**Ansible Documentation**

Reference: http://docs.ansible.com/

## About Ansible

Ansible is an IT automation tool. It can configure systems, deploy software, and orchestrate more advanced IT tasks such as continuous deployments or zero downtime rolling updates.

Ansible’s main goals are simplicity and ease-of-use. It also has a strong focus on security and reliability, featuring a minimum of moving parts, usage of OpenSSH for transport (with an accelerated socket mode and pull modes as alternatives), and a language that is designed around auditability by humans–even those not familiar with the program.

We believe simplicity is relevant to all sizes of environments, so we design for busy users of all types: developers, sysadmins, release engineers, IT managers, and everyone in between. Ansible is appropriate for managing all environments, from small setups with a handful of instances to enterprise environments with many thousands of instances.

Ansible manages machines in an agent-less manner. There is never a question of how to upgrade remote daemons or the problem of not being able to manage systems because daemons are uninstalled. Because OpenSSH is one of the most peer-reviewed open source components, security exposure is greatly reduced. Ansible is decentralized–it relies on your existing OS credentials to control access to remote machines. If needed, Ansible can easily connect with Kerberos, LDAP, and other centralized authentication management systems.

This documentation covers the current released version of Ansible (2.4) and also some development version features (‘devel’). For recent features, we note in each section the version of Ansible where the feature was added.

Ansible, Inc. releases a new major release of Ansible approximately every two months. The core application evolves somewhat conservatively, valuing simplicity in language design and setup. However, the community around new modules and plugins being developed and contributed moves very quickly, typically adding 20 or so new modules in each release.

[Basic Concepts](https://docs.ansible.com/ansible/2.5/network/getting_started/basic_concepts.html#basic-concepts)

* [Control Node](https://docs.ansible.com/ansible/2.5/network/getting_started/basic_concepts.html#control-node)
* [Managed Nodes](https://docs.ansible.com/ansible/2.5/network/getting_started/basic_concepts.html#managed-nodes)
* [Inventory](https://docs.ansible.com/ansible/2.5/network/getting_started/basic_concepts.html#inventory)
* [Modules](https://docs.ansible.com/ansible/2.5/network/getting_started/basic_concepts.html#modules)
* [Tasks](https://docs.ansible.com/ansible/2.5/network/getting_started/basic_concepts.html#tasks)
* [Playbooks](https://docs.ansible.com/ansible/2.5/network/getting_started/basic_concepts.html" \l "playbooks)

## [Installing the Control Machine](http://docs.ansible.com/ansible/latest/intro_installation.html#id13)

### [Latest Release Via Yum](http://docs.ansible.com/ansible/latest/intro_installation.html#id14)

RPMs for RHEL7 are available from [the Extras channel](https://access.redhat.com/solutions/912213)

**Or use below to install on Centos**

https://centos.pkgs.org/7/centos-extras-x86\_64/ansible-2.4.0.0-5.el7.noarch.rpm.html

### Control Nodes

Ansible is simple to install. The Ansible software only needs to be installed on the control node (or nodes) from which Ansible will be run. Hosts that are managed by Ansible do not need to have Ansible installed. This installation involves relatively few steps and has minimal requirements.

The control node should be a Linux or UNIX system. Microsoft Windows is not supported as a control node, although Windows systems can be managed hosts.

Python 2 (version 2.6 or later) needs to be installed on the control node. (Use of Python 3 with Ansible is still in technology preview and should not be used in production.) To see whether the appropriate version of Python is installed on a Red Hat Enterprise Linux system, use the **yum** command.

[root@controlnode ~]# **yum list installed python**

Loaded plugins: langpacks, search-disabled-repos

Installed Packages

python.x86\_64 2.7.5-48.el7 installed

Official instructions on how to obtain, install, and get updates for Ansible for Red Hat Enterprise Linux, as well as for other operating systems and Linux distributions, are available on the Ansible website at <https://www.ansible.com/get-started>. This course is based on Ansible 2.3.

Ansible control nodes need to communicate with managed hosts over the network. By default, SSH is used, but other protocols might be needed if network devices or Microsoft Windows systems are being managed. On Red Hat Enterprise Linux control nodes, if you are managing Microsoft Windows systems, you also need to have version 0.2.2 or later of the *python2-winrm* RPM package installed (which provides the *pywinrm* Python package).

### Managed Hosts

One of the benefits of Ansible is that managed hosts do not need to have a special agent installed. The Ansible control node connects to managed hosts using a standard network protocol to ensure that the systems are in the specified state.

Managed hosts might have some requirements depending on how the control node connects to them and what modules it will run on them.

Linux and UNIX managed hosts need to have Python 2 (version 2.4 or later) installed for most modules to work. For Red Hat Enterprise Linux, install the *python* package. If the version of Python installed on the managed host is earlier than Python 2.5, then the *python-simplejson* package must also be installed.

If SELinux is enabled on the managed hosts, you also need to install the *libselinux-python* package before using modules that are related to any copy, file, or template functions. (Note that if the other Python components are installed, you can use Ansible modules such as **yum** or **package** to ensure that this package is also installed.)

Some modules might have their own additional requirements. For example, the **dnf** module, which can be used to install packages on current Fedora systems, requires the *python-dnf* package.

**Microsoft Windows-based Managed Hosts**

Ansible includes a number of modules that are specifically designed for Microsoft Windows systems. These are listed in the [Windows Modules](https://docs.ansible.com/ansible/list_of_windows_modules.html) section of the Ansible module index.

Most of the modules specifically designed for Microsoft Windows managed hosts require PowerShell 3.0 or higher on the managed host rather than Python. In addition, the managed hosts need to have PowerShell remoting configured.

This course uses Linux-based managed hosts in its examples, and does not go into great depth on the specific differences and adjustments needed when managing Microsoft Windows-based managed hosts. If you're interested, more information is available on the Ansible web site at <https://docs.ansible.com/ansible/intro_windows.html>.

[student@workstation ~]$ **yum list installed python**

[student@workstation ~]$ **sudo yum install -y ansible**

**Ansible** [**Inventory**](http://docs.ansible.com/ansible/latest/intro_inventory.html#id5)

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](http://docs.ansible.com/ansible/latest/intro_dynamic_inventory.html). Introduced in version 2.4, Ansible has inventory plugins to make this flexible and customizable.

### 1.Static Inventory

## [Hosts and Groups](http://docs.ansible.com/ansible/latest/intro_inventory.html#id6)

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

Hosts can be in multiple groups. In fact, recommended practice is to organize your hosts into multiple groups, possibly organized in different ways depending on the role of the host, its physical location, whether it's in production or not, and so on. This allows you to more easily apply Ansible plays to specific hosts.

[webservers]

web1.example.com

web2.example.com

192.168.3.7

[db-servers]

db1.example.com

db2.example.com

[east-datacenter]

web1.example.com

db1.example.com

[west-datacenter]

web2.example.com

db2.example.com

[production]

web1.example.com

web2.example.com

db1.example.com

db2.example.com

[development]

192.0.2.42

Note:

Two host groups always exist:

* The **all** host group contains every host explicitly listed in the inventory.
* The **ungrouped** host group contains every host explicitly listed in the inventory that isn't a member of any other group.

**Defining Nested Groups**

Ansible host inventories can include groups of host groups. This is accomplished with the **:children** suffix. The following example creates a new group called **north-america**, which includes all of the hosts from the **usa** and **canada** groups.

[usa]

washington1.example.com

washington2.example.com

[canada]

ontario01.example.com

ontario02.example.com

[north-america:children]

canada

usa

A group can have both managed hosts and child groups as members. For example, in the previous inventory we could add a **[north-america]** section that has its own list of managed hosts. That list of hosts would be merged with the additional hosts the **north-america** group inherits from its child groups.

**Simplifying Host Specifications with Ranges**

Ansible host inventories can be simplified by specifying ranges in the host names or IP addresses. Numeric ranges can be specified, but alphabetic ranges are also supported. Ranges have the following syntax:

[***START***:***END***]

Ranges match all the values from ***START*** to ***END***, inclusive. Consider the following examples:

* **192.168.[4:7].[0:255]** will match all IPv4 addresses in the 192.168.4.0/22 network (192.168.4.0 through 192.168.7.255).
* **server[01:20].example.com** will match all hosts named **server01.example.com** through **server20.example.com**.
* **[a:c].dns.example.com** will match hosts named **a.dns.example.com**, **b.dns.example.com**, and **c.dns.example.com**.
* **2001:db8::[a:f]** will match all IPv6 addresses from **2001:db8::a** through **2001:db8::f**.

If leading zeros are included in numeric ranges, they are used in the pattern. The second example above does not match **server1.example.com** but does match**server07.example.com**. To illustrate this, the following example uses ranges to simplify the **usa** and **canada** group definitions from the earlier example:

[usa]

washington[1:2].example.com

[canada]

ontario[01:02].example.com

**Testing the Inventory**

When in doubt, test the machine's presence in the inventory with the **ansible** command:

[user@demo ~]$ **ansible washington1.example.com --list-hosts**

hosts (1):

washington1.example.com

[user@demo ~]$ **ansible washington01.example.com --list-hosts**

[WARNING]: provided hosts list is empty, only localhost is available

hosts (0):

You can run the following command to list all hosts in a group:

[user@demo ~]$ **ansible canada --list-hosts**

hosts (2):

ontario01.example.com

ontario02.example.com

**Overriding the Location of the Inventory**

The **/etc/ansible/hosts** file is considered the system's default static inventory file. However, normal practice is not to use that file but to define a different location for inventory files in your Ansible configuration file. This is covered in the next section.

The **ansible** and **ansible-playbook** commands you'll be using to run Ansible ad hoc commands and playbooks later in the course can also specify the location of an inventory file on the command line with the **--inventory *PATHNAME*** or **-i *PATHNAME*** option, where **PATHNAME** is the path to the desired inventory file.

**Defining Variables in the Inventory**

Values for variables used by playbooks can be specified in host inventory files. These variables only apply to specific hosts or host groups. Normally it is better to define these inventory variables in special directories and not directly in the inventory file. This topic is discussed in more depth elsewhere in the course.

### 2. Dynamic Inventory

Ansible inventory information can also be dynamically generated, using information provided by external databases. The open source community has written a number of dynamic inventory scripts that are available from the upstream Ansible project. If those scripts don't meet your needs, you can also write your own.

For example, a dynamic inventory program could contact your Red Hat Satellite server or Amazon EC2 account, and use information stored there to construct an Ansible inventory. Since the program does this when Ansible is run, it can populate the inventory with up-to-date information provided by the service as new hosts are added and old hosts are removed.

This topic is discussed in more detail later in this chapter.

### Generating Inventories Dynamically

The static inventory files you've worked with so far are easy to write, and are convenient for managing small infrastructures. When working with a large number of machines, however, or in an environment where machines come and go very quickly, it can be hard to keep the static inventory files up-to-date.

Most large IT environments have systems that keep track of which hosts are available and how they are organized. For example, there might be an external directory service maintained by a monitoring system such as Zabbix, or on FreeIPA or Active Directory servers. Installation servers such as Cobbler or management services such as Red Hat Satellite might track deployed bare-metal systems. In a similar way, cloud services such as Amazon Web Services EC2 or an OpenStack deployment, or virtual machine infrastructures based on VMware or Red Hat Virtualization might be sources of information about the instances and virtual machines that come and go.

Ansible supports dynamic inventory scripts that retrieve current information from these types of sources whenever Ansible executes, allowing the inventory to be updated in real time. These scripts are executable programs that collect information from some external source and output the inventory in JSON format.

Dynamic inventory scripts are used just like static inventory text files. The location of the inventory is specified either directly in the current **ansible.cfg** file, or using the **-i** option. If the inventory file is executable, it is treated as a dynamic inventory program and Ansible attempts to run it to generate the inventory. If the file is not executable, it is treated as a static inventory.

[student@workstation ~]$ **./inventoryscript --list**

{

"webservers" : [ "web1.lab.example.com", "web2.lab.example.com" ],

"databases" : [ "db1.lab.example.com", "db2.lab.example.com" ]

}

Create an **ansible.cfg** Ansible configuration file in the working directory and populate it with the following entries so that the **inventory** directory is configured as the default inventory.

[defaults]

inventory = inventory

$ **mkdir inventory**

http://docs.ansible.com/ansible/latest/intro\_dynamic\_inventory.html

Managing Ansible Configuration Files

### Configuring Ansible

The behavior of an Ansible installation can be customized by modifying settings in the Ansible configuration file. Ansible chooses its configuration file from one of several possible locations on the control node.

**Using /etc/ansible/ansible.cfg**

The *ansible* package provides a base configuration file located at **/etc/ansible/ansible.cfg**. This file is used if no other configuration file is found.

**Using ~/.ansible.cfg**

Ansible looks for a **~/.ansible.cfg** in the user's home directory. This configuration is used instead of the **/etc/ansible/ansible.cfg** if it exists and if there is no **ansible.cfg** file in the current working directory.

**Using ./ansible.cfg**

If an **ansible.cfg** file exists in the directory in which the **ansible** command is executed, it is used instead of the global file or the user's personal file. This allows administrators to create a directory structure where different environments or projects are stored in separate directories, with each directory containing a configuration file tailored with a unique set of settings.

Recommended Practice:

The recommended practice is to create an **ansible.cfg** file in a directory from which you run Ansible commands. This directory would also contain any files used by your Ansible project, such as an inventory and a playbook. This is the most common location used for the Ansible configuration file. It is unusual to use a**~/.ansible.cfg** or **/etc/ansible/ansible.cfg** file in practice.

**Using $ANSIBLE\_CONFIG**

You can use different configuration files by placing them in different directories and then executing Ansible commands from the appropriate directory, but this method can be restrictive and hard to manage as the number of configuration files grows. A more flexible option is to define the location of the configuration file with the **$ANSIBLE\_CONFIG** environment variable. When this variable is defined, Ansible uses the configuration file that the variable specifies instead of any of the previously mentioned configuration files.

### Configuration File Precedence

[student@controlnode ~]$ **ansible --version**

ansible 2.3.1.0

config file = /etc/ansible/ansible.cfg

...output omitted...

[student@controlnode ~]$ **ansible servers --list-hosts -v**

Using /etc/ansible/ansible.cfg as config file

...output omitted...

### Managing Settings in the Configuration File

The Ansible configuration file consists of several sections, with each section containing settings defined as key-value pairs. Section titles are enclosed in square brackets. Settings are grouped under the following six sections in the default Ansible configuration file:

[student@controlnode ~]$ **grep "^\[" /etc/ansible/ansible.cfg**

[defaults]

[privilege\_escalation]

[paramiko\_connection]

[ssh\_connection]

[accelerate]

[selinux]

Most of the settings in the configuration file are grouped under the **[defaults]** section. The **[privilege\_escalation]** section contains settings for defining how operations that require escalated privileges are executed on managed hosts. The **[paramiko\_connection]**, **[ssh\_connection]**, and **[accelerate]** sections contain settings for optimizing connections to managed hosts. The **[selinux]** section contains settings for defining how SELinux interactions are configured. Although not included in the default global Ansible configuration file provided by the *ansible* package, a **[galaxy]** section is also available for defining parameters related to Ansible Galaxy, which is discussed in a later chapter.

**Inventory Location**

In the **[defaults]** section, the **inventory** directive can point directly to a static inventory file, or to a directory that contains multiple static inventory files and/or dynamic inventory scripts:

[defaults]

inventory = ./inventory

**Connection Settings**

By default, Ansible connects to managed hosts using the SSH protocol. The most important parameters that control how Ansible connects to the managed hosts are set in the **[defaults]** section.

By default, Ansible attempts to connect to the managed host using the same user name as the local user running the Ansible commands. To specify a different remote user, set the **remote\_user** parameter to that user name.

If the local user running Ansible has a private SSH key or keys configured that allow them to authenticate as the remote user on the managed hosts, Ansible automatically logs in. If that's not the case, you can configure Ansible to prompt the local user for the password used by the remote user by setting the directive **ask\_pass = true**.

[defaults]

inventory = ./inventory

remote\_user = root

ask\_pass = true

Assuming that you're using a Linux control node and OpenSSH on your managed hosts, if you can log in to the remote user with a password then you can probably set up SSH key-based authentication, which would allow you to set **ask\_pass = false**.

The first step is to make sure that the user on the control node has an SSH key pair configured in **~/.ssh**. You can run the **ssh-keygen** command to accomplish this.

**Privilege Escalation**

For security and auditing reasons, Ansible might need to connect to remote hosts as a non-privileged user before escalating privileges to get administrative access as **root**. This can be set up in the **[privilege\_escalation]** section of the Ansible configuration file.

To enable privilege escalation by default, set the directive **become = true** in the configuration file. Even if this is set by default, there are various ways to override it when running ad hoc commands or Ansible Playbooks. (For example, there might be times when you want to run a task or play that does not escalate privileges.)

The **become\_method** directive specifies how to escalate privileges. Several options are available, but the default is to use **sudo**. Likewise, the **become\_user** directive specifies which user to escalate to, but the default is **root**.

If the **become\_method** mechanism chosen requires the user to enter a password to escalate privileges, you can set the **become\_ask\_pass = true** directive in the configuration file.

The following example **ansible.cfg** file assumes that you can connect to the managed hosts as **someuser** using SSH key-based authentication, and that **someuser** can use **sudo** to run commands as **root** without entering a password:

[defaults]

inventory = ./inventory

remote\_user = someuser

ask\_pass = false

[privilege\_escalation]

become = true

become\_method = sudo

become\_user = root

become\_ask\_pass = false

The following table summarizes some of the most commonly modified directives in the Ansible configuration file.

| **Setting** | **Description** |
| --- | --- |
| **inventory** | The location of the Ansible inventory. |
| **remote\_user** | The remote user account used to establish connections to managed hosts. |
| **ask\_pass** | Prompt for a password to use when connecting as the remote user. |
| **become** | Enable or disable privilege escalation for operations on managed hosts. |
| **become\_method** | The privilege escalation method to use on managed hosts. |
| **become\_user** | The user account to escalate privileges to on managed hosts. |
| **become\_ask\_pass** | Defines whether privilege escalation on managed hosts should prompt for a password. |

**Non-SSH Connections**

[student@controlnode ~]$ **ansible localhost --list-hosts**

[WARNING]: provided hosts list is empty, only localhost is available

hosts (1):

localhost

If you want to make sure that you connect to **localhost** using SSH like other managed hosts, one approach is to list it in your inventory. But this will include it in groups **all** and **ungrouped**, which you may not want to do.

Note:

You can also use group variables to change the connection type for an entire host group. This can be done by placing files with the same name as the group in a **group\_vars** directory, and ensuring that those files contain settings for the connection variables.

For example, you might want all your Microsoft Windows managed hosts to use the **winrm** protocol and port 5986 for connections. To configure this, you could put all of those managed hosts in group **windows**, and then create a file named **group\_vars/windows** containing the following lines:

ansible\_connection: winrm

ansible\_port: 5986

Inventory file

[myself]

localhost

[intranetweb]

servera.lab.example.com

[everyone:children]

myself

intranetweb

[student@workstation dep-manage]$ **ansible myself --list-hosts**

hosts (1):

localhost

[student@workstation dep-manage]$ **ansible intranetweb --list-hosts**

hosts (1):

servera.lab.example.com

[student@workstation dep-manage]$ **ansible everyone --list-hosts**

hosts (2):

localhost

servera.lab.example.com

[student@workstation dep-manage]$ **ansible intranetweb --list-hosts -v**

Using /home/student/dep-manage/ansible.cfg as config file

SUDO password: **student**

hosts (1):

servera.lab.example.com

The complete **ansible.cfg** file looks like this:

[defaults]

inventory = ./inventory

[privilege\_escalation]

become = true

become\_method = sudo

become\_user = root

become\_ask\_pass = true

[student@workstation dep-manage]$ **ansible intranetweb --list-hosts -v**

Using /home/student/dep-manage/ansible.cfg as config file

SUDO password: **student**

hosts (1):

servera.lab.example.com

# [Introduction To Ad-Hoc Commands](http://docs.ansible.com/ansible/latest/intro_adhoc.html#id7)

What’s an ad-hoc command?

An ad-hoc command is something that you might type in to do something really quick, but don’t want to save for later.

This is a good place to start to understand the basics of what Ansible can do prior to learning the playbooks language – ad-hoc commands can also be used to do quick things that you might not necessarily want to write a full playbook for.

Run an **ansible** command targeting the **everyone** group, but replace **--list-hosts** with **-m ping**. That will run the **ping** module, which confirms that you can successfully run Ansible modules that use Python on the managed hosts.

[student@workstation dep-manage]$ **ansible everyone -m ping**

SUDO password: **student**

localhost | SUCCESS => {

"changed": false,

"ping": "pong"

}

servera.lab.example.com | SUCCESS => {

"changed": false,

"ping": "pong"

}

## Adhoc [Parallelism and Shell Commands](http://docs.ansible.com/ansible/latest/intro_adhoc.html#id8)

$ ansible atlanta -a "/sbin/reboot" -f 10 (where f= fork, which means 10 servers at a time)

$ ansible raleigh -m shell -a 'echo $TERM'

ansible atlanta -m copy -a "src=/etc/hosts dest=/tmp/hosts"

$ ansible webservers -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"