

# Implementing an Ansible Playbook

# Building an Ansible Inventory

## What is Inventory?

- An inventory defines a **Collection of hosts** that Ansible will manage.
- These hosts can also be assigned to **groups**, which can be **managed collectively**.
- Groups can contain **child groups**, and hosts can be members of multiple groups.
- The inventory can also set variables that apply to the hosts and groups that it defines.

```
[New_York]
ansi-node1
ansi-node2
[Los_Angeles]
ansi-node1
ansi-node3
[US:children]
New_York
Los_Angeles
```

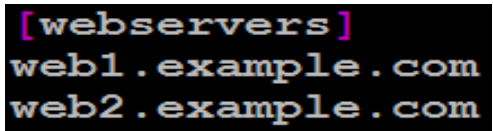
# Ways of defining inventory

- Host inventories can be defined in **two** different ways.
  - a. A **static host** inventory can be defined by a **text file**.
  - b. A **dynamic host** inventory can be generated by a **script** or **other program** as needed, using external information providers.

# Specifying Managed Hosts with a Static Inventory(1-3)

- A **static inventory** file is a **text file** that specifies the managed hosts that Ansible targets.
- You can write this file using a number of different formats, including **INI-style** or **YAML**.

# Specifying Managed Hosts with a Static Inventory(2-3)

- You organize managed hosts into **host groups**.
- Host groups allow you to more effectively run Ansible against a collection of systems.
- Each section starts with a host group name enclosed in square brackets [].  

- This is followed by the host name or an IP address for each managed host in the group, each on a single line.
- In the following example, the host inventory defines two host groups: **webservers** and **dbservers**.

# Specifying Managed Hosts with a Static Inventory(3-3)

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

```
[webservers]
web1.example.com
web2.example.com
192.0.2.42
[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
```

# Defining Nested Groups

- Ansible host inventories can include groups of host groups.
- A group can have both **managed hosts** and **child groups** as members.
- A list of hosts can be merged with the additional hosts groups which is inherited from its child group. This is accomplished by creating a host group name with the :children suffix.
- The following example creates a new group called north-america, which includes all hosts from the usa and canada groups.

```
[New_York]
ansi-node1
ansi-node2
[Los_Angeles]
ansi-node1
ansi-node3
[US:children]
New_York
Los_Angeles
```

# Simplifying Host Specifications with Ranges

You can specify ranges in the **host names** or IP addresses to simplify Ansible host inventories.

You can specify either **numeric** or **alphabetic** ranges. Ranges have the following syntax:

Ranges match all values from **START** to **END**, inclusively. Consider the following example:

- `server[01:20].example.com` matches all hosts named `server01.example.com` through `server20.example.com`.



# Simplifying Host Specifications with Ranges

If leading zeros are included in numeric ranges, they are used in the pattern. The 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]
washington1.example.com
washington2.example.com
[canada]
ontario01.example.com
ontario02.example.com
[north-america:children]
canada
usa
```

# Verifying the Inventory

use the ansible command to verify a machine's presence in the inventory:

```
[root@ansimaster ~]# ansible washington1.example.com --list-hosts
hosts (1):
    washington1.example.com
[root@ansimaster ~]# 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:

```
[root@ansimaster ~]# 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**.
- Normal practice is not to use that file but to define a different location for inventory files in your Ansible configuration file.
- The **ansible** and **ansible-playbook** commands that you use to run Ansible **ad hoc commands** and **playbooks** 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.



# Describing a Dynamic Inventory

- Ansible inventory information can also be dynamically generated, using information provided by **external databases**.
- A dynamic inventory program could contact your **Amazon EC2** account, and use information stored there to construct an Ansible inventory.
- Because the program does this when you run Ansible, it can populate the inventory with **up-to-date** information provided by the service as new hosts are added and old hosts are removed.

**Ansible  
Dynamic  
Inventory**

# Managing Ansible Configuration Files

- 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**
  - Using **~/.ansible.cfg**
  - Using **./ansible.cfg**

# 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** file 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.

# Using the **ANSIBLE\_CONFIG** environment variable

- 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

- The search order for a configuration file is the reverse of the preceding list.
- The first file located in the search order is the one that Ansible selects. Ansible only uses configuration settings from the first file that it finds.
- Any file specified by the **ANSIBLE\_CONFIG** environment variable overrides all other configuration files. If that variable is not set, the directory in which the ansible command was run is then checked for an **ansible.cfg** file. If that file is not present, the user's home directory is checked for a .ansible.cfg file. The global **/etc/ansible/ansible.cfg** file is only used if no other configuration file is found.
- If the **/etc/ansible/ansible.cfg** configuration file is not present, Ansible contains defaults which it uses.

# Configuration File Precedence

```
[root@ansi-master ansible]# ansible --version
ansible 2.9.27
  config file = /root/ansible/ansible.cfg
  configured module search path = ['/root/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3.6/site-packages/ansible
  executable location = /usr/bin/ansible
  python version = 3.6.8 (default, Sep 10 2021, 09:13:53) [GCC 8.5.0 20210514 (Red Hat 8.5.0-3)]
```

# 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. For basic operation use the following two sections:
  - **[defaults]** sets defaults for Ansible operation
  - **[privilege\_escalation]** configures how Ansible performs privilege escalation on managed hosts

The following is a typical ansible.cfg file:

```
[defaults]
inventory = ./inventory
remote_user = user
ask_pass = false

[privilege_escalation]
become = true
become_method = sudo
become_user = root
become_ask_pass = false
```

# Ansible Configuration

directive	description
<b>Inventory</b>	Specifies the path to the inventory file.
<b>Remote User</b>	The name of the user to log in as on the managed hosts. If not specified, the current user's name is used
<b>Ask_pass</b>	Whether or not to prompt for an SSH password. Can be false if using SSH public key authentication.
<b>Become</b>	Whether to automatically switch user on the managed host (typically to root) after connecting. This can also be specified by a play.
<b>Become_method</b>	How to switch user (typically sudo, which is the default, but su is an option).
<b>Become_user</b>	The user to switch to on the managed host (typically root, which is the default).
<b>Become_ask_pass</b>	Whether to prompt for a password for your become_method. Defaults to false.

# Configuring Connections

Ansible needs to know how to communicate with its managed hosts. One of the most common reasons to change the configuration file is to control which methods and users Ansible uses to administer managed hosts. Some of the information

- The location of the inventory that lists the **managed hosts** and **host groups**
- Which connection protocol to use to communicate with the managed hosts (by default, SSH), and whether or not a nonstandard network port is needed to connect to the server
- Which remote user to use on the managed hosts; this could be root or it could be an unprivileged user
- If the remote user is unprivileged, Ansible needs to know if it should try to escalate privileges to root and how to do it (for example, by using sudo)
- Whether or not to prompt for an SSH password or sudo password to log in or gain privileges needed includes:

# Inventory location

- In the **[defaults]** section, the inventory directive can point directly to a static inventory file, or to a directory containing multiple static inventory files and dynamic inventory scripts.

```
[defaults]  
inventory = ./inventory
```



# Connection Settings

- 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.
- 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.

```
[defaults]  
inventory = ./inventory  
remote_user = root  
ask_pass = true
```

# Connection Settings

- using a Linux control node and **OpenSSH** on your managed hosts, if you can log in as 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.
- For a single existing managed host, you can install your public key on the managed host and use the **ssh-copy-id** command to populate your local **~/.ssh/known\_hosts** file with its host key, as follows:

```
[root@ansimaster ~]# ssh-copy-id root@web1.example.com
The authenticity of host 'web1.example.com (192.168.122.181)' can't be established.
ECDSA key fingerprint is 70:9c:03:cd:de:ba:2f:11:98:fa:a0:b3:7c:40:86:4b.
Are you sure you want to continue connecting (yes/no)? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out a
ny that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted
now it is to install the new keys
root@web1.example.com's password:

Number of key(s) added: 1

Now try logging into the machine, with: "ssh 'root@web1.example.com'"
and check to make sure that only the key(s) you wanted were added.
```

## Note

You can also use an Ansible Playbook to deploy your public key to the remote\_user account on *all* managed hosts using the authorized\_key module.

A play that ensures that your public key is deployed to the managed hosts' root accounts might read as follows:

```
- name: Public key is deployed to managed hosts for Ansible
  hosts: all

  tasks:
    - name: Ensure key is in root's ~/.ssh/authorized_hosts
      authorized_key:
        user: root
        state: present
        key: '{{ item }}'
      with_file:
        - ~/.ssh/id_rsa.pub
```

# Escalating Privileges

- For **Security** and **Auditing reasons**, Ansible might need to connect to remote hosts as an **unprivileged 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 =** in the configuration file.
- Even if this is set by default, there are various ways to override it when running ad ho commands or Ansible Playbooks.

# Escalating Privileges

- 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 = true

[privilege_escalation]
become = true
become_method = sudo
become_user = root
become_ask_pass = false
```

# Configuration File Comments

- There are two comment characters allowed by Ansible configuration files: the **hash** or **number sign** (#) and the **semicolon** (;).
- The number sign at the start of a line comments out the entire line.
- It must not be on the same line with a directive.
- The semicolon character comments out everything to the right of it on the line. It can be on the same line as a directive, as long as that directive is to its left.



# Running Ad Hoc Commands

- An ad hoc command is a way of executing a single Ansible task quickly, one that you do not need to save to run again later.
- Ad hoc commands are useful for quick tests and changes.

For example,

- you can use an ad hoc command to make sure that a certain line exists in the **/etc/hosts** file on a group of servers
- You could use another ad hoc command to efficiently restart a service on many different machines, or to ensure that a particular software package is up-to-date.
- Ad hoc commands are very useful for quickly performing simple tasks with Ansible. They do have their limits, and in general you will want to use Ansible Playbooks to realize the full power of Ansible.



# Running Ad Hoc Commands

- Use the ansible command to run ad hoc commands:

```
ansible host-pattern -m module [-a 'module arguments'] [-i inventory]
```

- The host-pattern argument is used to specify the managed hosts on which the ad hoc command should be run. It could be a specific managed host or host group in the inventory.
- The -m option takes as an argument the name of the module that Ansible should run on the targeted hosts.
- One of the simplest ad hoc commands uses the ping module.

```
[root@ansi-master ~]# ansible all -m ping
BECOME password:
ansi-node1 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/libexec/platform-python"
  },
  "changed": false,
  "ping": "pong"
}
ansi-node2 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/libexec/platform-python"
  },
  "changed": false,
  "ping": "pong"
}
ansi-node3 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/libexec/platform-python"
  },
  "changed": false,
  "ping": "pong"
}
```

# Performing Tasks with Modules Using Ad Hoc Commands

- **Modules** are the tools that ad hoc commands use to accomplish tasks. Ansible provides **hundreds of modules** which do different things. You can usually find a **tested, special-purpose** module that does what you need as part of the standard installation.
- The **ansible-doc -l** command lists all modules installed on a system. You can use **ansibledoc** to view the documentation of particular modules by name, and find information about what arguments the modules take as options.

# Performing Tasks with Modules Using Ad Hoc Commands

```
[root@ansi-master ansible]# ansible-doc ping
> PING      (/usr/lib/python3.6/site-packages/ansible/modules/system/ping.py)

A trivial test module, this module always returns 'pong' on successful contact. It does not make sense in
playbooks, but it is useful from '/usr/bin/ansible' to verify the ability to login and that a usable Python
is configured. This is NOT ICMP ping, this is just a trivial test module that requires Python on the remote-
node. For Windows targets, use the [win_ping] module instead. For Network targets, use the [net_ping] module
instead.

* This module is maintained by The Ansible Core Team
OPTIONS (= is mandatory):

- data
  Data to return for the 'ping' return value.
  If this parameter is set to 'crash', the module will cause an exception.
  [Default: pong]
  type: str

SEE ALSO:
  * Module net_ping
    The official documentation on the net_ping module.
    https://docs.ansible.com/ansible/2.9/modules/net_ping_module.html
  * Module win_ping
    The official documentation on the win_ping module.
    https://docs.ansible.com/ansible/2.9/modules/win_ping_module.html

AUTHOR: Ansible Core Team, Michael DeHaan
METADATA:
  status:
  - stableinterface
  supported_by: core
```

The following table lists a number of useful modules as examples. Many others exist.

Module category	Modules
Files modules	<ul style="list-style-type: none"><li>• <code>copy</code>: Copy a local file to the managed host</li><li>• <code>file</code>: Set permissions and other properties of files</li><li>• <code>lineinfile</code>: Ensure a particular line is or is not in a file</li><li>• <code>synchronize</code>: Synchronize content using rsync</li></ul>
Software package	<ul style="list-style-type: none"><li>• <code>package</code>: Manage packages using autodetected package manager native to the operating system</li><li>• <code>yum</code>: Manage packages using the YUM package manager</li><li>• <code>apt</code>: Manage packages using the APT package manager</li><li>• <code>dnf</code>: Manage packages using the DNF package manager</li><li>• <code>gem</code>: Manage Ruby gems</li><li>• <code>pip</code>: Manage Python packages from PyPI</li></ul>
modules	<ul style="list-style-type: none"><li>• <code>firewalld</code>: Manage arbitrary ports and services using <code>firewalld</code></li><li>• <code>reboot</code>: Reboot a machine</li><li>• <code>service</code>: Manage services</li><li>• <code>user</code>: Add, remove, and manage user accounts</li></ul>
Net Tools modules	<ul style="list-style-type: none"><li>• <code>get_url</code>: Download files over HTTP, HTTPS, or FTP</li><li>• <code>nmcli</code>: Manage networking</li><li>• <code>uri</code>: Interact with web services</li></ul>

- Most modules are **idempotent**, which means that they can be run **safely multiple times**, and if the system is already in the **correct state**, they **do nothing**. For example, if you run the previous ad hoc command again, it should report **no change**:

```
[root@ansimaster ~]# ansible -m user -a 'name=newbie uid=4000 state=present' \
>servera.lab.example.com
servera.lab.example.com | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/libexec/platform-python"
  },
  "changed": true,
  "comment": "",
  "createhome": true,
  "group": 4000,
  "home": "/home/newbie",
  "name": "newbie",
  "shell": "/bin/bash",
  "state": "present",
  "system": false,
  "uid": 4000
}
```

# Running Arbitrary Commands on Managed Hosts

- The command module allows administrators to run arbitrary commands on the command line of managed hosts.
- The command to be run is specified as an argument to the module using the `-a` option.
- the following command runs the `hostname` command on the managed hosts referenced by the `mymanagedhosts` host pattern.

```
[root@ansimaster ~]# ansible mymanagedhosts -m command -a /usr/bin/hostname
host1.lab.example.com | CHANGED | rc=0 >>
host1.lab.example.com
host2.lab.example.com | CHANGED | rc=0 >>
host2.lab.example.com
```

The following table shows the analogous command-line options for each configuration file directive.

- Before configuring these directives using command-line options, their currently defined values can be determined by consulting the output of **ansible --help**.

[illegible]





ANSIBLE  
Playbooks

# Writing and Running Playbooks

- **Ansible Playbooks and Ad Hoc Commands**

Ad hoc commands can run a single, simple task against a set of targeted hosts as a one-time command.

- A **task** is the **application** of a module to perform a specific unit of work.
- A **play** is a sequence of tasks to be applied, in order, to one or more hosts selected from your inventory.
- A **playbook** is a text file containing a list of one or more plays to run in a specific order.
- Plays allow you to change a lengthy, complex set of manual administrative tasks into an easily repeatable routine with predictable and successful outcomes.
- In a playbook, you can **save** the sequence of tasks in a play into a **human-readable** and immediately runnable form.

# Formatting an Ansible Playbook

- A playbook is a **text file** written in **YAML format**, and is normally saved with the **extension yml**. The playbook uses **indentation** with space characters to indicate the structure of its data.
- **YAML** does not place **strict requirements** on how many spaces are used for the indentation, but there are **two basic** rules.
- Data elements at the same level in the hierarchy (such as items in the same list) must have the
  - **Same Indentation.**
  - **Items that are children of another item must be indented more than their parents.**

# Formatting an Ansible Playbook

- A playbook begins with a line consisting of three dashes (---) as a start of document marker. It may end with three dots (...) as an end of document marker
- In between those markers, the playbook is defined as a list of plays.
- An item in a YAML list starts with a single dash followed by a space. For example, a YAML list might appear as follows:

```
- apple  
- orange  
- grape
```

- In the preceding playbook example, the line after --- begins with a dash and starts the first (and only) play in the list of plays.
- The play itself is a collection of key-value pairs. Keys in the same play should have the same indentation.

# Formatting an Ansible Playbook

- The following example shows a **YAML** snippet with three keys. The first two keys have simple values. The third has a list of three items as a value.

```
name: just an example
hosts: webservers
tasks:
  - first
  - second
  - third
```

- The first line of the **example play** starts with a **dash** and a **space** (indicating the play is the first item of a list), and then the first key, the name attribute. The name key associates an arbitrary string with the play as a label. This identifies what the play is for. The name key is optional, but is recommended because it helps to document your playbook. This is especially useful when a playbook contains multiple plays.

# Formatting an Ansible Playbook

```
- name: Configure important user consistently
```

- The second key in the play is a hosts attribute, which specifies the hosts against which the play's tasks are run. Like the argument for the ansible command, the hosts attribute takes a host pattern as a value, such as the names of managed hosts or groups in the inventory.

```
hosts: servera.lab.example.com
```

- Finally, the last key in the play is the tasks attribute, whose value specifies a list of tasks to run for this play. This example has a single task, which runs the user module with specific arguments

```
tasks:  
  - name: newbie exists with UID 4000  
    user:  
      name: newbie  
      uid: 4000  
      state: present
```

# Formatting an Ansible Playbook

The tasks attribute is the part of the play that actually lists, in order, the tasks to be run on the managed hosts.

Each task in the list is itself a collection of key-value pairs.

In this example, the only task in the play has two keys:

- name is an optional label documenting the purpose of the task. It is a good idea to name all your tasks to help document the purpose of each step of the automation process.
- user is the module to run for this task. Its arguments are passed as a collection of key-value pairs, which are children of the module

The following is another example of a tasks attribute with multiple tasks, using the service module to ensure that several network services are enabled to start at boot:

```
tasks:
  - name: web server is enabled
    service:
      name: httpd
      enabled: true

  - name: NTP server is enabled
    service:
      name: chronyd
      enabled: true

  - name: Postfix is enabled
    service:
      name: postfix
      enabled: true
```

# Running playbooks

- The `ansible-playbook` command is used to run playbooks.
- The command is executed on the control node and the name of the playbook to be run is passed as an argument:

```
[root@ansimaster ~]# ansible-playbook site.yml
```

- When you run the playbook, output is generated to show the play and tasks being executed. The output also reports the results of each task executed.



# Increasing Output Verbosity

- The default output provided by the `ansible-playbook` command does not provide detailed task execution information. The `ansible-playbook -v` command provides additional information, with up to four total levels.

## Configuring the output verbosity of playbook Execution

Option	Description
<code>-v</code>	The task results are displayed.
<code>-vv</code> <small>~~~~~</small>	Both task results and task configuration are displayed
<code>-vvv</code> <small>~~~~~</small>	Includes information about connections to managed hosts
<code>-vvvv</code> <small>~~~~~</small>	Adds extra verbosity options to the connection plug-ins, including users being used in the managed hosts to execute scripts, and what scripts have been executed

# Syntax Verification

- Prior to executing a playbook, it is good practice to perform a verification to ensure that the syntax of its contents is correct.
- The `ansible-playbook` command offers a **--syntax-check** option that you can use to verify the syntax of a playbook. The following example shows the successful syntax verification of a playbook.

```
[root@ansimaster ~]# ansible-playbook --syntax-check webserver.yml  
playbook: webserver.yml
```

- When syntax verification fails, a syntax error is reported. The output also includes the approximate location of the syntax issue in the playbook.
- For Example:

```
[root@ansimaster ~]# ansible-playbook --syntax-check webserver.yml
ERROR! Syntax Error while loading YAML.
mapping values are not allowed in this context

The error appears to have been in ...output omitted... line 3, column 8, but may
be elsewhere in the file depending on the exact syntax problem.

The offending line appears to be:

- name:play to setup web server
hosts: servera.lab.example.com
    ^ here
```

# Executing a Dry Run

- You can use the `-C` option to perform a dry run of the playbook execution. This causes Ansible to report what changes would have occurred if the playbook were executed, but does not make any actual changes to managed hosts.
- The following example shows the dry run of a playbook containing a single task for ensuring that the latest version of *httpd* package is installed on a managed host.

```
[root@ansimaster ~]# ansible-playbook -C webserver.yml

PLAY [play to setup web server] *****

TASK [Gathering Facts] *****
ok: [servera.lab.example.com]

TASK [latest httpd version installed] *****
changed: [servera.lab.example.com]

PLAY RECAP *****
servera.lab.example.com : ok=2  changed=1    unreachable=0    failed=0
```

# Implementing Multiple Plays

## Writing Multiple Plays

- A playbook is a YAML file containing a list of one or more plays. Remember that a single play is an ordered list of tasks to execute against hosts selected from the inventory
- a playbook contains multiple plays, each play may apply its tasks to a separate set of hosts.
- You can write a playbook that runs one play against one set of hosts, and when that finishes runs another play against another set of hosts.
- Writing a playbook that contains multiple plays is very straightforward. Each play in the playbook is written as a top-level list item in the playbook. Each play is a list item containing the usual play keywords.

- The following example shows a simple playbook with two plays. The first play runs against web.example.com, and the second play runs against database.example.com.

```
---
# This is a simple playbook with two plays

- name: first play
  hosts: web.example.com
  tasks:
    - name: first task
      yum:
        name: httpd
        status: present

    - name: second task
      service:
        name: httpd
        enabled: true

- name: second play
  hosts: database.example.com
  tasks:
    - name: first task
      service:
        name: mariadb
        enabled: true
```

# Remote Users and Privilege Escalation in Plays

Plays can use different remote users or privilege escalation settings for a play than what is specified by the defaults in the configuration file. These are set in the play itself at the same level as the hosts or tasks keywords.

- **User Attributes**
- **Privilege Escalation Attributes**

# User Attributes

- Tasks in playbooks are normally executed through a network connection to the managed hosts. As with ad hoc commands, the user account used for task execution depends on various keywords in the Ansible configuration file, /etc/ansible/ansible.cfg.
- The user that runs the tasks can be defined by the `remote_user` keyword.
- if privilege escalation is enabled, other keywords such as `become_user` can also have an impact.
- If the remote user defined in the Ansible configuration for task execution is not suitable, it can be overridden by using the `remote_user` keyword within a play.

```
remote_user: remoteuser
```



# Privilege Escalation Attributes

- Additional keywords are also available to define privilege escalation parameters from within a playbook. The `become` boolean keyword can be used to enable or disable privilege escalation regardless of how it is defined in the Ansible configuration file. It can take `yes` or `true` to enable privilege escalation, or `no` or `false` to disable it.

```
become: true
```

- If privilege escalation is enabled, the `become_method` keyword can be used to define the privilege escalation method to use during a specific play.

For Example:

```
become_method: sudo
```

# Privilege Escalation Attributes

- Additionally, with privilege escalation enabled, the `become_user` keyword can define the user account to use for privilege escalation within the context of a specific play.

```
become_user: privileged_user
```

- The following example demonstrates the use of these keywords in a play:

```
- name: /etc/hosts is up to date
  hosts: datacenter-west
  remote_user: automation
  become: yes

tasks:
  - name: server.example.com in /etc/hosts
    lineinfile:
      path: /etc/hosts
      line: '192.0.2.42 server.example.com server'
      state: present
```

# Finding Modules for Tasks

## Module Documentation

- The large number of modules packaged with Ansible provides administrators with many tools for common administrative tasks.
- The Module Index on the website is an easy way to browse the list of modules shipped with Ansible.
- For example, modules for user and service management can be found under Systems Modules and modules for database administration can be found under Database Modules.
- For each module, the Ansible documentation website provides a summary of its functions and
- instructions on how each specific function can be invoked with options to the module.

- to see a list of the modules available on a control node, run the `ansible-doc -l` command. This displays a list of module names and a synopsis of their functions.

```
[root@ansimaster ~]# ansible-doc -l
a10_server          Manage A10 Networks ... devices' server object.
a10_server_axapi3    Manage A10 Networks ... devices
a10_service_group    Manage A10 Networks ... devices' service groups.
a10_virtual_server   Manage A10 Networks ... devices' virtual servers.
...output omitted...
zfs_facts            Gather facts about ZFS datasets.
znode                Create, ... and update znodes using ZooKeeper
zpool_facts          Gather facts about ZFS pools.
zypper               Manage packages on SUSE and openSUSE
zypper_repository    Add and remove Zypper repositories
```

- Use the `ansible-doc [module name]` command to display detailed documentation for a module.

# Finding Modules for Tasks

- The `ansible-doc` command also offers the `-s` option, which produces example output that can serve as a model for how to use a particular module in a playbook.
- output can serve as a starter template, which can be included in a playbook to implement the module for task execution.
- Comments are included in the output to remind administrators of the use of each option.

```
[root@ansimaster]# ansible-doc yum
>YUM      (/usr/lib/python3.6/site-packages/ansible/modules/packaging/os/yum.py)

Installs, upgrade, downgrades, removes, and lists packages and groups with
the 'yum' package manager. This module only works on Python 2. If you require
Python
    3 support see the [dnf] module.

*This module is maintained by The Ansible Core Team
*note: This module has a corresponding action plugin.

OPTIONS (= is mandatory):
```

- For Example, showing this output for yum module.

```
[root@ansimaster ~]# ansible-doc -s yum
- name: Manages packages with the `yum` package manager
  yum:
    allow_downgrade:      # Specify if the named package ...
    autoremove:           # If `yes`, removes all "leaf" packages ...
    bugfix:               # If set to `yes`, ...
    conf_file:            # The remote yum configuration file ...
    disable_excludes:     # Disable the excludes ...
    disable_gpg_check:    # Whether to disable the GPG ...
    disable_plugin:       # `Plugin` name to disable ...
    disablerepo:          # `RepoId` of repositories ...
    download_only:        # Only download the packages, ...
    enable_plugin:        # `Plugin` name to enable ...
    enablerepo:           # `RepoId` of repositories to enable ...
    exclude:              # Package name(s) to exclude ...
    installroot:          # Specifies an alternative installroot, ...
    list:                 # Package name to run ...
    name:                 # A package name or package specifier ...
    releasever:           # Specifies an alternative release ...
    security:             # If set to `yes`, ...
    skip_broken:          # Skip packages with ...
    state:                # Whether to install ... or remove ... a package.
    update_cache:         # Force yum to check if cache ...
    update_only:          # When using latest, only update ...
    use_backend:          # This module supports `yum` ...
    validate_certs:       # This only applies if using a https url ...
```

# Module Maintenance

- The ansible-doc documentation for the module is expected to specify who maintains that module in the upstream Ansible community, and what its development status is. This is indicated in the METADATA section at the end of the output of ansible-doc for that module.
- The status field records the development status of the module:
  - **Stable interface**
  - **Preview**
  - **Deprecated**
  - **removed**

# Module Maintenance

- **Stable interface:** The module's keywords are stable, and every effort will be made not to remove keywords or change their meaning.
- **preview:** The module is in technology preview, and might be unstable, its keywords might change, or it might require libraries or web services that are themselves subject to incompatible changes.
- **deprecated:** The module is deprecated, and will no longer be available in some future release.
- **removed:** The module has been removed from the release, but a stub exists for documentation purposes to help former users migrate to new modules.



# Module Maintenance

- The `supported_by` field records who maintains the module in the upstream Ansible community. Possible values are:
  - Core
  - Curated
  - Community
- **Core:** Maintained by the "core" Ansible developers upstream, and always included with Ansible.
- **Curated:** Modules submitted and maintained by partners or companies in the community. Maintainers of these modules must watch for any issues reported or pull requests raised against the module. Upstream "core" developers review proposed changes to curated modules after the community maintainers have approved the changes.

# Community:

- Modules not supported by the core upstream developers, partners, or companies, but maintained entirely by the general open source community.
- Modules in this category are still fully usable, but the response rate to issues is purely up to the community.
- As an end user, you can also write your own private modules, or get modules from a third party. Ansible searches for custom modules in the location specified by the `ANSIBLE_LIBRARY` environment variable, or if that is not set, by a `library` keyword in the current Ansible configuration file.
- Ansible also searches for custom modules in the `./library` directory relative to the playbook currently being run.

```
library = /usr/share/my_modules
```