags and an optional purge parameter and this function will return a dict containing key pairs you need to modify and a list of tag key names that you need to remove. Purge is True by default. If purge is False then any existing tags will not be modified.

This function is useful when using boto3 'add_tags' and 'remove_tags' functions. Be sure to use the other helper function *boto3_tag_list_to_aws_dict* to get an appropriate tag dict before calling this function. Since the AWS APIs are not uniform (e.g. EC2 versus Lambda) this will work without modification for some (Lambda) and others may need modification before using these values (such as EC2, which requires the tags to be in the form [{ 'Key' : key1}, { 'Key' : key2}]).

Integration Tests for AWS Modules

All new AWS modules should include integration tests to ensure that any changes in AWS APIs that affect the module are detected. At a minimum this should cover the key API calls and check the documented return values are present in the module result.

For general information on running the integration tests see the Integration Tests page of the Module Development Guide, especially the section on configuration for cloud tests.

The integration tests for your module should be added in `test/integration/targets/MODULE_NAME`.

You must also have a `aliases` file in `test/integration/targets/MODULE_NAME/aliases`. This file serves two purposes. First indicates it's in an AWS test causing the test framework to make AWS credentials available during the test run. Second putting the test in a test group causing it to be run in the continuous integration build.

Tests for new modules should be added to the same group as existing AWS tests. In general just copy an existing `aliases` file such as the `aws_s3 tests aliases` file.

AWS Credentials for Integration Tests

The testing framework handles running the test with appropriate AWS credentials, these are made available to your test in the following variables:

- `aws_region`
- `aws_access_key`
- `aws_secret_key`
- `security_token`

So all invocations of AWS modules in the test should set these parameters. To avoid duplicating these for every call, it's preferable to use `module_defaults`. For example:

```
- name: set connection information for aws modules and run tasks
  module_defaults:
    group/aws:
      aws_access_key: "{{ aws_access_key }}"
      aws_secret_key: "{{ aws_secret_key }}"
      security_token: "{{ security_token | default(omit) }}"
      region: "{{ aws_region }}"

  block:

- name: Do Something
  ec2_instance:
    ... params ...

- name: Do Something Else
  ec2_instance:
    ... params ...
```

AWS Permissions for Integration Tests

As explained in the Integration Test guide there are defined IAM policies in `hacking/aws_config/testing_policies/` that contain the necessary permissions to run the AWS integration test. The permissions used by CI are more restrictive than those in `hacking/aws_config/testing_policies`; for CI we want the most restrictive policy possible that still allows the given tests to pass.

If your module interacts with a new service or otherwise requires new permissions, tests will fail when you submit a pull request and the [Ansibullbot](#) will tag your PR as needing revision. We do not automatically grant additional permissions to the roles used by the continuous integration builds. You must provide the minimum IAM permissions required to run your integration test.

If your PR has test failures, check carefully to be certain the failure is only due to the missing permissions. If you've ruled out other sources of failure, add a comment with the `ready_for_review` tag and explain that it's due to missing permissions.

Your pull request cannot be merged until the tests are passing. If your pull request is failing due to missing permissions, you must collect the minimum IAM permissions required to run the tests.

There are two ways to figure out which IAM permissions you need for your PR to pass:

- Start with the most permissive IAM policy, run the tests to collect information about which resources your tests actually use, then construct a policy based on that output. This approach only works on modules that use *AnsibleAWSModule*.
- Start with the least permissive IAM policy, run the tests to discover a failure, add permissions for the resource that addresses that failure, then repeat. If your module uses *AnsibleModule* instead of *AnsibleAWSModule*, you must use this approach.

To start with the most permissive IAM policy:

- 1) [Create an IAM policy](#) that allows all actions (set **Action** and **Resource** to `*`).
- 2) Run your tests locally with this policy. On *AnsibleAWSModule*-based modules, the `debug_botocore_endpoint_logs` option is automatically set to `yes`, so you should see a list of AWS ACTIONS after the PLAY RECAP showing all the permissions used. If your tests use a `boto/AnsibleModule` module, you must start with the least permissive policy (see below).
- 3) Modify your policy to allow only the actions your tests use. Restrict account, region, and prefix where possible. Wait a few minutes for your policy to update.
- 4) Run the tests again with a user or role that allows only the new policy.
- 5) If the tests fail, troubleshoot (see tips below), modify the policy, run the tests again, and repeat the process until the tests pass with a restrictive policy.
- 6) Open a pull request proposing the minimum required policy to the [testing policies](#).

To start from the least permissive IAM policy:

- 1) Run the integration tests locally with no IAM permissions.

2) **Examine the error when the tests reach a failure.**

- a) If the error message indicates the action used in the request, add the action to your policy.
 - b) **If the error message does not indicate the action used in the request:**
 - Usually the action is a CamelCase version of the method name - for example, for an ec2 client the method `describe_security_groups` correlates to the action `ec2:DescribeSecurityGroups`.
 - Refer to the documentation to identify the action.
 - c) If the error message indicates the resource ARN used in the request, limit the action to that resource.
 - d) **If the error message does not indicate the resource ARN used:**
 - Determine if the action can be restricted to a resource by examining the documentation.
 - If the action can be restricted, use the documentation to construct the ARN and add it to the policy.
- 3) Add the action or resource that caused the failure to [an IAM policy](#). Wait a few minutes for your policy to update.
- 4) Run the tests again with this policy attached to your user or role.
- 5) If the tests still fail at the same place with the same error you will need to troubleshoot (see tips below). If the first test passes, repeat steps 2 and 3 for the next error. Repeat the process until the tests pass with a restrictive policy.
- 6) Open a pull request proposing the minimum required policy to the [testing policies](#).

Troubleshooting IAM policies

- When you make changes to a policy, wait a few minutes for the policy to update before re-running the tests.
- Use the [policy simulator](#) to verify that each action (limited by resource when applicable) in your policy is allowed.
- If you're restricting actions to certain resources, replace resources temporarily with `*`. If the tests pass with wildcard resources, there is a problem with the resource definition in your policy.
- If the initial troubleshooting above doesn't provide any more insight, AWS may be using additional undisclosed resources and actions.
- Examine the AWS FullAccess policy for the service for clues.
- Re-read the AWS documentation, especially the list of [Actions](#), [Resources](#) and [Condition Keys](#) for the various AWS services.

- Look at the [cloudfunder](#) documentation as a troubleshooting cross-reference.
- Use a search engine.
- Ask in the Ansible IRC channel `#ansible-aws` (on freenode IRC).

Some cases where tests should be marked as unsupported: 1) The tests take longer than 10 or 15 minutes to complete 2) The tests create expensive resources 3) The tests create inline policies

1.5.12 OpenStack Ansible Modules

These are a set of modules for interacting with OpenStack as either an admin or an end user. If the module does not begin with `os_`, it's either deprecated or soon to be. This document serves as developer coding guidelines for modules intended to be here.

- *Naming*
- *Interface*
- *Interoperability*
- *Libraries*
- *Testing*

Naming

- All module names should start with `os_`
- Name any module that a cloud consumer would expect to use after the logical resource it manages: `os_server` not `os_nova`. This naming convention acknowledges that the end user does not care which service manages the resource - that is a deployment detail. For example cloud consumers may not know whether their floating IPs are managed by Nova or Neutron.
- Name any module that a cloud admin would expect to use with the service and the resource: `os_keystone_domain`.
- If the module is one that a cloud admin and a cloud consumer could both use, the cloud consumer rules apply.

Interface

- If the resource being managed has an id, it should be returned.
- If the resource being managed has an associated object more complex than an id, it should also be returned.

Interoperability

- It should be assumed that the cloud consumer does not know a bazillion details about the deployment choices their cloud provider made, and a best effort should be made to present one sane interface to the Ansible user regardless of deployer insanity.
- All modules should work appropriately against all existing known public OpenStack clouds.
- It should be assumed that a user may have more than one cloud account that they wish to combine as part of a single Ansible-managed infrastructure.

Libraries

- All modules should use `openstack_full_argument_spec` to pick up the standard input such as auth and ssl support.
- All modules should include `extends_documentation_fragment: openstack`.
- All complex cloud interaction or interoperability code should be housed in the `openstacksdk` library.
- All OpenStack API interactions should happen via the `openstacksdk` and not via OpenStack Client libraries. The OpenStack Client libraries do not have end users as a primary audience, they are for intra-server communication.

Testing

- Integration testing is currently done in OpenStack's CI system
- Testing in `openstacksdk` produces an obvious chicken-and-egg scenario. Work is under way to trigger from and report on PRs directly.

1.5.13 oVirt Ansible Modules

This is a set of modules for interacting with oVirt/RHV. This document serves as developer coding guidelines for creating oVirt/RHV modules.

- *Naming*
- *Interface*
- *Interoperability*
- *Libraries*
- *New module development*
- *Testing*

Naming

- All modules should start with an `ovirt_` prefix.
- All modules should be named after the resource it manages in singular form.
- All modules that gather information should have a `_info` suffix.

Interface

- Every module should return the ID of the resource it manages.
- Every module should return the dictionary of the resource it manages.
- Never change the name of the parameter, as we guarantee backward compatibility. Use aliases instead.
- If a parameter can't achieve idempotency for any reason, please document it.

Interoperability

- All modules should work against all minor versions of version 4 of the API. Version 3 of the API is not supported.

Libraries

- All modules should use `ovirt_full_argument_spec` or `ovirt_info_full_argument_spec` to pick up the standard input (such as `auth` and `fetch_nested`).
- All modules should use `extends_documentation_fragment: ovirt` to go along with `ovirt_full_argument_spec`.
- All info modules should use `extends_documentation_fragment: ovirt_info` to go along with `ovirt_info_full_argument_spec`.
- Functions that are common to all modules should be implemented in the `module_utils/ovirt.py` file, so they can be reused.
- Python SDK version 4 must be used.

New module development

Please read *Should you develop a module?*, first to know what common properties, functions and features every module must have.

In order to achieve idempotency of oVirt entity attributes, a helper class was created. The first thing you need to do is to extend this class and override a few methods:

```

try:
    import ovirtsdk4.types as otypes
except ImportError:
    pass

from ansible.module_utils.ovirt import (
    BaseModule,
    equal
)

class ClustersModule(BaseModule):

    # The build method builds the entity we want to create.
    # Always be sure to build only the parameters the user specified
    # in their yaml file, so we don't change the values which we shouldn't
    # change. If you set the parameter to None, nothing will be changed.
    def build_entity(self):
        return otypes.Cluster(
            name=self.param('name'),
            comment=self.param('comment'),
            description=self.param('description'),
        )

    # The update_check method checks if the update is needed to be done on
    # the entity. The equal method doesn't check the values which are None,
    # which means it doesn't check the values which user didn't set in yaml.
    # All other values are checked and if there is found some mismatch,
    # the update method is run on the entity, the entity is build by
    # 'build_entity' method. You don't have to care about calling the update,
    # it's called behind the scene by the 'BaseModule' class.
    def update_check(self, entity):
        return (
            equal(self.param('comment'), entity.comment)
            and equal(self.param('description'), entity.description)
        )

```

The code above handle the check if the entity should be updated, so we don't update the entity if not needed and also it construct the needed entity of the SDK.

```

from ansible.module_utils.basic import AnsibleModule
from ansible.module_utils.ovirt import (

```

(下页继续)

(续上页)

```
        check_sdk,
        create_connection,
        ovirt_full_argument_spec,
    )

    # This module will support two states of the cluster,
    # either it will be present or absent. The user can
    # specify three parameters: name, comment and description,
    # The 'ovirt_full_argument_spec' function, will merge the
    # parameters created here with some common one like 'auth':
    argument_spec = ovirt_full_argument_spec(
        state=dict(
            choices=['present', 'absent'],
            default='present',
        ),
        name=dict(default=None, required=True),
        description=dict(default=None),
        comment=dict(default=None),
    )

    # Create the Ansible module, please always implement the
    # feature called 'check_mode', for 'create', 'update' and
    # 'delete' operations it's implemented by default in BaseModule:
    module = AnsibleModule(
        argument_spec=argument_spec,
        supports_check_mode=True,
    )

    # Check if the user has Python SDK installed:
    check_sdk(module)

    try:
        auth = module.params.pop('auth')

        # Create the connection to the oVirt engine:
        connection = create_connection(auth)

        # Create the service which manages the entity:
        clusters_service = connection.system_service().clusters_service()
```

(下页继续)

(续上页)

```

# Create the module which will handle create, update and delete flow:
clusters_module = ClustersModule(
    connection=connection,
    module=module,
    service=clusters_service,
)

# Check the state and call the appropriate method:
state = module.params['state']
if state == 'present':
    ret = clusters_module.create()
elif state == 'absent':
    ret = clusters_module.remove()

# The return value of the 'create' and 'remove' method is dictionary
# with the 'id' of the entity we manage and the type of the entity
# with filled in attributes of the entity. The 'change' status is
# also returned by those methods:
module.exit_json(**ret)
except Exception as e:
    # Modules can't raises exception, it always must exit with
    # 'module.fail_json' in case of exception. Always use
    # 'exception=traceback.format_exc' for debugging purposes:
    module.fail_json(msg=str(e), exception=traceback.format_exc())
finally:
    # Logout only in case the user passed the 'token' in 'auth'
    # parameter:
    connection.close(logout=auth.get('token') is None)

```

If your module must support action handling (for example, virtual machine start) you must ensure that you handle the states of the virtual machine correctly, and document the behavior of the module:

```

if state == 'running':
    ret = vms_module.action(
        action='start',
        post_action=vms_module._post_start_action,
        action_condition=lambda vm: (
            vm.status not in [
                otypes.VmStatus.MIGRATING,
                otypes.VmStatus.POWERING_UP,

```

(下页继续)

(续上页)

```

        otypes.VmStatus.REBOOT_IN_PROGRESS,
        otypes.VmStatus.WAIT_FOR_LAUNCH,
        otypes.VmStatus.UP,
        otypes.VmStatus.RESTORING_STATE,
    ]
),
wait_condition=lambda vm: vm.status == otypes.VmStatus.UP,
# Start action kwargs:
use_cloud_init=use_cloud_init,
use_sysprep=use_sysprep,
# ...
)

```

As you can see from the preceding example, the `action` method accepts the `action_condition` and `wait_condition`, which are methods which accept the virtual machine object as a parameter, so you can check whether the virtual machine is in a proper state before the action. The rest of the parameters are for the `start` action. You may also handle pre- or post- action tasks by defining `pre_action` and `post_action` parameters.

Testing

- Integration testing is currently done in oVirt' s CI system [on Jenkins](#) and [on GitHub](#).
- Please consider using these integration tests if you create a new module or add a new feature to an existing module.

1.5.14 Guidelines for VMware module development

The VMware modules and these guidelines are maintained by the VMware Working Group. For further information see the [team community](#) page.

- *Testing with govcsim*
- *Testing with your own infrastructure*
 - *Requirements*
 - * *NFS server configuration*
 - *Configure your installation*
 - *If you use an HTTP proxy*

- *Run the test-suite*
- *Unit-test*
- *Code style and best practice*
 - *datacenter argument with ESXi*
 - *esxi_hostname should not be mandatory*
 - *Functional tests*
 - * *Writing new tests*
 - * *No need to create too much resources*
 - * *VM names should be predictable*
 - * *Avoid the common boiler plate code in your test playbook*
- *Typographic convention*
 - *Nomenclature*

Testing with govcsim

Most of the existing modules are covered by functional tests. The tests are located in the `test/integration/targets/`.

By default, the tests run against a vCenter API simulator called `govcsim`. `ansible-test` will automatically pull a `govcsim container` <<https://quay.io/repository/ansible/vcenter-test-container>> and use it to set-up the test environment.

You can trigger the test of a module manually with the `ansible-test` command. For example, to trigger `vcenter_folder` tests:

```
source hacking/env-setup
ansible-test integration --python 3.7 vcenter_folder
```

`govcsim` is handy because it's much more fast than a regular test environment. However, it does not support all the ESXi or vCenter features.

注解: Do not confuse `govcsim` with `vcsm`. It's old outdated version of vCenter simulator whereas `govcsim` is new and written in go lang

Testing with your own infrastructure

You can also target a regular VMware environment. This paragraph explains step by step how you can run the test-suite yourself.

Requirements

- **2 ESXi hosts (6.5 or 6.7)**
 - with 2 NIC, the second ones should be available for the test
- a VCSA host
- a NFS server
- **Python dependencies:**
 - *pyvmomi* <<https://github.com/vmware/pyvmomi/tree/master/pyVmomi>>
 - *requests* <<https://2.python-requests.org/en/master/>>.

If you want to deploy your test environment in a hypervisor, both VMware or Libvirt <<https://github.com/goneri/vmware-on-libvirt>> work well.

NFS server configuration

Your NFS server must expose the following directory structure:

```
$ tree /srv/share/
/srv/share/
  isos
    base.iso
    centos.iso
    fedora.iso
  vms
2 directories, 3 files
```

On a Linux system, you can expose the directory over NFS with the following export file:

```
$ cat /etc/exports
/srv/share 192.168.122.0/255.255.255.0(rw,anonuid=1000,anongid=1000)
```

注解: With this configuration all the new files will be owned by the user with the UID and GID 1000/1000. Adjust the configuration to match your user' s UID/GID.

The service can be enabled with:

```
$ sudo systemctl enable --now nfs-server
```

Configure your installation

Prepare a configuration file that describes your set-up. The file should be called `test/integration/cloud-config-vcenter.ini` and based on `test/lib/ansible_test/config/cloud-config-vcenter.ini.template`. For instance, if you've deployed your lab with *vmware-on-libvirt* <<https://github.com/goneri/vmware-on-libvirt>>:

```
[DEFAULT]
vcenter_username: administrator@vsphere.local
vcenter_password: !234AaAa56
vcenter_hostname: vcenter.test
vmware_validate_certs: false
esxi1_username: root
esxi1_hostname: esxi1.test
esxi1_password: root
esxi2_username: root
esxi2_hostname: test2.test
esxi2_password: root
```

If you use an HTTP proxy

Support for hosting test infrastructure behind an HTTP proxy is currently in development. See the following pull requests for more information:

- ansible-test: vcenter behind an HTTP proxy <<https://github.com/ansible/ansible/pull/58208>>
- pyvmomi: proxy support <<https://github.com/vmware/pyvmomi/pull/799>>
- VMware: add support for HTTP proxy in connection API <<https://github.com/ansible/ansible/pull/52936>>

Once you have incorporated the code from those PRs, specify the location of the proxy server with the two extra keys:

```
vmware_proxy_host: esxi1-gw.ws.testing.ansible.com
vmware_proxy_port: 11153
```

In addition, you may need to adjust the variables of the following file to match the configuration of your lab: `test/integration/targets/prepare_vmware_tests/vars/real_lab.yml`. If you use *vmware-on-libvirt* <<https://github.com/goneri/vmware-on-libvirt>> to prepare you lab, you don't have anything to change.

Run the test-suite

Once your configuration is ready, you can trigger a run with the following command:

```
source hacking/env-setup
VMWARE_TEST_PLATFORM=static ansible-test integration --python 3.7 vmware_host_firewall_
↪manager
```

`vmware_host_firewall_manager` is the name of the module to test.

`vmware_guest` is much larger than any other test role and is rather slow. You can enable or disable some of its test playbooks in `test/integration/targets/vmware_guest/defaults/main.yml`.

Unit-test

The VMware modules have limited unit-test coverage. You can run the test suite with the following commands:

```
source hacking/env-setup
ansible-test units --venv --python 3.7 '.*vmware.*'
```

Code style and best practice

datacenter argument with ESXi

The `datacenter` parameter should not use `ha-datacenter` by default. This is because the user may not realize that Ansible silently targets the wrong data center.

esxi_hostname should not be mandatory

Depending upon the functionality provided by ESXi or vCenter, some modules can seamlessly work with both. In this case, `esxi_hostname` parameter should be optional.

```
if self.is_vcenter():
    esxi_hostname = module.params.get('esxi_hostname')
    if not esxi_hostname:
        self.module.fail_json("esxi_hostname parameter is mandatory")
    self.host = self.get_all_host_objs(cluster_name=cluster_name, esxi_host_name=esxi_
↪hostname)[0]
else:
    self.host = find_obj(self.content, [vim.HostSystem], None)
```

(下页继续)

(续上页)

```
if self.host is None:
    self.module.fail_json(msg="Failed to find host system.")
```

Functional tests

Writing new tests

If you are writing a new collection of integration tests, there are a few VMware-specific things to note beyond the standard Ansible integration testing process.

The test-suite uses a set of common, pre-defined vars located in the `test/integration/targets/prepare_vmware_tests/` role. The resources defined there are automatically created by importing that role at the start of your test:

```
- import_role:
    name: prepare_vmware_tests
vars:
    setup_datacenter: true
```

This will give you a ready to use cluster, datacenter, datastores, folder, switch, dvswitch, ESXi hosts, and VMs.

No need to create too much resources

Most of the time, it's not necessary to use `with_items` to create multiple resources. By avoiding it, you speed up the test execution and you simplify the clean up afterwards.

VM names should be predictable

If you need to create a new VM during your test, you can use `test_vm1`, `test_vm2` or `test_vm3`. This way it will be automatically clean up for you.

Avoid the common boiler plate code in your test playbook

From Ansible 2.10, the test suite uses `modules_defaults`. This module allow us to preinitialize the following default keys of the VMware modules:

- hostname
- username
- password

- `validate_certs`

For example, the following block:

```
- name: Add a VMware vSwitch
vmware_vswitch:
  hostname: '{{ vcenter_hostname }}'
  username: '{{ vcenter_username }}'
  password: '{{ vcenter_password }}'
  validate_certs: 'no'
  esxi_hostname: 'esxi1'
  switch_name: "boby"
  state: present
```

should be simplified to just:

```
- name: Add a VMware vSwitch
vmware_vswitch:
  esxi_hostname: 'esxi1'
  switch_name: "boby"
  state: present
```

Typographic convention

Nomenclature

We try to enforce the following rules in our documentation:

- VMware, not VMWare or vmware
- ESXi, not esxi or ESXI
- vCenter, not vcenter or VCenter

We also refer to `vcsim`'s Go implementation with `govcsim`. This to avoid any confusion with the outdated implementation.

1.5.15 Information for submitting a group of modules

Topics

- *Submitting a group of modules*
- *Before you start coding*

- *Naming convention*
- *Speak to us*
- *Where to get support*
- *Your first pull request*
- *Subsequent PRs*
- *Maintaining your modules*
- *New to git or GitHub*

Submitting a group of modules

This section discusses how to get multiple related modules into Ansible.

This document is intended for both companies wishing to add modules for their own products as well as users of 3rd party products wishing to add Ansible functionality.

It' s based on module development tips and tricks that the Ansible core team and community have accumulated.

注解: LICENSING REQUIREMENTS Ansible enforces the following licensing requirements:

- **Utilities (files in `lib/ansible/module_utils/`) may have one of two licenses:**
 - A file in `module_utils` used **only** for a specific vendor' s hardware, provider, or service may be licensed under GPLv3+. Adding a new file under `module_utils` with GPLv3+ needs to be approved by the core team.
 - All other `module_utils` must be licensed under BSD, so GPL-licensed third-party and Galaxy modules can use them.
 - If there' s doubt about the appropriate license for a file in `module_utils`, the Ansible Core Team will decide during an Ansible Core Community Meeting.
 - All other files shipped with Ansible, including all modules, must be licensed under the GPL license (GPLv3 or later).
-

Before you start coding

Although it' s tempting to get straight into coding, there are a few things to be aware of first. This list of prerequisites is designed to help ensure that you develop high-quality modules that flow easily through the review process and get into Ansible more quickly.

- Read though all the pages linked off *Ansible module development: getting started*; paying particular focus to the *Contributing your module to Ansible*.
- New modules must be PEP 8 compliant. See `testing_pep8` for more information.
- Starting with Ansible version 2.4, all new modules must *support Python 2.6+ and Python 3.5+*. If this is an issue, please contact us (see the “Speak to us” section later in this document to learn how).
- Have a look at the existing modules and how they’ve been named in the `all_modules`, especially in the same functional area (such as cloud, networking, databases).
- Shared code can be placed into `lib/ansible/module_utils/`
- Shared documentation (for example describing common arguments) can be placed in `lib/ansible/plugins/doc_fragments/`.
- With great power comes great responsibility: Ansible module maintainers have a duty to help keep modules up to date. As with all successful community projects, module maintainers should keep a watchful eye for reported issues and contributions.
- Although not required, unit and/or integration tests are strongly recommended. Unit tests are especially valuable when external resources (such as cloud or network devices) are required. For more information see *Testing Ansible* and the *Testing Working Group*. * Starting with Ansible 2.4 all `network_modules` MUST have unit tests.

Naming convention

As you may have noticed when looking under `lib/ansible/modules/` we support up to two directories deep (but no deeper), e.g. `databases/mysql`. This is used to group files on disk as well as group related modules into categories and topics the Module Index, for example: `database_modules`.

The directory name should represent the *product* or *OS* name, not the company name.

Each module should have the above (or similar) prefix; see existing `all_modules` for existing examples.

Note:

- File and directory names are always in lower case
- Words are separated with an underscore (`_`) character
- Module names should be in the singular, rather than plural, eg `command` not `commands`

Speak to us

Circulating your ideas before coding is a good way to help you set off in the right direction.

After reading the “Before you start coding” section you will hopefully have a reasonable idea of the structure of your modules.

We’ve found that writing a list of your proposed module names and a one or two line description of what they will achieve and having that reviewed by Ansible is a great way to ensure the modules fit the way people have used Ansible Modules before, and therefore make them easier to use.

Where to get support

Ansible has a thriving and knowledgeable community of module developers that is a great resource for getting your questions answered.

In the *Ansible Community Guide* you can find how to:

- Subscribe to the Mailing Lists - We suggest “Ansible Development List” (for codefreeze info) and “Ansible Announce list”
- `#ansible-devel` - We have found that IRC `#ansible-devel` on FreeNode’s IRC network works best for module developers so we can have an interactive dialogue.
- IRC meetings - Join the various weekly IRC meetings [meeting schedule and agenda page](#)

Your first pull request

Now that you’ve reviewed this document, you should be ready to open your first pull request.

The first PR is slightly different to the rest because it:

- defines the namespace
- provides a basis for detailed review that will help shape your future PRs
- may include shared documentation (*doc_fragments*) that multiple modules require
- may include shared code (*module_utils*) that multiple modules require

The first PR should include the following files:

- `lib/ansible/modules/$category/$topic/__init__.py` - An empty file to initialize namespace and allow Python to import the files. *Required new file*
- `lib/ansible/modules/$category/$topic/$yourfirstmodule.py` - A single module. *Required new file*
- `lib/ansible/plugins/doc_fragments/$topic.py` - Code documentation, such as details regarding common arguments. *Optional new file*
- `lib/ansible/module_utils/$topic.py` - Code shared between more than one module, such as common arguments. *Optional new file*

And that’s it.

Before pushing your PR to GitHub it’s a good idea to review the *Contributing your module to Ansible* again.

After publishing your PR to <https://github.com/ansible/ansible>, a Shippable CI test should run within a few minutes. Check the results (at the end of the PR page) to ensure that it's passing (green). If it's not passing, inspect each of the results. Most of the errors should be self-explanatory and are often related to badly formatted documentation (see *YAML Syntax*) or code that isn't valid Python 2.6 or valid Python 3.5 (see *Ansible and Python 3*). If you aren't sure what a Shippable test message means, copy it into the PR along with a comment and we will review.

If you need further advice, consider join the `#ansible-devel` IRC channel (see how in the “Where to get support”).

We have a `ansibullbot` helper that comments on GitHub Issues and PRs which should highlight important information.

Subsequent PRs

By this point you first PR that defined the module namespace should have been merged. You can take the lessons learned from the first PR and apply it to the rest of the modules.

Raise exactly one PR per module for the remaining modules.

Over the years we've experimented with different sized module PRs, ranging from one module to many tens of modules, and during that time we've found the following:

- A PR with a single file gets a higher quality review
- PRs with multiple modules are harder for the creator to ensure all feedback has been applied
- PRs with many modules take a lot more work to review, and tend to get passed over for easier-to-review PRs.

You can raise up to five PRs at once (5 PRs = 5 new modules) **after** your first PR has been merged. We've found this is a good batch size to keep the review process flowing.

Maintaining your modules

Now that your modules are integrated there are a few bits of housekeeping to be done.

Bot Meta Update *Ansibullbot* so it knows who to notify if/when bugs or PRs are raised against your modules `BOTMETA.yml`.

If there are multiple people that can be notified, please list them. That avoids waiting on a single person who may be unavailable for any reason. Note that in *BOTMETA.yml* you can take ownership of an entire directory.

Review Module web docs Review the autogenerated module documentation for each of your modules, found in Module Docs to ensure they are correctly formatted. If there are any issues please fix by raising a single PR.

If the module documentation hasn't been published live yet, please let a member of the Ansible Core Team know in the `#ansible-devel` IRC channel.

注解: Consider adding a scenario guide to cover how to use your set of modules. Use the sample scenario guide `rst` file to help you get started. For network modules, see *Documenting new network platforms* for further documentation requirements.

New to git or GitHub

We realize this may be your first use of Git or GitHub. The following guides may be of use:

- [How to create a fork of ansible/ansible](#)
- [How to sync \(update\) your fork](#)
- [How to create a Pull Request \(PR\)](#)

Please note that in the Ansible Git Repo the main branch is called `devel` rather than `master`, which is used in the official GitHub documentation

After your first PR has been merged ensure you “sync your fork” with `ansible/ansible` to ensure you've pulled in the directory structure and and shared code or documentation previously created.

As stated in the GitHub documentation, always use feature branches for your PRs, never commit directly into `devel`.

1.5.16 Testing Ansible

Topics

- *Why test your Ansible contributions?*
- *Types of tests*
- *Testing within GitHub & Shippable*
 - *Organization*
 - *Rerunning a failing CI job*
- *How to test a PR*
 - *Setup: Checking out a Pull Request*
 - *Testing the Pull Request*
 - * *Code Coverage Online*

- *Want to know more about testing?*

Why test your Ansible contributions?

If you're a developer, one of the most valuable things you can do is to look at GitHub issues and help fix bugs, since bug-fixing is almost always prioritized over feature development. Even for non-developers, helping to test pull requests for bug fixes and features is still immensely valuable.

Ansible users who understand how to write playbooks and roles should be able to test their work. GitHub pull requests will automatically run a variety of tests (e.g., Shippable) that show bugs in action. However, contributors must also test their work outside of the automated GitHub checks and show evidence of these tests in the PR to ensure that their work will be more likely to be reviewed and merged.

Read on to learn how Ansible is tested, how to test your contributions locally, and how to extend testing capabilities.

Types of tests

At a high level we have the following classifications of tests:

compile

- `testing__compile`
- Test python code against a variety of Python versions.

sanity

- `testing__sanity`
- Sanity tests are made up of scripts and tools used to perform static code analysis.
- The primary purpose of these tests is to enforce Ansible coding standards and requirements.

integration

- `testing__integration`
- Functional tests of modules and Ansible core functionality.

units

- `testing__units`
- Tests directly against individual parts of the code base.

If you're a developer, one of the most valuable things you can do is look at the GitHub issues list and help fix bugs. We almost always prioritize bug fixing over feature development.

Even for non developers, helping to test pull requests for bug fixes and features is still immensely valuable. Ansible users who understand writing playbooks and roles should be able to add integration tests and so GitHub pull requests with integration tests that show bugs in action will also be a great way to help.

Testing within GitHub & Shippable

Organization

When Pull Requests (PRs) are created they are tested using Shippable, a Continuous Integration (CI) tool. Results are shown at the end of every PR.

When Shippable detects an error and it can be linked back to a file that has been modified in the PR then the relevant lines will be added as a GitHub comment. For example:

```
The test `ansible-test sanity --test pep8` failed with the following errors:

lib/ansible/modules/network/foo/bar.py:509:17: E265 block comment should start with '# '

The test `ansible-test sanity --test validate-modules` failed with the following errors:
lib/ansible/modules/network/foo/bar.py:0:0: E307 version_added should be 2.4. Currently ↵
↵2.3
lib/ansible/modules/network/foo/bar.py:0:0: E316 ANSIBLE_METADATA.metadata_version: ↵
↵required key not provided @ data['metadata_version']. Got None
```

From the above example we can see that `--test pep8` and `--test validate-modules` have identified issues. The commands given allow you to run the same tests locally to ensure you've fixed the issues without having to push your changes to GitHub and wait for Shippable, for example:

If you haven't already got Ansible available, use the local checkout by running:

```
source hacking/env-setup
```

Then run the tests detailed in the GitHub comment:

```
ansible-test sanity --test pep8
ansible-test sanity --test validate-modules
```

If there isn't a GitHub comment stating what's failed you can inspect the results by clicking on the "Details" button under the "checks have failed" message at the end of the PR.

Rerunning a failing CI job

Occasionally you may find your PR fails due to a reason unrelated to your change. This could happen for several reasons, including:

- a temporary issue accessing an external resource, such as a yum or git repo
- a timeout creating a virtual machine to run the tests on

If either of these issues appear to be the case, you can rerun the Shippable test by:

- closing and re-opening the PR
- making another change to the PR and pushing to GitHub

If the issue persists, please contact us in `#ansible-devel` on Freenode IRC.

How to test a PR

Ideally, code should add tests that prove that the code works. That's not always possible and tests are not always comprehensive, especially when a user doesn't have access to a wide variety of platforms, or is using an API or web service. In these cases, live testing against real equipment can be more valuable than automation that runs against simulated interfaces. In any case, things should always be tested manually the first time as well.

Thankfully, helping to test Ansible is pretty straightforward, assuming you are familiar with how Ansible works.

Setup: Checking out a Pull Request

You can do this by:

- checking out Ansible
- fetching the proposed changes into a test branch
- testing
- commenting on that particular issue on GitHub

Here's how:

警告: Testing source code from GitHub pull requests sent to us does have some inherent risk, as the source code sent may have mistakes or malicious code that could have a negative impact on your system. We recommend doing all testing on a virtual machine, whether a cloud instance, or locally. Some users like Vagrant or Docker for this, but they are optional. It is also useful to have virtual machines of different Linux or other flavors, since some features (apt vs. yum, for example) are specific to those OS versions.

Create a fresh area to work:

```
git clone https://github.com/ansible/ansible.git ansible-pr-testing
cd ansible-pr-testing
```

Next, find the pull request you'd like to test and make note of its number. It will look something like this:

```
Use os.path.sep instead of hardcoding / #65381
```

注解: Only test `ansible:devel`

It is important that the PR request target be `ansible:devel`, as we do not accept pull requests into any other branch. Dot releases are cherry-picked manually by Ansible staff.

Use the pull request number when you fetch the proposed changes and create your branch for testing:

```
git fetch origin refs/pull/XXXX/head:testing_PRXXXX
git checkout testing_PRXXXX
```

The first command fetches the proposed changes from the pull request and creates a new branch named `testing_PRXXXX`, where the `XXXX` is the actual number associated with the pull request (for example, 65381). The second command checks out the newly created branch.

注解: If the GitHub user interface shows that the pull request will not merge cleanly, we do not recommend proceeding if you are not somewhat familiar with git and coding, as you will have to resolve a merge conflict. This is the responsibility of the original pull request contributor.

注解: Some users do not create feature branches, which can cause problems when they have multiple, unrelated commits in their version of `devel`. If the source looks like `someuser:devel`, make sure there is only one commit listed on the pull request.

The Ansible source includes a script that allows you to use Ansible directly from source without requiring a full installation that is frequently used by developers on Ansible.

Simply source it (to use the Linux/Unix terminology) to begin using it immediately:

```
source ./hacking/env-setup
```

This script modifies the `PYTHONPATH` environment variables (along with a few other things), which will be temporarily set as long as your shell session is open.

Testing the Pull Request

At this point, you should be ready to begin testing!

Some ideas of what to test are:

- Create a test Playbook with the examples in and check if they function correctly
- Test to see if any Python backtraces returned (that's a bug)
- Test on different operating systems, or against different library versions

Any potential issues should be added as comments on the pull request (and it's acceptable to comment if the feature works as well), remembering to include the output of `ansible --version`

Example:

```
Works for me! Tested on `Ansible 2.3.0`. I verified this on CentOS 6.5 and also Ubuntu ↵  
↪ 14.04.
```

If the PR does not resolve the issue, or if you see any failures from the unit/integration tests, just include that output instead:

This doesn't work for me.

When I ran this Ubuntu 16.04 it failed with the following:

```
““  
some output  
StackTrace  
some other output  
““
```

Code Coverage Online

The [online code coverage reports](#) are a good way to identify areas for testing improvement in Ansible. By following red colors you can drill down through the reports to find files which have no tests at all. Adding both integration and unit tests which show clearly how code should work, verify important Ansible functions and increase testing coverage in areas where there is none is a valuable way to help improve Ansible.

The code coverage reports only cover the `devel` branch of Ansible where new feature development takes place. Pull requests and new code will be missing from the `codecov.io` coverage reports so local reporting is needed. Most `ansible-test` commands allow you to collect code coverage, this is particularly useful to indicate where to extend testing. See `testing_running_locally` for more information.

Want to know more about testing?

If you'd like to know more about the plans for improving testing Ansible then why not join the [Testing Working Group](#).

1.5.17 The lifecycle of an Ansible module

Modules in the main Ansible repo have a defined life cycle, from first introduction to final removal. The module life cycle is tied to the *Ansible release cycle* `<release_cycle>` and reflected in the `ANSIBLE_METADATA` block. A module may move through these four states:

1. When a module is first accepted into Ansible, we consider it in tech preview and mark it **preview**. Modules in **preview** are not stable. You may change the parameters or dependencies, expand or reduce the functionality of **preview** modules. Many modules remain **preview** for years.
2. If a module matures, we may mark it **stableinterface** and commit to maintaining its parameters, dependencies, and functionality. We support (though we cannot guarantee) backwards compatibility for **stableinterface** modules, which means their parameters should be maintained with stable meanings.
3. If a module's target API changes radically, or if someone creates a better implementation of its functionality, we may mark it **deprecated**. Modules that are **deprecated** are still available but they are reaching the end of their life cycle. We retain deprecated modules for 4 release cycles with deprecation warnings to help users update playbooks and roles that use them.
4. When a module has been deprecated for four release cycles, we remove the code and mark the stub file **removed**. Modules that are **removed** are no longer shipped with Ansible. The stub file helps users find alternative modules.

Deprecating modules

To deprecate a module, you must:

1. Rename the file so it starts with an `_`, for example, rename `old_cloud.py` to `_old_cloud.py`. This keeps the module available and marks it as deprecated on the module index pages.
2. Mention the deprecation in the relevant **CHANGELOG**.
3. Reference the deprecation in the relevant `porting_guide_x.y.rst`.
4. Update `ANSIBLE_METADATA` to contain `status: ['deprecated']`.
5. Add `deprecated:` to the documentation with the following sub-values:

removed_in A string, such as "2.9"; the version of Ansible where the module will be replaced with a docs-only module stub. Usually current release +4.

why Optional string that used to detail why this has been removed.

alternative Inform users they should do instead, i.e. Use
M(whatmoduletouseinstead) instead..

- For an example of documenting deprecation, see this [PR that deprecates multiple modules](#).

Changing a module name

You can also rename a module and keep an alias to the old name by using a symlink that starts with `_`. This example allows the `stat` module to be called with `fileinfo`, making the following examples equivalent:

```
EXAMPLES = '''
ln -s stat.py _fileinfo.py
ansible -m stat -a "path=/tmp" localhost
ansible -m fileinfo -a "path=/tmp" localhost
'''
```

1.5.18 Developing plugins

- *Writing plugins in Python*
- *Raising errors*
- *String encoding*
- *Plugin configuration & documentation standards*
- *Developing particular plugin types*
 - *Action plugins*
 - *Cache plugins*
 - *Callback plugins*
 - *Connection plugins*
 - *Filter plugins*
 - *Inventory plugins*
 - *Lookup plugins*
 - *Test plugins*
 - *Vars plugins*

Plugins augment Ansible's core functionality with logic and features that are accessible to all modules. Ansible ships with a number of handy plugins, and you can easily write your own. All plugins must:

- be written in Python
- raise errors
- return strings in unicode
- conform to Ansible's configuration and documentation standards

Once you've reviewed these general guidelines, you can skip to the particular type of plugin you want to develop.

Writing plugins in Python

You must write your plugin in Python so it can be loaded by the `PluginLoader` and returned as a Python object that any module can use. Since your plugin will execute on the controller, you must write it in a *compatible version of Python*.

Raising errors

You should return errors encountered during plugin execution by raising `AnsibleError()` or a similar class with a message describing the error. When wrapping other exceptions into error messages, you should always use the `to_native` Ansible function to ensure proper string compatibility across Python versions:

```
from ansible.module_utils._text import to_native

try:
    cause_an_exception()
except Exception as e:
    raise AnsibleError('Something happened, this was original exception: %s' % to_
↳native(e))
```

Check the different `AnsibleError` objects and see which one applies best to your situation.

String encoding

You must convert any strings returned by your plugin into Python's unicode type. Converting to unicode ensures that these strings can run through Jinja2. To convert strings:

```
from ansible.module_utils._text import to_text
result_string = to_text(result_string)
```

Plugin configuration & documentation standards

To define configurable options for your plugin, describe them in the `DOCUMENTATION` section of the python file. Callback and connection plugins have declared configuration requirements this way since Ansible version 2.4; most plugin types now do the same. This approach ensures that the documentation of your plugin's options will always be correct and up-to-date. To add a configurable option to your plugin, define it in this format:

```
options:
  option_name:
    description: describe this config option
    default: default value for this config option
    env:
      - name: NAME_OF_ENV_VAR
    ini:
      - section: section_of_ansible.cfg_where_this_config_option_is_defined
        key: key_used_in_ansible.cfg
    required: True/False
    type: boolean/float/integer/list/none/path/pathlist/pathspec/string/tmppath
    version_added: X.x
```

To access the configuration settings in your plugin, use `self.get_option(<option_name>)`. For most plugin types, the controller pre-populates the settings. If you need to populate settings explicitly, use a `self.set_options()` call.

Plugins that support embedded documentation (see `ansible-doc` for the list) must include well-formed doc strings to be considered for merge into the Ansible repo. If you inherit from a plugin, you must document the options it takes, either via a documentation fragment or as a copy. See [Module format and documentation](#) for more information on correct documentation. Thorough documentation is a good idea even if you're developing a plugin for local use.

Developing particular plugin types

Action plugins

Action plugins let you integrate local processing and local data with module functionality.

To create an action plugin, create a new class with the `Base(ActionBase)` class as the parent:

```
from ansible.plugins.action import ActionBase

class ActionModule(ActionBase):
    pass
```

From there, execute the module using the `_execute_module` method to call the original module. After successful execution of the module, you can modify the module return data.

```
module_return = self._execute_module(module_name='<NAME_OF_MODULE>',
                                     module_args=module_args,
                                     task_vars=task_vars, tmp=tmp)
```

For example, if you wanted to check the time difference between your Ansible controller and your target machine(s), you could write an action plugin to check the local time and compare it to the return data from Ansible's `setup` module:

```
#!/usr/bin/python
# Make coding more python3-ish, this is required for contributions to Ansible
from __future__ import (absolute_import, division, print_function)
__metaclass__ = type

from ansible.plugins.action import ActionBase
from datetime import datetime

class ActionModule(ActionBase):
    def run(self, tmp=None, task_vars=None):
        super(ActionModule, self).run(tmp, task_vars)
        module_args = self._task.args.copy()
        module_return = self._execute_module(module_name='setup',
                                             module_args=module_args,
                                             task_vars=task_vars, tmp=tmp)

        ret = dict()
        remote_date = None
        if not module_return.get('failed'):
            for key, value in module_return['ansible_facts'].items():
                if key == 'ansible_date_time':
                    remote_date = value['iso8601']

        if remote_date:
            remote_date_obj = datetime.strptime(remote_date, '%Y-%m-%dT%H:%M:%SZ')
            time_delta = datetime.now() - remote_date_obj
            ret['delta_seconds'] = time_delta.seconds
            ret['delta_days'] = time_delta.days
            ret['delta_microseconds'] = time_delta.microseconds

        return dict(ansible_facts=dict(ret))
```

This code checks the time on the controller, captures the date and time for the remote machine using the `setup` module, and calculates the difference between the captured time and the local time, returning the time delta in days, seconds and microseconds.

For practical examples of action plugins, see the source code for the [action plugins included with Ansible Core](#)

Cache plugins

Cache plugins store gathered facts and data retrieved by inventory plugins. Only fact caching is currently supported by cache plugins in collections.

Import cache plugins using the `cache_loader` so you can use `self.set_options()` and `self.get_option(<option_name>)`. If you import a cache plugin directly in the code base, you can only access options via `ansible.constants`, and you break the cache plugin's ability to be used by an inventory plugin.

```
from ansible.plugins.loader import cache_loader
[...]
plugin = cache_loader.get('custom_cache', **cache_kwargs)
```

There are two base classes for cache plugins, `BaseCacheModule` for database-backed caches, and `BaseCacheFileModule` for file-backed caches.

To create a cache plugin, start by creating a new `CacheModule` class with the appropriate base class. If you're creating a plugin using an `__init__` method you should initialize the base class with any provided args and kwargs to be compatible with inventory plugin cache options. The base class calls `self.set_options(direct=kwargs)`. After the base class `__init__` method is called `self.get_option(<option_name>)` should be used to access cache options.

New cache plugins should take the options `_uri`, `_prefix`, and `_timeout` to be consistent with existing cache plugins.

```
from ansible.plugins.cache import BaseCacheModule

class CacheModule(BaseCacheModule):
    def __init__(self, *args, **kwargs):
        super(CacheModule, self).__init__(*args, **kwargs)
        self._connection = self.get_option('_uri')
        self._prefix = self.get_option('_prefix')
        self._timeout = self.get_option('_timeout')
```

If you use the `BaseCacheModule`, you must implement the methods `get`, `contains`, `keys`, `set`, `delete`, `flush`, and `copy`. The `contains` method should return a boolean that indicates if the key exists and has not expired. Unlike file-based caches, the `get` method does not raise a `KeyError` if the cache has expired.

If you use the `BaseFileCacheModule`, you must implement `_load` and `_dump` methods that will be called from the base class methods `get` and `set`.

If your cache plugin stores JSON, use `AnsibleJSONEncoder` in the `_dump` or `set` method and `AnsibleJSONDecoder` in the `_load` or `get` method.

For example cache plugins, see the source code for the [cache plugins included with Ansible Core](#).

Callback plugins

Callback plugins add new behaviors to Ansible when responding to events. By default, callback plugins control most of the output you see when running the command line programs.

To create a callback plugin, create a new class with the `Base(Callbacks)` class as the parent:

```
from ansible.plugins.callback import CallbackBase

class CallbackModule(CallbackBase):
    pass
```

From there, override the specific methods from the `CallbackBase` that you want to provide a callback for. For plugins intended for use with Ansible version 2.0 and later, you should only override methods that start with `v2`. For a complete list of methods that you can override, please see `__init__.py` in the `lib/ansible/plugins/callback` directory.

The following is a modified example of how Ansible's timer plugin is implemented, but with an extra option so you can see how configuration works in Ansible version 2.4 and later:

```
# Make coding more python3-ish, this is required for contributions to Ansible
from __future__ import (absolute_import, division, print_function)

__metaclass__ = type

# not only visible to ansible-doc, it also 'declares' the options the plugin requires
↪and how to configure them.
DOCUMENTATION = '''
    callback: timer
    callback_type: aggregate
    requirements:
        - whitelist in configuration
    short_description: Adds time to play stats
    version_added: "2.0"
    description:
        - This callback just adds total play duration to the play stats.
    options:
```

(下页继续)

(续上页)

```

format_string:
    description: format of the string shown to user at play end
    ini:
        - section: callback_timer
          key: format_string
    env:
        - name: ANSIBLE_CALLBACK_TIMER_FORMAT
    default: "Playbook run took %s days, %s hours, %s minutes, %s seconds"
'''

from datetime import datetime

from ansible.plugins.callback import CallbackBase

class CallbackModule(CallbackBase):
    """
    This callback module tells you how long your plays ran for.
    """
    CALLBACK_VERSION = 2.0
    CALLBACK_TYPE = 'aggregate'
    CALLBACK_NAME = 'namespace.collection_name.timer'

    # only needed if you ship it and don't want to enable by default
    CALLBACK_NEEDS_WHITELIST = True

    def __init__(self):
        # make sure the expected objects are present, calling the base's __init__
        super(CallbackModule, self).__init__()

        # start the timer when the plugin is loaded, the first play should start a few
        ↪ milliseconds after.
        self.start_time = datetime.now()

    def _days_hours_minutes_seconds(self, runtime):
        ''' internal helper method for this callback '''
        minutes = (runtime.seconds // 60) % 60
        r_seconds = runtime.seconds - (minutes * 60)
        return runtime.days, runtime.seconds // 3600, minutes, r_seconds

```

(下页继续)

(续上页)

```

    # this is only event we care about for display, when the play shows its summary
    ↳ stats; the rest are ignored by the base class
    def v2_playbook_on_stats(self, stats):
        end_time = datetime.now()
        runtime = end_time - self.start_time

        # Shows the usage of a config option declared in the DOCUMENTATION variable.
    ↳ Ansible will have set it when it loads the plugin.

        # Also note the use of the display object to print to screen. This is available
    ↳ to all callbacks, and you should use this over printing yourself

        self._display.display(self._plugin_options['format_string'] % (self._days_hours_
    ↳ minutes_seconds(runtime)))

```

Note that the `CALLBACK_VERSION` and `CALLBACK_NAME` definitions are required for properly functioning plugins for Ansible version 2.0 and later. `CALLBACK_TYPE` is mostly needed to distinguish ‘stdout’ plugins from the rest, since you can only load one plugin that writes to stdout.

For example callback plugins, see the source code for the [callback plugins included with Ansible Core](#)

Connection plugins

Connection plugins allow Ansible to connect to the target hosts so it can execute tasks on them. Ansible ships with many connection plugins, but only one can be used per host at a time. The most commonly used connection plugins are the `paramiko` SSH, native `ssh` (just called `ssh`), and `local` connection types. All of these can be used in playbooks and with `/usr/bin/ansible` to connect to remote machines.

Ansible version 2.1 introduced the `smart` connection plugin. The `smart` connection type allows Ansible to automatically select either the `paramiko` or `openssh` connection plugin based on system capabilities, or the `ssh` connection plugin if OpenSSH supports `ControlPersist`.

To create a new connection plugin (for example, to support SNMP, Message bus, or other transports), copy the format of one of the existing connection plugins and drop it into `connection` directory on your *local plugin path*.

For example connection plugins, see the source code for the [connection plugins included with Ansible Core](#).

Filter plugins

Filter plugins manipulate data. They are a feature of Jinja2 and are also available in Jinja2 templates used by the `template` module. As with all plugins, they can be easily extended, but instead of having a file for each one you can have several per file. Most of the filter plugins shipped with Ansible reside in a `core.py`.

Filter plugins do not use the standard configuration and documentation system described above.

For example filter plugins, see the source code for the [filter plugins](#) included with Ansible Core.

Inventory plugins

Inventory plugins parse inventory sources and form an in-memory representation of the inventory. Inventory plugins were added in Ansible version 2.4.

You can see the details for inventory plugins in the [Developing dynamic inventory](#) page.

Lookup plugins

Lookup plugins pull in data from external data stores. Lookup plugins can be used within playbooks both for looping —playbook language constructs like `with_fileglob` and `with_items` are implemented via lookup plugins—and to return values into a variable or parameter.

Lookup plugins are very flexible, allowing you to retrieve and return any type of data. When writing lookup plugins, always return data of a consistent type that can be easily consumed in a playbook. Avoid parameters that change the returned data type. If there is a need to return a single value sometimes and a complex dictionary other times, write two different lookup plugins.

Ansible includes many [filters](#) which can be used to manipulate the data returned by a lookup plugin. Sometimes it makes sense to do the filtering inside the lookup plugin, other times it is better to return results that can be filtered in the playbook. Keep in mind how the data will be referenced when determining the appropriate level of filtering to be done inside the lookup plugin.

Here's a simple lookup plugin implementation—this lookup returns the contents of a text file as a variable:

```
# python 3 headers, required if submitting to Ansible
from __future__ import (absolute_import, division, print_function)
__metaclass__ = type

DOCUMENTATION = """
    lookup: file
    author: Daniel Hokka Zakrisson <daniel@hozac.com>
    version_added: "0.9"
    short_description: read file contents
    description:
        - This lookup returns the contents from a file on the Ansible controller's
↵file system.
    options:
        _terms:
            description: path(s) of files to read
            required: True
```

(下页继续)

(续上页)

```

    notes:
        - if read in variable context, the file can be interpreted as YAML if the
↪content is valid to the parser.
        - this lookup does not understand globing --- use the fileglob lookup instead.
"""
from ansible.errors import AnsibleError, AnsibleParserError
from ansible.plugins.lookup import LookupBase
from ansible.utils.display import Display

display = Display()

class LookupModule(LookupBase):

    def run(self, terms, variables=None, **kwargs):

        # lookups in general are expected to both take a list as input and output a list
        # this is done so they work with the looping construct 'with_'.
        ret = []
        for term in terms:
            display.debug("File lookup term: %s" % term)

            # Find the file in the expected search path, using a class method
            # that implements the 'expected' search path for Ansible plugins.
            lookupfile = self.find_file_in_search_path(variables, 'files', term)

            # Don't use print or your own logging, the display class
            # takes care of it in a unified way.
            display.vvvv("File lookup using %s as file" % lookupfile)
            try:
                if lookupfile:
                    contents, show_data = self._loader._get_file_contents(lookupfile)
                    ret.append(contents.rstrip())
                else:
                    # Always use ansible error classes to throw 'final' exceptions,
                    # so the Ansible engine will know how to deal with them.
                    # The Parser error indicates invalid options passed
                    raise AnsibleParserError()
            except AnsibleParserError:

```

(下页继续)

(续上页)

```

        raise AnsibleError("could not locate file in lookup: %s" % term)

    return ret

```

The following is an example of how this lookup is called:

```

---
- hosts: all
  vars:
    contents: "{{ lookup('namespace.collection_name.file', '/etc/foo.txt') }}"

  tasks:

    - debug:
        msg: the value of foo.txt is {{ contents }} as seen today {{ lookup('pipe',
↪ 'date +%Y-%m-%d') }}

```

For example lookup plugins, see the source code for the [lookup plugins included with Ansible Core](#).

For more usage examples of lookup plugins, see [Using Lookups](#).

Test plugins

Test plugins verify data. They are a feature of Jinja2 and are also available in Jinja2 templates used by the `template` module. As with all plugins, they can be easily extended, but instead of having a file for each one you can have several per file. Most of the test plugins shipped with Ansible reside in a `core.py`. These are specially useful in conjunction with some filter plugins like `map` and `select`; they are also available for conditional directives like `when`.

Test plugins do not use the standard configuration and documentation system described above.

For example test plugins, see the source code for the [test plugins included with Ansible Core](#).

Vars plugins

Vars plugins inject additional variable data into Ansible runs that did not come from an inventory source, playbook, or command line. Playbook constructs like `host_vars` and `group_vars` work using vars plugins.

Vars plugins were partially implemented in Ansible 2.0 and rewritten to be fully implemented starting with Ansible 2.4. Vars plugins are unsupported by collections.

Older plugins used a `run` method as their main body/work:

```
def run(self, name, vault_password=None):
    pass # your code goes here
```

Ansible 2.0 did not pass passwords to older plugins, so vaults were unavailable. Most of the work now happens in the `get_vars` method which is called from the `VariableManager` when needed.

```
def get_vars(self, loader, path, entities):
    pass # your code goes here
```

The parameters are:

- `loader`: Ansible's `DataLoader`. The `DataLoader` can read files, auto-load JSON/YAML and decrypt vaulted data, and cache read files.
- `path`: this is 'directory data' for every inventory source and the current play's playbook directory, so they can search for data in reference to them. `get_vars` will be called at least once per available path.
- `entities`: these are host or group names that are pertinent to the variables needed. The plugin will get called once for hosts and again for groups.

This `get_vars` method just needs to return a dictionary structure with the variables.

Since Ansible version 2.4, vars plugins only execute as needed when preparing to execute a task. This avoids the costly 'always execute' behavior that occurred during inventory construction in older versions of Ansible. Since Ansible version 2.10, vars plugin execution can be toggled by the user to run when preparing to execute a task or after importing an inventory source.

Since Ansible 2.10, vars plugins can require whitelisting. Vars plugins that don't require whitelisting will run by default. To require whitelisting for your plugin set the class variable `REQUIRES_WHITELIST`:

```
class VarsModule(BaseVarsPlugin):
    REQUIRES_WHITELIST = True
```

Include the `vars_plugin_staging` documentation fragment to allow users to determine when vars plugins run.

```
DOCUMENTATION = '''
    vars: custom_hostvars
    version_added: "2.10"
    short_description: Load custom host vars
    description: Load custom host vars
    options:
        stage:
            ini:
```

(下页继续)

```

    - key: stage
      section: vars_custom_hostvars
  env:
    - name: ANSIBLE_VARS_PLUGIN_STAGE
  extends_documentation_fragment:
    - vars_plugin_staging
'''

```

Also since Ansible 2.10, vars plugins can reside in collections. Vars plugins in collections must require whitelisting to be functional.

For example vars plugins, see the source code for the [vars plugins](#) included with Ansible Core.

参见:

all_modules List of all modules

Python API Learn about the Python API for task execution

Developing dynamic inventory Learn about how to develop dynamic inventory sources

Ansible module development: getting started Learn about how to write Ansible modules

Mailing List The development mailing list

irc.freenode.net #ansible IRC chat channel

1.5.19 Developing dynamic inventory

Topics

- *Inventory sources*
- *Inventory plugins*
 - *Developing an inventory plugin*
 - * *verify_file*
 - * *parse*
 - * *inventory cache*
 - *Inventory source common format*
 - *The ‘auto’ plugin*
- *Inventory scripts*
 - *Inventory script conventions*

– *Tuning the external inventory script*

As described in [动态 Inventory 清单配置](#), Ansible can pull inventory information from dynamic sources, including cloud sources, using the supplied *inventory plugins*. If the source you want is not currently covered by existing plugins, you can create your own as with any other plugin type.

In previous versions you had to create a script or program that can output JSON in the correct format when invoked with the proper arguments. You can still use and write inventory scripts, as we ensured backwards compatibility via the script inventory plugin and there is no restriction on the programming language used. If you choose to write a script, however, you will need to implement some features yourself. i.e caching, configuration management, dynamic variable and group composition, etc. While with *inventory plugins* you can leverage the Ansible codebase to add these common features.

Inventory sources

Inventory sources are strings (i.e what you pass to `-i` in the command line), they can represent a path to a file/script or just be the raw data for the plugin to use. Here are some plugins and the type of source they use:

Plugin	Source
host list	A comma separated list of hosts
yaml	Path to a YAML format data file
constructed	Path to a YAML configuration file
ini	Path to an INI formatted data file
virtualbox	Path to a YAML configuration file
script plugin	Path to an executable outputting JSON

Inventory plugins

Like most plugin types (except modules) they must be developed in Python, since they execute on the controller they should match the same requirements [管理机环境要求](#).

Most of the documentation in [Developing plugins](#) also applies here, so as to not repeat ourselves, you should read that document first and we'll include inventory plugin specifics next.

Inventory plugins normally only execute at the start of a run, before playbooks/plays and roles are loaded, but they can be 're-executed' via the `meta: refresh_inventory` task, which will clear out the existing inventory and rebuild it.

When using the 'persistent' cache, inventory plugins can also use the configured cache plugin to store and retrieve data to avoid costly external calls.

Developing an inventory plugin

The first thing you want to do is use the base class:

```
from ansible.plugins.inventory import BaseInventoryPlugin

class InventoryModule(BaseInventoryPlugin):

    NAME = 'myplugin' # used internally by Ansible, it should match the file name but ↵
    ↪not required
```

If the inventory plugin is in a collection the NAME should be in the format of 'namespace.collection_name.myplugin' .

This class has a couple of methods each plugin should implement and a few helpers for parsing the inventory source and updating the inventory.

After you have the basic plugin working you might want to to incorporate other features by adding more base classes:

```
from ansible.plugins.inventory import BaseInventoryPlugin, Constructable, Cacheable

class InventoryModule(BaseInventoryPlugin, Constructable, Cacheable):

    NAME = 'myplugin'
```

For the bulk of the work in the plugin, We mostly want to deal with 2 methods `verify_file` and `parse`.

verify_file

This method is used by Ansible to make a quick determination if the inventory source is usable by the plugin. It does not need to be 100% accurate as there might be overlap in what plugins can handle and Ansible will try the enabled plugins (in order) by default.

```
def verify_file(self, path):
    ''' return true/false if this is possibly a valid file for this plugin to consume '''
    valid = False
    if super(InventoryModule, self).verify_file(path):
        # base class verifies that file exists and is readable by current user
        if path.endswith(('virtualbox.yaml', 'virtualbox.yml', 'vbox.yaml', 'vbox.yml')):
            valid = True
    return valid
```


In this case, from the virtualbox inventory plugin, we screen for specific file name patterns to avoid attempting to consume any valid yaml file. You can add any type of condition here, but the most common one is ‘extension matching’ . If you implement extension matching for YAML configuration files the path suffix `<plugin_name>.<yaml|yaml>` should be accepted. All valid extensions should be documented in the plugin description.

Another example that actually does not use a ‘file’ but the inventory source string itself, from the host list plugin:

```
def verify_file(self, path):
    ''' don't call base class as we don't expect a path, but a host list '''
    host_list = path
    valid = False
    b_path = to_bytes(host_list, errors='surrogate_or_strict')
    if not os.path.exists(b_path) and ',' in host_list:
        # the path does NOT exist and there is a comma to indicate this is a 'host list'
        valid = True
    return valid
```

This method is just to expedite the inventory process and avoid unnecessary parsing of sources that are easy to filter out before causing a parse error.

parse

This method does the bulk of the work in the plugin.

It takes the following parameters:

- inventory: inventory object with existing data and the methods to add hosts/groups/variables to inventory
- loader: Ansible’s DataLoader. The DataLoader can read files, auto load JSON/YAML and decrypt vaulted data, and cache read files.
- path: string with inventory source (this is usually a path, but is not required)
- cache: indicates whether the plugin should use or avoid caches (cache plugin and/or loader)

The base class does some minimal assignment for reuse in other methods.

```
def parse(self, inventory, loader, path, cache=True):

    self.loader = loader
    self.inventory = inventory
    self.templar = Templar(loader=loader)
```

It is up to the plugin now to deal with the inventory source provided and translate that into the Ansible inventory. To facilitate this, the example below uses a few helper functions:

```
NAME = 'myplugin'

def parse(self, inventory, loader, path, cache=True):

    # call base method to ensure properties are available for use with other helper
    ↪ methods
    super(InventoryModule, self).parse(inventory, loader, path, cache)

    # this method will parse 'common format' inventory sources and
    # update any options declared in DOCUMENTATION as needed
    config = self._read_config_data(path)

    # if NOT using _read_config_data you should call set_options directly,
    # to process any defined configuration for this plugin,
    # if you don't define any options you can skip
    #self.set_options()

    # example consuming options from inventory source
    mysession = apilib.session(user=self.get_option('api_user'),
                               password=self.get_option('api_pass'),
                               server=self.get_option('api_server')
    )

    # make requests to get data to feed into inventory
    mydata = mysession.getitall()

    #parse data and create inventory objects:
    for colo in mydata:
        for server in mydata[colo]['servers']:
            self.inventory.add_host(server['name'])
            self.inventory.set_variable(server['name'], 'ansible_host', server[
    ↪ 'external_ip'])
```

The specifics will vary depending on API and structure returned. But one thing to keep in mind, if the inventory source or any other issue crops up you should raise `AnsibleParserError` to let Ansible know that the source was invalid or the process failed.

For examples on how to implement an inventory plugin, see the source code here: `lib/ansible/plugins/inventory`.

inventory cache

Extend the inventory plugin documentation with the `inventory__cache` documentation fragment and use the `Cacheable` base class to have the caching system at your disposal.

```
extends_documentation_fragment:
    - inventory_cache
```

```
class InventoryModule(BaseInventoryPlugin, Constructable, Cacheable):

    NAME = 'myplugin'
```

Next, load the cache plugin specified by the user to read from and update the cache. If your inventory plugin uses YAML based configuration files and the `_read_config_data` method, the cache plugin is loaded within that method. If your inventory plugin does not use `_read_config_data`, you must load the cache explicitly with `load_cache_plugin`.

```
NAME = 'myplugin'

def parse(self, inventory, loader, path, cache=True):
    super(InventoryModule, self).parse(inventory, loader, path)

    self.load_cache_plugin()
```

Before using the cache, retrieve a unique cache key using the `get_cache_key` method. This needs to be done by all inventory modules using the cache, so you don't use/overwrite other parts of the cache.

```
def parse(self, inventory, loader, path, cache=True):
    super(InventoryModule, self).parse(inventory, loader, path)

    self.load_cache_plugin()
    cache_key = self.get_cache_key(path)
```

Now that you've enabled caching, loaded the correct plugin, and retrieved a unique cache key, you can set up the flow of data between the cache and your inventory using the `cache` parameter of the `parse` method. This value comes from the inventory manager and indicates whether the inventory is being refreshed (such as via `--flush-cache` or the meta task `refresh_inventory`). Although the cache shouldn't be used to populate the inventory when being refreshed, the cache should be updated with the new inventory if the user has enabled caching. You can use `self._cache` like a dictionary. The following pattern allows refreshing the inventory to work in conjunction with caching.

```
def parse(self, inventory, loader, path, cache=True):
```

(下页继续)

```

super(InventoryModule, self).parse(inventory, loader, path)

self.load_cache_plugin()
cache_key = self.get_cache_key(path)

    # cache may be True or False at this point to indicate if the inventory is being
↪refreshed
    # get the user's cache option too to see if we should save the cache if it is
↪changing
    user_cache_setting = self.get_option('cache')

    # read if the user has caching enabled and the cache isn't being refreshed
    attempt_to_read_cache = user_cache_setting and cache
    # update if the user has caching enabled and the cache is being refreshed; update
↪this value to True if the cache has expired below
    cache_needs_update = user_cache_setting and not cache

    # attempt to read the cache if inventory isn't being refreshed and the user has
↪caching enabled
    if attempt_to_read_cache:
        try:
            results = self._cache[cache_key]
        except KeyError:
            # This occurs if the cache_key is not in the cache or if the cache_key
↪expired, so the cache needs to be updated
            cache_needs_update = True

    if cache_needs_update:
        results = self.get_inventory()

        # set the cache
        self._cache[cache_key] = results

self.populate(results)

```

After the `parse` method is complete, the contents of `self._cache` is used to set the cache plugin if the contents of the cache have changed.

You have three other cache methods available:

- `set_cache_plugin` forces the cache plugin to be set with the contents of `self._cache` before the `parse` method completes

- `update_cache_if_changed` sets the cache plugin only if `self._cache` has been modified before the `parse` method completes
- `clear_cache` deletes the keys in `self._cache` from your cache plugin

Inventory source common format

To simplify development, most plugins use a mostly standard configuration file as the inventory source, YAML based and with just one required field `plugin` which should contain the name of the plugin that is expected to consume the file. Depending on other common features used, other fields might be needed, but each plugin can also add its own custom options as needed. For example, if you use the integrated caching, `cache_plugin`, `cache_timeout` and other cache related fields could be present.

The ‘auto’ plugin

Since Ansible 2.5, we include the auto inventory plugin enabled by default, which itself just loads other plugins if they use the common YAML configuration format that specifies a `plugin` field that matches an inventory plugin name, this makes it easier to use your plugin w/o having to update configurations.

Inventory scripts

Even though we now have inventory plugins, we still support inventory scripts, not only for backwards compatibility but also to allow users to leverage other programming languages.

Inventory script conventions

Inventory scripts must accept the `--list` and `--host <hostname>` arguments, other arguments are allowed but Ansible will not use them. They might still be useful for when executing the scripts directly.

When the script is called with the single argument `--list`, the script must output to stdout a JSON-encoded hash or dictionary containing all of the groups to be managed. Each group’s value should be either a hash or dictionary containing a list of each host, any child groups, and potential group variables, or simply a list of hosts:

```
{
  "group001": {
    "hosts": ["host001", "host002"],
    "vars": {
      "var1": true
    },
    "children": ["group002"]
  }
}
```

(下页继续)

(续上页)

```

    },
    "group002": {
      "hosts": ["host003", "host004"],
      "vars": {
        "var2": 500
      },
      "children": []
    }
  }
}

```

If any of the elements of a group are empty they may be omitted from the output.

When called with the argument `--host <hostname>` (where `<hostname>` is a host from above), the script must print either an empty JSON hash/dictionary, or a hash/dictionary of variables to make available to templates and playbooks. For example:

```

{
  "VAR001": "VALUE",
  "VAR002": "VALUE",
}

```

Printing variables is optional. If the script does not do this, it should print an empty hash or dictionary.

Tuning the external inventory script

1.3 新版功能.

The stock inventory script system detailed above works for all versions of Ansible, but calling `--host` for every host can be rather inefficient, especially if it involves API calls to a remote subsystem.

To avoid this inefficiency, if the inventory script returns a top level element called “`__meta`”, it is possible to return all of the host variables in one script execution. When this meta element contains a value for “`hostvars`”, the inventory script will not be invoked with `--host` for each host. This results in a significant performance increase for large numbers of hosts.

The data to be added to the top level JSON dictionary looks like this:

```

{

  # results of inventory script as above go here
  # ...
}

```

(下页继续)

(续上页)

```

    "_meta": {
        "hostvars": {
            "host001": {
                "var001" : "value"
            },
            "host002": {
                "var002": "value"
            }
        }
    }
}

```

To satisfy the requirements of using `_meta`, to prevent ansible from calling your inventory with `--host` you must at least populate `_meta` with an empty `hostvars` dictionary. For example:

```

{

    # results of inventory script as above go here
    # ...

    "_meta": {
        "hostvars": {}
    }
}

```

If you intend to replace an existing static inventory file with an inventory script, it must return a JSON object which contains an ‘all’ group that includes every host in the inventory as a member and every group in the inventory as a child. It should also include an ‘ungrouped’ group which contains all hosts which are not members of any other group. A skeleton example of this JSON object is:

```

{
    "_meta": {
        "hostvars": {}
    },
    "all": {
        "children": [
            "ungrouped"
        ]
    },
    "ungrouped": {
        "children": [

```

(下页继续)

(续上页)

```
    ]  
  }  
}
```

An easy way to see how this should look is using `ansible-inventory`, which also supports `--list` and `--host` parameters like an inventory script would.

参见:

Python API Python API to Playbooks and Ad Hoc Task Execution

Ansible module development: getting started Get started with developing a module

Developing plugins How to develop plugins

Ansible Tower REST API endpoint and GUI for Ansible, syncs with dynamic inventory

Development Mailing List Mailing list for development topics

irc.freenode.net #ansible IRC chat channel

1.5.20 Developing the Ansible Core Engine

Although many of the pieces of the Ansible Core Engine are plugins that can be swapped out via playbook directives or configuration, there are still pieces of the Engine that are not modular. The documents here give insight into how those pieces work together.

Ansible module architecture

If you're working on Ansible's Core code, writing an Ansible module, or developing an action plugin, this deep dive helps you understand how Ansible's program flow executes. If you're just using Ansible Modules in playbooks, you can skip this section.

- *Types of modules*
 - *Action plugins*
 - *New-style modules*
 - * *Python*
 - * *PowerShell*
 - *JSONARGS modules*
 - *Non-native want JSON modules*
 - *Binary modules*

- *Old-style modules*
- *How modules are executed*
 - *Executor/task_executor*
 - *The **normal** action plugin*
 - *Executor/module_common.py*
 - *Assembler frameworks*
 - * *Module Replacer framework*
 - * *Ansiballz framework*
 - *Passing args*
 - *Internal arguments*
 - * *__ansible_no_log*
 - * *__ansible_debug*
 - * *__ansible_diff*
 - * *__ansible_verbosity*
 - * *__ansible_selinux_special_fs*
 - * *__ansible_syslog_facility*
 - * *__ansible_version*
 - *Module return values & Unsafe strings*
 - *Special considerations*
 - * *Pipelining*
 - * *Why pass args over stdin?*
 - *AnsibleModule*
 - * *Argument spec*
 - *type*
 - *elements*
 - *default*
 - *fallback*
 - *choices*
 - *required*
 - *no_log*

- *aliases*
- *options*
- *apply_defaults*
- *removed_in_version*

Types of modules

Ansible supports several different types of modules in its code base. Some of these are for backwards compatibility and others are to enable flexibility.

Action plugins

Action plugins look like modules to anyone writing a playbook. Usage documentation for most action plugins lives inside a module of the same name. Some action plugins do all the work, with the module providing only documentation. Some action plugins execute modules. The `normal` action plugin executes modules that don't have special action plugins. Action plugins always execute on the controller.

Some action plugins do all their work on the controller. For example, the `debug` action plugin (which prints text for the user to see) and the `assert` action plugin (which tests whether values in a playbook satisfy certain criteria) execute entirely on the controller.

Most action plugins set up some values on the controller, then invoke an actual module on the managed node that does something with these values. For example, the `template` action plugin takes values from the user to construct a file in a temporary location on the controller using variables from the playbook environment. It then transfers the temporary file to a temporary file on the remote system. After that, it invokes the `copy` module which operates on the remote system to move the file into its final location, sets file permissions, and so on.

New-style modules

All of the modules that ship with Ansible fall into this category. While you can write modules in any language, all official modules (shipped with Ansible) use either Python or PowerShell.

New-style modules have the arguments to the module embedded inside of them in some manner. Old-style modules must copy a separate file over to the managed node, which is less efficient as it requires two over-the-wire connections instead of only one.

Python

New-style Python modules use the *Ansiballz framework* for constructing modules. These modules use imports from `ansible.module_utils` to pull in boilerplate module code, such as argument parsing, formatting of return values as *JSON*, and various file operations.

注解: In Ansible, up to version 2.0.x, the official Python modules used the *Module Replacer framework* framework. For module authors, *Ansiballz framework* is largely a superset of *Module Replacer framework* functionality, so you usually do not need to know about one versus the other.

PowerShell

New-style PowerShell modules use the *Module Replacer framework* framework for constructing modules. These modules get a library of PowerShell code embedded in them before being sent to the managed node.

JSONARGS modules

These modules are scripts that include the string `<<INCLUDE_ANSIBLE_MODULE_JSON_ARGS>>` in their body. This string is replaced with the JSON-formatted argument string. These modules typically set a variable to that value like this:

```
json_arguments = "<<INCLUDE_ANSIBLE_MODULE_JSON_ARGS>>"
```

Which is expanded as:

```
json_arguments = '{"param1": "test's quotes", "param2": "\"To be or not to be\" -\u2192Hamlet"}'
```

注解: Ansible outputs a *JSON* string with bare quotes. Double quotes are used to quote string values, double quotes inside of string values are backslash escaped, and single quotes may appear unescaped inside of a string value. To use JSONARGS, your scripting language must have a way to handle this type of string. The example uses Python's triple quoted strings to do this. Other scripting languages may have a similar quote character that won't be confused by any quotes in the JSON or it may allow you to define your own start-of-quote and end-of-quote characters. If the language doesn't give you any of these then you'll need to write a *non-native JSON module* or *Old-style module* instead.

These modules typically parse the contents of `json_arguments` using a JSON library and then use them as native variables throughout the code.

Non-native want JSON modules

If a module has the string `WANT_JSON` in it anywhere, Ansible treats it as a non-native module that accepts a filename as its only command line parameter. The filename is for a temporary file containing a *JSON* string containing the module's parameters. The module needs to open the file, read and parse the parameters, operate on the data, and print its return data as a JSON encoded dictionary to stdout before exiting.

These types of modules are self-contained entities. As of Ansible 2.1, Ansible only modifies them to change a shebang line if present.

参见:

Examples of Non-native modules written in ruby are in the [Ansible for Rubyists](#) repository.

Binary modules

From Ansible 2.2 onwards, modules may also be small binary programs. Ansible doesn't perform any magic to make these portable to different systems so they may be specific to the system on which they were compiled or require other binary runtime dependencies. Despite these drawbacks, you may have to compile a custom module against a specific binary library if that's the only way to get access to certain resources.

Binary modules take their arguments and return data to Ansible in the same way as *want JSON modules*.

参见:

One example of a [binary module](#) written in go.

Old-style modules

Old-style modules are similar to *want JSON modules*, except that the file that they take contains `key=value` pairs for their parameters instead of *JSON*. Ansible decides that a module is old-style when it doesn't have any of the markers that would show that it is one of the other types.

How modules are executed

When a user uses `ansible` or `ansible-playbook`, they specify a task to execute. The task is usually the name of a module along with several parameters to be passed to the module. Ansible takes these values and processes them in various ways before they are finally executed on the remote machine.

Executor/task_executor

The TaskExecutor receives the module name and parameters that were parsed from the *playbook* (or from the command line in the case of `/usr/bin/ansible`). It uses the name to decide whether it's looking at

a module or an *Action Plugin*. If it's a module, it loads the *Normal Action Plugin* and passes the name, variables, and other information about the task and play to that Action Plugin for further processing.

The normal action plugin

The **normal** action plugin executes the module on the remote host. It is the primary coordinator of much of the work to actually execute the module on the managed machine.

- It loads the appropriate connection plugin for the task, which then transfers or executes as needed to create a connection to that host.
- It adds any internal Ansible properties to the module's parameters (for instance, the ones that pass along `no_log` to the module).
- It works with other plugins (connection, shell, become, other action plugins) to create any temporary files on the remote machine and cleans up afterwards.
- It pushes the module and module parameters to the remote host, although the *module_common* code described in the next section decides which format those will take.
- It handles any special cases regarding modules (for instance, async execution, or complications around Windows modules that must have the same names as Python modules, so that internal calling of modules from other Action Plugins work.)

Much of this functionality comes from the *BaseAction* class, which lives in `plugins/action/__init__.py`. It uses the `Connection` and `Shell` objects to do its work.

注解: When *tasks* are run with the `async:` parameter, Ansible uses the *async* Action Plugin instead of the **normal** Action Plugin to invoke it. That program flow is currently not documented. Read the source for information on how that works.

Executor/module_common.py

Code in `executor/module_common.py` assembles the module to be shipped to the managed node. The module is first read in, then examined to determine its type:

- *PowerShell* and *JSON-args modules* are passed through *Module Replacer*.
- New-style *Python modules* are assembled by *Ansiballz framework*.
- *Non-native-want-JSON*, *Binary modules*, and *Old-Style modules* aren't touched by either of these and pass through unchanged.

After the assembling step, one final modification is made to all modules that have a shebang line. Ansible checks whether the interpreter in the shebang line has a specific path configured via an `ansible_$X_interpreter` inventory variable. If it does, Ansible substitutes that path for the interpreter

path given in the module. After this, Ansible returns the complete module data and the module type to the *Normal Action* which continues execution of the module.

Assembler frameworks

Ansible supports two assembler frameworks: Ansiballz and the older Module Replacer.

Module Replacer framework

The Module Replacer framework is the original framework implementing new-style modules, and is still used for PowerShell modules. It is essentially a preprocessor (like the C Preprocessor for those familiar with that programming language). It does straight substitutions of specific substring patterns in the module file. There are two types of substitutions:

- Replacements that only happen in the module file. These are public replacement strings that modules can utilize to get helpful boilerplate or access to arguments.
 - `from ansible.module_utils.MOD_LIB_NAME import *` is replaced with the contents of the `ansible/module_utils/MOD_LIB_NAME.py`. These should only be used with *new-style Python modules*.
 - `#<<INCLUDE_ANSIBLE_MODULE_COMMON>>` is equivalent to `from ansible.module_utils.basic import *` and should also only apply to new-style Python modules.
 - `# POWERSHELL_COMMON` substitutes the contents of `ansible/module_utils/powershell.ps1`. It should only be used with *new-style Powershell modules*.
- Replacements that are used by `ansible.module_utils` code. These are internal replacement patterns. They may be used internally, in the above public replacements, but shouldn't be used directly by modules.
 - `"<<ANSIBLE_VERSION>>"` is substituted with the Ansible version. In *new-style Python modules* under the *Ansiballz framework* the proper way is to instead instantiate an *AnsibleModule* and then access the version from `:attr:AnsibleModule.ansible_version`.
 - `"<<INCLUDE_ANSIBLE_MODULE_COMPLEX_ARGS>>"` is substituted with a string which is the Python `repr` of the *JSON* encoded module parameters. Using `repr` on the JSON string makes it safe to embed in a Python file. In new-style Python modules under the Ansiballz framework this is better accessed by instantiating an *AnsibleModule* and then using `AnsibleModule.params`.
 - `<<SELINUX_SPECIAL_FILESYSTEMS>>` substitutes a string which is a comma separated list of file systems which have a file system dependent security context in SELinux. In new-style Python modules, if you really need this you should instantiate an *AnsibleModule* and then use `AnsibleModule._selinux_special_fs`. The variable has also changed from a comma separated string of file system names to an actual python list of filesystem names.

- `<<INCLUDE_ANSIBLE_MODULE_JSON_ARGS>>` substitutes the module parameters as a JSON string. Care must be taken to properly quote the string as JSON data may contain quotes. This pattern is not substituted in new-style Python modules as they can get the module parameters another way.
- The string `syslog.LOG_USER` is replaced wherever it occurs with the `syslog_facility` which was named in `ansible.cfg` or any `ansible_syslog_facility` inventory variable that applies to this host. In new-style Python modules this has changed slightly. If you really need to access it, you should instantiate an *AnsibleModule* and then use `AnsibleModule._syslog_facility` to access it. It is no longer the actual syslog facility and is now the name of the syslog facility. See the *documentation on internal arguments* for details.

Ansiballz framework

The Ansiballz framework was adopted in Ansible 2.1 and is used for all new-style Python modules. Unlike the Module Replacer, Ansiballz uses real Python imports of things in `ansible/module_utils` instead of merely preprocessing the module. It does this by constructing a zipfile – which includes the module file, files in `ansible/module_utils` that are imported by the module, and some boilerplate to pass in the module’s parameters. The zipfile is then Base64 encoded and wrapped in a small Python script which decodes the Base64 encoding and places the zipfile into a temp directory on the managed node. It then extracts just the Ansible module script from the zip file and places that in the temporary directory as well. Then it sets the `PYTHONPATH` to find Python modules inside of the zip file and imports the Ansible module as the special name, `__main__`. Importing it as `__main__` causes Python to think that it is executing a script rather than simply importing a module. This lets Ansible run both the wrapper script and the module code in a single copy of Python on the remote machine.

注解:

- Ansible wraps the zipfile in the Python script for two reasons:
 - for compatibility with Python 2.6 which has a less functional version of Python’s `-m` command line switch.
 - so that pipelining will function properly. Pipelining needs to pipe the Python module into the Python interpreter on the remote node. Python understands scripts on stdin but does not understand zip files.
- Prior to Ansible 2.7, the module was executed via a second Python interpreter instead of being executed inside of the same process. This change was made once Python-2.4 support was dropped to speed up module execution.

In Ansiballz, any imports of Python modules from the `ansible.module_utils` package trigger inclusion of that Python file into the zipfile. Instances of `#<<INCLUDE_ANSIBLE_MODULE_COMMON>>` in the module are turned into `from ansible.module_utils.basic import *` and `ansible/module-utils/basic.py` is then

included in the zipfile. Files that are included from `module_utils` are themselves scanned for imports of other Python modules from `module_utils` to be included in the zipfile as well.

警告: At present, the Ansiballz Framework cannot determine whether an import should be included if it is a relative import. Always use an absolute import that has `ansible.module_utils` in it to allow Ansiballz to determine that the file should be included.

Passing args

Arguments are passed differently by the two frameworks:

- In *Module Replacer framework*, module arguments are turned into a JSON-ified string and substituted into the combined module file.
- In *Ansiballz framework*, the JSON-ified string is part of the script which wraps the zipfile. Just before the wrapper script imports the Ansible module as `__main__`, it monkey-patches the private, `_ANSIBLE_ARGS` variable in `basic.py` with the variable values. When a `ansible.module_utils.basic.AnsibleModule` is instantiated, it parses this string and places the args into `AnsibleModule.params` where it can be accessed by the module's other code.

警告: If you are writing modules, remember that the way we pass arguments is an internal implementation detail: it has changed in the past and will change again as soon as changes to the common `module_utils` code allow Ansible modules to forgo using `ansible.module_utils.basic.AnsibleModule`. Do not rely on the internal global `_ANSIBLE_ARGS` variable.

Very dynamic custom modules which need to parse arguments before they instantiate an `AnsibleModule` may use `_load_params` to retrieve those parameters. Although `_load_params` may change in breaking ways if necessary to support changes in the code, it is likely to be more stable than either the way we pass parameters or the internal global variable.

注解: Prior to Ansible 2.7, the Ansible module was invoked in a second Python interpreter and the arguments were then passed to the script over the script's stdin.

Internal arguments

Both *Module Replacer framework* and *Ansiballz framework* send additional arguments to the module beyond those which the user specified in the playbook. These additional arguments are internal parameters that help implement global Ansible features. Modules often do not need to know about these explicitly as the features

are implemented in `ansible.module_utils.basic` but certain features need support from the module so it's good to know about them.

The internal arguments listed here are global. If you need to add a local internal argument to a custom module, create an action plugin for that specific module - see `_original_basename` in the [copy action plugin](#) for an example.

`__ansible_no_log`

Boolean. Set to True whenever a parameter in a task or play specifies `no_log`. Any module that calls `AnsibleModule.log()` handles this automatically. If a module implements its own logging then it needs to check this value. To access in a module, instantiate an `AnsibleModule` and then check the value of `AnsibleModule.no_log`.

注解: `no_log` specified in a module's argument_spec is handled by a different mechanism.

`__ansible_debug`

Boolean. Turns more verbose logging on or off and turns on logging of external commands that the module executes. If a module uses `AnsibleModule.debug()` rather than `AnsibleModule.log()` then the messages are only logged if `__ansible_debug` is set to True. To set, add `debug: True` to `ansible.cfg` or set the environment variable `ANSIBLE_DEBUG`. To access in a module, instantiate an `AnsibleModule` and access `AnsibleModule._debug`.

`__ansible_diff`

Boolean. If a module supports it, tells the module to show a unified diff of changes to be made to templated files. To set, pass the `--diff` command line option. To access in a module, instantiate an `AnsibleModule` and access `AnsibleModule._diff`.

`__ansible_verbosity`

Unused. This value could be used for finer grained control over logging.

`__ansible_selinux_special_fs`

List. Names of filesystems which should have a special SELinux context. They are used by the `AnsibleModule` methods which operate on files (changing attributes, moving, and copying). To set, add a comma separated string of filesystem names in `ansible.cfg`:

```
# ansible.cfg
[selinux]
special_context_filesystems=nfs,vboxsf,fuse,ramfs,vfat
```

Most modules can use the built-in `AnsibleModule` methods to manipulate files. To access in a module that needs to know about these special context filesystems, instantiate an `AnsibleModule` and examine the list in `AnsibleModule._selinux_special_fs`.

This replaces `ansible.module_utils.basic.SELINUX_SPECIAL_FS` from *Module Replacer framework*. In module replacer it was a comma separated string of filesystem names. Under Ansiballz it's an actual list.

2.1 新版功能.

`_ansible_syslog_facility`

This parameter controls which syslog facility Ansible module logs to. To set, change the `syslog_facility` value in `ansible.cfg`. Most modules should just use `AnsibleModule.log()` which will then make use of this. If a module has to use this on its own, it should instantiate an *AnsibleModule* and then retrieve the name of the syslog facility from `AnsibleModule._syslog_facility`. The Ansiballz code is less hacky than the old *Module Replacer framework* code:

```
# Old module_replacer way
import syslog
syslog.openlog(NAME, 0, syslog.LOG_USER)

# New Ansiballz way
import syslog
facility_name = module._syslog_facility
facility = getattr(syslog, facility_name, syslog.LOG_USER)
syslog.openlog(NAME, 0, facility)
```

2.1 新版功能.

`_ansible_version`

This parameter passes the version of Ansible that runs the module. To access it, a module should instantiate an *AnsibleModule* and then retrieve it from `AnsibleModule.ansible_version`. This replaces `ansible.module_utils.basic.ANSIBLE_VERSION` from *Module Replacer framework*.

2.1 新版功能.

Module return values & Unsafe strings

At the end of a module's execution, it formats the data that it wants to return as a JSON string and prints the string to its stdout. The normal action plugin receives the JSON string, parses it into a Python dictionary, and returns it to the executor.

If Ansible templated every string return value, it would be vulnerable to an attack from users with access to managed nodes. If an unscrupulous user disguised malicious code as Ansible return value strings, and if those strings were then templated on the controller, Ansible could execute arbitrary code. To prevent this scenario, Ansible marks all strings inside returned data as **Unsafe**, emitting any Jinja2 templates in the strings verbatim, not expanded by Jinja2.

Strings returned by invoking a module through `ActionPlugin._execute_module()` are automatically marked as **Unsafe** by the normal action plugin. If another action plugin retrieves information from a module through some other means, it must mark its return data as **Unsafe** on its own.

In case a poorly-coded action plugin fails to mark its results as “Unsafe,” Ansible audits the results again when they are returned to the executor, marking all strings as **Unsafe**. The normal action plugin protects itself and any other code that it calls with the result data as a parameter. The check inside the executor protects the output of all other action plugins, ensuring that subsequent tasks run by Ansible will not template anything from those results either.

Special considerations

Pipelining

Ansible can transfer a module to a remote machine in one of two ways:

- it can write out the module to a temporary file on the remote host and then use a second connection to the remote host to execute it with the interpreter that the module needs
- or it can use what's known as pipelining to execute the module by piping it into the remote interpreter's stdin.

Pipelining only works with modules written in Python at this time because Ansible only knows that Python supports this mode of operation. Supporting pipelining means that whatever format the module payload takes before being sent over the wire must be executable by Python via stdin.

Why pass args over stdin?

Passing arguments via stdin was chosen for the following reasons:

- When combined with `ANSIBLE_PIPELINING`, this keeps the module's arguments from temporarily being saved onto disk on the remote machine. This makes it harder (but not impossible) for a malicious user on the remote machine to steal any sensitive information that may be present in the arguments.

- Command line arguments would be insecure as most systems allow unprivileged users to read the full commandline of a process.
- Environment variables are usually more secure than the commandline but some systems limit the total size of the environment. This could lead to truncation of the parameters if we hit that limit.

AnsibleModule

Argument spec

The `argument_spec` provided to `AnsibleModule` defines the supported arguments for a module, as well as their type, defaults and more.

Example `argument_spec`:

```
module = AnsibleModule(argument_spec=dict(
    top_level=dict(
        type='dict',
        options=dict(
            second_level=dict(
                default=True,
                type='bool',
            )
        )
    )
))
```

This section will discuss the behavioral attributes for arguments:

type

`type` allows you to define the type of the value accepted for the argument. The default value for `type` is `str`. Possible values are:

- `str`
- `list`
- `dict`
- `bool`
- `int`
- `float`
- `path`

- raw
- jsonarg
- json
- bytes
- bits

The `raw` type, performs no type validation or type casing, and maintains the type of the passed value.

elements

`elements` works in combination with `type` when `type='list'`. `elements` can then be defined as `elements='int'` or any other type, indicating that each element of the specified list should be of that type.

default

The `default` option allows sets a default value for the argument for the scenario when the argument is not provided to the module. When not specified, the default value is `None`.

fallback

`fallback` accepts a `tuple` where the first argument is a callable (function) that will be used to perform the lookup, based on the second argument. The second argument is a list of values to be accepted by the callable.

The most common callable used is `env_fallback` which will allow an argument to optionally use an environment variable when the argument is not supplied.

Example:

```
username=dict(fallback=(env_fallback, ['ANSIBLE_NET_USERNAME']))
```

choices

`choices` accepts a list of choices that the argument will accept. The types of `choices` should match the type.

required

`required` accepts a boolean, either `True` or `False` that indicates that the argument is required. This should not be used in combination with `default`.

no_log

`no_log` accepts a boolean, either `True` or `False`, that indicates explicitly whether or not the argument value should be masked in logs and output.

注解: In the absence of `no_log`, if the parameter name appears to indicate that the argument value is a password or passphrase (such as “`admin_password`”), a warning will be shown and the value will be masked in logs but **not** output. To disable the warning and masking for parameters that do not contain sensitive information, set `no_log` to `False`.

aliases

`aliases` accepts a list of alternative argument names for the argument, such as the case where the argument is `name` but the module accepts `aliases=['pkg']` to allow `pkg` to be interchangeably with `name`

options

`options` implements the ability to create a sub-argument_spec, where the sub options of the top level argument are also validated using the attributes discussed in this section. The example at the top of this section demonstrates use of `options`. `type` or `elements` should be `dict` in this case.

apply_defaults

`apply_defaults` works alongside `options` and allows the `default` of the sub-options to be applied even when the top-level argument is not supplied.

In the example of the `argument_spec` at the top of this section, it would allow `module.params['top_level']['second_level']` to be defined, even if the user does not provide `top_level` when calling the module.

removed_in_version

`removed_in_version` indicates which version of Ansible a deprecated argument will be removed in.

参见:

Python API Learn about the Python API for task execution

Developing plugins Learn about developing plugins

Mailing List The development mailing list

irc.freenode.net #ansible-devel IRC chat channel

1.5.21 Python API

Topics

- *Python API*
 - *Python API example*

注解: This API is intended for internal Ansible use. Ansible may make changes to this API at any time that could break backward compatibility with older versions of the API. Because of this, external use is not supported by Ansible.

There are several ways to use Ansible from an API perspective. You can use the Ansible Python API to control nodes, you can extend Ansible to respond to various Python events, you can write plugins, and you can plug in inventory data from external data sources. This document gives a basic overview and examples of the Ansible execution and playbook API.

If you would like to use Ansible programmatically from a language other than Python, trigger events asynchronously, or have access control and logging demands, please see the [Ansible Tower documentation](#).

注解: Because Ansible relies on forking processes, this API is not thread safe.

Python API example

This example is a simple demonstration that shows how to minimally run a couple of tasks:

```
#!/usr/bin/env python

import json
import shutil
from ansible.module_utils.common.collections import ImmutableDict
from ansible.parsing.dataloader import DataLoader
```

(下页继续)

(续上页)

```

from ansible.vars.manager import VariableManager
from ansible.inventory.manager import InventoryManager
from ansible.playbook.play import Play
from ansible.executor.task_queue_manager import TaskQueueManager
from ansible.plugins.callback import CallbackBase
from ansible import context
import ansible.constants as C

class ResultCallback(CallbackBase):
    """A sample callback plugin used for performing an action as results come in

    If you want to collect all results into a single object for processing at
    the end of the execution, look into utilizing the ``json`` callback plugin
    or writing your own custom callback plugin
    """
    def v2_runner_on_ok(self, result, **kwargs):
        """Print a json representation of the result

        This method could store the result in an instance attribute for retrieval later
        """
        host = result._host
        print(json.dumps({host.name: result._result}, indent=4))

# since the API is constructed for CLI it expects certain options to always be set in
↳ the context object
context.CLIARGS = ImmutableDict(connection='local', module_path=['/to/mymodules'],
↳ forks=10, become=None,
                                become_method=None, become_user=None, check=False,
↳ diff=False)

# initialize needed objects
loader = DataLoader() # Takes care of finding and reading yaml, json and ini files
passwords = dict(vault_pass='secret')

# Instantiate our ResultCallback for handling results as they come in. Ansible expects
↳ this to be one of its main display outlets
results_callback = ResultCallback()

# create inventory, use path to host config file as source or hosts in a comma separated
↳ string

```

(下页继续)

(续上页)

```

inventory = InventoryManager(loader=loader, sources='localhost,')

# variable manager takes care of merging all the different sources to give you a unified
↳ view of variables available in each context
variable_manager = VariableManager(loader=loader, inventory=inventory)

# create data structure that represents our play, including tasks, this is basically
↳ what our YAML loader does internally.
play_source = dict(
    name = "Ansible Play",
    hosts = 'localhost',
    gather_facts = 'no',
    tasks = [
        dict(action=dict(module='shell', args='ls'), register='shell_out'),
        dict(action=dict(module='debug', args=dict(msg='{{shell_out.stdout}}')))
    ]
)

# Create play object, playbook objects use .load instead of init or new methods,
# this will also automatically create the task objects from the info provided in play
↳ source
play = Play().load(play_source, variable_manager=variable_manager, loader=loader)

# Run it - instantiate task queue manager, which takes care of forking and setting up
↳ all objects to iterate over host list and tasks
tqm = None
try:
    tqm = TaskQueueManager(
        inventory=inventory,
        variable_manager=variable_manager,
        loader=loader,
        passwords=passwords,
        stdout_callback=results_callback, # Use our custom callback instead of
↳ the ``default`` callback plugin, which prints to stdout
    )
    result = tqm.run(play) # most interesting data for a play is actually sent to the
↳ callback's methods
finally:
    # we always need to cleanup child procs and the structures we use to communicate
↳ with them

```

(下页继续)

(续上页)

```
if tqm is not None:
    tqm.cleanup()

# Remove ansible tmpdir
shutil.rmtree(C.DEFAULT_LOCAL_TMP, True)
```

注解: Ansible emits warnings and errors via the display object, which prints directly to stdout, stderr and the Ansible log.

The source code for the **ansible** command line tools (`lib/ansible/cli/`) is [available on GitHub](#).

参见:

Developing dynamic inventory Developing dynamic inventory integrations

Ansible module development: getting started Getting started on developing a module

Developing plugins How to develop plugins

Development Mailing List Mailing list for development topics

irc.freenode.net #ansible IRC chat channel

1.5.22 Rebasing a pull request

You may find that your pull request (PR) is out-of-date and needs to be rebased. This can happen for several reasons:

- Files modified in your PR are in conflict with changes which have already been merged.
- Your PR is old enough that significant changes to automated test infrastructure have occurred.

Rebasing the branch used to create your PR will resolve both of these issues.

Configuring your remotes

Before you can rebase your PR, you need to make sure you have the proper remotes configured. Assuming you cloned your fork in the usual fashion, the **origin** remote will point to your fork:

```
$ git remote -v
origin  git@github.com:YOUR_GITHUB_USERNAME/ansible.git (fetch)
origin  git@github.com:YOUR_GITHUB_USERNAME/ansible.git (push)
```

However, you also need to add a remote which points to the upstream repository:

```
$ git remote add upstream https://github.com/ansible/ansible.git
```

Which should leave you with the following remotes:

```
$ git remote -v
origin  git@github.com:YOUR_GITHUB_USERNAME/ansible.git (fetch)
origin  git@github.com:YOUR_GITHUB_USERNAME/ansible.git (push)
upstream      https://github.com/ansible/ansible.git (fetch)
upstream      https://github.com/ansible/ansible.git (push)
```

Checking the status of your branch should show you're up-to-date with your fork at the `origin` remote:

```
$ git status
On branch YOUR_BRANCH
Your branch is up-to-date with 'origin/YOUR_BRANCH'.
nothing to commit, working tree clean
```

Rebasing your branch

Once you have an `upstream` remote configured, you can rebase the branch for your PR:

```
$ git pull --rebase upstream devel
```

This will replay the changes in your branch on top of the changes made in the upstream `devel` branch. If there are merge conflicts, you will be prompted to resolve those before you can continue.

Once you've rebased, the status of your branch will have changed:

```
$ git status
On branch YOUR_BRANCH
Your branch and 'origin/YOUR_BRANCH' have diverged,
and have 4 and 1 different commits each, respectively.
    (use "git pull" to merge the remote branch into yours)
nothing to commit, working tree clean
```

Don't worry, this is normal after a rebase. You should ignore the `git status` instructions to use `git pull`. We'll cover what to do next in the following section.

Updating your pull request

Now that you've rebased your branch, you need to push your changes to GitHub to update your PR.

Since rebasing re-writes git history, you will need to use a force push:

```
$ git push --force-with-lease
```

Your PR on GitHub has now been updated. This will automatically trigger testing of your changes. You should check in on the status of your PR after tests have completed to see if further changes are required.

Getting help rebasing

For help with rebasing your PR, or other development related questions, join us on our [#ansible-devel](#) IRC chat channel on [freenode.net](#).

参见:

The Ansible Development Cycle Information on roadmaps, opening PRs, Ansibullbot, and more

1.5.23 Using and Developing Module Utilities

Ansible provides a number of module utilities, or snippets of shared code, that provide helper functions you can use when developing your own modules. The `basic.py` module utility provides the main entry point for accessing the Ansible library, and all Python Ansible modules must import something from `ansible.module_utils`. A common option is to import `AnsibleModule`:

```
from ansible.module_utils.basic import AnsibleModule
```

The `ansible.module_utils` namespace is not a plain Python package: it is constructed dynamically for each task invocation, by extracting imports and resolving those matching the namespace against a *search path* derived from the active configuration.

To reduce the maintenance burden on your own local modules, you can extract duplicated code into one or more module utilities and import them into your modules. For example, if you have your own custom modules that import a `my_shared_code` library, you can place that into a `./module_utils/my_shared_code.py` file like this:

```
from ansible.module_utils.my_shared_code import MySharedCodeClient
```

When you run `ansible-playbook`, Ansible will merge any files in your local `module_utils` directories into the `ansible.module_utils` namespace in the order defined by the *Ansible search path*.

Naming and finding module utilities

You can generally tell what a module utility does from its name and/or its location. For example, `openstack.py` contains utilities for modules that work with OpenStack instances. Generic utilities (shared code used by many different kinds of modules) live in the `common` subdirectory or in the root directory. Utilities used by a particular set of modules generally live in a sub-directory that mirrors the directory for those modules. For example:

- `lib/ansible/module_utils/urls.py` contains shared code for parsing URLs
- `lib/ansible/module_utils/storage/emc/` contains shared code related to EMC
- `lib/ansible/modules/storage/emc/` contains modules related to EMC

Following this pattern with your own module utilities makes everything easy to find and use.

Standard module utilities

Ansible ships with an extensive library of `module_utils` files. You can find the module utility source code in the `lib/ansible/module_utils` directory under your main Ansible path. We've described the most widely used utilities below. For more details on any specific module utility, please see the [source code for module_utils](#).

注解: LICENSING REQUIREMENTS Ansible enforces the following licensing requirements:

- **Utilities (files in `lib/ansible/module_utils/`) may have one of two licenses:**
 - A file in `module_utils` used **only** for a specific vendor's hardware, provider, or service may be licensed under GPLv3+. Adding a new file under `module_utils` with GPLv3+ needs to be approved by the core team.
 - All other `module_utils` must be licensed under BSD, so GPL-licensed third-party and Galaxy modules can use them.
 - If there's doubt about the appropriate license for a file in `module_utils`, the Ansible Core Team will decide during an Ansible Core Community Meeting.
- All other files shipped with Ansible, including all modules, must be licensed under the GPL license (GPLv3 or later).

-
- `api.py` - Supports generic API modules
 - `basic.py` - General definitions and helper utilities for Ansible modules
 - `common/dict_transformations.py` - Helper functions for dictionary transformations
 - `common/file.py` - Helper functions for working with files
 - `common/text/` - Helper functions for converting and formatting text.
 - `common/parameters.py` - Helper functions for dealing with module parameters
 - `common/sys_info.py` - Functions for getting distribution and platform information
 - `common/validation.py` - Helper functions for validating module parameters against a module argument spec
 - `facts/` - Directory of utilities for modules that return facts. See [PR 23012](#) for more information

- `ismount.py` - Single helper function that fixes `os.path.ismount`
- `json_utils.py` - Utilities for filtering unrelated output around module JSON output, like leading and trailing lines
- `known_hosts.py` - utilities for working with `known_hosts` file
- `network/common/config.py` - Configuration utility functions for use by networking modules
- `network/common/netconf.py` - Definitions and helper functions for modules that use Netconf transport
- `network/common/parsing.py` - Definitions and helper functions for Network modules
- `network/common/network.py` - Functions for running commands on networking devices
- `network/common/utils.py` - Defines commands and comparison operators and other utilises for use in networking modules
- `powershell/` - Directory of definitions and helper functions for Windows PowerShell modules
- `pycompat24.py` - Exception workaround for Python 2.4
- `service.py` - Utilities to enable modules to work with Linux services (placeholder, not in use)
- `shell.py` - Functions to allow modules to create shells and work with shell commands
- `six/__init__.py` - Bundled copy of the [Six Python library](#) to aid in writing code compatible with both Python 2 and Python 3
- `splitter.py` - String splitting and manipulation utilities for working with Jinja2 templates
- `urls.py` - Utilities for working with http and https requests

1.5.24 Developing collections

Collections are a distribution format for Ansible content. You can use collections to package and distribute playbooks, roles, modules, and plugins. You can publish and use collections through [Ansible Galaxy](#).

- For details on how to *use* collections see [Using collections](#).
- For the current development status of Collections and FAQ see [Ansible Collections Community Guide](#).

- *Collection structure*
 - *galaxy.yml*
 - *docs directory*
 - *plugins directory*
 - *roles directory*
 - *playbooks directory*

- *tests directory*
- *Creating a collection skeleton*
- *Creating collections*
 - *Using documentation fragments in collections*
 - *Building collections*
 - *Trying collections locally*
 - *Publishing collections*
 - *Collection versions*
- *Migrating Ansible content to a collection*
 - *BOTMETA.yml*

Collection structure

Collections follow a simple data structure. None of the directories are required unless you have specific content that belongs in one of them. A collection does require a `galaxy.yml` file at the root level of the collection. This file contains all of the metadata that Galaxy and other tools need in order to package, build and publish the collection:

```
collection/
  docs/
  galaxy.yml
  plugins/
    modules/
      module1.py
    inventory/
    .../
  README.md
  roles/
    role1/
    role2/
    .../
  playbooks/
    files/
    vars/
    templates/
    tasks/
  tests/
```

注解:

- Ansible only accepts `.yaml` extensions for `galaxy.yaml`, and `.md` for the `README` file and any files in the `/docs` folder.
 - See the [ansible-collections](#) GitHub Org for examples of collection structure.
 - Not all directories are currently in use. Those are placeholders for future features.
-

galaxy.yaml

A collection must have a `galaxy.yaml` file that contains the necessary information to build a collection artifact. See `collections_galaxy_meta` for details.

docs directory

Put general documentation for the collection here. Keep the specific documentation for plugins and modules embedded as Python docstrings. Use the `docs` folder to describe how to use the roles and plugins the collection provides, role requirements, and so on. Use markdown and do not add subfolders.

Use `ansible-doc` to view documentation for plugins inside a collection:

```
ansible-doc -t lookup my_namespace.my_collection.lookup1
```

The `ansible-doc` command requires the fully qualified collection name (FQCN) to display specific plugin documentation. In this example, `my_namespace` is the namespace and `my_collection` is the collection name within that namespace.

注解: The Ansible collection namespace is defined in the `galaxy.yaml` file and is not equivalent to the GitHub repository name.

plugins directory

Add a ‘per plugin type’ specific subdirectory here, including `module_utils` which is usable not only by modules, but by most plugins by using their FQCN. This is a way to distribute modules, lookups, filters, and so on, without having to import a role in every play.

Vars plugins are unsupported in collections. Cache plugins may be used in collections for fact caching, but are not supported for inventory plugins.

module_utils

When coding with `module_utils` in a collection, the Python import statement needs to take into account the FQCN along with the `ansible_collections` convention. The resulting Python import will look like `from ansible_collections.{namespace}.{collection}.plugins.module_utils.{util} import {something}`

The following example snippets show a Python and PowerShell module using both default Ansible `module_utils` and those provided by a collection. In this example the namespace is `ansible_example`, the collection is `community`. In the Python example the `module_util` in question is called `qradar` such that the FQCN is `ansible_example.community.plugins.module_utils.qradar`:

```
from ansible.module_utils.basic import AnsibleModule
from ansible.module_utils._text import to_text

from ansible.module_utils.six.moves.urllib.parse import urlencode, quote_plus
from ansible.module_utils.six.moves.urllib.error import HTTPError
from ansible_collections.ansible_example.community.plugins.module_utils.qradar import QRadarRequest

argspec = dict(
    name=dict(required=True, type='str'),
    state=dict(choices=['present', 'absent'], required=True),
)

module = AnsibleModule(
    argument_spec=argspec,
    supports_check_mode=True
)

qradar_request = QRadarRequest(
    module,
    headers={"Content-Type": "application/json"},
    not_rest_data_keys=['state']
)
```

Note that importing something from an `__init__.py` file requires using the file name:

```
from ansible_collections.namespace.collection_name.plugins.callback.__init__ import CustomBaseClass
```

In the PowerShell example the `module_util` in question is called `hyperv` such that the FQCN is `ansible_example.community.plugins.module_utils.hyperv`:

```
#!/powershell
#AnsibleRequires -CSharpUtil Ansible.Basic
#AnsibleRequires -PowerShell ansible_collections.ansible_example.community.plugins.
↳ module_utils.hyperv

$spec = @{}
    name = @{ required = $true; type = "str" }
    state = @{ required = $true; choices = @("present", "absent") }
}
$module = [Ansible.Basic.AnsibleModule]::Create($args, $spec)

Invoke-HyperVFunction -Name $module.Params.name

$module.ExitJson()
```

roles directory

Collection roles are mostly the same as existing roles, but with a couple of limitations:

- Role names are now limited to contain only lowercase alphanumeric characters, plus `_` and start with an alpha character.
- Roles in a collection cannot contain plugins any more. Plugins must live in the collection `plugins` directory tree. Each plugin is accessible to all roles in the collection.

The directory name of the role is used as the role name. Therefore, the directory name must comply with the above role name rules. The collection import into Galaxy will fail if a role name does not comply with these rules.

You can migrate ‘traditional roles’ into a collection but they must follow the rules above. You may need to rename roles if they don’t conform. You will have to move or link any role-based plugins to the collection specific directories.

注解: For roles imported into Galaxy directly from a GitHub repository, setting the `role_name` value in the role’s metadata overrides the role name used by Galaxy. For collections, that value is ignored. When importing a collection, Galaxy uses the role directory as the name of the role and ignores the `role_name` metadata value.

playbooks directory

TBD.

tests directory

Ansible Collections are tested much like Ansible itself, by using the *ansible-test* utility which is released as part of Ansible, version 2.9.0 and newer. Because Ansible Collections are tested using the same tooling as Ansible itself, via *ansible-test*, all Ansible developer documentation for testing is applicable for authoring Collections Tests with one key concept to keep in mind.

When reading the *Testing Ansible* documentation, there will be content that applies to running Ansible from source code via a git clone, which is typical of an Ansible developer. However, it's not always typical for an Ansible Collection author to be running Ansible from source but instead from a stable release, and to create Collections it is not necessary to run Ansible from source. Therefore, when references of dealing with *ansible-test* binary paths, command completion, or environment variables are presented throughout the *Testing Ansible* documentation; keep in mind that it is not needed for Ansible Collection Testing because the act of installing the stable release of Ansible containing *ansible-test* is expected to setup those things for you.

Creating a collection skeleton

To start a new collection:

```
collection_dir#> ansible-galaxy collection init my_namespace.my_collection
```

Once the skeleton exists, you can populate the directories with the content you want inside the collection. See [ansible-collections GitHub Org](#) to get a better idea of what you can place inside a collection.

Creating collections

To create a collection:

1. Create a collection skeleton with the `collection init` command. See *Creating a collection skeleton* above.
2. Add your content to the collection.
3. Build the collection into a collection artifact with *ansible-galaxy collection build*.
4. Publish the collection artifact to Galaxy with *ansible-galaxy collection publish*.

A user can then install your collection on their systems.

Currently the `ansible-galaxy collection` command implements the following sub commands:

- **init:** Create a basic collection skeleton based on the default template included with Ansible or your own template.
- **build:** Create a collection artifact that can be uploaded to Galaxy or your own repository.
- **publish:** Publish a built collection artifact to Galaxy.

- `install`: Install one or more collections.

To learn more about the `ansible-galaxy` cli tool, see the `ansible-galaxy` man page.

Using documentation fragments in collections

To include documentation fragments in your collection:

1. Create the documentation fragment: `plugins/doc_fragments/fragment_name`.
2. Refer to the documentation fragment with its FQCN.

```
extends_documentation_fragment:
- community.kubernetes.k8s_name_options
- community.kubernetes.k8s_auth_options
- community.kubernetes.k8s_resource_options
- community.kubernetes.k8s_scale_options
```

Documentation fragments covers the basics for documentation fragments. The `kubernetes` collection includes a complete example.

You can also share documentation fragments across collections with the FQCN.

Building collections

To build a collection, run `ansible-galaxy collection build` from inside the root directory of the collection:

```
collection_dir#> ansible-galaxy collection build
```

This creates a tarball of the built collection in the current directory which can be uploaded to Galaxy.:

```
my_collection/
  galaxy.yml
  ...
  my_namespace-my_collection-1.0.0.tar.gz
  ...
```

注解:

- Certain files and folders are excluded when building the collection artifact. See *Ignoring files and folders* to exclude other files you would not wish to distribute.
- If you used the now-deprecated **Mazer** tool for any of your collections, delete any and all files it added to your `releases/` directory before you build your collection with `ansible-galaxy`.

- The current Galaxy maximum tarball size is 2 MB.
-

This tarball is mainly intended to upload to Galaxy as a distribution method, but you can use it directly to install the collection on target systems.

Ignoring files and folders

By default the build step will include all the files in the collection directory in the final build artifact except for the following:

- `galaxy.yml`
- `*.pyc`
- `*.retry`
- `tests/output`
- previously built artifacts in the root directory
- Various version control directories like `.git/`

To exclude other files and folders when building the collection, you can set a list of file glob-like patterns in the `build_ignore` key in the collection's `galaxy.yml` file. These patterns use the following special characters for wildcard matching:

- `*`: Matches everything
- `?`: Matches any single character
- `[seq]`: Matches and character in seq
- `[!seq]`: Matches any character not in seq

For example, if you wanted to exclude the `sensitive` folder within the `playbooks` folder as well any `.tar.gz` archives you can set the following in your `galaxy.yml` file:

```
build_ignore:
- playbooks/sensitive
- '*.tar.gz'
```

注解: This feature is only supported when running `ansible-galaxy collection build` with Ansible 2.10 or newer.

Trying collections locally

You can try your collection locally by installing it from the tarball. The following will enable an adjacent playbook to access the collection:

```
ansible-galaxy collection install my_namespace-my_collection-1.0.0.tar.gz -p ./  
↳collections
```

You should use one of the values configured in `COLLECTIONS_PATHS` for your path. This is also where Ansible itself will expect to find collections when attempting to use them. If you don't specify a path value, `ansible-galaxy collection install` installs the collection in the first path defined in `COLLECTIONS_PATHS`, which by default is `~/.ansible/collections`.

Next, try using the local collection inside a playbook. For examples and more details see [Using collections](#)

Publishing collections

You can publish collections to Galaxy using the `ansible-galaxy collection publish` command or the Galaxy UI itself. You need a namespace on Galaxy to upload your collection. See [Galaxy namespaces](#) on the Galaxy docsite for details.

注解: Once you upload a version of a collection, you cannot delete or modify that version. Ensure that everything looks okay before you upload it.

Getting your API token

To upload your collection to Galaxy, you must first obtain an API token (`--token` in the `ansible-galaxy` CLI command or `token` in the `ansible.cfg` file under the `galaxy_server` section). The API token is a secret token used to protect your content.

To get your API token:

- For Galaxy, go to the [Galaxy profile preferences](#) page and click *API Key*.
- For Automation Hub, go to <https://cloud.redhat.com/ansible/automation-hub/token/> and click *Load token* from the version dropdown.

Storing or using your API token

Once you have retrieved your API token, you can store or use the token for collections in two ways:

- Pass the token to the `ansible-galaxy` command using the `--token`.

- Specify the token within a Galaxy server list in your `ansible.cfg` file.

Using the token argument

You can use the `--token` argument with the `ansible-galaxy` command (in conjunction with the `--server` argument or `GALAXY_SERVER` setting in your `ansible.cfg` file). You cannot use `apt-key` with any servers defined in your *Galaxy server list*.

```
ansible-galaxy collection publish ./geerlingguy-collection-1.2.3.tar.gz --token=<key↵
↵goes here>
```

Specify the token within a Galaxy server list

With this option, you configure one or more servers for Galaxy in your `ansible.cfg` file under the `galaxy_server_list` section. For each server, you also configure the token.

```
[galaxy]
server_list = release_galaxy

[galaxy_server.release_galaxy]
url=https://galaxy.ansible.com/
token=my_token
```

See *Configuring the ansible-galaxy client* for complete details.

Upload using ansible-galaxy

注解: By default, `ansible-galaxy` uses <https://galaxy.ansible.com> as the Galaxy server (as listed in the `ansible.cfg` file under `galaxy_server`). If you are only publishing your collection to Ansible Galaxy, you do not need any further configuration. If you are using Red Hat Automation Hub or any other Galaxy server, see *Configuring the ansible-galaxy client*.

To upload the collection artifact with the `ansible-galaxy` command:

```
ansible-galaxy collection publish path/to/my_namespace-my_collection-1.0.0.tar.gz
```

注解: The above command assumes you have retrieved and stored your API token as part of a Galaxy server list. See *Getting your API token* for details.

The `ansible-galaxy collection publish` command triggers an import process, just as if you uploaded the collection through the Galaxy website. The command waits until the import process completes before reporting the status back. If you wish to continue without waiting for the import result, use the `--no-wait` argument and manually look at the import progress in your [My Imports](#) page.

Upload a collection from the Galaxy website

To upload your collection artifact directly on Galaxy:

1. Go to the [My Content](#) page, and click the **Add Content** button on one of your namespaces.
2. From the **Add Content** dialogue, click **Upload New Collection**, and select the collection archive file from your local filesystem.

When uploading collections it doesn't matter which namespace you select. The collection will be uploaded to the namespace specified in the collection metadata in the `galaxy.yml` file. If you're not an owner of the namespace, the upload request will fail.

Once Galaxy uploads and accepts a collection, you will be redirected to the **My Imports** page, which displays output from the import process, including any errors or warnings about the metadata and content contained in the collection.

Collection versions

Once you upload a version of a collection, you cannot delete or modify that version. Ensure that everything looks okay before uploading. The only way to change a collection is to release a new version. The latest version of a collection (by highest version number) will be the version displayed everywhere in Galaxy; however, users will still be able to download older versions.

Collection versions use [Semantic Versioning](#) for version numbers. Please read the official documentation for details and examples. In summary:

- Increment major (for example: `x` in `x.y.z`) version number for an incompatible API change.
- Increment minor (for example: `y` in `x.y.z`) version number for new functionality in a backwards compatible manner.
- Increment patch (for example: `z` in `x.y.z`) version number for backwards compatible bug fixes.

Migrating Ansible content to a collection

You can experiment with migrating existing modules into a collection using the `content_collector` tool. The `content_collector` is a playbook that helps you migrate content from an Ansible distribution into a collection.

警告: This tool is in active development and is provided only for experimentation and feedback at this point.

See the [content_collector README](#) for full details and usage guidelines.

BOTMETA.yml

The [BOTMETA.yml](#) is the source of truth for: * ansibullbot * the docs build for collections-based modules. Ansibullbot will know how to redirect existing issues and PRs to the new repo. The build process for docs.ansible.com will know where to find the module docs.

```
$modules/monitoring/grafana/grafana_plugin.py:
    migrated_to: community.grafana
$modules/monitoring/grafana/grafana_dashboard.py:
    migrated_to: community.grafana
$modules/monitoring/grafana/grafana_datasource.py:
    migrated_to: community.grafana
$plugins/callback/grafana_annotations.py:
    maintainers: $team_grafana
    labels: monitoring grafana
    migrated_to: community.grafana
$plugins/doc_fragments/grafana.py:
    maintainers: $team_grafana
    labels: monitoring grafana
    migrated_to: community.grafana
```

Example PR

- The `migrated_to:` key must be added explicitly for every *file*. You cannot add `migrated_to` at the directory level. This is to allow module and plugin webdocs to be redirected to the new collection docs.
- `migrated_to:` MUST be added for every:
 - module
 - plugin
 - module_utils
 - contrib/inventory script
- You do NOT need to add `migrated_to` for:
 - Unit tests
 - Integration tests

- ReStructured Text docs (anything under `docs/docsite/rst/`)
- Files that never existed in `ansible/ansible:devel`

参见:

Using collections Learn how to install and use collections.

`collections__galaxy__meta` Understand the collections metadata structure.

Ansible module development: getting started Learn about how to write Ansible modules

Mailing List The development mailing list

`irc.freenode.net` #ansible IRC chat channel

1.5.25 Ansible architecture

Ansible is a radically simple IT automation engine that automates cloud provisioning, configuration management, application deployment, intra-service orchestration, and many other IT needs.

Being designed for multi-tier deployments since day one, Ansible models your IT infrastructure by describing how all of your systems inter-relate, rather than just managing one system at a time.

It uses no agents and no additional custom security infrastructure, so it's easy to deploy - and most importantly, it uses a very simple language (YAML, in the form of Ansible Playbooks) that allow you to describe your automation jobs in a way that approaches plain English.

In this section, we'll give you a really quick overview of how Ansible works so you can see how the pieces fit together.

- *Modules*
- *Module utilities*
- *Plugins*
- *Inventory*
- *Playbooks*
- *The Ansible search path*

Modules

Ansible works by connecting to your nodes and pushing out scripts called “Ansible modules” to them. Most modules accept parameters that describe the desired state of the system. Ansible then executes these modules (over SSH by default), and removes them when finished. Your library of modules can reside on any machine, and there are no servers, daemons, or databases required.

You can *write your own modules*, though you should first consider *whether you should*. Typically you'll work with your favorite terminal program, a text editor, and probably a version control system to keep track of changes to your content. You may write specialized modules in any language that can return JSON (Ruby, Python, bash, etc).

Module utilities

When multiple modules use the same code, Ansible stores those functions as module utilities to minimize duplication and maintenance. For example, the code that parses URLs is `lib/ansible/module_utils/url.py`. You can *write your own module utilities* as well. Module utilities may only be written in Python or in PowerShell.

Plugins

Plugins augment Ansible's core functionality. While modules execute on the target system in separate processes (usually that means on a remote system), plugins execute on the control node within the `/usr/bin/ansible` process. Plugins offer options and extensions for the core features of Ansible - transforming data, logging output, connecting to inventory, and more. Ansible ships with a number of handy plugins, and you can easily *write your own*. For example, you can write an *inventory plugin* to connect to any datasource that returns JSON. Plugins must be written in Python.

Inventory

By default, Ansible represents the machines it manages in a file (INI, YAML, etc.) that puts all of your managed machines in groups of your own choosing.

To add new machines, there is no additional SSL signing server involved, so there's never any hassle deciding why a particular machine didn't get linked up due to obscure NTP or DNS issues.

If there's another source of truth in your infrastructure, Ansible can also connect to that. Ansible can draw inventory, group, and variable information from sources like EC2, Rackspace, OpenStack, and more.

Here's what a plain text inventory file looks like:

```
---
[webservers]
www1.example.com
www2.example.com

[dbservers]
db0.example.com
db1.example.com
```

Once inventory hosts are listed, variables can be assigned to them in simple text files (in a subdirectory called ‘group_vars/’ or ‘host_vars/’ or directly in the inventory file.

Or, as already mentioned, use a dynamic inventory to pull your inventory from data sources like EC2, Rackspace, or OpenStack.

Playbooks

Playbooks can finely orchestrate multiple slices of your infrastructure topology, with very detailed control over how many machines to tackle at a time. This is where Ansible starts to get most interesting.

Ansible’ s approach to orchestration is one of finely-tuned simplicity, as we believe your automation code should make perfect sense to you years down the road and there should be very little to remember about special syntax or features.

Here’ s what a simple playbook looks like:

```
---
- hosts: webservers
  serial: 5 # update 5 machines at a time
  roles:
    - common
    - webapp

- hosts: content_servers
  roles:
    - common
    - content
```

The Ansible search path

Modules, module utilities, plugins, playbooks, and roles can live in multiple locations. If you write your own code to extend Ansible’ s core features, you may have multiple files with similar or the same names in different locations on your Ansible control node. The search path determines which of these files Ansible will discover and use on any given playbook run.

Ansible’ s search path grows incrementally over a run. As Ansible finds each playbook and role included in a given run, it appends any directories related to that playbook or role to the search path. Those directories remain in scope for the duration of the run, even after the playbook or role has finished executing. Ansible loads modules, module utilities, and plugins in this order:

1. Directories adjacent to a playbook specified on the command line. If you run Ansible with `ansible-playbook /path/to/play.yml`, Ansible appends these directories if they exist:

```
/path/to/modules
/path/to/module_utils
/path/to/plugins
```

2. Directories adjacent to a playbook that is statically imported by a playbook specified on the command line. If `play.yml` includes `- import_playbook: /path/to/subdir/play1.yml`, Ansible appends these directories if they exist:

```
/path/to/subdir/modules
/path/to/subdir/module_utils
/path/to/subdir/plugins
```

3. Subdirectories of a role directory referenced by a playbook. If `play.yml` runs `myrole`, Ansible appends these directories if they exist:

```
/path/to/roles/myrole/modules
/path/to/roles/myrole/module_utils
/path/to/roles/myrole/plugins
```

4. Directories specified as default paths in `ansible.cfg` or by the related environment variables, including the paths for the various plugin types. See `ansible_configuration_settings` for more information. Sample `ansible.cfg` fields:

```
DEFAULT_MODULE_PATH
DEFAULT_MODULE_UTILS_PATH
DEFAULT_CACHE_PLUGIN_PATH
DEFAULT_FILTER_PLUGIN_PATH
```

Sample environment variables:

```
ANSIBLE_LIBRARY
ANSIBLE_MODULE_UTILS
ANSIBLE_CACHE_PLUGINS
ANSIBLE_FILTER_PLUGINS
```

5. The standard directories that ship as part of the Ansible distribution.

警告: Modules, module utilities, and plugins in user-specified directories will override the standard versions. This includes some files with generic names. For example, if you have a file named `basic.py` in a user-specified directory, it will override the standard `ansible.module_utils.basic`.

If you have more than one module, module utility, or plugin with the same name in different user-

specified directories, the order of commands at the command line and the order of includes and roles in each play will affect which one is found and used on that particular play.

1.6 Public Cloud Guides

The guides in this section cover using Ansible with a range of public cloud platforms. They explore particular use cases in greater depth and provide a more “top-down” explanation of some basic features.

1.6.1 Alibaba Cloud Compute Services Guide

Introduction

Ansible contains several modules for controlling and managing Alibaba Cloud Compute Services (Alicloud). This guide explains how to use the Alicloud Ansible modules together.

All Alicloud modules require `footmark` - install it on your control machine with `pip install footmark`.

Cloud modules, including Alicloud modules, execute on your local machine (the control machine) with `connection: local`, rather than on remote machines defined in your hosts.

Normally, you’ ll use the following pattern for plays that provision Alicloud resources:

```
- hosts: localhost
  connection: local
  vars:
    - ...
  tasks:
    - ...
```

Authentication

You can specify your Alicloud authentication credentials (access key and secret key) by passing them as environment variables or by storing them in a vars file.

To pass authentication credentials as environment variables:

```
export ALICLOUD_ACCESS_KEY='Alicloud123'
export ALICLOUD_SECRET_KEY='AlicloudSecret123'
```

To store authentication credentials in a vars_file, encrypt them with *Ansible Vault* to keep them secure, then list them:

```
---
alicloud_access_key: "--REMOVED--"
alicloud_secret_key: "--REMOVED--"
```

Note that if you store your credentials in a `vars_file`, you need to refer to them in each Alicloud module. For example:

```
- ali_instance:
    alicloud_access_key: "{{alicloud_access_key}}"
    alicloud_secret_key: "{{alicloud_secret_key}}"
    image_id: "..."
```

Provisioning

Alicloud modules create Alicloud ECS instances, disks, virtual private clouds, virtual switches, security groups and other resources.

You can use the `count` parameter to control the number of resources you create or terminate. For example, if you want exactly 5 instances tagged `NewECS`, set the `count` of instances to 5 and the `count_tag` to `NewECS`, as shown in the last task of the example playbook below. If there are no instances with the tag `NewECS`, the task creates 5 new instances. If there are 2 instances with that tag, the task creates 3 more. If there are 8 instances with that tag, the task terminates 3 of those instances.

If you do not specify a `count_tag`, the task creates the number of instances you specify in `count` with the `instance_name` you provide.

```
# alicloud_setup.yml

- hosts: localhost
  connection: local

  tasks:

    - name: Create VPC
      ali_vpc:
        cidr_block: '{{ cidr_block }}'
        vpc_name: new_vpc
        register: created_vpc

    - name: Create VSwitch
      ali_vswitch:
        alicloud_zone: '{{ alicloud_zone }}'
```

(下页继续)

```

    cidr_block: '{{ vsw_cidr }}'
    vswitch_name: new_vswitch
    vpc_id: '{{ created_vpc.vpc.id }}'
    register: created_vsw

- name: Create security group
  ali_security_group:
    name: new_group
    vpc_id: '{{ created_vpc.vpc.id }}'
    rules:
      - proto: tcp
        port_range: 22/22
        cidr_ip: 0.0.0.0/0
        priority: 1
    rules_egress:
      - proto: tcp
        port_range: 80/80
        cidr_ip: 192.168.0.54/32
        priority: 1
    register: created_group

- name: Create a set of instances
  ali_instance:
    security_groups: '{{ created_group.group_id }}'
    instance_type: ecs.n4.small
    image_id: "{{ ami_id }}"
    instance_name: "My-new-instance"
    instance_tags:
      Name: NewECS
      Version: 0.0.1
    count: 5
    count_tag:
      Name: NewECS
    allocate_public_ip: true
    max_bandwidth_out: 50
    vswitch_id: '{{ created_vsw.vswitch.id }}'
    register: create_instance

```

In the example playbook above, data about the vpc, vswitch, group, and instances created by this playbook are saved in the variables defined by the “register” keyword in each task.

Each Alicloud module offers a variety of parameter options. Not all options are demonstrated in the above example. See each individual module for further details and examples.

1.6.2 Amazon Web Services Guide

Introduction

Ansible contains a number of modules for controlling Amazon Web Services (AWS). The purpose of this section is to explain how to put Ansible modules together (and use inventory scripts) to use Ansible in AWS context.

Requirements for the AWS modules are minimal.

All of the modules require and are tested against recent versions of boto. You' ll need this Python module installed on your control machine. Boto can be installed from your OS distribution or python' s “pip install boto” .

Whereas classically ansible will execute tasks in its host loop against multiple remote machines, most cloud-control steps occur on your local machine with reference to the regions to control.

In your playbook steps we' ll typically be using the following pattern for provisioning steps:

```
- hosts: localhost
  gather_facts: False
  tasks:
    - ...
```

Authentication

Authentication with the AWS-related modules is handled by either specifying your access and secret key as ENV variables or module arguments.

For environment variables:

```
export AWS_ACCESS_KEY_ID='AK123'
export AWS_SECRET_ACCESS_KEY='abc123'
```

For storing these in a vars_file, ideally encrypted with ansible-vault:

```
---
ec2_access_key: "--REMOVED--"
ec2_secret_key: "--REMOVED--"
```

Note that if you store your credentials in vars_file, you need to refer to them in each AWS-module. For example:

```
- ec2
  aws_access_key: "{{ec2_access_key}}"
  aws_secret_key: "{{ec2_secret_key}}"
  image: "..."
```

Provisioning

The ec2 module provisions and de-provisions instances within EC2.

An example of making sure there are only 5 instances tagged ‘Demo’ in EC2 follows.

In the example below, the “exact_count” of instances is set to 5. This means if there are 0 instances already existing, then 5 new instances would be created. If there were 2 instances, only 3 would be created, and if there were 8 instances, 3 instances would be terminated.

What is being counted is specified by the “count_tag” parameter. The parameter “instance_tags” is used to apply tags to the newly created instance.:

```
# demo_setup.yml

- hosts: localhost
  gather_facts: False

  tasks:

    - name: Provision a set of instances
      ec2:
        key_name: my_key
        group: test
        instance_type: t2.micro
        image: "{{ ami_id }}"
        wait: true
        exact_count: 5
        count_tag:
          Name: Demo
        instance_tags:
          Name: Demo
      register: ec2
```

The data about what instances are created is being saved by the “register” keyword in the variable named “ec2” .

From this, we’ ll use the add_host module to dynamically create a host group consisting of these new

instances. This facilitates performing configuration actions on the hosts immediately in a subsequent task.:

```
# demo_setup.yml

- hosts: localhost
  gather_facts: False

  tasks:

    - name: Provision a set of instances
      ec2:
        key_name: my_key
        group: test
        instance_type: t2.micro
        image: "{{ ami_id }}"
        wait: true
        exact_count: 5
        count_tag:
          Name: Demo
        instance_tags:
          Name: Demo
        register: ec2

    - name: Add all instance public IPs to host group
      add_host: hostname={{ item.public_ip }} groups=ec2hosts
      loop: "{{ ec2.instances }}"
```

With the host group now created, a second play at the bottom of the same provisioning playbook file might now have some configuration steps:

```
# demo_setup.yml

- name: Provision a set of instances
  hosts: localhost
  # ... AS ABOVE ...

- hosts: ec2hosts
  name: configuration play
  user: ec2-user
  gather_facts: true

  tasks:
```

(下页继续)

(续上页)

```
- name: Check NTP service
  service: name=ntpd state=started
```

Security Groups

Security groups on AWS are stateful. The response of a request from your instance is allowed to flow in regardless of inbound security group rules and vice-versa. In case you only want allow traffic with AWS S3 service, you need to fetch the current IP ranges of AWS S3 for one region and apply them as an egress rule.:

```
- name: fetch raw ip ranges for aws s3
  set_fact:
    raw_s3_ranges: "{{ lookup('aws_service_ip_ranges', region='eu-central-1', service='S3
↪', wantlist=True) }}"

- name: prepare list structure for ec2_group module
  set_fact:
    s3_ranges: "{{ s3_ranges | default([]) + [{'proto': 'all', 'cidr_ip': item, 'rule_
↪desc': 'S3 Service IP range'}] }}"
    loop: "{{ raw_s3_ranges }}"

- name: set S3 IP ranges to egress rules
  ec2_group:
    name: aws_s3_ip_ranges
    description: allow outgoing traffic to aws S3 service
    region: eu-central-1
    state: present
    vpc_id: vpc-123456
    purge_rules: true
    purge_rules_egress: true
    rules: []
    rules_egress: "{{ s3_ranges }}"
    tags:
      Name: aws_s3_ip_ranges
```

Host Inventory

Once your nodes are spun up, you' ll probably want to talk to them again. With a cloud setup, it' s best to not maintain a static list of cloud hostnames in text files. Rather, the best way to handle this is to use the ec2 dynamic inventory script. See [动态 Inventory 清单配置](#).

This will also dynamically select nodes that were even created outside of Ansible, and allow Ansible to manage them.

See [动态 Inventory 清单配置](#) for how to use this, then return to this chapter.

Tags And Groups And Variables

When using the ec2 inventory script, hosts automatically appear in groups based on how they are tagged in EC2.

For instance, if a host is given the “class” tag with the value of “webserver” , it will be automatically discoverable via a dynamic group like so:

```
- hosts: tag_class_webserver
  tasks:
    - ping
```

Using this philosophy can be a great way to keep systems separated by the function they perform.

In this example, if we wanted to define variables that are automatically applied to each machine tagged with the ‘class’ of ‘webserver’ , ‘group_vars’ in ansible can be used. See [编排主机和组变量](#).

Similar groups are available for regions and other classifications, and can be similarly assigned variables using the same mechanism.

Autoscaling with Ansible Pull

Amazon Autoscaling features automatically increase or decrease capacity based on load. There are also Ansible modules shown in the cloud documentation that can configure autoscaling policy.

When nodes come online, it may not be sufficient to wait for the next cycle of an ansible command to come along and configure that node.

To do this, pre-bake machine images which contain the necessary ansible-pull invocation. Ansible-pull is a command line tool that fetches a playbook from a git server and runs it locally.

One of the challenges of this approach is that there needs to be a centralized way to store data about the results of pull commands in an autoscaling context. For this reason, the autoscaling solution provided below in the next section can be a better approach.

Read [ansible-pull](#) for more information on pull-mode playbooks.

Autoscaling with Ansible Tower

[Red Hat Ansible Tower](#) also contains a very nice feature for auto-scaling use cases. In this mode, a simple curl script can call a defined URL and the server will “dial out” to the requester and configure an instance

that is spinning up. This can be a great way to reconfigure ephemeral nodes. See the Tower install and product documentation for more details.

A benefit of using the callback in Tower over pull mode is that job results are still centrally recorded and less information has to be shared with remote hosts.

Ansible With (And Versus) CloudFormation

CloudFormation is a Amazon technology for defining a cloud stack as a JSON or YAML document.

Ansible modules provide an easier to use interface than CloudFormation in many examples, without defining a complex JSON/YAML document. This is recommended for most users.

However, for users that have decided to use CloudFormation, there is an Ansible module that can be used to apply a CloudFormation template to Amazon.

When using Ansible with CloudFormation, typically Ansible will be used with a tool like Packer to build images, and CloudFormation will launch those images, or ansible will be invoked through user data once the image comes online, or a combination of the two.

Please see the examples in the Ansible CloudFormation module for more details.

AWS Image Building With Ansible

Many users may want to have images boot to a more complete configuration rather than configuring them entirely after instantiation. To do this, one of many programs can be used with Ansible playbooks to define and upload a base image, which will then get its own AMI ID for usage with the ec2 module or other Ansible AWS modules such as ec2_asg or the cloudformation module. Possible tools include Packer, aminator, and Ansible's ec2_ami module.

Generally speaking, we find most users using Packer.

See the Packer documentation of the [Ansible local Packer provisioner](#) and [Ansible remote Packer provisioner](#).

If you do not want to adopt Packer at this time, configuring a base-image with Ansible after provisioning (as shown above) is acceptable.

Next Steps: Explore Modules

Ansible ships with lots of modules for configuring a wide array of EC2 services. Browse the “Cloud” category of the module documentation for a full list with examples.

参见:

all_modules All the documentation for Ansible modules

Working With Playbooks An introduction to playbooks

Delegation, Rolling Updates, and Local Actions Delegation, useful for working with load balancers, clouds, and locally executed steps.

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

1.6.3 CloudStack Cloud Guide

Introduction

The purpose of this section is to explain how to put Ansible modules together to use Ansible in a CloudStack context. You will find more usage examples in the details section of each module.

Ansible contains a number of extra modules for interacting with CloudStack based clouds. All modules support check mode, are designed to be idempotent, have been created and tested, and are maintained by the community.

注解: Some of the modules will require domain admin or root admin privileges.

Prerequisites

Prerequisites for using the CloudStack modules are minimal. In addition to Ansible itself, all of the modules require the python library **cs** <https://pypi.org/project/cs/>

You'll need this Python module installed on the execution host, usually your workstation.

```
$ pip install cs
```

Or alternatively starting with Debian 9 and Ubuntu 16.04:

```
$ sudo apt install python-cs
```

注解: **cs** also includes a command line interface for ad-hoc interaction with the CloudStack API e.g. `$ cs listVirtualMachines state=Running`.

Limitations and Known Issues

VPC support has been improved since Ansible 2.3 but is still not yet fully implemented. The community is working on the VPC integration.

Credentials File

You can pass credentials and the endpoint of your cloud as module arguments, however in most cases it is a far less work to store your credentials in the cloudstack.ini file.

The python library cs looks for the credentials file in the following order (last one wins):

- A `.cloudstack.ini` (note the dot) file in the home directory.
- A `CLOUDSTACK_CONFIG` environment variable pointing to an `.ini` file.
- A `cloudstack.ini` (without the dot) file in the current working directory, same directory as your playbooks are located.

The structure of the ini file must look like this:

```
$ cat $HOME/.cloudstack.ini
[cloudstack]
endpoint = https://cloud.example.com/client/api
key = api key
secret = api secret
timeout = 30
```

注解: The section `[cloudstack]` is the default section. `CLOUDSTACK_REGION` environment variable can be used to define the default section.

2.4 新版功能.

The ENV variables support `CLOUDSTACK_*` as written in the documentation of the library `cs`, like e.g `CLOUDSTACK_TIMEOUT`, `CLOUDSTACK_METHOD`, etc. has been implemented into Ansible. It is even possible to have some incomplete config in your `cloudstack.ini`:

```
$ cat $HOME/.cloudstack.ini
[cloudstack]
endpoint = https://cloud.example.com/client/api
timeout = 30
```

and fulfill the missing data by either setting ENV variables or tasks params:

```
---
- name: provision our VMs
  hosts: cloud-vm
  tasks:
    - name: ensure VMs are created and running
```

(下页继续)

(续上页)

```

delegate_to: localhost
cs_instance:
  api_key: your api key
  api_secret: your api secret
  ...

```

Regions

If you use more than one CloudStack region, you can define as many sections as you want and name them as you like, e.g.:

```

$ cat $HOME/.cloudstack.ini
[exoscale]
endpoint = https://api.exoscale.ch/compute
key = api key
secret = api secret

[example_cloud_one]
endpoint = https://cloud-one.example.com/client/api
key = api key
secret = api secret

[example_cloud_two]
endpoint = https://cloud-two.example.com/client/api
key = api key
secret = api secret

```

提示: Sections can also be used to for login into the same region using different accounts.

By passing the argument `api_region` with the CloudStack modules, the region wanted will be selected.

```

- name: ensure my ssh public key exists on Exoscale
  cs_sshkeypair:
    name: my-ssh-key
    public_key: "{{ lookup('file', '~/.ssh/id_rsa.pub') }}"
    api_region: exoscale
    delegate_to: localhost

```

Or by looping over a regions list if you want to do the task in every region:

```
- name: ensure my ssh public key exists in all CloudStack regions
  local_action: cs_sshkeypair
    name: my-ssh-key
    public_key: "{{ lookup('file', '~/.ssh/id_rsa.pub') }}"
    api_region: "{{ item }}"
  loop:
    - exoscale
    - example_cloud_one
    - example_cloud_two
```

Environment Variables

2.3 新版功能.

Since Ansible 2.3 it is possible to use environment variables for domain (CLOUDSTACK_DOMAIN), account (CLOUDSTACK_ACCOUNT), project (CLOUDSTACK_PROJECT), VPC (CLOUDSTACK_VPC) and zone (CLOUDSTACK_ZONE). This simplifies the tasks by not repeating the arguments for every tasks.

Below you see an example how it can be used in combination with Ansible' s block feature:

```
- hosts: cloud-vm
  tasks:
    - block:
      - name: ensure my ssh public key
        cs_sshkeypair:
          name: my-ssh-key
          public_key: "{{ lookup('file', '~/.ssh/id_rsa.pub') }}"

      - name: ensure my ssh public key
        cs_instance:
          display_name: "{{ inventory_hostname_short }}"
          template: Linux Debian 7 64-bit 20GB Disk
          service_offering: "{{ cs_offering }}"
          ssh_key: my-ssh-key
          state: running

    delegate_to: localhost
    environment:
      CLOUDSTACK_DOMAIN: root/customers
      CLOUDSTACK_PROJECT: web-app
      CLOUDSTACK_ZONE: sf-1
```

注解: You are still able overwrite the environment variables using the module arguments, e.g. `zone: sf-2`

注解: Unlike `CLOUDSTACK_REGION` these additional environment variables are ignored in the CLI `cs`.

Use Cases

The following should give you some ideas how to use the modules to provision VMs to the cloud. As always, there isn't only one way to do it. But as always: keep it simple for the beginning is always a good start.

Use Case: Provisioning in a Advanced Networking CloudStack setup

Our CloudStack cloud has an advanced networking setup, we would like to provision web servers, which get a static NAT and open firewall ports 80 and 443. Further we provision database servers, to which we do not give any access to. For accessing the VMs by SSH we use a SSH jump host.

This is how our inventory looks like:

```
[cloud-vm:children]
webserver
db-server
jumphost

[webserver]
web-01.example.com  public_ip=198.51.100.20
web-02.example.com  public_ip=198.51.100.21

[db-server]
db-01.example.com
db-02.example.com

[jumphost]
jump.example.com  public_ip=198.51.100.22
```

As you can see, the public IPs for our web servers and jumphost has been assigned as variable `public_ip` directly in the inventory.

To configure the jumphost, web servers and database servers, we use `group_vars`. The `group_vars` directory contains 4 files for configuration of the groups: `cloud-vm`, `jumphost`, `webserver` and `db-server`. The `cloud-vm` is there for specifying the defaults of our cloud infrastructure.

```
# file: group_vars/cloud-vm
---
cs_offering: Small
cs_firewall: []
```

Our database servers should get more CPU and RAM, so we define to use a **Large** offering for them.

```
# file: group_vars/db-server
---
cs_offering: Large
```

The web servers should get a **Small** offering as we would scale them horizontally, which is also our default offering. We also ensure the known web ports are opened for the world.

```
# file: group_vars/webserver
---
cs_firewall:
  - { port: 80 }
  - { port: 443 }
```

Further we provision a jump host which has only port 22 opened for accessing the VMs from our office IPv4 network.

```
# file: group_vars/jumphost
---
cs_firewall:
  - { port: 22, cidr: "17.17.17.0/24" }
```

Now to the fun part. We create a playbook to create our infrastructure we call it **infra.yml**:

```
# file: infra.yml
---
- name: provision our VMs
  hosts: cloud-vm
  tasks:
    - name: run all enclosed tasks from localhost
      delegate_to: localhost
      block:
        - name: ensure VMs are created and running
          cs_instance:
            name: "{{ inventory_hostname_short }}"
            template: Linux Debian 7 64-bit 20GB Disk
```

(下页继续)

(续上页)

```

    service_offering: "{{ cs_offering }}"
    state: running

- name: ensure firewall ports opened
  cs_firewall:
    ip_address: "{{ public_ip }}"
    port: "{{ item.port }}"
    cidr: "{{ item.cidr | default('0.0.0.0/0') }}"
    loop: "{{ cs_firewall }}"
    when: public_ip is defined

- name: ensure static NATs
  cs_staticnat: vm="{{ inventory_hostname_short }}" ip_address="{{ public_ip }}"
  when: public_ip is defined

```

In the above play we defined 3 tasks and use the group `cloud-vm` as target to handle all VMs in the cloud but instead SSH to these VMs, we use `delegate_to: localhost` to execute the API calls locally from our workstation.

In the first task, we ensure we have a running VM created with the Debian template. If the VM is already created but stopped, it would just start it. If you like to change the offering on an existing VM, you must add `force: yes` to the task, which would stop the VM, change the offering and start the VM again.

In the second task we ensure the ports are opened if we give a public IP to the VM.

In the third task we add static NAT to the VMs having a public IP defined.

注解: The public IP addresses must have been acquired in advance, also see `cs_ip_address`

注解: For some modules, e.g. `cs_sshkeypair` you usually want this to be executed only once, not for every VM. Therefore you would make a separate play for it targeting `localhost`. You find an example in the use cases below.

Use Case: Provisioning on a Basic Networking CloudStack setup

A basic networking CloudStack setup is slightly different: Every VM gets a public IP directly assigned and security groups are used for access restriction policy.

This is how our inventory looks like:

```
[cloud-vm:children]
webserver

[webserver]
web-01.example.com
web-02.example.com
```

The default for your VMs looks like this:

```
# file: group_vars/cloud-vm
---
cs_offering: Small
cs_securitygroups: [ 'default' ]
```

Our webserver will also be in security group **web**:

```
# file: group_vars/webserver
---
cs_securitygroups: [ 'default', 'web' ]
```

The playbook looks like the following:

```
# file: infra.yaml
---
- name: cloud base setup
  hosts: localhost
  tasks:
    - name: upload ssh public key
      cs_sshkeypair:
        name: defaultkey
        public_key: "{{ lookup('file', '~/.ssh/id_rsa.pub') }}"

    - name: ensure security groups exist
      cs_securitygroup:
        name: "{{ item }}"
      loop:
        - default
        - web

    - name: add inbound SSH to security group default
      cs_securitygroup_rule:
        security_group: default
```

(下页继续)

(续上页)

```

        start_port: "{{ item }}"
        end_port: "{{ item }}"
    loop:
        - 22

- name: add inbound TCP rules to security group web
  cs_securitygroup_rule:
    security_group: web
    start_port: "{{ item }}"
    end_port: "{{ item }}"
  loop:
    - 80
    - 443

- name: install VMs in the cloud
  hosts: cloud-vm
  tasks:
    - delegate_to: localhost
      block:
        - name: create and run VMs on CloudStack
          cs_instance:
            name: "{{ inventory_hostname_short }}"
            template: Linux Debian 7 64-bit 20GB Disk
            service_offering: "{{ cs_offering }}"
            security_groups: "{{ cs_securitygroups }}"
            ssh_key: defaultkey
            state: Running
            register: vm

        - name: show VM IP
          debug: msg="VM {{ inventory_hostname }} {{ vm.default_ip }}"

        - name: assign IP to the inventory
          set_fact: ansible_ssh_host={{ vm.default_ip }}

        - name: waiting for SSH to come up
          wait_for: port=22 host={{ vm.default_ip }} delay=5

```

In the first play we setup the security groups, in the second play the VMs will created be assigned to these groups. Further you see, that we assign the public IP returned from the modules to the host inventory. This is needed as we do not know the IPs we will get in advance. In a next step you would configure the DNS

servers with these IPs for accessing the VMs with their DNS name.

In the last task we wait for SSH to be accessible, so any later play would be able to access the VM by SSH without failure.

1.6.4 Google Cloud Platform Guide

Introduction

Ansible + Google have been working together on a set of auto-generated Ansible modules designed to consistently and comprehensively cover the entirety of the Google Cloud Platform (GCP).

Ansible contains modules for managing Google Cloud Platform resources, including creating instances, controlling network access, working with persistent disks, managing load balancers, and a lot more.

These new modules can be found under a new consistent name scheme “gcp_*” (Note: gcp_target_proxy and gcp_url_map are legacy modules, despite the “gcp_*” name. Please use gcp_compute_target_proxy and gcp_compute_url_map instead).

Additionally, the gcp_compute inventory plugin can discover all Google Compute Engine (GCE) instances and make them automatically available in your Ansible inventory.

You may see a collection of other GCP modules that do not conform to this naming convention. These are the original modules primarily developed by the Ansible community. You will find some overlapping functionality such as with the “gce” module and the new “gcp_compute_instance” module. Either can be used, but you may experience issues trying to use them together.

While the community GCP modules are not going away, Google is investing effort into the new “gcp_*” modules. Google is committed to ensuring the Ansible community has a great experience with GCP and therefore recommends adopting these new modules if possible.

Requisites

The GCP modules require both the **requests** and the **google-auth** libraries to be installed.

```
$ pip install requests google-auth
```

Alternatively for RHEL / CentOS, the **python-requests** package is also available to satisfy **requests** libraries.

```
$ yum install python-requests
```


Credentials

It's easy to create a GCP account with credentials for Ansible. You have multiple options to get your credentials - here are two of the most common options:

- Service Accounts (Recommended): Use JSON service accounts with specific permissions.
- Machine Accounts: Use the permissions associated with the GCP Instance you're using Ansible on.

For the following examples, we'll be using service account credentials.

To work with the GCP modules, you'll first need to get some credentials in the JSON format:

1. [Create a Service Account](#)
2. [Download JSON credentials](#)

Once you have your credentials, there are two different ways to provide them to Ansible:

- by specifying them directly as module parameters
- by setting environment variables

Providing Credentials as Module Parameters

For the GCE modules you can specify the credentials as arguments:

- **auth_kind**: type of authentication being used (choices: machineaccount, serviceaccount, application)
- **service_account_email**: email associated with the project
- **service_account_file**: path to the JSON credentials file
- **project**: id of the project
- **scopes**: The specific scopes that you want the actions to use.

For example, to create a new IP address using the `gcp_compute_address` module, you can use the following configuration:

```
- name: Create IP address
  hosts: localhost
  gather_facts: no

  vars:
    service_account_file: /home/my_account.json
    project: my-project
    auth_kind: serviceaccount
    scopes:
      - https://www.googleapis.com/auth/compute
```

(下页继续)

(续上页)

```
tasks:

- name: Allocate an IP Address
  gcp_compute_address:
    state: present
    name: 'test-address1'
    region: 'us-west1'
    project: "{{ project }}"
    auth_kind: "{{ auth_kind }}"
    service_account_file: "{{ service_account_file }}"
    scopes: "{{ scopes }}"
```

Providing Credentials as Environment Variables

Set the following environment variables before running Ansible in order to configure your credentials:

```
GCP_AUTH_KIND
GCP_SERVICE_ACCOUNT_EMAIL
GCP_SERVICE_ACCOUNT_FILE
GCP_SCOPES
```

GCE Dynamic Inventory

The best way to interact with your hosts is to use the `gcp_compute` inventory plugin, which dynamically queries GCE and tells Ansible what nodes can be managed.

To be able to use this GCE dynamic inventory plugin, you need to enable it first by specifying the following in the `ansible.cfg` file:

```
[inventory]
enable_plugins = gcp_compute
```

Then, create a file that ends in `.gcp.yml` in your root directory.

The `gcp_compute` inventory script takes in the same authentication information as any module.

Here's an example of a valid inventory file:

```
plugin: gcp_compute
projects:
```

(下页继续)

(续上页)

```

- graphite-playground
auth_kind: serviceaccount
service_account_file: /home/alexstephen/my_account.json

```

Executing `ansible-inventory --list -i <filename>.gcp.yml` will create a list of GCP instances that are ready to be configured using Ansible.

Create an instance

The full range of GCP modules provide the ability to create a wide variety of GCP resources with the full support of the entire GCP API.

The following playbook creates a GCE Instance. This instance relies on a GCP network and a Disk. By creating the Disk and Network separately, we can give as much detail as necessary about how we want the disk and network formatted. By registering a Disk/Network to a variable, we can simply insert the variable into the instance task. The `gcp_compute_instance` module will figure out the rest.

```

- name: Create an instance
  hosts: localhost
  gather_facts: no
  vars:
    gcp_project: my-project
    gcp_cred_kind: serviceaccount
    gcp_cred_file: /home/my_account.json
    zone: "us-central1-a"
    region: "us-central1"

  tasks:
    - name: create a disk
      gcp_compute_disk:
        name: 'disk-instance'
        size_gb: 50
        source_image: 'projects/ubuntu-os-cloud/global/images/family/ubuntu-1604-lts'
        zone: "{{ zone }}"
        project: "{{ gcp_project }}"
        auth_kind: "{{ gcp_cred_kind }}"
        service_account_file: "{{ gcp_cred_file }}"
        scopes:
          - https://www.googleapis.com/auth/compute
        state: present

```

(下页继续)

(续上页)

```

    register: disk
- name: create a network
  gcp_compute_network:
    name: 'network-instance'
    project: "{{ gcp_project }}"
    auth_kind: "{{ gcp_cred_kind }}"
    service_account_file: "{{ gcp_cred_file }}"
    scopes:
      - https://www.googleapis.com/auth/compute
    state: present
  register: network
- name: create a address
  gcp_compute_address:
    name: 'address-instance'
    region: "{{ region }}"
    project: "{{ gcp_project }}"
    auth_kind: "{{ gcp_cred_kind }}"
    service_account_file: "{{ gcp_cred_file }}"
    scopes:
      - https://www.googleapis.com/auth/compute
    state: present
  register: address
- name: create a instance
  gcp_compute_instance:
    state: present
    name: test-vm
    machine_type: n1-standard-1
    disks:
      - auto_delete: true
        boot: true
        source: "{{ disk }}"
    network_interfaces:
      - network: "{{ network }}"
        access_configs:
          - name: 'External NAT'
            nat_ip: "{{ address }}"
            type: 'ONE_TO_ONE_NAT'
    zone: "{{ zone }}"
    project: "{{ gcp_project }}"
    auth_kind: "{{ gcp_cred_kind }}"

```

(下页继续)

(续上页)

```

    service_account_file: "{{ gcp_cred_file }}"
    scopes:
      - https://www.googleapis.com/auth/compute
    register: instance

- name: Wait for SSH to come up
  wait_for: host={{ address.address }} port=22 delay=10 timeout=60

- name: Add host to groupname
  add_host: hostname={{ address.address }} groupname=new_instances

- name: Manage new instances
  hosts: new_instances
  connection: ssh
  become: True
  roles:
    - base_configuration
    - production_server

```

Note that use of the “add_host” module above creates a temporary, in-memory group. This means that a play in the same playbook can then manage machines in the ‘new_instances’ group, if so desired. Any sort of arbitrary configuration is possible at this point.

For more information about Google Cloud, please visit the [Google Cloud website](#).

Migration Guides

`gce.py` -> `gcp_compute_instance.py`

As of Ansible 2.8, we’re encouraging everyone to move from the `gce` module to the `gcp_compute_instance` module. The `gcp_compute_instance` module has better support for all of GCP’s features, fewer dependencies, more flexibility, and better supports GCP’s authentication systems.

The `gcp_compute_instance` module supports all of the features of the `gce` module (and more!). Below is a mapping of `gce` fields over to `gcp_compute_instance` fields.

gce.py	gcp_compute_instance.py	
state	state/status	State on gce has multiple values: “present”, “absent”, “stopped”, “started”, “terminated”. State on gcp_compute_instance is used to describe if the instance exists (present) or does not (absent). Status is used to describe if the instance is “started”, “stopped” or “terminated”.
image	disks[].initialize_params.source_image	You can use to create a single disk using the disks[] parameter and set it to be the boot disk (disks[].boot = true)
image_family	disks[].initialize_params.source_image	See above.
external_projects	disks[].initialize_params.source_image	The params.source_image will include the name of the project.
instance_name	Use a loop or multiple tasks.	Using loops is a more Ansible-centric way of creating multiple instances and gives you the most flexibility.
service_account_email	service_account_email	This is the service_account email address that you want the instance to be associated with. It is not the service_account email address that is used for the credentials necessary to create the instance.
service_account_permissions	service_account_permissions	These are the permissions you want to grant to the instance.
pem_file	Not supported.	We recommend using JSON service account credentials instead of PEM files.
credentials_file	service_account_file	
project_id	project	
name	name	This field does not accept an array of names. Use a loop to create multiple instances.
num_instances	Uses a loop	For maximum flexibility, we’re encouraging users to use Ansible features to create multiple instances, rather than letting the module do it for you.
network	network_interfaces[].network	
subnet-work	network_interfaces[].subnetwork	
persistent_boot_disk	disks[].type =disk ‘PERSISTENT’	
disks	disks[]	
ip_forward	can_ip_forward	
external_ip	network_interfaces[].public_ip	This field takes multiple types of values. You can create an IP address with gcp_compute_addresses and place the name/output of the address here. You can also place the string value of the IP address’s GCP name or the actual IP address.
disks_auto_delete	disks[].auto_delete	
pre-emptible	scheduling.preemptible	
disk_size	disks[].initialize_params.disk_size_gb	

An example playbook is below:

```
gcp_compute_instance:
  name: "{{ item }}"
  machine_type: n1-standard-1
  ... # any other settings
  zone: us-central1-a
  project: "my-project"
  auth_kind: "service_account_file"
  service_account_file: "~/my_account.json"
  state: present
loop:
- instance-1
- instance-2
```

1.6.5 Microsoft Azure Guide

Ansible includes a suite of modules for interacting with Azure Resource Manager, giving you the tools to easily create and orchestrate infrastructure on the Microsoft Azure Cloud.

Requirements

Using the Azure Resource Manager modules requires having specific Azure SDK modules installed on the host running Ansible.

```
$ pip install 'ansible[azure]'
```

If you are running Ansible from source, you can install the dependencies from the root directory of the Ansible repo.

```
$ pip install .[azure]
```

You can also directly run Ansible in [Azure Cloud Shell](#), where Ansible is pre-installed.

Authenticating with Azure

Using the Azure Resource Manager modules requires authenticating with the Azure API. You can choose from two authentication strategies:

- Active Directory Username/Password
- Service Principal Credentials

Follow the directions for the strategy you wish to use, then proceed to *Providing Credentials to Azure Modules* for instructions on how to actually use the modules and authenticate with the Azure API.

Using Service Principal

There is now a detailed official tutorial describing [how to create a service principal](#).

After stepping through the tutorial you will have:

- Your Client ID, which is found in the “client id” box in the “Configure” page of your application in the Azure portal
- Your Secret key, generated when you created the application. You cannot show the key after creation. If you lost the key, you must create a new one in the “Configure” page of your application.
- And finally, a tenant ID. It’s a UUID (e.g. ABCDEFGH-1234-ABCD-1234-ABCDEFGHIJKL) pointing to the AD containing your application. You will find it in the URL from within the Azure portal, or in the “view endpoints” of any given URL.

Using Active Directory Username/Password

To create an Active Directory username/password:

- Connect to the Azure Classic Portal with your admin account
- Create a user in your default AAD. You must NOT activate Multi-Factor Authentication
- Go to Settings - Administrators
- Click on Add and enter the email of the new user.
- Check the checkbox of the subscription you want to test with this user.
- Login to Azure Portal with this new user to change the temporary password to a new one. You will not be able to use the temporary password for OAuth login.

Providing Credentials to Azure Modules

The modules offer several ways to provide your credentials. For a CI/CD tool such as Ansible Tower or Jenkins, you will most likely want to use environment variables. For local development you may wish to store your credentials in a file within your home directory. And of course, you can always pass credentials as parameters to a task within a playbook. The order of precedence is parameters, then environment variables, and finally a file found in your home directory.

Using Environment Variables

To pass service principal credentials via the environment, define the following variables:

- AZURE_CLIENT_ID
- AZURE_SECRET
- AZURE_SUBSCRIPTION_ID
- AZURE_TENANT

To pass Active Directory username/password via the environment, define the following variables:

- AZURE_AD_USER
- AZURE_PASSWORD
- AZURE_SUBSCRIPTION_ID

To pass Active Directory username/password in ADFS via the environment, define the following variables:

- AZURE_AD_USER
- AZURE_PASSWORD
- AZURE_CLIENT_ID
- AZURE_TENANT
- AZURE_ADFS_AUTHORITY_URL

“AZURE_ADFS_AUTHORITY_URL” is optional. It’s necessary only when you have own ADFS authority like <https://yourdomain.com/adfs>.

Storing in a File

When working in a development environment, it may be desirable to store credentials in a file. The modules will look for credentials in `$HOME/.azure/credentials`. This file is an ini style file. It will look as follows:

```
[default]
subscription_id=xxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxx
client_id=xxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxx
secret=xxxxxxxxxxxxxxxxxxxxx
tenant=xxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxx
```

注解: If your secret values contain non-ASCII characters, you must [URL Encode](#) them to avoid login errors.

It is possible to store multiple sets of credentials within the credentials file by creating multiple sections. Each section is considered a profile. The modules look for the [default] profile automatically. Define AZURE_PROFILE in the environment or pass a profile parameter to specify a specific profile.

Passing as Parameters

If you wish to pass credentials as parameters to a task, use the following parameters for service principal:

- `client_id`
- `secret`
- `subscription_id`
- `tenant`

Or, pass the following parameters for Active Directory username/password:

- `ad_user`
- `password`
- `subscription_id`

Or, pass the following parameters for ADFS username/pasword:

- `ad_user`
- `password`
- `client_id`
- `tenant`
- `adfs_authority_url`

“`adfs_authority_url`” is optional. It’ s necessary only when you have own ADFS authority like <https://yourdomain.com/adfs>.

Other Cloud Environments

To use an Azure Cloud other than the default public cloud (eg, Azure China Cloud, Azure US Government Cloud, Azure Stack), pass the “`cloud_environment`” argument to modules, configure it in a credential profile, or set the “`AZURE_CLOUD_ENVIRONMENT`” environment variable. The value is either a cloud name as defined by the Azure Python SDK (eg, “`AzureChinaCloud`” , “`AzureUSGovernment`” ; defaults to “`AzureCloud`”) or an Azure metadata discovery URL (for Azure Stack).

Creating Virtual Machines

There are two ways to create a virtual machine, both involving the `azure_rm_virtualmachine` module. We can either create a storage account, network interface, security group and public IP address and pass the names of these objects to the module as parameters, or we can let the module do the work for us and accept the defaults it chooses.

Creating Individual Components

An Azure module is available to help you create a storage account, virtual network, subnet, network interface, security group and public IP. Here is a full example of creating each of these and passing the names to the `azure_rm_virtualmachine` module at the end:

```
- name: Create storage account
  azure_rm_storageaccount:
    resource_group: Testing
    name: testaccount001
    account_type: Standard_LRS

- name: Create virtual network
  azure_rm_virtualnetwork:
    resource_group: Testing
    name: testvn001
    address_prefixes: "10.10.0.0/16"

- name: Add subnet
  azure_rm_subnet:
    resource_group: Testing
    name: subnet001
    address_prefix: "10.10.0.0/24"
    virtual_network: testvn001

- name: Create public ip
  azure_rm_publicipaddress:
    resource_group: Testing
    allocation_method: Static
    name: publicip001

- name: Create security group that allows SSH
  azure_rm_securitygroup:
    resource_group: Testing
    name: secgroup001
    rules:
      - name: SSH
        protocol: Tcp
        destination_port_range: 22
        access: Allow
        priority: 101
```

(下页继续)

(续上页)

```
        direction: Inbound

- name: Create NIC
  azure_rm_networkinterface:
    resource_group: Testing
    name: testnic001
    virtual_network: testvn001
    subnet: subnet001
    public_ip_name: publicip001
    security_group: secgroup001

- name: Create virtual machine
  azure_rm_virtualmachine:
    resource_group: Testing
    name: testvm001
    vm_size: Standard_D1
    storage_account: testaccount001
    storage_container: testvm001
    storage_blob: testvm001.vhd
    admin_username: admin
    admin_password: Password!
    network_interfaces: testnic001
    image:
      offer: CentOS
      publisher: OpenLogic
      sku: '7.1'
      version: latest
```

Each of the Azure modules offers a variety of parameter options. Not all options are demonstrated in the above example. See each individual module for further details and examples.

Creating a Virtual Machine with Default Options

If you simply want to create a virtual machine without specifying all the details, you can do that as well. The only caveat is that you will need a virtual network with one subnet already in your resource group. Assuming you have a virtual network already with an existing subnet, you can run the following to create a VM:

```
azure_rm_virtualmachine:
  resource_group: Testing
```

(下页继续)

(续上页)

```

name: testvm10
vm_size: Standard_D1
admin_username: chouseknecht
ssh_password_enabled: false
ssh_public_keys: "{{ ssh_keys }}"
image:
  offer: CentOS
  publisher: OpenLogic
  sku: '7.1'
  version: latest

```

Creating a Virtual Machine in Availability Zones

If you want to create a VM in an availability zone, consider the following:

- Both OS disk and data disk must be a ‘managed disk’ , not an ‘unmanaged disk’ .
- When creating a VM with the `azure_rm_virtualmachine` module, you need to explicitly set the `managed_disk_type` parameter to change the OS disk to a managed disk. Otherwise, the OS disk becomes an unmanaged disk..
- When you create a data disk with the `azure_rm_manageddisk` module, you need to explicitly specify the `storage_account_type` parameter to make it a managed disk. Otherwise, the data disk will be an unmanaged disk.
- A managed disk does not require a storage account or a storage container, unlike a n unmanaged disk. In particular, note that once a VM is created on an unmanaged disk, an unnecessary storage container named “vhds” is automatically created.
- When you create an IP address with the `azure_rm_publicipaddress` module, you must set the `sku` parameter to `standard`. Otherwise, the IP address cannot be used in an availability zone.

Dynamic Inventory Script

If you are not familiar with Ansible’ s dynamic inventory scripts, check out *Intro to Dynamic Inventory*.

The Azure Resource Manager inventory script is called `azure_rm.py`. It authenticates with the Azure API exactly the same as the Azure modules, which means you will either define the same environment variables described above in *Using Environment Variables*, create a `$HOME/.azure/credentials` file (also described above in *Storing in a File*), or pass command line parameters. To see available command line options execute the following:

```
$ ./ansible/contrib/inventory/azure_rm.py --help
```

As with all dynamic inventory scripts, the script can be executed directly, passed as a parameter to the ansible command, or passed directly to ansible-playbook using the -i option. No matter how it is executed the script produces JSON representing all of the hosts found in your Azure subscription. You can narrow this down to just hosts found in a specific set of Azure resource groups, or even down to a specific host.

For a given host, the inventory script provides the following host variables:

```
{
  "ansible_host": "XXX.XXX.XXX.XXX",
  "computer_name": "computer_name2",
  "fqdn": null,
  "id": "/subscriptions/subscription-id/resourceGroups/galaxy-production/providers/
↪Microsoft.Compute/virtualMachines/object-name",
  "image": {
    "offer": "CentOS",
    "publisher": "OpenLogic",
    "sku": "7.1",
    "version": "latest"
  },
  "location": "westus",
  "mac_address": "00-00-5E-00-53-FE",
  "name": "object-name",
  "network_interface": "interface-name",
  "network_interface_id": "/subscriptions/subscription-id/resourceGroups/galaxy-
↪production/providers/Microsoft.Network/networkInterfaces/object-name1",
  "network_security_group": null,
  "network_security_group_id": null,
  "os_disk": {
    "name": "object-name",
    "operating_system_type": "Linux"
  },
  "plan": null,
  "powerstate": "running",
  "private_ip": "172.26.3.6",
  "private_ip_alloc_method": "Static",
  "provisioning_state": "Succeeded",
  "public_ip": "XXX.XXX.XXX.XXX",
  "public_ip_alloc_method": "Static",
  "public_ip_id": "/subscriptions/subscription-id/resourceGroups/galaxy-production/
↪providers/Microsoft.Network/publicIPAddresses/object-name",
```

(下页继续)

(续上页)

```

"public_ip_name": "object-name",
"resource_group": "galaxy-production",
"security_group": "object-name",
"security_group_id": "/subscriptions/subscription-id/resourceGroups/galaxy-production/
↳ providers/Microsoft.Network/networkSecurityGroups/object-name",
"tags": {
    "db": "mysql"
},
"type": "Microsoft.Compute/virtualMachines",
"virtual_machine_size": "Standard_DS4"
}

```

Host Groups

By default hosts are grouped by:

- azure (all hosts)
- location name
- resource group name
- security group name
- tag key
- tag key__value
- os_disk operating_system_type (Windows/Linux)

You can control host groupings and host selection by either defining environment variables or creating an `azure_rm.ini` file in your current working directory.

NOTE: An `.ini` file will take precedence over environment variables.

NOTE: The name of the `.ini` file is the basename of the inventory script (i.e. `'azure_rm'`) with a `'ini'` extension. This allows you to copy, rename and customize the inventory script and have matching `.ini` files all in the same directory.

Control grouping using the following variables defined in the environment:

- `AZURE_GROUP_BY_RESOURCE_GROUP=yes`
- `AZURE_GROUP_BY_LOCATION=yes`
- `AZURE_GROUP_BY_SECURITY_GROUP=yes`
- `AZURE_GROUP_BY_TAG=yes`

- `AZURE_GROUP_BY_OS_FAMILY=yes`

Select hosts within specific resource groups by assigning a comma separated list to:

- `AZURE_RESOURCE_GROUPS=resource_group_a,resource_group_b`

Select hosts for specific tag key by assigning a comma separated list of tag keys to:

- `AZURE_TAGS=key1,key2,key3`

Select hosts for specific locations by assigning a comma separated list of locations to:

- `AZURE_LOCATIONS=eastus,eastus2,westus`

Or, select hosts for specific tag key:value pairs by assigning a comma separated list key:value pairs to:

- `AZURE_TAGS=key1:value1,key2:value2`

If you don't need the powerstate, you can improve performance by turning off powerstate fetching:

- `AZURE_INCLUDE_POWERSTATE=no`

A sample `azure_rm.ini` file is included along with the inventory script in `contrib/inventory`. An `.ini` file will contain the following:

```
[azure]
# Control which resource groups are included. By default all resources groups are
↳ included.
# Set resource_groups to a comma separated list of resource groups names.
#resource_groups=

# Control which tags are included. Set tags to a comma separated list of keys or
↳ key:value pairs
#tags=

# Control which locations are included. Set locations to a comma separated list of
↳ locations.
#locations=

# Include powerstate. If you don't need powerstate information, turning it off improves
↳ runtime performance.
# Valid values: yes, no, true, false, True, False, 0, 1.
include_powerstate=yes

# Control grouping with the following boolean flags. Valid values: yes, no, true, false,
↳ True, False, 0, 1.
group_by_resource_group=yes
group_by_location=yes
```

(下页继续)

(续上页)

```
group_by_security_group=yes
group_by_tag=yes
group_by_os_family=yes
```

Examples

Here are some examples using the inventory script:

```
# Execute /bin/uname on all instances in the Testing resource group
$ ansible -i azure_rm.py Testing -m shell -a "/bin/uname -a"

# Execute win_ping on all Windows instances
$ ansible -i azure_rm.py windows -m win_ping

# Execute ping on all Linux instances
$ ansible -i azure_rm.py linux -m ping

# Use the inventory script to print instance specific information
$ ./ansible/contrib/inventory/azure_rm.py --host my_instance_host_name --resource-
→groups=Testing --pretty

# Use the inventory script with ansible-playbook
$ ansible-playbook -i ./ansible/contrib/inventory/azure_rm.py test_playbook.yml
```

Here is a simple playbook to exercise the Azure inventory script:

```
- name: Test the inventory script
  hosts: azure
  connection: local
  gather_facts: no
  tasks:
    - debug: msg="{{ inventory_hostname }}" has powerstate {{ powerstate }}
```

You can execute the playbook with something like:

```
$ ansible-playbook -i ./ansible/contrib/inventory/azure_rm.py test_azure_inventory.yml
```

Disabling certificate validation on Azure endpoints

When an HTTPS proxy is present, or when using Azure Stack, it may be necessary to disable certificate validation for Azure endpoints in the Azure modules. This is not a recommended security practice, but may be necessary when the system CA store cannot be altered to include the necessary CA certificate. Certificate validation can be controlled by setting the “cert_validation_mode” value in a credential profile, via the “AZURE_CERT_VALIDATION_MODE” environment variable, or by passing the “cert_validation_mode” argument to any Azure module. The default value is “validate”; setting the value to “ignore” will prevent all certificate validation. The module argument takes precedence over a credential profile value, which takes precedence over the environment value.

1.6.6 Online.net Guide

Introduction

Online is a French hosting company mainly known for providing bare-metal servers named Dedibox. Check it out: <https://www.online.net/en>

Dynamic inventory for Online resources

Ansible has a dynamic inventory plugin that can list your resources.

1. Create a YAML configuration such as `online_inventory.yml` with this content:

```
plugin: online
```

2. Set your `ONLINE_TOKEN` environment variable with your token. You need to open an account and log into it before you can get a token. You can find your token at the following page: <https://console.online.net/en/api/access>

3. You can test that your inventory is working by running:

```
$ ansible-inventory -v -i online_inventory.yml --list
```

4. Now you can run your playbook or any other module with this inventory:

```
$ ansible all -i online_inventory.yml -m ping
sd-96735 | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
```

1.6.7 Oracle Cloud Infrastructure Guide

Introduction

Oracle provides a number of Ansible modules to interact with Oracle Cloud Infrastructure (OCI). In this guide, we will explain how you can use these modules to orchestrate, provision and configure your infrastructure on OCI.

Requirements

To use the OCI Ansible modules, you must have the following prerequisites on your control node, the computer from which Ansible playbooks are executed.

1. [An Oracle Cloud Infrastructure account](#).
2. A user created in that account, in a security group with a policy that grants the necessary permissions for working with resources in those compartments. For guidance, see [How Policies Work](#).
3. The necessary credentials and OCID information.

Installation

1. Install the Oracle Cloud Infrastructure Python SDK ([detailed installation instructions](#)):

```
pip install oci
```

2. Install the Ansible OCI Modules in one of two ways:

- a. From Galaxy:

```
ansible-galaxy install oracle.oci_ansible_modules
```

- b. From GitHub:

```
$ git clone https://github.com/oracle/oci-ansible-modules.git
```

```
$ cd oci-ansible-modules
```

Run one of the following commands:

- If Ansible is installed only for your user:

```
$ ./install.py
```

- If Ansible is installed as root:

```
$ sudo ./install.py
```

Configuration

When creating and configuring Oracle Cloud Infrastructure resources, Ansible modules use the authentication information outlined [here](#). .

Examples

Launch a compute instance

This [sample launch playbook](#) launches a public Compute instance and then accesses the instance from an Ansible module over an SSH connection. The sample illustrates how to:

- Generate a temporary, host-specific SSH key pair.
- Specify the public key from the key pair for connecting to the instance, and then launch the instance.
- Connect to the newly launched instance using SSH.

Create and manage Autonomous Data Warehouses

This [sample warehouse playbook](#) creates an Autonomous Data Warehouse and manage its lifecycle. The sample shows how to:

- Set up an Autonomous Data Warehouse.
- List all of the Autonomous Data Warehouse instances available in a compartment, filtered by the display name.
- Get the “facts” for a specified Autonomous Data Warehouse.
- Stop and start an Autonomous Data Warehouse instance.
- Delete an Autonomous Data Warehouse instance.

Create and manage Autonomous Transaction Processing

This [sample playbook](#) creates an Autonomous Transaction Processing database and manage its lifecycle. The sample shows how to:

- Set up an Autonomous Transaction Processing database instance.
- List all of the Autonomous Transaction Processing instances in a compartment, filtered by the display name.
- Get the “facts” for a specified Autonomous Transaction Processing instance.

- Delete an Autonomous Transaction Processing database instance.

You can find more examples here: [Sample Ansible Playbooks](#).

1.6.8 Packet.net Guide

Introduction

[Packet.net](#) is a bare metal infrastructure host that's supported by Ansible (≥ 2.3) via a dynamic inventory script and two cloud modules. The two modules are:

- `packet_sshkey`: adds a public SSH key from file or value to the Packet infrastructure. Every subsequently-created device will have this public key installed in `.ssh/authorized_keys`.
- `packet_device`: manages servers on Packet. You can use this module to create, restart and delete devices.

Note, this guide assumes you are familiar with Ansible and how it works. If you're not, have a look at their *docs* before getting started.

Requirements

The Packet modules and inventory script connect to the Packet API using the `packet-python` package. You can install it with `pip`:

```
$ pip install packet-python
```

In order to check the state of devices created by Ansible on Packet, it's a good idea to install one of the [Packet CLI clients](#). Otherwise you can check them via the [Packet portal](#).

To use the modules and inventory script you'll need a Packet API token. You can generate an API token via the Packet portal [here](#). The simplest way to authenticate yourself is to set the Packet API token in an environment variable:

```
$ export PACKET_API_TOKEN=Bfse9F24SFtfs423Gsd3ifGsd43sSdfs
```

If you're not comfortable exporting your API token, you can pass it as a parameter to the modules.

On Packet, devices and reserved IP addresses belong to [projects](#). In order to use the `packet_device` module, you need to specify the UUID of the project in which you want to create or manage devices. You can find a project's UUID in the Packet portal [here](#) (it's just under the project table) or via one of the available CLIs.

If you want to use a new SSH keypair in this tutorial, you can generate it to `./id_rsa` and `./id_rsa.pub` as:

```
$ ssh-keygen -t rsa -f ./id_rsa
```

If you want to use an existing keypair, just copy the private and public key over to the playbook directory.

Device Creation

The following code block is a simple playbook that creates one [Type 0](#) server (the ‘plan’ parameter). You have to supply ‘plan’ and ‘operating_system’. ‘location’ defaults to ‘ewr1’ (Parsippany, NJ). You can find all the possible values for the parameters via a [CLI client](#).

```
# playbook_create.yml

- name: create ubuntu device
  hosts: localhost
  tasks:

    - packet_sshkey:
      key_file: ./id_rsa.pub
      label: tutorial key

    - packet_device:
      project_id: <your_project_id>
      hostnames: myserver
      operating_system: ubuntu_16_04
      plan: baremetal_0
      facility: sjc1
```

After running `ansible-playbook playbook_create.yml`, you should have a server provisioned on Packet. You can verify via a CLI or in the [Packet portal](#).

If you get an error with the message “failed to set machine state present, error: Error 404: Not Found” , please verify your project UUID.

Updating Devices

The two parameters used to uniquely identify Packet devices are: “device_ids” and “hostnames”. Both parameters accept either a single string (later converted to a one-element list), or a list of strings.

The ‘device_ids’ and ‘hostnames’ parameters are mutually exclusive. The following values are all acceptable:

- device_ids: a27b7a83-fc93-435b-a128-47a5b04f2dcf
- hostnames: mydev1
- device_ids: [a27b7a83-fc93-435b-a128-47a5b04f2dcf, 4887130f-0ccd-49a0-99b0-323c1ceb527b]

- `hostnames: [mydev1, mydev2]`

In addition, `hostnames` can contain a special `'%d'` formatter along with a `'count'` parameter that lets you easily expand hostnames that follow a simple name and number pattern; i.e. `hostnames: "mydev%d"`, `count: 2` will expand to `[mydev1, mydev2]`.

If your playbook acts on existing Packet devices, you can only pass the `'hostname'` and `'device_ids'` parameters. The following playbook shows how you can reboot a specific Packet device by setting the `'hostname'` parameter:

```
# playbook_reboot.yml

- name: reboot myserver
  hosts: localhost
  tasks:

  - packet_device:
      project_id: <your_project_id>
      hostnames: myserver
      state: rebooted
```

You can also identify specific Packet devices with the `'device_ids'` parameter. The device's UUID can be found in the [Packet Portal](#) or by using a [CLI](#). The following playbook removes a Packet device using the `'device_ids'` field:

```
# playbook_remove.yml

- name: remove a device
  hosts: localhost
  tasks:

  - packet_device:
      project_id: <your_project_id>
      device_ids: <myserver_device_id>
      state: absent
```

More Complex Playbooks

In this example, we'll create a CoreOS cluster with [user data](#).

The CoreOS cluster will use [etcd](#) for discovery of other servers in the cluster. Before provisioning your servers, you'll need to generate a discovery token for your cluster:

```
$ curl -w "\n" 'https://discovery.etcd.io/new?size=3'
```

The following playbook will create an SSH key, 3 Packet servers, and then wait until SSH is ready (or until 5 minutes passed). Make sure to substitute the discovery token URL in ‘user_data’, and the ‘project_id’ before running `ansible-playbook`. Also, feel free to change ‘plan’ and ‘facility’.

```
# playbook_coreos.yml

- name: Start 3 CoreOS nodes in Packet and wait until SSH is ready
  hosts: localhost
  tasks:

  - packet_sshkey:
    key_file: ./id_rsa.pub
    label: new

  - packet_device:
    hostnames: [coreos-one, coreos-two, coreos-three]
    operating_system: coreos_beta
    plan: baremetal_0
    facility: ewr1
    project_id: <your_project_id>
    wait_for_public_IPv: 4
    user_data: |
      #cloud-config
      coreos:
        etcd2:
          discovery: https://discovery.etcd.io/<token>
          advertise-client-urls: http://$private_ipv4:2379,http://$private_ipv4:4001
          initial-advertise-peer-urls: http://$private_ipv4:2380
          listen-client-urls: http://0.0.0.0:2379,http://0.0.0.0:4001
          listen-peer-urls: http://$private_ipv4:2380
        fleet:
          public-ip: $private_ipv4
        units:
          - name: etcd2.service
            command: start
          - name: fleet.service
            command: start

    register: newhosts
```

(下页继续)

(续上页)

```
- name: wait for ssh
  wait_for:
    delay: 1
    host: "{{ item.public_ipv4 }}"
    port: 22
    state: started
    timeout: 500
  loop: "{{ newhosts.results[0].devices }}"
```

As with most Ansible modules, the default states of the Packet modules are idempotent, meaning the resources in your project will remain the same after re-runs of a playbook. Thus, we can keep the `packet_sshkey` module call in our playbook. If the public key is already in your Packet account, the call will have no effect.

The second module call provisions 3 Packet Type 0 (specified using the ‘plan’ parameter) servers in the project identified via the ‘project_id’ parameter. The servers are all provisioned with CoreOS beta (the ‘operating_system’ parameter) and are customized with cloud-config user data passed to the ‘user_data’ parameter.

The `packet_device` module has a `wait_for_public_IPv` that is used to specify the version of the IP address to wait for (valid values are 4 or 6 for IPv4 or IPv6). If specified, Ansible will wait until the GET API call for a device contains an Internet-routeable IP address of the specified version. When referring to an IP address of a created device in subsequent module calls, it’s wise to use the `wait_for_public_IPv` parameter, or `state: active` in the `packet_device` module call.

Run the playbook:

```
$ ansible-playbook playbook_coreos.yml
```

Once the playbook quits, your new devices should be reachable via SSH. Try to connect to one and check if `etcd` has started properly:

```
tomk@work $ ssh -i id_rsa core@$one_of_the_servers_ip
core@coreos-one ~ $ etcdctl cluster-health
```

Once you create a couple of devices, you might appreciate the dynamic inventory script...

Dynamic Inventory Script

The dynamic inventory script queries the Packet API for a list of hosts, and exposes it to Ansible so you can easily identify and act on Packet devices. You can find it in Ansible’s git repo at [contrib/inventory/packet_net.py](https://github.com/ansible/ansible/blob/master/contrib/inventory/packet_net.py).

The inventory script is configurable via a [ini file](#).

If you want to use the inventory script, you must first export your Packet API token to a `PACKET_API_TOKEN` environment variable.

You can either copy the inventory and ini config out from the cloned git repo, or you can download it to your working directory like so:

```
$ wget https://github.com/ansible/ansible/raw/devel/contrib/inventory/packet_net.py
$ chmod +x packet_net.py
$ wget https://github.com/ansible/ansible/raw/devel/contrib/inventory/packet_net.ini
```

In order to understand what the inventory script gives to Ansible you can run:

```
$ ./packet_net.py --list
```

It should print a JSON document looking similar to following trimmed dictionary:

```
{
  "_meta": {
    "hostvars": {
      "147.75.64.169": {
        "packet_billing_cycle": "hourly",
        "packet_created_at": "2017-02-09T17:11:26Z",
        "packet_facility": "ewr1",
        "packet_hostname": "coreos-two",
        "packet_href": "/devices/d0ab8972-54a8-4bff-832b-28549d1bec96",
        "packet_id": "d0ab8972-54a8-4bff-832b-28549d1bec96",
        "packet_locked": false,
        "packet_operating_system": "coreos_beta",
        "packet_plan": "baremetal_0",
        "packet_state": "active",
        "packet_updated_at": "2017-02-09T17:16:35Z",
        "packet_user": "core",
        "packet_userdata": "#cloud-config\ncoreos:\n  etcd2:\n    discovery: https://
↪discovery.etcd.io/e0c8a4a9b8fe61acd51ec599e2a4f68e\n    advertise-client-urls: http://
↪$private_ipv4:2379,http://$private_ipv4:4001\n    initial-advertise-peer-urls: http://
↪$private_ipv4:2380\n    listen-client-urls: http://0.0.0.0:2379,http://0.0.0.0:4001\n
↪listen-peer-urls: http://$private_ipv4:2380\n  fleet:\n    public-ip: $private_
↪ipv4\n  units:\n    - name: etcd2.service\n    command: start\n    - name: fleet.
↪service\n    command: start"
      }
    }
  },
  "baremetal_0": [
```

(下页继续)

(续上页)

```

    "147.75.202.255",
    "147.75.202.251",
    "147.75.202.249",
    "147.75.64.129",
    "147.75.192.51",
    "147.75.64.169"
  ],
  "coreos_beta": [
    "147.75.202.255",
    "147.75.202.251",
    "147.75.202.249",
    "147.75.64.129",
    "147.75.192.51",
    "147.75.64.169"
  ],
  "ewr1": [
    "147.75.64.129",
    "147.75.192.51",
    "147.75.64.169"
  ],
  "sjc1": [
    "147.75.202.255",
    "147.75.202.251",
    "147.75.202.249"
  ],
  "coreos-two": [
    "147.75.64.169"
  ],
  "d0ab8972-54a8-4bff-832b-28549d1bec96": [
    "147.75.64.169"
  ]
}

```

In the `['_meta']['hostvars']` key, there is a list of devices (uniquely identified by their public IPv4 address) with their parameters. The other keys under `['_meta']` are lists of devices grouped by some parameter. Here, it is type (all devices are of type `baremetal_0`), operating system, and facility (`ewr1` and `sjc1`).

In addition to the parameter groups, there are also one-item groups with the UUID or hostname of the device.

You can now target groups in playbooks! The following playbook will install a role that supplies resources for an Ansible target into all devices in the “`coreos_beta`” group:

```
# playbook_bootstrap.yml

- hosts: coreos_beta
  gather_facts: false
  roles:
    - defunctzombie.coreos-bootstrap
```

Don't forget to supply the dynamic inventory in the `-i` argument!

```
$ ansible-playbook -u core -i packet_net.py playbook_bootstrap.yml
```

If you have any questions or comments let us know! help@packet.net

1.6.9 Rackspace Cloud Guide

Introduction

注解: This section of the documentation is under construction. We are in the process of adding more examples about the Rackspace modules and how they work together. Once complete, there will also be examples for Rackspace Cloud in [ansible-examples](#).

Ansible contains a number of core modules for interacting with Rackspace Cloud.

The purpose of this section is to explain how to put Ansible modules together (and use inventory scripts) to use Ansible in a Rackspace Cloud context.

Prerequisites for using the `rax` modules are minimal. In addition to ansible itself, all of the modules require and are tested against `pyrax` 1.5 or higher. You'll need this Python module installed on the execution host.

`pyrax` is not currently available in many operating system package repositories, so you will likely need to install it via `pip`:

```
$ pip install pyrax
```

Ansible creates an implicit localhost that executes in the same context as the `ansible-playbook` and the other CLI tools. If for any reason you need or want to have it in your inventory you should do something like the following:

```
[localhost]
localhost ansible_connection=local ansible_python_interpreter=/usr/local/bin/python2
```

For more information see Implicit Localhost

In playbook steps, we' ll typically be using the following pattern:

```
- hosts: localhost
  gather_facts: False
  tasks:
```

Credentials File

The *rax.py* inventory script and all *rax* modules support a standard *pyrax* credentials file that looks like:

```
[rackspace_cloud]
username = myraxusername
api_key = d41d8cd98f00b204e9800998ecf8427e
```

Setting the environment parameter `RAX_CREDS_FILE` to the path of this file will help Ansible find how to load this information.

More information about this credentials file can be found at https://github.com/pycontribs/pyrax/blob/master/docs/getting_started.md#authenticating

Running from a Python Virtual Environment (Optional)

Most users will not be using `virtualenv`, but some users, particularly Python developers sometimes like to.

There are special considerations when Ansible is installed to a Python `virtualenv`, rather than the default of installing at a global scope. Ansible assumes, unless otherwise instructed, that the python binary will live at `/usr/bin/python`. This is done via the interpreter line in modules, however when instructed by setting the inventory variable `'ansible_python_interpreter'`, Ansible will use this specified path instead to find Python. This can be a cause of confusion as one may assume that modules running on `'localhost'`, or perhaps running via `'local_action'`, are using the `virtualenv` Python interpreter. By setting this line in the inventory, the modules will execute in the `virtualenv` interpreter and have available the `virtualenv` packages, specifically `pyrax`. If using `virtualenv`, you may wish to modify your `localhost` inventory definition to find this location as follows:

```
[localhost]
localhost ansible_connection=local ansible_python_interpreter=/path/to/ansible_venv/bin/
↳python
```

注解: `pyrax` may be installed in the global Python package scope or in a virtual environment. There are no special considerations to keep in mind when installing `pyrax`.

Provisioning

Now for the fun parts.

The ‘rax’ module provides the ability to provision instances within Rackspace Cloud. Typically the provisioning task will be performed from your Ansible control server (in our example, localhost) against the Rackspace cloud API. This is done for several reasons:

- Avoiding installing the pyrax library on remote nodes
- No need to encrypt and distribute credentials to remote nodes
- Speed and simplicity

注解: Authentication with the Rackspace-related modules is handled by either specifying your username and API key as environment variables or passing them as module arguments, or by specifying the location of a credentials file.

Here is a basic example of provisioning an instance in ad-hoc mode:

```
$ ansible localhost -m rax -a "name=awx flavor=4 image=ubuntu-1204-lts-precise-pangolin_↵
↵wait=yes"
```

Here’s what it would look like in a playbook, assuming the parameters were defined in variables:

```
tasks:
- name: Provision a set of instances
  rax:
    name: "{{ rax_name }}"
    flavor: "{{ rax_flavor }}"
    image: "{{ rax_image }}"
    count: "{{ rax_count }}"
    group: "{{ group }}"
    wait: yes
  register: rax
  delegate_to: localhost
```

The rax module returns data about the nodes it creates, like IP addresses, hostnames, and login passwords. By registering the return value of the step, it is possible used this data to dynamically add the resulting hosts to inventory (temporarily, in memory). This facilitates performing configuration actions on the hosts in a follow-on task. In the following example, the servers that were successfully created using the above task are dynamically added to a group called “raxhosts” , with each nodes hostname, IP address, and root password being added to the inventory.

```
- name: Add the instances we created (by public IP) to the group 'raxhosts'
  add_host:
    hostname: "{{ item.name }}"
    ansible_host: "{{ item.rax_accessipv4 }}"
    ansible_password: "{{ item.rax_adminpass }}"
    groups: raxhosts
  loop: "{{ rax.success }}"
  when: rax.action == 'create'
```

With the host group now created, the next play in this playbook could now configure servers belonging to the raxhosts group.

```
- name: Configuration play
  hosts: raxhosts
  user: root
  roles:
    - ntp
    - webserver
```

The method above ties the configuration of a host with the provisioning step. This isn't always what you want, and leads us to the next section.

Host Inventory

Once your nodes are spun up, you'll probably want to talk to them again. The best way to handle this is to use the “rax” inventory plugin, which dynamically queries Rackspace Cloud and tells Ansible what nodes you have to manage. You might want to use this even if you are spinning up cloud instances via other tools, including the Rackspace Cloud user interface. The inventory plugin can be used to group resources by metadata, region, OS, etc. Utilizing metadata is highly recommended in “rax” and can provide an easy way to sort between host groups and roles. If you don't want to use the `rax.py` dynamic inventory script, you could also still choose to manually manage your INI inventory file, though this is less recommended.

In Ansible it is quite possible to use multiple dynamic inventory plugins along with INI file data. Just put them in a common directory and be sure the scripts are `chmod +x`, and the INI-based ones are not.

`rax.py`

To use the Rackspace dynamic inventory script, copy `rax.py` into your inventory directory and make it executable. You can specify a credentials file for `rax.py` utilizing the `RAX_CREDS_FILE` environment variable.

注解: Dynamic inventory scripts (like `rax.py`) are saved in `/usr/share/ansible/inventory` if Ansible

has been installed globally. If installed to a virtualenv, the inventory scripts are installed to `$VIRTUALENV/share/inventory`.

注解: Users of *Red Hat Ansible Tower* will note that dynamic inventory is natively supported by Tower, and all you have to do is associate a group with your Rackspace Cloud credentials, and it will easily synchronize without going through these steps:

```
$ RAX_CREDS_FILE=~/.raxpub ansible all -i rax.py -m setup
```

`rax.py` also accepts a `RAX_REGION` environment variable, which can contain an individual region, or a comma separated list of regions.

When using `rax.py`, you will not have a `'localhost'` defined in the inventory.

As mentioned previously, you will often be running most of these modules outside of the host loop, and will need `'localhost'` defined. The recommended way to do this, would be to create an `inventory` directory, and place both the `rax.py` script and a file containing `localhost` in it.

Executing `ansible` or `ansible-playbook` and specifying the `inventory` directory instead of an individual file, will cause ansible to evaluate each file in that directory for inventory.

Let's test our inventory script to see if it can talk to Rackspace Cloud.

```
$ RAX_CREDS_FILE=~/.raxpub ansible all -i inventory/ -m setup
```

Assuming things are properly configured, the `rax.py` inventory script will output information similar to the following information, which will be utilized for inventory and variables.

```
{
  "ORD": [
    "test"
  ],
  "_meta": {
    "hostvars": {
      "test": {
        "ansible_host": "198.51.100.1",
        "rax_accessipv4": "198.51.100.1",
        "rax_accessipv6": "2001:DB8::2342",
        "rax_addresses": {
          "private": [
            {
              "addr": "192.0.2.2",
```

(下页继续)

(续上页)

```

        "version": 4
    },
    ],
    "public": [
        {
            "addr": "198.51.100.1",
            "version": 4
        },
        {
            "addr": "2001:DB8::2342",
            "version": 6
        }
    ]
},
"rax_config_drive": "",
"rax_created": "2013-11-14T20:48:22Z",
"rax_flavor": {
    "id": "performance1-1",
    "links": [
        {
            "href": "https://ord.servers.api.rackspacecloud.com/111111/
↪ flavors/performance1-1",
            "rel": "bookmark"
        }
    ]
},
"rax_hostid":
↪ "e7b6961a9bd943ee82b13816426f1563bfda6846aad84d52af45a4904660cde0",
"rax_human_id": "test",
"rax_id": "099a447b-a644-471f-87b9-a7f580eb0c2a",
"rax_image": {
    "id": "b211c7bf-b5b4-4ede-a8de-a4368750c653",
    "links": [
        {
            "href": "https://ord.servers.api.rackspacecloud.com/111111/
↪ images/b211c7bf-b5b4-4ede-a8de-a4368750c653",
            "rel": "bookmark"
        }
    ]
},
},

```

(下页继续)

```

        "rax_key_name": null,
        "rax_links": [
            {
                "href": "https://ord.servers.api.rackspacecloud.com/v2/111111/
↪servers/099a447b-a644-471f-87b9-a7f580eb0c2a",
                "rel": "self"
            },
            {
                "href": "https://ord.servers.api.rackspacecloud.com/111111/
↪servers/099a447b-a644-471f-87b9-a7f580eb0c2a",
                "rel": "bookmark"
            }
        ],
        "rax_metadata": {
            "foo": "bar"
        },
        "rax_name": "test",
        "rax_name_attr": "name",
        "rax_networks": {
            "private": [
                "192.0.2.2"
            ],
            "public": [
                "198.51.100.1",
                "2001:DB8::2342"
            ]
        },
        "rax_os-dcf_diskconfig": "AUTO",
        "rax_os-ext-sts_power_state": 1,
        "rax_os-ext-sts_task_state": null,
        "rax_os-ext-sts_vm_state": "active",
        "rax_progress": 100,
        "rax_status": "ACTIVE",
        "rax_tenant_id": "111111",
        "rax_updated": "2013-11-14T20:49:27Z",
        "rax_user_id": "22222"
    }
}
}
}

```

Standard Inventory

When utilizing a standard ini formatted inventory file (as opposed to the inventory plugin), it may still be advantageous to retrieve discoverable hostvar information from the Rackspace API.

This can be achieved with the `rax_facts` module and an inventory file similar to the following:

```
[test_servers]
hostname1 rax_region=ORD
hostname2 rax_region=ORD

- name: Gather info about servers
  hosts: test_servers
  gather_facts: False
  tasks:
    - name: Get facts about servers
      rax_facts:
        credentials: ~/.raxpub
        name: "{{ inventory_hostname }}"
        region: "{{ rax_region }}"
        delegate_to: localhost
    - name: Map some facts
      set_fact:
        ansible_host: "{{ rax_accessip4 }}"
```

While you don't need to know how it works, it may be interesting to know what kind of variables are returned.

The `rax_facts` module provides facts as followings, which match the `rax.py` inventory script:

```
{
  "ansible_facts": {
    "rax_accessip4": "198.51.100.1",
    "rax_accessip6": "2001:DB8::2342",
    "rax_addresses": {
      "private": [
        {
          "addr": "192.0.2.2",
          "version": 4
        }
      ],
      "public": [
        {
```

(下页继续)

(续上页)

```

        "addr": "198.51.100.1",
        "version": 4
    },
    {
        "addr": "2001:DB8::2342",
        "version": 6
    }
]
},
"rax_config_drive": "",
"rax_created": "2013-11-14T20:48:22Z",
"rax_flavor": {
    "id": "performance1-1",
    "links": [
        {
            "href": "https://ord.servers.api.rackspacecloud.com/111111/flavors/
↪performance1-1",
            "rel": "bookmark"
        }
    ]
},
"rax_hostid": "e7b6961a9bd943ee82b13816426f1563bfda6846aad84d52af45a4904660cde0",
"rax_human_id": "test",
"rax_id": "099a447b-a644-471f-87b9-a7f580eb0c2a",
"rax_image": {
    "id": "b211c7bf-b5b4-4ede-a8de-a4368750c653",
    "links": [
        {
            "href": "https://ord.servers.api.rackspacecloud.com/111111/images/
↪b211c7bf-b5b4-4ede-a8de-a4368750c653",
            "rel": "bookmark"
        }
    ]
},
"rax_key_name": null,
"rax_links": [
    {
        "href": "https://ord.servers.api.rackspacecloud.com/v2/111111/servers/
↪099a447b-a644-471f-87b9-a7f580eb0c2a",
        "rel": "self"
    }
]

```

(下页继续)

(续上页)

```

    },
    {
      "href": "https://ord.servers.api.rackspacecloud.com/111111/servers/
↪099a447b-a644-471f-87b9-a7f580eb0c2a",
      "rel": "bookmark"
    }
  ],
  "rax_metadata": {
    "foo": "bar"
  },
  "rax_name": "test",
  "rax_name_attr": "name",
  "rax_networks": {
    "private": [
      "192.0.2.2"
    ],
    "public": [
      "198.51.100.1",
      "2001:DB8::2342"
    ]
  },
  "rax_os-dcf_diskconfig": "AUTO",
  "rax_os-ext-sts_power_state": 1,
  "rax_os-ext-sts_task_state": null,
  "rax_os-ext-sts_vm_state": "active",
  "rax_progress": 100,
  "rax_status": "ACTIVE",
  "rax_tenant_id": "111111",
  "rax_updated": "2013-11-14T20:49:27Z",
  "rax_user_id": "22222"
},
"changed": false
}

```

Use Cases

This section covers some additional usage examples built around a specific use case.

Network and Server

Create an isolated cloud network and build a server

```
- name: Build Servers on an Isolated Network
  hosts: localhost
  gather_facts: False
  tasks:
    - name: Network create request
      rax_network:
        credentials: ~/.raxpub
        label: my-net
        cidr: 192.168.3.0/24
        region: IAD
        state: present
        delegate_to: localhost

    - name: Server create request
      rax:
        credentials: ~/.raxpub
        name: web%04d.example.org
        flavor: 2
        image: ubuntu-1204-lts-precise-pangolin
        disk_config: manual
        networks:
          - public
          - my-net
        region: IAD
        state: present
        count: 5
        exact_count: yes
        group: web
        wait: yes
        wait_timeout: 360
        register: rax
        delegate_to: localhost
```

Complete Environment

Build a complete webserver environment with servers, custom networks and load balancers, install nginx and create a custom index.html

```

---
- name: Build environment
  hosts: localhost
  gather_facts: False
  tasks:
    - name: Load Balancer create request
      rax_clb:
        credentials: ~/.raxpub
        name: my-lb
        port: 80
        protocol: HTTP
        algorithm: ROUND_ROBIN
        type: PUBLIC
        timeout: 30
        region: IAD
        wait: yes
        state: present
        meta:
          app: my-cool-app
        register: clb

    - name: Network create request
      rax_network:
        credentials: ~/.raxpub
        label: my-net
        cidr: 192.168.3.0/24
        state: present
        region: IAD
        register: network

    - name: Server create request
      rax:
        credentials: ~/.raxpub
        name: web%04d.example.org
        flavor: performance1-1
        image: ubuntu-1204-lts-precise-pangolin
        disk_config: manual
        networks:
          - public
          - private

```

(下页继续)

(续上页)

```
    - my-net
    region: IAD
    state: present
    count: 5
    exact_count: yes
    group: web
    wait: yes
    register: rax

- name: Add servers to web host group
  add_host:
    hostname: "{{ item.name }}"
    ansible_host: "{{ item.rax_accessip4 }}"
    ansible_password: "{{ item.rax_adminpass }}"
    ansible_user: root
    groups: web
    loop: "{{ rax.success }}"
    when: rax.action == 'create'

- name: Add servers to Load balancer
  rax_clb_nodes:
    credentials: ~/.raxpub
    load_balancer_id: "{{ clb.balancer.id }}"
    address: "{{ item.rax_networks.private|first }}"
    port: 80
    condition: enabled
    type: primary
    wait: yes
    region: IAD
    loop: "{{ rax.success }}"
    when: rax.action == 'create'

- name: Configure servers
  hosts: web
  handlers:
    - name: restart nginx
      service: name=nginx state=restarted

  tasks:
    - name: Install nginx
```

(下页继续)

(续上页)

```

apt: pkg=nginx state=latest update_cache=yes cache_valid_time=86400
notify:
  - restart nginx

- name: Ensure nginx starts on boot
  service: name=nginx state=started enabled=yes

- name: Create custom index.html
  copy: content="{{ inventory_hostname }}" dest=/usr/share/nginx/www/index.html
       owner=root group=root mode=0644

```

RackConnect and Managed Cloud

When using RackConnect version 2 or Rackspace Managed Cloud there are Rackspace automation tasks that are executed on the servers you create after they are successfully built. If your automation executes before the RackConnect or Managed Cloud automation, you can cause failures and unusable servers.

These examples show creating servers, and ensuring that the Rackspace automation has completed before Ansible continues onwards.

For simplicity, these examples are joined, however both are only needed when using RackConnect. When only using Managed Cloud, the RackConnect portion can be ignored.

The RackConnect portions only apply to RackConnect version 2.

Using a Control Machine

```

- name: Create an exact count of servers
  hosts: localhost
  gather_facts: False
  tasks:
    - name: Server build requests
      rax:
        credentials: ~/.raxpub
        name: web%03d.example.org
        flavor: performance1-1
        image: ubuntu-1204-lts-precise-pangolin
        disk_config: manual
        region: DFW
        state: present

```

(下页继续)

(续上页)

```

    count: 1
    exact_count: yes
    group: web
    wait: yes
    register: rax

- name: Add servers to in memory groups
  add_host:
    hostname: "{{ item.name }}"
    ansible_host: "{{ item.rax_accessip4 }}"
    ansible_password: "{{ item.rax_adminpass }}"
    ansible_user: root
    rax_id: "{{ item.rax_id }}"
    groups: web,new_web
    loop: "{{ rax.success }}"
    when: rax.action == 'create'

- name: Wait for rackconnect and managed cloud automation to complete
  hosts: new_web
  gather_facts: false
  tasks:
    - name: ensure we run all tasks from localhost
      delegate_to: localhost
      block:
        - name: Wait for rackconnect automation to complete
          rax_facts:
            credentials: ~/.raxpub
            id: "{{ rax_id }}"
            region: DFW
            register: rax_facts
            until: rax_facts.ansible_facts['rax_metadata']['rackconnect_automation_status
↪']|default('') == 'DEPLOYED'
            retries: 30
            delay: 10

        - name: Wait for managed cloud automation to complete
          rax_facts:
            credentials: ~/.raxpub
            id: "{{ rax_id }}"
            region: DFW

```

(下页继续)

(续上页)

```

        register: rax_facts
        until: rax_facts.ansible_facts['rax_metadata']['rax_service_level_automation
↪']|default('') == 'Complete'
        retries: 30
        delay: 10

- name: Update new_web hosts with IP that RackConnect assigns
  hosts: new_web
  gather_facts: false
  tasks:
    - name: Get facts about servers
      rax_facts:
        name: "{{ inventory_hostname }}"
        region: DFW
        delegate_to: localhost
    - name: Map some facts
      set_fact:
        ansible_host: "{{ rax_accessip4 }}"

- name: Base Configure Servers
  hosts: web
  roles:
    - role: users

    - role: openssh
      opensshd_PermitRootLogin: "no"

    - role: ntp

```

Using Ansible Pull

```

---
- name: Ensure Rackconnect and Managed Cloud Automation is complete
  hosts: all
  tasks:
    - name: ensure we run all tasks from localhost
      delegate_to: localhost
      block:
        - name: Check for completed bootstrap

```

(下页继续)

(续上页)

```

    stat:
      path: /etc/bootstrap_complete
      register: bootstrap

- name: Get region
  command: xenstore-read vm-data/provider_data/region
  register: rax_region
  when: bootstrap.stat.exists != True

- name: Wait for rackconnect automation to complete
  uri:
    url: "https://{{ rax_region.stdout|trim }}.api.rackconnect.rackspace.com/v1/
↪automation_status?format=json"
    return_content: yes
    register: automation_status
    when: bootstrap.stat.exists != True
    until: automation_status['automation_status']|default('') == 'DEPLOYED'
    retries: 30
    delay: 10

- name: Wait for managed cloud automation to complete
  wait_for:
    path: /tmp/rs_managed_cloud_automation_complete
    delay: 10
    when: bootstrap.stat.exists != True

- name: Set bootstrap completed
  file:
    path: /etc/bootstrap_complete
    state: touch
    owner: root
    group: root
    mode: 0400

- name: Base Configure Servers
  hosts: all
  roles:
    - role: users

    - role: openssh

```

(下页继续)

(续上页)

```

    opensshd_PermitRootLogin: "no"

- role: ntp

```

Using Ansible Pull with XenStore

```

---
- name: Ensure Rackconnect and Managed Cloud Automation is complete
  hosts: all
  tasks:
    - name: Check for completed bootstrap
      stat:
        path: /etc/bootstrap_complete
      register: bootstrap

    - name: Wait for rackconnect_automation_status xenstore key to exist
      command: xenstore-exists vm-data/user-metadata/rackconnect_automation_status
      register: rcas_exists
      when: bootstrap.stat.exists != True
      failed_when: rcas_exists.rc|int > 1
      until: rcas_exists.rc|int == 0
      retries: 30
      delay: 10

    - name: Wait for rackconnect automation to complete
      command: xenstore-read vm-data/user-metadata/rackconnect_automation_status
      register: rcas
      when: bootstrap.stat.exists != True
      until: rcas.stdout|replace('"', '') == 'DEPLOYED'
      retries: 30
      delay: 10

    - name: Wait for rax_service_level_automation xenstore key to exist
      command: xenstore-exists vm-data/user-metadata/rax_service_level_automation
      register: rsla_exists
      when: bootstrap.stat.exists != True
      failed_when: rsla_exists.rc|int > 1
      until: rsla_exists.rc|int == 0
      retries: 30

```

(下页继续)

```
    delay: 10

- name: Wait for managed cloud automation to complete
  command: xenstore-read vm-data/user-metadata/rackconnect_automation_status
  register: rsla
  when: bootstrap.stat.exists != True
  until: rsla.stdout|replace('\"', '\"') == 'DEPLOYED'
  retries: 30
  delay: 10

- name: Set bootstrap completed
  file:
    path: /etc/bootstrap_complete
    state: touch
    owner: root
    group: root
    mode: 0400

- name: Base Configure Servers
  hosts: all
  roles:
    - role: users

    - role: openssh
      opensshd_PermitRootLogin: "no"

    - role: ntp
```

Advanced Usage

Autoscaling with Tower

Red Hat Ansible Tower also contains a very nice feature for auto-scaling use cases. In this mode, a simple curl script can call a defined URL and the server will “dial out” to the requester and configure an instance that is spinning up. This can be a great way to reconfigure ephemeral nodes. See the Tower documentation for more details.

A benefit of using the callback in Tower over pull mode is that job results are still centrally recorded and less information has to be shared with remote hosts.

Orchestration in the Rackspace Cloud

Ansible is a powerful orchestration tool, and `rax` modules allow you the opportunity to orchestrate complex tasks, deployments, and configurations. The key here is to automate provisioning of infrastructure, like any other piece of software in an environment. Complex deployments might have previously required manual manipulation of load balancers, or manual provisioning of servers. Utilizing the `rax` modules included with Ansible, one can make the deployment of additional nodes contingent on the current number of running nodes, or the configuration of a clustered application dependent on the number of nodes with common metadata. One could automate the following scenarios, for example:

- Servers that are removed from a Cloud Load Balancer one-by-one, updated, verified, and returned to the load balancer pool
- Expansion of an already-online environment, where nodes are provisioned, bootstrapped, configured, and software installed
- A procedure where app log files are uploaded to a central location, like Cloud Files, before a node is decommissioned
- Servers and load balancers that have DNS records created and destroyed on creation and decommissioning, respectively

1.6.10 Scaleway Guide

Introduction

[Scaleway](#) is a cloud provider supported by Ansible, version 2.6 or higher via a dynamic inventory plugin and modules. Those modules are:

- `scaleway_sshkey_module`: adds a public SSH key from a file or value to the Packet infrastructure. Every subsequently-created device will have this public key installed in `.ssh/authorized_keys`.
- `scaleway_compute_module`: manages servers on Scaleway. You can use this module to create, restart and delete servers.
- `scaleway_volume_module`: manages volumes on Scaleway.

注解: This guide assumes you are familiar with Ansible and how it works. If you're not, have a look at *Ansible* 「2.9」 中文官方文档 before getting started.

Requirements

The Scaleway modules and inventory script connect to the Scaleway API using [Scaleway REST API](#). To use the modules and inventory script you'll need a Scaleway API token. You can generate an API token via

the Scaleway console [here](#). The simplest way to authenticate yourself is to set the Scaleway API token in an environment variable:

```
$ export SCW_TOKEN=00000000-1111-2222-3333-444444444444
```

If you're not comfortable exporting your API token, you can pass it as a parameter to the modules using the `api_token` argument.

If you want to use a new SSH keypair in this tutorial, you can generate it to `./id_rsa` and `./id_rsa.pub` as:

```
$ ssh-keygen -t rsa -f ./id_rsa
```

If you want to use an existing keypair, just copy the private and public key over to the playbook directory.

How to add an SSH key?

Connection to Scaleway Compute nodes use Secure Shell. SSH keys are stored at the account level, which means that you can re-use the same SSH key in multiple nodes. The first step to configure Scaleway compute resources is to have at least one SSH key configured.

`scaleway_sshkey` module is a module that manages SSH keys on your Scaleway account. You can add an SSH key to your account by including the following task in a playbook:

```
- name: "Add SSH key"
  scaleway_sshkey:
    ssh_pub_key: "ssh-rsa AAAA..."
    state: "present"
```

The `ssh_pub_key` parameter contains your ssh public key as a string. Here is an example inside a playbook:

```
# SCW_API_KEY='XXX' ansible-playbook ./test/legacy/scaleway_ssh_playbook.yml

- name: Test SSH key lifecycle on a Scaleway account
  hosts: localhost
  gather_facts: no
  environment:
    SCW_API_KEY: ""

  tasks:

    - scaleway_sshkey:
        ssh_pub_key: "ssh-rsa AAAAB...424242 developer@example.com"
```

(下页继续)

(续上页)

```

    state: present
    register: result

- assert:
    that:
        - result is success and result is changed

```

How to create a compute instance?

Now that we have an SSH key configured, the next step is to spin up a server! `scaleway_compute` module is a module that can create, update and delete Scaleway compute instances:

```

- name: Create a server
  scaleway_compute:
    name: foobar
    state: present
    image: 00000000-1111-2222-3333-444444444444
    organization: 00000000-1111-2222-3333-444444444444
    region: ams1
    commercial_type: START1-S

```

Here are the parameter details for the example shown above:

- `name` is the name of the instance (the one that will show up in your web console).
- `image` is the UUID of the system image you would like to use. A list of all images is available for each availability zone.
- `organization` represents the organization that your account is attached to.
- `region` represents the Availability Zone which your instance is in (for this example, `par1` and `ams1`).
- `commercial_type` represents the name of the commercial offers. You can check out the Scaleway pricing page to find which instance is right for you.

Take a look at this short playbook to see a working example using `scaleway_compute`:

```

# SCW_TOKEN='XXX' ansible-playbook ./test/legacy/scaleway_compute.yml

- name: Test compute instance lifecycle on a Scaleway account
  hosts: localhost
  gather_facts: no
  environment:
    SCW_API_KEY: ""

```

(下页继续)

```
tasks:

- name: Create a server
  register: server_creation_task
  scaleway_compute:
    name: foobar
    state: present
    image: 00000000-1111-2222-3333-444444444444
    organization: 00000000-1111-2222-3333-444444444444
    region: ams1
    commercial_type: START1-S
    wait: true

- debug: var=server_creation_task

- assert:
  that:
    - server_creation_task is success
    - server_creation_task is changed

- name: Run it
  scaleway_compute:
    name: foobar
    state: running
    image: 00000000-1111-2222-3333-444444444444
    organization: 00000000-1111-2222-3333-444444444444
    region: ams1
    commercial_type: START1-S
    wait: true
    tags:
      - web_server
  register: server_run_task

- debug: var=server_run_task

- assert:
  that:
    - server_run_task is success
    - server_run_task is changed
```

Dynamic Inventory Script

Ansible ships with `scaleway_inventory`. You can now get a complete inventory of your Scaleway resources through this plugin and filter it on different parameters (`regions` and `tags` are currently supported).

Let's create an example! Suppose that we want to get all hosts that got the tag `web_server`. Create a file named `scaleway_inventory.yml` with the following content:

```
plugin: scaleway
regions:
  - ams1
  - par1
tags:
  - web_server
```

This inventory means that we want all hosts that got the tag `web_server` on the zones `ams1` and `par1`. Once you have configured this file, you can get the information using the following command:

```
$ ansible-inventory --list -i scaleway_inventory.yml
```

The output will be:

```
{
  "_meta": {
    "hostvars": {
      "dd8e3ae9-0c7c-459e-bc7b-aba8bfa1bb8d": {
        "ansible_verbosity": 6,
        "arch": "x86_64",
        "commercial_type": "START1-S",
        "hostname": "foobar",
        "ipv4": "192.0.2.1",
        "organization": "00000000-1111-2222-3333-444444444444",
        "state": "running",
        "tags": [
          "web_server"
        ]
      }
    }
  },
  "all": {
    "children": [
      "ams1",
      "par1",
```

(下页继续)

(续上页)

```

        "ungrouped",
        "web_server"
    ]
},
"ams1": {},
"par1": {
    "hosts": [
        "dd8e3ae9-0c7c-459e-bc7b-aba8bfa1bb8d"
    ]
},
"ungrouped": {},
"web_server": {
    "hosts": [
        "dd8e3ae9-0c7c-459e-bc7b-aba8bfa1bb8d"
    ]
}
}

```

As you can see, we get different groups of hosts. `par1` and `ams1` are groups based on location. `web_server` is a group based on a tag.

In case a filter parameter is not defined, the plugin supposes all values possible are wanted. This means that for each tag that exists on your Scaleway compute nodes, a group based on each tag will be created.

Scaleway S3 object storage

Object Storage allows you to store any kind of objects (documents, images, videos, etc.). As the Scaleway API is S3 compatible, Ansible supports it natively through the modules: `s3_bucket_module`, `aws_s3_module`.

You can find many examples in `./test/legacy/roles/scaleway_s3`

```

- hosts: myserver
  vars:
    scaleway_region: nl-ams
    s3_url: https://s3.nl-ams.scw.cloud
  environment:
    # AWS_ACCESS_KEY matches your scaleway organization id available at https://cloud.
    ↪ scaleway.com/#/account
    AWS_ACCESS_KEY: 00000000-1111-2222-3333-444444444444
    # AWS_SECRET_KEY matches a secret token that you can retrieve at https://cloud.
    ↪ scaleway.com/#/credentials

```

(下页继续)

(续上页)

```

AWS_SECRET_KEY: aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeeeee
module_defaults:
  group/aws:
    s3_url: '{{ s3_url }}'
    region: '{{ scaleway_region }}'
tasks:
  # use a fact instead of a variable, otherwise template is evaluate each time variable_
  ↳ is used
  - set_fact:
    bucket_name: "{{ 99999999 | random | to_uuid }}"

  # "requester_pays:" is mandatory because Scaleway doesn't implement related API
  # another way is to use aws_s3 and "mode: create" !
  - s3_bucket:
    name: '{{ bucket_name }}'
    requester_pays:

  - name: Another way to create the bucket
    aws_s3:
    bucket: '{{ bucket_name }}'
    mode: create
    encrypt: false
    register: bucket_creation_check

  - name: add something in the bucket
    aws_s3:
    mode: put
    bucket: '{{ bucket_name }}'
    src: /tmp/test.txt # needs to be created before
    object: test.txt
    encrypt: false # server side encryption must be disabled

```

1.6.11 Vultr Guide

Ansible offers a set of modules to interact with [Vultr](#) cloud platform.

This set of module forms a framework that allows one to easily manage and orchestrate one's infrastructure on Vultr cloud platform.

Requirements

There is actually no technical requirement; simply an already created Vultr account.

Configuration

Vultr modules offer a rather flexible way with regard to configuration.

Configuration is read in that order:

- Environment Variables (eg. `VULTR_API_KEY`, `VULTR_API_TIMEOUT`)
- File specified by environment variable `VULTR_API_CONFIG`
- `vultr.ini` file located in current working directory
- `$HOME/.vultr.ini`

Ini file are structured this way:

```
[default]
key = MY_API_KEY
timeout = 60

[personal_account]
key = MY_PERSONAL_ACCOUNT_API_KEY
timeout = 30
```

If `VULTR_API_ACCOUNT` environment variable or `api_account` module parameter is not specified, modules will look for the section named “default” .

Authentication

Before using the Ansible modules to interact with Vultr, ones need an API key. If one doesn’ t own one yet, log in to [Vultr](#) go to Account, then API, enable API then the API key should show up.

Ensure you allow the usage of the API key from the proper IP addresses.

Refer to the Configuration section to find out where to put this information.

To check that everything is working properly run the following command:

```
#> VULTR_API_KEY=XXX ansible -m vultr_account_info localhost
localhost | SUCCESS => {
  "changed": false,
  "vultr_account_info": {
    "balance": -8.9,
```

(下页继续)

(续上页)

```

    "last_payment_amount": -10.0,
    "last_payment_date": "2018-07-21 11:34:46",
    "pending_charges": 6.0
  },
  "vultr_api": {
    "api_account": "default",
    "api_endpoint": "https://api.vultr.com",
    "api_retries": 5,
    "api_timeout": 60
  }
}

```

If a similar output displays then everything is setup properly, else please ensure the proper `VULTR_API_KEY` has been specified and that Access Control on Vultr > Account > API page are accurate.

Usage

Since [Vultr](#) offers a public API, the execution of the module to manage the infrastructure on their platform will happen on localhost. This translates to:

```

---
- hosts: localhost
  tasks:
    - name: Create a 10G volume
      vultr_block_storage:
        name: my_disk
        size: 10
        region: New Jersey

```

From that point on, only your creativity is the limit. Make sure to read the documentation of the [available modules](#).

Dynamic Inventory

Ansible provides a dynamic inventory plugin for [Vultr](#). The configuration process is exactly the same as the one for the modules.

To be able to use it you need to enable it first by specifying the following in the `ansible.cfg` file:

```

[inventory]
enable_plugins=vultr

```

And provide a configuration file to be used with the plugin, the minimal configuration file looks like this:

```
---
plugin: vultr
```

To list the available hosts one can simply run:

```
#> ansible-inventory -i vultr.yml --list
```

For example, this allows you to take action on nodes grouped by location or OS name:

```
---
- hosts: Amsterdam
  tasks:
    - name: Rebooting the machine
      shell: reboot
      become: True
```

Integration tests

Ansible includes integration tests for all Vultr modules.

These tests are meant to run against the public Vultr API and that is why they require a valid key to access the API.

Prepare the test setup:

```
$ cd ansible # location the ansible source is
$ source ./hacking/env-setup
```

Set the Vultr API key:

```
$ cd test/integration
$ cp cloud-config-vultr.ini.template cloud-config-vultr.ini
$ vi cloud-config-vultr.ini
```

Run all Vultr tests:

```
$ ansible-test integration cloud/vultr/ -v --diff --allow-unsupported
```

To run a specific test, e.g. `vultr_account_info`:

```
$ ansible-test integration cloud/vultr/vultr_account_info -v --diff --allow-unsupported
```


1.7 Network Technology Guides

The guides in this section cover using Ansible with specific network technologies. They explore particular use cases in greater depth and provide a more “top-down” explanation of some basic features.

1.7.1 Cisco ACI Guide

What is Cisco ACI ?

Application Centric Infrastructure (ACI)

The Cisco Application Centric Infrastructure (ACI) allows application requirements to define the network. This architecture simplifies, optimizes, and accelerates the entire application deployment life cycle.

Application Policy Infrastructure Controller (APIC)

The APIC manages the scalable ACI multi-tenant fabric. The APIC provides a unified point of automation and management, policy programming, application deployment, and health monitoring for the fabric. The APIC, which is implemented as a replicated synchronized clustered controller, optimizes performance, supports any application anywhere, and provides unified operation of the physical and virtual infrastructure.

The APIC enables network administrators to easily define the optimal network for applications. Data center operators can clearly see how applications consume network resources, easily isolate and troubleshoot application and infrastructure problems, and monitor and profile resource usage patterns.

The Cisco Application Policy Infrastructure Controller (APIC) API enables applications to directly connect with a secure, shared, high-performance resource pool that includes network, compute, and storage capabilities.

ACI Fabric

The Cisco Application Centric Infrastructure (ACI) Fabric includes Cisco Nexus 9000 Series switches with the APIC to run in the leaf/spine ACI fabric mode. These switches form a “fat-tree” network by connecting each leaf node to each spine node; all other devices connect to the leaf nodes. The APIC manages the ACI fabric.

The ACI fabric provides consistent low-latency forwarding across high-bandwidth links (40 Gbps, with a 100-Gbps future capability). Traffic with the source and destination on the same leaf switch is handled locally, and all other traffic travels from the ingress leaf to the egress leaf through a spine switch. Although this architecture appears as two hops from a physical perspective, it is actually a single Layer 3 hop because the fabric operates as a single Layer 3 switch.

The ACI fabric object-oriented operating system (OS) runs on each Cisco Nexus 9000 Series node. It enables programming of objects for each configurable element of the system. The ACI fabric OS renders policies from the APIC into a concrete model that runs in the physical infrastructure. The concrete model is analogous to compiled software; it is the form of the model that the switch operating system can execute.

All the switch nodes contain a complete copy of the concrete model. When an administrator creates a policy in the APIC that represents a configuration, the APIC updates the logical model. The APIC then performs the intermediate step of creating a fully elaborated policy that it pushes into all the switch nodes where the concrete model is updated.

The APIC is responsible for fabric activation, switch firmware management, network policy configuration, and instantiation. While the APIC acts as the centralized policy and network management engine for the fabric, it is completely removed from the data path, including the forwarding topology. Therefore, the fabric can still forward traffic even when communication with the APIC is lost.

More information

Various resources exist to start learning ACI, here is a list of interesting articles from the community.

- [Adam Raffae: Learning ACI](#)
- [Luca Relandini: ACI for dummies](#)
- [Cisco DevNet Learning Labs about ACI](#)

Using the ACI modules

The Ansible ACI modules provide a user-friendly interface to managing your ACI environment using Ansible playbooks.

For instance ensuring that a specific tenant exists, is done using the following Ansible task using module `aci_tenant`:

```
- name: Ensure tenant customer-xyz exists
  aci_tenant:
    host: my-apic-1
    username: admin
    password: my-password

    tenant: customer-xyz
    description: Customer XYZ
    state: present
```

A complete list of existing ACI modules is available for the latest stable release on the list of network modules. You can also view the [current development version](#).

If you want to learn how to write your own ACI modules to contribute, look at the *Developing Cisco ACI modules* section.

Querying ACI configuration

A module can also be used to query a specific object.

```
- name: Query tenant customer-xyz
  aci_tenant:
    host: my-apic-1
    username: admin
    password: my-password

    tenant: customer-xyz
    state: query
  register: my_tenant
```

Or query all objects.

```
- name: Query all tenants
  aci_tenant:
    host: my-apic-1
    username: admin
    password: my-password

    state: query
  register: all_tenants
```

After registering the return values of the `aci_tenant` task as shown above, you can access all tenant information from variable `all_tenants`.

Running on the controller locally

As originally designed, Ansible modules are shipped to and run on the remote target(s), however the ACI modules (like most network-related modules) do not run on the network devices or controller (in this case the APIC), but they talk directly to the APIC's REST interface.

For this very reason, the modules need to run on the local Ansible controller (or are delegated to another system that *can* connect to the APIC).

Gathering facts

Because we run the modules on the Ansible controller gathering facts will not work. That is why when using these ACI modules it is mandatory to disable facts gathering. You can do this globally in your `ansible.cfg` or by adding `gather_facts: no` to every play.

```
- name: Another play in my playbook
  hosts: my-apic-1
  gather_facts: no
  tasks:
    - name: Create a tenant
      aci_tenant:
        ...
```

Delegating to localhost

So let us assume we have our target configured in the inventory using the FQDN name as the `ansible_host` value, as shown below.

```
apics:
  my-apic-1:
    ansible_host: apic01.fqdn.intra
    ansible_user: admin
    ansible_password: my-password
```

One way to set this up is to add to every task the directive: `delegate_to: localhost`.

```
- name: Query all tenants
  aci_tenant:
    host: '{{ ansible_host }}'
    username: '{{ ansible_user }}'
    password: '{{ ansible_password }}'

    state: query
  delegate_to: localhost
  register: all_tenants
```

If one would forget to add this directive, Ansible will attempt to connect to the APIC using SSH and attempt to copy the module and run it remotely. This will fail with a clear error, yet may be confusing to some.

Using the local connection method

Another option frequently used, is to tie the `local` connection method to this target so that every subsequent task for this target will use the local connection method (hence run it locally, rather than use SSH).

In this case the inventory may look like this:

```
apics:
  my-apic-1:
    ansible_host: apic01.fqdn.intra
    ansible_user: admin
    ansible_password: my-password
    ansible_connection: local
```

But used tasks do not need anything special added.

```
- name: Query all tenants
  aci_tenant:
    host: '{{ ansible_host }}'
    username: '{{ ansible_user }}'
    password: '{{ ansible_password }}'

    state: query
  register: all_tenants
```

提示: For clarity we have added `delegate_to: localhost` to all the examples in the module documentation. This helps to ensure first-time users can easily copy&paste parts and make them work with a minimum of effort.

Common parameters

Every Ansible ACI module accepts the following parameters that influence the module's communication with the APIC REST API:

host Hostname or IP address of the APIC.

port Port to use for communication. (Defaults to 443 for HTTPS, and 80 for HTTP)

username User name used to log on to the APIC. (Defaults to `admin`)

password Password for `username` to log on to the APIC, using password-based authentication.

private_key Private key for `username` to log on to APIC, using signature-based authentication.

This could either be the raw private key content (include header/footer) or a file that stores

the key content. *New in version 2.5*

certificate_name Name of the certificate in the ACI Web GUI. This defaults to either the **username** value or the **private_key** file base name). *New in version 2.5*

timeout Timeout value for socket-level communication.

use_proxy Use system proxy settings. (Defaults to **yes**)

use_ssl Use HTTPS or HTTP for APIC REST communication. (Defaults to **yes**)

validate_certs Validate certificate when using HTTPS communication. (Defaults to **yes**)

output_level Influence the level of detail ACI modules return to the user. (One of **normal**, **info** or **debug**) *New in version 2.5*

Proxy support

By default, if an environment variable `<protocol>_proxy` is set on the target host, requests will be sent through that proxy. This behaviour can be overridden by setting a variable for this task (see [Setting the Environment \(and Working With Proxies\)](#)), or by using the `use_proxy` module parameter.

HTTP redirects can redirect from HTTP to HTTPS so ensure that the proxy environment for both protocols is correctly configured.

If proxy support is not needed, but the system may have it configured nevertheless, use the parameter `use_proxy: no` to avoid accidental system proxy usage.

提示: Selective proxy support using the `no_proxy` environment variable is also supported.

Return values

2.5 新版功能.

The following values are always returned:

current The resulting state of the managed object, or results of your query.

The following values are returned when `output_level: info`:

previous The original state of the managed object (before any change was made).

proposed The proposed config payload, based on user-supplied values.

sent The sent config payload, based on user-supplied values and the existing configuration.

The following values are returned when `output_level: debug` or `ANSIBLE_DEBUG=1`:

filter_string The filter used for specific APIC queries.

method The HTTP method used for the sent payload. (Either GET for queries, DELETE or POST for changes)

response The HTTP response from the APIC.

status The HTTP status code for the request.

url The url used for the request.

注解: The module return values are documented in detail as part of each module's documentation.

More information

Various resources exist to start learn more about ACI programmability, we recommend the following links:

- [Developing Cisco ACI modules](#)
- [Jacob McGill: Automating Cisco ACI with Ansible](#)
- [Cisco DevNet Learning Labs about ACI and Ansible](#)

ACI authentication

Password-based authentication

If you want to log on using a username and password, you can use the following parameters with your ACI modules:

```
username: admin
password: my-password
```

Password-based authentication is very simple to work with, but it is not the most efficient form of authentication from ACI's point-of-view as it requires a separate login-request and an open session to work. To avoid having your session time-out and requiring another login, you can use the more efficient Signature-based authentication.

注解: Password-based authentication also may trigger anti-DoS measures in ACI v3.1+ that causes session throttling and results in HTTP 503 errors and login failures.

警告: Never store passwords in plain text.

The “Vault” feature of Ansible allows you to keep sensitive data such as passwords or keys in encrypted files, rather than as plain text in your playbooks or roles. These vault files can then be distributed or placed in source control. See *Using Vault in playbooks* for more information.

Signature-based authentication using certificates

2.5 新版功能.

Using signature-based authentication is more efficient and more reliable than password-based authentication.

Generate certificate and private key

Signature-based authentication requires a (self-signed) X.509 certificate with private key, and a configuration step for your AAA user in ACI. To generate a working X.509 certificate and private key, use the following procedure:

```
$ openssl req -new -newkey rsa:1024 -days 36500 -nodes -x509 -keyout admin.key -out ↵
↵admin.crt -subj '/CN=Admin/O=Your Company/C=US'
```

Configure your local user

Perform the following steps:

- Add the X.509 certificate to your ACI AAA local user at *ADMIN » AAA*
- Click *AAA Authentication*
- Check that in the *Authentication* field the *Realm* field displays *Local*
- Expand *Security Management » Local Users*
- Click the name of the user you want to add a certificate to, in the *User Certificates* area
- Click the *+* sign and in the *Create X509 Certificate* enter a certificate name in the *Name* field
 - If you use the basename of your private key here, you don’ t need to enter `certificate_name` in Ansible
- Copy and paste your X.509 certificate in the *Data* field.

You can automate this by using the following Ansible task:

```
- name: Ensure we have a certificate installed
  aci_aaa_user_certificate:
    host: my-apic-1
    username: admin
```

(下页继续)

(续上页)

```

password: my-password

aaa_user: admin
certificate_name: admin
certificate: "{{ lookup('file', 'pki/admin.crt') }}" # This will read the
↪ certificate data from a local file

```

注解: Signature-based authentication only works with local users.

Use signature-based authentication with Ansible

You need the following parameters with your ACI module(s) for it to work:

```

username: admin
private_key: pki/admin.key
certificate_name: admin # This could be left out !

```

or you can use the private key content:

```

username: admin
private_key: |
    -----BEGIN PRIVATE KEY-----
    <<your private key content>>
    -----END PRIVATE KEY-----
certificate_name: admin # This could be left out !

```

提示: If you use a certificate name in ACI that matches the private key's basename, you can leave out the `certificate_name` parameter like the example above.

Using Ansible Vault to encrypt the private key

2.8 新版功能.

To start, encrypt the private key and give it a strong password.

```
ansible-vault encrypt admin.key
```

Use a text editor to open the private-key. You should have an encrypted cert now.

```
$ANSIBLE_VAULT;1.1;AES256
56484318584354658465121889743213151843149454864654151618131547984132165489484654
45641818198456456489479874513215489484843614848456466655432455488484654848489498
....
```

Copy and paste the new encrypted cert into your playbook as a new variable.

```
private_key: !vault |
    $ANSIBLE_VAULT;1.1;AES256
    56484318584354658465121889743213151843149454864654151618131547984132165489484654
    45641818198456456489479874513215489484843614848456466655432455488484654848489498
    ....
```

Use the new variable for the `private_key`:

```
username: admin
private_key: "{{ private_key }}"
certificate_name: admin # This could be left out !
```

When running the playbook, use “`--ask-vault-pass`” to decrypt the private key.

```
ansible-playbook site.yaml --ask-vault-pass
```

More information

- Detailed information about Signature-based Authentication is available from [Cisco APIC Signature-Based Transactions](#).
- More information on Ansible Vault can be found on the [Ansible Vault](#) page.

Using ACI REST with Ansible

While already a lot of ACI modules exists in the Ansible distribution, and the most common actions can be performed with these existing modules, there’s always something that may not be possible with off-the-shelf modules.

The `aci_rest` module provides you with direct access to the APIC REST API and enables you to perform any task not already covered by the existing modules. This may seem like a complex undertaking, but you can generate the needed REST payload for any action performed in the ACI web interface effortlessly.

Built-in idempotency

Because the APIC REST API is intrinsically idempotent and can report whether a change was made, the `aci_rest` module automatically inherits both capabilities and is a first-class solution for automating your ACI infrastructure. As a result, users that require more powerful low-level access to their ACI infrastructure don't have to give up on idempotency and don't have to guess whether a change was performed when using the `aci_rest` module.

Using the `aci_rest` module

The `aci_rest` module accepts the native XML and JSON payloads, but additionally accepts inline YAML payload (structured like JSON). The XML payload requires you to use a path ending with `.xml` whereas JSON or YAML require the path to end with `.json`.

When you're making modifications, you can use the POST or DELETE methods, whereas doing just queries require the GET method.

For instance, if you would like to ensure a specific tenant exists on ACI, these below four examples are functionally identical:

XML (Native ACI REST)

```
- aci_rest:
    host: my-apic-1
    private_key: pki/admin.key

    method: post
    path: /api/mo/uni.xml
    content: |
        <fvTenant name="customer-xyz" descr="Customer XYZ"/>
```

JSON (Native ACI REST)

```
- aci_rest:
    host: my-apic-1
    private_key: pki/admin.key

    method: post
    path: /api/mo/uni.json
    content:
        {
            "fvTenant": {
                "attributes": {
```

(下页继续)

(续上页)

```
    "name": "customer-xyz",
    "descr": "Customer XYZ"
  }
}
```

YAML (Ansible-style REST)

```
- aci_rest:
  host: my-apic-1
  private_key: pki/admin.key

  method: post
  path: /api/mo/uni.json
  content:
    fvTenant:
      attributes:
        name: customer-xyz
        descr: Customer XYZ
```

Ansible task (Dedicated module)

```
- aci_tenant:
  host: my-apic-1
  private_key: pki/admin.key

  tenant: customer-xyz
  description: Customer XYZ
  state: present
```

提示: The XML format is more practical when there is a need to template the REST payload (inline), but the YAML format is more convenient for maintaining your infrastructure-as-code and feels more naturally integrated with Ansible playbooks. The dedicated modules offer a more simple, abstracted, but also a more limited experience. Use what feels best for your use-case.

More information

Plenty of resources exist to learn about ACI' s APIC REST interface, we recommend the links below:

- The `aci_rest` module documentation

- [APIC REST API Configuration Guide](#) – Detailed guide on how the APIC REST API is designed and used, incl. many examples
- [APIC Management Information Model reference](#) – Complete reference of the APIC object model
- [Cisco DevNet Learning Labs about ACI and REST](#)

Operational examples

Here is a small overview of useful operational tasks to reuse in your playbooks.

Feel free to contribute more useful snippets.

Waiting for all controllers to be ready

You can use the below task after you started to build your APICs and configured the cluster to wait until all the APICs have come online. It will wait until the number of controllers equals the number listed in the apic inventory group.

```
- name: Waiting for all controllers to be ready
  aci_rest:
    host: my-apic-1
    private_key: pki/admin.key
    method: get
    path: /api/node/class/topSystem.json?query-target-filter=eq(topSystem.role,
    ↪ "controller")
    register: topsystem
    until: topsystem|success and topsystem.totalCount|int >= groups['apic']|count >= 3
    retries: 20
    delay: 30
```

Waiting for cluster to be fully-fit

The below example waits until the cluster is fully-fit. In this example you know the number of APICs in the cluster and you verify each APIC reports a ‘fully-fit’ status.

```
- name: Waiting for cluster to be fully-fit
  aci_rest:
    host: my-apic-1
    private_key: pki/admin.key
    method: get
    path: /api/node/class/infraWiNode.json?query-target-filter=wcard(infraWiNode.dn,
    ↪ "topology/pod-1/node-1/av")
```

(下页继续)

(续上页)

```

register: infrawinode
until: >
    infrawinode|success and
    infrawinode.totalCount|int >= groups['apic']|count >= 3 and
    infrawinode.imdata[0].infraWiNode.attributes.health == 'fully-fit' and
    infrawinode.imdata[1].infraWiNode.attributes.health == 'fully-fit' and
    infrawinode.imdata[2].infraWiNode.attributes.health == 'fully-fit'
retries: 30
delay: 30

```

APIC error messages

The following error messages may occur and this section can help you understand what exactly is going on and how to fix/avoid them.

APIC Error 122: unknown managed object class ‘polUni’ In case you receive this error while you are certain your aci_rest payload and object classes are seemingly correct, the issue might be that your payload is not in fact correct JSON (e.g. the sent payload is using single quotes, rather than double quotes), and as a result the APIC is not correctly parsing your object classes from the payload. One way to avoid this is by using a YAML or an XML formatted payload, which are easier to construct correctly and modify later.

APIC Error 400: invalid data at line ‘1’ . Attributes are missing, tag ‘attributes’ must be specified

Although the JSON specification allows unordered elements, the APIC REST API requires that the JSON `attributes` element precede the `children` array or other elements. So you need to ensure that your payload conforms to this requirement. Sorting your dictionary keys will do the trick just fine. If you don’t have any attributes, it may be necessary to add: `attributes: {}` as the APIC does expect the entry to precede any `children`.

APIC Error 801: property descr of uni/tn-TENANT/ap-AP failed validation for value ‘A “legacy” r

Some values in the APIC have strict format-rules to comply to, and the internal APIC validation check for the provided value failed. In the above case, the `description` parameter (internally known as `descr`) only accepts values conforming to [Regex](#): `[a-zA-Z0-9\!#$%()*,-./:;@ _{}~?&+]+`, in general it must not include quotes or square brackets.

Known issues

The `aci_rest` module is a wrapper around the APIC REST API. As a result any issues related to the APIC will be reflected in the use of this module.

All below issues either have been reported to the vendor, and most can simply be avoided.

Too many consecutive API calls may result in connection throttling Starting with ACI v3.1 the APIC will actively throttle password-based authenticated connection rates over a specific threshold. This is as part of an anti-DDOS measure but can act up when using Ansible with ACI using password-based authentication. Currently, one solution is to increase this threshold within the nginx configuration, but using signature-based authentication is recommended.

NOTE: It is advisable to use signature-based authentication with ACI as it not only prevents connection-throttling, but also improves general performance when using the ACI modules.

Specific requests may not reflect changes correctly (#35401) There is a known issue where specific requests to the APIC do not properly reflect changed in the resulting output, even when we request those changes explicitly from the APIC. In one instance using the path `api/node/mo/uni/infra.xml` fails, where `api/node/mo/uni/infra/.xml` does work correctly.

NOTE: A workaround is to register the task return values (e.g. `register: this`) and influence when the task should report a change by adding: `changed_when: this.imdata != []`.

Specific requests are known to not be idempotent (#35050) The behaviour of the APIC is inconsistent to the use of `status="created"` and `status="deleted"`. The result is that when you use `status="created"` in your payload the resulting tasks are not idempotent and creation will fail when the object was already created. However this is not the case with `status="deleted"` where such call to an non-existing object does not cause any failure whatsoever.

NOTE: A workaround is to avoid using `status="created"` and instead use `status="modified"` when idempotency is essential to your workflow..

Setting user password is not idempotent (#35544) Due to an inconsistency in the APIC REST API, a task that sets the password of a locally-authenticated user is not idempotent. The APIC will complain with message `Password history check: user dag should not use previous 5 passwords`.

NOTE: There is no workaround for this issue.

ACI Ansible community

If you have specific issues with the ACI modules, or a feature request, or you like to contribute to the ACI project by proposing changes or documentation updates, look at the Ansible Community wiki ACI page at: <https://github.com/ansible/community/wiki/Network:-ACI>

You will find our roadmap, an overview of open ACI issues and pull-requests, and more information about who we are. If you have an interest in using ACI with Ansible, feel free to join! We occasionally meet online

to track progress and prepare for new Ansible releases.

参见:

List of ACI modules A complete list of supported ACI modules.

Developing Cisco ACI modules A walkthrough on how to develop new Cisco ACI modules to contribute back.

ACI community The Ansible ACI community wiki page, includes roadmap, ideas and development documentation.

network_guide A detailed guide on how to use Ansible for automating network infrastructure.

Network Working Group The Ansible Network community page, includes contact information and meeting information.

#ansible-network The #ansible-network IRC chat channel on Freenode.net.

User Mailing List Have a question? Stop by the google group!

1.7.2 Cisco Meraki Guide

- *What is Cisco Meraki?*
 - *MS Switches*
 - *MX Firewalls*
 - *MR Wireless Access Points*
- *Using the Meraki modules*
- *Common Parameters*
- *Meraki Authentication*
- *Returned Data Structures*
- *Handling Returned Data*
- *Merging Existing and New Data*
- *Error Handling*

What is Cisco Meraki?

Cisco Meraki is an easy-to-use, cloud-based, network infrastructure platform for enterprise environments. While most network hardware uses command-line interfaces (CLIs) for configuration, Meraki uses an easy-to-use Dashboard hosted in the Meraki cloud. No on-premises management hardware or software is required

- only the network infrastructure to run your business.

MS Switches

Meraki MS switches come in multiple flavors and form factors. Meraki switches support 10/100/1000/10000 ports, as well as Cisco's mGig technology for 2.5/5/10Gbps copper connectivity. 8, 24, and 48 port flavors are available with PoE (802.3af/802.3at/UPoE) available on many models.

MX Firewalls

Meraki's MX firewalls support full layer 3-7 deep packet inspection. MX firewalls are compatible with a variety of VPN technologies including IPSec, SSL VPN, and Meraki's easy-to-use AutoVPN.

MR Wireless Access Points

MR access points are enterprise-class, high-performance access points for the enterprise. MR access points have MIMO technology and integrated beamforming built-in for high performance applications. BLE allows for advanced location applications to be developed with no on-premises analytics platforms.

Using the Meraki modules

Meraki modules provide a user-friendly interface to manage your Meraki environment using Ansible. For example, details about SNMP settings for a particular organization can be discovered using the module *meraki_snmp* <*meraki_snmp_module*>.

```
- name: Query SNMP settings
  meraki_snmp:
    api_key: abc123
    org_name: AcmeCorp
    state: query
    delegate_to: localhost
```

Information about a particular object can be queried. For example, the *meraki_admin* <*meraki_admin_module*> module supports

```
- name: Gather information about Jane Doe
  meraki_admin:
    api_key: abc123
    org_name: AcmeCorp
    state: query
```

(下页继续)

(续上页)

```
email: janedoe@email.com
delegate_to: localhost
```

Common Parameters

All Ansible Meraki modules support the following parameters which affect communication with the Meraki Dashboard API. Most of these should only be used by Meraki developers and not the general public.

host Hostname or IP of Meraki Dashboard.

use_https Specifies whether communication should be over HTTPS. (Defaults to **yes**)

use_proxy Whether to use a proxy for any communication.

validate_certs Determine whether certificates should be validated or trusted. (Defaults to **yes**)

These are the common parameters which are used for most every module.

org_name Name of organization to perform actions in.

org_id ID of organization to perform actions in.

net_name Name of network to perform actions in.

net_id ID of network to perform actions in.

state General specification of what action to take. **query** does lookups. **present** creates or edits. **absent** deletes.

提示: Use the **org_id** and **net_id** parameters when possible. **org_name** and **net_name** require additional behind-the-scenes API calls to learn the ID values. **org_id** and **net_id** will perform faster.

Meraki Authentication

All API access with the Meraki Dashboard requires an API key. An API key can be generated from the organization's settings page. Each play in a playbook requires the **api_key** parameter to be specified.

The “Vault” feature of Ansible allows you to keep sensitive data such as passwords or keys in encrypted files, rather than as plain text in your playbooks or roles. These vault files can then be distributed or placed in source control. See *Using Vault in playbooks* for more information.

Meraki's API returns a 404 error if the API key is not correct. It does not provide any specific error saying the key is incorrect. If you receive a 404 error, check the API key first.

Returned Data Structures

Meraki and its related Ansible modules return most information in the form of a list. For example, this is returned information by `meraki_admin` querying administrators. It returns a list even though there's only one.

```
[
  {
    "orgAccess": "full",
    "name": "John Doe",
    "tags": [],
    "networks": [],
    "email": "john@doe.com",
    "id": "12345677890"
  }
]
```

Handling Returned Data

Since Meraki's response data uses lists instead of properly keyed dictionaries for responses, certain strategies should be used when querying data for particular information. For many situations, use the `selectattr()` Jinja2 function.

Merging Existing and New Data

Ansible's Meraki modules do not allow for manipulating data. For example, you may need to insert a rule in the middle of a firewall ruleset. Ansible and the Meraki modules lack a way to directly merge to manipulate data. However, a playlist can use a few tasks to split the list where you need to insert a rule and then merge them together again with the new rule added. The steps involved are as follows:

1. Create blank “front” and “back” lists.

```
vars:
  - front_rules: []
  - back_rules: []
```

2. Get existing firewall rules from Meraki and create a new variable.

```
- name: Get firewall rules
  meraki_mx_13_firewall:
    auth_key: abc123
    org_name: YourOrg
```

(下页继续)

(续上页)

```

    net_name: YourNet
    state: query
    delegate_to: localhost
    register: rules
- set_fact:
    original_ruleset: '{{rules.data}}'

```

3. Write the new rule. The new rule needs to be in a list so it can be merged with other lists in an upcoming task.

```

- set_fact:
    new_rule:
    -
      - comment: Block traffic to server
        src_cidr: 192.0.1.0/24
        src_port: any
        dst_cidr: 192.0.1.2/32
        dst_port: any
        protocol: any
        policy: deny

```

4. Split the rules into two lists. This assumes the existing ruleset is 2 rules long.

```

- set_fact:
    front_rules: '{{front_rules + [ original_ruleset[:1] ]}}'
- set_fact:
    back_rules: '{{back_rules + [ original_ruleset[1:] ]}}'

```

5. Merge rules with the new rule in the middle.

```

- set_fact:
    new_ruleset: '{{front_rules + new_rule + back_rules}}'

```

6. Upload new ruleset to Meraki.

```

- name: Set two firewall rules
  meraki_mx_l3_firewall:
    auth_key: abc123
    org_name: YourOrg
    net_name: YourNet
    state: present

```

(下页继续)

(续上页)

```
rules: '{{ new_ruleset }}'
delegate_to: localhost
```

Error Handling

Ansible's Meraki modules will often fail if improper or incompatible parameters are specified. However, there will likely be scenarios where the module accepts the information but the Meraki API rejects the data. If this happens, the error will be returned in the `body` field for HTTP status of 400 return code.

Meraki's API returns a 404 error if the API key is not correct. It does not provide any specific error saying the key is incorrect. If you receive a 404 error, check the API key first. 404 errors can also occur if improper object IDs (ex. `org_id`) are specified.

1.7.3 Infoblox Guide

Topics

- *Infoblox Guide*
 - *Prerequisites*
 - *Credentials and authenticating*
 - *NIOS lookup plugins*
 - * *Retrieving all network views*
 - * *Retrieving a host record*
 - *Use cases with modules*
 - * *Configuring an IPv4 network*
 - * *Creating a host record*
 - * *Creating a forward DNS zone*
 - * *Creating a reverse DNS zone*
 - *Dynamic inventory script*

This guide describes how to use Ansible with the Infoblox Network Identity Operating System (NIOS). With Ansible integration, you can use Ansible playbooks to automate Infoblox Core Network Services for IP address management (IPAM), DNS, and inventory tracking.

You can review simple example tasks in the documentation for any of the NIOS modules or look at the *Use cases with modules* section for more elaborate examples. See the [Infoblox](#) website for more information on

the Infoblox product.

注解: You can retrieve most of the example playbooks used in this guide from the [network-automation/infoblox__ansible](#) GitHub repository.

Prerequisites

Before using Ansible `nios` modules with Infoblox, you must install the `infoblox-client` on your Ansible control node:

```
$ sudo pip install infoblox-client
```

注解: You need an NIOS account with the WAPI feature enabled to use Ansible with Infoblox.

Credentials and authenticating

To use Infoblox `nios` modules in playbooks, you need to configure the credentials to access your Infoblox system. The examples in this guide use credentials stored in `<playbookdir>/group_vars/nios.yml`. Replace these values with your Infoblox credentials:

```
---
nios_provider:
  host: 192.0.0.2
  username: admin
  password: ansible
```

NIOS lookup plugins

Ansible includes the following lookup plugins for NIOS:

- `nios` Uses the Infoblox WAPI API to fetch NIOS specified objects, for example network views, DNS views, and host records.
- `nios_next_ip` Provides the next available IP address from a network. You'll see an example of this in *Creating a host record*.
- `nios_next_network` - Returns the next available network range for a network-container.

You must run the NIOS lookup plugins locally by specifying `connection: local`. See *lookup plugins* for more detail.

Retrieving all network views

To retrieve all network views and save them in a variable, use the `set_fact` module with the `nios` lookup plugin:

```
---
- hosts: nios
  connection: local
  tasks:
    - name: fetch all networkview objects
      set_fact:
        networkviews: "{{ lookup('nios', 'networkview', provider=nios_provider) }}"

    - name: check the networkviews
      debug:
        var: networkviews
```

Retrieving a host record

To retrieve a set of host records, use the `set_fact` module with the `nios` lookup plugin and include a filter for the specific hosts you want to retrieve:

```
---
- hosts: nios
  connection: local
  tasks:
    - name: fetch host leaf01
      set_fact:
        host: "{{ lookup('nios', 'record:host', filter={'name': 'leaf01.ansible.com'},
↪provider=nios_provider) }}"

    - name: check the leaf01 return variable
      debug:
        var: host

    - name: debug specific variable (ipv4 address)
      debug:
        var: host.ipv4addrs[0].ipv4addr

    - name: fetch host leaf02
      set_fact:
```

(下页继续)

(续上页)

```

    host: "{{ lookup('nios', 'record:host', filter={'name': 'leaf02.ansible.com'},
↪provider=nios_provider) }}"

  - name: check the leaf02 return variable
    debug:
      var: host

```

If you run this `get_host_record.yml` playbook, you should see results similar to the following:

```

$ ansible-playbook get_host_record.yml

PLAY [localhost]
↪*****

TASK [fetch host leaf01]
↪*****

ok: [localhost]

TASK [check the leaf01 return variable]
↪*****

ok: [localhost] => {
< ...output shortened...>
  "host": {
    "ipv4addrs": [
      {
        "configure_for_dhcp": false,
        "host": "leaf01.ansible.com",
      }
    ],
    "name": "leaf01.ansible.com",
    "view": "default"
  }
}

TASK [debug specific variable (ipv4 address)]
↪*****

ok: [localhost] => {
  "host.ipv4addrs[0].ipv4addr": "192.168.1.11"
}

```

(下页继续)

(续上页)

```

TASK [fetch host leaf02]
  ↳*****
ok: [localhost]

TASK [check the leaf02 return variable]
  ↳*****
ok: [localhost] => {
< ...output shortened...>
  "host": {
    "ipv4addrs": [
      {
        "configure_for_dhcp": false,
        "host": "leaf02.example.com",
        "ipv4addr": "192.168.1.12"
      }
    ],
  }
}

PLAY RECAP
  ↳*****
localhost           : ok=5    changed=0    unreachable=0    failed=0

```

The output above shows the host record for `leaf01.ansible.com` and `leaf02.ansible.com` that were retrieved by the `nios` lookup plugin. This playbook saves the information in variables which you can use in other playbooks. This allows you to use Infoblox as a single source of truth to gather and use information that changes dynamically. See [Using Variables](#) for more information on using Ansible variables. See the `nios` examples for more data options that you can retrieve.

You can access these playbooks at [Infoblox lookup playbooks](#).

Use cases with modules

You can use the `nios` modules in tasks to simplify common Infoblox workflows. Be sure to set up your *NIOS credentials* before following these examples.

Configuring an IPv4 network

To configure an IPv4 network, use the `nios_network` module:

```
---
- hosts: nios
  connection: local
  tasks:
    - name: Create a network on the default network view
      nios_network:
        network: 192.168.100.0/24
        comment: sets the IPv4 network
        options:
          - name: domain-name
            value: ansible.com
        state: present
        provider: "{{nios_provider}}"
```

Notice the last parameter, `provider`, uses the variable `nios_provider` defined in the `group_vars/` directory.

Creating a host record

To create a host record named *leaf03.ansible.com* on the newly-created IPv4 network:

```
---
- hosts: nios
  connection: local
  tasks:
    - name: configure an IPv4 host record
      nios_host_record:
        name: leaf03.ansible.com
        ipv4addrs:
          - ipv4addr:
              "{{ lookup('nios_next_ip', '192.168.100.0/24', provider=nios_provider)[0] }}"
        state: present
      provider: "{{nios_provider}}"
```

Notice the IPv4 address in this example uses the `nios_next_ip` lookup plugin to find the next available IPv4 address on the network.

Creating a forward DNS zone

To configure a forward DNS zone use, the `nios_zone` module:

```

---
- hosts: nios
  connection: local
  tasks:
    - name: Create a forward DNS zone called ansible-test.com
      nios_zone:
        name: ansible-test.com
        comment: local DNS zone
        state: present
        provider: "{{ nios_provider }}"

```

Creating a reverse DNS zone

To configure a reverse DNS zone:

```

---
- hosts: nios
  connection: local
  tasks:
    - name: configure a reverse mapping zone on the system using IPV6 zone format
      nios_zone:
        name: 100::1/128
        zone_format: IPV6
        state: present
        provider: "{{ nios_provider }}"

```

Dynamic inventory script

You can use the Infoblox dynamic inventory script to import your network node inventory with Infoblox NIOS. To gather the inventory from Infoblox, you need two files:

- `infoblox.yaml` - A file that specifies the NIOS provider arguments and optional filters.
- `infoblox.py` - The python script that retrieves the NIOS inventory.

To use the Infoblox dynamic inventory script:

1. Download the `infoblox.yaml` file and save it in the `/etc/ansible` directory.
2. Modify the `infoblox.yaml` file with your NIOS credentials.
3. Download the `infoblox.py` file and save it in the `/etc/ansible/hosts` directory.
4. Change the permissions on the `infoblox.py` file to make the file an executable:

```
$ sudo chmod +x /etc/ansible/hosts/infoblox.py
```

You can optionally use `./infoblox.py --list` to test the script. After a few minutes, you should see your Infoblox inventory in JSON format. You can explicitly use the Infoblox dynamic inventory script as follows:

```
$ ansible -i infoblox.py all -m ping
```

You can also implicitly use the Infoblox dynamic inventory script by including it in your inventory directory (`etc/ansible/hosts` by default). See [动态 Inventory 清单配置](#) for more details.

参见:

Infoblox website The Infoblox website

Infoblox and Ansible Deployment Guide The deployment guide for Ansible integration provided by Infoblox.

Infoblox Integration in Ansible 2.5 Ansible blog post about Infoblox.

Ansible NIOS modules The list of supported NIOS modules, with examples.

Infoblox Ansible Examples Infoblox example playbooks.

To learn more about Network Automation with Ansible, see [Network Automation Getting Started](#) and [Network Automation Advanced Topics](#).

1.8 Virtualization and Containerization Guides

The guides in this section cover integrating Ansible with popular tools for creating virtual machines and containers. They explore particular use cases in greater depth and provide a more “top-down” explanation of some basic features.

1.8.1 Docker Guide

Ansible offers the following modules for orchestrating Docker containers:

docker_compose Use your existing Docker compose files to orchestrate containers on a single Docker daemon or on Swarm. Supports compose versions 1 and 2.

docker_container Manages the container lifecycle by providing the ability to create, update, stop, start and destroy a container.

docker_image Provides full control over images, including: build, pull, push, tag and remove.

docker_image_info Inspects one or more images in the Docker host’s image cache, providing the information for making decision or assertions in a playbook.

docker_login Authenticates with Docker Hub or any Docker registry and updates the Docker Engine config file, which in turn provides password-free pushing and pulling of images to and from the registry.

docker (dynamic inventory) Dynamically builds an inventory of all the available containers from a set of one or more Docker hosts.

Ansible 2.1.0 includes major updates to the Docker modules, marking the start of a project to create a complete and integrated set of tools for orchestrating containers. In addition to the above modules, we are also working on the following:

Still using Dockerfile to build images? Check out [ansible-bender](#), and start building images from your Ansible playbooks.

Use [Ansible Operator](#) to launch your docker-compose file on [OpenShift](#). Go from an app on your laptop to a fully scalable app in the cloud with Kubernetes in just a few moments.

There's more planned. See the latest ideas and thinking at the [Ansible proposal repo](#).

Requirements

Using the docker modules requires having the [Docker SDK for Python](#) installed on the host running Ansible. You will need to have `>= 1.7.0` installed. For Python 2.7 or Python 3, you can install it as follows:

```
$ pip install docker
```

For Python 2.6, you need a version before 2.0. For these versions, the SDK was called `docker-py`, so you need to install it as follows:

```
$ pip install 'docker-py>=1.7.0'
```

Please note that only one of `docker` and `docker-py` must be installed. Installing both will result in a broken installation. If this happens, Ansible will detect it and inform you about it:

```
Cannot have both the docker-py and docker python modules installed together as they use
↳ the same
namespace and cause a corrupt installation. Please uninstall both packages, and re-
↳ install only
the docker-py or docker python module. It is recommended to install the docker module if
↳ no support
for Python 2.6 is required. Please note that simply uninstalling one of the modules can
↳ leave the
other module in a broken state.
```

The `docker_compose` module also requires [docker-compose](#)

```
$ pip install 'docker-compose>=1.7.0'
```

Connecting to the Docker API

You can connect to a local or remote API using parameters passed to each task or by setting environment variables. The order of precedence is command line parameters and then environment variables. If neither a command line option or an environment variable is found, a default value will be used. The default values are provided under *Parameters*

Parameters

Control how modules connect to the Docker API by passing the following parameters:

docker_host The URL or Unix socket path used to connect to the Docker API. Defaults to `unix:///var/run/docker.sock`. To connect to a remote host, provide the TCP connection string. For example: `tcp://192.0.2.23:2376`. If TLS is used to encrypt the connection to the API, then the module will automatically replace ‘tcp’ in the connection URL with ‘https’.

api_version The version of the Docker API running on the Docker Host. Defaults to the latest version of the API supported by docker-py.

timeout The maximum amount of time in seconds to wait on a response from the API. Defaults to 60 seconds.

tls Secure the connection to the API by using TLS without verifying the authenticity of the Docker host server. Defaults to False.

tls_verify Secure the connection to the API by using TLS and verifying the authenticity of the Docker host server. Default is False.

cacert_path Use a CA certificate when performing server verification by providing the path to a CA certificate file.

cert_path Path to the client’s TLS certificate file.

key_path Path to the client’s TLS key file.

tls_hostname When verifying the authenticity of the Docker Host server, provide the expected name of the server. Defaults to ‘localhost’.

ssl_version Provide a valid SSL version number. Default value determined by docker-py, which at the time of this writing was 1.0

Environment Variables

Control how the modules connect to the Docker API by setting the following variables in the environment of the host running Ansible:

DOCKER_HOST The URL or Unix socket path used to connect to the Docker API.

DOCKER_API_VERSION The version of the Docker API running on the Docker Host.
Defaults to the latest version of the API supported by docker-py.

DOCKER_TIMEOUT The maximum amount of time in seconds to wait on a response from the API.

DOCKER_CERT_PATH Path to the directory containing the client certificate, client key and CA certificate.

DOCKER_SSL_VERSION Provide a valid SSL version number.

DOCKER_TLS Secure the connection to the API by using TLS without verifying the authenticity of the Docker Host.

DOCKER_TLS_VERIFY Secure the connection to the API by using TLS and verify the authenticity of the Docker Host.

Dynamic Inventory Script

The inventory script generates dynamic inventory by making API requests to one or more Docker APIs. It's dynamic because the inventory is generated at run-time rather than being read from a static file. The script generates the inventory by connecting to one or many Docker APIs and inspecting the containers it finds at each API. Which APIs the script contacts can be defined using environment variables or a configuration file.

Groups

The script will create the following host groups:

- container id
- container name
- container short id
- image_name (image_<image name>)
- docker_host
- running
- stopped

Examples

You can run the script interactively from the command line or pass it as the inventory to a playbook. Here are few examples to get you started:

```
# Connect to the Docker API on localhost port 4243 and format the JSON output
DOCKER_HOST=tcp://localhost:4243 ./docker.py --pretty

# Any container's ssh port exposed on 0.0.0.0 will be mapped to
# another IP address (where Ansible will attempt to connect via SSH)
DOCKER_DEFAULT_IP=192.0.2.5 ./docker.py --pretty

# Run as input to a playbook:
ansible-playbook -i ~/projects/ansible/contrib/inventory/docker.py docker_inventory_test.
↪.yml

# Simple playbook to invoke with the above example:

- name: Test docker_inventory, this will not connect to any hosts
  hosts: all
  gather_facts: no
  tasks:
    - debug: msg="Container - {{ inventory_hostname }}"
```

Configuration

You can control the behavior of the inventory script by defining environment variables, or creating a docker.yml file (sample provided in ansible/contrib/inventory). The order of precedence is the docker.yml file and then environment variables.

Environment Variables

To connect to a single Docker API the following variables can be defined in the environment to control the connection options. These are the same environment variables used by the Docker modules.

DOCKER_HOST The URL or Unix socket path used to connect to the Docker API. Defaults to unix://var/run/docker.sock.

DOCKER_API_VERSION: The version of the Docker API running on the Docker Host. Defaults to the latest version of the API supported by docker-py.

DOCKER_TIMEOUT: The maximum amount of time in seconds to wait on a response from the API. Defaults to 60 seconds.

DOCKER_TLS: Secure the connection to the API by using TLS without verifying the authenticity of the Docker host server. Defaults to False.

DOCKER_TLS_VERIFY: Secure the connection to the API by using TLS and verifying the authenticity of the Docker host server. Default is False

DOCKER_TLS_HOSTNAME: When verifying the authenticity of the Docker Host server, provide the expected name of the server. Defaults to localhost.

DOCKER_CERT_PATH: Path to the directory containing the client certificate, client key and CA certificate.

DOCKER_SSL_VERSION: Provide a valid SSL version number. Default value determined by docker-py, which at the time of this writing was 1.0

In addition to the connection variables there are a couple variables used to control the execution and output of the script:

DOCKER_CONFIG_FILE Path to the configuration file. Defaults to ./docker.yml.

DOCKER_PRIVATE_SSH_PORT: The private port (container port) on which SSH is listening for connections. Defaults to 22.

DOCKER_DEFAULT_IP: The IP address to assign to ansible_host when the container's SSH port is mapped to interface '0.0.0.0'.

Configuration File

Using a configuration file provides a means for defining a set of Docker APIs from which to build an inventory.

The default name of the file is derived from the name of the inventory script. By default the script will look for basename of the script (i.e. docker) with an extension of '.yml'.

You can also override the default name of the script by defining DOCKER_CONFIG_FILE in the environment.

Here's what you can define in docker_inventory.yml:

defaults Defines a default connection. Defaults will be taken from this and applied to any values not provided for a host defined in the hosts list.

hosts If you wish to get inventory from more than one Docker host, define a hosts list.

For the default host and each host in the hosts list define the following attributes:

```
host:
  description: The URL or Unix socket path used to connect to the Docker API.
  required: yes
```

(下页继续)

(续上页)

```
tls:
    description: Connect using TLS without verifying the authenticity of the Docker host↵
↵server.
    default: false
    required: false

tls_verify:
    description: Connect using TLS without verifying the authenticity of the Docker host↵
↵server.
    default: false
    required: false

cert_path:
    description: Path to the client's TLS certificate file.
    default: null
    required: false

cacert_path:
    description: Use a CA certificate when performing server verification by providing↵
↵the path to a CA certificate file.
    default: null
    required: false

key_path:
    description: Path to the client's TLS key file.
    default: null
    required: false

version:
    description: The Docker API version.
    required: false
    default: will be supplied by the docker-py module.

timeout:
    description: The amount of time in seconds to wait on an API response.
    required: false
    default: 60

default_ip:
    description: The IP address to assign to ansible_host when the container's SSH port↵
↵is mapped to interface
```

(下页继续)

(续上页)

```
'0.0.0.0'.
required: false
default: 127.0.0.1

private_ssh_port:
  description: The port containers use for SSH
  required: false
  default: 22
```

1.8.2 Kubernetes and OpenShift Guide

Modules for interacting with the Kubernetes (K8s) and OpenShift API are under development, and can be used in preview mode. To use them, review the requirements, and then follow the installation and use instructions.

Requirements

To use the modules, you'll need the following:

- Run Ansible from source. For assistance, view [从源码运行 Ansible \(devel\)](#).
- [OpenShift Rest Client](#) installed on the host that will execute the modules

Installation and use

The individual modules, as of this writing, are not part of the Ansible repository, but they can be accessed by installing the role, [ansible.kubernetes-modules](#), and including it in a playbook.

To install, run the following:

```
$ ansible-galaxy install ansible.kubernetes-modules
```

Next, include it in a playbook, as follows:

```
---
- hosts: localhost
  remote_user: root
  roles:
    - role: ansible.kubernetes-modules
    - role: hello-world
```

Because the role is referenced, `hello-world` is able to access the modules, and use them to deploy an application.

The modules are found in the `library` folder of the role. Each includes full documentation for parameters and the returned data structure. However, not all modules include examples, only those where `testing data` has been created.

Authenticating with the API

By default the OpenShift Rest Client will look for `~/.kube/config`, and if found, connect using the active context. You can override the location of the file using the `“kubeconfig”` parameter, and the context, using the `context` parameter.

Basic authentication is also supported using the `username` and `password` options. You can override the URL using the `host` parameter. Certificate authentication works through the `ssl_ca_cert`, `cert_file`, and `key_file` parameters, and for token authentication, use the `api_key` parameter.

To disable SSL certificate verification, set `verify_ssl` to false.

Filing issues

If you find a bug or have a suggestion regarding individual modules or the role, please file issues at [OpenShift Rest Client issues](#).

There is also a utility module, `k8s_common.py`, that is part of the [Ansible](#) repo. If you find a bug or have suggestions regarding it, please file issues at [Ansible issues](#).

1.8.3 Vagrant Guide

Introduction

[Vagrant](#) is a tool to manage virtual machine environments, and allows you to configure and use reproducible work environments on top of various virtualization and cloud platforms. It also has integration with Ansible as a provisioner for these virtual machines, and the two tools work together well.

This guide will describe how to use Vagrant 1.7+ and Ansible together.

If you're not familiar with Vagrant, you should visit [the documentation](#).

This guide assumes that you already have Ansible installed and working. Running from a Git checkout is fine. Follow the [安装 Ansible](#) guide for more information.

Vagrant Setup

The first step once you've installed Vagrant is to create a **Vagrantfile** and customize it to suit your needs. This is covered in detail in the Vagrant documentation, but here is a quick example that includes a section to use the Ansible provisioner to manage a single machine:

```
# This guide is optimized for Vagrant 1.8 and above.
# Older versions of Vagrant put less info in the inventory they generate.
Vagrant.require_version ">= 1.8.0"

Vagrant.configure(2) do |config|

  config.vm.box = "ubuntu/bionic64"

  config.vm.provision "ansible" do |ansible|
    ansible.verbose = "v"
    ansible.playbook = "playbook.yml"
  end
end
```

Notice the `config.vm.provision` section that refers to an Ansible playbook called `playbook.yml` in the same directory as the **Vagrantfile**. Vagrant runs the provisioner once the virtual machine has booted and is ready for SSH access.

There are a lot of Ansible options you can configure in your **Vagrantfile**. Visit the [Ansible Provisioner documentation](#) for more information.

```
$ vagrant up
```

This will start the VM, and run the provisioning playbook (on the first VM startup).

To re-run a playbook on an existing VM, just run:

```
$ vagrant provision
```

This will re-run the playbook against the existing VM.

Note that having the `ansible.verbose` option enabled will instruct Vagrant to show the full `ansible-playbook` command used behind the scene, as illustrated by this example:

```
$ PYTHONUNBUFFERED=1 ANSIBLE_FORCE_COLOR=true ANSIBLE_HOST_KEY_CHECKING=false ANSIBLE_
↪SSH_ARGS='-o UserKnownHostsFile=/dev/null -o IdentitiesOnly=yes -o ControlMaster=auto -
↪o ControlPersist=60s' ansible-playbook --connection=ssh --timeout=30 --limit="default"
↪--inventory-file=/home/someone/coding-in-a-project/.vagrant/provisioners/ansible/
↪inventory -v playbook.yml
```

This information can be quite useful to debug integration issues and can also be used to manually execute Ansible from a shell, as explained in the next section.

Running Ansible Manually

Sometimes you may want to run Ansible manually against the machines. This is faster than kicking `vagrant provision` and pretty easy to do.

With our `Vagrantfile` example, Vagrant automatically creates an Ansible inventory file in `.vagrant/provisioners/ansible/inventory/vagrant_ansible_inventory`. This inventory is configured according to the SSH tunnel that Vagrant automatically creates. A typical automatically-created inventory file for a single machine environment may look something like this:

```
# Generated by Vagrant

default ansible_host=127.0.0.1 ansible_port=2222 ansible_user='vagrant' ansible_ssh_
↪private_key_file='/home/someone/coding-in-a-project/.vagrant/machines/default/
↪virtualbox/private_key'
```

If you want to run Ansible manually, you will want to make sure to pass `ansible` or `ansible-playbook` commands the correct arguments, at least for the *inventory*.

```
$ ansible-playbook -i .vagrant/provisioners/ansible/inventory/vagrant_ansible_inventory
↪playbook.yml
```

Advanced Usages

The “Tips and Tricks” chapter of the [Ansible Provisioner documentation](#) provides detailed information about more advanced Ansible features like:

- how to execute a playbook in parallel within a multi-machine environment
- how to integrate a local `ansible.cfg` configuration file

参见:

[Vagrant Home](#) The Vagrant homepage with downloads

[Vagrant Documentation](#) Vagrant Documentation

[Ansible Provisioner](#) The Vagrant documentation for the Ansible provisioner

[Vagrant Issue Tracker](#) The open issues for the Ansible provisioner in the Vagrant project

[Working With Playbooks](#) An introduction to playbooks

1.8.4 VMware Guide

Welcome to the Ansible for VMware Guide!

The purpose of this guide is to teach you everything you need to know about using Ansible with VMware.

To get started, please select one of the following topics.

Introduction to Ansible for VMware

Topics

- *Introduction to Ansible for VMware*
 - *Introduction*
 - *Requirements*
 - *vmware__guest module*

Introduction

Ansible provides various modules to manage VMware infrastructure, which includes datacenter, cluster, host system and virtual machine.

Requirements

Ansible VMware modules are written on top of [pyVmomi](#). [pyVmomi](#) is the Python SDK for the VMware vSphere API that allows user to manage ESX, ESXi, and vCenter infrastructure. You can install [pyVmomi](#) using `pip`:

```
$ pip install pyvmomi
```

Ansible VMware modules leveraging latest vSphere(6.0+) features are using [vSphere Automation Python SDK](#). The vSphere Automation Python SDK also has client libraries, documentation, and sample code for VMware Cloud on AWS Console APIs, NSX VMware Cloud on AWS integration APIs, VMware Cloud on AWS site recovery APIs, NSX-T APIs.

You can install vSphere Automation Python SDK using `pip`:

```
$ pip install --upgrade git+https://github.com/vmware/vsphere-automation-sdk-python.git
```

Note: Installing vSphere Automation Python SDK also installs `pyvmomi`. A separate installation of `pyvmomi` is not required.

vmware_guest module

The `vmware_guest` module manages various operations related to virtual machines in the given ESXi or vCenter server.

参见:

pyVmomi The GitHub Page of pyVmomi

pyVmomi Issue Tracker The issue tracker for the pyVmomi project

govc govc is a vSphere CLI built on top of govmomi

Working With Playbooks An introduction to playbooks

Ansible for VMware Concepts

Some of these concepts are common to all uses of Ansible, including VMware automation; some are specific to VMware. You need to understand them to use Ansible for VMware automation. This introduction provides the background you need to follow the *scenarios* in this guide.

- *Control Node*
- *Delegation*
- *Modules*
- *Playbooks*
- *pyVmomi*

Control Node

Any machine with Ansible installed. You can run commands and playbooks, invoking `/usr/bin/ansible` or `/usr/bin/ansible-playbook`, from any control node. You can use any computer that has Python installed on it as a control node - laptops, shared desktops, and servers can all run Ansible. However, you cannot use a Windows machine as a control node. You can have multiple control nodes.

Delegation

Delegation allows you to select the system that executes a given task. If you do not have `pyVmomi` installed on your control node, use the `delegate_to` keyword on VMware-specific tasks to execute them on any host where you have `pyVmomi` installed.

Modules

The units of code Ansible executes. Each module has a particular use, from creating virtual machines on vCenter to managing distributed virtual switches in the vCenter environment. You can invoke a single module with a task, or invoke several different modules in a playbook. For an idea of how many modules Ansible includes, take a look at the list of cloud modules, which includes VMware modules.

Playbooks

Ordered lists of tasks, saved so you can run those tasks in that order repeatedly. Playbooks can include variables as well as tasks. Playbooks are written in YAML and are easy to read, write, share and understand.

pyVmomi

Ansible VMware modules are written on top of [pyVmomi](#). [pyVmomi](#) is the official Python SDK for the VMware vSphere API that allows user to manage ESX, ESXi, and vCenter infrastructure.

You need to install this Python SDK on host from where you want to invoke VMware automation. For example, if you are using control node then [pyVmomi](#) must be installed on control node.

If you are using any `delegate_to` host which is different from your control node then you need to install [pyVmomi](#) on that `delegate_to` node.

You can install [pyVmomi](#) using pip:

```
$ pip install pyvmomi
```

VMware Prerequisites

- *Installing SSL Certificates*
 - *Installing vCenter SSL certificates for Ansible*
 - *Installing ESXi SSL certificates for Ansible*

Installing SSL Certificates

All vCenter and ESXi servers require SSL encryption on all connections to enforce secure communication. You must enable SSL encryption for Ansible by installing the server's SSL certificates on your Ansible control node or delegate node.

If the SSL certificate of your vCenter or ESXi server is not correctly installed on your Ansible control node, you will see the following warning when using Ansible VMware modules:

```
Unable to connect to vCenter or ESXi API at xx.xx.xx.xx on TCP/443: [SSL:
CERTIFICATE_VERIFY_FAILED] certificate verify failed (_ssl.c:777)
```

To install the SSL certificate for your VMware server, and run your Ansible VMware modules in encrypted mode, please follow the instructions for the server you are running with VMware.

Installing vCenter SSL certificates for Ansible

- From any web browser, go to the base URL of the vCenter Server without port number like `https://vcenter-domain.example.com`
- Click the “Download trusted root CA certificates” link at the bottom of the grey box on the right and download the file.
- Change the extension of the file to `.zip`. The file is a ZIP file of all root certificates and all CRLs.
- Extract the contents of the zip file. The extracted directory contains a `.certs` directory that contains two types of files. Files with a number as the extension (`.0`, `.1`, and so on) are root certificates.
- Install the certificate files are trusted certificates by the process that is appropriate for your operating system.

Installing ESXi SSL certificates for Ansible

- Enable SSH Service on ESXi either by using Ansible VMware module `vmware_host_service_manager` or manually using vSphere Web interface.
- SSH to ESXi server using administrative credentials, and navigate to directory `/etc/vmware/ssl`
- Secure copy (SCP) `rui.crt` located in `/etc/vmware/ssl` directory to Ansible control node.
- Install the certificate file by the process that is appropriate for your operating system.

Using VMware dynamic inventory plugin

Topics

- *Using VMware dynamic inventory plugin*
 - *VMware Dynamic Inventory Plugin*
 - * *Requirements*
 - *Using vaulted configuration files*

VMware Dynamic Inventory Plugin

The best way to interact with your hosts is to use the VMware dynamic inventory plugin, which dynamically queries VMware APIs and tells Ansible what nodes can be managed.

Requirements

To use the VMware dynamic inventory plugins, you must install `pyVmomi` on your control node (the host running Ansible).

To include tag-related information for the virtual machines in your dynamic inventory, you also need the `vSphere Automation SDK`, which supports REST API features like tagging and content libraries, on your control node. You can install the `vSphere Automation SDK` following [these instructions](#).

```
$ pip install pyvmomi
```

To use this VMware dynamic inventory plugin, you need to enable it first by specifying the following in the `ansible.cfg` file:

```
[inventory]
enable_plugins = vmware_vm_inventory
```

Then, create a file that ends in `.vmware.yml` or `.vmware.yaml` in your working directory.

The `vmware_vm_inventory` script takes in the same authentication information as any VMware module.

Here's an example of a valid inventory file:

```
plugin: vmware_vm_inventory
strict: False
hostname: 10.65.223.31
username: administrator@vsphere.local
password: Esxi@123$%
validate_certs: False
with_tags: True
```

Executing `ansible-inventory --list -i <filename>.vmware.yml` will create a list of VMware instances that are ready to be configured using Ansible.

Using vaulted configuration files

Since the inventory configuration file contains vCenter password in plain text, a security risk, you may want to encrypt your entire inventory configuration file.

You can encrypt a valid inventory configuration file as follows:

```
$ ansible-vault encrypt <filename>.vmware.yml
New Vault password:
Confirm New Vault password:
Encryption successful
```

And you can use this vaulted inventory configuration file using:

```
$ ansible-inventory -i filename.vmware.yml --list --vault-password-file=/path/to/vault_
↪password_file
```

参见:

[pyVmomi](#) The GitHub Page of pyVmomi

[pyVmomi Issue Tracker](#) The issue tracker for the pyVmomi project

[vSphere Automation SDK GitHub Page](#) The GitHub Page of vSphere Automation SDK for Python

[vSphere Automation SDK Issue Tracker](#) The issue tracker for vSphere Automation SDK for Python

Working With Playbooks An introduction to playbooks

Using Vault in playbooks Using Vault in playbooks

Ansible for VMware Scenarios

These scenarios teach you how to accomplish common VMware tasks using Ansible. To get started, please select the task you want to accomplish.

Deploy a virtual machine from a template

Topics

- *Deploy a virtual machine from a template*
 - *Introduction*
 - *Scenario Requirements*
 - *Assumptions*
 - *Caveats*
 - *Example Description*
 - * *What to expect*
 - * *Troubleshooting*

Introduction

This guide will show you how to utilize Ansible to clone a virtual machine from already existing VMware template or existing VMware guest.

Scenario Requirements

- Software
 - Ansible 2.5 or later must be installed
 - The Python module `Pyvmomi` must be installed on the Ansible (or Target host if not executing against localhost)
 - Installing the latest `Pyvmomi` via `pip` is recommended [as the OS provided packages are usually out of date and incompatible]
- Hardware
 - vCenter Server with at least one ESXi server
- Access / Credentials
 - Ansible (or the target server) must have network access to the either vCenter server or the ESXi server you will be deploying to
 - Username and Password
 - Administrator user with following privileges
 - * `Datastore.AllocateSpace` on the destination datastore or datastore folder
 - * `Network.Assign` on the network to which the virtual machine will be assigned
 - * `Resource.AssignVMToPool` on the destination host, cluster, or resource pool
 - * `VirtualMachine.Config.AddNewDisk` on the datacenter or virtual machine folder
 - * `VirtualMachine.Config.AddRemoveDevice` on the datacenter or virtual machine folder
 - * `VirtualMachine.Interact.PowerOn` on the datacenter or virtual machine folder
 - * `VirtualMachine.Inventory.CreateFromExisting` on the datacenter or virtual machine folder
 - * `VirtualMachine.Provisioning.Clone` on the virtual machine you are cloning
 - * `VirtualMachine.Provisioning.Customize` on the virtual machine or virtual machine folder if you are customizing the guest operating system
 - * `VirtualMachine.Provisioning.DeployTemplate` on the template you are using

- * `VirtualMachine.Provisioning.ReadCustSpecs` on the root vCenter Server if you are customizing the guest operating system

Depending on your requirements, you could also need one or more of the following privileges:

- * `VirtualMachine.Config.CPUCount` on the datacenter or virtual machine folder
- * `VirtualMachine.Config.Memory` on the datacenter or virtual machine folder
- * `VirtualMachine.Config.DiskExtend` on the datacenter or virtual machine folder
- * `VirtualMachine.Config.Annotation` on the datacenter or virtual machine folder
- * `VirtualMachine.Config.AdvancedConfig` on the datacenter or virtual machine folder
- * `VirtualMachine.Config.EditDevice` on the datacenter or virtual machine folder
- * `VirtualMachine.Config.Resource` on the datacenter or virtual machine folder
- * `VirtualMachine.Config.Settings` on the datacenter or virtual machine folder
- * `VirtualMachine.Config.UpgradeVirtualHardware` on the datacenter or virtual machine folder
- * `VirtualMachine.Interact.SetCDMedia` on the datacenter or virtual machine folder
- * `VirtualMachine.Interact.SetFloppyMedia` on the datacenter or virtual machine folder
- * `VirtualMachine.Interact.DeviceConnection` on the datacenter or virtual machine folder

Assumptions

- All variable names and VMware object names are case sensitive
- VMware allows creation of virtual machine and templates with same name across datacenters and within datacenters
- You need to use Python 2.7.9 version in order to use `validate_certs` option, as this version is capable of changing the SSL verification behaviours

Caveats

- Hosts in the ESXi cluster must have access to the datastore that the template resides on.
- Multiple templates with the same name will cause module failures.

- In order to utilize Guest Customization, VMware Tools must be installed on the template. For Linux, the `open-vm-tools` package is recommended, and it requires that `Perl` be installed.

Example Description

In this use case / example, we will be selecting a virtual machine template and cloning it into a specific folder in our Datacenter / Cluster. The following Ansible playbook showcases the basic parameters that are needed for this.

```
---
- name: Create a VM from a template
  hosts: localhost
  gather_facts: no
  tasks:
  - name: Clone the template
    vmware_guest:
      hostname: "{{ vcenter_ip }}"
      username: "{{ vcenter_username }}"
      password: "{{ vcenter_password }}"
      validate_certs: False
      name: testvm_2
      template: template_e17
      datacenter: "{{ datacenter_name }}"
      folder: /DC1/vm
      state: poweredon
      cluster: "{{ cluster_name }}"
      wait_for_ip_address: yes
```

Since Ansible utilizes the VMware API to perform actions, in this use case we will be connecting directly to the API from our localhost. This means that our playbooks will not be running from the vCenter or ESXi Server. We do not necessarily need to collect facts about our localhost, so the `gather_facts` parameter will be disabled. You can run these modules against another server that would then connect to the API if your localhost does not have access to vCenter. If so, the required Python modules will need to be installed on that target server.

To begin, there are a few bits of information we will need. First and foremost is the hostname of the ESXi server or vCenter server. After this, you will need the username and password for this server. For now, you will be entering these directly, but in a more advanced playbook this can be abstracted out and stored in a more secure fashion using `ansible-vault` or using [Ansible Tower credentials](#). If your vCenter or ESXi server is not setup with proper CA certificates that can be verified from the Ansible server, then it is necessary to disable validation of these certificates by using the `validate_certs` parameter. To do this you need to set `validate_certs=False` in your playbook.

Now you need to supply the information about the virtual machine which will be created. Give your virtual machine a name, one that conforms to all VMware requirements for naming conventions. Next, select the display name of the template from which you want to clone new virtual machine. This must match what's displayed in VMware Web UI exactly. Then you can specify a folder to place this new virtual machine in. This path can either be a relative path or a full path to the folder including the Datacenter. You may need to specify a state for the virtual machine. This simply tells the module which action you want to take, in this case you will ensure that the virtual machine exists and is powered on. An optional parameter is `wait_for_ip_address`, this will tell Ansible to wait for the virtual machine to fully boot up and VMware Tools is running before completing this task.

What to expect

- You will see a bit of JSON output after this playbook completes. This output shows various parameters that are returned from the module and from vCenter about the newly created VM.

```
{
  "changed": true,
  "instance": {
    "annotation": "",
    "current_snapshot": null,
    "customvalues": {},
    "guest_consolidation_needed": false,
    "guest_question": null,
    "guest_tools_status": "guestToolsNotRunning",
    "guest_tools_version": "0",
    "hw_cores_per_socket": 1,
    "hw_datastores": [
      "ds_215"
    ],
    "hw_esxi_host": "192.0.2.44",
    "hw_eth0": {
      "addresstype": "assigned",
      "ipaddresses": null,
      "label": "Network adapter 1",
      "macaddress": "00:50:56:8c:19:f4",
      "macaddress_dash": "00-50-56-8c-19-f4",
      "portgroup_key": "dvportgroup-17",
      "portgroup_portkey": "0",
      "summary": "DVSwitch: 50 0c 5b 22 b6 68 ab 89-fc 0b 59 a4 08 6e 80 fa"
    },
    "hw_files": [
```

(下页继续)

(续上页)

```

        "[ds_215] testvm_2/testvm_2.vmx",
        "[ds_215] testvm_2/testvm_2.vmsd",
        "[ds_215] testvm_2/testvm_2.vmdk"
    ],
    "hw_folder": "/DC1/vm",
    "hw_guest_full_name": null,
    "hw_guest_ha_state": null,
    "hw_guest_id": null,
    "hw_interfaces": [
        "eth0"
    ],
    "hw_is_template": false,
    "hw_memtotal_mb": 512,
    "hw_name": "testvm_2",
    "hw_power_status": "poweredOff",
    "hw_processor_count": 2,
    "hw_product_uuid": "420cb25b-81e8-8d3b-dd2d-a439ee54fcc5",
    "hw_version": "vmx-13",
    "instance_uuid": "500cd53b-ed57-d74e-2da8-0dc0eddf54d5",
    "ipv4": null,
    "ipv6": null,
    "module_hw": true,
    "snapshots": []
},
"invocation": {
    "module_args": {
        "annotation": null,
        "cdrom": {},
        "cluster": "DC1_C1",
        "customization": {},
        "customization_spec": null,
        "customvalues": [],
        "datacenter": "DC1",
        "disk": [],
        "esxi_hostname": null,
        "folder": "/DC1/vm",
        "force": false,
        "guest_id": null,
        "hardware": {},
        "hostname": "192.0.2.44",

```

(下页继续)

```
    "is_template": false,
    "linked_clone": false,
    "name": "testvm_2",
    "name_match": "first",
    "networks": [],
    "password": "VALUE_SPECIFIED_IN_NO_LOG_PARAMETER",
    "port": 443,
    "resource_pool": null,
    "snapshot_src": null,
    "state": "present",
    "state_change_timeout": 0,
    "template": "template_e17",
    "username": "administrator@vsphere.local",
    "uuid": null,
    "validate_certs": false,
    "vapp_properties": [],
    "wait_for_ip_address": true
  }
}
```

- State is changed to **True** which notifies that the virtual machine is built using given template. The module will not complete until the clone task in VMware is finished. This can take some time depending on your environment.
- If you utilize the **wait_for_ip_address** parameter, then it will also increase the clone time as it will wait until virtual machine boots into the OS and an IP Address has been assigned to the given NIC.

Troubleshooting

Things to inspect

- Check if the values provided for username and password are correct
- Check if the datacenter you provided is available
- Check if the template specified exists and you have permissions to access the datastore
- Ensure the full folder path you specified already exists. It will not create folders automatically for you

Rename an existing virtual machine

Topics

- *Rename an existing virtual machine*
 - *Introduction*
 - *Scenario Requirements*
 - *Caveats*
 - *Example Description*
 - * *What to expect*
 - * *Troubleshooting*

Introduction

This guide will show you how to utilize Ansible to rename an existing virtual machine.

Scenario Requirements

- Software
 - Ansible 2.5 or later must be installed.
 - The Python module `Pyvmomi` must be installed on the Ansible control node (or Target host if not executing against localhost).
 - We recommend installing the latest version with pip: `pip install Pyvmomi` (as the OS packages are usually out of date and incompatible).
- Hardware
 - At least one standalone ESXi server or
 - vCenter Server with at least one ESXi server
- Access / Credentials
 - Ansible (or the target server) must have network access to the either vCenter server or the ESXi server
 - Username and Password for vCenter or ESXi server
 - Hosts in the ESXi cluster must have access to the datastore that the template resides on.

Caveats

- All variable names and VMware object names are case sensitive.

- You need to use Python 2.7.9 version in order to use `validate_certs` option, as this version is capable of changing the SSL verification behaviours.

Example Description

With the following Ansible playbook you can rename an existing virtual machine by changing the UUID.

```
---
- name: Rename virtual machine from old name to new name using UUID
  gather_facts: no
  vars_files:
    - vcenter_vars.yml
  vars:
    ansible_python_interpreter: "/usr/bin/env python3"
  hosts: localhost
  tasks:
    - set_fact:
        vm_name: "old_vm_name"
        new_vm_name: "new_vm_name"
        datacenter: "DC1"
        cluster_name: "DC1_C1"

    - name: Get VM "{{ vm_name }}" uuid
      vmware_guest_facts:
        hostname: "{{ vcenter_server }}"
        username: "{{ vcenter_user }}"
        password: "{{ vcenter_pass }}"
        validate_certs: False
        datacenter: "{{ datacenter }}"
        folder: "{{ datacenter }}/vm"
        name: "{{ vm_name }}"
      register: vm_facts

    - name: Rename "{{ vm_name }}" to "{{ new_vm_name }}"
      vmware_guest:
        hostname: "{{ vcenter_server }}"
        username: "{{ vcenter_user }}"
        password: "{{ vcenter_pass }}"
        validate_certs: False
        cluster: "{{ cluster_name }}"
        uuid: "{{ vm_facts.instance.hw_product_uuid }}"
```

(下页继续)

(续上页)

```
name: "{{ new_vm_name }}"
```

Since Ansible utilizes the VMware API to perform actions, in this use case it will be connecting directly to the API from localhost.

This means that playbooks will not be running from the vCenter or ESXi Server.

Note that this play disables the `gather_facts` parameter, since you don't want to collect facts about localhost.

You can run these modules against another server that would then connect to the API if localhost does not have access to vCenter. If so, the required Python modules will need to be installed on that target server. We recommend installing the latest version with pip: `pip install Pyvmomi` (as the OS packages are usually out of date and incompatible).

Before you begin, make sure you have:

- Hostname of the ESXi server or vCenter server
- Username and password for the ESXi or vCenter server
- The UUID of the existing Virtual Machine you want to rename

For now, you will be entering these directly, but in a more advanced playbook this can be abstracted out and stored in a more secure fashion using `ansible-vault` or using [Ansible Tower credentials](#).

If your vCenter or ESXi server is not setup with proper CA certificates that can be verified from the Ansible server, then it is necessary to disable validation of these certificates by using the `validate_certs` parameter. To do this you need to set `validate_certs=False` in your playbook.

Now you need to supply the information about the existing virtual machine which will be renamed. For renaming virtual machine, `vmware_guest` module uses VMware UUID, which is unique across vCenter environment. This value is autogenerated and can not be changed. You will use `vmware_guest_facts` module to find virtual machine and get information about VMware UUID of the virtual machine.

This value will be used input for `vmware_guest` module. Specify new name to virtual machine which conforms to all VMware requirements for naming conventions as `name` parameter. Also, provide `uuid` as the value of VMware UUID.

What to expect

Running this playbook can take some time, depending on your environment and network connectivity. When the run is complete you will see

```
{
  "changed": true,
  "instance": {
```

(下页继续)

(续上页)

```

"annotation": "",
"current_snapshot": null,
"customvalues": {},
"guest_consolidation_needed": false,
"guest_question": null,
"guest_tools_status": "guestToolsNotRunning",
"guest_tools_version": "10247",
"hw_cores_per_socket": 1,
"hw_datastores": ["ds_204_2"],
"hw_esxi_host": "10.x.x.x",
"hw_eth0": {
    "addresstype": "assigned",
    "ipaddresses": [],
    "label": "Network adapter 1",
    "macaddress": "00:50:56:8c:b8:42",
    "macaddress_dash": "00-50-56-8c-b8-42",
    "portgroup_key": "dvportgroup-31",
    "portgroup_portkey": "15",
    "summary": "DVSwitch: 50 0c 3a 69 df 78 2c 7b-6e 08 0a 89 e3 a6 31 17"
},
"hw_files": ["[ds_204_2] old_vm_name/old_vm_name.vmx", "[ds_204_2] old_vm_name/
↪old_vm_name.nvram", "[ds_204_2] old_vm_name/old_vm_name.vmsd", "[ds_204_2] old_vm_name/
↪vmware.log", "[ds_204_2] old_vm_name/old_vm_name.vmdk"],
"hw_folder": "/DC1/vm",
"hw_guest_full_name": null,
"hw_guest_ha_state": null,
"hw_guest_id": null,
"hw_interfaces": ["eth0"],
"hw_is_template": false,
"hw_memtotal_mb": 1024,
"hw_name": "new_vm_name",
"hw_power_status": "poweredOff",
"hw_processor_count": 1,
"hw_product_uuid": "420cbebb-835b-980b-7050-8aea9b7b0a6d",
"hw_version": "vmx-13",
"instance_uuid": "500c60a6-b7b4-8ae5-970f-054905246a6f",
"ipv4": null,
"ipv6": null,
"module_hw": true,
"snapshots": []

```

(下页继续)

(续上页)

```
}
}
```

confirming that you've renamed the virtual machine.

Troubleshooting

If your playbook fails:

- Check if the values provided for username and password are correct.
- Check if the datacenter you provided is available.
- Check if the virtual machine specified exists and you have permissions to access the datastore.
- Ensure the full folder path you specified already exists.

Remove an existing VMware virtual machine

Topics

- *Remove an existing VMware virtual machine*
 - *Introduction*
 - *Scenario Requirements*
 - *Caveats*
 - *Example Description*
 - * *What to expect*
 - * *Troubleshooting*

Introduction

This guide will show you how to utilize Ansible to remove an existing VMware virtual machine.

Scenario Requirements

- Software
 - Ansible 2.5 or later must be installed.

- The Python module `Pyvmomi` must be installed on the Ansible control node (or Target host if not executing against localhost).
- We recommend installing the latest version with pip: `pip install Pyvmomi` (as the OS packages are usually out of date and incompatible).
- Hardware
 - At least one standalone ESXi server or
 - vCenter Server with at least one ESXi server
- Access / Credentials
 - Ansible (or the target server) must have network access to the either vCenter server or the ESXi server
 - Username and Password for vCenter or ESXi server
 - Hosts in the ESXi cluster must have access to the datastore that the template resides on.

Caveats

- All variable names and VMware object names are case sensitive.
- You need to use Python 2.7.9 version in order to use `validate_certs` option, as this version is capable of changing the SSL verification behaviours.
- `vmware_guest` module tries to mimic VMware Web UI and workflow, so the virtual machine must be in powered off state in order to remove it from the VMware inventory.

警告: The removal VMware virtual machine using `vmware_guest` module is destructive operation and can not be reverted, so it is strongly recommended to take the backup of virtual machine and related files (vmx and vmdk files) before proceeding.

Example Description

In this use case / example, user will be removing a virtual machine using name. The following Ansible playbook showcases the basic parameters that are needed for this.

```
---
- name: Remove virtual machine
  gather_facts: no
  vars_files:
    - vcenter_vars.yml
```

(下页继续)

(续上页)

```

vars:
    ansible_python_interpreter: "/usr/bin/env python3"
hosts: localhost
tasks:
    - set_fact:
        vm_name: "VM_0003"
        datacenter: "DC1"

    - name: Remove "{{ vm_name }}"
      vmware_guest:
        hostname: "{{ vcenter_server }}"
        username: "{{ vcenter_user }}"
        password: "{{ vcenter_pass }}"
        validate_certs: no
        cluster: "DC1_C1"
        name: "{{ vm_name }}"
        state: absent
      delegate_to: localhost
      register: facts

```

Since Ansible utilizes the VMware API to perform actions, in this use case it will be connecting directly to the API from localhost.

This means that playbooks will not be running from the vCenter or ESXi Server.

Note that this play disables the `gather_facts` parameter, since you don't want to collect facts about localhost.

You can run these modules against another server that would then connect to the API if localhost does not have access to vCenter. If so, the required Python modules will need to be installed on that target server. We recommend installing the latest version with pip: `pip install Pyvmomi` (as the OS packages are usually out of date and incompatible).

Before you begin, make sure you have:

- Hostname of the ESXi server or vCenter server
- Username and password for the ESXi or vCenter server
- Name of the existing Virtual Machine you want to remove

For now, you will be entering these directly, but in a more advanced playbook this can be abstracted out and stored in a more secure fashion using `ansible-vault` or using [Ansible Tower credentials](#).

If your vCenter or ESXi server is not setup with proper CA certificates that can be verified from the Ansible server, then it is necessary to disable validation of these certificates by using the `validate_certs` parameter.

To do this you need to set `validate_certs=False` in your playbook.

The name of existing virtual machine will be used as input for `vmware_guest` module via `name` parameter.

What to expect

- You will not see any JSON output after this playbook completes as compared to other operations performed using `vmware_guest` module.

```
{  
  "changed": true  
}
```

- State is changed to `True` which notifies that the virtual machine is removed from the VMware inventory. This can take some time depending upon your environment and network connectivity.

Troubleshooting

If your playbook fails:

- Check if the values provided for username and password are correct.
- Check if the datacenter you provided is available.
- Check if the virtual machine specified exists and you have permissions to access the datastore.
- Ensure the full folder path you specified already exists. It will not create folders automatically for you.

Find folder path of an existing VMware virtual machine

Topics

- *Find folder path of an existing VMware virtual machine*
 - *Introduction*
 - *Scenario Requirements*
 - *Caveats*
 - *Example Description*
 - * *What to expect*
 - * *Troubleshooting*

Introduction

This guide will show you how to utilize Ansible to find folder path of an existing VMware virtual machine.

Scenario Requirements

- Software
 - Ansible 2.5 or later must be installed.
 - The Python module `Pyvmomi` must be installed on the Ansible control node (or Target host if not executing against localhost).
 - We recommend installing the latest version with pip: `pip install Pyvmomi` (as the OS packages are usually out of date and incompatible).
- Hardware
 - At least one standalone ESXi server or
 - vCenter Server with at least one ESXi server
- Access / Credentials
 - Ansible (or the target server) must have network access to the either vCenter server or the ESXi server
 - Username and Password for vCenter or ESXi server

Caveats

- All variable names and VMware object names are case sensitive.
- You need to use Python 2.7.9 version in order to use `validate_certs` option, as this version is capable of changing the SSL verification behaviours.

Example Description

With the following Ansible playbook you can find the folder path of an existing virtual machine using name.

```
---
- name: Find folder path of an existing virtual machine
  hosts: localhost
  gather_facts: False
  vars_files:
    - vcenter_vars.yml
```

(下页继续)

(续上页)

```
vars:
  ansible_python_interpreter: "/usr/bin/env python3"
tasks:
  - set_fact:
      vm_name: "DCO_HO_VMO"

  - name: "Find folder for VM - {{ vm_name }}"
    vmware_guest_find:
      hostname: "{{ vcenter_server }}"
      username: "{{ vcenter_user }}"
      password: "{{ vcenter_pass }}"
      validate_certs: False
      name: "{{ vm_name }}"
    delegate_to: localhost
    register: vm_facts
```

Since Ansible utilizes the VMware API to perform actions, in this use case it will be connecting directly to the API from localhost.

This means that playbooks will not be running from the vCenter or ESXi Server.

Note that this play disables the `gather_facts` parameter, since you don't want to collect facts about localhost.

You can run these modules against another server that would then connect to the API if localhost does not have access to vCenter. If so, the required Python modules will need to be installed on that target server. We recommend installing the latest version with pip: `pip install Pyvmomi` (as the OS packages are usually out of date and incompatible).

Before you begin, make sure you have:

- Hostname of the ESXi server or vCenter server
- Username and password for the ESXi or vCenter server
- Name of the existing Virtual Machine for which you want to collect folder path

For now, you will be entering these directly, but in a more advanced playbook this can be abstracted out and stored in a more secure fashion using `ansible-vault` or using [Ansible Tower credentials](#).

If your vCenter or ESXi server is not setup with proper CA certificates that can be verified from the Ansible server, then it is necessary to disable validation of these certificates by using the `validate_certs` parameter. To do this you need to set `validate_certs=False` in your playbook.

The name of existing virtual machine will be used as input for `vmware_guest_find` module via `name` parameter.

What to expect

Running this playbook can take some time, depending on your environment and network connectivity. When the run is complete you will see

```
"vm_facts": {
  "changed": false,
  "failed": false,
  ...
  "folders": [
    "/FO/DC0/vm/FO"
  ]
}
```

Troubleshooting

If your playbook fails:

- Check if the values provided for username and password are correct.
- Check if the datacenter you provided is available.
- Check if the virtual machine specified exists and you have respective permissions to access VMware object.
- Ensure the full folder path you specified already exists.

Using VMware HTTP API using Ansible

Topics

- *Using VMware HTTP API using Ansible*
 - *Introduction*
 - *Scenario Requirements*
 - *Caveats*
 - *Example Description*
 - * *What to expect*
 - * *Troubleshooting*

Introduction

This guide will show you how to utilize Ansible to use VMware HTTP APIs to automate various tasks.

Scenario Requirements

- Software
 - Ansible 2.5 or later must be installed.
 - We recommend installing the latest version with pip: `pip install Pyvmomi` on the Ansible control node (as the OS packages are usually out of date and incompatible) if you are planning to use any existing VMware modules.
- Hardware
 - vCenter Server 6.5 and above with at least one ESXi server
- Access / Credentials
 - Ansible (or the target server) must have network access to either the vCenter server or the ESXi server
 - Username and Password for vCenter

Caveats

- All variable names and VMware object names are case sensitive.
- You need to use Python 2.7.9 version in order to use `validate_certs` option, as this version is capable of changing the SSL verification behaviours.
- VMware HTTP APIs are introduced in vSphere 6.5 and above so minimum level required in 6.5.
- There are very limited number of APIs exposed, so you may need to rely on XMLRPC based VMware modules.

Example Description

With the following Ansible playbook you can find the VMware ESXi host system(s) and can perform various tasks depending on the list of host systems. This is a generic example to show how Ansible can be utilized to consume VMware HTTP APIs.

```
---
- name: Example showing VMware HTTP API utilization
  hosts: localhost
```

(下页继续)

(续上页)

```

gather_facts: no
vars_files:
  - vcenter_vars.yml
vars:
  ansible_python_interpreter: "/usr/bin/env python3"
tasks:
  - name: Login into vCenter and get cookies
    uri:
      url: https://{ vcenter_server }}/rest/com/vmware/cis/session
      force_basic_auth: yes
      validate_certs: no
      method: POST
      user: "{{ vcenter_user }}"
      password: "{{ vcenter_pass }}"
      register: login

  - name: Get all hosts from vCenter using cookies from last task
    uri:
      url: https://{ vcenter_server }}/rest/vcenter/host
      force_basic_auth: yes
      validate_certs: no
      headers:
        Cookie: "{{ login.set_cookie }}"
      register: vhosts

  - name: Change Log level configuration of the given hostsystem
    vmware_host_config_manager:
      hostname: "{{ vcenter_server }}"
      username: "{{ vcenter_user }}"
      password: "{{ vcenter_pass }}"
      esxi_hostname: "{{ item.name }}"
      options:
        'Config.HostAgent.log.level': 'error'
      validate_certs: no
      loop: "{{ vhosts.json.value }}"
      register: host_config_results

```

Since Ansible utilizes the VMware HTTP API using the `uri` module to perform actions, in this use case it will be connecting directly to the VMware HTTP API from localhost.

This means that playbooks will not be running from the vCenter or ESXi Server.

Note that this play disables the `gather_facts` parameter, since you don't want to collect facts about localhost.

Before you begin, make sure you have:

- Hostname of the vCenter server
- Username and password for the vCenter server
- Version of vCenter is at least 6.5

For now, you will be entering these directly, but in a more advanced playbook this can be abstracted out and stored in a more secure fashion using `ansible-vault` or using [Ansible Tower credentials](#).

If your vCenter server is not setup with proper CA certificates that can be verified from the Ansible server, then it is necessary to disable validation of these certificates by using the `validate_certs` parameter. To do this you need to set `validate_certs=False` in your playbook.

As you can see, we are using the `uri` module in first task to login into the vCenter server and storing result in the `login` variable using `register`. In the second task, using cookies from the first task we are gathering information about the ESXi host system.

Using this information, we are changing the ESXi host system's advance configuration.

What to expect

Running this playbook can take some time, depending on your environment and network connectivity. When the run is complete you will see

```
"results": [
  {
    ...
    "invocation": {
      "module_args": {
        "cluster_name": null,
        "esxi_hostname": "10.76.33.226",
        "hostname": "10.65.223.114",
        "options": {
          "Config.HostAgent.log.level": "error"
        },
        "password": "VALUE_SPECIFIED_IN_NO_LOG_PARAMETER",
        "port": 443,
        "username": "administrator@vsphere.local",
        "validate_certs": false
      }
    },
  ],
],
```

(下页继续)

(续上页)

```

    "item": {
        "connection_state": "CONNECTED",
        "host": "host-21",
        "name": "10.76.33.226",
        "power_state": "POWERED_ON"
    },
    "msg": "Config.HostAgent.log.level changed."
    ...
}
]

```

Troubleshooting

If your playbook fails:

- Check if the values provided for username and password are correct.
- Check if you are using vCenter 6.5 and onwards to use this HTTP APIs.

参见:

VMware vSphere and Ansible From Zero to Useful by @arielsanchezmor vBrownBag session video related to VMware HTTP APIs

Sample Playbooks for using VMware HTTP APIs GitHub repo for examples of Ansible playbook to manage VMware using HTTP APIs

Troubleshooting Ansible for VMware

Topics

- *Troubleshooting Ansible for VMware*
 - *Debugging Ansible for VMware*
 - *Known issues with Ansible for VMware*
 - * *Network settings with vmware__guest in Ubuntu 18.04*
 - *Potential Workarounds*

This section lists things that can go wrong and possible ways to fix them.

Debugging Ansible for VMware

When debugging or creating a new issue, you will need information about your VMware infrastructure. You can get this information using `govc`. For example:

```
$ export GOVC_USERNAME=ESXI_OR_VCENTER_USERNAME
$ export GOVC_PASSWORD=ESXI_OR_VCENTER_PASSWORD
$ export GOVC_URL=https://ESXI_OR_VCENTER_HOSTNAME:443
$ govc find /
```

Known issues with Ansible for VMware

Network settings with `vmware_guest` in Ubuntu 18.04

Setting the network with `vmware_guest` in Ubuntu 18.04 is known to be broken, due to missing support for `netplan` in the `open-vm-tools`. This issue is tracked via:

- <https://github.com/vmware/open-vm-tools/issues/240>
- <https://github.com/ansible/ansible/issues/41133>

Potential Workarounds

There are several workarounds for this issue.

- 1) Modify the Ubuntu 18.04 images and installing `ifupdown` in them via `sudo apt install ifupdown`. If so you need to remove `netplan` via `sudo apt remove netplan.io` and you need stop `systemd-networkd` via `sudo systemctl disable systemd-networkd`.
- 2) Generate the `systemd-networkd` files with a task in your VMware Ansible role:

```
- name: make sure cache directory exists
  file: path="{{ inventory_dir }}/cache" state=directory
  delegate_to: localhost

- name: generate network templates
  template: src=network.j2 dest="{{ inventory_dir }}/cache/{{ inventory_hostname }}.
↪network"
  delegate_to: localhost

- name: copy generated files to vm
  vmware_guest_file_operation:
    hostname: "{{ vmware_general.hostname }}"
```

(下页继续)

(续上页)

```

    username: "{{ vmware_username }}"
    password: "{{ vmware_password }}"
    datacenter: "{{ vmware_general.datacenter }}"
    validate_certs: "{{ vmware_general.validate_certs }}"
    vm_id: "{{ inventory_hostname }}"
    vm_username: root
    vm_password: "{{ template_password }}"
    copy:
        src: "{{ inventory_dir }}/cache/{{ inventory_hostname }}.network"
        dest: "/etc/systemd/network/ens160.network"
        overwrite: False
    delegate_to: localhost

- name: restart systemd-networkd
  vmware_vm_shell:
    hostname: "{{ vmware_general.hostname }}"
    username: "{{ vmware_username }}"
    password: "{{ vmware_password }}"
    datacenter: "{{ vmware_general.datacenter }}"
    folder: /vm
    vm_id: "{{ inventory_hostname }}"
    vm_username: root
    vm_password: "{{ template_password }}"
    vm_shell: /bin/systemctl
    vm_shell_args: " restart systemd-networkd"
  delegate_to: localhost

- name: restart systemd-resolved
  vmware_vm_shell:
    hostname: "{{ vmware_general.hostname }}"
    username: "{{ vmware_username }}"
    password: "{{ vmware_password }}"
    datacenter: "{{ vmware_general.datacenter }}"
    folder: /vm
    vm_id: "{{ inventory_hostname }}"
    vm_username: root
    vm_password: "{{ template_password }}"
    vm_shell: /bin/systemctl
    vm_shell_args: " restart systemd-resolved"
  delegate_to: localhost

```

- 3) Wait for netplan support in open-vm-tools

Other useful VMware resources

- [PyVmomi Documentation](#)
- [VMware API and SDK Documentation](#)
- [VCSIM test container image](#)
- [Ansible VMware community wiki page](#)
- [VMware's official Guest Operating system customization matrix](#)
- [VMware Compatibility Guide](#)

Ansible VMware FAQ

`vmware_guest`

Can I deploy a virtual machine on a standalone ESXi server ?

Yes. `vmware_guest` can deploy a virtual machine with required settings on a standalone ESXi server. However, you must have a paid license to deploy virtual machines this way. If you are using the free version, the API is read-only.

Is `/vm` required for `vmware_guest` module ?

Prior to Ansible version 2.5, `folder` was an optional parameter with a default value of `/vm`.

The `folder` parameter was used to discover information about virtual machines in the given infrastructure.

Starting with Ansible version 2.5, `folder` is still an optional parameter with no default value. This parameter will be now used to identify a user's virtual machine, if multiple virtual machines or virtual machine templates are found with same name. VMware does not restrict the system administrator from creating virtual machines with same name.

1.9 Network Automation Getting Started

Ansible modules support a wide range of vendors, device types, and actions, so you can manage your entire network with a single automation tool. With Ansible, you can:

- Automate repetitive tasks to speed routine network changes and free up your time for more strategic work

- Leverage the same simple, powerful, and agentless automation tool for network tasks that operations and development use
- Separate the data model (in a playbook or role) from the execution layer (via Ansible modules) to manage heterogeneous network devices
- Benefit from community and vendor-generated sample playbooks and roles to help accelerate network automation projects
- Communicate securely with network hardware over SSH or HTTPS

Who should use this guide?

This guide is intended for network engineers using Ansible for the first time. If you understand networks but have never used Ansible, work through the guide from start to finish.

This guide is also useful for experienced Ansible users automating network tasks for the first time. You can use Ansible commands, playbooks and modules to configure hubs, switches, routers, bridges and other network devices. But network modules are different from Linux/Unix and Windows modules, and you must understand some network-specific concepts to succeed. If you understand Ansible but have never automated a network task, start with the second section.

This guide introduces basic Ansible concepts and guides you through your first Ansible commands, playbooks and inventory entries.

1.9.1 Basic Concepts

These concepts are common to all uses of Ansible, including network automation. You need to understand them to use Ansible for network automation. This basic introduction provides the background you need to follow the examples in this guide.

- 管理机
- 受控节点
- *Inventory* 仓库
- *Modules* 模块
- *Tasks* 任务
- *Playbooks* 任务剧本

管理机

任何安装了 Ansible 的服务器，你都可以使用 `ansible` or `ansible-playbook` 命令。任何安装了 Ansible 的机器都可以做为管理节点，便携式计算机，共享桌面和服务端都可以。你可以配置多个管理节点。唯一需要

注意的是，管理节点不支持 Windows 系统。

受控节点

Ansible 管理的服务器或者网络设备都称为受控节点。受控节点有时候也叫做“hosts”（主机）。受控节点不需要安装 Ansible。

Inventory 仓库

Inventory 仓库是保存受控节点信息的列表，因为有时候也叫“hostfile”，类似于系统的 hosts 文件。Inventory 仓库可以以 IP 的方式指定受控节点。Inventory 同样可以组织管理节点、新增、嵌套组等方式，非常便于扩展。更多请参考[the Working with Inventory](#)

Modules 模块

Modules 模块是 Ansible 执行代码的最小单元。每个模块都是特殊用途，从特殊类型的数据库用户管理，到特殊类型的网络设备 VLAN 接口管理。你可以在通过执行单个任务调用一个模块，也可以通过 playbook 同时调用执行钩具模块。在链接中查看 Ansible 总共包括了多少个模块。[:ref:‘模块列表 <modules_by_category>’](#)。

Tasks 任务

Ansible 执行操作的最小单位。ad-hoc 更适合临时执行命令的执行场景。

Playbooks 任务剧本

Playbooks 是任务列表的组合，通常会把常用的命令列表通过正确的语法写入到 playbook 中。Playbook 可以像普通 tasks 一样调用变量，其使用 YAML 语法，便于读、写、分享、理解。更多请参考[Intro to Playbooks](#)。

1.9.2 How Network Automation is Different

Network automation leverages the basic Ansible concepts, but there are important differences in how the network modules work. This introduction prepares you to understand the exercises in this guide.

Topics

- *How Network Automation is Different*
 - *Execution on the Control Node*
 - *Multiple Communication Protocols*
 - *Modules Organized by Network Platform*

- *Privilege Escalation: enable mode, become, and authorize*
 - * *Using become for privilege escalation*
 - * *Legacy playbooks: authorize for privilege escalation*

Execution on the Control Node

Unlike most Ansible modules, network modules do not run on the managed nodes. From a user's point of view, network modules work like any other modules. They work with ad-hoc commands, playbooks, and roles. Behind the scenes, however, network modules use a different methodology than the other (Linux/Unix and Windows) modules use. Ansible is written and executed in Python. Because the majority of network devices can not run Python, the Ansible network modules are executed on the Ansible control node, where `ansible` or `ansible-playbook` runs.

Network modules also use the control node as a destination for backup files, for those modules that offer a `backup` option. With Linux/Unix modules, where a configuration file already exists on the managed node(s), the backup file gets written by default in the same directory as the new, changed file. Network modules do not update configuration files on the managed nodes, because network configuration is not written in files. Network modules write backup files on the control node, usually in the `backup` directory under the playbook root directory.

Multiple Communication Protocols

Because network modules execute on the control node instead of on the managed nodes, they can support multiple communication protocols. The communication protocol (XML over SSH, CLI over SSH, API over HTTPS) selected for each network module depends on the platform and the purpose of the module. Some network modules support only one protocol; some offer a choice. The most common protocol is CLI over SSH. You set the communication protocol with the `ansible_connection` variable:

Value of <code>ansible_connection</code>	Protocol	Requires	Persistent?
<code>network_cli</code>	CLI over SSH	<code>network_os</code> setting	yes
<code>netconf</code>	XML over SSH	<code>network_os</code> setting	yes
<code>httpapi</code>	API over HTTP/HTTPS	<code>network_os</code> setting	yes
<code>local</code>	depends on provider	provider setting	no

注解: `httpapi` deprecates `eos_eapi` and `nxos_nxapi`. See [Httpapi Plugins](#) for details and an example.

Beginning with Ansible 2.6, we recommend using one of the persistent connection types listed above instead of `local`. With persistent connections, you can define the hosts and credentials only once, rather than in every task. You also need to set the `network_os` variable for the specific network platform you are communicating with. For more details on using each connection type on various platforms, see the [platform-specific](#) pages.

Modules Organized by Network Platform

A network platform is a set of network devices with a common operating system that can be managed by a collection of modules. The modules for each network platform share a prefix, for example:

- Arista: `eos_`
- Cisco: `ios_`, `iosxr_`, `nxos_`
- Juniper: `junos_`
- VyOS `vyos_`

All modules within a network platform share certain requirements. Some network platforms have specific differences - see the [platform-specific](#) documentation for details.

Privilege Escalation: `enable mode`, `become`, and `authorize`

Several network platforms support privilege escalation, where certain tasks must be done by a privileged user. On network devices this is called `enable mode` (the equivalent of `sudo` in *nix administration). Ansible network modules offer privilege escalation for those network devices that support it. For details of which platforms support `enable mode`, with examples of how to use it, see the [platform-specific](#) documentation.

Using `become` for privilege escalation

As of Ansible 2.6, you can use the top-level Ansible parameter `become: yes` with `become_method: enable` to run a task, play, or playbook with escalated privileges on any network platform that supports privilege escalation. You must use either `connection: network_cli` or `connection: httpapi` with `become: yes` with `become_method: enable`. If you are using `network_cli` to connect Ansible to your network devices, a `group_vars` file would look like:

```
ansible_connection: network_cli
ansible_network_os: ios
ansible_become: yes
ansible_become_method: enable
```


Legacy playbooks: authorize for privilege escalation

If you are running Ansible 2.5 or older, some network platforms support privilege escalation but not `network_cli` or `httpapi` connections. This includes all platforms in versions 2.4 and older, and HTTPS connections using `eapi` in version 2.5. With a `local` connection, you must use a `provider` dictionary and include `authorize: yes` and `auth_pass: my_enable_password`. For that use case, a `group_vars` file looks like:

```
ansible_connection: local
ansible_network_os: eos
# provider settings
eapi:
  authorize: yes
  auth_pass: " {{ secret_auth_pass }}"
  port: 80
  transport: eapi
  use_ssl: no
```

And you use the `eapi` variable in your task(s):

```
tasks:
- name: provider demo with eos
  eos_banner:
    banner: motd
    text: |
      this is test
      of multiline
      string
    state: present
    provider: "{{ eapi }}"
```

Note that while Ansible 2.6 supports the use of `connection: local` with `provider` dictionaries, this usage will be deprecated in the future and eventually removed.

For more information, see *Become and Networks*

1.9.3 Run Your First Command and Playbook

Put the concepts you learned to work with this quick tutorial. Install Ansible, execute a network configuration command manually, execute the same command with Ansible, then create a playbook so you can execute the command any time on multiple network devices.

Topics

- *Run Your First Command and Playbook*
 - *Prerequisites*
 - *Install Ansible*
 - *Establish a Manual Connection to a Managed Node*
 - *Run Your First Network Ansible Command*
 - *Create and Run Your First Network Ansible Playbook*
 - *Gathering facts from network devices*

Prerequisites

Before you work through this tutorial you need:

- Ansible 2.5 (or higher) installed
- One or more network devices that are compatible with Ansible
- Basic Linux command line knowledge
- Basic knowledge of network switch & router configuration

Install Ansible

Install Ansible using your preferred method. See [安装 Ansible](#). Then return to this tutorial.

Confirm the version of Ansible (must be ≥ 2.5):

```
ansible --version
```

Establish a Manual Connection to a Managed Node

To confirm your credentials, connect to a network device manually and retrieve its configuration. Replace the sample user and device name with your real credentials. For example, for a VyOS router:

```
ssh my_vyos_user@vyos.example.net
show config
exit
```

This manual connection also establishes the authenticity of the network device, adding its RSA key fingerprint to your list of known hosts. (If you have connected to the device before, you have already established its authenticity.)

Run Your First Network Ansible Command

Instead of manually connecting and running a command on the network device, you can retrieve its configuration with a single, stripped-down Ansible command:

```
ansible all -i vyos.example.net, -c network_cli -u my_vyos_user -k -m vyos_facts -e↵
↵ansible_network_os=vyos
```

The flags in this command set seven values:

- the host group(s) to which the command should apply (in this case, all)
- the inventory (-i, the device or devices to target - without the trailing comma -i points to an inventory file)
- the connection method (-c, the method for connecting and executing ansible)
- the user (-u, the username for the SSH connection)
- the SSH connection method (-k, please prompt for the password)
- the module (-m, the ansible module to run)
- an extra variable (-e, in this case, setting the network OS value)

NOTE: If you use `ssh-agent` with ssh keys, Ansible loads them automatically. You can omit `-k` flag.

注解: If you are running Ansible in a virtual environment, you will also need to add the variable `ansible_python_interpreter=/path/to/venv/bin/python`

Create and Run Your First Network Ansible Playbook

If you want to run this command every day, you can save it in a playbook and run it with `ansible-playbook` instead of `ansible`. The playbook can store a lot of the parameters you provided with flags at the command line, leaving less to type at the command line. You need two files for this - a playbook and an inventory file.

1. Download `first_playbook.yml`, which looks like this:

```
---
- name: Network Getting Started First Playbook
  connection: network_cli
  gather_facts: false
  hosts: all
  tasks:
```

(下页继续)

(续上页)

```

- name: Get config for VyOS devices
  vyos_facts:
    gather_subset: all

- name: Display the config
  debug:
    msg: "The hostname is {{ ansible_net_hostname }} and the OS is {{ ansible_net_
↪version }}"

```

The playbook sets three of the seven values from the command line above: the group (`hosts: all`), the connection method (`connection: network_cli`) and the module (in each task). With those values set in the playbook, you can omit them on the command line. The playbook also adds a second task to show the config output. When a module runs in a playbook, the output is held in memory for use by future tasks instead of written to the console. The debug task here lets you see the results in your shell.

2. Run the playbook with the command:

```

ansible-playbook -i vyos.example.net, -u ansible -k -e ansible_network_os=vyos first_
↪playbook.yml

```

The playbook contains one play with two tasks, and should generate output like this:

```

$ ansible-playbook -i vyos.example.net, -u ansible -k -e ansible_network_os=vyos first_
↪playbook.yml

PLAY [First Playbook]
*****

TASK [Get config for VyOS devices]
*****

ok: [vyos.example.net]

TASK [Display the config]
*****

ok: [vyos.example.net] => {
  "msg": "The hostname is vyos and the OS is VyOS"
}

```

3. Now that you can retrieve the device config, try updating it with Ansible. Download `first_playbook_ext.yml`, which is an extended version of the first playbook:

```

---

- name: Network Getting Started First Playbook Extended
  connection: network_cli
  gather_facts: false
  hosts: all
  tasks:

    - name: Get config for VyOS devices
      vyos_facts:
        gather_subset: all

    - name: Display the config
      debug:
        msg: "The hostname is {{ ansible_net_hostname }} and the OS is {{ ansible_net_
↪version }}"

    - name: Update the hostname
      vyos_config:
        backup: yes
        lines:
          - set system host-name vyos-changed

    - name: Get changed config for VyOS devices
      vyos_facts:
        gather_subset: all

    - name: Display the changed config
      debug:
        msg: "The hostname is {{ ansible_net_hostname }} and the OS is {{ ansible_net_
↪version }}"

```

The extended first playbook has four tasks in a single play. Run it with the same command you used above. The output shows you the change Ansible made to the config:

```

$ ansible-playbook -i vyos.example.net, -u ansible -k -e ansible_network_os=vyos first_
↪playbook_ext.yml

```

```

PLAY [First Playbook]

```

```

*****

```

(下页继续)

(续上页)

```

TASK [Get config for VyOS devices]
*****
ok: [vyos.example.net]

TASK [Display the config]
*****
ok: [vyos.example.net] => {
  "msg": "The hostname is vyos and the OS is VyOS"
}

TASK [Update the hostname]
*****
changed: [vyos.example.net]

TASK [Get changed config for VyOS devices]
*****
ok: [vyos.example.net]

TASK [Display the changed config]
*****
ok: [vyos.example.net] => {
  "msg": "The hostname is vyos-changed and the OS is VyOS"
}

PLAY RECAP
*****
vyos.example.net      : ok=6    changed=1    unreachable=0    failed=0

```

Gathering facts from network devices

The `gather_facts` keyword now supports gathering network device facts in standardized key/value pairs. You can feed these network facts into further tasks to manage the network device.

You can also use the new `gather_network_resources` parameter with the `network *_facts` modules (such as `eos_facts`) to return just a subset of the device configuration, as shown below.

```

- hosts: arista
  gather_facts: True
  gather_subset: min
  module_defaults:

```

(下页继续)

(续上页)

```
eos_facts:
  gather_network_resources: interfaces
```

The playbook returns the following interface facts:

```
ansible_facts:
  ansible_network_resources:
    interfaces:
      - enabled: true
        name: Ethernet1
        mtu: '1476'
      - enabled: true
        name: Loopback0
      - enabled: true
        name: Loopback1
      - enabled: true
        mtu: '1476'
        name: Tunnel0
      - enabled: true
        name: Ethernet1
      - enabled: true
        name: Tunnel1
      - enabled: true
        name: Ethernet1
```

Note that this returns a subset of what is returned by just setting `gather_subset: interfaces`.

You can store these facts and use them directly in another task, such as with the `eos_interfaces` resource module.

1.9.4 Build Your Inventory

Running a playbook without an inventory requires several command-line flags. Also, running a playbook against a single device is not a huge efficiency gain over making the same change manually. The next step to harnessing the full power of Ansible is to use an inventory file to organize your managed nodes into groups with information like the `ansible_network_os` and the SSH user. A fully-featured inventory file can serve as the source of truth for your network. Using an inventory file, a single playbook can maintain hundreds of network devices with a single command. This page shows you how to build an inventory file, step by step.

Topics

- *Build Your Inventory*
 - *Basic Inventory*
 - *Add Variables to Inventory*
 - *Group Variables within Inventory*
 - *Variable Syntax*
 - *Group Inventory by Platform*
 - *Protecting Sensitive Variables with `ansible-vault`*

Basic Inventory

First, group your inventory logically. Best practice is to group servers and network devices by their What (application, stack or microservice), Where (datacenter or region), and When (development stage):

- **What:** db, web, leaf, spine
- **Where:** east, west, floor_19, building_A
- **When:** dev, test, staging, prod

Avoid spaces, hyphens, and preceding numbers (use `floor_19`, not `19th_floor`) in your group names. Group names are case sensitive.

This tiny example data center illustrates a basic group structure. You can group groups using the syntax `[metagroupname:children]` and listing groups as members of the metagroup. Here, the group `network` includes all leafs and all spines; the group `datacenter` includes all network devices plus all web servers.

```
[leafs]
leaf01
leaf02

[spines]
spine01
spine02

[network:children]
leafs
spines

[web servers]
```

(下页继续)

(续上页)

```
webserver01
webserver02

[datacenter:children]
network
webservers
```

Add Variables to Inventory

Next, you can set values for many of the variables you needed in your first Ansible command in the inventory, so you can skip them in the ansible-playbook command. In this example, the inventory includes each network device's IP, OS, and SSH user. If your network devices are only accessible by IP, you must add the IP to the inventory file. If you access your network devices using hostnames, the IP is not necessary.

```
[leafs]
leaf01 ansible_host=10.16.10.11 ansible_network_os=vynos ansible_user=my_vynos_user
leaf02 ansible_host=10.16.10.12 ansible_network_os=vynos ansible_user=my_vynos_user

[spines]
spine01 ansible_host=10.16.10.13 ansible_network_os=vynos ansible_user=my_vynos_user
spine02 ansible_host=10.16.10.14 ansible_network_os=vynos ansible_user=my_vynos_user

[network:children]
leafs
spines

[servers]
server01 ansible_host=10.16.10.15 ansible_user=my_server_user
server02 ansible_host=10.16.10.16 ansible_user=my_server_user

[datacenter:children]
leafs
spines
servers
```

Group Variables within Inventory

When devices in a group share the same variable values, such as OS or SSH user, you can reduce duplication and simplify maintenance by consolidating these into group variables:

```
[leafs]
leaf01 ansible_host=10.16.10.11
leaf02 ansible_host=10.16.10.12

[leafs:vars]
ansible_network_os=vynos
ansible_user=my_vynos_user

[spines]
spine01 ansible_host=10.16.10.13
spine02 ansible_host=10.16.10.14

[spines:vars]
ansible_network_os=vynos
ansible_user=my_vynos_user

[network:children]
leafs
spines

[servers]
server01 ansible_host=10.16.10.15
server02 ansible_host=10.16.10.16

[datacenter:children]
leafs
spines
servers
```

Variable Syntax

The syntax for variable values is different in inventory, in playbooks and in `group_vars` files, which are covered below. Even though playbook and `group_vars` files are both written in YAML, you use variables differently in each.

- In an ini-style inventory file you **must** use the syntax `key=value` for variable values:
`ansible_network_os=vynos.`
- In any file with the `.yaml` or `.yml` extension, including playbooks and `group_vars` files, you **must** use YAML syntax: `key: value`
 - In `group_vars` files, use the full key name: `ansible_network_os: vynos.`

- In playbooks, use the short-form key name, which drops the `ansible` prefix: `network_os: vyos`

Group Inventory by Platform

As your inventory grows, you may want to group devices by platform. This allows you to specify platform-specific variables easily for all devices on that platform:

```
[vyos_leafs]
leaf01 ansible_host=10.16.10.11
leaf02 ansible_host=10.16.10.12

[vyos_spines]
spine01 ansible_host=10.16.10.13
spine02 ansible_host=10.16.10.14

[vyos:children]
vyos_leafs
vyos_spines

[vyos:vars]
ansible_connection=network_cli
ansible_network_os=vyos
ansible_user=my_vyos_user

[network:children]
vyos

[servers]
server01 ansible_host=10.16.10.15
server02 ansible_host=10.16.10.16

[datacenter:children]
vyos
servers
```

With this setup, you can run `first_playbook.yml` with only two flags:

```
ansible-playbook -i inventory -k first_playbook.yml
```

With the `-k` flag, you provide the SSH password(s) at the prompt. Alternatively, you can store SSH and other secrets and passwords securely in your `group_vars` files with `ansible-vault`.

Protecting Sensitive Variables with `ansible-vault`

The `ansible-vault` command provides encryption for files and/or individual variables like passwords. This tutorial will show you how to encrypt a single SSH password. You can use the commands below to encrypt other sensitive information, such as database passwords, privilege-escalation passwords and more.

First you must create a password for `ansible-vault` itself. It is used as the encryption key, and with this you can encrypt dozens of different passwords across your Ansible project. You can access all those secrets (encrypted values) with a single password (the `ansible-vault` password) when you run your playbooks. Here's a simple example.

Create a file and write your password for `ansible-vault` to it:

```
echo "my-ansible-vault-pw" > ~/my-ansible-vault-pw-file
```

Create the encrypted ssh password for your VyOS network devices, pulling your `ansible-vault` password from the file you just created:

```
ansible-vault encrypt_string --vault-id my_user@~/my-ansible-vault-pw-file 'VyOS_SSH_
↪password' --name 'ansible_password'
```

If you prefer to type your `ansible-vault` password rather than store it in a file, you can request a prompt:

```
ansible-vault encrypt_string --vault-id my_user@prompt 'VyOS_SSH_password' --name
↪'ansible_password'
```

and type in the vault password for `my_user`.

The `--vault-id` flag allows different vault passwords for different users or different levels of access. The output includes the user name `my_user` from your `ansible-vault` command and uses the YAML syntax `key: value`:

```
ansible_password: !vault |
    $ANSIBLE_VAULT;1.2;AES256;my_user
    66386134653765386232383236303063623663343437643766386435663632343266393064373933
    3661666132363339303639353538316662616638356631650a316338316663666439383138353032
    63393934343937373637306162366265383461316334383132626462656463363630613832313562
    3837646266663835640a313164343535316666653031353763613037656362613535633538386539
    65656439626166666363323435613131643066353762333232326232323565376635
Encryption successful
```

This is an example using an extract from a YAML inventory, as the INI format does not support inline vaults:

```
...

vyos: # this is a group in yaml inventory, but you can also do under a host
  vars:
    ansible_connection: network_cli
    ansible_network_os: vyos
    ansible_user: my_vyos_user
    ansible_password: !vault |
        $ANSIBLE_VAULT;1.2;AES256;my_user
        66386134653765386232383236303063623663343437643766386435663632343266393064373933
        3661666132363339303639353538316662616638356631650a316338316663666439383138353032
        63393934343937373637306162366265383461316334383132626462656463363630613832313562
        3837646266663835640a313164343535316666653031353763613037656362613535633538386539
        65656439626166666363323435613131643066353762333232326232323565376635
    ...
```

To use an inline vaulted variables with an INI inventory you need to store it in a ‘vars’ file in YAML format, it can reside in `host_vars/` or `group_vars/` to be automatically picked up or referenced from a play via `vars_files` or `include_vars`.

To run a playbook with this setup, drop the `-k` flag and add a flag for your `vault-id`:

```
ansible-playbook -i inventory --vault-id my_user@~/my-ansible-vault-pw-file first_
↪playbook.yml
```

Or with a prompt instead of the vault password file:

```
ansible-playbook -i inventory --vault-id my_user@prompt first_playbook.yml
```

To see the original value, you can use the debug module. Please note if your YAML file defines the `ansible_connection` variable (as we used in our example), it will take effect when you execute the command below. To prevent this, please make a copy of the file without the `ansible_connection` variable.

```
cat vyos.yml | grep -v ansible_connection >> vyos_no_connection.yml

ansible localhost -m debug -a var="ansible_password" -e "@vyos_no_connection.yml" --ask-
↪vault-pass
Vault password:

localhost | SUCCESS => {
  "ansible_password": "VyOS_SSH_password"
```

(下页继续)

```
}

```

警告: Vault content can only be decrypted with the password that was used to encrypt it. If you want to stop using one password and move to a new one, you can update and re-encrypt existing vault content with `ansible-vault rekey myfile`, then provide the old password and the new password. Copies of vault content still encrypted with the old password can still be decrypted with old password.

For more details on building inventory files, see *the introduction to inventory*; for more details on ansible-vault, see *the full Ansible Vault documentation*.

Now that you understand the basics of commands, playbooks, and inventory, it's time to explore some more complex Ansible Network examples.

1.9.5 Use Ansible network roles

Roles are sets of Ansible defaults, files, tasks, templates, variables, and other Ansible components that work together. As you saw on *Run Your First Command and Playbook*, moving from a command to a playbook makes it easy to run multiple tasks and repeat the same tasks in the same order. Moving from a playbook to a role makes it even easier to reuse and share your ordered tasks. You can look at *Ansible Galaxy*, which lets you share your roles and use others' roles, either directly or as inspiration.

- *Understanding roles*
 - *A sample DNS playbook*
 - *Convert the playbook into a role*
 - *Variable precedence*
 - * *Lowest precedence*
 - * *Highest precedence*
- *Ansible supported network roles*
- *Network roles release cycle*
 - *Update an installed role*

Understanding roles

So what exactly is a role, and why should you care? Ansible roles are basically playbooks broken up into a known file structure. Moving to roles from a playbook makes sharing, reading, and updating your Ansible

workflow easier. Users can write their own roles. So for example, you don't have to write your own DNS playbook. Instead, you specify a DNS server and a role to configure it for you.

To simplify your workflow even further, the Ansible Network team has written a series of roles for common network use cases. Using these roles means you don't have to reinvent the wheel. Instead of writing and maintaining your own `create_vlan` playbooks or roles, you can concentrate on designing, codifying and maintaining the parser templates that describe your network topologies and inventory, and let Ansible's network roles do the work. See the [network-related roles](#) on Ansible Galaxy.

A sample DNS playbook

To demonstrate the concept of what a role is, the example `playbook.yml` below is a single YAML file containing a two-task playbook. This Ansible Playbook configures the hostname on a Cisco IOS XE device, then it configures the DNS (domain name system) servers.

```
---
- name: configure cisco routers
  hosts: routers
  connection: network_cli
  gather_facts: no
  vars:
    dns: "8.8.8.8 8.8.4.4"

  tasks:
    - name: configure hostname
      ios_config:
        lines: hostname {{ inventory_hostname }}

    - name: configure DNS
      ios_config:
        lines: ip name-server {{dns}}
```

If you run this playbook using the `ansible-playbook` command, you'll see the output below. This example used `-l` option to limit the playbook to only executing on the `rtr1` node.

```
[user@ansible ~]$ ansible-playbook playbook.yml -l rtr1

PLAY [configure cisco routers] *****

TASK [configure hostname] *****
changed: [rtr1]
```

(下页继续)

(续上页)

```
TASK [configure DNS] *****
changed: [rtr1]

PLAY RECAP *****
rtr1                : ok=2    changed=2    unreachable=0    failed=0
```

This playbook configured the hostname and DNS servers. You can verify that configuration on the Cisco IOS XE **rtr1** router:

```
rtr1#sh run | i name
hostname rtr1
ip name-server 8.8.8.8 8.8.4.4
```

Convert the playbook into a role

The next step is to convert this playbook into a reusable role. You can create the directory structure manually, or you can use `ansible-galaxy init` to create the standard framework for a role.

```
[user@ansible ~]$ ansible-galaxy init system-demo
[user@ansible ~]$ cd system-demo/
[user@ansible system-demo]$ tree
.
├── defaults
│   └── main.yml
├── files
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── README.md
├── tasks
│   └── main.yml
├── templates
├── tests
│   ├── inventory
│   └── test.yml
├── vars
│   └── main.yml
```

This first demonstration uses only the **tasks** and **vars** directories. The directory structure would look as follows:


```
[user@ansible system-demo]$ tree
.
├── tasks
│   └── main.yml
└── vars
    └── main.yml
```

Next, move the content of the `vars` and `tasks` sections from the original Ansible Playbook into the role. First, move the two tasks into the `tasks/main.yml` file:

```
[user@ansible system-demo]$ cat tasks/main.yml
---
- name: configure hostname
  ios_config:
    lines: hostname {{ inventory_hostname }}

- name: configure DNS
  ios_config:
    lines: ip name-server {{ dns }}
```

Next, move the variables into the `vars/main.yml` file:

```
[user@ansible system-demo]$ cat vars/main.yml
---
dns: "8.8.8.8 8.8.4.4"
```

Finally, modify the original Ansible Playbook to remove the `tasks` and `vars` sections and add the keyword `roles` with the name of the role, in this case `system-demo`. You'll have this playbook:

```
---
- name: configure cisco routers
  hosts: routers
  connection: network_cli
  gather_facts: no

  roles:
    - system-demo
```

To summarize, this demonstration now has a total of three directories and three YAML files. There is the `system-demo` folder, which represents the role. This `system-demo` contains two folders, `tasks` and `vars`. There is a `main.yml` in each respective folder. The `vars/main.yml` contains the variables from `playbook.yml`. The `tasks/main.yml` contains the tasks from `playbook.yml`. The `playbook.yml` file has been modified

to call the role rather than specifying vars and tasks directly. Here is a tree of the current working directory:

```
[user@ansible ~]$ tree
.
├── playbook.yml
├── system-demo
│   ├── tasks
│   │   └── main.yml
│   └── vars
│       └── main.yml
```

Running the playbook results in identical behavior with slightly different output:

```
[user@ansible ~]$ ansible-playbook playbook.yml -l rtr1

PLAY [configure cisco routers] *****

TASK [system-demo : configure hostname] *****
ok: [rtr1]

TASK [system-demo : configure DNS] *****
ok: [rtr1]

PLAY RECAP *****
rtr1                : ok=2    changed=0    unreachable=0    failed=0
```

As seen above each task is now prepended with the role name, in this case `system-demo`. When running a playbook that contains several roles, this will help pinpoint where a task is being called from. This playbook returned `ok` instead of `changed` because it has identical behavior for the single file playbook we started from.

As before, the playbook will generate the following configuration on a Cisco IOS-XE router:

```
rtr1#sh run | i name
hostname rtr1
ip name-server 8.8.8.8 8.8.4.4
```

This is why Ansible roles can be simply thought of as deconstructed playbooks. They are simple, effective and reusable. Now another user can simply include the `system-demo` role instead of having to create a custom “hard coded” playbook.

Variable precedence

What if you want to change the DNS servers? You aren't expected to change the `vars/main.yml` within the role structure. Ansible has many places where you can specify variables for a given play. See [Using Variables](#) for details on variables and precedence. There are actually 21 places to put variables. While this list can seem overwhelming at first glance, the vast majority of use cases only involve knowing the spot for variables of least precedence and how to pass variables with most precedence. See [Variable precedence: Where should I put a variable?](#) for more guidance on where you should put variables.

Lowest precedence

The lowest precedence is the `defaults` directory within a role. This means all the other 20 locations you could potentially specify the variable will all take higher precedence than `defaults`, no matter what. To immediately give the vars from the `system-demo` role the least precedence, rename the `vars` directory to `defaults`.

```
[user@ansible system-demo]$ mv vars defaults
[user@ansible system-demo]$ tree
.
├── defaults
│   └── main.yml
├── tasks
│   └── main.yml
```

Add a new `vars` section to the playbook to override the default behavior (where the variable `dns` is set to 8.8.8.8 and 8.8.4.4). For this demonstration, set `dns` to 1.1.1.1, so `playbook.yml` becomes:

```
---
- name: configure cisco routers
  hosts: routers
  connection: network_cli
  gather_facts: no
  vars:
    dns: 1.1.1.1
  roles:
    - system-demo
```

Run this updated playbook on **rtr2**:

```
[user@ansible ~]$ ansible-playbook playbook.yml -l rtr2
```

The configuration on the **rtr2** Cisco router will look as follows:

```
rtr2#sh run | i name-server
ip name-server 1.1.1.1
```

The variable configured in the playbook now has precedence over the `defaults` directory. In fact, any other spot you configure variables would win over the values in the `defaults` directory.

Highest precedence

Specifying variables in the `defaults` directory within a role will always take the lowest precedence, while specifying `vars` as extra vars with the `-e` or `--extra-vars=` will always take the highest precedence, no matter what. Re-running the playbook with the `-e` option overrides both the `defaults` directory (8.8.4.4 and 8.8.8.8) as well as the newly created `vars` within the playbook that contains the 1.1.1.1 dns server.

```
[user@ansible ~]$ ansible-playbook playbook.yml -e "dns=192.168.1.1" -l rtr3
```

The result on the Cisco IOS XE router will only contain the highest precedence setting of 192.168.1.1:

```
rtr3#sh run | i name-server
ip name-server 192.168.1.1
```

How is this useful? Why should you care? Extra vars are commonly used by network operators to override defaults. A powerful example of this is with Red Hat Ansible Tower and the Survey feature. It is possible through the web UI to prompt a network operator to fill out parameters with a Web form. This can be really simple for non-technical playbook writers to execute a playbook using their Web browser. See [Ansible Tower Job Template Surveys](#) for more details.

Ansible supported network roles

The Ansible Network team develops and supports a set of [network-related roles](#) on Ansible Galaxy. You can use these roles to jump start your network automation efforts. These roles are updated approximately every two weeks to give you access to the latest Ansible networking content.

These roles come in the following categories:

- **User roles** - User roles focus on tasks, such as managing your configuration. Use these roles, such as `config_manager` and `cloud_vpn`, directly in your playbooks. These roles are platform/provider agnostic, allowing you to use the same roles and playbooks across different network platforms or cloud providers.
- **Platform provider roles** - Provider roles translate between the user roles and the various network OSs, each of which has a different API. Each provider role accepts input from a supported user role and translates it for a specific network OS. Network user roles depend on these provider roles to implement their functions. For example, the `config_manager` user role uses the `cisco_ios` provider role to implement tasks on Cisco IOS network devices.

- **Cloud provider and provisioner roles** - Similarly, cloud user roles depend on cloud provider and provisioner roles to implement cloud functions for specific cloud providers. For example, the `cloud_vpn` role depends on the `aws` provider role to communicate with AWS.

You need to install at least one platform provider role for your network user roles, and set `ansible_network_provider` to that provider (for example, `ansible_network_provider: ansible-network.cisco_ios`). Ansible Galaxy automatically installs any other dependencies listed in the role details on Ansible Galaxy.

For example, to use the `config_manager` role with Cisco IOS devices, you would use the following commands:

```
[user@ansible]$ ansible-galaxy install ansible-network.cisco_ios
[user@ansible]$ ansible-galaxy install ansible-network.config_manager
```

Roles are fully documented with examples in Ansible Galaxy on the **Read Me** tab for each role.

Network roles release cycle

The Ansible network team releases updates and new roles every two weeks. The role details on Ansible Galaxy lists the role versions available, and you can look in the GitHub repository to find the changelog file (for example, the `cisco_ios CHANGELOG.rst`) that lists what has changed in each version of the role.

The Ansible Galaxy role version has two components:

- Major release number - (for example, 2.6) which shows the Ansible engine version this role supports.
- Minor release number (for example .1) which denotes the role release cycle and does not reflect the Ansible engine minor release version.

Update an installed role

The Ansible Galaxy page for a role lists all available versions. To update a locally installed role to a new or different version, use the `ansible-galaxy install` command with the version and `--force` option. You may also need to manually update any dependent roles to support this version. See the role **Read Me** tab in Galaxy for dependent role minimum version requirements.

```
[user@ansible]$ ansible-galaxy install ansible-network.network_engine,v2.7.0 --force
[user@ansible]$ ansible-galaxy install ansible-network.cisco_nxos,v2.7.1 --force
```

参见:

[Ansible Galaxy documentation](#) Ansible Galaxy user guide

[Ansible supported network roles](#) List of Ansible-supported network and cloud roles on Ansible Galaxy

1.9.6 Beyond the basics

This page introduces some concepts that help you manage your Ansible workflow with directory structure and source control. Like the Basic Concepts at the beginning of this guide, these intermediate concepts are common to all uses of Ansible.

- *A typical Ansible filetree*
- *Tracking changes to inventory and playbooks: source control with git*

A typical Ansible filetree

Ansible expects to find certain files in certain places. As you expand your inventory and create and run more network playbooks, keep your files organized in your working Ansible project directory like this:

```
.
├── backup
│   ├── vyos.example.net_config.2018-02-08@11:10:15
│   └── vyos.example.net_config.2018-02-12@08:22:41
├── first_playbook.yml
├── inventory
├── group_vars
│   ├── vyos.yml
│   └── eos.yml
├── roles
│   ├── static_route
│   └── system
├── second_playbook.yml
└── third_playbook.yml
```

The `backup` directory and the files in it get created when you run modules like `vyos_config` with the `backup: yes` parameter.

Tracking changes to inventory and playbooks: source control with git

As you expand your inventory, roles and playbooks, you should place your Ansible projects under source control. We recommend `git` for source control. `git` provides an audit trail, letting you track changes, roll back mistakes, view history and share the workload of managing, maintaining and expanding your Ansible ecosystem. There are plenty of tutorials and guides to using `git` available.

1.9.7 Working with network connection options

Network modules can support multiple connection protocols, such as `network_cli`, `netconf`, and `httpapi`. These connections include some common options you can set to control how the connection to your network device behaves.

Common options are:

- `become` and `become_method` as described in *Privilege Escalation: enable mode, become, and authorize*.
- `network_os` - set to match your network platform you are communicating with. See the *platform-specific* pages.
- `remote_user` as described in *Setting a remote user*.
- Timeout options - `persistent_command_timeout`, `persistent_connect_timeout`, and `timeout`.

Setting timeout options

When communicating with a remote device, you have control over how long Ansible maintains the connection to that device, as well as how long Ansible waits for a command to complete on that device. Each of these options can be set as variables in your playbook files, environment variables, or settings in your `ansible.cfg` file.

For example, the three options for controlling the connection timeout are as follows.

Using vars (per task):

```
- name: save running-config
  ios_command:
    commands: copy running-config startup-config
    vars:
      ansible_command_timeout: 30
```

Using the environment variable:

```
$export ANSIBLE_PERSISTENT_COMMAND_TIMEOUT=30
```

Using the global configuration (in `ansible.cfg`)

```
[persistent_connection ]
command_timeout = 30
```

See *Variable precedence: Where should I put a variable?* for details on the relative precedence of each of these variables. See the individual connection type to understand each option.

1.9.8 Resources and next steps

- *Documents*
- *Events (on video and in person)*
- *GitHub repos*
- *IRC and Slack*

Documents

Read more about Ansible for Network Automation:

- Network Automation on the [Ansible website](#)
- Ansible Network [Blog posts](#)

Events (on video and in person)

All sessions at Ansible events are recorded and include many Network-related topics (use Filter by Category to view only Network topics). You can also join us for future events in your area. See:

- [Recorded AnsibleFests](#)
- [Recorded AnsibleAutomates](#)
- [Upcoming Ansible Events](#) page.

GitHub repos

Ansible hosts module code, examples, demonstrations, and other content on GitHub. Anyone with a GitHub account is able to create Pull Requests (PRs) or issues on these repos:

- [Network-Automation](#) is an open community for all things network automation. Have an idea, some playbooks, or roles to share? Email ansible-network@redhat.com and we will add you as a contributor to the repository.
- [Ansible](#) is the main codebase, including code for network modules
- [ansible-network](#) is the main codebase for the Ansible network team roles

IRC and Slack

Join us on:

- Freenode IRC - [#ansible-network](#) Freenode channel

- Slack - <https://ansiblenetwork.slack.com>

1.10 Network Automation Advanced Topics

Once you have mastered the basics of network automation with Ansible, as presented in *Network Automation Getting Started*, use this guide understand platform-specific details, optimization, and troubleshooting tips for Ansible for network automation.

Who should use this guide?

This guide is intended for network engineers using Ansible for automation. It covers advanced topics. If you understand networks and Ansible, this guide is for you. You may read through the entire guide if you choose, or use the links below to find the specific information you need.

If you're new to Ansible, or new to using Ansible for network automation, start with the *Network Automation Getting Started*.

1.10.1 Network resource modules

Ansible 2.9 introduced network resource modules to simplify and standardize how you manage different network devices.

- *Understanding network resource modules*
- *Network resource module states*
- *Using network resource modules*
- *Example: Verifying the network device configuration has not changed*

Understanding network resource modules

Network devices separate configuration into sections (such as interfaces, VLANs, etc) that apply to a network service. Ansible network resource modules take advantage of this to allow you to configure subsections or *resources* within the network device configuration. Network resource modules provide a consistent experience across different network devices.

Network resource module states

You use the network resource modules by assigning a state to what you want the module to do. The resource modules support the following states:

merged Ansible merges the on-device configuration with the provided configuration in the task.

replaced Ansible replaces the on-device configuration subsection with the provided configuration subsection in the task.

overridden Ansible overrides the on-device configuration for the resource with the provided configuration in the task. Use caution with this state as you could remove your access to the device (for example, by overriding the management interface configuration).

deleted Ansible deletes the on-device configuration subsection and restores any default settings.

gathered Ansible displays the resource details gathered from the network device and accessed with the **gathered** key in the result.

rendered Ansible renders the provided configuration in the task in the device-native format (for example, Cisco IOS CLI). Ansible returns this rendered configuration in the **rendered** key in the result. Note this state does not communicate with the network device and can be used offline.

parsed Ansible parses the configuration from the **running_configuration** option into Ansible structured data in the **parsed** key in the result. Note this does not gather the configuration from the network device so this state can be used offline.

Using network resource modules

This example configures L3 interface resource on a Cisco IOS device, based on different state settings.

```
- name: configure l3 interface
  ios_l3_interfaces:
    config: "{{ config }}"
    state: <state>
```

The following table shows an example of how an initial resource configuration changes with this task for different states.

Resource starting configuration	task-provided configuration (YAML)	Final resource configuration on device
<pre>interface loopback100 ip address 10.10.1.100 ↪255.255.255.0 ipv6 address FC00:100/64</pre>	<pre>config: - ipv6: - address: fc00::100/64 - address: fc00::101/64 name: loopback100</pre>	<p><i>merged</i></p> <pre>interface loopback100 ip address 10.10.1. ↪100 255.255.255.0 ipv6 address ↪FC00:100/64 ipv6 address ↪FC00:101/64</pre>
		<p><i>replaced</i></p> <pre>interface loopback100 no ip address ipv6 address ↪FC00:100/64 ipv6 address ↪FC00:101/64</pre>
		<p><i>overridden</i> Incorrect use case. This would remove all interfaces from the device (including the mgmt interface) except the configured loopback100</p>
		<p><i>deleted</i></p> <pre>interface loopback100 no ip address</pre>

Network resource modules return the following details:

- The *before* state - the existing resource configuration before the task was executed.
- The *after* state - the new resource configuration that exists on the network device after the task was executed.
- Commands - any commands configured on the device.

```
ok: [nxos101] =>
  result:
```

(下页继续)

(续上页)

```
after:
  contact: IT Support
  location: Room E, Building 6, Seattle, WA 98134
  users:
    - algorithm: md5
      group: network-admin
      localized_key: true
      password: '0x73fd9a2cc8c53ed3dd4ed8f4ff157e69'
      privacy_password: '0x73fd9a2cc8c53ed3dd4ed8f4ff157e69'
      username: admin
before:
  contact: IT Support
  location: Room E, Building 5, Seattle HQ
  users:
    - algorithm: md5
      group: network-admin
      localized_key: true
      password: '0x73fd9a2cc8c53ed3dd4ed8f4ff157e69'
      privacy_password: '0x73fd9a2cc8c53ed3dd4ed8f4ff157e69'
      username: admin
changed: true
commands:
  - snmp-server location Room E, Building 6, Seattle, WA 98134
failed: false
```

Example: Verifying the network device configuration has not changed

The following playbook uses the `eos_l3_interfaces` module to gather a subset of the network device configuration (Layer 3 interfaces only) and verifies the information is accurate and has not changed. This playbook passes the results of `eos_facts` directly to the `eos_l3_interfaces` module.

```
- name: Example of facts being pushed right back to device.
  hosts: arista
  gather_facts: false
  tasks:
    - name: grab arista eos facts
      eos_facts:
        gather_subset: min
        gather_network_resources: l3_interfaces
```

(下页继续)

(续上页)

```

- name: Ensure that the IP address information is accurate.
  eos_l3_interfaces:
    config: "{{ ansible_network_resources['l3_interfaces'] }}"
    register: result

- name: Ensure config did not change.
  assert:
    that: not result.changed

```

参见:

Network Features in Ansible 2.9 A introductory blog post on network resource modules.

Deep Dive into Network Resource Modules A deeper dive presentation into network resource modules.

1.10.2 Ansible Network FAQ

Topics

- *Ansible Network FAQ*
 - *How can I improve performance for network playbooks?*
 - * *Consider strategy: free if you are running on multiple hosts*
 - * *Execute show running only if you absolutely must*
 - * *Use ProxyCommand only if you absolutely must*
 - * *Set --forks to match your needs*
 - *Why is my output sometimes replaced with *****?*
 - *Why do the *_config modules always return changed=true with abbreviated commands?*

How can I improve performance for network playbooks?

Consider strategy: free if you are running on multiple hosts

The **strategy** plugin tells Ansible how to order multiple tasks on multiple hosts. *Strategy* is set at the playbook level.

The default strategy is **linear**. With strategy set to **linear**, Ansible waits until the current task has run on all hosts before starting the next task on any host. Ansible may have forks free, but will not use them until all hosts have completed the current task. If each task in your playbook must succeed on all hosts before you run the next task, use the **linear** strategy.

Using the **free** strategy, Ansible uses available forks to execute tasks on each host as quickly as possible. Even if an earlier task is still running on one host, Ansible executes later tasks on other hosts. The **free** strategy uses available forks more efficiently. If your playbook stalls on each task, waiting for one slow host, consider using **strategy: free** to boost overall performance.

Execute **show running** only if you absolutely must

The **show running** command is the most resource-intensive command to execute on a network device, because of the way queries are handled by the network OS. Using the command in your Ansible playbook will slow performance significantly, especially on large devices; repeating it will multiply the performance hit. If you have a playbook that checks the running config, then executes changes, then checks the running config again, you should expect that playbook to be very slow.

Use **ProxyCommand** only if you absolutely must

Network modules support the use of a *proxy or jump host* with the **ProxyCommand** parameter. However, when you use a jump host, Ansible must open a new SSH connection for every task, even if you are using a persistent connection type (**network_cli** or **netconf**). To maximize the performance benefits of the persistent connection types introduced in version 2.5, avoid using jump hosts whenever possible.

Set **--forks** to match your needs

Every time Ansible runs a task, it forks its own process. The **--forks** parameter defines the number of concurrent tasks - if you retain the default setting, which is **--forks=5**, and you are running a playbook on 10 hosts, five of those hosts will have to wait until a fork is available. Of course, the more forks you allow, the more memory and processing power Ansible will use. Since most network tasks are run on the control host, this means your laptop can quickly become cpu- or memory-bound.

Why is my output sometimes replaced with *********?

Ansible replaces any string marked **no_log**, including passwords, with ********* in Ansible output. This is done by design, to protect your sensitive data. Most users are happy to have their passwords redacted. However, Ansible replaces every string that matches your password with *********. If you use a common word for your password, this can be a problem. For example, if you choose **Admin** as your password, Ansible will replace every instance of the word **Admin** with ********* in your output. This may make your output

harder to read. To avoid this problem, select a secure password that will not occur elsewhere in your Ansible output.

Why do the `*_config` modules always return `changed=true` with abbreviated commands?

When you issue commands directly on a network device, you can use abbreviated commands. For example, `int g1/0/11` and `interface GigabitEthernet1/0/11` do the same thing; `shut` and `shutdown` do the same thing. Ansible Network `*_command` modules work with abbreviations, because they run commands through the network OS.

When committing configuration, however, the network OS converts abbreviations into long-form commands. Whether you use `shut` or `shutdown` on `GigabitEthernet1/0/11`, the result in the configuration is the same: `shutdown`.

Ansible Network `*_config` modules compare the text of the commands you specify in `lines` to the text in the configuration. If you use `shut` in the `lines` section of your task, and the configuration reads `shutdown`, the module returns `changed=true` even though the configuration is already correct. Your task will update the configuration every time it runs.

To avoid this problem, use long-form commands with the `*_config` modules:

```
---
- hosts: all
  gather_facts: no
  tasks:
    - ios_config:
      lines:
        - shutdown
      parents: interface GigabitEthernet1/0/11
```

1.10.3 Ansible Network Examples

This document describes some examples of using Ansible to manage your network infrastructure.

- *Prerequisites*
- *Groups and variables in an inventory file*
 - *Ansible vault for password encryption*
 - *Common inventory variables*
 - *Privilege escalation*
 - *Jump hosts*

- *Example 1: collecting facts and creating backup files with a playbook*
 - *Step 1: Creating the inventory*
 - *Step 2: Creating the playbook*
 - *Step 3: Running the playbook*
 - *Step 4: Examining the playbook results*
- *Example 2: simplifying playbooks with network agnostic modules*
 - *Sample playbook with platform-specific modules*
 - *Simplified playbook with `cli_command` network agnostic module*
 - *Using multiple prompts with the `cli_command`*
- *Implementation Notes*
 - *Demo variables*
 - *Get running configuration*
- *Troubleshooting*

Prerequisites

This example requires the following:

- **Ansible 2.5** (or higher) installed. See [安装 Ansible](#) for more information.
- One or more network devices that are compatible with Ansible.
- Basic understanding of YAML *YAML Syntax*.
- Basic understanding of Jinja2 templates. See *Templating (Jinja2)* for more information.
- Basic Linux command line use.
- Basic knowledge of network switch & router configurations.

Groups and variables in an inventory file

An **inventory** file is a YAML or INI-like configuration file that defines the mapping of hosts into groups.

In our example, the inventory file defines the groups **eos**, **ios**, **vyos** and a “group of groups” called **switches**. Further details about subgroups and inventory files can be found in the *Ansible inventory Group documentation*.

Because Ansible is a flexible tool, there are a number of ways to specify connection information and credentials. We recommend using the `[my_group:vars]` capability in your inventory file. Here’s what it would look like if you specified your SSH passwords (encrypted with Ansible Vault) among your variables:


```

[all:vars]
# these defaults can be overridden for any group in the [group:vars] section
ansible_connection=network_cli
ansible_user=ansible

[switches:children]
eos
ios
vyos

[eos]
veos01 ansible_host=veos-01.example.net
veos02 ansible_host=veos-02.example.net
veos03 ansible_host=veos-03.example.net
veos04 ansible_host=veos-04.example.net

[eos:vars]
ansible_become=yes
ansible_become_method=enable
ansible_network_os=eos
ansible_user=my_eos_user
ansible_password= !vault |
                   $ANSIBLE_VAULT;1.1;AES256
                   ␣
↪37373735393636643261383066383235363664386633386432343236663533343730353361653735
                   ␣
↪6131363539383931353931653533356337353539373165320a316465383138636532343463633236
                   ␣
↪37623064393838353962386262643230303438323065356133373930646331623731656163623333
                   ␣
↪3431353332343530650a373038366364316135383063356531633066343434623631303166626532
                   9562

[ios]
ios01 ansible_host=ios-01.example.net
ios02 ansible_host=ios-02.example.net
ios03 ansible_host=ios-03.example.net

[ios:vars]
ansible_become=yes

```

(下页继续)

(续上页)

```

ansible_become_method=enable
ansible_network_os=ios
ansible_user=my_ios_user
ansible_password= !vault |
    $ANSIBLE_VAULT;1.1;AES256
    34623431313336343132373235313066376238386138316466636437653938623965383732373130
    3466363834613161386538393463663861636437653866620a373136356366623765373530633735
    34323262363835346637346261653137626539343534643962376139366330626135393365353739
    3431373064656165320a333834613461613338626161633733343566666630366133623265303563
    8472

[vyos]
vyos01 ansible_host=vyos-01.example.net
vyos02 ansible_host=vyos-02.example.net
vyos03 ansible_host=vyos-03.example.net

[vyos:vars]
ansible_network_os=vyos
ansible_user=my_vyos_user
ansible_password= !vault |
    $ANSIBLE_VAULT;1.1;AES256
    39336231636137663964343966653162353431333566633762393034646462353062633264303765
    6331643066663534383564343537343334633031656538370a333737656236393835383863306466
    62633364653238323333633337313163616566383836643030336631333431623631396364663533
    3665626431626532630a353564323566316162613432373738333064366130303637616239396438
    9853

```

If you use ssh-agent, you do not need the `ansible_password` lines. If you use ssh keys, but not ssh-agent, and you have multiple keys, specify the key to use for each connection in the `[group:vars]` section with `ansible_ssh_private_key_file=/path/to/correct/key`. For more information on `ansible_ssh_` options see [主机连接: Inventory 参数设置](#).

警告: Never store passwords in plain text.

Ansible vault for password encryption

The “Vault” feature of Ansible allows you to keep sensitive data such as passwords or keys in encrypted files, rather than as plain text in your playbooks or roles. These vault files can then be distributed or placed in source control. See *Using Vault in playbooks* for more information.

Common inventory variables

The following variables are common for all platforms in the inventory, though they can be overwritten for a particular inventory group or host.

ansible_connection Ansible uses the `ansible_connection` setting to determine how to connect to a remote device. When working with Ansible Networking, set this to `network_cli` so Ansible treats the remote node as a network device with a limited execution environment. Without this setting, Ansible would attempt to use `ssh` to connect to the remote and execute the Python script on the network device, which would fail because Python generally isn't available on network devices.

ansible_network_os Informs Ansible which Network platform this hosts corresponds to. This is required when using `network_cli` or `netconf`.

ansible_user The user to connect to the remote device (switch) as. Without this the user that is running `ansible-playbook` would be used. Specifies which user on the network device the connection

ansible_password The corresponding password for **ansible_user** to log in as. If not specified SSH key will be used.

ansible_become If enable mode (privilege mode) should be used, see the next section.

ansible_become_method Which type of *become* should be used, for `network_cli` the only valid choice is `enable`.

Privilege escalation

Certain network platforms, such as Arista EOS and Cisco IOS, have the concept of different privilege modes. Certain network modules, such as those that modify system state including users, will only work in high privilege states. Ansible supports `become` when using `connection: network_cli`. This allows privileges to be raised for the specific tasks that need them. Adding `become: yes` and `become_method: enable` informs Ansible to go into privilege mode before executing the task, as shown here:

```
[eos:vars]
ansible_connection=network_cli
ansible_network_os=eos
ansible_become=yes
ansible_become_method=enable
```

For more information, see the *using become with network modules* guide.

Jump hosts

If the Ansible Controller doesn't have a direct route to the remote device and you need to use a Jump Host, please see the *Ansible Network Proxy Command* guide for details on how to achieve this.

Example 1: collecting facts and creating backup files with a playbook

Ansible facts modules gather system information ‘facts’ that are available to the rest of your playbook.

Ansible Networking ships with a number of network-specific facts modules. In this example, we use the `_facts` modules `eos_facts`, `ios_facts` and `vyos_facts` to connect to the remote networking device. As the credentials are not explicitly passed via module arguments, Ansible uses the username and password from the inventory file.

Ansible's “Network Fact modules” gather information from the system and store the results in facts prefixed with `ansible_net_`. The data collected by these modules is documented in the *Return Values* section of the module docs, in this case `eos_facts` and `vyos_facts`. We can use the facts, such as `ansible_net_version` late on in the “Display some facts” task.

To ensure we call the correct mode (`*_facts`) the task is conditionally run based on the group defined in the inventory file, for more information on the use of conditionals in Ansible Playbooks see *The When Statement*.

In this example, we will create an inventory file containing some network switches, then run a playbook to connect to the network devices and return some information about them.

Step 1: Creating the inventory

First, create a file called `inventory`, containing:

```
[switches:children]
eos
ios
vyos
```

(下页继续)

(续上页)

```
[eos]
eos01.example.net

[ios]
ios01.example.net

[vynos]
vynos01.example.net
```

Step 2: Creating the playbook

Next, create a playbook file called `facts-demo.yml` containing the following:

```
- name: "Demonstrate connecting to switches"
  hosts: switches
  gather_facts: no

  tasks:
    ###
    # Collect data
    #
    - name: Gather facts (eos)
      eos_facts:
      when: ansible_network_os == 'eos'

    - name: Gather facts (ops)
      ios_facts:
      when: ansible_network_os == 'ios'

    - name: Gather facts (vynos)
      vyos_facts:
      when: ansible_network_os == 'vyos'

    ###
    # Demonstrate variables
    #
    - name: Display some facts
      debug:
        msg: "The hostname is {{ ansible_net_hostname }} and the OS is {{ ansible_net_
version }}"
```

(下页继续)

(续上页)

```

- name: Facts from a specific host
  debug:
    var: hostvars['vyos01.example.net']

- name: Write facts to disk using a template
  copy:
    content: |
      #jinja2: lstrip_blocks: True
      EOS device info:
        {% for host in groups['eos'] %}
        Hostname: {{ hostvars[host].ansible_net_hostname }}
        Version: {{ hostvars[host].ansible_net_version }}
        Model: {{ hostvars[host].ansible_net_model }}
        Serial: {{ hostvars[host].ansible_net_serialnum }}
        {% endfor %}

      IOS device info:
        {% for host in groups['ios'] %}
        Hostname: {{ hostvars[host].ansible_net_hostname }}
        Version: {{ hostvars[host].ansible_net_version }}
        Model: {{ hostvars[host].ansible_net_model }}
        Serial: {{ hostvars[host].ansible_net_serialnum }}
        {% endfor %}

      VyOS device info:
        {% for host in groups['vyos'] %}
        Hostname: {{ hostvars[host].ansible_net_hostname }}
        Version: {{ hostvars[host].ansible_net_version }}
        Model: {{ hostvars[host].ansible_net_model }}
        Serial: {{ hostvars[host].ansible_net_serialnum }}
        {% endfor %}

    dest: /tmp/switch-facts
    run_once: yes

    ###
    # Get running configuration
    #

- name: Backup switch (eos)

```

(下页继续)

(续上页)

```

eos_config:
  backup: yes
  register: backup_eos_location
  when: ansible_network_os == 'eos'

- name: backup switch (vyos)
  vyos_config:
    backup: yes
    register: backup_vyos_location
    when: ansible_network_os == 'vyos'

- name: Create backup dir
  file:
    path: "/tmp/backups/{{ inventory_hostname }}"
    state: directory
    recurse: yes

- name: Copy backup files into /tmp/backups/ (eos)
  copy:
    src: "{{ backup_eos_location.backup_path }}"
    dest: "/tmp/backups/{{ inventory_hostname }}/{{ inventory_hostname }}.bck"
    when: ansible_network_os == 'eos'

- name: Copy backup files into /tmp/backups/ (vyos)
  copy:
    src: "{{ backup_vyos_location.backup_path }}"
    dest: "/tmp/backups/{{ inventory_hostname }}/{{ inventory_hostname }}.bck"
    when: ansible_network_os == 'vyos'

```

Step 3: Running the playbook

To run the playbook, run the following from a console prompt:

```
ansible-playbook -i inventory facts-demo.yml
```

This should return output similar to the following:

```

PLAY RECAP
eos01.example.net      : ok=7    changed=2    unreachable=0    failed=0
ios01.example.net      : ok=7    changed=2    unreachable=0    failed=0

```

(下页继续)

(续上页)

```
vyos01.example.net      : ok=6    changed=2    unreachable=0    failed=0
```

Step 4: Examining the playbook results

Next, look at the contents of the file we created containing the switch facts:

```
cat /tmp/switch-facts
```

You can also look at the backup files:

```
find /tmp/backups
```

If *ansible-playbook* fails, please follow the debug steps in *Network Debug and Troubleshooting Guide*.

Example 2: simplifying playbooks with network agnostic modules

(This example originally appeared in the [Deep Dive on cli_command](#) for [Network Automation](#) blog post by Sean Cavanaugh -@IPvSean).

If you have two or more network platforms in your environment, you can use the network agnostic modules to simplify your playbooks. You can use network agnostic modules such as `cli_command` or `cli_config` in place of the platform-specific modules such as `eos_config`, `ios_config`, and `junos_config`. This reduces the number of tasks and conditionals you need in your playbooks.

注解: Network agnostic modules require the `network_cli` connection plugin.

Sample playbook with platform-specific modules

This example assumes three platforms, Arista EOS, Cisco NXOS, and Juniper JunOS. Without the network agnostic modules, a sample playbook might contain the following three tasks with platform-specific commands:

```
---
- name: Run Arista command
  eos_command:
    commands: show ip int br
  when: ansible_network_os == 'eos'

- name: Run Cisco NXOS command
```

(下页继续)

(续上页)

```

nxos_command:
  commands: show ip int br
when: ansible_network_os == 'nxos'

- name: Run Vynos command
  vyos_command:
    commands: show interface
  when: ansible_network_os == 'vyos'

```

Simplified playbook with cli_command network agnostic module

You can replace these platform-specific modules with the network agnostic cli_command module as follows:

```

---
- hosts: network
  gather_facts: false
  connection: network_cli

  tasks:
    - name: Run cli_command on Arista and display results
      block:
        - name: Run cli_command on Arista
          cli_command:
            command: show ip int br
            register: result

        - name: Display result to terminal window
          debug:
            var: result.stdout_lines
          when: ansible_network_os == 'eos'

    - name: Run cli_command on Cisco IOS and display results
      block:
        - name: Run cli_command on Cisco IOS
          cli_command:
            command: show ip int br
            register: result

        - name: Display result to terminal window

```

(下页继续)

(续上页)

```
    debug:
      var: result.stdout_lines
  when: ansible_network_os == 'ios'

- name: Run cli_command on Vynos and display results
  block:
    - name: Run cli_command on Vynos
      cli_command:
        command: show interfaces
        register: result

    - name: Display result to terminal window
      debug:
        var: result.stdout_lines
  when: ansible_network_os == 'vyos'
```

If you use groups and group_vars by platform type, this playbook can be further simplified to :

```
---
- name: Run command and print to terminal window
  hosts: routers
  gather_facts: false

  tasks:
    - name: Run show command
      cli_command:
        command: "{{show_interfaces}}"
        register: command_output
```

You can see a full example of this using group_vars and also a configuration backup example at [Network agnostic examples](#).

Using multiple prompts with the cli_command

The cli_command also supports multiple prompts.

```
---
- name: Change password to default
  cli_command:
    command: "{{ item }}"
```

(下页继续)

(续上页)

```
prompt:
  - "New password"
  - "Retype new password"
answer:
  - "mypassword123"
  - "mypassword123"
check_all: True
loop:
  - "configure"
  - "rollback"
  - "set system root-authentication plain-text-password"
  - "commit"
```

See the `cli_command` for full documentation on this command.

Implementation Notes

Demo variables

Although these tasks are not needed to write data to disk, they are used in this example to demonstrate some methods of accessing facts about the given devices or a named host.

Ansible `hostvars` allows you to access variables from a named host. Without this we would return the details for the current host, rather than the named host.

For more information, see *Accessing information about other hosts with magic variables*.

Get running configuration

The `eos_config` and `vyos_config` modules have a `backup:` option that when set will cause the module to create a full backup of the current `running-config` from the remote device before any changes are made. The backup file is written to the `backup` folder in the playbook root directory. If the directory does not exist, it is created.

To demonstrate how we can move the backup file to a different location, we register the result and move the file to the path stored in `backup_path`.

Note that when using variables from tasks in this way we use double quotes (") and double curly-brackets ({{...}}) to tell Ansible that this is a variable.

Troubleshooting

If you receive an connection error please double check the inventory and playbook for typos or missing lines. If the issue still occurs follow the debug steps in *Network Debug and Troubleshooting Guide*.

参见:

- `network_guide`
- *Inventory 使用进阶*
- *Vault best practices*

1.10.4 Network Debug and Troubleshooting Guide

- *Introduction*
- *How to troubleshoot*
 - *Enabling Networking logging and how to read the logfile*
 - *Enabling Networking device interaction logging*
 - *Isolating an error*
- *Troubleshooting socket path issues*
- *Category “Unable to open shell”*
 - *Error: “[Errno -2] Name or service not known”*
 - *Error: “Authentication failed”*
 - *Error: “connecting to host <hostname> returned an error” or “Bad address”*
 - *Error: “No authentication methods available”*
 - *Clearing Out Persistent Connections*
- *Timeout issues*
 - *Persistent connection idle timeout*
 - *Command timeout*
 - *Persistent connection retry timeout*
 - *Timeout issue due to platform specific login menu with `network_cli` connection type*
- *Playbook issues*
 - *Error: “Unable to enter configuration mode”*
- *Proxy Issues*

- *delegate_to vs ProxyCommand*
- *Using bastion/jump host with netconf connection*
- *Enabling jump host setting*
- *Example ssh config file (~/.ssh/config)*
- *Miscellaneous Issues*
 - *Intermittent failure while using `network_cli` connection type*
 - *Task failure due to mismatched error regex within command response using `network_cli` connection type*
 - *Intermittent failure while using `network_cli` connection type due to slower network or remote target host*

Introduction

Starting with Ansible version 2.1, you can now use the familiar Ansible models of playbook authoring and module development to manage heterogeneous networking devices. Ansible supports a growing number of network devices using both CLI over SSH and API (when available) transports.

This section discusses how to debug and troubleshoot network modules in Ansible 2.3.

How to troubleshoot

This section covers troubleshooting issues with Network Modules.

Errors generally fall into one of the following categories:

Authentication issues

- Not correctly specifying credentials
- Remote device (network switch/router) not falling back to other authentication methods
- SSH key issues

Timeout issues

- Can occur when trying to pull a large amount of data
- May actually be masking a authentication issue

Playbook issues

- Use of `delegate_to`, instead of `ProxyCommand`. See *network proxy guide* for more information.

警告: unable to open shell

The `unable to open shell` message is new in Ansible 2.3, it means that the `ansible-connection` daemon has not been able to successfully talk to the remote network device. This generally means that there is an authentication issue. See the “Authentication and connection issues” section in this document for more information.

Enabling Networking logging and how to read the logfile

Platforms: Any

Ansible 2.3 features improved logging to help diagnose and troubleshoot issues regarding Ansible Networking modules.

Because logging is very verbose it is disabled by default. It can be enabled via the `ANSIBLE_LOG_PATH` and `ANSIBLE_DEBUG` options on the ansible-controller, that is the machine running ansible-playbook.

Before running `ansible-playbook` run the following commands to enable logging:

```
# Specify the location for the log file
export ANSIBLE_LOG_PATH=~/.ansible.log
# Enable Debug
export ANSIBLE_DEBUG=True

# Run with 4*v for connection level verbosity
ansible-playbook -vvvv ...
```

After Ansible has finished running you can inspect the log file which has been created on the ansible-controller:

```
less $ANSIBLE_LOG_PATH

2017-03-30 13:19:52,740 p=28990 u=fred | creating new control socket for host veos01:22
↳as user admin
2017-03-30 13:19:52,741 p=28990 u=fred | control socket path is /home/fred/.ansible/pc/
↳ca5960d27a
2017-03-30 13:19:52,741 p=28990 u=fred | current working directory is /home/fred/
↳ansible/test/integration
2017-03-30 13:19:52,741 p=28990 u=fred | using connection plugin network_cli
...
2017-03-30 13:20:14,771 paramiko.transport userauth is OK
2017-03-30 13:20:15,283 paramiko.transport Authentication (keyboard-interactive)
↳successful!
```

(下页继续)

(续上页)

```
2017-03-30 13:20:15,302 p=28990 u=fred | ssh connection done, setting terminal
2017-03-30 13:20:15,321 p=28990 u=fred | ssh connection has completed successfully
2017-03-30 13:20:15,322 p=28990 u=fred | connection established to veos01 in 0:00:22.
↪580626
```

From the log notice:

- `p=28990` Is the PID (Process ID) of the `ansible-connection` process
- `u=fred` Is the user *running* ansible, not the remote-user you are attempting to connect as
- `creating new control socket for host veos01:22 as user admin host:port as user`
- `control socket path` is location on disk where the persistent connection socket is created
- `using connection plugin network_cli` Informs you that persistent connection is being used
- `connection established to veos01 in 0:00:22.580626` Time taken to obtain a shell on the remote device

Because the log files are verbose, you can use `grep` to look for specific information. For example, once you have identified the `pid` from the `creating new control socket for host` line you can search for other connection log entries:

```
grep "p=28990" $ANSIBLE_LOG_PATH
```

Enabling Networking device interaction logging

Platforms: Any

Ansible 2.8 features added logging of device interaction in log file to help diagnose and troubleshoot issues regarding Ansible Networking modules. The messages are logged in file pointed by `log_path` configuration option in Ansible configuration file or by set `ANSIBLE_LOG_PATH` as mentioned in above section.

警告: The device interaction messages consist of command executed on target device and the returned response, as this log data can contain sensitive information including passwords in plain text it is disabled by default. Additionally, in order to prevent accidental leakage of data, a warning will be shown on every task with this setting enabled specifying which host has it enabled and where the data is being logged.

Be sure to fully understand the security implications of enabling this option. The device interaction logging can be enabled either globally by setting in configuration file or by setting environment or enabled on per task basis by passing special variable to task.

Before running `ansible-playbook` run the following commands to enable logging:

```
# Specify the location for the log file
export ANSIBLE_LOG_PATH=~/.ansible.log
```

Enable device interaction logging for a given task

```
- name: get version information
  ios_command:
    commands:
      - show version
  vars:
    ansible_persistent_log_messages: True
```

To make this a global setting, add the following to your `ansible.cfg` file:

```
[persistent_connection]
log_messages = True
```

or enable environment variable `ANSIBLE_PERSISTENT_LOG_MESSAGES`

```
# Enable device interaction logging export ANSIBLE_PERSISTENT_LOG_MESSAGES=True
```

If the task is failing at the time on connection initialization itself it is recommended to enable this option globally else if an individual task is failing intermittently this option can be enabled for that task itself to find the root cause.

After Ansible has finished running you can inspect the log file which has been created on the ansible-controller

注解: Be sure to fully understand the security implications of enabling this option as it can log sensitive information in log file thus creating security vulnerability.

Isolating an error

Platforms: Any

As with any effort to troubleshoot it's important to simplify the test case as much as possible.

For Ansible this can be done by ensuring you are only running against one remote device:

- Using `ansible-playbook --limit switch1.example.net...`
- Using an ad-hoc `ansible` command

ad-hoc refers to running Ansible to perform some quick command using `/usr/bin/ansible`, rather than the orchestration language, which is `/usr/bin/ansible-playbook`. In this case we can ensure connectivity by attempting to execute a single command on the remote device:


```
ansible -m eos_command -a 'commands=?' -i inventory switch1.example.net -e 'ansible_
↪connection=local' -u admin -k
```

In the above example, we:

- connect to `switch1.example.net` specified in the inventory file `inventory`
- use the module `eos_command`
- run the command ?
- connect using the username `admin`
- inform ansible to prompt for the ssh password by specifying `-k`

If you have SSH keys configured correctly, you don't need to specify the `-k` parameter

If the connection still fails you can combine it with the `enable_network_logging` parameter. For example:

```
# Specify the location for the log file
export ANSIBLE_LOG_PATH=~/.ansible.log
# Enable Debug
export ANSIBLE_DEBUG=True
# Run with 4*v for connection level verbosity
ansible -m eos_command -a 'commands=?' -i inventory switch1.example.net -e 'ansible_
↪connection=local' -u admin -k
```

Then review the log file and find the relevant error message in the rest of this document.

Troubleshooting socket path issues

Platforms: Any

The Socket path does not exist or cannot be found and Unable to connect to socket messages are new in Ansible 2.5. These messages indicate that the socket used to communicate with the remote network device is unavailable or does not exist.

For example:

```
fatal: [spine02]: FAILED! => {
  "changed": false,
  "failed": true,
  "module_stderr": "Traceback (most recent call last):\n  File \"/tmp/ansible_TSqk5J/
↪ansible_modlib.zip/ansible/module_utils/connection.py", line 115, in _exec_
↪jsonrpc\nansible.module_utils.connection.ConnectionError: Socket path XX does not
↪exist or cannot be found. See Troubleshooting socket path issues in the Network Debug
↪and Troubleshooting Guide\n",
```

(下页继续)

(续上页)

```

    "module_stdout": "",
    "msg": "MODULE FAILURE",
    "rc": 1
}

```

or

```

fatal: [spine02]: FAILED! => {
  "changed": false,
  "failed": true,
  "module_stderr": "Traceback (most recent call last):\n  File \"/tmp/ansible_TSqk5J/
↪ansible_modlib.zip/ansible/module_utils/connection.py", line 123, in _exec_
↪jsonrpc\nansible.module_utils.connection.ConnectionError: Unable to connect to socket_
↪XX. See Troubleshooting socket path issues in Network Debug and Troubleshooting Guide\n
↪",
  "module_stdout": "",
  "msg": "MODULE FAILURE",
  "rc": 1
}

```

Suggestions to resolve:

1. Verify that you have write access to the socket path described in the error message.
2. Follow the steps detailed in [enable network logging](#).

If the identified error message from the log file is:

```

2017-04-04 12:19:05,670 p=18591 u=fred | command timeout triggered, timeout value is 30_
↪secs

```

or

```

2017-04-04 12:19:05,670 p=18591 u=fred | persistent connection idle timeout triggered,_
↪timeout value is 30 secs

```

Follow the steps detailed in [timeout issues](#)

Category “Unable to open shell”

Platforms: Any

The `unable to open shell` message is new in Ansible 2.3. This message means that the `ansible-connection` daemon has not been able to successfully talk to the remote network device. This

generally means that there is an authentication issue. It is a “catch all” message, meaning you need to enable `:ref:logging'a_note_about_logging'` to find the underlying issues.

For example:

```
TASK [prepare_eos_tests : enable cli on remote device]
↳*****
fatal: [veos01]: FAILED! => {"changed": false, "failed": true, "msg": "unable to open
↳shell"}
```

or:

```
TASK [ios_system : configure name_servers]
↳*****
task path:
fatal: [ios-csr1000v]: FAILED! => {
    "changed": false,
    "failed": true,
    "msg": "unable to open shell",
}
```

Suggestions to resolve:

Follow the steps detailed in [enable_network_logging](#).

Once you’ve identified the error message from the log file, the specific solution can be found in the rest of this document.

Error: “[Errno -2] Name or service not known”

Platforms: Any

Indicates that the remote host you are trying to connect to can not be reached

For example:

```
2017-04-04 11:39:48,147 p=15299 u=fred | control socket path is /home/fred/.ansible/pc/
↳ca5960d27a
2017-04-04 11:39:48,147 p=15299 u=fred | current working directory is /home/fred/git/
↳ansible-inc/stable-2.3/test/integration
2017-04-04 11:39:48,147 p=15299 u=fred | using connection plugin network_cli
2017-04-04 11:39:48,340 p=15299 u=fred | connecting to host veos01 returned an error
2017-04-04 11:39:48,340 p=15299 u=fred | [Errno -2] Name or service not known
```

Suggestions to resolve:

- If you are using the `provider:` options ensure that its suboption `host:` is set correctly.
- If you are not using `provider:` nor top-level arguments ensure your inventory file is correct.

Error: “Authentication failed”**Platforms:** Any

Occurs if the credentials (username, passwords, or ssh keys) passed to `ansible-connection` (via `ansible` or `ansible-playbook`) can not be used to connect to the remote device.

For example:

```
<ios01> ESTABLISH CONNECTION FOR USER: cisco on PORT 22 TO ios01
<ios01> Authentication failed.
```

Suggestions to resolve:

If you are specifying credentials via `password:` (either directly or via `provider:`) or the environment variable `ANSIBLE_NET_PASSWORD` it is possible that `paramiko` (the Python SSH library that Ansible uses) is using ssh keys, and therefore the credentials you are specifying are being ignored. To find out if this is the case, disable “look for keys”. This can be done like this:

```
export ANSIBLE_PARAMIKO_LOOK_FOR_KEYS=False
```

To make this a permanent change, add the following to your `ansible.cfg` file:

```
[paramiko_connection]
look_for_keys = False
```

Error: “connecting to host <hostname> returned an error” or “Bad address”

This may occur if the SSH fingerprint hasn't been added to Paramiko's (the Python SSH library) known hosts file.

When using persistent connections with Paramiko, the connection runs in a background process. If the host doesn't already have a valid SSH key, by default Ansible will prompt to add the host key. This will cause connections running in background processes to fail.

For example:

```
2017-04-04 12:06:03,486 p=17981 u=fred | using connection plugin network_cli
2017-04-04 12:06:04,680 p=17981 u=fred | connecting to host veos01 returned an error
2017-04-04 12:06:04,682 p=17981 u=fred | (14, 'Bad address')
```

(下页继续)

(续上页)

```
2017-04-04 12:06:33,519 p=17981 u=fred | number of connection attempts exceeded, unable
↳to connect to control socket
2017-04-04 12:06:33,520 p=17981 u=fred | persistent_connect_interval=1, persistent_
↳connect_retries=30
```

Suggestions to resolve:

Use `ssh-keyscan` to pre-populate the `known_hosts`. You need to ensure the keys are correct.

```
ssh-keyscan veos01
```

or

You can tell Ansible to automatically accept the keys

Environment variable method:

```
export ANSIBLE_PARAMIKO_HOST_KEY_AUTO_ADD=True
ansible-playbook ...
```

`ansible.cfg` method:

`ansible.cfg`

```
[paramiko_connection]
host_key_auto_add = True
```

Error: “No authentication methods available”

For example:

```
2017-04-04 12:19:05,670 p=18591 u=fred | creating new control socket for host
↳veos01:None as user admin
2017-04-04 12:19:05,670 p=18591 u=fred | control socket path is /home/fred/.ansible/pc/
↳ca5960d27a
2017-04-04 12:19:05,670 p=18591 u=fred | current working directory is /home/fred/git/
↳ansible-inc/ansible-workspace-2/test/integration
2017-04-04 12:19:05,670 p=18591 u=fred | using connection plugin network_cli
2017-04-04 12:19:06,606 p=18591 u=fred | connecting to host veos01 returned an error
2017-04-04 12:19:06,606 p=18591 u=fred | No authentication methods available
2017-04-04 12:19:35,708 p=18591 u=fred | connect retry timeout expired, unable to
↳connect to control socket
2017-04-04 12:19:35,709 p=18591 u=fred | persistent_connect_retry_timeout is 15 secs
```

Suggestions to resolve:

No password or SSH key supplied

Clearing Out Persistent Connections

Platforms: Any

In Ansible 2.3, persistent connection sockets are stored in `~/.ansible/pc` for all network devices. When an Ansible playbook runs, the persistent socket connection is displayed when verbose output is specified.

```
<switch> socket_path: /home/fred/.ansible/pc/f64ddfa760
```

To clear out a persistent connection before it times out (the default timeout is 30 seconds of inactivity), simply delete the socket file.

Timeout issues

Persistent connection idle timeout

By default, `ANSIBLE_PERSISTENT_CONNECT_TIMEOUT` is set to 30 (seconds). You may see the following error if this value is too low:

```
2017-04-04 12:19:05,670 p=18591 u=fred | persistent connection idle timeout triggered,↵
↵timeout value is 30 secs
```

Suggestions to resolve:

Increase value of persistent connection idle timeout:

```
export ANSIBLE_PERSISTENT_CONNECT_TIMEOUT=60
```

To make this a permanent change, add the following to your `ansible.cfg` file:

```
[persistent_connection]
connect_timeout = 60
```

Command timeout

By default, `ANSIBLE_PERSISTENT_COMMAND_TIMEOUT` is set to 30 (seconds). Prior versions of Ansible had this value set to 10 seconds by default. You may see the following error if this value is too low:

```
2017-04-04 12:19:05,670 p=18591 u=fred | command timeout triggered, timeout value is 30↵
↵secs
```

Suggestions to resolve:

- Option 1 (Global command timeout setting): Increase value of command timeout in configuration file or by setting environment variable.

```
export ANSIBLE_PERSISTENT_COMMAND_TIMEOUT=60
```

To make this a permanent change, add the following to your `ansible.cfg` file:

```
[persistent_connection]
command_timeout = 60
```

- Option 2 (Per task command timeout setting): Increase command timeout per task basis. All network modules support a timeout value that can be set on a per task basis. The timeout value controls the amount of time in seconds before the task will fail if the command has not returned.

For local connection type:

Suggestions to resolve:

```
- name: save running-config
  ios_command:
    commands: copy running-config startup-config
    provider: "{{ cli }}"
    timeout: 30
```

For `network_cli`, `netconf` connection type (applicable from 2.7 onwards):

Suggestions to resolve:

```
- name: save running-config
  ios_command:
    commands: copy running-config startup-config
  vars:
    ansible_command_timeout: 60
```

Some operations take longer than the default 30 seconds to complete. One good example is saving the current running config on IOS devices to startup config. In this case, changing the timeout value from the default 30 seconds to 60 seconds will prevent the task from failing before the command completes successfully.

Persistent connection retry timeout

By default, `ANSIBLE_PERSISTENT_CONNECT_RETRY_TIMEOUT` is set to 15 (seconds). You may see the following error if this value is too low:

```
2017-04-04 12:19:35,708 p=18591 u=fred | connect retry timeout expired, unable to
↪connect to control socket
2017-04-04 12:19:35,709 p=18591 u=fred | persistent_connect_retry_timeout is 15 secs
```

Suggestions to resolve:

Increase the value of the persistent connection idle timeout. Note: This value should be greater than the SSH timeout value (the timeout value under the defaults section in the configuration file) and less than the value of the persistent connection idle timeout (connect_timeout).

```
export ANSIBLE_PERSISTENT_CONNECT_RETRY_TIMEOUT=30
```

To make this a permanent change, add the following to your `ansible.cfg` file:

```
[persistent_connection]
connect_retry_timeout = 30
```

Timeout issue due to platform specific login menu with `network_cli` connection type

In Ansible 2.9 and later, the `network_cli` connection plugin configuration options are added to handle the platform specific login menu. These options can be set as group/host or tasks variables.

Example: Handle single login menu prompts with host variables

```
$cat host_vars/<hostname>.yaml
---
ansible_terminal_initial_prompt:
  - "Connect to a host"
ansible_terminal_initial_answer:
  - "3"
```

Example: Handle remote host multiple login menu prompts with host variables

```
$cat host_vars/<inventory-hostname>.yaml
---
ansible_terminal_initial_prompt:
  - "Press any key to enter main menu"
  - "Connect to a host"
ansible_terminal_initial_answer:
  - "\\r"
  - "3"
ansible_terminal_initial_prompt_checkall: True
```


To handle multiple login menu prompts:

- The values of `ansible_terminal_initial_prompt` and `ansible_terminal_initial_answer` should be a list.
- The prompt sequence should match the answer sequence.
- The value of `ansible_terminal_initial_prompt_checkall` should be set to `True`.

注解: If all the prompts in sequence are not received from remote host at the time connection initialization it will result in a timeout.

Playbook issues

This section details issues are caused by issues with the Playbook itself.

Error: “Unable to enter configuration mode”

Platforms: eos and ios

This occurs when you attempt to run a task that requires privileged mode in a user mode shell.

For example:

```
TASK [ios_system : configure name_servers]
↳*****
task path:
fatal: [ios-csr1000v]: FAILED! => {
  "changed": false,
  "failed": true,
  "msg": "unable to enter configuration mode",
}
```

Suggestions to resolve:

In Ansible prior to 2.5 : Add `authorize: yes` to the task. For example:

```
- name: configure hostname
  ios_system:
    provider:
      hostname: foo
      authorize: yes
  register: result
```

If the user requires a password to go into privileged mode, this can be specified with `auth_pass`; if `auth_pass` isn't set, the environment variable `ANSIBLE_NET_AUTHORIZE` will be used instead.

Add `authorize: yes` to the task. For example:

```
- name: configure hostname
  ios_system:
  provider:
    hostname: foo
    authorize: yes
    auth_pass: "{{ mypasswordvar }}"
  register: result
```

注解: Starting with Ansible 2.5 we recommend using `connection: network_cli` and `become: yes`

Proxy Issues

delegate_to vs ProxyCommand

The new connection framework for Network Modules in Ansible 2.3 that uses `cli` transport no longer supports the use of the `delegate_to` directive. In order to use a bastion or intermediate jump host to connect to network devices over `cli` transport, network modules now support the use of `ProxyCommand`.

To use `ProxyCommand`, configure the proxy settings in the Ansible inventory file to specify the proxy host.

```
[nxos]
nxos01
nxos02

[nxos:vars]
ansible_ssh_common_args='-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

With the configuration above, simply build and run the playbook as normal with no additional changes necessary. The network module will now connect to the network device by first connecting to the host specified in `ansible_ssh_common_args`, which is `bastion01` in the above example.

You can also set the proxy target for all hosts by using environment variables.

```
export ANSIBLE_SSH_ARGS='-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

Using bastion/jump host with netconf connection

Enabling jump host setting

Bastion/jump host with netconf connection can be enabled by:

- Setting Ansible variable `ansible_netconf_ssh_config` either to `True` or custom ssh config file path
- Setting environment variable `ANSIBLE_NETCONF_SSH_CONFIG` to `True` or custom ssh config file path
- Setting `ssh_config = 1` or `ssh_config = <ssh-file-path>` under `netconf_connection` section

If the configuration variable is set to 1 the proxycommand and other ssh variables are read from default ssh config file (`~/.ssh/config`).

If the configuration variable is set to file path the proxycommand and other ssh variables are read from the given custom ssh file path

Example ssh config file (`~/.ssh/config`)

```
Host jumphost
  HostName jumphost.domain.name.com
  User jumphost-user
  IdentityFile "/path/to/ssh-key.pem"
  Port 22

# Note: Due to the way that Paramiko reads the SSH Config file,
# you need to specify the NETCONF port that the host uses.
# i.e. It does not automatically use ansible_port
# As a result you need either:

Host junos01
  HostName junos01
  ProxyCommand ssh -W %h:22 jumphost

# OR

Host junos01
  HostName junos01
  ProxyCommand ssh -W %h:830 jumphost
```

(下页继续)

(续上页)

```
# Depending on the netconf port used.
```

Example Ansible inventory file

```
[junos]
junos01

[junos:vars]
ansible_connection=netconf
ansible_network_os=junos
ansible_user=myuser
ansible_password=!vault...
```

注解: Using ProxyCommand with passwords via variables

By design, SSH doesn't support providing passwords via environment variables. This is done to prevent secrets from leaking out, for example in `ps` output.

We recommend using SSH Keys, and if needed an ssh-agent, rather than passwords, where ever possible.

Miscellaneous Issues

Intermittent failure while using `network_cli` connection type

If the command prompt received in response is not matched correctly within the `network_cli` connection plugin the task might fail intermittently with truncated response or with the error message `operation requires privilege escalation`. Starting in 2.7.1 a new buffer read timer is added to ensure prompts are matched properly and a complete response is send in output. The timer default value is 0.2 seconds and can be adjusted on a per task basis or can be set globally in seconds.

Example Per task timer setting

```
- name: gather ios facts
  ios_facts:
    gather_subset: all
    register: result
  vars:
    ansible_buffer_read_timeout: 2
```

To make this a global setting, add the following to your `ansible.cfg` file:

```
[persistent_connection]
buffer_read_timeout = 2
```

This timer delay per command executed on remote host can be disabled by setting the value to zero.

Task failure due to mismatched error regex within command response using `network_cli` connection type

In Ansible 2.9 and later, the `network_cli` connection plugin configuration options are added to handle the stdout and stderr regex to identify if the command execution response consist of a normal response or an error response. These options can be set group/host variables or as tasks variables.

Example: For mismatched error response

```
- name: fetch logs from remote host
  ios_command:
    commands:
      - show logging
```

Playbook run output:

```
TASK [first fetch logs] *****
fatal: [ios01]: FAILED! => {
  "changed": false,
  "msg": "RF Name:\r\n\r\n <--nsip-->
        \"IPSEC-3-REPLAY_ERROR: Test log\"\\r\\n*Aug  1 08:36:18.483: %SYS-7-USERLOG_
↪DEBUG:
        Message from tty578(user id: ansible): test\\r\\nan-ios-02#\"}
```

Suggestions to resolve:

Modify the error regex for individual task.

```
- name: fetch logs from remote host
  ios_command:
    commands:
      - show logging
  vars:
    ansible_terminal_stderr_re:
      - pattern: 'connection timed out'
      flags: 're.I'
```

The terminal plugin regex options `ansible_terminal_stderr_re` and `ansible_terminal_stdout_re` have

`pattern` and `flags` as keys. The value of the `flags` key should be a value that is accepted by the `re.compile` python method.

Intermittent failure while using `network_cli` connection type due to slower network or remote target host

In Ansible 2.9 and later, the `network_cli` connection plugin configuration option is added to control the number of attempts to connect to a remote host. The default number of attempts is three. After every retry attempt the delay between retries is increased by power of 2 in seconds until either the maximum attempts are exhausted or either the `persistent_command_timeout` or `persistent_connect_timeout` timers are triggered.

To make this a global setting, add the following to your `ansible.cfg` file:

```
[persistent_connection]
network_cli_retries = 5
```

1.10.5 Working with command output and prompts in network modules

- *Conditionals in networking modules*
- *Handling prompts in network modules*

Conditionals in networking modules

Ansible allows you to use conditionals to control the flow of your playbooks. Ansible networking command modules use the following unique conditional statements.

- `eq` - Equal
- `neq` - Not equal
- `gt` - Greater than
- `ge` - Greater than or equal
- `lt` - Less than
- `le` - Less than or equal
- `contains` - Object contains specified item

Conditional statements evaluate the results from the commands that are executed remotely on the device. Once the task executes the command set, the `wait_for` argument can be used to evaluate the results before returning control to the Ansible playbook.

For example:

```
---
- name: wait for interface to be admin enabled
  eos_command:
    commands:
      - show interface Ethernet4 | json
    wait_for:
      - "result[0].interfaces.Ethernet4.interfaceStatus eq connected"
```

In the above example task, the command `show interface Ethernet4 | json` is executed on the remote device and the results are evaluated. If the path `(result[0].interfaces.Ethernet4.interfaceStatus)` is not equal to “connected”, then the command is retried. This process continues until either the condition is satisfied or the number of retries has expired (by default, this is 10 retries at 1 second intervals).

The `commands` module can also evaluate more than one set of command results in an interface. For instance:

```
---
- name: wait for interfaces to be admin enabled
  eos_command:
    commands:
      - show interface Ethernet4 | json
      - show interface Ethernet5 | json
    wait_for:
      - "result[0].interfaces.Ethernet4.interfaceStatus eq connected"
      - "result[1].interfaces.Ethernet5.interfaceStatus eq connected"
```

In the above example, two commands are executed on the remote device, and the results are evaluated. By specifying the result index value (0 or 1), the correct result output is checked against the conditional.

The `wait_for` argument must always start with `result` and then the command index in `[]`, where 0 is the first command in the `commands` list, 1 is the second command, 2 is the third and so on.

Handling prompts in network modules

Network devices may require that you answer a prompt before performing a change on the device. Individual network modules such as `ios_command` and `nxos_command` can handle this with a `prompt` parameter.

注解: `prompt` is a Python regex. If you add special characters such as `?` in the `prompt` value, the prompt won't match and you will get a timeout. To avoid this, ensure that the `prompt` value is a Python regex that matches the actual device prompt. Any special characters must be handled correctly in the `prompt` regex.

You can also use the `cli_command` to handle multiple prompts.

```
---
- name: multiple prompt, multiple answer (mandatory check for all prompts)
  cli_command:
    command: "copy sftp sftp://user@host//user/test.img"
    check_all: True
    prompt:
      - "Confirm download operation"
      - "Password"
      - "Do you want to change that to the standby image"
    answer:
      - 'y'
      - <password>
      - 'y'
```

You must list the prompt and the answers in the same order (that is, `prompt[0]` is answered by `answer[0]`).

In the above example, `check_all: True` ensures that the task gives the matching answer to each prompt. Without that setting, a task with multiple prompts would give the first answer to every prompt.

In the following example, the second answer would be ignored and `y` would be the answer given to both prompts. That is, this task only works because both answers are identical. Also notice again that `prompt` must be a Python regex, which is why the `?` is escaped in the first prompt.

```
---
- name: reboot ios device
  cli_command:
    command: reload
    prompt:
      - Save\?
      - confirm
    answer:
      - y
      - y
```

参见:

[Rebooting network devices with Ansible](#) Examples using `wait_for`, `wait_for_connection`, and `prompt` for network devices.

[Deep dive on cli_command](#) Detailed overview of how to use the `cli_command`.

1.10.6 Platform Options

Some Ansible Network platforms support multiple connection types, privilege escalation (`enable` mode), or other options. The pages in this section offer standardized guides to understanding available options on each network platform. We welcome contributions from community-maintained platforms to this section.

CloudEngine OS Platform Options

CloudEngine CE OS supports multiple connections. This page offers details on how each connection works in Ansible and how to use it.

Topics

- *CloudEngine OS Platform Options*
 - *Connections Available*
 - *Using CLI in Ansible*
 - * *Example CLI inventory [ce:vars]*
 - * *Example CLI Task*
 - *Using NETCONF in Ansible*
 - * *Enabling NETCONF*
 - * *Example NETCONF inventory [ce:vars]*
 - * *Example NETCONF Task*
 - *Notes*
 - * *Modules work with connection C(network_cli)*
 - * *Modules work with connection C(netconf)*

Connections Available

	CLI	NETCONF
Protocol	SSH	XML over SSH
Credentials	uses SSH keys / SSH-agent if present accepts <code>-u myuser -k</code> if using password	uses SSH keys / SSH-agent if present accepts <code>-u myuser -k</code> if using password
Indirect Access	via a bastion (jump host)	via a bastion (jump host)
Connection Settings	<code>ansible_connection: network_cli</code>	<code>ansible_connection: netconf</code>
Enable Mode (Privilege Escalation)	not supported by ce OS	not supported by ce OS
Returned Data Format	Refer to individual module documentation	Refer to individual module documentation

For legacy playbooks, Ansible still supports `ansible_connection=local` on all CloudEngine modules. We recommend modernizing to use `ansible_connection=netconf` or `ansible_connection=network_cli` as soon as possible.

Using CLI in Ansible

Example CLI inventory [ce:vars]

```
[ce:vars]
ansible_connection=network_cli
ansible_network_os=ce
ansible_user=myuser
ansible_password=!vault...
ansible_ssh_common_args='-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

- If you are using SSH keys (including an ssh-agent) you can remove the `ansible_password` configuration.
- If you are accessing your host directly (not through a bastion/jump host) you can remove the `ansible_ssh_common_args` configuration.
- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the `ProxyCommand` directive. To prevent secrets from leaking out (for example in `ps` output), SSH does not support providing passwords via environment variables.

Example CLI Task

```
- name: Retrieve CE OS version
  ce_command:
    commands: display version
  when: ansible_network_os == 'ce'
```

Using NETCONF in Ansible

Enabling NETCONF

Before you can use NETCONF to connect to a switch, you must:

- install the `ncclient` python package on your control node(s) with `pip install ncclient`
- enable NETCONF on the CloudEngine OS device(s)

To enable NETCONF on a new switch via Ansible, use the `ce_config` module via the CLI connection. Set up your platform-level variables just like in the CLI example above, then run a playbook task like this:

```
- name: Enable NETCONF
  connection: network_cli
  ce_config:
    lines:
      - snetconf server enable
  when: ansible_network_os == 'ce'
```

Once NETCONF is enabled, change your variables to use the NETCONF connection.

Example NETCONF inventory [ce:vars]

```
[ce:vars]
ansible_connection=netconf
ansible_network_os=ce
ansible_user=myuser
ansible_password=!vault |
ansible_ssh_common_args='-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

Example NETCONF Task

```
- name: Create a vlan, id is 50(ce)
  ce_vlan:
    vlan_id: 50
    name: WEB
  when: ansible_network_os == 'ce'
```

Notes

Modules work with connection C(network_cli)

```
ce_acl_interface
ce_command
ce_config
ce_evpn_bgp
ce_evpn_bgp_rr
ce_evpn_global
ce_facts
ce_mlag_interface
ce_mtu
ce_netstream_aging
ce_netstream_export
ce_netstream_global
ce_netstream_template
ce_ntp_auth
ce_rollback
ce_snmp_contact
ce_snmp_location
ce_snmp_traps
ce_startup
ce_stp
ce_vxlan_arp
ce_vxlan_gateway
ce_vxlan_global
```

Modules work with connection C(netconf)

```
ce_aaa_server
ce_aaa_server_host
ce_acl
ce_acl_advance
ce_bfd_global
ce_bfd_session
ce_bfd_view
ce_bgp
ce_bgp_af
ce_bgp_neighbor
ce_bgp_neighbor_af
ce_dldp
ce_dldp_interface
ce_eth_trunk
ce_evpn_bd_vni
ce_file_copy
ce_info_center_debug
ce_info_center_global
ce_info_center_log
ce_info_center_trap
ce_interface
ce_interface_ospf
ce_ip_interface
ce_lacp
ce_link_status
ce_lldp
ce_lldp_interface
ce_mlag_config
ce_netconf
ce_ntp
ce_ospf
ce_ospf_vrf
ce_reboot
ce_sflow
ce_snmp_community
ce_snmp_target_host
ce_snmp_user
ce_static_route
```

(下页继续)

(续上页)

```
ce_static_route_bfd
ce_switchport
ce_vlan
ce_vrf
ce_vrf_af
ce_vrf_interface
ce_vrrp
ce_vxlan_tunnel
ce_vxlan_vap
```

警告: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

参见:

Setting timeout options

CNOS Platform Options

CNOS supports Enable Mode (Privilege Escalation). This page offers details on how to use Enable Mode on CNOS in Ansible.

Topics

- *CNOS Platform Options*
 - *Connections Available*
 - *Using CLI in Ansible*
 - * *Example CLI group_vars/cnos.yml*
 - * *Example CLI Task*

Connections Available

	CLI
Protocol	SSH
Credentials	uses SSH keys / SSH-agent if present accepts <code>-u myuser -k</code> if using password
Indirect Access	via a bastion (jump host)
Connection Settings	<code>ansible_connection: network_cli</code>
Enable Mode (Privilege Escalation)	supported: use <code>ansible_become: yes</code> with <code>ansible_become_method: enable</code> and <code>ansible_become_password:</code>
Returned Data Format	<code>stdout[0]</code> .

For legacy playbooks, CNOS still supports `ansible_connection: local`. We recommend modernizing to use `ansible_connection: network_cli` as soon as possible.

Using CLI in Ansible

Example CLI group_vars/cnos.yml

```
ansible_connection: network_cli
ansible_network_os: cnos
ansible_user: myuser
ansible_password: !vault...
ansible_become: yes
ansible_become_method: enable
ansible_become_password: !vault...
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

- If you are using SSH keys (including an ssh-agent) you can remove the `ansible_password` configuration.
- If you are accessing your host directly (not through a bastion/jump host) you can remove the `ansible_ssh_common_args` configuration.
- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the `ProxyCommand` directive. To prevent secrets from leaking out (for example in `ps` output), SSH does not support providing passwords via environment variables.

Example CLI Task

```
- name: Retrieve CNOS OS version
  cnos_command:
    commands: show version
  when: ansible_network_os == 'cnos'
```

警告: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

参见:

Setting timeout options

Dell OS6 Platform Options

OS6 supports Enable Mode (Privilege Escalation). This page offers details on how to use Enable Mode on OS6 in Ansible.

Topics

- *Dell OS6 Platform Options*
 - *Connections Available*
 - *Using CLI in Ansible*
 - * *Example CLI group_vars/dellos6.yml*
 - * *Example CLI Task*

Connections Available

	CLI
Protocol	SSH
Credentials	uses SSH keys / SSH-agent if present accepts <code>-u myuser -k</code> if using password
Indirect Access	via a bastion (jump host)
Connection Settings	<code>ansible_connection: network_cli</code>
Enable Mode (Privilege Escalation)	supported: use <code>ansible_become: yes</code> with <code>ansible_become_method: enable</code> and <code>ansible_become_password:</code>
Returned Data Format	<code>stdout[0]</code> .

For legacy playbooks, OS6 still supports `ansible_connection: local`. We recommend modernizing to use `ansible_connection: network_cli` as soon as possible.

Using CLI in Ansible

Example CLI group_vars/dellos6.yml

```
ansible_connection: network_cli
ansible_network_os: dellos6
ansible_user: myuser
ansible_password: !vault...
ansible_become: yes
ansible_become_method: enable
ansible_become_password: !vault...
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

- If you are using SSH keys (including an ssh-agent) you can remove the `ansible_password` configuration.
- If you are accessing your host directly (not through a bastion/jump host) you can remove the `ansible_ssh_common_args` configuration.
- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the `ProxyCommand` directive. To prevent secrets from leaking out (for example in `ps` output), SSH does not support providing passwords via environment variables.

Example CLI Task

```
- name: Backup current switch config (delloso6)
  dellos6_config:
    backup: yes
  register: backup_delloso6_location
  when: ansible_network_os == 'delloso6'
```

警告: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

参见:

Setting timeout options

Dell OS9 Platform Options

OS9 supports Enable Mode (Privilege Escalation). This page offers details on how to use Enable Mode on OS9 in Ansible.

Topics

- *Dell OS9 Platform Options*
 - *Connections Available*
 - *Using CLI in Ansible*
 - * *Example CLI group_vars/delloso9.yml*
 - * *Example CLI Task*

Connections Available

	CLI
Protocol	SSH
Credentials	uses SSH keys / SSH-agent if present accepts <code>-u myuser -k</code> if using password
Indirect Access	via a bastion (jump host)
Connection Settings	<code>ansible_connection: network_cli</code>
Enable Mode (Privilege Escalation)	supported: use <code>ansible_become: yes</code> with <code>ansible_become_method: enable</code> and <code>ansible_become_password:</code>
Returned Data Format	<code>stdout[0]</code> .

For legacy playbooks, OS9 still supports `ansible_connection: local`. We recommend modernizing to use `ansible_connection: network_cli` as soon as possible.

Using CLI in Ansible

Example CLI group_vars/dellos9.yml

```
ansible_connection: network_cli
ansible_network_os: dellos9
ansible_user: myuser
ansible_password: !vault...
ansible_become: yes
ansible_become_method: enable
ansible_become_password: !vault...
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

- If you are using SSH keys (including an ssh-agent) you can remove the `ansible_password` configuration.
- If you are accessing your host directly (not through a bastion/jump host) you can remove the `ansible_ssh_common_args` configuration.
- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the `ProxyCommand` directive. To prevent secrets from leaking out (for example in `ps` output), SSH does not support providing passwords via environment variables.

Example CLI Task

```
- name: Backup current switch config (dello9)
  dellos9_config:
    backup: yes
  register: backup_dellos9_location
  when: ansible_network_os == 'dellos9'
```

警告: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

参见:

Setting timeout options

Dell OS10 Platform Options

OS10 supports Enable Mode (Privilege Escalation). This page offers details on how to use Enable Mode on OS10 in Ansible.

Topics

- *Dell OS10 Platform Options*
 - *Connections Available*
 - *Using CLI in Ansible*
 - * *Example CLI group_vars/dellos10.yml*
 - * *Example CLI Task*

Connections Available

	CLI
Protocol	SSH
Credentials	uses SSH keys / SSH-agent if present accepts <code>-u myuser -k</code> if using password
Indirect Access	via a bastion (jump host)
Connection Settings	<code>ansible_connection: network_cli</code>
Enable Mode (Privilege Escalation)	supported: use <code>ansible_become: yes</code> with <code>ansible_become_method: enable</code> and <code>ansible_become_password:</code>
Returned Data Format	<code>stdout[0]</code> .

For legacy playbooks, OS10 still supports `ansible_connection: local`. We recommend modernizing to use `ansible_connection: network_cli` as soon as possible.

Using CLI in Ansible

Example CLI group_vars/dellos10.yml

```
ansible_connection: network_cli
ansible_network_os: dellos10
ansible_user: myuser
ansible_password: !vault...
ansible_become: yes
ansible_become_method: enable
ansible_become_password: !vault...
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

- If you are using SSH keys (including an ssh-agent) you can remove the `ansible_password` configuration.
- If you are accessing your host directly (not through a bastion/jump host) you can remove the `ansible_ssh_common_args` configuration.
- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the `ProxyCommand` directive. To prevent secrets from leaking out (for example in `ps` output), SSH does not support providing passwords via environment variables.

Example CLI Task

```
- name: Backup current switch config (dellos10)
  dellos10_config:
    backup: yes
  register: backup_dellos10_location
  when: ansible_network_os == 'dellos10'
```

警告: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

参见:

Setting timeout options

ENOS Platform Options

ENOS supports Enable Mode (Privilege Escalation). This page offers details on how to use Enable Mode on ENOS in Ansible.

Topics

- *ENOS Platform Options*
 - *Connections Available*
 - *Using CLI in Ansible*
 - * *Example CLI group_vars/enos.yml*
 - * *Example CLI Task*

Connections Available

	CLI
Protocol	SSH
Credentials	uses SSH keys / SSH-agent if present accepts <code>-u myuser -k</code> if using password
Indirect Access	via a bastion (jump host)
Connection Settings	<code>ansible_connection: network_cli</code>
Enable Mode (Privilege Escalation)	supported: use <code>ansible_become: yes</code> with <code>ansible_become_method: enable</code> and <code>ansible_become_password:</code>
Returned Data Format	<code>stdout[0]</code> .

For legacy playbooks, ENOS still supports `ansible_connection: local`. We recommend modernizing to use `ansible_connection: network_cli` as soon as possible.

Using CLI in Ansible

Example CLI `group_vars/enos.yml`

```

ansible_connection: network_cli
ansible_network_os: enos
ansible_user: myuser
ansible_password: !vault...
ansible_become: yes
ansible_become_method: enable
ansible_become_password: !vault...
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q bastion01"'

```

- If you are using SSH keys (including an ssh-agent) you can remove the `ansible_password` configuration.
- If you are accessing your host directly (not through a bastion/jump host) you can remove the `ansible_ssh_common_args` configuration.
- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the `ProxyCommand` directive. To prevent secrets from leaking out (for example in `ps` output), SSH does not support providing passwords via environment variables.

Example CLI Task

```
- name: Retrieve ENOS OS version
  enos_command:
    commands: show version
  when: ansible_network_os == 'enos'
```

警告: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

参见:

Setting timeout options

EOS Platform Options

Arista EOS supports multiple connections. This page offers details on how each connection works in Ansible and how to use it.

Topics

- *EOS Platform Options*
 - *Connections Available*
 - *Using CLI in Ansible*
 - * *Example CLI group_vars/eos.yml*
 - * *Example CLI Task*
 - *Using eAPI in Ansible*
 - * *Enabling eAPI*
 - * *Example eAPI group_vars/eos.yml*
 - * *Example eAPI Task*
 - * *eAPI examples with connection: local*

Connections Available

	CLI	eAPI
Protocol	SSH	HTTP(S)
Credentials	uses SSH keys / SSH-agent if present accepts <code>-u myuser -k</code> if using password	uses HTTPS certificates if present
Indirect Access	via a bastion (jump host)	via a web proxy
Connection Settings	<code>ansible_connection:</code> <code>network_cli</code>	<code>ansible_connection: httpapi</code> OR <code>ansible_connection: local</code> with <code>transport: eapi</code> in the provider dictionary
Enable Mode (Privilege Escalation)	supported: <ul style="list-style-type: none"> • use <code>ansible_become:</code> yes with <code>ansible_become_method:</code> enable 	supported: <ul style="list-style-type: none"> • <code>httpapi</code> uses <code>ansible_become:</code> yes with <code>ansible_become_method:</code> enable • <code>local</code> uses <code>authorize:</code> yes and <code>auth_pass:</code> in the provider dictionary
Returned Data Format	<code>stdout[0]</code> .	<code>stdout[0].messages[0]</code> .

For legacy playbooks, EOS still supports `ansible_connection: local`. We recommend modernizing to use `ansible_connection: network_cli` or `ansible_connection: httpapi` as soon as possible.

Using CLI in Ansible

Example CLI `group_vars/eos.yml`

```

ansible_connection: network_cli
ansible_network_os: eos
ansible_user: myuser
ansible_password: !vault...
ansible_become: yes
ansible_become_method: enable

```

(下页继续)

(续上页)

```
ansible_become_password: !vault...
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

- If you are using SSH keys (including an ssh-agent) you can remove the `ansible_password` configuration.
- If you are accessing your host directly (not through a bastion/jump host) you can remove the `ansible_ssh_common_args` configuration.
- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the `ProxyCommand` directive. To prevent secrets from leaking out (for example in `ps` output), SSH does not support providing passwords via environment variables.

Example CLI Task

```
- name: Backup current switch config (eos)
  eos_config:
    backup: yes
  register: backup_eos_location
  when: ansible_network_os == 'eos'
```

Using eAPI in Ansible

Enabling eAPI

Before you can use eAPI to connect to a switch, you must enable eAPI. To enable eAPI on a new switch via Ansible, use the `eos_eapi` module via the CLI connection. Set up `group_vars/eos.yml` just like in the CLI example above, then run a playbook task like this:

```
- name: Enable eAPI
  eos_eapi:
    enable_http: yes
    enable_https: yes
  become: true
  become_method: enable
  when: ansible_network_os == 'eos'
```

You can find more options for enabling HTTP/HTTPS connections in the `eos_eapi` module documentation. Once eAPI is enabled, change your `group_vars/eos.yml` to use the eAPI connection.

Example eAPI group_vars/eos.yml

```

ansible_connection: httpapi
ansible_network_os: eos
ansible_user: myuser
ansible_password: !vault...
ansible_become: yes
ansible_become_method: enable
proxy_env:
  http_proxy: http://proxy.example.com:8080

```

- If you are accessing your host directly (not through a web proxy) you can remove the `proxy_env` configuration.
- If you are accessing your host through a web proxy using `https`, change `http_proxy` to `https_proxy`.

Example eAPI Task

```

- name: Backup current switch config (eos)
  eos_config:
    backup: yes
    register: backup_eos_location
    environment: "{{ proxy_env }}"
    when: ansible_network_os == 'eos'

```

In this example the `proxy_env` variable defined in `group_vars` gets passed to the `environment` option of the module in the task.

eAPI examples with connection: local

group_vars/eos.yml:

```

ansible_connection: local
ansible_network_os: eos
ansible_user: myuser
ansible_password: !vault...
eapi:
  host: "{{ inventory_hostname }}"
  transport: eapi
  authorize: yes
  auth_pass: !vault...

```

(下页继续)

(续上页)

```
proxy_env:
  http_proxy: http://proxy.example.com:8080
```

eAPI task:

```
- name: Backup current switch config (eos)
  eos_config:
    backup: yes
    provider: "{{ eapi }}"
    register: backup_eos_location
    environment: "{{ proxy_env }}"
    when: ansible_network_os == 'eos'
```

In this example two variables defined in `group_vars` get passed to the module of the task:

- the `eapi` variable gets passed to the `provider` option of the module
- the `proxy_env` variable gets passed to the `environment` option of the module

警告: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

参见:

Setting timeout options

ERIC_ECCLI Platform Options

Extreme ERIC_ECCLI Ansible modules only supports CLI connections today. This page offers details on how to use `network_cli` on ERIC_ECCLI in Ansible.

Topics

- *ERIC_ECCLI Platform Options*
 - *Connections Available*
 - *Using CLI in Ansible*
 - * *Example CLI group_vars/eric_eccli.yml*
 - * *Example CLI Task*

Connections Available

	CLI
Protocol	SSH
Credentials	uses SSH keys / SSH-agent if present accepts <code>-u myuser -k</code> if using password
Indirect Access	via a bastion (jump host)
Connection Settings	<code>ansible_connection: network_cli</code>
Enable Mode (Privilege Escalation)	not supported by ERIC_ECCLI
Returned Data Format	<code>stdout[0]</code> .

ERIC_ECCLI does not support `ansible_connection: local`. You must use `ansible_connection: network_cli`.

Using CLI in Ansible

Example CLI group_vars/eric_eccli.yml

```
ansible_connection: network_cli
ansible_network_os: eric_eccli
ansible_user: myuser
ansible_password: !vault...
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

- If you are using SSH keys (including an ssh-agent) you can remove the `ansible_password` configuration.
- If you are accessing your host directly (not through a bastion/jump host) you can remove the `ansible_ssh_common_args` configuration.
- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the `ProxyCommand` directive. To prevent secrets from leaking out (for example in `ps` output), SSH does not support providing passwords via environment variables.

Example CLI Task

```
- name: run show version on remote devices (eric_eccli)
  eric_eccli_command:
```

(下页继续)

(续上页)

```
commands: show version
when: ansible_network_os == 'eric_eccli'
```

警告: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

参见:

Setting timeout options

EXOS Platform Options

Extreme EXOS Ansible modules support multiple connections. This page offers details on how each connection works in Ansible and how to use it.

Topics

- *EXOS Platform Options*
 - *Connections Available*
 - *Using CLI in Ansible*
 - * *Example CLI group_vars/exos.yml*
 - * *Example CLI Task*
 - *Using EXOS-API in Ansible*
 - * *Example EXOS-API group_vars/exos.yml*
 - * *Example EXOS-API Task*

Connections Available

	CLI	EXOS-API
Protocol	SSH	HTTP(S)
Credentials	uses SSH keys / SSH-agent if present accepts <code>-u myuser -k</code> if using password	uses HTTPS certificates if present
Indirect Access	via a bastion (jump host)	via a web proxy
Connection Settings	<code>ansible_connection: network_cli</code>	<code>ansible_connection: httpapi</code>
Enable Mode (Privilege Escalation)	not supported by EXOS	not supported by EXOS
Returned Data Format	<code>stdout[0]</code> .	<code>stdout[0]</code> . <code>messages[0]</code> .

EXOS does not support `ansible_connection: local`. You must use `ansible_connection: network_cli` or `ansible_connection: httpapi`

Using CLI in Ansible

Example CLI `group_vars/exos.yml`

```

ansible_connection: network_cli
ansible_network_os: exos
ansible_user: myuser
ansible_password: !vault...
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q bastion01"'

```

- If you are using SSH keys (including an ssh-agent) you can remove the `ansible_password` configuration.
- If you are accessing your host directly (not through a bastion/jump host) you can remove the `ansible_ssh_common_args` configuration.
- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the `ProxyCommand` directive. To prevent secrets from leaking out (for example in `ps` output), SSH does not support providing passwords via environment variables.

Example CLI Task

```
- name: Retrieve EXOS OS version
  exos_command:
    commands: show version
  when: ansible_network_os == 'exos'
```

Using EXOS-API in Ansible

Example EXOS-API group_vars/exos.yml

```
ansible_connection: httpapi
ansible_network_os: exos
ansible_user: myuser
ansible_password: !vault...
proxy_env:
  http_proxy: http://proxy.example.com:8080
```

- If you are accessing your host directly (not through a web proxy) you can remove the `proxy_env` configuration.
- If you are accessing your host through a web proxy using `https`, change `http_proxy` to `https_proxy`.

Example EXOS-API Task

```
- name: Retrieve EXOS OS version
  exos_command:
    commands: show version
  when: ansible_network_os == 'exos'
```

In this example the `proxy_env` variable defined in `group_vars` gets passed to the `environment` option of the module used in the task.

警告: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports `ssh-agent` to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

参见:

Setting timeout options

ICX Platform Options

ICX supports Enable Mode (Privilege Escalation). This page offers details on how to use Enable Mode on ICX in Ansible.

Topics

- *ICX Platform Options*
 - *Connections Available*
 - *Using CLI in Ansible*
 - * *Example CLI group_vars/icx.yml*
 - * *Example CLI Task*

Connections Available

	CLI
Protocol	SSH
Credentials	uses SSH keys / SSH-agent if present accepts <code>-u myuser -k</code> if using password
Indirect Access	via a bastion (jump host)
Connection Settings	<code>ansible_connection: network_cli</code>
Enable Mode (Privilege Escalation)	supported: use <code>ansible_become: yes</code> with <code>ansible_become_method: enable</code> and <code>ansible_become_password:</code>
Returned Data Format	<code>stdout[0]</code> .

Using CLI in Ansible

Example CLI group_vars/icx.yml

```

ansible_connection: network_cli
ansible_network_os: icx
ansible_user: myuser
ansible_password: !vault...
ansible_become: yes
ansible_become_method: enable
ansible_become_password: !vault...
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q bastion01"'

```

- If you are using SSH keys (including an ssh-agent) you can remove the `ansible_password` configuration.
- If you are accessing your host directly (not through a bastion/jump host) you can remove the `ansible_ssh_common_args` configuration.
- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the `ProxyCommand` directive. To prevent secrets from leaking out (for example in `ps` output), SSH does not support providing passwords via environment variables.

Example CLI Task

```
- name: Backup current switch config (icx)
  icx_config:
    backup: yes
  register: backup_icx_location
  when: ansible_network_os == 'icx'
```

警告: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

参见:

Setting timeout options

IOS Platform Options

IOS supports Enable Mode (Privilege Escalation). This page offers details on how to use Enable Mode on IOS in Ansible.

Topics

- *IOS Platform Options*
 - *Connections Available*
 - *Using CLI in Ansible*
 - * *Example CLI group_vars/ios.yml*
 - * *Example CLI Task*

Connections Available

	CLI
Protocol	SSH
Credentials	uses SSH keys / SSH-agent if present accepts <code>-u myuser -k</code> if using password
Indirect Access	via a bastion (jump host)
Connection Settings	<code>ansible_connection: network_cli</code>
Enable Mode (Privilege Escalation)	supported: use <code>ansible_become: yes</code> with <code>ansible_become_method: enable</code> and <code>ansible_become_password:</code>
Returned Data Format	<code>stdout[0]</code> .

For legacy playbooks, IOS still supports `ansible_connection: local`. We recommend modernizing to use `ansible_connection: network_cli` as soon as possible.

Using CLI in Ansible

Example CLI group_vars/ios.yml

```

ansible_connection: network_cli
ansible_network_os: ios
ansible_user: myuser
ansible_password: !vault...
ansible_become: yes
ansible_become_method: enable
ansible_become_password: !vault...
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q bastion01"'

```

- If you are using SSH keys (including an ssh-agent) you can remove the `ansible_password` configuration.
- If you are accessing your host directly (not through a bastion/jump host) you can remove the `ansible_ssh_common_args` configuration.
- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the `ProxyCommand` directive. To prevent secrets from leaking out (for example in `ps` output), SSH does not support providing passwords via environment variables.

Example CLI Task

```
- name: Backup current switch config (ios)
  ios_config:
    backup: yes
  register: backup_ios_location
  when: ansible_network_os == 'ios'
```

警告: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

参见:

Setting timeout options

IOS-XR Platform Options

IOS-XR supports multiple connections. This page offers details on how each connection works in Ansible and how to use it.

Topic

- *IOS-XR Platform Options*
 - *Connections Available*
 - *Using CLI in Ansible*
 - * *Example CLI inventory [iosxr:vars]*
 - * *Example CLI Task*
 - *Using NETCONF in Ansible*
 - * *Enabling NETCONF*
 - * *Example NETCONF inventory [iosxr:vars]*
 - * *Example NETCONF Task*

Connections Available

	CLI	NETCONF only for modules <code>iosxr_banner</code> , <code>iosxr_interface</code> , <code>iosxr_logging</code> , <code>iosxr_system</code> , <code>iosxr_user</code>
Protocol	SSH	XML over SSH
Credentials	uses SSH keys / SSH-agent if present accepts <code>-u myuser -k</code> if using password	uses SSH keys / SSH-agent if present accepts <code>-u myuser -k</code> if using password
Indirect Access	via a bastion (jump host)	via a bastion (jump host)
Connection Settings	<code>ansible_connection:</code> <code>network_cli</code>	<code>ansible_connection:</code> <code>netconf</code>
Enable Mode (Privilege Escalation)	not supported	not supported
Returned Data Format	Refer to individual module documentation	Refer to individual module documentation

For legacy playbooks, Ansible still supports `ansible_connection=local` on all IOS-XR modules. We recommend modernizing to use `ansible_connection=netconf` or `ansible_connection=network_cli` as soon as possible.

Using CLI in Ansible

Example CLI inventory [iosxr:vars]

```
[iosxr:vars]
ansible_connection=network_cli
ansible_network_os=iosxr
ansible_user=myuser
ansible_password=!vault...
ansible_ssh_common_args='-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

- If you are using SSH keys (including an ssh-agent) you can remove the `ansible_password` configuration.
- If you are accessing your host directly (not through a bastion/jump host) you can remove the `ansible_ssh_common_args` configuration.

- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the `ProxyCommand` directive. To prevent secrets from leaking out (for example in `ps` output), SSH does not support providing passwords via environment variables.

Example CLI Task

```
- name: Retrieve IOS-XR version
  iosxr_command:
    commands: show version
  when: ansible_network_os == 'iosxr'
```

Using NETCONF in Ansible

Enabling NETCONF

Before you can use NETCONF to connect to a switch, you must:

- install the `ncclient` python package on your control node(s) with `pip install ncclient`
- enable NETCONF on the Cisco IOS-XR device(s)

To enable NETCONF on a new switch via Ansible, use the `iosxr_netconf` module via the CLI connection. Set up your platform-level variables just like in the CLI example above, then run a playbook task like this:

```
- name: Enable NETCONF
  connection: network_cli
  iosxr_netconf:
  when: ansible_network_os == 'iosxr'
```

Once NETCONF is enabled, change your variables to use the NETCONF connection.

Example NETCONF inventory [iosxr:vars]

```
[iosxr:vars]
ansible_connection=netconf
ansible_network_os=iosxr
ansible_user=myuser
ansible_password=!vault |
ansible_ssh_common_args='-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

Example NETCONF Task

```
- name: Configure hostname and domain-name
  iosxr_system:
    hostname: iosxr01
    domain_name: test.example.com
    domain_search:
      - ansible.com
      - redhat.com
      - cisco.com
```

警告: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

参见:

Setting timeout options

IronWare Platform Options

IronWare supports Enable Mode (Privilege Escalation). This page offers details on how to use Enable Mode on IronWare in Ansible.

Topics

- *IronWare Platform Options*
 - *Connections Available*
 - *Using CLI in Ansible*
 - * *Example CLI group_vars/mlx.yml*
 - * *Example CLI Task*

Connections Available

	CLI
Protocol	SSH
Credentials	uses SSH keys / SSH-agent if present accepts <code>-u myuser -k</code> if using password
Indirect Access	via a bastion (jump host)
Connection Settings	<code>ansible_connection: network_cli</code>
Enable Mode (Privilege Escalation)	supported: use <code>ansible_become: yes</code> with <code>ansible_become_method: enable</code> and <code>ansible_become_password:</code>
Returned Data Format	<code>stdout[0]</code> .

For legacy playbooks, IronWare still supports `ansible_connection: local`. We recommend modernizing to use `ansible_connection: network_cli` as soon as possible.

Using CLI in Ansible

Example CLI group_vars/mlx.yml

```

ansible_connection: network_cli
ansible_network_os: ironware
ansible_user: myuser
ansible_password: !vault...
ansible_become: yes
ansible_become_method: enable
ansible_become_password: !vault...
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q bastion01"'

```

- If you are using SSH keys (including an ssh-agent) you can remove the `ansible_password` configuration.
- If you are accessing your host directly (not through a bastion/jump host) you can remove the `ansible_ssh_common_args` configuration.
- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the `ProxyCommand` directive. To prevent secrets from leaking out (for example in `ps` output), SSH does not support providing passwords via environment variables.

Example CLI Task

```
- name: Backup current switch config (ironware)
  ironware_config:
    backup: yes
  register: backup_ironware_location
  when: ansible_network_os == 'ironware'
```

警告: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

参见:

Setting timeout options

Junos OS Platform Options

Juniper Junos OS supports multiple connections. This page offers details on how each connection works in Ansible and how to use it.

Topics

- *Junos OS Platform Options*
 - *Connections Available*
 - *Using CLI in Ansible*
 - * *Example CLI inventory [junos:vars]*
 - * *Example CLI Task*
 - *Using NETCONF in Ansible*
 - * *Enabling NETCONF*
 - * *Example NETCONF inventory [junos:vars]*
 - * *Example NETCONF Task*

Connections Available

	CLI junos_netconf & junos_command modules only	NETCONF all modules except junos_netconf, which enables NETCONF
Protocol	SSH	XML over SSH
Credentials	uses SSH keys / SSH-agent if present accepts -u myuser -k if using password	uses SSH keys / SSH-agent if present accepts -u myuser -k if using password
Indirect Access	via a bastion (jump host)	via a bastion (jump host)
Connection Settings	ansible_connection: network_cli	ansible_connection: netconf
Enable Mode (Privilege Escalation)	not supported by Junos OS	not supported by Junos OS
Returned Data Format	stdout[0].	<ul style="list-style-type: none"> json: result[0]['software-information'][0]['foo lo0'] text: result[1]. interface-information[0]. physical-interface[0]. name[0].data foo lo0 xml: result[1]. rpc-reply. interface-information[0]. physical-interface[0]. name[0].data foo lo0

For legacy playbooks, Ansible still supports `ansible_connection=local` on all JUNOS modules. We recommend modernizing to use `ansible_connection=netconf` or `ansible_connection=network_cli` as soon as possible.

Using CLI in Ansible

Example CLI inventory [junos:vars]

```
[junos:vars]
ansible_connection=network_cli
ansible_network_os=junos
ansible_user=myuser
ansible_password=!vault...
ansible_ssh_common_args='-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

- If you are using SSH keys (including an ssh-agent) you can remove the `ansible_password` configuration.
- If you are accessing your host directly (not through a bastion/jump host) you can remove the `ansible_ssh_common_args` configuration.
- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the `ProxyCommand` directive. To prevent secrets from leaking out (for example in `ps` output), SSH does not support providing passwords via environment variables.

Example CLI Task

```
- name: Retrieve Junos OS version
  junos_command:
    commands: show version
  when: ansible_network_os == 'junos'
```

Using NETCONF in Ansible

Enabling NETCONF

Before you can use NETCONF to connect to a switch, you must:

- install the `ncclient` python package on your control node(s) with `pip install ncclient`
- enable NETCONF on the Junos OS device(s)

To enable NETCONF on a new switch via Ansible, use the `junos_netconf` module via the CLI connection. Set up your platform-level variables just like in the CLI example above, then run a playbook task like this:

```
- name: Enable NETCONF
  connection: network_cli
  junos_netconf:
  when: ansible_network_os == 'junos'
```

Once NETCONF is enabled, change your variables to use the NETCONF connection.

Example NETCONF inventory [junos:vars]

```
[junos:vars]
ansible_connection=netconf
ansible_network_os=junos
ansible_user=myuser
ansible_password=!vault |
ansible_ssh_common_args='-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

Example NETCONF Task

```
- name: Backup current switch config (junos)
  junos_config:
    backup: yes
  register: backup_junos_location
  when: ansible_network_os == 'junos'
```

警告: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

参见:

Setting timeout options

Meraki Platform Options

Meraki only support supports the local connection type at this time.

Topics

- *Meraki Platform Options*
 - *Connections Available*
 - * *Example Meraki Task*

Connections Available

	Dashboard API
Protocol	HTTP(S)
Credentials	uses API key from Dashboard
Connection Settings	<code>ansible_connection: localhost</code>
Returned Data Format	<code>data.</code>

Example Meraki Task

```
meraki_organization:
  auth_key: abc12345
  org_name: YourOrg
  state: present
delegate_to: localhost
```

参见:

Setting timeout options

Pluribus NETVISOR Platform Options

Pluribus NETVISOR Ansible modules only support CLI connections today. `httpapi` modules may be added in future. This page offers details on how to use `network_cli` on NETVISOR in Ansible.

Topics

- *Pluribus NETVISOR Platform Options*
 - *Connections Available*
 - *Using CLI in Ansible*
 - * *Example CLI group_vars/netvisor.yml*
 - * *Example CLI Task*

Connections Available

	CLI
Protocol	SSH
Credentials	uses SSH keys / SSH-agent if present accepts <code>-u myuser -k</code> if using password
Indirect Access	via a bastion (jump host)
Connection Settings	<code>ansible_connection: network_cli</code>
Enable Mode (Privilege Escalation)	not supported by NETVISOR
Returned Data Format	<code>stdout[0]</code> .

Pluribus NETVISOR does not support `ansible_connection: local`. You must use `ansible_connection: network_cli`.

Using CLI in Ansible

Example CLI group_vars/netvisor.yml

```
ansible_connection: network_cli
ansible_network_os: netvisor
ansible_user: myuser
ansible_password: !vault...
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

- If you are using SSH keys (including an ssh-agent) you can remove the `ansible_password` configuration.
- If you are accessing your host directly (not through a bastion/jump host) you can remove the `ansible_ssh_common_args` configuration.
- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the `ProxyCommand` directive. To prevent secrets from leaking out (for example in `ps` output), SSH does not support providing passwords via environment variables.

Example CLI Task

```
- name: Create access list
  pn_access_list:
    pn_name: "foo"
```

(下页继续)

(续上页)

```
pn_scope: "local"
state: "present"
register: acc_list
when: ansible_network_os == 'netvisor'
```

警告: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

参见:

Setting timeout options

NOS Platform Options

Extreme NOS Ansible modules only support CLI connections today. `httpapi` modules may be added in future. This page offers details on how to use `network_cli` on NOS in Ansible.

Topics

- *NOS Platform Options*
 - *Connections Available*
 - *Using CLI in Ansible*
 - * *Example CLI group_vars/nos.yml*
 - * *Example CLI Task*

Connections Available

	CLI
Protocol	SSH
Credentials	uses SSH keys / SSH-agent if present accepts <code>-u myuser -k</code> if using password
Indirect Access	via a bastion (jump host)
Connection Settings	<code>ansible_connection: network_cli</code>
Enable Mode (Privilege Escalation)	not supported by NOS
Returned Data Format	<code>stdout[0]</code> .

NOS does not support `ansible_connection: local`. You must use `ansible_connection: network_cli`.

Using CLI in Ansible

Example CLI group_vars/nos.yml

```
ansible_connection: network_cli
ansible_network_os: nos
ansible_user: myuser
ansible_password: !vault...
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

- If you are using SSH keys (including an ssh-agent) you can remove the `ansible_password` configuration.
- If you are accessing your host directly (not through a bastion/jump host) you can remove the `ansible_ssh_common_args` configuration.
- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the `ProxyCommand` directive. To prevent secrets from leaking out (for example in `ps` output), SSH does not support providing passwords via environment variables.

Example CLI Task

```
- name: Get version information (nos)
  nos_command:
    commands: "show version"
```

(下页继续)

(续上页)

```
register: show_ver
when: ansible_network_os == 'nos'
```

警告: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

参见:

Setting timeout options

NXOS Platform Options

Cisco NXOS supports multiple connections. This page offers details on how each connection works in Ansible and how to use it.

Topics

- *NXOS Platform Options*
 - *Connections Available*
 - *Using CLI in Ansible*
 - * *Example CLI group_vars/nxos.yml*
 - * *Example CLI Task*
 - *Using NX-API in Ansible*
 - * *Enabling NX-API*
 - * *Example NX-API group_vars/nxos.yml*
 - * *Example NX-API Task*
 - *Cisco Nexus Platform Support Matrix*

Connections Available

	CLI	NX-API
Protocol	SSH	HTTP(S)
Credentials	uses SSH keys / SSH-agent if present accepts <code>-u myuser -k</code> if using password	uses HTTPS certificates if present
Indirect Access	via a bastion (jump host)	via a web proxy
Connection Settings	<code>ansible_connection: network_cli</code>	<code>ansible_connection: httpapi</code> OR <code>ansible_connection: local</code> with <code>transport: nxapi</code> in the provider dictionary
Enable Mode (Privilege Escalation) supported as of 2.5.3	supported: use <code>ansible_become: yes</code> with <code>ansible_become_method: enable</code> and <code>ansible_become_password:</code>	not supported by NX-API
Returned Data Format	<code>stdout[0]</code> .	<code>stdout[0].messages[0]</code> .

For legacy playbooks, NXOS still supports `ansible_connection: local`. We recommend modernizing to use `ansible_connection: network_cli` or `ansible_connection: httpapi` as soon as possible.

Using CLI in Ansible

Example CLI `group_vars/nxos.yml`

```

ansible_connection: network_cli
ansible_network_os: nxos
ansible_user: myuser
ansible_password: !vault...
ansible_become: yes
ansible_become_method: enable
ansible_become_password: !vault...
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q bastion01"'

```

- If you are using SSH keys (including an ssh-agent) you can remove the `ansible_password` configuration.
- If you are accessing your host directly (not through a bastion/jump host) you can remove the `ansible_ssh_common_args` configuration.

- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the `ProxyCommand` directive. To prevent secrets from leaking out (for example in `ps` output), SSH does not support providing passwords via environment variables.

Example CLI Task

```
- name: Backup current switch config (nxos)
  nxos_config:
    backup: yes
  register: backup_nxos_location
  when: ansible_network_os == 'nxos'
```

Using NX-API in Ansible

Enabling NX-API

Before you can use NX-API to connect to a switch, you must enable NX-API. To enable NX-API on a new switch via Ansible, use the `nxos_nxapi` module via the CLI connection. Set up `group_vars/nxos.yml` just like in the CLI example above, then run a playbook task like this:

```
- name: Enable NX-API
  nxos_nxapi:
    enable_http: yes
    enable_https: yes
  when: ansible_network_os == 'nxos'
```

To find out more about the options for enabling HTTP/HTTPS and local http see the `nxos_nxapi` module documentation.

Once NX-API is enabled, change your `group_vars/nxos.yml` to use the NX-API connection.

Example NX-API `group_vars/nxos.yml`

```
ansible_connection: httpapi
ansible_network_os: nxos
ansible_user: myuser
ansible_password: !vault...
proxy_env:
  http_proxy: http://proxy.example.com:8080
```

- If you are accessing your host directly (not through a web proxy) you can remove the `proxy_env` configuration.
- If you are accessing your host through a web proxy using `https`, change `http_proxy` to `https_proxy`.

Example NX-API Task

```
- name: Backup current switch config (nxos)
  nxos_config:
    backup: yes
  register: backup_nxos_location
  environment: "{{ proxy_env }}"
  when: ansible_network_os == 'nxos'
```

In this example the `proxy_env` variable defined in `group_vars` gets passed to the `environment` option of the module used in the task.

警告: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

Cisco Nexus Platform Support Matrix

The following platforms and software versions have been certified by Cisco to work with this version of Ansible.

表 2: Platform / Software Minimum Requirements

Supported Platforms	Minimum NX-OS Version
Cisco Nexus N3k	7.0(3)I2(5) and later
Cisco Nexus N9k	7.0(3)I2(5) and later
Cisco Nexus N5k	7.3(0)N1(1) and later
Cisco Nexus N6k	7.3(0)N1(1) and later
Cisco Nexus N7k	7.3(0)D1(1) and later

表 3: Platform Models

Platform	Description
N3k	Support includes N30xx, N31xx and N35xx models
N5k	Support includes all N5xxx models
N6k	Support includes all N6xxx models
N7k	Support includes all N7xxx models
N9k	Support includes all N9xxx models

参见:

Setting timeout options

RouterOS Platform Options

Topics

- *RouterOS Platform Options*
 - *Connections Available*
 - *Using CLI in Ansible*
 - * *Example CLI group_vars/routeros.yml*
 - * *Example CLI Task*

Connections Available

	CLI
Protocol	SSH
Credentials	uses SSH keys / SSH-agent if present accepts <code>-u myuser -k</code> if using password
Indirect Access	via a bastion (jump host)
Connection Settings	<code>ansible_connection: network_cli</code>
Enable Mode (Privilege Escalation)	not supported by RouterOS
Returned Data Format	<code>stdout[0]</code> .

RouterOS does not support `ansible_connection: local`. You must use `ansible_connection: network_cli`.

Using CLI in Ansible

Example CLI group_vars/routeros.yml

```
ansible_connection: network_cli
ansible_network_os: routeros
ansible_user: myuser
ansible_password: !vault...
ansible_become: yes
ansible_become_method: enable
ansible_become_password: !vault...
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

- If you are using SSH keys (including an ssh-agent) you can remove the `ansible_password` configuration.
- If you are accessing your host directly (not through a bastion/jump host) you can remove the `ansible_ssh_common_args` configuration.
- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the `ProxyCommand` directive. To prevent secrets from leaking out (for example in `ps` output), SSH does not support providing passwords via environment variables.
- If you are getting timeout errors you may want to add `+cet1024w` suffix to your username which will disable console colors, enable “dumb” mode, tell RouterOS not to try detecting terminal capabilities and set terminal width to 1024 columns. See article [Console login process](#) in MikroTik wiki for more information.

Example CLI Task

```
- name: Display resource statistics (routeros)
  routeros_command:
    commands: /system resource print
  register: routeros_resources
  when: ansible_network_os == 'routeros'
```

警告: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

参见:

Setting timeout options

SLX-OS Platform Options

Extreme SLX-OS Ansible modules only support CLI connections today. `httpapi` modules may be added in future. This page offers details on how to use `network_cli` on SLX-OS in Ansible.

Topics

- *SLX-OS Platform Options*
 - *Connections Available*
 - *Using CLI in Ansible*
 - * *Example CLI group_vars/slxos.yml*
 - * *Example CLI Task*

Connections Available

	CLI
Protocol	SSH
Credentials	uses SSH keys / SSH-agent if present accepts <code>-u myuser -k</code> if using password
Indirect Access	via a bastion (jump host)
Connection Settings	<code>ansible_connection: network_cli</code>
Enable Mode (Privilege Escalation)	not supported by SLX-OS
Returned Data Format	<code>stdout[0]</code> .

SLX-OS does not support `ansible_connection: local`. You must use `ansible_connection: network_cli`.

Using CLI in Ansible

Example CLI group_vars/slxos.yml

```
ansible_connection: network_cli
ansible_network_os: slxos
ansible_user: myuser
```

(下页继续)

(续上页)

```
ansible_password: !vault...
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

- If you are using SSH keys (including an ssh-agent) you can remove the `ansible_password` configuration.
- If you are accessing your host directly (not through a bastion/jump host) you can remove the `ansible_ssh_common_args` configuration.
- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the `ProxyCommand` directive. To prevent secrets from leaking out (for example in `ps` output), SSH does not support providing passwords via environment variables.

Example CLI Task

```
- name: Backup current switch config (slxos)
  slxos_config:
    backup: yes
  register: backup_slxos_location
  when: ansible_network_os == 'slxos'
```

警告: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

参见:

Setting timeout options

VOSS Platform Options

Extreme VOSS Ansible modules only support CLI connections today. This page offers details on how to use `network_cli` on VOSS in Ansible.

Topics

- *VOSS Platform Options*
 - *Connections Available*
 - *Using CLI in Ansible*

* *Example CLI group_vars/voss.yml*

* *Example CLI Task*

Connections Available

	CLI
Protocol	SSH
Credentials	uses SSH keys / SSH-agent if present accepts <code>-u myuser -k</code> if using password
Indirect Access	via a bastion (jump host)
Connection Settings	<code>ansible_connection: network_cli</code>
Enable Mode (Privilege Escalation)	supported: use <code>ansible_become: yes</code> with <code>ansible_become_method: enable</code>
Returned Data Format	<code>stdout[0]</code> .

VOSS does not support `ansible_connection: local`. You must use `ansible_connection: network_cli`.

Using CLI in Ansible

Example CLI group_vars/voss.yml

```
ansible_connection: network_cli
ansible_network_os: voss
ansible_user: myuser
ansible_become: yes
ansible_become_method: enable
ansible_password: !vault...
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

- If you are using SSH keys (including an ssh-agent) you can remove the `ansible_password` configuration.
- If you are accessing your host directly (not through a bastion/jump host) you can remove the `ansible_ssh_common_args` configuration.
- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the `ProxyCommand` directive. To prevent secrets from leaking out (for example in `ps` output), SSH does not support providing passwords via environment variables.

Example CLI Task

```
- name: Retrieve VOSS info
  voss_command:
    commands: show sys-info
  when: ansible_network_os == 'voss'
```

警告: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

参见:

Setting timeout options

VyOS Platform Options

VyOS supports Enable Mode (Privilege Escalation). This page offers details on how to use Enable Mode on VyOS in Ansible.

Topics

- *VyOS Platform Options*
 - *Connections Available*
 - *Using CLI in Ansible*
 - * *Example CLI group_vars/vyos.yml*
 - * *Example CLI Task*

Connections Available

	CLI
Protocol	SSH
Credentials	uses SSH keys / SSH-agent if present accepts <code>-u myuser -k</code> if using password
Indirect Access	via a bastion (jump host)
Connection Settings	<code>ansible_connection: network_cli</code>
Enable Mode (Privilege Escalation)	not supported
Returned Data Format	Refer to individual module documentation

For legacy playbooks, VyOS still supports `ansible_connection: local`. We recommend modernizing to use `ansible_connection: network_cli` as soon as possible.

Using CLI in Ansible

Example CLI `group_vars/vyos.yml`

```
ansible_connection: network_cli
ansible_network_os: vyos
ansible_user: myuser
ansible_password: !vault...
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

- If you are using SSH keys (including an ssh-agent) you can remove the `ansible_password` configuration.
- If you are accessing your host directly (not through a bastion/jump host) you can remove the `ansible_ssh_common_args` configuration.
- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the `ProxyCommand` directive. To prevent secrets from leaking out (for example in `ps` output), SSH does not support providing passwords via environment variables.

Example CLI Task

```
- name: Retrieve VyOS version info
  vyos_command:
```

(下页继续)

(续上页)

```
commands: show version
when: ansible_network_os == 'vyos'
```

警告: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

参见:

Setting timeout options

Netconf enabled Platform Options

This page offers details on how the netconf connection works in Ansible and how to use it.

Topics

- *Netconf enabled Platform Options*
 - *Connections Available*
 - *Using NETCONF in Ansible*
 - * *Enabling NETCONF*
 - * *Example NETCONF inventory [junos:vars]*
 - * *Example NETCONF Task*
 - * *Example NETCONF Task with configurable variables*
 - * *Bastion/Jumphost Configuration*
 - * *ansible_network_os auto-detection*

Connections Available

	NETCONF all modules except <code>junos_netconf</code> , which enables NETCONF
Protocol	XML over SSH
Credentials	uses SSH keys / SSH-agent if present accepts <code>-u myuser -k</code> if using password
Indirect Access	via a bastion (jump host)
Connection Settings	<code>ansible_connection: netconf</code>

For legacy playbooks, Ansible still supports `ansible_connection=local` for the `netconf_config` module only. We recommend modernizing to use `ansible_connection=netconf` as soon as possible.

Using NETCONF in Ansible

Enabling NETCONF

Before you can use NETCONF to connect to a switch, you must:

- install the `ncclient` Python package on your control node(s) with `pip install ncclient`
- enable NETCONF on the Junos OS device(s)

To enable NETCONF on a new switch via Ansible, use the platform specific module via the CLI connection or set it manually. For example set up your platform-level variables just like in the CLI example above, then run a playbook task like this:

```
- name: Enable NETCONF
  connection: network_cli
  junos_netconf:
  when: ansible_network_os == 'junos'
```

Once NETCONF is enabled, change your variables to use the NETCONF connection.

Example NETCONF inventory [junos:vars]

```
[junos:vars]
ansible_connection=netconf
ansible_network_os=junos
ansible_user=myuser
ansible_password=!vault |
```

Example NETCONF Task

```
- name: Backup current switch config
  netconf_config:
    backup: yes
    register: backup_junos_location
```

Example NETCONF Task with configurable variables

```
- name: configure interface while providing different private key file path
  netconf_config:
    backup: yes
    register: backup_junos_location
    vars:
      ansible_private_key_file: /home/admin/.ssh/newprivatekeyfile
```

Note: For netconf connection plugin configurable variables see netconf.

Bastion/Jumphost Configuration

To use a jump host to connect to a NETCONF enabled device you must set the `ANSIBLE_NETCONF_SSH_CONFIG` environment variable.

`ANSIBLE_NETCONF_SSH_CONFIG` can be set to either:

- 1 or TRUE (to trigger the use of the default SSH config file `~/.ssh/config`)
- The absolute path to a custom SSH config file.

The SSH config file should look something like:

```
Host *
  proxycommand ssh -o StrictHostKeyChecking=no -W %h:%p jumphost-username@jumphost.fqdn.
↪com
  StrictHostKeyChecking no
```

Authentication for the jump host must use key based authentication.

You can either specify the private key used in the SSH config file:

```
IdentityFile "/absolute/path/to/private-key.pem"
```

Or you can use an ssh-agent.

ansible_network_os auto-detection

If `ansible_network_os` is not specified for a host, then Ansible will attempt to automatically detect what `network_os` plugin to use.

`ansible_network_os` auto-detection can also be triggered by using `auto` as the `ansible_network_os`. (Note: Previously `default` was used instead of `auto`).

警告: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

参见:

Setting timeout options

Settings by Platform

		ansible_connection: settings available			
Network OS	ansible_network_os:	net-work_cli	netconf	httpapi	local
Arista EOS [†]	eos	✓		✓	✓
Cisco ASA	asa	✓			✓
Cisco IOS [†]	ios	✓			✓
Cisco IOS XR [†]	iosxr	✓			✓
Cisco NX-OS [†]	nxos	✓		✓	✓
Cloudengine OS [†]	ce	✓	✓		✓
Dell OS6	dello6	✓			✓
Dell OS9	dello9	✓			✓
Dell OS10	dello10	✓			✓
Ericsson ECCLI	eric_eccli	✓			✓
Extreme EXOS	exos	✓		✓	
Extreme IronWare	ironware	✓			✓
Extreme NOS	nos	✓			
Extreme SLX-OS	slxos	✓			
Extreme VOSS	voss	✓			
F5 BIG-IP					✓
F5 BIG-IQ					✓
Junos OS [†]	junos	✓	✓		✓
Lenovo CNOS	cnos	✓			✓
Lenovo ENOS	enos	✓			✓
Meraki	meraki				✓
MikroTik RouterOS	routeros	✓			
Nokia SR OS	sros	✓			✓
Pluribus Netvisor	netvisor	✓			
Ruckus ICX [†]	icx	✓			
VyOS [†]	vyos	✓			✓
OS that supports Netconf [†]	<network-os>		✓		✓

[†] Maintained by Ansible Network Team

1.11 Network Automation Developer Guide

Welcome to the Developer Guide for Ansible Network Automation!

Who should use this guide?

If you want to extend Ansible for Network Automation by creating a module or plugin, this guide is for you. This guide is specific to networking. You should already be familiar with how to create, test, and document modules and plugins, as well as the prerequisites for getting your module or plugin accepted into the main Ansible repository. See the *Developer Guide* for details. Before you proceed, please read:

- How to *add a custom plugin or module locally*.
- How to figure out if *developing a module is the right approach* for my use case.
- How to *set up my Python development environment*.
- How to *get started writing a module*.

Find the network developer task that best describes what you want to do:

- I want to *develop a network resource module*.
- I want to *develop a network connection plugin*.
- I want to *document my set of modules for a network platform*.

If you prefer to read the entire guide, here's a list of the pages in order.

1.11.1 Developing network resource modules

- *Accessing the resource module builder*
- *Creating a model*
- *Using the resource module builder*
- *Examples*
 - *Collection directory layout*
 - *Role directory layout*
 - *Using the collection*
 - *Using the role*
- *Resource module structure and workflow*
- *Developer notes*
- *Unit testing Ansible network resource modules*
 - *Using mock objects to unit test Ansible network resource modules*
 - *Mocking device data*

The resource module builder is an Ansible Playbook that helps developers scaffold and maintain an Ansible network resource module.

The resource module builder has the following capabilities:

- Uses a defined model to scaffold a resource module directory layout and initial class files.
- Scaffolds either an Ansible role or a collection.
- Subsequent uses of the resource module builder will only replace the module argspec and file containing the module docstring.
- Allows you to store complex examples along side the model in the same directory.
- Maintains the model as the source of truth for the module and use resource module builder to update the source files as needed.
- Generates working sample modules for both `<network_os>_<resource>` and `<network_os>_facts`.

Accessing the resource module builder

To access the resource module builder:

1. clone the github repository:

```
git clone https://github.com/ansible-network/resource_module_builder.git
```

2. Install the requirements:

```
pip install -r requirements.txt
```

Creating a model

You must create a model for your new resource. The resource module builder uses this model to create:

- The scaffold for a new module
- The argspec for the new module
- The docstring for the new module

The model is then the single source of truth for both the argspec and docstring, keeping them in sync. Use the resource module builder to generate this scaffolding. For any subsequent updates to the module, update the model first and use the resource module builder to update the module argspec and docstring.

For example, the resource model builder includes the `myos_interfaces.yml` sample in the `models` directory, as seen below:

```
---
GENERATOR_VERSION: '1.0'
ANSIBLE_METADATA: |
  {
```

(下页继续)

(续上页)

```

        'metadata_version': '1.1',
        'status': ['preview'],
        'supported_by': '<support_group>'
    }
NETWORK_OS: myos
RESOURCE: interfaces
COPYRIGHT: Copyright 2019 Red Hat
LICENSE: gpl-3.0.txt

DOCUMENTATION: |
    module: myos_interfaces
    version_added: 2.9
    short_description: 'Manages <xxxx> attributes of <network_os> <resource>'
    description: 'Manages <xxxx> attributes of <network_os> <resource>.'
    author: Ansible Network Engineer
notes:
    - 'Tested against <network_os> <version>'
options:
    config:
        description: The provided configuration
        type: list
        elements: dict
        suboptions:
            name:
                type: str
                description: The name of the <resource>
            some_string:
                type: str
                description:
                    - The some_string_01
                choices:
                    - choice_a
                    - choice_b
                    - choice_c
                default: choice_a
            some_bool:
                description:
                    - The some_bool.
                type: bool
            some_int:

```

(下页继续)

(续上页)

```

    description:
      - The some_int.
    type: int
    version_added: '1.1'
  some_dict:
    type: dict
    description:
      - The some_dict.
    suboptions:
      property_01:
        description:
          - The property_01
        type: str
  state:
    description:
      - The state of the configuration after module completion.
    type: str
    choices:
      - merged
      - replaced
      - overridden
      - deleted
    default: merged
EXAMPLES:
- deleted_example_01.txt
- merged_example_01.txt
- overridden_example_01.txt
- replaced_example_01.txt

```

Notice that you should include examples for each of the states that the resource supports. The resource module builder also includes these in the sample model.

See [Ansible network resource models](#) for more examples.

Using the resource module builder

To use the resource module builder to create a collection scaffold from your resource model:

```

ansible-playbook -e rm_dest=<destination for modules and module utils> \
                  -e structure=collection \
                  -e collection_org=<collection_org> \

```

(下页继续)

(续上页)

```
-e collection_name=<collection_name> \
-e model=<model> \
site.yml
```

Where the parameters are as follows:

- **rm_dest**: The directory where the resource module builder places the files and directories for the resource module and facts modules.
- **structure**: The directory layout type (role or collection)
 - **role**: Generate a role directory layout.
 - **collection**: Generate a collection directory layout.
- **collection_org**: The organization of the collection, required when *structure=collection*.
- **collection_name**: The name of the collection, required when *structure=collection*.
- **model**: The path to the model file.

To use the resource module builder to create a role scaffold:

```
ansible-playbook -e rm_dest=<destination for modules and module utils> \
-e structure=role \
-e model=<model> \
site.yml
```

Examples

Collection directory layout

This example shows the directory layout for the following:

- **network_os**: myos
- **resource**: interfaces

```
ansible-playbook -e rm_dest=~/.github/rm_example \
-e structure=collection \
-e collection_org=cidrblock \
-e collection_name=my_collection \
-e model=models/myos/interfaces/myos_interfaces.yml \
site.yml
```

```
docs
LICENSE.txt
playbooks
plugins
|   action
|   filter
|   inventory
|   modules
| |   __init__.py
| |   myos_facts.py
| |   myos_interfaces.py
|   module_utils
|       __init__.py
|       network
|           __init__.py
|           myos
|               argspec
|                   |   facts
|                   | |   facts.py
|                   | |   __init__.py
|                   |   __init__.py
|                   |   interfaces
|                   |       __init__.py
|                   |       interfaces.py
|                   config
|                   |   __init__.py
|                   |   interfaces
|                   |       __init__.py
|                   |       interfaces.py
|                   facts
|                   |   facts.py
|                   |   __init__.py
|                   |   interfaces
|                   |       __init__.py
|                   |       interfaces.py
|                   __init__.py
|                   utils
|                       __init__.py
|                       utils.py
README.md
roles
```

Role directory layout

This example displays the role directory layout for the following:

- `network_os`: `myos`
- `resource`: `interfaces`

```
ansible-playbook -e rm_dest=~/.github/rm_example/roles/my_role \
                  -e structure=role \
                  -e model=models/myos/interfaces/myos_interfaces.yml \
                  site.yml
```

```
roles
  my_role
    library
      __init__.py
      myos_facts.py
      myos_interfaces.py
    LICENSE.txt
    module_utils
      __init__.py
    network
      __init__.py
      myos
        argspec
          facts
            facts.py
            __init__.py
          __init__.py
          interfaces
            __init__.py
            interfaces.py
        config
          __init__.py
          interfaces
            __init__.py
            interfaces.py
        facts
          facts.py
          __init__.py
          interfaces
```

(下页继续)

(续上页)

```
        __init__.py
        interfaces.py
    __init__.py
    utils
        __init__.py
        utils.py
README.md
```

Using the collection

This example shows how to use the generated collection in a playbook:

```
----
- hosts: myos101
  gather_facts: False
  tasks:
    - cidrblock.my_collection.myos_interfaces:
      register: result
    - debug:
      var: result
    - cidrblock.my_collection.myos_facts:
    - debug:
      var: ansible_network_resources
```

Using the role

This example shows how to use the generated role in a playbook:

```
- hosts: myos101
  gather_facts: False
  roles:
    - my_role

- hosts: myos101
  gather_facts: False
  tasks:
    - myos_interfaces:
      register: result
    - debug:
```

(下页继续)

(续上页)

```

var: result
- myos_facts:
- debug:
  var: ansible_network_resources

```

Resource module structure and workflow

The resource module structure includes the following components:

Module

- library/<ansible_network_os>_<resource>.py.
- Imports the module_utils resource package and calls execute_module API

```

def main():
    result = <resource_package>(module).execute_module()

```

Module argspec

- module_utils/<ansible_network_os>/argspec/<resource>/.
- Argspec for the resource.

Facts

- module_utils/<ansible_network_os>/facts/<resource>/.
- Populate facts for the resource.
- Entry in module_utils/<ansible_network_os>/facts/facts.py for get_facts API to keep <ansible_network_os>_facts module and facts gathered for the resource module in sync for every subset.
- Entry of Resource subset in FACTS_RESOURCE_SUBSETS list in module_utils/<ansible_network_os>/facts/facts.py to make facts collection work.

Module package in module_utils

- module_utils/<ansible_network_os>/<config>/<resource>/.
- Implement execute_module API that loads the configuration to device and generates the result with changed, commands, before and after keys.
- Call get_facts API that returns the <resource> configuration facts or return the difference if the device has onbox diff support.
- Compare facts gathered and given key-values if diff is not supported.
- Generate final configuration.

Utils

- `module_utils/<ansible_network_os>/utils`.
- Utilities for the `<ansible_network_os>` platform.

Developer notes

The tests rely on a role generated by the resource module builder. After changes to the resource module builder, the role should be regenerated and the tests modified and run as needed. To generate the role after changes:

```
rm -rf rmb_tests/roles/my_role
ansible-playbook -e rm_dest=./rmb_tests/roles/my_role \
                  -e structure=role \
                  -e model=models/myos/interfaces/myos_interfaces.yml \
                  site.yml
```

Unit testing Ansible network resource modules

This section walks through an example of how to develop unit tests for Ansible network resource modules. See `testing_units` and `testing_units_modules` for general documentation on Ansible unit tests for modules. Please read those pages first to understand unit tests and why and when you should use them.

注解: The structure of the unit tests matches the structure of the code base, so the tests that reside in the `test/units/modules/network` directory are organized by module groups.

Using mock objects to unit test Ansible network resource modules

Mock objects (from <https://docs.python.org/3/library/unittest.mock.html>) can be very useful in building unit tests for special or difficult cases, but they can also lead to complex and confusing coding situations. One good use for mocks would be to simulate an API. The `mock` Python package is bundled with Ansible (use `import unittest.mock`).

You can mock the device connection and output from the device as follows:

```
self.mock_get_config = patch('ansible.module_utils.network.common.network.Config.get_
↪config')
self.get_config = self.mock_get_config.start()

self.mock_load_config = patch('ansible.module_utils.network.common.network.Config.load_
↪config')
```

(下页继续)

(续上页)

```

self.load_config = self.mock_load_config.start()

self.mock_get_resource_connection_config = patch('ansible.module_utils.network.common.
↪cfg.base.get_resource_connection')
self.get_resource_connection_config = self.mock_get_resource_connection_config.start()

self.mock_get_resource_connection_facts = patch('ansible.module_utils.network.common.
↪facts.facts.get_resource_connection')
self.get_resource_connection_facts = self.mock_get_resource_connection_facts.start()

self.mock_edit_config = patch('ansible.module_utils.network.eos.providers.providers.
↪CliProvider.edit_config')
self.edit_config = self.mock_edit_config.start()

self.mock_execute_show_command = patch('ansible.module_utils.network.eos.facts.l2_
↪interfaces.l2_interfaces.L2_interfacesFacts.get_device_data')
self.execute_show_command = self.mock_execute_show_command.start()

```

The facts file of the module now includes a new method, `get_device_data`. Call `get_device_data` here to emulate the device output.

Mocking device data

To mock fetching results from devices or provide other complex data structures that come from external libraries, you can use `fixtures` to read in pre-generated data. The text files for this pre-generated data live in `test/units/modules/network/PLATFORM/fixtures/`. See for example the `eos_l2_interfaces.cfg` file.

Load data using the `load_fixture` method and set this data as the return value of the `get_device_data` method in the facts file:

```

def load_fixtures(self, commands=None, transport='cli'):
    def load_from_file(*args, **kwargs):
        return load_fixture('eos_l2_interfaces_config.cfg')
    self.execute_show_command.side_effect = load_from_file

```

See the unit test file `test_eos_l2_interfaces` for a practical example.

参见:

`testing__units` Ansible unit tests documentation

`testing__units` Deep dive into developing unit tests for Ansible modules

`testing__running__locally` Running tests locally including gathering and reporting coverage data

Ansible module development: getting started Get started developing a module

1.11.2 Network connection plugins

Each network connection plugin has a set of its own plugins which provide a specification of the connection for a particular set of devices. The specific plugin used is selected at runtime based on the value of the `ansible_network_os` variable assigned to the host. This variable should be set to the same value as the name of the plugin to be loaded. Thus, `ansible_network_os=nxos` will try to load a plugin in a file named `nxos.py`, so it is important to name the plugin in a way that will be sensible to users.

Public methods of these plugins may be called from a module or `module_utils` with the connection proxy object just as other connection methods can. The following is a very simple example of using such a call in a `module_utils` file so it may be shared with other modules.

```
from ansible.module_utils.connection import Connection

def get_config(module):
    # module is your AnsibleModule instance.
    connection = Connection(module._socket_path)

    # You can now call any method (that doesn't start with '_') of the connection
    # plugin or its platform-specific plugin
    return connection.get_config()
```

- *Developing httpapi plugins*
 - *Making requests*
 - *Authenticating*
 - *Error handling*
- *Developing NETCONF plugins*
- *Developing network_cli plugins*

Developing httpapi plugins

httpapi plugins serve as adapters for various HTTP(S) APIs for use with the `httpapi` connection plugin. They should implement a minimal set of convenience methods tailored to the API you are attempting to use.

Specifically, there are a few methods that the `httpapi` connection plugin expects to exist.

Making requests

The `httpapi` connection plugin has a `send()` method, but an `httpapi` plugin needs a `send_request(self, data, **message_kwargs)` method as a higher-level wrapper to `send()`. This method should prepare requests by adding fixed values like common headers or URL root paths. This method may do more complex work such as turning data into formatted payloads, or determining which path or method to request. It may then also unpack responses to be more easily consumed by the caller.

```
from ansible.module_utils.six.moves.urllib.error import HTTPError

def send_request(self, data, path, method='POST'):
    # Fixed headers for requests
    headers = {'Content-Type': 'application/json'}
    try:
        response, response_content = self.connection.send(path, data, method=method,
↳headers=headers)
    except HTTPError as exc:
        return exc.code, exc.read()

    # handle_response (defined separately) will take the format returned by the device
    # and transform it into something more suitable for use by modules.
    # This may be JSON text to Python dictionaries, for example.
    return handle_response(response_content)
```

Authenticating

By default, all requests will authenticate with HTTP Basic authentication. If a request can return some kind of token to stand in place of HTTP Basic, the `update_auth(self, response, response_text)` method should be implemented to inspect responses for such tokens. If the token is meant to be included with the headers of each request, it is sufficient to return a dictionary which will be merged with the computed headers for each request. The default implementation of this method does exactly this for cookies. If the token is used in another way, say in a query string, you should instead save that token to an instance variable, where the `send_request()` method (above) can add it to each request

```
def update_auth(self, response, response_text):
    cookie = response.info().get('Set-Cookie')
    if cookie:
        return {'Cookie': cookie}

    return None
```

If instead an explicit login endpoint needs to be requested to receive an authentication token, the `login(self, username, password)` method can be implemented to call that endpoint. If implemented, this method will be called once before requesting any other resources of the server. By default, it will also be attempted once when a HTTP 401 is returned from a request.

```
def login(self, username, password):
    login_path = '/my/login/path'
    data = {'user': username, 'password': password}

    response = self.send_request(data, path=login_path)
    try:
        # This is still sent as an HTTP header, so we can set our connection's _auth
        # variable manually. If the token is returned to the device in another way,
        # you will have to keep track of it another way and make sure that it is sent
        # with the rest of the request from send_request()
        self.connection._auth = {'X-api-token': response['token']}
    except KeyError:
        raise AnsibleAuthenticationFailure(message="Failed to acquire login token.")
```

Similarly, `logout(self)` can be implemented to call an endpoint to invalidate and/or release the current token, if such an endpoint exists. This will be automatically called when the connection is closed (and, by extension, when reset).

```
def logout(self):
    logout_path = '/my/logout/path'
    self.send_request(None, path=logout_path)

    # Clean up tokens
    self.connection._auth = None
```

Error handling

The `handle_httperror(self, exception)` method can deal with status codes returned by the server. The return value indicates how the plugin will continue with the request:

- A value of `true` means that the request can be retried. This may be used to indicate a transient error, or one that has been resolved. For example, the default implementation will try to call `login()` when presented with a 401, and return `true` if successful.
- A value of `false` means that the plugin is unable to recover from this response. The status code will be returned to the calling module as an exception. Any other value will be taken as a nonfatal response from the request. This may be useful if the server returns error messages in the body of the response.

Returning the original exception is usually sufficient in this case, as `HTTPError` objects have the same interface as a successful response.

For example `httpapi` plugins, see the [source code for the `httpapi` plugins](#) included with Ansible Core.

Developing NETCONF plugins

The `netconf` connection plugin provides a connection to remote devices over the SSH NETCONF subsystem. Network devices typically use this connection plugin to send and receive RPC calls over NETCONF.

The `netconf` connection plugin uses the `ncclient` Python library under the hood to initiate a NETCONF session with a NETCONF-enabled remote network device. `ncclient` also executes NETCONF RPC requests and receives responses. You must install the `ncclient` on the local Ansible controller.

To use the `netconf` connection plugin for network devices that support standard NETCONF ([RFC 6241](#)) operations such as `get`, `get-config`, `edit-config`, set `ansible_network_os=default`. You can use `netconf_get`, `netconf_config` and `netconf_rpc` modules to talk to a NETCONF enabled remote host.

As a contributor and user, you should be able to use all the methods under the `NetconfBase` class if your device supports standard NETCONF. You can contribute a new plugin if the device you are working with has a vendor specific NETCONF RPC. To support a vendor specific NETCONF RPC, add the implementation in the network OS specific NETCONF plugin.

For Junos for example:

- See the vendor-specific Junos RPC methods implemented in `plugins/netconf/junos.py`.
- Set the value of `ansible_network_os` to the name of the netconf plugin file, that is `junos` in this case.

Developing network_cli plugins

The `network_cli` connection type uses `paramiko_ssh` under the hood which creates a pseudo terminal to send commands and receive responses. `network_cli` loads two platform specific plugins based on the value of `ansible_network_os`:

- Terminal plugin (for example `plugins/terminal/ios.py`) - Controls the parameters related to terminal, such as setting terminal length and width, page disabling and privilege escalation. Also defines regex to identify the command prompt and error prompts.
- *Cliconf Plugins* (for example, `ios cliconf`) - Provides an abstraction layer for low level send and receive operations. For example, the `edit_config()` method ensures that the prompt is in `config` mode before executing configuration commands.

To contribute a new network operating system to work with the `network_cli` connection, implement the `cliconf` and `terminal` plugins for that network OS.

The plugins can reside in:

- Adjacent to playbook in folders

```
cliconf_plugins/  
terminal_plugins/
```

- Roles

```
myrole/cliconf_plugins/  
myrole/terminal_plugins/
```

- Collections

```
myorg/mycollection/plugins/terminal/  
myorg/mycollection/plugins/cliconf/
```

The user can also set the `DEFAULT_CLICONF_PLUGIN_PATH` to configure the `cliconf` plugin path.

After adding the `cliconf` and `terminal` plugins in the expected locations, users can:

- Use the `cli_command` to run an arbitrary command on the network device.
- Use the `cli_config` to implement configuration changes on the remote hosts without platform-specific modules.

1.11.3 Documenting new network platforms

- *Modifying the platform options table*
- *Adding a platform-specific options section*
- *Adding your new file to the table of contents*

When you create network modules for a new platform, or modify the connections provided by an existing network platform(such as `network_cli` and `httpapi`), you also need to update the *Settings by Platform* table and add or modify the Platform Options file for your platform.

You should already have documented each module as described in *Module format and documentation*.

Modifying the platform options table

The *Settings by Platform* table is a convenient summary of the connections options provided by each network platform that has modules in Ansible. Add a row for your platform to this table, in alphabetical order. For example:

+-----+-----+-----+-----+-----+-----+					
↩ --+					
My OS	``myos``	✓	✓		✓ ↩
↩					

Ensure that the table stays formatted correctly. That is:

- Each row is inserted in alphabetical order.
- The cell division | markers line up with the + markers.
- The check marks appear only for the connection types provided by the network modules.

Adding a platform-specific options section

The platform- specific sections are individual `.rst` files that provide more detailed information for the users of your network platform modules. Name your new file `platform_<name>.rst` (for example, `platform_myos.rst`). The platform name should match the module prefix. See `platform_eos.rst` and *EOS Platform Options* for an example of the details you should provide in your platform-specific options section.

Your platform-specific section should include the following:

- **Connections available table** - a deeper dive into each connection type, including details on credentials, indirect access, connections settings, and enable mode.
- **How to use each connection type** - with working examples of each connection type.

If your network platform supports SSH connections, also include the following at the bottom of your `.rst` file:

```
.. include:: shared_snippets/SSH_warning.txt
```

Adding your new file to the table of contents

As a final step, add your new file in alphabetical order in the `platform_index.rst` file. You should then build the documentation to verify your additions. See *Contributing to the Ansible Documentation* for more details.

1.12 Galaxy User Guide

Ansible Galaxy refers to the [Galaxy](#) website, a free site for finding, downloading, and sharing community developed roles.

Use Galaxy to jump-start your automation project with great content from the Ansible community. Galaxy provides pre-packaged units of work such as *roles*, and new in Galaxy 3.2, *collections*. You can find roles

for provisioning infrastructure, deploying applications, and all of the tasks you do everyday. The collection format provides a comprehensive package of automation that may include multiple playbooks, roles, modules, and plugins.

- *Finding collections on Galaxy*
- *Installing collections*
 - *Installing a collection from Galaxy*
 - *Downloading a collection from Automation Hub*
 - *Installing an older version of a collection*
 - *Install multiple collections with a requirements file*
 - *Downloading a collection for offline use*
 - *Listing installed collections*
 - *Configuring the `ansible-galaxy` client*
- *Finding roles on Galaxy*
 - *Get more information about a role*
- *Installing roles from Galaxy*
 - *Installing roles*
 - *Installing a specific version of a role*
 - *Installing multiple roles from a file*
 - *Installing roles and collections from the same requirements.yml file*
 - *Installing multiple roles from multiple files*
 - *Dependencies*
 - *List installed roles*
 - *Remove an installed role*

1.12.1 Finding collections on Galaxy

To find collections on Galaxy:

1. Click the *Search* icon in the left-hand navigation.
2. Set the filter to *collection*.
3. Set other filters and press *enter*.

Galaxy presents a list of collections that match your search criteria.

1.12.2 Installing collections

Installing a collection from Galaxy

By default, `ansible-galaxy collection install` uses <https://galaxy.ansible.com> as the Galaxy server (as listed in the `ansible.cfg` file under `galaxy_server`). You do not need any further configuration.

See *Configuring the ansible-galaxy client* if you are using any other Galaxy server, such as Red Hat Automation Hub.

To install a collection hosted in Galaxy:

```
ansible-galaxy collection install my_namespace.my_collection
```

You can also directly use the tarball from your build:

```
ansible-galaxy collection install my_namespace-my_collection-1.0.0.tar.gz -p ./
↳ collections
```

注解: The install command automatically appends the path `ansible_collections` to the one specified with the `-p` option unless the parent directory is already in a folder called `ansible_collections`.

When using the `-p` option to specify the install path, use one of the values configured in `COLLECTIONS_PATHS`, as this is where Ansible itself will expect to find collections. If you don't specify a path, `ansible-galaxy collection install` installs the collection to the first path defined in `COLLECTIONS_PATHS`, which by default is `~/.ansible/collections`

You can also keep a collection adjacent to the current playbook, under a `collections/ansible_collections/` directory structure.

```
./
  play.yml
  collections/
    ansible_collections/
      my_namespace/
        my_collection/<collection structure lives here>
```

See *Collection structure* for details on the collection directory structure.

Downloading a collection from Automation Hub

You can download collections from Automation Hub at the command line. Automation Hub content is available to subscribers only, so you must download an API token and configure your local environment to provide it before you can you download collections. To download a collection from Automation Hub with the `ansible-galaxy` command:

1. Get your Automation Hub API token. Go to <https://cloud.redhat.com/ansible/automation-hub/token/> and click *Get API token* from the version dropdown to copy your API token.
2. Configure Red Hat Automation Hub server in the `server_list` option under the `[galaxy]` section in your `ansible.cfg` file.

```
[galaxy]
server_list = automation_hub

[galaxy_server.automation_hub]
url=https://cloud.redhat.com/api/automation-hub/
auth_url=https://sso.redhat.com/auth/realms/redhat-external/protocol/openid-
↪connect/token
token=my_ah_token
```

3. Download the collection hosted in Automation Hub.

```
ansible-galaxy collection install my_namespace.my_collection
```

参见:

Getting started with Automation Hub An introduction to Automation Hub

Installing an older version of a collection

You can only have one version of a collection installed at a time. By default `ansible-galaxy` installs the latest available version. If you want to install a specific version, you can add a version range identifier. For example, to install the 1.0.0-beta.1 version of the collection:

```
ansible-galaxy collection install my_namespace.my_collection:==1.0.0-beta.1
```

You can specify multiple range identifiers separated by `,`. Use single quotes so the shell passes the entire command, including `>`, `!`, and other operators, along. For example, to install the most recent version that is greater than or equal to 1.0.0 and less than 2.0.0:

```
ansible-galaxy collection install 'my_namespace.my_collection:>=1.0.0,<2.0.0'
```

Ansible will always install the most recent version that meets the range identifiers you specify. You can use the following range identifiers:

- *: The most recent version. This is the default.
- !=: Not equal to the version specified.
- ==: Exactly the version specified.
- >=: Greater than or equal to the version specified.
- >: Greater than the version specified.
- <=: Less than or equal to the version specified.
- <: Less than the version specified.

注解: By default `ansible-galaxy` ignores pre-release versions. To install a pre-release version, you must use the `==` range identifier to require it explicitly.

Install multiple collections with a requirements file

You can also setup a `requirements.yml` file to install multiple collections in one command. This file is a YAML file in the format:

```
---
collections:
  # With just the collection name
  - my_namespace.my_collection

  # With the collection name, version, and source options
  - name: my_namespace.my_other_collection
    version: 'version range identifiers (default: ``*``)'
    source: 'The Galaxy URL to pull the collection from (default: ``--api-server`` from
↳ cmdline)'
```

The `version` key can take in the same range identifier format documented above.

Roles can also be specified and placed under the `roles` key. The values follow the same format as a requirements file used in older Ansible releases.

```
---
roles:
  # Install a role from Ansible Galaxy.
  - name: geerlingguy.java
```

(下页继续)

```
version: 1.9.6

collections:
  # Install a collection from Ansible Galaxy.
  - name: geerlingguy.php_roles
    version: 0.9.3
    source: https://galaxy.ansible.com
```

注解: While both roles and collections can be specified in one requirements file, they need to be installed separately. The `ansible-galaxy role install -r requirements.yml` will only install roles and `ansible-galaxy collection install -r requirements.yml -p ./` will only install collections.

Downloading a collection for offline use

To download the collection tarball from Galaxy for offline use:

1. Navigate to the collection page.
2. Click on *Download tarball*.

You may also need to manually download any dependent collections.

Listing installed collections

To list installed collections, run `ansible-galaxy collection list`. See *Listing collections* for more details.

Configuring the ansible-galaxy client

By default, `ansible-galaxy` uses `https://galaxy.ansible.com` as the Galaxy server (as listed in the `ansible.cfg` file under `galaxy__server`).

You can use either option below to configure `ansible-galaxy collection` to use other servers (such as Red Hat Automation Hub or a custom Galaxy server):

- Set the server list in the `galaxy_server_list` configuration option in `ansible_configuration_settings_locations`.
- Use the `--server` command line argument to limit to an individual server.

To configure a Galaxy server list in `ansible.cfg`:

1. Add the `server_list` option under the `[galaxy]` section to one or more server names.
2. Create a new section for each server name.

3. Set the `url` option for each server name.
4. Optionally, set the API token for each server name. See *API token* for details.

注解: The `url` option for each server name must end with a forward slash `/`. If you do not set the API token in your Galaxy server list, use the `--api-key` argument to pass in the token to the `ansible-galaxy collection publish` command.

For Automation Hub, you additionally need to:

1. Set the `auth_url` option for each server name.
2. Set the API token for each server name. Go to <https://cloud.redhat.com/ansible/automation-hub/token/> and click `:Get API token` from the version dropdown to copy your API token.

The following example shows how to configure multiple servers:

```
[galaxy]
server_list = automation_hub, my_org_hub, release_galaxy, test_galaxy

[galaxy_server.automation_hub]
url=https://cloud.redhat.com/api/automation-hub/
auth_url=https://sso.redhat.com/auth/realms/redhat-external/protocol/openid-connect/token
token=my_ah_token

[galaxy_server.my_org_hub]
url=https://automation.my_org/
username=my_user
password=my_pass

[galaxy_server.release_galaxy]
url=https://galaxy.ansible.com/
token=my_token

[galaxy_server.test_galaxy]
url=https://galaxy-dev.ansible.com/
token=my_test_token
```

注解: You can use the `--server` command line argument to select an explicit Galaxy server in the `server_list` and the value of this argument should match the name of the server. To use a server not in the server list, set the value to the URL to access that server (all servers in the server list will be ignored). Also you cannot use the `--api-key` argument for any of the predefined servers. You can only use the `api_key`

argument if you did not define a server list or if you specify a URL in the `--server` argument.

Galaxy server list configuration options

The `galaxy_server_list` option is a list of server identifiers in a prioritized order. When searching for a collection, the install process will search in that order, for example, `automation_hub` first, then `my_org_hub`, `release_galaxy`, and finally `test_galaxy` until the collection is found. The actual Galaxy instance is then defined under the section `[galaxy_server.{{ id }}]` where `{{ id }}` is the server identifier defined in the list. This section can then define the following keys:

- **url**: The URL of the Galaxy instance to connect to. Required.
- **token**: An API token key to use for authentication against the Galaxy instance. Mutually exclusive with **username**.
- **username**: The username to use for basic authentication against the Galaxy instance. Mutually exclusive with **token**.
- **password**: The password to use, in conjunction with **username**, for basic authentication.
- **auth_url**: The URL of a Keycloak server 'token_endpoint' if using SSO authentication (for example, Automation Hub). Mutually exclusive with **username**. Requires **token**.

As well as defining these server options in the `ansible.cfg` file, you can also define them as environment variables. The environment variable is in the form `ANSIBLE_GALAXY_SERVER_{{ id }}_{{ key }}` where `{{ id }}` is the upper case form of the server identifier and `{{ key }}` is the key to define. For example I can define `token` for `release_galaxy` by setting `ANSIBLE_GALAXY_SERVER_RELEASE_GALAXY_TOKEN=secret_token`.

For operations that use only one Galaxy server (for example, the `publish`, `info`, or `install` commands), the `ansible-galaxy collection` command uses the first entry in the `server_list`, unless you pass in an explicit server with the `--server` argument.

注解: Once a collection is found, any of its requirements are only searched within the same Galaxy instance as the parent collection. The install process will not search for a collection requirement in a different Galaxy instance.

1.12.3 Finding roles on Galaxy

Search the Galaxy database by tags, platforms, author and multiple keywords. For example:

```
$ ansible-galaxy search elasticsearch --author geerlingguy
```

The search command will return a list of the first 1000 results matching your search:

Found 2 roles matching your search:

Name	Description
----	-----
geerlingguy.elasticsearch	Elasticsearch for Linux.
geerlingguy.elasticsearch-curator	Elasticsearch curator for Linux.

Get more information about a role

Use the `info` command to view more detail about a specific role:

```
$ ansible-galaxy info username.role_name
```

This returns everything found in Galaxy for the role:

```
Role: username.role_name
  description: Installs and configures a thing, a distributed, highly available NoSQL
  ↳ thing.
  active: True
  commit: c01947b7bc89ebc0b8a2e298b87ab416aed9dd57
  commit_message: Adding travis
  commit_url: https://github.com/username/repo_name/commit/
  ↳ c01947b7bc89ebc0b8a2e298b87ab
  company: My Company, Inc.
  created: 2015-12-08T14:17:52.773Z
  download_count: 1
  forks_count: 0
  github_branch:
  github_repo: repo_name
  github_user: username
  id: 6381
  is_valid: True
  issue_tracker_url:
  license: Apache
  min_ansible_version: 1.4
  modified: 2015-12-08T18:43:49.085Z
  namespace: username
  open_issues_count: 0
  path: /Users/username/projects/roles
  scm: None
```

(下页继续)

(续上页)

```
src: username.repo_name
stargazers_count: 0
travis_status_url: https://travis-ci.org/username/repo_name.svg?branch=master
version:
watchers_count: 1
```

1.12.4 Installing roles from Galaxy

The `ansible-galaxy` command comes bundled with Ansible, and you can use it to install roles from Galaxy or directly from a git based SCM. You can also use it to create a new role, remove roles, or perform tasks on the Galaxy website.

The command line tool by default communicates with the Galaxy website API using the server address *https://galaxy.ansible.com*. Since the [Galaxy project](#) is an open source project, you may be running your own internal Galaxy server and wish to override the default server address. You can do this using the `--server` option or by setting the Galaxy server value in your *ansible.cfg* file. For information on setting the value in *ansible.cfg* see `galaxy__server`.

Installing roles

Use the `ansible-galaxy` command to download roles from the [Galaxy website](#)

```
$ ansible-galaxy install namespace.role_name
```

Setting where to install roles

By default, Ansible downloads roles to the first writable directory in the default list of paths `~/.ansible/roles:/usr/share/ansible/roles:/etc/ansible/roles`. This installs roles in the home directory of the user running `ansible-galaxy`.

You can override this with one of the following options:

- Set the environment variable `ANSIBLE_ROLES_PATH` in your session.
- Define `roles_path` in an *ansible.cfg* file.
- Use the `--roles-path` option for the `ansible-galaxy` command.

The following provides an example of using `--roles-path` to install the role into the current working directory:

```
$ ansible-galaxy install --roles-path . geerlingguy.apache
```

参见:

配置 *Ansible* All about configuration files

Installing a specific version of a role

When the Galaxy server imports a role, it imports any git tags matching the Semantic Version format as versions. In turn, you can download a specific version of a role by specifying one of the imported tags.

To see the available versions for a role:

1. Locate the role on the Galaxy search page.
2. Click on the name to view more details, including the available versions.

You can also navigate directly to the role using the `/<namespace>/<role name>`. For example, to view the role `geerlingguy.apache`, go to <https://galaxy.ansible.com/geerlingguy/apache>.

To install a specific version of a role from Galaxy, append a comma and the value of a GitHub release tag. For example:

```
$ ansible-galaxy install geerlingguy.apache,v1.0.0
```

It is also possible to point directly to the git repository and specify a branch name or commit hash as the version. For example, the following will install a specific commit:

```
$ ansible-galaxy install git+https://github.com/geerlingguy/ansible-role-apache.git,  
↪0b7cd353c0250e87a26e0499e59e7fd265cc2f25
```

Installing multiple roles from a file

You can install multiple roles by including the roles in a `requirements.yml` file. The format of the file is YAML, and the file extension must be either `.yml` or `.yaml`.

Use the following command to install roles included in `requirements.yml`:

```
$ ansible-galaxy install -r requirements.yml
```

Again, the extension is important. If the `.yml` extension is left off, the `ansible-galaxy` CLI assumes the file is in an older, now deprecated, “basic” format.

Each role in the file will have one or more of the following attributes:

- src** The source of the role. Use the format `namespace.role_name`, if downloading from Galaxy; otherwise, provide a URL pointing to a repository within a git based SCM. See the examples below. This is a required attribute.

scm Specify the SCM. As of this writing only *git* or *hg* are allowed. See the examples below. Defaults to *git*.

version: The version of the role to download. Provide a release tag value, commit hash, or branch name. Defaults to the branch set as a default in the repository, otherwise defaults to the *master*.

name: Download the role to a specific name. Defaults to the Galaxy name when downloading from Galaxy, otherwise it defaults to the name of the repository.

Use the following example as a guide for specifying roles in *requirements.yml*:

```
# from galaxy
- name: yatesr.timezone

# from GitHub
- src: https://github.com/bennojoy/nginx

# from GitHub, overriding the name and specifying a specific tag
- name: nginx_role
  src: https://github.com/bennojoy/nginx
  version: master

# from GitHub, specifying a specific commit hash
- src: https://github.com/bennojoy/nginx
  version: "ee8aa41"

# from a webserver, where the role is packaged in a tar.gz
- name: http-role-gz
  src: https://some.webserver.example.com/files/master.tar.gz

# from a webserver, where the role is packaged in a tar.bz2
- name: http-role-bz2
  src: https://some.webserver.example.com/files/master.tar.bz2

# from a webserver, where the role is packaged in a tar.xz (Python 3.x only)
- name: http-role-xz
  src: https://some.webserver.example.com/files/master.tar.xz

# from Bitbucket
- src: git+https://bitbucket.org/willthames/git-ansible-galaxy
  version: v1.4
```

(下页继续)

(续上页)

```
# from Bitbucket, alternative syntax and caveats
- src: https://bitbucket.org/willthames/hg-ansible-galaxy
  scm: hg

# from GitLab or other git-based scm, using git+ssh
- src: git@gitlab.company.com:mygroup/ansible-base.git
  scm: git
  version: "0.1" # quoted, so YAML doesn't parse this as a floating-point value
```

Installing roles and collections from the same requirements.yml file

You can install roles and collections from the same requirements files, with some caveats.

```
---
roles:
  # Install a role from Ansible Galaxy.
  - name: geerlingguy.java
    version: 1.9.6

collections:
  # Install a collection from Ansible Galaxy.
  - name: geerlingguy.php_roles
    version: 0.9.3
    source: https://galaxy.ansible.com
```

注解: While both roles and collections can be specified in one requirements file, they need to be installed separately. The `ansible-galaxy role install -r requirements.yml` will only install roles and `ansible-galaxy collection install -r requirements.yml -p ./` will only install collections.

Installing multiple roles from multiple files

For large projects, the `include` directive in a `requirements.yml` file provides the ability to split a large file into multiple smaller files.

For example, a project may have a `requirements.yml` file, and a `webserver.yml` file.

Below are the contents of the `webserver.yml` file:

```
# from github
- src: https://github.com/bennojoy/nginx

# from Bitbucket
- src: git+http://bitbucket.org/willthames/git-ansible-galaxy
  version: v1.4
```

The following shows the contents of the `requirements.yml` file that now includes the `webserver.yml` file:

```
# from galaxy
- name: yatesr.timezone
- include: <path_to_requirements>/webserver.yml
```

To install all the roles from both files, pass the root file, in this case `requirements.yml` on the command line, as follows:

```
$ ansible-galaxy install -r requirements.yml
```

Dependencies

Roles can also be dependent on other roles, and when you install a role that has dependencies, those dependencies will automatically be installed.

You specify role dependencies in the `meta/main.yml` file by providing a list of roles. If the source of a role is Galaxy, you can simply specify the role in the format `namespace.role_name`. You can also use the more complex format in `requirements.yml`, allowing you to provide `src`, `scm`, `version`, and `name`.

The following shows an example `meta/main.yml` file with dependent roles:

```
---
dependencies:
  - geerlingguy.java

galaxy_info:
  author: geerlingguy
  description: Elasticsearch for Linux.
  company: "Midwestern Mac, LLC"
  license: "license (BSD, MIT)"
  min_ansible_version: 2.4
  platforms:
    - name: EL
      versions:
```

(下页继续)

(续上页)

```

- all
- name: Debian
  versions:
  - all
- name: Ubuntu
  versions:
  - all
galaxy_tags:
- web
- system
- monitoring
- logging
- lucene
- elk
- elasticsearch

```

Tags are inherited *down* the dependency chain. In order for tags to be applied to a role and all its dependencies, the tag should be applied to the role, not to all the tasks within a role.

Roles listed as dependencies are subject to conditionals and tag filtering, and may not execute fully depending on what tags and conditionals are applied.

If the source of a role is Galaxy, specify the role in the format *namespace.role_name*:

```

dependencies:
- geerlingguy.apache
- geerlingguy.ansible

```

Alternately, you can specify the role dependencies in the complex form used in `requirements.yml` as follows:

```

dependencies:
- name: geerlingguy.ansible
- name: composer
  src: git+https://github.com/geerlingguy/ansible-role-composer.git
  version: 775396299f2da1f519f0d8885022ca2d6ee80ee8

```

When dependencies are encountered by `ansible-galaxy`, it will automatically install each dependency to the `roles_path`. To understand how dependencies are handled during play execution, see [Roles](#).

注解: Galaxy expects all role dependencies to exist in Galaxy, and therefore dependencies to be specified in the `namespace.role_name` format. If you import a role with a dependency where the `src` value is a URL,

the import process will fail.

List installed roles

Use `list` to show the name and version of each role installed in the `roles_path`.

```
$ ansible-galaxy list
- ansible-network.network-engine, v2.7.2
- ansible-network.config_manager, v2.6.2
- ansible-network.cisco_nxos, v2.7.1
- ansible-network.vyos, v2.7.3
- ansible-network.cisco_ios, v2.7.0
```

Remove an installed role

Use `remove` to delete a role from `roles_path`:

```
$ ansible-galaxy remove namespace.role_name
```

参见:

Using collections Shareable collections of modules, playbooks and roles

Roles Reusable tasks, handlers, and other files in a known directory structure

1.13 Galaxy Developer Guide

You can host collections and roles on Galaxy to share with the Ansible community. Galaxy content is formatted in pre-packaged units of work such as *roles*, and new in Galaxy 3.2, *collections*. You can create roles for provisioning infrastructure, deploying applications, and all of the tasks you do everyday. Taking this a step further, you can create collections which provide a comprehensive package of automation that may include multiple playbooks, roles, modules, and plugins.

- *Creating collections for Galaxy*
- *Creating roles for Galaxy*
 - *Force*
 - *Container enabled*
 - *Using a custom role skeleton*

- *Authenticate with Galaxy*
- *Import a role*
- *Delete a role*
- *Travis integrations*

1.13.1 Creating collections for Galaxy

Collections are a distribution format for Ansible content. You can use collections to package and distribute playbooks, roles, modules, and plugins. You can publish and use collections through [Ansible Galaxy](#).

See *Developing collections* for details on how to create collections.

1.13.2 Creating roles for Galaxy

Use the `init` command to initialize the base structure of a new role, saving time on creating the various directories and `main.yml` files a role requires

```
$ ansible-galaxy init role_name
```

The above will create the following directory structure in the current working directory:

```
role_name/
  README.md
  .travis.yml
  defaults/
    main.yml
  files/
  handlers/
    main.yml
  meta/
    main.yml
  templates/
  tests/
    inventory
    test.yml
  vars/
    main.yml
```

If you want to create a repository for the role the repository root should be *role_name*.

Force

If a directory matching the name of the role already exists in the current working directory, the `init` command will result in an error. To ignore the error use the `-force` option. Force will create the above subdirectories and files, replacing anything that matches.

Container enabled

If you are creating a Container Enabled role, pass `--type container` to `ansible-galaxy init`. This will create the same directory structure as above, but populate it with default files appropriate for a Container Enabled role. For instance, the `README.md` has a slightly different structure, the `.travis.yml` file tests the role using [Ansible Container](#), and the meta directory includes a `container.yml` file.

Using a custom role skeleton

A custom role skeleton directory can be supplied as follows:

```
$ ansible-galaxy init --role-skeleton=/path/to/skeleton role_name
```

When a skeleton is provided, `init` will:

- copy all files and directories from the skeleton to the new role
- any `.j2` files found outside of a `templates` folder will be rendered as templates. The only useful variable at the moment is `role_name`
- The `.git` folder and any `.git_keep` files will not be copied

Alternatively, the `role_skeleton` and ignoring of files can be configured via `ansible.cfg`

```
[galaxy]
role_skeleton = /path/to/skeleton
role_skeleton_ignore = ^.git$,^.*/.git_keep$
```

Authenticate with Galaxy

Using the `import`, `delete` and `setup` commands to manage your roles on the Galaxy website requires authentication, and the `login` command can be used to do just that. Before you can use the `login` command, you must create an account on the Galaxy website.

The `login` command requires using your GitHub credentials. You can use your username and password, or you can create a [personal access token](#). If you choose to create a token, grant minimal access to the token, as it is used just to verify identity.

The following shows authenticating with the Galaxy website using a GitHub username and password:

```
$ ansible-galaxy login
```

We need your GitHub login to identify you.

This information will not be sent to Galaxy, only to api.github.com.

The password will not be displayed.

Use `--github-token` if you do not want to enter your password.

GitHub Username: dsmith

Password for dsmith:

Successfully logged into Galaxy as dsmith

When you choose to use your username and password, your password is not sent to Galaxy. It is used to authenticate with GitHub and create a personal access token. It then sends the token to Galaxy, which in turn verifies that your identity and returns a Galaxy access token. After authentication completes the GitHub token is destroyed.

If you do not wish to use your GitHub password, or if you have two-factor authentication enabled with GitHub, use the `-github-token` option to pass a personal access token that you create.

Import a role

The `import` command requires that you first authenticate using the `login` command. Once authenticated you can import any GitHub repository that you own or have been granted access.

Use the following to import to role:

```
$ ansible-galaxy import github_user github_repo
```

By default the command will wait for Galaxy to complete the import process, displaying the results as the import progresses:

```
Successfully submitted import request 41
Starting import 41: role_name=myrole repo=githubuser/ansible-role-repo ref=
Retrieving GitHub repo githubuser/ansible-role-repo
Accessing branch: master
Parsing and validating meta/main.yml
Parsing galaxy_tags
Parsing platforms
Adding dependencies
Parsing and validating README.md
Adding repo tags as role versions
```

(下页继续)

(续上页)

```
Import completed
Status SUCCESS : warnings=0 errors=0
```

Branch

Use the `-branch` option to import a specific branch. If not specified, the default branch for the repo will be used.

Role name

By default the name given to the role will be derived from the GitHub repository name. However, you can use the `-role-name` option to override this and set the name.

No wait

If the `-no-wait` option is present, the command will not wait for results. Results of the most recent import for any of your roles is available on the Galaxy web site by visiting *My Imports*.

Delete a role

The `delete` command requires that you first authenticate using the `login` command. Once authenticated you can remove a role from the Galaxy web site. You are only allowed to remove roles where you have access to the repository in GitHub.

Use the following to delete a role:

```
$ ansible-galaxy delete github_user github_repo
```

This only removes the role from Galaxy. It does not remove or alter the actual GitHub repository.

Travis integrations

You can create an integration or connection between a role in Galaxy and [Travis](#). Once the connection is established, a build in Travis will automatically trigger an import in Galaxy, updating the search index with the latest information about the role.

You create the integration using the `setup` command, but before an integration can be created, you must first authenticate using the `login` command; you will also need an account in Travis, and your Travis token. Once you're ready, use the following command to create the integration:

```
$ ansible-galaxy setup travis github_user github_repo xxx-travis-token-xxx
```

The setup command requires your Travis token, however the token is not stored in Galaxy. It is used along with the GitHub username and repo to create a hash as described in [the Travis documentation](#). The hash is stored in Galaxy and used to verify notifications received from Travis.

The setup command enables Galaxy to respond to notifications. To configure Travis to run a build on your repository and send a notification, follow the [Travis getting started guide](#).

To instruct Travis to notify Galaxy when a build completes, add the following to your `.travis.yml` file:

```
notifications:
  webhooks: https://galaxy.ansible.com/api/v1/notifications/
```

List Travis integrations

Use the `-list` option to display your Travis integrations:

```
$ ansible-galaxy setup --list
```

ID	Source	Repo
2	travis	github_user/github_repo
1	travis	github_user/github_repo

Remove Travis integrations

Use the `-remove` option to disable and remove a Travis integration:

```
$ ansible-galaxy setup --remove ID
```

Provide the ID of the integration to be disabled. You can find the ID by using the `-list` option.

参见:

[Using collections](#) Shareable collections of modules, playbooks and roles

[Roles](#) All about ansible roles

[Mailing List](#) Questions? Help? Ideas? Stop by the list on Google Groups

[irc.freenode.net](#) #ansible IRC chat channel

1.14 Controlling how Ansible behaves: precedence rules

To give you maximum flexibility in managing your environments, Ansible offers many ways to control how Ansible behaves: how it connects to managed nodes, how it works once it has connected. If you use Ansible to manage a large number of servers, network devices, and cloud resources, you may define Ansible behavior in several different places and pass that information to Ansible in several different ways. This flexibility is convenient, but it can backfire if you do not understand the precedence rules.

These precedence rules apply to any setting that can be defined in multiple ways (by configuration settings, command-line options, playbook keywords, variables).

- *Precedence categories*
 - *Configuration settings*
 - *Command-line options*
 - *Playbook keywords*
 - *Variables*
 - * *Variable scope: how long is a value available?*
 - *Using `-e` extra variables at the command line*

1.14.1 Precedence categories

Ansible offers four sources for controlling its behavior. In order of precedence from lowest (most easily overridden) to highest (overrides all others), the categories are:

- Configuration settings
- Command-line options
- Playbook keywords
- Variables

Each category overrides any information from all lower-precedence categories. For example, a playbook keyword will override any configuration setting.

Within each precedence category, specific rules apply. However, generally speaking, ‘last defined’ wins and overrides any previous definitions.

Configuration settings

Configuration settings include both values from the `ansible.cfg` file and environment variables. Within this category, values set in configuration files have lower precedence. Ansible uses the first `ansible.cfg` file

it finds, ignoring all others. Ansible searches for `ansible.cfg` in these locations in order:

- `ANSIBLE_CONFIG` (environment variable if set)
- `ansible.cfg` (in the current directory)
- `~/.ansible.cfg` (in the home directory)
- `/etc/ansible/ansible.cfg`

Environment variables have a higher precedence than entries in `ansible.cfg`. If you have environment variables set on your control node, they override the settings in whichever `ansible.cfg` file Ansible loads. The value of any given environment variable follows normal shell precedence: the last value defined overwrites previous values.

Command-line options

Any command-line option will override any configuration setting.

When you type something directly at the command line, you may feel that your hand-crafted values should override all others, but Ansible does not work that way. Command-line options have low precedence - they override configuration only. They do not override playbook keywords, variables from inventory or variables from playbooks.

You can override all other settings from all other sources in all other precedence categories at the command line by *Using -e extra variables at the command line*, but that is not a command-line option, it is a way of passing a *variable*.

At the command line, if you pass multiple values for a parameter that accepts only a single value, the last defined value wins. For example, this *ad-hoc task* will connect as `carol`, not as `mike`:

```
ansible -u mike -m ping myhost -u carol
```

Some parameters allow multiple values. In this case, Ansible will append all values from the hosts listed in inventory files `inventory1` and `inventory2`:

```
ansible -i /path/inventory1 -i /path/inventory2 -m ping all
```

The help for each *command-line tool* lists available options for that tool.

Playbook keywords

Any playbook keyword will override any command-line option and any configuration setting.

Within playbook keywords, precedence flows with the playbook itself; the more specific wins against the more general:

- `play` (most general)

- blocks/includes/imports/roles (optional and can contain tasks and each other)
- tasks (most specific)

A simple example:

```
- hosts: all
  connection: ssh
  tasks:
    - name: This task uses ssh.
      ping:

    - name: This task uses paramiko.
      connection: paramiko
      ping:
```

In this example, the `connection` keyword is set to `ssh` at the play level. The first task inherits that value, and connects using `ssh`. The second task inherits that value, overrides it, and connects using `paramiko`. The same logic applies to blocks and roles as well. All tasks, blocks, and roles within a play inherit play-level keywords; any task, block, or role can override any keyword by defining a different value for that keyword within the task, block, or role.

Remember that these are KEYWORDS, not variables. Both playbooks and variable files are defined in YAML but they have different significance. Playbooks are the command or ‘state description’ structure for Ansible, variables are data we use to help make playbooks more dynamic.

Variables

Any variable will override any playbook keyword, any command-line option, and any configuration setting.

Variables that have equivalent playbook keywords, command-line options, and configuration settings are known as *Connection variables*. Originally designed for connection parameters, this category has expanded to include other core variables like the temporary directory and the python interpreter.

Connection variables, like all variables, can be set in multiple ways and places. You can define variables for hosts and groups in *inventory*. You can define variables for tasks and plays in `vars:` blocks in *playbooks*. However, they are still variables - they are data, not keywords or configuration settings. Variables that override playbook keywords, command-line options, and configuration settings follow the same rules of *variable precedence* as any other variables.

When set in a playbook, variables follow the same inheritance rules as playbook keywords. You can set a value for the play, then override it in a task, block, or role:

```
- hosts: cloud
  gather_facts: false
```

(下页继续)

(续上页)

```

become: yes
vars:
    ansible_become_user: admin
tasks:
    - name: This task uses admin as the become user.
      dnf:
        name: some-service
        state: latest
    - block:
        - name: This task uses service-admin as the become user.
          # a task to configure the new service
        - name: This task also uses service-admin as the become user, defined in the
↪block.
          # second task to configure the service
      vars:
        ansible_become_user: service-admin
    - name: This task (outside of the block) uses admin as the become user again.
      service:
        name: some-service
        state: restarted

```

Variable scope: how long is a value available?

Variable values set in a playbook exist only within the playbook object that defines them. These ‘playbook object scope’ variables are not available to subsequent objects, including other plays.

Variable values associated directly with a host or group, including variables defined in inventory, by vars plugins, or using modules like `set_fact` and `include_vars`, are available to all plays. These ‘host scope’ variables are also available via the `hostvars[]` dictionary.

Using `-e` extra variables at the command line

To override all other settings in all other categories, you can use extra variables: `--extra-vars` or `-e` at the command line. Values passed with `-e` are variables, not command-line options, and they will override configuration settings, command-line options, and playbook keywords as well as variables set elsewhere. For example, this task will connect as `brian` not as `carol`:

```
ansible -u carol -e 'ansible_user=brian' -a whoami all
```

You must specify both the variable name and the value with `--extra-vars`.

1.15 YAML Syntax

This page provides a basic overview of correct YAML syntax, which is how Ansible playbooks (our configuration management language) are expressed.

We use YAML because it is easier for humans to read and write than other common data formats like XML or JSON. Further, there are libraries available in most programming languages for working with YAML.

You may also wish to read *Working With Playbooks* at the same time to see how this is used in practice.

1.15.1 YAML Basics

For Ansible, nearly every YAML file starts with a list. Each item in the list is a list of key/value pairs, commonly called a “hash” or a “dictionary”. So, we need to know how to write lists and dictionaries in YAML.

There’s another small quirk to YAML. All YAML files (regardless of their association with Ansible or not) can optionally begin with `---` and end with `...`. This is part of the YAML format and indicates the start and end of a document.

All members of a list are lines beginning at the same indentation level starting with a `-` (a dash and a space):

```
---
# A list of tasty fruits
- Apple
- Orange
- Strawberry
- Mango
...
```

A dictionary is represented in a simple `key: value` form (the colon must be followed by a space):

```
# An employee record
martin:
  name: Martin D'vloper
  job: Developer
  skill: Elite
```

More complicated data structures are possible, such as lists of dictionaries, dictionaries whose values are lists or a mix of both:

```
# Employee records
- martin:
```

(下页继续)

(续上页)

```

    name: Martin D'vloper
    job: Developer
    skills:
      - python
      - perl
      - pascal
- tabitha:
    name: Tabitha Bitumen
    job: Developer
    skills:
      - lisp
      - fortran
      - erlang

```

Dictionaries and lists can also be represented in an abbreviated form if you really want to:

```

---
martin: {name: Martin D'vloper, job: Developer, skill: Elite}
['Apple', 'Orange', 'Strawberry', 'Mango']

```

These are called “Flow collections” .

Ansible doesn’ t really use these too much, but you can also specify a boolean value (true/false) in several forms:

```

create_key: yes
needs_agent: no
knows_oop: True
likes_emacs: TRUE
uses_cvs: false

```

Use lowercase ‘true’ or ‘false’ for boolean values in dictionaries if you want to be compatible with default yamllint options.

Values can span multiple lines using | or >. Spanning multiple lines using a “Literal Block Scalar” | will include the newlines and any trailing spaces. Using a “Folded Block Scalar” > will fold newlines to spaces; it’ s used to make what would otherwise be a very long line easier to read and edit. In either case the indentation will be ignored. Examples are:

```

include_newlines: |
    exactly as you see
    will appear these three

```

(下页继续)

(续上页)

```
        lines of poetry

fold_newlines: >
    this is really a
    single line of text
    despite appearances
```

While in the above > example all newlines are folded into spaces, there are two ways to enforce a newline to be kept:

```
fold_some_newlines: >
    a
    b

    c
    d
    e
    f
same_as: "a b\nc d\n e\nf\n"
```

Let's combine what we learned so far in an arbitrary YAML example. This really has nothing to do with Ansible, but will give you a feel for the format:

```
---
# An employee record
name: Martin D'vloper
job: Developer
skill: Elite
employed: True
foods:
  - Apple
  - Orange
  - Strawberry
  - Mango
languages:
  perl: Elite
  python: Elite
  pascal: Lame
education: |
  4 GCSEs
  3 A-Levels
```

(下页继续)

(续上页)

BSc in the Internet of Things

That's all you really need to know about YAML to start writing *Ansible* playbooks.

1.15.2 Gotchas

While you can put just about anything into an unquoted scalar, there are some exceptions. A colon followed by a space (or newline) ": " is an indicator for a mapping. A space followed by the pound sign " #" starts a comment.

Because of this, the following is going to result in a YAML syntax error:

```
foo: somebody said I should put a colon here: so I did

windows_drive: c:
```

...but this will work:

```
windows_path: c:\windows
```

You will want to quote hash values using colons followed by a space or the end of the line:

```
foo: 'somebody said I should put a colon here: so I did'

windows_drive: 'c:'
```

...and then the colon will be preserved.

Alternatively, you can use double quotes:

```
foo: "somebody said I should put a colon here: so I did"

windows_drive: "c:"
```

The difference between single quotes and double quotes is that in double quotes you can use escapes:

```
foo: "a \t TAB and a \n NEWLINE"
```

The list of allowed escapes can be found in the YAML Specification under “Escape Sequences” (YAML 1.1) or “Escape Characters” (YAML 1.2).

The following is invalid YAML:

```
foo: "an escaped \' single quote"
```

Further, Ansible uses “`{{ var }}`” for variables. If a value after a colon starts with a “`{`”, YAML will think it is a dictionary, so you must quote it, like so:

```
foo: "{{ variable }}"
```

If your value starts with a quote the entire value must be quoted, not just part of it. Here are some additional examples of how to properly quote things:

```
foo: "{{ variable }}/additional/string/literal"
foo2: "{{ variable }}\\backslashes\\are\\also\\special\\characters"
foo3: "even if it's just a string literal it must all be quoted"
```

Not valid:

```
foo: "E:\\path\\"rest\\of\\path"
```

In addition to `'` and `"` there are a number of characters that are special (or reserved) and cannot be used as the first character of an unquoted scalar: `[] {} > | * & ! % # ` @ , .`

You should also be aware of `?` `:` `-`. In YAML, they are allowed at the beginning of a string if a non-space character follows, but YAML processor implementations differ, so it's better to use quotes.

In Flow Collections, the rules are a bit more strict:

```
a scalar in block mapping: this } is [ all , valid

flow mapping: { key: "you { should [ use , quotes here" }
```

Boolean conversion is helpful, but this can be a problem when you want a literal *yes* or other boolean values as a string. In these cases just use quotes:

```
non_boolean: "yes"
other_string: "False"
```

YAML converts certain strings into floating-point values, such as the string *1.0*. If you need to specify a version number (in a `requirements.yml` file, for example), you will need to quote the value if it looks like a floating-point value:

```
version: "1.0"
```

参见:

Working With Playbooks Learn what playbooks can do and how to write/run them.

YAMLLint [YAML Lint](#) (online) helps you debug YAML syntax if you are having problems

GitHub examples directory Complete playbook files from the github project source

Wikipedia YAML syntax reference A good guide to YAML syntax

Mailing List Questions? Help? Ideas? Stop by the list on Google Groups

irc.freenode.net #ansible IRC chat channel and #yaml for YAML specific questions

YAML 1.1 Specification The Specification for YAML 1.1, which PyYAML and libyaml are currently implementing

YAML 1.2 Specification For completeness, YAML 1.2 is the successor of 1.1

1.16 Python 3 Support

Ansible 2.5 and above work with Python 3. Previous to 2.5, using Python 3 was considered a tech preview. This topic discusses how to set up your controller and managed machines to use Python 3.

注解: On the controller we support Python 3.5 or greater and Python 2.7 or greater. Module-side, we support Python 3.5 or greater and Python 2.6 or greater.

1.16.1 On the controller side

The easiest way to run `/usr/bin/ansible` under Python 3 is to install it with the Python3 version of pip. This will make the default `/usr/bin/ansible` run with Python3:

```
$ pip3 install ansible
$ ansible --version | grep "python version"
python version = 3.6.2 (default, Sep 22 2017, 08:28:09) [GCC 7.2.1 20170915 (Red Hat 7.2.1-2)]
```

If you are running Ansible [从源码运行 Ansible \(devel\)](#) and want to use Python 3 with your source checkout, run your command via `python3`. For example:

```
$ source ./hacking/env-setup
$ python3 $(which ansible) localhost -m ping
$ python3 $(which ansible-playbook) sample-playbook.yml
```

注解: Individual Linux distribution packages may be packaged for Python2 or Python3. When running from distro packages you'll only be able to use Ansible with the Python version for which it was installed. Sometimes distros will provide a means of installing for several Python versions (via a separate package or

via some commands that are run after install). You' ll need to check with your distro to see if that applies in your case.

1.16.2 Using Python 3 on the managed machines with commands and playbooks

- Ansible will automatically detect and use Python 3 on many platforms that ship with it. To explicitly configure a Python 3 interpreter, set the `ansible_python_interpreter` inventory variable at a group or host level to the location of a Python 3 interpreter, such as `/usr/bin/python3`. The default interpreter path may also be set in `ansible.cfg`.

参见:

Interpreter Discovery for more information.

```
# Example inventory that makes an alias for localhost that uses Python3
localhost-py3 ansible_host=localhost ansible_connection=local ansible_python_
↪interpreter=/usr/bin/python3

# Example of setting a group of hosts to use Python3
[py3-hosts]
ubuntu16
fedora27

[py3-hosts:vars]
ansible_python_interpreter=/usr/bin/python3
```

参见:

Inventory 使用进阶 for more information.

- Run your command or playbook:

```
$ ansible localhost-py3 -m ping
$ ansible-playbook sample-playbook.yml
```

Note that you can also use the `-e` command line option to manually set the python interpreter when you run a command. This can be useful if you want to test whether a specific module or playbook has any bugs under Python 3. For example:

```
$ ansible localhost -m ping -e 'ansible_python_interpreter=/usr/bin/python3'
$ ansible-playbook sample-playbook.yml -e 'ansible_python_interpreter=/usr/bin/python3'
```


1.16.3 What to do if an incompatibility is found

We have spent several releases squashing bugs and adding new tests so that Ansible's core feature set runs under both Python 2 and Python 3. However, bugs may still exist in edge cases and many of the modules shipped with Ansible are maintained by the community and not all of those may be ported yet.

If you find a bug running under Python 3 you can submit a bug report on [Ansible's GitHub project](#). Be sure to mention Python3 in the bug report so that the right people look at it.

If you would like to fix the code and submit a pull request on github, you can refer to *Ansible and Python 3* for information on how we fix common Python3 compatibility issues in the Ansible codebase.

1.17 Interpreter Discovery

Most Ansible modules that execute under a POSIX environment require a Python interpreter on the target host. Unless configured otherwise, Ansible will attempt to discover a suitable Python interpreter on each target host the first time a Python module is executed for that host.

To control the discovery behavior:

- for individual hosts and groups, use the `ansible_python_interpreter` inventory variable
- globally, use the `interpreter_python` key in the `[defaults]` section of `ansible.cfg`

Use one of the following values:

auto_legacy [(default in 2.8)] Detects the target OS platform, distribution, and version, then consults a table listing the correct Python interpreter and path for each platform/distribution/version. If an entry is found, and `/usr/bin/python` is absent, uses the discovered interpreter (and path). If an entry is found, and `/usr/bin/python` is present, uses `/usr/bin/python` and issues a warning. This exception provides temporary compatibility with previous versions of Ansible that always defaulted to `/usr/bin/python`, so if you have installed Python and other dependencies at `/usr/bin/python` on some hosts, Ansible will find and use them with this setting. If no entry is found, or the listed Python is not present on the target host, searches a list of common Python interpreter paths and uses the first one found; also issues a warning that future installation of another Python interpreter could alter the one chosen.

auto [(future default in 2.12)] Detects the target OS platform, distribution, and version, then consults a table listing the correct Python interpreter and path for each platform/distribution/version. If an entry is found, uses the discovered interpreter. If no entry is found, or the listed Python is not present on the target host, searches a list of common Python interpreter paths and uses the first one found; also issues a warning that future installation of another Python interpreter could alter the one chosen.

auto_legacy_silent Same as `auto_legacy`, but does not issue warnings.

auto_silent Same as `auto`, but does not issue warnings.

You can still set `ansible_python_interpreter` to a specific path at any variable level (for example, in `host_vars`, in vars files, in playbooks, etc.). Setting a specific path completely disables automatic interpreter discovery; Ansible always uses the path specified.

1.18 Release and maintenance

- *Release cycle*
- *Release status*
- *Development and stable version maintenance workflow*
 - *Changelogs*
 - *Release candidates*
 - *Feature freeze*
- *Deprecation Cycle*

1.18.1 Release cycle

Ansible is developed and released on a flexible six month release cycle. This cycle can be extended in order to allow for larger changes to be properly implemented and tested before a new release is made available.

Ansible has a graduated maintenance structure that extends to three major releases. For more information, read about the *Development and stable version maintenance workflow* or see the chart in *Release status* for the degrees to which current releases are maintained.

If you are using a release of Ansible that is no longer maintained, we strongly encourage you to upgrade as soon as possible in order to benefit from the latest features and security fixes.

Older, unmaintained versions of Ansible can contain unfixed security vulnerabilities (*CVE*).

You can refer to the *porting guides* for tips on updating your Ansible playbooks to run on newer versions.

1.18.2 Release status

This table links to the release notes for each major release. These release notes (changelogs) contain the dates and significant changes in each minor release.

Ansible Release	Status
devel	In development (2.10 unreleased, trunk)
2.9 Release Notes	Maintained (security and general bug fixes)
2.8 Release Notes	Maintained (security and critical bug fixes)
2.7 Release Notes	Maintained (security fixes)
2.6 Release Notes	Unmaintained (end of life)
2.5 Release Notes	Unmaintained (end of life)
<2.5	Unmaintained (end of life)

You can download the releases from <https://releases.ansible.com/ansible/>.

注解: Ansible maintenance continues for 3 releases. Thus the latest Ansible release receives security and general bug fixes when it is first released, security and critical bug fixes when the next Ansible version is released, and **only** security fixes once the follow on to that version is released.

1.18.3 Development and stable version maintenance workflow

The Ansible community develops and maintains Ansible on [GitHub](#).

New modules, plugins, features and bugfixes will always be integrated in what will become the next major version of Ansible. This work is tracked on the `devel` git branch.

Ansible provides bugfixes and security improvements for the most recent major release. The previous major release will only receive fixes for security issues and critical bugs. Ansible only applies security fixes to releases which are two releases old. This work is tracked on the `stable-<version>` git branches.

The fixes that land in maintained stable branches will eventually be released as a new version when necessary.

Note that while there are no guarantees for providing fixes for Unmaintained releases of Ansible, there can sometimes be exceptions for critical issues.

Changelogs

Since Ansible 2.5, we have generated changelogs based on fragments. Here is the generated changelog for [2.9](#) as an example. When creating new features or fixing bugs, create a changelog fragment describing the change. A changelog entry is not needed for new modules or plugins. Details for those items will be generated from the module documentation.

We' ve got *examples and instructions on creating changelog fragments* in the Community Guide.

Older versions logged changes in `stable-<version>` branches at `stable-<version>/CHANGELOG.md`. For example, here is the changelog for [2.4](#) on GitHub.

Release candidates

Before a new release or version of Ansible can be done, it will typically go through a release candidate process.

This provides the Ansible community the opportunity to test Ansible and report bugs or issues they might come across.

Ansible tags the first release candidate (RC1) which is usually scheduled to last five business days. The final release is done if no major bugs or issues are identified during this period.

If there are major problems with the first candidate, a second candidate will be tagged (RC2) once the necessary fixes have landed. This second candidate lasts for a shorter duration than the first. If no problems have been reported after two business days, the final release is done.

More release candidates can be tagged as required, so long as there are bugs that the Ansible core maintainers consider should be fixed before the final release.

Feature freeze

While there is a pending release candidate, the focus of core developers and maintainers will on fixes towards the release candidate.

Merging new features or fixes that are not related to the release candidate may be delayed in order to allow the new release to be shipped as soon as possible.

1.18.4 Deprecation Cycle

Sometimes we need to remove a feature, normally in favor of a reimplementation that we hope does a better job. To do this we have a deprecation cycle. First we mark a feature as ‘deprecated’. This is normally accompanied with warnings to the user as to why we deprecated it, what alternatives they should switch to and when (which version) we are scheduled to remove the feature permanently.

The cycle is normally across 4 feature releases (2.x.y, where the x marks a feature release and the y a bugfix release), so the feature is normally removed in the 4th release after we announce the deprecation. For example, something deprecated in 2.7 will be removed in 2.11, assuming we don’t jump to 3.x before that point. The tracking is tied to the number of releases, not the release numbering.

For modules/plugins, we keep the documentation after the removal for users of older versions.

参见:

Committers Guidelines Guidelines for Ansible core contributors and maintainers

Testing Strategies Testing strategies

Ansible Community Guide Community information and contributing

Ansible release tarballs Ansible release tarballs

Development Mailing List Mailing list for development topics

irc.freenode.net #ansible IRC chat channel

1.19 Testing Strategies

1.19.1 Integrating Testing With Ansible Playbooks

Many times, people ask, “how can I best integrate testing with Ansible playbooks?” There are many options. Ansible is actually designed to be a “fail-fast” and ordered system, therefore it makes it easy to embed testing directly in Ansible playbooks. In this chapter, we’ ll go into some patterns for integrating tests of infrastructure and discuss the right level of testing that may be appropriate.

注解: This is a chapter about testing the application you are deploying, not the chapter on how to test Ansible modules during development. For that content, please hop over to the Development section.

By incorporating a degree of testing into your deployment workflow, there will be fewer surprises when code hits production and, in many cases, tests can be leveraged in production to prevent failed updates from migrating across an entire installation. Since it’ s push-based, it’ s also very easy to run the steps on the localhost or testing servers. Ansible lets you insert as many checks and balances into your upgrade workflow as you would like to have.

1.19.2 The Right Level of Testing

Ansible resources are models of desired-state. As such, it should not be necessary to test that services are started, packages are installed, or other such things. Ansible is the system that will ensure these things are declaratively true. Instead, assert these things in your playbooks.

```
tasks:
  - service:
      name: foo
      state: started
      enabled: yes
```

If you think the service may not be started, the best thing to do is request it to be started. If the service fails to start, Ansible will yell appropriately. (This should not be confused with whether the service is doing something functional, which we’ ll show more about how to do later).

1.19.3 Check Mode As A Drift Test

In the above setup, `-check` mode in Ansible can be used as a layer of testing as well. If running a deployment playbook against an existing system, using the `-check` flag to the `ansible` command will report if Ansible thinks it would have had to have made any changes to bring the system into a desired state.

This can let you know up front if there is any need to deploy onto the given system. Ordinarily scripts and commands don't run in check mode, so if you want certain steps to execute in normal mode even when the `-check` flag is used, such as calls to the script module, disable check mode for those tasks:

```
roles:
  - webserver

tasks:
  - script: verify.sh
    check_mode: no
```

1.19.4 Modules That Are Useful for Testing

Certain playbook modules are particularly good for testing. Below is an example that ensures a port is open:

```
tasks:

  - wait_for:
      host: "{{ inventory_hostname }}"
      port: 22
      delegate_to: localhost
```

Here's an example of using the URI module to make sure a web service returns:

```
tasks:

  - action: uri url=http://www.example.com return_content=yes
    register: webpage

  - fail:
      msg: 'service is not happy'
      when: "'AWESOME' not in webpage.content"
```

It's easy to push an arbitrary script (in any language) on a remote host and the script will automatically fail if it has a non-zero return code:

```
tasks:

- script: test_script1
- script: test_script2 --parameter value --parameter2 value
```

If using roles (you should be, roles are great!), scripts pushed by the script module can live in the ‘files/’ directory of a role.

And the assert module makes it very easy to validate various kinds of truth:

```
tasks:

- shell: /usr/bin/some-command --parameter value
  register: cmd_result

- assert:
  that:
    - "'not ready' not in cmd_result.stderr"
    - "'gizmo enabled' in cmd_result.stdout"
```

Should you feel the need to test for existence of files that are not declaratively set by your Ansible configuration, the ‘stat’ module is a great choice:

```
tasks:

- stat:
  path: /path/to/something
  register: p

- assert:
  that:
    - p.stat.exists and p.stat.isdir
```

As mentioned above, there’s no need to check things like the return codes of commands. Ansible is checking them automatically. Rather than checking for a user to exist, consider using the user module to make it exist.


Ansible is a fail-fast system, so when there is an error creating that user, it will stop the playbook run. You do not have to check up behind it.

1.19.5 Testing Lifecycle

If writing some degree of basic validation of your application into your playbooks, they will run every time you deploy.

As such, deploying into a local development VM and a staging environment will both validate that things are according to plan ahead of your production deploy.

Your workflow may be something like this:

- Use the same playbook all the time with embedded tests in development
- Use the playbook to deploy to a staging environment (with the same playbooks) that  simulates production
- Run an integration test battery written by your QA team against staging
- Deploy to production, with the same integrated tests.

Something like an integration test battery should be written by your QA team if you are a production webservice. This would include things like Selenium tests or automated API tests and would usually not be something embedded into your Ansible playbooks.

However, it does make sense to include some basic health checks into your playbooks, and in some cases it may be possible to run a subset of the QA battery against remote nodes. This is what the next section covers.

1.19.6 Integrating Testing With Rolling Updates

If you have read into *Delegation*, *Rolling Updates*, and *Local Actions* it may quickly become apparent that the rolling update pattern can be extended, and you can use the success or failure of the playbook run to decide whether to add a machine into a load balancer or not.

This is the great culmination of embedded tests:

```
---

- hosts: webserver
  serial: 5

  pre_tasks:

    - name: take out of load balancer pool
      command: /usr/bin/take_out_of_pool {{ inventory_hostname }}
      delegate_to: 127.0.0.1

  roles:
```

(下页继续)

(续上页)

```

- common
- webserver
- apply_testing_checks

post_tasks:

- name: add back to load balancer pool
  command: /usr/bin/add_back_to_pool {{ inventory_hostname }}
  delegate_to: 127.0.0.1

```

Of course in the above, the “take out of the pool” and “add back” steps would be replaced with a call to a Ansible load balancer module or appropriate shell command. You might also have steps that use a monitoring module to start and end an outage window for the machine.

However, what you can see from the above is that tests are used as a gate – if the “apply_testing_checks” step is not performed, the machine will not go back into the pool.

Read the delegation chapter about “max_fail_percentage” and you can also control how many failing tests will stop a rolling update from proceeding.

This above approach can also be modified to run a step from a testing machine remotely against a machine:

```

---

- hosts: webservers
  serial: 5

  pre_tasks:

    - name: take out of load balancer pool
      command: /usr/bin/take_out_of_pool {{ inventory_hostname }}
      delegate_to: 127.0.0.1

  roles:

    - common
    - webserver

  tasks:
    - script: /srv/qa_team/app_testing_script.sh --server {{ inventory_hostname }}
      delegate_to: testing_server

```

(下页继续)

(续上页)

```
post_tasks:

- name: add back to load balancer pool
  command: /usr/bin/add_back_to_pool {{ inventory_hostname }}
  delegate_to: 127.0.0.1
```

In the above example, a script is run from the testing server against a remote node prior to bringing it back into the pool.

In the event of a problem, fix the few servers that fail using Ansible's automatically generated retry file to repeat the deploy on just those servers.

1.19.7 Achieving Continuous Deployment

If desired, the above techniques may be extended to enable continuous deployment practices.

The workflow may look like this:

- Write and use automation to deploy local development VMs
- Have a CI system like Jenkins deploy to a staging environment on every code change
- The deploy job calls testing scripts to pass/fail a build on every deploy
- If the deploy job succeeds, it runs the same deploy playbook against production
↪ inventory

Some Ansible users use the above approach to deploy a half-dozen or dozen times an hour without taking all of their infrastructure offline. A culture of automated QA is vital if you wish to get to this level.

If you are still doing a large amount of manual QA, you should still make the decision on whether to deploy manually as well, but it can still help to work in the rolling update patterns of the previous section and incorporate some basic health checks using modules like 'script', 'stat', 'uri', and 'assert'.

1.19.8 Conclusion

Ansible believes you should not need another framework to validate basic things of your infrastructure is true. This is the case because Ansible is an order-based system that will fail immediately on unhandled errors for a host, and prevent further configuration of that host. This forces errors to the top and shows them in a summary at the end of the Ansible run.

However, as Ansible is designed as a multi-tier orchestration system, it makes it very easy to incorporate tests into the end of a playbook run, either using loose tasks or roles. When used with rolling updates, testing steps can decide whether to put a machine back into a load balanced pool or not.

Finally, because Ansible errors propagate all the way up to the return code of the Ansible program itself, and Ansible by default runs in an easy push-based mode, Ansible is a great step to put into a build environment if you wish to use it to roll out systems as part of a Continuous Integration/Continuous Delivery pipeline, as is covered in sections above.

The focus should not be on infrastructure testing, but on application testing, so we strongly encourage getting together with your QA team and ask what sort of tests would make sense to run every time you deploy development VMs, and which sort of tests they would like to run against the staging environment on every deploy. Obviously at the development stage, unit tests are great too. But don't unit test your playbook. Ansible describes states of resources declaratively, so you don't have to. If there are cases where you want to be sure of something though, that's great, and things like `stat/assert` are great go-to modules for that purpose.

In all, testing is a very organizational and site-specific thing. Everybody should be doing it, but what makes the most sense for your environment will vary with what you are deploying and who is using it – but everyone benefits from a more robust and reliable deployment system.

参见:

all_modules All the documentation for Ansible modules

Working With Playbooks An introduction to playbooks

Delegation, Rolling Updates, and Local Actions Delegation, useful for working with load balancers, clouds, and locally executed steps.

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

1.20 Frequently Asked Questions

Here are some commonly asked questions and their answers.

1.20.1 How can I set the PATH or any other environment variable for a task or entire playbook?

Setting environment variables can be done with the *environment* keyword. It can be used at the task or other levels in the play:

```
environment:
  PATH: "{{ ansible_env.PATH }}:/thingy/bin"
  SOME: value
```

注解: starting in 2.0.1 the setup task from gather_facts also inherits the environment directive from the play, you might need to use the `/default` filter to avoid errors if setting this at play level.

1.20.2 How do I handle different machines needing different user accounts or ports to log in with?

Setting inventory variables in the inventory file is the easiest way.

For instance, suppose these hosts have different usernames and ports:

```
[webservers]
asdf.example.com  ansible_port=5000  ansible_user=alice
jkl.example.com   ansible_port=5001  ansible_user=bob
```

You can also dictate the connection type to be used, if you want:

```
[testcluster]
localhost          ansible_connection=local
/path/to/chroot1    ansible_connection=chroot
foo.example.com     ansible_connection=paramiko
```

You may also wish to keep these in group variables instead, or file them in a `group_vars/<groupname>` file. See the rest of the documentation for more information about how to organize variables.

1.20.3 How do I get ansible to reuse connections, enable Kerberized SSH, or have Ansible pay attention to my local SSH config file?

Switch your default connection type in the configuration file to `'ssh'`, or use `'-c ssh'` to use Native OpenSSH for connections instead of the python paramiko library. In Ansible 1.2.1 and later, `'ssh'` will be used by default if OpenSSH is new enough to support ControlPersist as an option.

Paramiko is great for starting out, but the OpenSSH type offers many advanced options. You will want to run Ansible from a machine new enough to support ControlPersist, if you are using this connection type. You can still manage older clients. If you are using RHEL 6, CentOS 6, SLES 10 or SLES 11 the version of OpenSSH is still a bit old, so consider managing from a Fedora or openSUSE client even though you are managing older nodes, or just use paramiko.

We keep paramiko as the default as if you are first installing Ansible on an EL box, it offers a better experience for new users.

1.20.4 How do I configure a jump host to access servers that I have no direct access to?

You can set a *ProxyCommand* in the *ansible_ssh_common_args* inventory variable. Any arguments specified in this variable are added to the sftp/scp/ssh command line when connecting to the relevant host(s). Consider the following inventory group:

```
[gatewayed]
foo ansible_host=192.0.2.1
bar ansible_host=192.0.2.2
```

You can create *group_vars/gatewayed.yml* with the following contents:

```
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q user@gateway.example.com"'
```

Ansible will append these arguments to the command line when trying to connect to any hosts in the group *gatewayed*. (These arguments are used in addition to any *ssh_args* from *ansible.cfg*, so you do not need to repeat global *ControlPersist* settings in *ansible_ssh_common_args*.)

Note that *ssh -W* is available only with OpenSSH 5.4 or later. With older versions, it's necessary to execute *nc %h:%p* or some equivalent command on the bastion host.

With earlier versions of Ansible, it was necessary to configure a suitable *ProxyCommand* for one or more hosts in *~/.ssh/config*, or globally by setting *ssh_args* in *ansible.cfg*.

1.20.5 How do I get Ansible to notice a dead target in a timely manner?

You can add *-o ServerAliveInterval=NumberOfSeconds* in *ssh_args* from *ansible.cfg*. Without this option, SSH and therefore Ansible will wait until the TCP connection times out. Another solution is to add *ServerAliveInterval* into your global SSH configuration. A good value for *ServerAliveInterval* is up to you to decide; keep in mind that *ServerAliveCountMax=3* is the SSH default so any value you set will be tripled before terminating the SSH session.

1.20.6 How do I speed up management inside EC2?

Don't try to manage a fleet of EC2 machines from your laptop. Connect to a management node inside EC2 first and run Ansible from there.

1.20.7 How do I handle not having a Python interpreter at */usr/bin/python* on a remote machine?

While you can write Ansible modules in any language, most Ansible modules are written in Python, including the ones central to letting Ansible work.

By default, Ansible assumes it can find a `/usr/bin/python` on your remote system that is either Python2, version 2.6 or higher or Python3, 3.5 or higher.

Setting the inventory variable `ansible_python_interpreter` on any host will tell Ansible to auto-replace the Python interpreter with that value instead. Thus, you can point to any Python you want on the system if `/usr/bin/python` on your system does not point to a compatible Python interpreter.

Some platforms may only have Python 3 installed by default. If it is not installed as `/usr/bin/python`, you will need to configure the path to the interpreter via `ansible_python_interpreter`. Although most core modules will work with Python 3, there may be some special purpose ones which do not or you may encounter a bug in an edge case. As a temporary workaround you can install Python 2 on the managed host and configure Ansible to use that Python via `ansible_python_interpreter`. If there's no mention in the module's documentation that the module requires Python 2, you can also report a bug on our [bug tracker](#) so that the incompatibility can be fixed in a future release.

Do not replace the shebang lines of your python modules. Ansible will do this for you automatically at deploy time.

Also, this works for ANY interpreter, i.e ruby: `ansible_ruby_interpreter`, perl: `ansible_perl_interpreter`, etc, so you can use this for custom modules written in any scripting language and control the interpreter location.

Keep in mind that if you put `env` in your module shebang line (`#!/usr/bin/env <other>`), this facility will be ignored so you will be at the mercy of the remote `$PATH`.

1.20.8 How do I handle the package dependencies required by Ansible package dependencies during Ansible installation ?

While installing Ansible, sometimes you may encounter errors such as *No package 'libffi' found* or *fatal error: Python.h: No such file or directory*. These errors are generally caused by the missing packages, which are dependencies of the packages required by Ansible. For example, `libffi` package is dependency of `pynacl` and `paramiko` (Ansible -> paramiko -> pynacl -> libffi).

In order to solve these kinds of dependency issues, you might need to install required packages using the OS native package managers, such as `yum`, `dnf`, or `apt`, or as mentioned in the package installation guide.

Refer to the documentation of the respective package for such dependencies and their installation methods.

1.20.9 Common Platform Issues

What customer platforms does Red Hat support?

A number of them! For a definitive list please see this [Knowledge Base article](#).

Running in a virtualenv

You can install Ansible into a virtualenv on the controller quite simply:

```
$ virtualenv ansible
$ source ./ansible/bin/activate
$ pip install ansible
```

If you want to run under Python 3 instead of Python 2 you may want to change that slightly:

```
$ virtualenv -p python3 ansible
$ source ./ansible/bin/activate
$ pip install ansible
```

If you need to use any libraries which are not available via pip (for instance, SELinux Python bindings on systems such as Red Hat Enterprise Linux or Fedora that have SELinux enabled), then you need to install them into the virtualenv. There are two methods:

- When you create the virtualenv, specify `--system-site-packages` to make use of any libraries installed in the system's Python:

```
$ virtualenv ansible --system-site-packages
```

- Copy those files in manually from the system. For instance, for SELinux bindings you might do:

```
$ virtualenv ansible --system-site-packages
$ cp -r -v /usr/lib64/python3.*/site-packages/selinux/ ./py3-ansible/lib64/python3.
↪*/site-packages/
$ cp -v /usr/lib64/python3.*/site-packages/*selinux*.so ./py3-ansible/lib64/python3.
↪*/site-packages/
```

Running on BSD

参见:

Ansible and BSD

Running on Solaris

By default, Solaris 10 and earlier run a non-POSIX shell which does not correctly expand the default tmp directory Ansible uses (`~/ansible/tmp`). If you see module failures on Solaris machines, this is likely the problem. There are several workarounds:

- You can set `remote_tmp` to a path that will expand correctly with the shell you are using (see the plugin documentation for C shell, fish shell, and Powershell). For example, in the ansible config file you can set:

```
remote_tmp=$HOME/.ansible/tmp
```

In Ansible 2.5 and later, you can also set it per-host in inventory like this:

```
solaris1 ansible_remote_tmp=$HOME/.ansible/tmp
```

- You can set `ansible_shell_executable` to the path to a POSIX compatible shell. For instance, many Solaris hosts have a POSIX shell located at `/usr/xpg4/bin/sh` so you can set this in inventory like so:

```
solaris1 ansible_shell_executable=/usr/xpg4/bin/sh
```

(bash, ksh, and zsh should also be POSIX compatible if you have any of those installed).

Running on z/OS

There are a few common errors that one might run into when trying to execute Ansible on z/OS as a target.

- Version 2.7.6 of python for z/OS will not work with Ansible because it represents strings internally as EBCDIC.

To get around this limitation, download and install a later version of [python for z/OS](#) (2.7.13 or 3.6.1) that represents strings internally as ASCII. Version 2.7.13 is verified to work.

- When `pipelining = False` in `/etc/ansible/ansible.cfg` then Ansible modules are transferred in binary mode via sftp however execution of python fails with

```
错误: SyntaxError: Non-UTF-8 code starting with '\x83' in file /a/user1/.ansible/tmp/ansible-tmp-1548232945.35-274513842609025/AnsiballZ_stat.py on line 1, but no encoding declared; see https://python.org/dev/peps/pep-0263/ for details
```

To fix it set `pipelining = True` in `/etc/ansible/ansible.cfg`.

- Python interpret cannot be found in default location `/usr/bin/python` on target host.

```
错误: /usr/bin/python: EDC5129I No such file or directory
```

To fix this set the path to the python installation in your inventory like so:


```
zos1 ansible_python_interpreter=/usr/lpp/python/python-2017-04-12-py27/python27/bin/
↪python
```

- Start of python fails with The module libpython2.7.so was not found.

错误: EE3501S The module libpython2.7.so was not found.

On z/OS, you must execute python from gnu bash. If gnu bash is installed at `/usr/lpp/bash`, you can fix this in your inventory by specifying an `ansible_shell_executable`:

```
zos1 ansible_shell_executable=/usr/lpp/bash/bin/bash
```

1.20.10 What is the best way to make content reusable/redistributable?

If you have not done so already, read all about “Roles” in the playbooks documentation. This helps you make playbook content self-contained, and works well with things like git submodules for sharing content with others.

If some of these plugin types look strange to you, see the API documentation for more details about ways Ansible can be extended.

1.20.11 Where does the configuration file live and what can I configure in it?

See 配置 *Ansible*.

1.20.12 How do I disable cowsay?

If cowsay is installed, Ansible takes it upon itself to make your day happier when running playbooks. If you decide that you would like to work in a professional cow-free environment, you can either uninstall cowsay, set `nocows=1` in `ansible.cfg`, or set the `ANSIBLE_NOCOWS` environment variable:

```
export ANSIBLE_NOCOWS=1
```

1.20.13 How do I see a list of all of the ansible_ variables?

Ansible by default gathers “facts” about the machines under management, and these facts can be accessed in Playbooks and in templates. To see a list of all of the facts that are available about a machine, you can run the “setup” module as an ad-hoc action:

```
ansible -m setup hostname
```

This will print out a dictionary of all of the facts that are available for that particular host. You might want to pipe the output to a pager. This does NOT include inventory variables or internal ‘magic’ variables. See the next question if you need more than just ‘facts’ .

1.20.14 How do I see all the inventory variables defined for my host?

By running the following command, you can see inventory variables for a host:

```
ansible-inventory --list --yaml
```

1.20.15 How do I see all the variables specific to my host?

To see all host specific variables, which might include facts and other sources:

```
ansible -m debug -a "var=hostvars['hostname']" localhost
```

Unless you are using a fact cache, you normally need to use a play that gathers facts first, for facts included in the task above.

1.20.16 How do I loop over a list of hosts in a group, inside of a template?

A pretty common pattern is to iterate over a list of hosts inside of a host group, perhaps to populate a template configuration file with a list of servers. To do this, you can just access the “\$groups” dictionary in your template, like this:

```
{% for host in groups['db_servers'] %}
    {{ host }}
{% endfor %}
```

If you need to access facts about these hosts, for instance, the IP address of each hostname, you need to make sure that the facts have been populated. For example, make sure you have a play that talks to db_servers:

```
- hosts: db_servers
  tasks:
    - debug: msg="doesn't matter what you do, just that they were talked to previously."
```

Then you can use the facts inside your template, like this:

```
{% for host in groups['db_servers'] %}
    {{ hostvars[host]['ansible_eth0']['ipv4']['address'] }}
{% endfor %}
```

1.20.17 How do I access a variable name programmatically?

An example may come up where we need to get the ipv4 address of an arbitrary interface, where the interface to be used may be supplied via a role parameter or other input. Variable names can be built by adding strings together, like so:

```
{{ hostvars[inventory_hostname]['ansible_' + which_interface]['ipv4']['address'] }}
```

The trick about going through `hostvars` is necessary because it's a dictionary of the entire namespace of variables. `inventory_hostname` is a magic variable that indicates the current host you are looping over in the host loop.

In the example above, if your interface names have dashes, you must replace them with underscores:

```
{{ hostvars[inventory_hostname]['ansible_' + which_interface | replace('-', '_') ]['ipv4'
→ ]['address'] }}
```

Also see *dynamic_variables*.

1.20.18 How do I access a group variable?

Technically, you don't, Ansible does not really use groups directly. Groups are label for host selection and a way to bulk assign variables, they are not a first class entity, Ansible only cares about Hosts and Tasks.

That said, you could just access the variable by selecting a host that is part of that group, see *first_host_in_a_group* below for an example.

1.20.19 How do I access a variable of the first host in a group?

What happens if we want the ip address of the first webserver in the webserver group? Well, we can do that too. Note that if we are using dynamic inventory, which host is the 'first' may not be consistent, so you wouldn't want to do this unless your inventory is static and predictable. (If you are using *Red Hat Ansible Tower*, it will use database order, so this isn't a problem even if you are using cloud based inventory scripts).

Anyway, here's the trick:

```
{{ hostvars[groups['webserver'][0]]['ansible_eth0']['ipv4']['address'] }}
```

Notice how we’re pulling out the hostname of the first machine of the webservers group. If you are doing this in a template, you could use the Jinja2 ‘#set’ directive to simplify this, or in a playbook, you could also use `set_fact`:

```
- set_fact: headnode={{ groups[['webservers']][0] }}

- debug: msg={{ hostvars[headnode].ansible_eth0.ipv4.address }}
```

Notice how we interchanged the bracket syntax for dots – that can be done anywhere.

1.20.20 How do I copy files recursively onto a target host?

The “copy” module has a recursive parameter. However, take a look at the “synchronize” module if you want to do something more efficient for a large number of files. The “synchronize” module wraps `rsync`. See the module index for info on both of these modules.

1.20.21 How do I access shell environment variables?

If you just need to access existing variables ON THE CONTROLLER, use the ‘env’ lookup plugin. For example, to access the value of the HOME environment variable on the management machine:

```
---
# ...
vars:
    local_home: "{{ lookup('env', 'HOME') }}"
```

For environment variables on the TARGET machines, they are available via facts in the ‘ansible_env’ variable:

```
{{ ansible_env.SOME_VARIABLE }}
```

If you need to set environment variables for TASK execution, see *Setting the Environment (and Working With Proxies)* in the *Advanced Playbooks* section. There are several ways to set environment variables on your target machines. You can use the template, replace, or lineinfile modules to introduce environment variables into files. The exact files to edit vary depending on your OS and distribution and local configuration.

1.20.22 How do I generate encrypted passwords for the user module?

Ansible ad-hoc command is the easiest option:

```
ansible all -i localhost, -m debug -a "msg={{ 'mypassword' | password_hash('sha512',
↪ 'mysecretsalt') }}"
```

The `mkpasswd` utility that is available on most Linux systems is also a great option:

```
mkpasswd --method=sha-512
```

If this utility is not installed on your system (e.g. you are using macOS) then you can still easily generate these passwords using Python. First, ensure that the [Passlib](#) password hashing library is installed:

```
pip install passlib
```

Once the library is ready, SHA512 password values can then be generated as follows:

```
python -c "from passlib.hash import sha512_crypt; import getpass; print(sha512_crypt.
↳using(rounds=5000).hash(getpass.getpass()))"
```

Use the integrated [Encryption filters](#) to generate a hashed version of a password. You shouldn't put plaintext passwords in your playbook or `host_vars`; instead, use [Using Vault in playbooks](#) to encrypt sensitive data.

In OpenBSD, a similar option is available in the base system called `encrypt(1)`:

```
encrypt
```

1.20.23 Ansible allows dot notation and array notation for variables. Which notation should I use?

The dot notation comes from Jinja and works fine for variables without special characters. If your variable contains dots (`.`), colons (`:`), or dashes (`-`), if a key begins and ends with two underscores, or if a key uses any of the known public attributes, it is safer to use the array notation. See [Using Variables](#) for a list of the known public attributes.

```
item[0]['checksum:md5']
item['section']['2.1']
item['region']['Mid-Atlantic']
It is {{ temperature['Celsius']['-3'] }} outside.
```

Also array notation allows for dynamic variable composition, see [dynamic_variables](#).

Another problem with 'dot notation' is that some keys can cause problems because they collide with attributes and methods of python dictionaries.

```
item.update # this breaks if item is a dictionary, as 'update()' is a python method for
↳dictionaries
item['update'] # this works
```

1.20.24 When is it unsafe to bulk-set task arguments from a variable?

You can set all of a task's arguments from a dictionary-typed variable. This technique can be useful in some dynamic execution scenarios. However, it introduces a security risk. We do not recommend it, so Ansible issues a warning when you do something like this:

```
#...
vars:
  usermod_args:
    name: testuser
    state: present
    update_password: always
tasks:
- user: '{{ usermod_args }}'
```

This particular example is safe. However, constructing tasks like this is risky because the parameters and values passed to `usermod_args` could be overwritten by malicious values in the `host facts` on a compromised target machine. To mitigate this risk:

- set bulk variables at a level of precedence greater than `host facts` in the order of precedence found in *Variable precedence: Where should I put a variable?* (the example above is safe because play vars take precedence over facts)
- disable the `inject_facts_as_vars` configuration setting to prevent fact values from colliding with variables (this will also disable the original warning)

1.20.25 Can I get training on Ansible?

Yes! See our [services](#) page for information on our services and training offerings. Email info@ansible.com for further details.

We also offer free web-based training classes on a regular basis. See our [webinar](#) page for more info on upcoming webinars.

1.20.26 Is there a web interface / REST API / etc?

Yes! Ansible, Inc makes a great product that makes Ansible even more powerful and easy to use. See *Red Hat Ansible Tower*.

1.20.27 How do I submit a change to the documentation?

Great question! Documentation for Ansible is kept in the main project git repository, and complete instructions for contributing can be found in the docs README [viewable on GitHub](#). Thanks!

1.20.28 How do I keep secret data in my playbook?

If you would like to keep secret data in your Ansible content and still share it publicly or keep things in source control, see *Using Vault in playbooks*.

If you have a task that you don't want to show the results or command given to it when using -v (verbose) mode, the following task or playbook attribute can be useful:

```
- name: secret task
  shell: /usr/bin/do_something --value={{ secret_value }}
  no_log: True
```

This can be used to keep verbose output but hide sensitive information from others who would otherwise like to be able to see the output.

The no_log attribute can also apply to an entire play:

```
- hosts: all
  no_log: True
```

Though this will make the play somewhat difficult to debug. It's recommended that this be applied to single tasks only, once a playbook is completed. Note that the use of the no_log attribute does not prevent data from being shown when debugging Ansible itself via the ANSIBLE_DEBUG environment variable.

1.20.29 When should I use {{ }}? Also, how to interpolate variables or dynamic variable names

A steadfast rule is 'always use {{ }} except when when:'. Conditionals are always run through Jinja2 as to resolve the expression, so when:, failed_when: and changed_when: are always templated and you should avoid adding {{ }}.

In most other cases you should always use the brackets, even if previously you could use variables without specifying (like loop or with_ clauses), as this made it hard to distinguish between an undefined variable and a string.

Another rule is 'moustaches don't stack'. We often see this:

```
{{ somevar_{{other_var}} }}
```

The above DOES NOT WORK as you expect, if you need to use a dynamic variable use the following as appropriate:

```
{{ hostvars[inventory_hostname]['somevar_' + other_var] }}
```

For 'non host vars' you can use the vars lookup plugin:

```
{{ lookup('vars', 'somevar_' + other_var) }}
```

1.20.30 Why don't you ship in X format?

In most cases it has to do with maintainability. There are many ways to ship software and we do not have the resources to release Ansible on every platform. In some cases there are technical issues. For example, our dependencies are not present on Python Wheels.

1.20.31 How do I get the original `ansible_host` when I delegate a task?

As the documentation states, connection variables are taken from the `delegate_to` host so `ansible_host` is overwritten, but you can still access the original via `hostvars`:

```
original_host: "{{ hostvars[inventory_hostname]['ansible_host'] }}"
```

This works for all overridden connection variables, like `ansible_user`, `ansible_port`, etc.

1.20.32 How do I fix 'protocol error: filename does not match request' when fetching a file?

Newer releases of OpenSSH have a [bug](#) in the SCP client that can trigger this error on the Ansible controller when using SCP as the file transfer mechanism:

```
failed to transfer file to /tmp/ansible/file.txt\r\nprotocol error: filename does not
↪match request
```

In these releases, SCP tries to validate that the path of the file to fetch matches the requested path. The validation fails if the remote filename requires quotes to escape spaces or non-ascii characters in its path. To avoid this error:

- Use SFTP instead of SCP by setting `scp_if_ssh` to `smart` (which tries SFTP first) or to `False`. You can
 - Rely on the default setting, which is `smart` - this works if `scp_if_ssh` is not explicitly set anywhere
 - Set a *host variable* or *group variable* in inventory: `ansible_scp_if_ssh: False`
 - Set an environment variable on your control node: `export ANSIBLE_SCP_IF_SSH=False`
 - Pass an environment variable when you run Ansible: `ANSIBLE_SCP_IF_SSH=smart ansible-playbook`
 - Modify your `ansible.cfg` file: add `scp_if_ssh=False` to the `[ssh_connection]` section

- If you must use SCP, set the `-T` arg to tell the SCP client to ignore path validation. You can do this in
 - Set a *host variable* or *group variable*: `ansible_scp_extra_args=-T`,
 - Export or pass an environment variable: `ANSIBLE_SCP_EXTRA_ARGS=-T`
 - Modify your `ansible.cfg` file: add `scp_extra_args=-T` to the `[ssh_connection]` section

注解: If you see an `invalid argument` error when using `-T`, then your SCP client is not performing filename validation and will not trigger this error.

1.20.33 I don' t see my question here

Please see the section below for a link to IRC and the Google Group, where you can ask your question there.

参见:

Working With Playbooks An introduction to playbooks

Tips and tricks Best practices advice

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

1.21 Glossary

The following is a list (and re-explanation) of term definitions used elsewhere in the Ansible documentation.

Consult the documentation home page for the full documentation and to see the terms in context, but this should be a good resource to check your knowledge of Ansible' s components and understand how they fit together. It' s something you might wish to read for review or when a term comes up on the mailing list.

Action An action is a part of a task that specifies which of the modules to run and which arguments to pass to that module. Each task can have only one action, but it may also have other parameters.

Ad Hoc Refers to running Ansible to perform some quick command, using `/usr/bin/ansible`, rather than the *orchestration* language, which is `/usr/bin/ansible-playbook`. An example of an ad hoc command might be rebooting 50 machines in your infrastructure. Anything you can do ad hoc can be accomplished by writing a *playbook* and playbooks can also glue lots of other operations together.

Async Refers to a task that is configured to run in the background rather than waiting for completion. If you have a long process that would run longer than the SSH timeout, it would make sense to launch that task in async mode. Async modes can poll for completion every so many seconds or can be configured to “fire and forget” , in which case Ansible will not even check on the task again; it will

just kick it off and proceed to future steps. Async modes work with both `/usr/bin/ansible` and `/usr/bin/ansible-playbook`.

Callback Plugin Refers to some user-written code that can intercept results from Ansible and do something with them. Some supplied examples in the GitHub project perform custom logging, send email, or even play sound effects.

Check Mode Refers to running Ansible with the `--check` option, which does not make any changes on the remote systems, but only outputs the changes that might occur if the command ran without this flag. This is analogous to so-called “dry run” modes in other systems, though the user should be warned that this does not take into account unexpected command failures or cascade effects (which is true of similar modes in other systems). Use this to get an idea of what might happen, but do not substitute it for a good staging environment.

Connection Plugin By default, Ansible talks to remote machines through pluggable libraries. Ansible uses native OpenSSH (*SSH (Native)*) or a Python implementation called *paramiko*. OpenSSH is preferred if you are using a recent version, and also enables some features like Kerberos and jump hosts. This is covered in the *getting started section*. There are also other connection types like `accelerate` mode, which must be bootstrapped over one of the SSH-based connection types but is very fast, and local mode, which acts on the local system. Users can also write their own connection plugins.

Conditionals A conditional is an expression that evaluates to true or false that decides whether a given task is executed on a given machine or not. Ansible’s conditionals are powered by the ‘when’ statement, which are discussed in the *Working With Playbooks*.

Declarative An approach to achieving a task that uses a description of the final state rather than a description of the sequence of steps necessary to achieve that state. For a real world example, a declarative specification of a task would be: “put me in California” . Depending on your current location, the sequence of steps to get you to California may vary, and if you are already in California, nothing at all needs to be done. Ansible’s Resources are declarative; it figures out the steps needed to achieve the final state. It also lets you know whether or not any steps needed to be taken to get to the final state.

Diff Mode A `--diff` flag can be passed to Ansible to show what changed on modules that support it. You can combine it with `--check` to get a good ‘dry run’ . File diffs are normally in unified diff format.

Executor A core software component of Ansible that is the power behind `/usr/bin/ansible` directly – and corresponds to the invocation of each task in a *playbook*. The Executor is something Ansible developers may talk about, but it’s not really user land vocabulary.

Facts Facts are simply things that are discovered about remote nodes. While they can be used in *playbooks* and templates just like variables, facts are things that are inferred, rather than set. Facts are automatically discovered by Ansible when running plays by executing the internal setup module on the remote nodes. You never have to call the setup module explicitly, it just runs, but it can be disabled to save time if it is not needed or you can tell ansible to collect only a subset of the full facts via the `gather_subset:` option. For the convenience of users who are switching from other configuration

management systems, the fact module will also pull in facts from the **ohai** and **facter** tools if they are installed. These are fact libraries from Chef and Puppet, respectively. (These may also be disabled via `gather_subset:`)

Filter Plugin A filter plugin is something that most users will never need to understand. These allow for the creation of new *Jinja2* filters, which are more or less only of use to people who know what Jinja2 filters are. If you need them, you can learn how to write them in the *API docs section*.

Forks Ansible talks to remote nodes in parallel and the level of parallelism can be set either by passing `--forks` or editing the default in a configuration file. The default is a very conservative five (5) forks, though if you have a lot of RAM, you can easily set this to a value like 50 for increased parallelism.

Gather Facts (Boolean) *Facts* are mentioned above. Sometimes when running a multi-play *playbook*, it is desirable to have some plays that don't bother with fact computation if they aren't going to need to utilize any of these values. Setting `gather_facts: False` on a playbook allows this implicit fact gathering to be skipped.

Globbering Globbing is a way to select lots of hosts based on wildcards, rather than the name of the host specifically, or the name of the group they are in. For instance, it is possible to select `ww*` to match all hosts starting with `ww`. This concept is pulled directly from **Func**, one of Michael DeHaan's (an Ansible Founder) earlier projects. In addition to basic globbing, various set operations are also possible, such as 'hosts in this group and not in another group', and so on.

Group A group consists of several hosts assigned to a pool that can be conveniently targeted together, as well as given variables that they share in common.

Group Vars The `group_vars/` files are files that live in a directory alongside an inventory file, with an optional filename named after each group. This is a convenient place to put variables that are provided to a given group, especially complex data structures, so that these variables do not have to be embedded in the *inventory* file or *playbook*.

Handlers Handlers are just like regular tasks in an Ansible *playbook* (see *Tasks*) but are only run if the Task contains a **notify** directive and also indicates that it changed something. For example, if a config file is changed, then the task referencing the config file templating operation may notify a service restart handler. This means services can be bounced only if they need to be restarted. Handlers can be used for things other than service restarts, but service restarts are the most common usage.

Host A host is simply a remote machine that Ansible manages. They can have individual variables assigned to them, and can also be organized in groups. All hosts have a name they can be reached at (which is either an IP address or a domain name) and, optionally, a port number, if they are not to be accessed on the default SSH port.

Host Specifier Each *Play* in Ansible maps a series of *tasks* (which define the role, purpose, or orders of a system) to a set of systems.

This `hosts:` directive in each play is often called the hosts specifier.

It may select one system, many systems, one or more groups, or even some hosts that are in one group

and explicitly not in another.

Host Vars Just like *Group Vars*, a directory alongside the inventory file named `host_vars/` can contain a file named after each hostname in the inventory file, in *YAML* format. This provides a convenient place to assign variables to the host without having to embed them in the *inventory* file. The Host Vars file can also be used to define complex data structures that can't be represented in the inventory file.

Idempotency An operation is idempotent if the result of performing it once is exactly the same as the result of performing it repeatedly without any intervening actions.

Includes The idea that *playbook* files (which are nothing more than lists of *plays*) can include other lists of plays, and task lists can externalize lists of *tasks* in other files, and similarly with *handlers*. Includes can be parameterized, which means that the loaded file can pass variables. For instance, an included play for setting up a WordPress blog may take a parameter called `user` and that play could be included more than once to create a blog for both `alice` and `bob`.

Inventory A file (by default, Ansible uses a simple INI format) that describes *Hosts* and *Groups* in Ansible. Inventory can also be provided via an *Inventory Script* (sometimes called an “External Inventory Script”).

Inventory Script A very simple program (or a complicated one) that looks up *hosts*, *group* membership for hosts, and variable information from an external resource – whether that be a SQL database, a CMDB solution, or something like LDAP. This concept was adapted from Puppet (where it is called an “External Nodes Classifier”) and works more or less exactly the same way.

Jinja2 Jinja2 is the preferred templating language of Ansible's template module. It is a very simple Python template language that is generally readable and easy to write.

JSON Ansible uses JSON for return data from remote modules. This allows modules to be written in any language, not just Python.

Lazy Evaluation In general, Ansible evaluates any variables in *playbook* content at the last possible second, which means that if you define a data structure that data structure itself can define variable values within it, and everything “just works” as you would expect. This also means variable strings can include other variables inside of those strings.

Library A collection of modules made available to `/usr/bin/ansible` or an Ansible *playbook*.

Limit Groups By passing `--limit somegroup` to `ansible` or `ansible-playbook`, the commands can be limited to a subset of *hosts*. For instance, this can be used to run a *playbook* that normally targets an entire set of servers to one particular server.

Local Action A `local_action` directive in a *playbook* targeting remote machines means that the given step will actually occur on the local machine, but that the variable `{{ ansible_hostname }}` can be passed in to reference the remote hostname being referred to in that step. This can be used to trigger, for example, an `rsync` operation.

Local Connection By using `connection: local` in a *playbook*, or passing `-c local` to `/usr/bin/ansible`, this indicates that we are managing the local host and not a remote machine.

Lookup Plugin A lookup plugin is a way to get data into Ansible from the outside world. Lookup plugins are an extension of Jinja2 and can be accessed in templates, e.g., `{{ lookup('file','/path/to/file') }}`. These are how such things as `with_items`, are implemented. There are also lookup plugins like `file` which loads data from a file and ones for querying environment variables, DNS text records, or key value stores.

Loops Generally, Ansible is not a programming language. It prefers to be more declarative, though various constructs like `loop` allow a particular task to be repeated for multiple items in a list. Certain modules, like `yum` and `apt`, actually take lists directly, and can install all packages given in those lists within a single transaction, dramatically speeding up total time to configuration, so they can be used without loops.

Modules Modules are the units of work that Ansible ships out to remote machines. Modules are kicked off by either `/usr/bin/ansible` or `/usr/bin/ansible-playbook` (where multiple tasks use lots of different modules in conjunction). Modules can be implemented in any language, including Perl, Bash, or Ruby – but can leverage some useful communal library code if written in Python. Modules just have to return *JSON*. Once modules are executed on remote machines, they are removed, so no long running daemons are used. Ansible refers to the collection of available modules as a *library*.

Multi-Tier The concept that IT systems are not managed one system at a time, but by interactions between multiple systems and groups of systems in well defined orders. For instance, a web server may need to be updated before a database server and pieces on the web server may need to be updated after *THAT* database server and various load balancers and monitoring servers may need to be contacted. Ansible models entire IT topologies and workflows rather than looking at configuration from a “one system at a time” perspective.

Notify The act of a *task* registering a change event and informing a *handler* task that another *action* needs to be run at the end of the *play*. If a handler is notified by multiple tasks, it will still be run only once. Handlers are run in the order they are listed, not in the order that they are notified.

Orchestration Many software automation systems use this word to mean different things. Ansible uses it as a conductor would conduct an orchestra. A datacenter or cloud architecture is full of many systems, playing many parts – web servers, database servers, maybe load balancers, monitoring systems, continuous integration systems, etc. In performing any process, it is necessary to touch systems in particular orders, often to simulate rolling updates or to deploy software correctly. Some system may perform some steps, then others, then previous systems already processed may need to perform more steps. Along the way, emails may need to be sent or web services contacted. Ansible orchestration is all about modeling that kind of process.

paramiko By default, Ansible manages machines over SSH. The library that Ansible uses by default to do this is a Python-powered library called paramiko. The paramiko library is generally fast and easy to manage, though users who want to use Kerberos or Jump Hosts may wish to switch to a native SSH binary such as OpenSSH by specifying the connection type in their *playbooks*, or using the `-c ssh` flag.

Playbooks Playbooks are the language by which Ansible orchestrates, configures, administers, or deploys systems. They are called playbooks partially because it's a sports analogy, and it's supposed to be fun using them. They aren't workbooks :)

Plays A *playbook* is a list of plays. A play is minimally a mapping between a set of *hosts* selected by a host specifier (usually chosen by *groups* but sometimes by hostname *globs*) and the *tasks* which run on those hosts to define the role that those systems will perform. There can be one or many plays in a playbook.

Pull Mode By default, Ansible runs in *push mode*, which allows it very fine-grained control over when it talks to each system. Pull mode is provided for when you would rather have nodes check in every N minutes on a particular schedule. It uses a program called **ansible-pull** and can also be set up (or reconfigured) using a push-mode *playbook*. Most Ansible users use push mode, but pull mode is included for variety and the sake of having choices.

ansible-pull works by checking configuration orders out of git on a crontab and then managing the machine locally, using the *local connection* plugin.

Push Mode Push mode is the default mode of Ansible. In fact, it's not really a mode at all – it's just how Ansible works when you aren't thinking about it. Push mode allows Ansible to be fine-grained and conduct nodes through complex orchestration processes without waiting for them to check in.

Register Variable The result of running any *task* in Ansible can be stored in a variable for use in a template or a conditional statement. The keyword used to define the variable is called **register**, taking its name from the idea of registers in assembly programming (though Ansible will never feel like assembly programming). There are an infinite number of variable names you can use for registration.

Resource Model Ansible modules work in terms of resources. For instance, the file module will select a particular file and ensure that the attributes of that resource match a particular model. As an example, we might wish to change the owner of `/etc/motd` to `root` if it is not already set to `root`, or set its mode to `0644` if it is not already set to `0644`. The resource models are *idempotent* meaning change commands are not run unless needed, and Ansible will bring the system back to a desired state regardless of the actual state – rather than you having to tell it how to get to the state.

Roles Roles are units of organization in Ansible. Assigning a role to a group of *hosts* (or a set of *groups*, or *host patterns*, etc.) implies that they should implement a specific behavior. A role may include applying certain variable values, certain *tasks*, and certain *handlers* – or just one or more of these things. Because of the file structure associated with a role, roles become redistributable units that allow you to share behavior among *playbooks* – or even with other users.

Rolling Update The act of addressing a number of nodes in a group N at a time to avoid updating them all at once and bringing the system offline. For instance, in a web topology of 500 nodes handling very large volume, it may be reasonable to update 10 or 20 machines at a time, moving on to the next 10 or 20 when done. The **serial:** keyword in an Ansible *playbooks* control the size of the rolling update pool. The default is to address the batch size all at once, so this is something that you must opt-in to. OS configuration (such as making sure config files are correct) does not typically have to use the

rolling update model, but can do so if desired.

Serial

参见:

Rolling Update

Sudo Ansible does not require root logins, and since it's daemonless, definitely does not require root level daemons (which can be a security concern in sensitive environments). Ansible can log in and perform many operations wrapped in a sudo command, and can work with both password-less and password-based sudo. Some operations that don't normally work with sudo (like scp file transfer) can be achieved with Ansible's copy, template, and fetch modules while running in sudo mode.

SSH (Native) Native OpenSSH as an Ansible transport is specified with `-c ssh` (or a config file, or a directive in the *playbook*) and can be useful if wanting to login via Kerberized SSH or using SSH jump hosts, etc. In 1.2.1, `ssh` will be used by default if the OpenSSH binary on the control machine is sufficiently new. Previously, Ansible selected `paramiko` as a default. Using a client that supports `ControlMaster` and `ControlPersist` is recommended for maximum performance – if you don't have that and don't need Kerberos, jump hosts, or other features, `paramiko` is a good choice. Ansible will warn you if it doesn't detect `ControlMaster/ControlPersist` capability.

Tags Ansible allows tagging resources in a *playbook* with arbitrary keywords, and then running only the parts of the playbook that correspond to those keywords. For instance, it is possible to have an entire OS configuration, and have certain steps labeled `ntp`, and then run just the `ntp` steps to reconfigure the time server information on a remote host.

Task *Playbooks* exist to run tasks. Tasks combine an *action* (a module and its arguments) with a name and optionally some other keywords (like *looping directives*). *Handlers* are also tasks, but they are a special kind of task that do not run unless they are notified by name when a task reports an underlying change on a remote system.

Tasks A list of *Task*.

Templates Ansible can easily transfer files to remote systems but often it is desirable to substitute variables in other files. Variables may come from the *inventory* file, *Host Vars*, *Group Vars*, or *Facts*. Templates use the *Jinja2* template engine and can also include logical constructs like loops and if statements.

Transport Ansible uses `:term:Connection Plugins` to define types of available transports. These are simply how Ansible will reach out to managed systems. Transports included are *paramiko*, *ssh* (using OpenSSH), and *local*.

When An optional conditional statement attached to a *task* that is used to determine if the task should run or not. If the expression following the `when:` keyword evaluates to false, the task will be ignored.

Vars (Variables) As opposed to *Facts*, variables are names of values (they can be simple scalar values – integers, booleans, strings) or complex ones (dictionaries/hashes, lists) that can be used in templates and *playbooks*. They are declared things, not things that are inferred from the remote system's current state or nature (which is what Facts are).

YAML Ansible does not want to force people to write programming language code to automate infrastructure, so Ansible uses YAML to define *playbook* configuration languages and also variable files. YAML is nice because it has a minimum of syntax and is very clean and easy for people to skim. It is a good data format for configuration files and humans, but also machine readable. Ansible's usage of YAML stemmed from Michael DeHaan's first use of it inside of Cobbler around 2006. YAML is fairly popular in the dynamic language community and the format has libraries available for serialization in many languages (Python, Perl, Ruby, etc.).

参见:

Frequently Asked Questions Frequently asked questions

Working With Playbooks An introduction to playbooks

Tips and tricks Best practices advice

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

1.22 Ansible Reference: Module Utilities

This page documents utilities intended to be helpful when writing Ansible modules in Python.

1.22.1 AnsibleModule

To use this functionality, include `from ansible.module_utils.basic import AnsibleModule` in your module.

1.22.2 Basic

To use this functionality, include `import ansible.module_utils.basic` in your module.

1.23 Special Variables

1.23.1 Magic

These variables cannot be set directly by the user; Ansible will always override them to reflect internal state.

ansible_check_mode Boolean that indicates if we are in check mode or not

ansible_config_file The full path of used Ansible configuration file

ansible_dependent_role_names The names of the roles currently imported into the current play as dependencies of other plays

ansible_diff_mode Boolean that indicates if we are in diff mode or not

ansible_forks Integer reflecting the number of maximum forks available to this run

ansible_inventory_sources List of sources used as inventory

ansible_limit Contents of the `--limit` CLI option for the current execution of Ansible

ansible_loop A dictionary/map containing extended loop information when enabled via `loop_control.extended`

ansible_loop_var The name of the value provided to `loop_control.loop_var`. Added in 2.8

ansible_index_var The name of the value provided to `loop_control.index_var`. Added in 2.9

ansible_parent_role_names When the current role is being executed by means of an `include_role` or `import_role` action, this variable contains a list of all parent roles, with the most recent role (i.e. the role that included/imported this role) being the first item in the list. When multiple inclusions occur, this list lists the *last* role (i.e. the role that included this role) as the *first* item in the list. It is also possible that a specific role exists more than once in this list.

For example: When role **A** includes role **B**, inside role **B**, **ansible_parent_role_names** will equal to `['A']`. If role **B** then includes role **C**, the list becomes `['B', 'A']`.

ansible_parent_role_paths When the current role is being executed by means of an `include_role` or `import_role` action, this variable contains a list of all parent roles, with the most recent role (i.e. the role that included/imported this role) being the first item in the list. Please refer to **ansible_parent_role_names** for the order of items in this list.

ansible_play_batch List of active hosts in the current play run limited by the serial, aka ‘batch’ . Failed/Unreachable hosts are not considered ‘active’ .

ansible_play_hosts The same as **ansible_play_batch**

ansible_play_hosts_all List of all the hosts that were targeted by the play

ansible_play_role_names The names of the roles currently imported into the current play. This list does **not** contain the role names that are implicitly included via dependencies.

ansible_playbook_python The path to the python interpreter being used by Ansible on the controller

ansible_role_names The names of the roles currently imported into the current play, or roles referenced as dependencies of the roles imported into the current play.

ansible_run_tags Contents of the `--tags` CLI option, which specifies which tags will be included for the current run.

ansible_search_path Current search path for action plugins and lookups, i.e where we search for relative paths when you do `template: src=myfile`

ansible_skip_tags Contents of the `--skip-tags` CLI option, which specifies which tags will be skipped for the current run.

ansible__verbosity Current verbosity setting for Ansible

ansible__version Dictionary/map that contains information about the current running version of ansible, it has the following keys: full, major, minor, revision and string.

group__names List of groups the current host is part of

groups A dictionary/map with all the groups in inventory and each group has the list of hosts that belong to it

hostvars A dictionary/map with all the hosts in inventory and variables assigned to them

inventory__hostname The inventory name for the ‘current’ host being iterated over in the play

inventory__hostname__short The short version of *inventory__hostname*

inventory__dir The directory of the inventory source in which the *inventory__hostname* was first defined

inventory__file The file name of the inventory source in which the *inventory__hostname* was first defined

omit Special variable that allows you to ‘omit’ an option in a task, i.e - `user: name=bob home={{ bobs_home|default(omit) }}`

play__hosts Deprecated, the same as *ansible__play__batch*

ansible__play__name The name of the currently executed play. Added in 2.8.

playbook__dir The path to the directory of the playbook that was passed to the `ansible-playbook` command line.

role__name The name of the role currently being executed.

role__names Deprecated, the same as *ansible__play__role__names*

role__path The path to the dir of the currently running role

1.23.2 Facts

These are variables that contain information pertinent to the current host (*inventory__hostname*). They are only available if gathered first.

ansible__facts Contains any facts gathered or cached for the *inventory__hostname* Facts are normally gathered by the setup module automatically in a play, but any module can return facts.

ansible__local Contains any ‘local facts’ gathered or cached for the *inventory__hostname*. The keys available depend on the custom facts created. See the setup module for more details.

1.23.3 Connection variables

Connection variables are normally used to set the specifics on how to execute actions on a target. Most of them correspond to connection plugins, but not all are specific to them; other plugins like shell, terminal and become are normally involved. Only the common ones are described as each `connection/become/shell/etc`

plugin can define its own overrides and specific variables. See [Controlling how Ansible behaves: precedence rules](#) for how connection variables interact with configuration settings, [command-line options](#), and playbook keywords.

ansible__become__user The user Ansible ‘becomes’ after using privilege escalation. This must be available to the ‘login user’ .

ansible__connection The connection plugin actually used for the task on the target host.

ansible__host The ip/name of the target host to use instead of *inventory_hostname*.

ansible__python__interpreter The path to the Python executable Ansible should use on the target host.

ansible__user The user Ansible ‘logs in’ as.

1.24 Red Hat Ansible Tower

[Red Hat Ansible Tower](#) is a web console and REST API for operationalizing Ansible across your team, organization, and enterprise. It’ s designed to be the hub for all of your automation tasks.

Ansible Tower gives you role-based access control, including control over the use of securely stored credentials for SSH and other services. You can sync your Ansible Tower inventory with a wide variety of cloud sources, and powerful multi-playbook workflows allow you to model complex processes.

It logs all of your jobs, integrates well with LDAP, SAML, and other authentication sources, and has an amazing browsable REST API. Command line tools are available for easy integration with Jenkins as well.

Ansible Tower is the downstream Red-Hat supported product version of Ansible AWX. Find out more about Ansible Tower features and how to download it on the [Ansible Tower webpage](#). Ansible Tower is part of the Red Hat Ansible Automation subscription, and comes bundled with amazing support from Red Hat, Inc.

1.25 Ansible Automation Hub

[Ansible Automation Hub](#) is the official location to discover and download supported *collections*, included as part of an Ansible Automation Platform subscription. These content collections contain modules, plugins, roles, and playbooks in a downloadable package.

Ansible Automation Hub gives you direct access to trusted content collections from Red Hat and Certified Partners. You can find content by topic or Ansible Partner organizations.

Ansible Automation Hub is the downstream Red Hat supported product version of Ansible Galaxy. Find out more about Ansible Automation Hub features and how to access it at [Ansible Automation Hub](#). Ansible Automation Hub is part of the Red Hat Ansible Automation Platform subscription, and comes bundled with support from Red Hat, Inc.

1.26 Logging Ansible output

By default Ansible sends output about plays, tasks, and module arguments to your screen (STDOUT) on the control node. If you want to capture Ansible output in a log, you have three options:

- To save Ansible output in a single log on the control node, set the `log_path` *configuration file setting*. You may also want to set `display_args_to_stdout`, which helps to differentiate similar tasks by including variable values in the Ansible output.
- To save Ansible output in separate logs, one on each managed node, set the `no_target_syslog` and `syslog_facility` *configuration file settings*.
- To save Ansible output to a secure database, use *Ansible Tower*. Tower allows you to review history based on hosts, projects, and particular inventories over time, using graphs and/or a REST API.

1.26.1 Protecting sensitive data with `no_log`

If you save Ansible output to a log, you expose any secret data in your Ansible output, such as passwords and user names. To keep sensitive values out of your logs, mark tasks that expose them with the `no_log: True` attribute. However, the `no_log` attribute does not affect debugging output, so be careful not to debug playbooks in a production environment. See *How do I keep secret data in my playbook?* for an example.

1.27 Ansible Roadmap

The Ansible team develops a roadmap for each major and minor Ansible release. The latest roadmap shows current work; older roadmaps provide a history of the project. We don't publish roadmaps for subminor versions. So 2.0 and 2.8 have roadmaps, but 2.7.1 does not.

We incorporate team and community feedback in each roadmap, and aim for further transparency and better inclusion of both community desires and submissions.

Each roadmap offers a *best guess*, based on the Ansible team's experience and on requests and feedback from the community, of what will be included in a given release. However, some items on the roadmap may be dropped due to time constraints, lack of community maintainers, etc.

Each roadmap is published both as an idea of what is upcoming in Ansible, and as a medium for seeking further feedback from the community.

You can submit feedback on the current roadmap in multiple ways:

- Edit the agenda of an IRC *Core Team Meeting* (preferred)
- Post on the `#ansible-devel` Freenode IRC channel
- Email the ansible-devel list

See *Ansible communication channels* for details on how to join and use the email lists and IRC channels.

1.27.1 Ansible 2.10

- *Release Schedule*
 - *Expected*
- *Release Manager*
 - *Planned work*

Release Schedule

Expected

PRs must be raised well in advance of the dates below to have a chance of being included in this Ansible release.

注解: There is no Alpha phase in 2.10.

- TBD 2020-??-?? Beta 1 **Feature freeze** No new functionality (including modules/plugins) to any code
- 2020-??-?? Release Candidate 1
- 2020-??-?? Release Candidate 2 if needed)
- 2020-??-?? Release

Release Manager

TBD

Temporarily, Matt Davis (@nitzmahone) or Matt Clay (@mattclay) on IRC or github.

Planned work

See the [Ansible 2.10 Project Board](#)

1.27.2 Ansible 2.9

- *Release Schedule*
 - *Expected*
- *Release Manager*
 - *Planned work*

Release Schedule

Expected

PRs must be raised well in advance of the dates below to have a chance of being included in this Ansible release.

注解: There is no Alpha phase in 2.9.

- 2019-08-29 Beta 1 **Feature freeze** No new functionality (including modules/plugins) to any code
- 2019-09-19 Release Candidate 1
- 2019-10-03 Release Candidate 2
- 2019-10-10 Release Candidate 3
- 2019-10-17 Release Candidate 4 (if needed)
- 2019-10-24 Release Candidate 5 (if needed)
- 2019-10-31 Release

Release Manager

TBD

Temporarily, Matt Davis (@nitzmahone) or Matt Clay (@mattclay) on IRC or github.

Planned work

See the [Ansible 2.9 Project Board](#)

1.27.3 Ansible 2.8

- *Release Schedule*
 - *Expected*
- *Release Manager*
 - *Planned work*

Release Schedule

Expected

PRs must be raised well in advance of the dates below to have a chance of being included in this Ansible release.

- 2019-04-04 Alpha 1 **Core freeze** No new features to `support:core` code. Includes no new options to existing Core modules
- 2019-04-11 Beta 1 **Feature freeze** No new functionality (including modules/plugins) to any code
- 2019-04-25 Release Candidate 1
- 2019-05-02 Release Candidate 2
- 2019-05-10 Release Candidate 3
- 2019-05-16 Release

Release Manager

Toshio Kuratomi (IRC: abadger1999; GitHub: @abadger)

Planned work

See the [Ansible 2.8 Project Board](#)

1.27.4 Ansible 2.7

Topics

- *Ansible 2.7*
 - *Release Schedule*
 - * *Expected*

- *Release Manager*
- *Cleaning Duty*
- *Engine Improvements*
- *Core Modules*
- *Cloud Modules*
 - * *General*
 - * *AWS*
 - * *Azure*
- *Network*
 - * *General*
 - * *Modules*
- *Windows*
 - * *General*
 - * *Modules*

Release Schedule

Expected

- 2018-08-23 Core Freeze (Engine and Core Modules/Plugins)
- 2018-08-23 Alpha Release 1
- 2018-08-30 Community Freeze (Non-Core Modules/Plugins)
- 2018-08-30 Beta Release 1
- 2018-09-06 Release Candidate 1 (If needed)
- 2018-09-13 Release Candidate 2 (If needed)
- 2018-09-20 Release Candidate 3 (If needed)
- 2018-09-27 Release Candidate 4 (If needed)
- 2018-10-04 General Availability

Release Manager

Toshio Kuratomi (IRC: abadger1999; GitHub: @abadger)

Cleaning Duty

- Drop Py2.6 for controllers [Docs PR #42971](#) and [issue #42972](#)
- Remove dependency on simplejson [issue #42761](#)

Engine Improvements

- Performance improvement invoking Python modules [pr #41749](#)
- Jinja native types will allow for users to render a Python native type. [pr #32738](#)

Core Modules

- Include feature changes and improvements
 - Create new argument `apply` that will allow for included tasks to inherit explicitly provided attributes. [pr #39236](#)
 - Create “private” functionality for allowing vars/default to be exposed outside of roles. [pr #41330](#)
- Provide a parameter for the `template` module to output to different encoding formats [pr #42171](#)
- `reboot` module for Linux hosts (@samdoran) [pr #35205](#)

Cloud Modules

General

- Cloud auth plugin [proposal #24](#)

AWS

- Inventory plugin for RDS [pr #41919](#)
- Count support for `ec2_instance`
- `aws_eks` module [pr #41183](#)
- Cloudformation stack sets support ([PR#41669](#))
- RDS instance and snapshot modules [pr #39994](#) [pr #43789](#)
- Diff mode improvements for cloud modules [pr #44533](#)

Azure

- Azure inventory plugin [issue #42769](#)

Network

General

- Refactor the APIs in cliconf (issue #39056) and netconf (issue #39160) plugins so that they have a uniform signature across supported network platforms. **done** (PR #41846) (PR #43643) (PR #43837) (PR #43203) (PR #42300) (PR #44157)

Modules

- New cli_config module issue #39228 **done** PR #42413.
- New cli_command module issue #39284
- Refactor netconf_config module to add additional functionality. **done** proposal #104 (PR #44379)

Windows

General

- Added new connection plugin that uses PSRP as the connection protocol pr #41729

Modules

- Revamp Chocolatey to fix bugs and support offline installation pr #43013.
- Add Chocolatey modules that can manage the following Chocolatey features
 - Sources pr #42790
 - Features pr #42848
 - Config pr #42915

1.27.5 Ansible 2.6

Topics

- *Ansible 2.6*
 - *Release Schedule*
 - * *Actual*
 - *Release Manager*

- *Engine improvements*
- *Core Modules*
- *Cloud Modules*
- *Network*
 - * *Connection work*
 - * *Modules*
 - * *Other Features*
- *Windows*

Release Schedule

Actual

- 2018-05-17 Core Freeze (Engine and Core Modules/Plugins)
- 2018-05-21 Alpha Release 1
- 2018-05-25 Community Freeze (Non-Core Modules/Plugins)
- 2018-05-25 Branch stable-2.6
- 2018-05-30 Alpha Release 2
- 2018-06-05 Release Candidate 1
- 2018-06-08 Release Candidate 2
- 2018-06-18 Release Candidate 3
- 2018-06-25 Release Candidate 4
- 2018-06-26 Release Candidate 5
- 2018-06-28 Final Release

Release Manager

- 2.6.0-2.6.12 Matt Clay (IRC/GitHub: @mattclay)
- 2.6.13+ Toshio Kuratomi (IRC: abadger1999; GitHub: @abadger)

Engine improvements

- Version 2.6 is largely going to be a stabilization release for Core code.

- Some of the items covered in this release, but are not limited to are the following:
 - `ansible-inventory`
 - `import_*`
 - `include_*`
 - Test coverage
 - Performance Testing

Core Modules

- Adopt-a-module Campaign
 - Review current status of all Core Modules
 - Reduce backlog of open issues against these modules

Cloud Modules

Network

Connection work

- New connection plugin: [eAPI proposal#102](#)
- New connection plugin: NX-API
- Support for configurable options for `network_cli` & `netconf`

Modules

- New `net_get` - platform agnostic module for pulling configuration via SCP/SFTP over `network_cli`
- New `net_put` - platform agnostic module for pushing configuration via SCP/SFTP over `network_cli`
- New `netconf_get` - Netconf module to fetch configuration and state data [proposal#104](#)

Other Features

- Stretch & tech preview: Configuration caching for `network_cli`. Opt-in feature to avoid `show running` performance hit

Windows

1.27.6 Ansible 2.5

Core Engine Freeze and Module Freeze: 22 January 2018

Core and Curated Module Freeze: 22 January 2018

Community Module Freeze: 7 February 2018

Release Candidate 1 will be 21 February, 2018

Target: March 2018

Service Release schedule: every 2-3 weeks

Topics

- *Ansible 2.5*
 - *Release Manager*
 - *Engine improvements*
 - *Ansible-Config*
 - *Inventory*
 - *Facts*
 - *Static Loop Keyword*
 - *Vault*
 - *Runtime Check on Modules for Blacklisting*
 - *Windows*
 - *General Cloud*
 - *AWS*
 - *Azure*
 - *Network Roadmap*
 - *Documentation*
 - *Contributor Quality of Life*

Release Manager

Matt Davis (IRC/GitHub: @nitzmahone)

Engine improvements

- Assemble module improvements - assemble just skips when in check mode, it should be able to test if there is a difference and changed=true/false. - The same with diff, it should work as template modules does
- Handle Password reset prompts cleaner
- Tasks stats for rescues and ignores
- Normalize temp dir usage across all subsystems
- Add option to set playbook dir for adhoc, inventory and console to allow for ‘relative path loading’

Ansible-Config

- Extend config to more plugin types and update plugins to support the new config

Inventory

- ansible-inventory option to output group variable assignment and data (-export)
- Create inventory plugins for: - aws

Facts

- Namespacing fact variables (via a config option) implemented in ansible/ansible PR [#18445](#). Proposal found in ansible/proposals issue [#17](#).
- Make fact collectors and gather_subset specs finer grained
- Eliminate unneeded deps between fact collectors
- Allow fact collectors to indicate if they need information from another fact collector to be gathered first.

Static Loop Keyword

- A simpler alternative to with_, loop: only takes a list
- Remove complexity from loops, lookups are still available to users
- Less confusing having a static directive vs a one that is dynamic depending on plugins loaded.

Vault

- Vault secrets client inc new ‘keyring’ client

Runtime Check on Modules for Blacklisting

- Filter on things like “supported_by” in module metadata
- Provide users with an option of “warning, error or allow/ignore”
- Configurable via ansible.cfg and environment variable

Windows

- Implement gather_subset on Windows facts
- Fix Windows async + become to allow them to work together
- Implement Windows become flags for controlling various modes (**done**) - logontype - elevation behavior
- Convert win_updates to action plugin for auto reboot and extra features (**done**)
- Spike out changing the connection over to PSRP instead of WSMV (**done- it's possible**)
- Module updates
 - win_updates (**done**)
 - * Fix win_updates to detect (or request) become
 - * Add whitelist/blacklist features to win_updates
 - win_dsc further improvements (**done**)

General Cloud

- Make multi-cloud provisioning easier
- Diff mode will output provisioning task results of ansible-playbook runs
- Terraform module

AWS

- Focus on pull requests for various modules
- Triage existing merges for modules
- Module work
 - ec2_instance
 - ec2_vpc: Allow the addition of secondary IPv4 CIDRS to existing VPCs.
 - AWS Network Load Balancer support (NLB module, ASG support, etc)
 - rds_instance

Azure

- Azure CLI auth (**done**)
- Fix Azure module results to have “high-level” output instead of raw REST API dictionary (**partial, more to come in 2.6**)
- Deprecate Azure automatic storage accounts in `azure_rm_virtualmachine` (**breaks on Azure Stack, punted until AS supports managed disks**)

Network Roadmap

- Refactor common network shared code into package (**done**)
- Convert various `nxos` modules to leverage declarative intent (**done**)
- Refactor various modules to leverage the `cliconf` plugin (**done**)
- Add various missing declarative modules for supported platforms and functions (**done**)
- Implement a feature that handles platform differences and feature unavailability (**done**)
- `netconf-config.py` should provide control for deployment strategy
- Create `netconf` connection plugin (**done**)
- Create `netconf` fact module
- Turn `network_cli` into a usable connection type (**done**)
- Implements `jsonrpc` message passing for `ansible-connection` (**done**)
- Improve logging for `ansible-connection` (**done**)
- Improve `stdout` output for failures whilst using persistent connection (**done**)
- Create `IOS-XR NetConf Plugin` and refactor `iosxr` modules to leverage `netconf` plugin (**done**)
- Refactor `junos` modules to use `netconf` plugin (**done**)
- Filters: Add a filter to convert XML response from a network device to JSON object (**done**)

Documentation

- Extend documentation to more plugins
- Document `vault-password-client` scripts.
- Network Documentation
 - New landing page (to replace `intro_networking`) (**done**)
 - Platform specific guides (**done**)

- Walk through: Getting Started (**done**)
- Networking and `become` (**done**)
- Best practice (**done**)

Contributor Quality of Life

- Finish PSScriptAnalyser integration with ansible-test (for enforcing Powershell style) (**done**)
- Resolve issues requiring skipping of some integration tests on Python 3.

A

Action, [1053](#)
Ad Hoc, [1053](#)
ANSIBLE_DEBUG, [685](#), [906](#), [1051](#)
ANSIBLE_HOST_KEY_CHECKING, [137](#)
ANSIBLE_INVENTORY, [78](#), [113](#), [338](#)
ANSIBLE_INVENTORY_IGNORE, [127](#)
ANSIBLE_KEEP_REMOTE_FILES, [574](#), [600](#)
ANSIBLE_LOG_PATH, [906](#), [907](#)
ANSIBLE_NOCOWS, [1045](#)
ANSIBLE_NULL_REPRESENTATION, [91](#)
ANSIBLE_ROLES_PATH, [1006](#)
ANSIBLE_VAULT_PASSWORD_FILE, [365](#)
Async, [1053](#)

C

Callback Plugin, [1054](#)
Check Mode, [1054](#)
Conditionals, [1054](#)
Connection Plugin, [1054](#)

D

Declarative, [1054](#)
Diff Mode, [1054](#)

E

Executor, [1054](#)

F

Facts, [1054](#)
Filter Plugin, [1055](#)

Forks, [1055](#)

G

Gather Facts (*Boolean*), [1055](#)
Globbing, [1055](#)
Group, [1055](#)
Group Vars, [1055](#)

H

Handlers, [1055](#)
Host, [1055](#)
Host Specifier, [1055](#)
Host Vars, [1056](#)

I

Idempotency, [1056](#)
Includes, [1056](#)
Inventory, [1056](#)
Inventory Script, [1056](#)

J

Jinja2, [1056](#)
JSON, [1056](#)

L

Lazy Evaluation, [1056](#)
Library, [1056](#)
Limit Groups, [1056](#)
Local Action, [1056](#)
Local Connection, [1057](#)
Lookup Plugin, [1057](#)

Loops, [1057](#)

M

Modules, [1057](#)

Multi-Tier, [1057](#)

N

Notify, [1057](#)

O

Orchestration, [1057](#)

P

paramiko, [1057](#)

Playbooks, [1058](#)

Plays, [1058](#)

Pull Mode, [1058](#)

Push Mode, [1058](#)

R

Register Variable, [1058](#)

Resource Model, [1058](#)

RFC

RFC 6241, [995](#)

Roles, [1058](#)

Rolling Update, [1058](#)

S

Serial, [1059](#)

SSH (*Native*), [1059](#)

Sudo, [1059](#)

T

Tags, [1059](#)

Task, [1059](#)

Tasks, [1059](#)

Templates, [1059](#)

Transport, [1059](#)

V

Vars (*Variables*), [1059](#)

W

When, [1059](#)

Y

YAML, [1060](#)

环境变量

ANSIBLE_DEBUG, [685](#), [906](#), [1051](#)

ANSIBLE_HOST_KEY_CHECKING, [137](#)

ANSIBLE_INVENTORY, [78](#), [113](#), [338](#)

ANSIBLE_INVENTORY_IGNORE, [127](#)

ANSIBLE_KEEP_REMOTE_FILES, [574](#), [600](#)

ANSIBLE_LOG_PATH, [906](#), [907](#)

ANSIBLE_NOCOWS, [1045](#)

ANSIBLE_NULL_REPRESENTATION, [91](#)

ANSIBLE_ROLES_PATH, [1006](#)

ANSIBLE_VAULT_PASSWORD_FILE, [365](#)