**Getting Started**

**Topics**

* Getting Started
  + Foreword
  + Remote Connection Information
  + Your first commands
  + Host Key Checking

**Foreword**

Now that you’ve read Installation and installed Ansible, it’s time to get started with some ad-hoc commands.

What we are showing first are not the powerful configuration/deployment/orchestration features of Ansible. These features are handled by playbooks which are covered in a separate section.

This section is about how to initially get Ansible running. Once you understand these concepts, read Introduction To Ad-Hoc Commands for some more detail, and then you’ll be ready to begin learning about playbooks and explore the most interesting parts!

**Remote Connection Information**

Before we get started, it’s important to understand how Ansible communicates with remote machines over SSH.

By default, Ansible 1.3 and later will try to use native OpenSSH for remote communication when possible. This enables ControlPersist (a performance feature), Kerberos, and options in ~/.ssh/config such as Jump Host setup. However, when using Enterprise Linux 6 operating systems as the control machine (Red Hat Enterprise Linux and derivatives such as CentOS), the version of OpenSSH may be too old to support ControlPersist. On these operating systems, Ansible will fallback into using a high-quality Python implementation of OpenSSH called ‘paramiko’. If you wish to use features like Kerberized SSH and more, consider using Fedora, OS X, or Ubuntu as your control machine until a newer version of OpenSSH is available for your platform – or engage ‘accelerated mode’ in Ansible. See Accelerated Mode.

In releases up to and including Ansible 1.2, the default was strictly paramiko. Native SSH had to be explicitly selected with the -c ssh option or set in the configuration file.

Occasionally you’ll encounter a device that doesn’t support SFTP. This is rare, but should it occur, you can switch to SCP mode in Configuration file.

When speaking with remote machines, Ansible by default assumes you are using SSH keys. SSH keys are encouraged but password authentication can also be used where needed by supplying the option --ask-pass. If using sudo features and when sudo requires a password, also supply --ask-become-pass(previously --ask-sudo-pass which has been deprecated).

While it may be common sense, it is worth sharing: Any management system benefits from being run near the machines being managed. If you are running Ansible in a cloud, consider running it from a machine inside that cloud. In most cases this will work better than on the open Internet.

As an advanced topic, Ansible doesn’t just have to connect remotely over SSH. The transports are pluggable, and there are options for managing things locally, as well as 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.

**Your first commands**

Now that you’ve installed Ansible, it’s time to get started with some basics.

Edit (or create) /etc/ansible/hosts and put one or more remote systems in it. Your public SSH key should be located in authorized\_keys on those systems:

192.0.2.50

aserver.example.org

bserver.example.org

This is an inventory file, which is also explained in greater depth here: Inventory.

We’ll assume you are using SSH keys for authentication. 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 option to specify a pem file instead)

Now ping all your nodes:

$ ansible all -m ping

Ansible will attempt to remote connect to the machines using your current user name, just like SSH would. To override the remote user name, just use the ‘-u’ parameter.

If you would like to access sudo mode, there are also flags to do that:

*# as bruce*

$ ansible all -m ping -u bruce

*# as bruce, sudoing to root*

$ ansible all -m ping -u bruce --sudo

*# as bruce, sudoing to batman*

$ ansible all -m ping -u bruce --sudo --sudo-user batman

*# With latest version of ansible `sudo` is deprecated so use become*

*# as bruce, sudoing to root*

$ ansible all -m ping -u bruce -b

*# as bruce, sudoing to batman*

$ ansible all -m ping -u bruce -b --become-user batman

(The sudo implementation is changeable in Ansible’s configuration file if you happen to want to use a sudo replacement. Flags passed to sudo (like -H) can also be set there.)

Now run a live command on all of your nodes:

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

Congratulations! You’ve just contacted your nodes with Ansible. It’s soon going to be time to: read about some more real-world cases in Introduction To Ad-Hoc Commands, explore what you can do with different modules, and to learn about the Ansible Playbooks language. Ansible is not just about running commands, it also has powerful configuration management and deployment features. There’s more to explore, but you already have a fully working infrastructure!

Tips

When running commands, you can specify the local server by using “localhost” or “127.0.0.1” for the server name.

Example:

$ 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"

**Host Key Checking**

Ansible 1.2.1 and later have host key checking enabled by default.

If a host is reinstalled and has a different key in ‘known\_hosts’, this will result in an error message until corrected. If a host is not initially in ‘known\_hosts’ this will result in prompting 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

**Inventory**

**Topics**

* Inventory
  + Hosts and Groups
  + Host Variables
  + Group Variables
  + Groups of Groups, and Group Variables
  + Default groups
  + Splitting Out Host and Group Specific Data
  + List of Behavioral Inventory Parameters
  + Non-SSH connection types

Ansible works against multiple systems in your infrastructure at the same time. It does this by selecting portions of systems listed in Ansible’s inventory, which defaults to being saved in the location /etc/ansible/hosts. You can specify a different inventory file using the -i <path> option on the command line.

Not only is this inventory configurable, but you can also use multiple inventory files at the same time and pull inventory from dynamic or cloud sources or different formats (YAML, ini, etc), as described in Dynamic Inventory. Introduced in version 2.4, Ansible has inventory plugins to make this flexible and customizable.

**Hosts and Groups**

The inventory file can be in one of many formats, depending on the inventory plugins you have. For this example, the format for /etc/ansible/hosts is an INI-like (one of Ansible’s defaults) and looks like this:

mail.example.com

**[webservers]**

foo.example.com

bar.example.com

**[dbservers]**

one.example.com

two.example.com

three.example.com

The headings in brackets are group names, which are used in classifying systems and deciding what systems you are controlling at what times and for what purpose.

A YAML version would look like:

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:

It is ok to put systems in more than one group, for instance a server could be both a webserver and a dbserver. If you do, note that variables will come from all of the groups they are a member of. Variable precedence is detailed in a later chapter.

If you have hosts that run on non-standard SSH ports you can put the port number after the hostname with a colon. Ports listed in your SSH config file won’t be used with the *paramiko* connection but will be used with the *openssh* connection.

To make things explicit, it is suggested that you set them if things are not running on the default port:

badwolf.example.com:5309

Suppose you have just static IPs and want to set up some aliases that live in your host file, or you are connecting through tunnels. You can also describe hosts via variables:

In INI:

jumper ansible\_port**=**5555 ansible\_host=192.0.2.50

In YAML:

hosts:

jumper:

ansible\_port: 5555

ansible\_host: 192.0.2.50

In the above example, trying to ansible against the host alias “jumper” (which may not even be a real hostname) will contact 192.0.2.50 on port 5555. Note that this is using a feature of the inventory file to define some special variables. Generally speaking, this is not the best way to define variables that describe your system policy, but we’ll share suggestions on doing this later.

**Note**

Values passed in the INI format using the key=value syntax are not interpreted as Python literal structure (strings, numbers, tuples, lists, dicts, booleans, None), but as a string. For example var=FALSE would create a string equal to ‘FALSE’. Do not rely on types set during definition, always make sure you specify type with a filter when needed when consuming the variable.

If you are adding a lot of hosts following similar patterns, you can do this rather than listing each hostname:

**[webservers]**

www[01:50].example.com

For numeric patterns, leading zeros can be included or removed, as desired. Ranges are inclusive. You can also define alphabetic ranges:

**[databases]**

db-[a:f].example.com

**Note**

Ansible 2.0 has deprecated the “ssh” from ansible\_ssh\_user, ansible\_ssh\_host, and ansible\_ssh\_port to become ansible\_user, ansible\_host, and ansible\_port. If you are using a version of Ansible prior to 2.0, you should continue using the older style variables (ansible\_ssh\_\*). These shorter variables are ignored, without warning, in older versions of Ansible.

You can also select the connection type and user on a per host basis:

**[targets]**

localhost ansible\_connection**=**local

other1.example.com ansible\_connection**=**ssh ansible\_user=mpdehaan

other2.example.com ansible\_connection**=**ssh ansible\_user=mdehaan

As mentioned above, setting these in the inventory file is only a shorthand, and we’ll discuss how to store them in individual files in the ‘host\_vars’ directory a bit later on.

**Host Variables**

As described above, it is easy to assign variables to hosts that will be used later in playbooks:

**[atlanta]**

host1 http\_port**=**80 maxRequestsPerChild=808

host2 http\_port**=**303 maxRequestsPerChild=909

**Group Variables**

Variables can also be applied to an entire group at once:

The INI way:

**[atlanta]**

host1

host2

**[atlanta:vars]**

ntp\_server**=**ntp.atlanta.example.com

proxy**=**proxy.atlanta.example.com

The YAML version:

atlanta:

hosts:

host1:

host2:

vars:

ntp\_server: ntp.atlanta.example.com

proxy: proxy.atlanta.example.com

Be aware that this is only a convenient way to apply variables to multiple hosts at once; even though you can target hosts by group, **variables are always flattened to the host level** before a play is executed.

**Groups of Groups, and Group Variables**

It is also possible to make groups of groups using the :children suffix in INI or the children: entry in YAML. You can apply variables using :vars or vars::

**[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

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:

If you need to store lists or hash data, or prefer to keep host and group specific variables separate from the inventory file, see the next section. Child groups have a couple of properties to note:

* Any host that is member of a child group is automatically a member of the parent group.
* A child group’s variables will have higher precedence (override) a parent group’s variables.
* Groups can have multiple parents and children, but not circular relationships.
* Hosts can also be in multiple groups, but there will only be **one** instance of a host, merging the data from the multiple groups.

**Default groups**

There are two default groups: all and ungrouped. all contains every host. ungrouped contains all hosts that don’t have another group aside from all. Every host will always belong to at least 2 groups. Though all and ungrouped are always present, they can be implicit and not appear in group listings like group\_names.

**Splitting Out Host and Group Specific Data**

The preferred practice in Ansible is to not store variables in the main inventory file.

In addition to storing variables directly in the inventory file, host and group variables can be stored in individual files relative to the inventory file (not directory, it is always the file).

These variable files are in YAML format. Valid file extensions include ‘.yml’, ‘.yaml’, ‘.json’, or no file extension. See YAML Syntax if you are new to YAML.

Assuming the inventory file path is:

/etc/ansible/hosts

If the host is named ‘foosball’, and in groups ‘raleigh’ and ‘webservers’, variables in YAML files at the following locations will be made available to the host:

/etc/ansible/group\_vars/raleigh *# can optionally end in '.yml', '.yaml', or '.json'*

/etc/ansible/group\_vars/webservers

/etc/ansible/host\_vars/foosball

For instance, suppose you have hosts grouped by datacenter, and each datacenter uses some different servers. The data in the groupfile ‘/etc/ansible/group\_vars/raleigh’ for the ‘raleigh’ group might look like:

---

ntp\_server: acme.example.org

database\_server: storage.example.org

It is okay if these files do not exist, as this is an optional feature.

As an advanced use case, you can create *directories* named after your groups or hosts, and Ansible will read all the files in these directories. An example with the ‘raleigh’ group:

/etc/ansible/group\_vars/raleigh/db\_settings

/etc/ansible/group\_vars/raleigh/cluster\_settings

All hosts that are in the ‘raleigh’ group will have the variables defined in these files available to them. This can be very useful to keep your variables organized when a single file starts to be too big, or when you want to use Ansible Vault on a part of a group’s variables. Note that this only works on Ansible 1.4 or later.

Tip: In Ansible 1.2 or later the group\_vars/ and host\_vars/ directories can exist in the playbook directory OR the inventory directory. If both paths exist, variables in the playbook directory will override variables set in the inventory directory.

Tip: Keeping your inventory file and variables in a git repo (or other version control) is an excellent way to track changes to your inventory and host variables.

**List of Behavioral Inventory Parameters**

As alluded to above, setting the following variables controls how ansible interacts with remote hosts.

Host connection:

**ansible\_connection**

Connection type to the host. This can be the name of any of ansible’s connection plugins. SSH protocol types are smart, ssh or paramiko. The default is smart. Non-SSH based types are described in the next section.

**Note**

Ansible 2.0 has deprecated the “ssh” from ansible\_ssh\_user, ansible\_ssh\_host, and ansible\_ssh\_port to become ansible\_user, ansible\_host, and ansible\_port. If you are using a version of Ansible prior to 2.0, you should continue using the older style variables (ansible\_ssh\_\*). These shorter variables are ignored, without warning, in older versions of Ansible.

General for all connections:

**ansible\_host**

The name of the host to connect to, if different from the alias you wish to give to it.

**ansible\_port**

The ssh port number, if not 22

**ansible\_user**

The default ssh user name to use.

Specific to the SSH connection:

**ansible\_ssh\_pass**

The ssh password to use (never store this variable in plain text; always use a vault. See Variables and Vaults)

**ansible\_ssh\_private\_key\_file**

Private key file used by ssh. Useful if using multiple keys and you don’t want to use SSH agent.

**ansible\_ssh\_common\_args**

This setting is always appended to the default command line for **sftp**, **scp**, and **ssh**. Useful to configure a ProxyCommand for a certain host (or group).

**ansible\_sftp\_extra\_args**

This setting is always appended to the default **sftp** command line.

**ansible\_scp\_extra\_args**

This setting is always appended to the default **scp** command line.

**ansible\_ssh\_extra\_args**

This setting is always appended to the default **ssh** command line.

**ansible\_ssh\_pipelining**

Determines whether or not to use SSH pipelining. This can override the pipelining setting in ansible.cfg.

**ansible\_ssh\_executable (added in version 2.2)**

This setting overrides the default behavior to use the system **ssh**. This can override the ssh\_executable setting in ansible.cfg.

Privilege escalation (see Ansible Privilege Escalation for further details):

**ansible\_become**

Equivalent to ansible\_sudo or ansible\_su, allows to force privilege escalation

**ansible\_become\_method**

Allows to set privilege escalation method

**ansible\_become\_user**

Equivalent to ansible\_sudo\_user or ansible\_su\_user, allows to set the user you become through privilege escalation

**ansible\_become\_pass**

Equivalent to ansible\_sudo\_pass or ansible\_su\_pass, allows you to set the privilege escalation password (never store this variable in plain text; always use a vault. See Variables and Vaults)

**ansible\_become\_exe**

Equivalent to ansible\_sudo\_exe or ansible\_su\_exe, allows you to set the executable for the escalation method selected

**ansible\_become\_flags**

Equivalent to ansible\_sudo\_flags or ansible\_su\_flags, allows you to set the flags passed to the selected escalation method. This can be also set globally in ansible.cfg in the sudo\_flags option

Remote host environment parameters:

**ansible\_shell\_type**

The shell type of the target system. You should not use this setting unless you have set the ansible\_shell\_executable to a non-Bourne (sh) compatible shell. By default commands are formatted using sh-style syntax. Setting this to csh or fish will cause commands executed on target systems to follow those shell’s syntax instead.

**ansible\_python\_interpreter**

The target host python path. This is useful for systems with more than one Python or not located at **/usr/bin/python** such as \*BSD, or where **/usr/bin/python** is not a 2.X series Python. We do not use the **/usr/bin/env** mechanism as that requires the remote user’s path to be set right and also assumes the **python** executable is named python, where the executable might be named something like **python2.6**.

**ansible\_\*\_interpreter**

Works for anything such as ruby or perl and works just like ansible\_python\_interpreter. This replaces shebang of modules which will run on that host.

New in version 2.1.

**ansible\_shell\_executable**

This sets the shell the ansible controller will use on the target machine, overrides executable in ansible.cfg which defaults to **/bin/sh**. You should really only change it if is not possible to use **/bin/sh** (i.e. **/bin/sh** is not installed on the target machine or cannot be run from sudo.).

Examples from an Ansible-INI host file:

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 connection types**

As stated in the previous section, Ansible executes playbooks over SSH but it is not limited to this connection type. With the host specific parameter ansible\_connection=<connector>, the connection type can be changed. The following non-SSH based connectors are available:

**local**

This connector can be used to deploy the playbook to the control machine itself.

**docker**

This connector deploys the playbook directly into Docker containers using the local Docker client. The following parameters are processed by this connector:

**ansible\_host**

The name of the Docker container to connect to.

**ansible\_user**

The user name to operate within the container. The user must exist inside the container.

**ansible\_become**

If set to true the become\_user will be used to operate within the container.

**ansible\_docker\_extra\_args**

Could be a string with any additional arguments understood by Docker, which are not command specific. This parameter is mainly used to configure a remote Docker daemon to use.

Here is an example of how to instantly deploy to created containers:

- 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

- name: create directory for ssh keys

delegate\_to: my\_jenkins

file:

path: "/var/jenkins\_home/.ssh/jupiter"

state: directory