

# Ansible Tutorial for Beginners: Playbook, Commands & Example

## What is Ansible?

**Ansible** is an open source automation and orchestration tool for software provisioning, configuration management, and software deployment. Ansible can easily run and configure Unix-like systems as well as Windows systems to provide infrastructure as code. It contains its own declarative programming language for system configuration and management.

Ansible is popular for its simplicity of installation, ease of use in what concerns the connectivity to clients, its lack of agent for Ansible clients and the multitude of skills. It functions by connecting via [SSH](#) to the clients, so it doesn't need a special agent on the client-side, and by pushing modules to the clients, the modules are then executed locally on the client-side and the output is pushed back to the Ansible server.

Since it uses SSH, it can very easily connect to clients using SSH-Keys, simplifying though the whole process. Client details, like hostnames or IP addresses and SSH ports, are stored in files called inventory files. Once you have created an inventory file and populated it, ansible can use it.

In this Ansible tutorial for beginners, you will learn Ansible step by step:

- [What is Ansible?](#)
- [Why use Ansible?](#)
- [History of Ansible](#)
- [Important terms used in Ansible](#)
- [Ansible Installation in Linux](#)
- [Ansible ad-hoc commands](#)
- [Ansible Playbooks](#)
- [Ansible Roles](#)
- [Ansible Case Study](#)
- [Ansible Commands Cheat Sheet](#)

## Why use Ansible?

Here are some important pros/benefits of using Ansible

- One of the most significant advantages of Ansible is that it is free to use by everyone.
- It does not need any special system administrator skills to install and use Ansible, and the official documentation is very comprehensive.
- Its modularity regarding plugins, modules, inventories, and playbooks make Ansible the perfect companion to orchestrate large environments.

- Ansible is very lightweight and consistent, and no constraints regarding the operating system or underlying hardware are present.
- It is also very secure due to its agentless capabilities and due to the use of OpenSSH security features.
- Another advantage that encourages the adoption of Ansible is its smooth learning curve determined by the comprehensive documentation and easy to learn structure and configuration.

## History of Ansible

Here, are important land marks from the history of ansible:

- In February 2012 the Ansible project began. It was first developed by Michael DeHaan, the creator of Cobbler and Func, Fedora Unified Network Controller.
- Initially called AnsibleWorks Inc, the company funding the ansible tool was acquired in 2015 by RedHat and later on, along with RedHat, moved under the umbrella of IBM.
- In the present, Ansible comes included in distributions like Fedora Linux, RHEL, Centos and Oracle Linux.

## Important terms used in Ansible

- **Ansible server:**

The machine where Ansible is installed and from which all tasks and playbooks will be ran

- **Module:**

Basically, a module is a command or set of similar Ansible commands meant to be executed on the client-side

- **Task:**

A task is a section that consists of a single procedure to be completed

- **Role:**

A way of organizing tasks and related files to be later called in a playbook

- **Fact:**

Information fetched from the client system from the global variables with the gather-facts operation

- **Inventory:**

File containing data about the ansible client servers. Defined in later examples as hosts file

- **Play:**

Execution of a playbook

- **Handler:**

Task which is called only if a notifier is present

- **Notifier:**

Section attributed to a task which calls a handler if the output is changed

- **Tag:**

Name set to a task which can be used later on to issue just that specific task or group of tasks.

## Ansible Installation in Linux

Once you have compared and weighed your options and decided to go for Ansible, the next step is to have it installed on your system. We will go through the steps of installation in different [Linux](#) distributions, the most popular ones, in the next small tutorial.

### Install Ansible on Centos/RedHat systems

#### Step 1) Install EPEL repo

```
[root@ansible-server ~]# sudo yum install epel-release
```

#### Step 2) Install ansible package

```
[root@ansible-server ~]# sudo yum install -y ansible
```

```
Installed:
ansible.noarch 0:2.4.2.0-2.el7

Dependency Installed:
python-cffi.x86_64 0:1.6.0-5.el7          python-enum34.noarch 0:1.0.4-1.el7          python-httplib2.noarch 0:0.9.2-1.el7
python-idna.noarch 0:2.4-1.el7           python-paramiko.noarch 0:2.1.1-9.el7          python-passlib.noarch 0:1.6.5-2.el7
python-ply.noarch 0:3.4-11.el7           python-pycparser.noarch 0:2.14-1.el7          python2-cryptography.x86_64 0:1.7.2-2.el7
python2-jmespath.noarch 0:0.9.0-3.el7        python2-pyasn1.noarch 0:0.1.9-7.el7          sshpass.x86_64 0:1.06-2.el7

Complete!
```

## Install ansible on Ubuntu/Debian systems

**Step 1)** Perform an update to the packages

```
$ sudo apt update
```

**Step 2)** Install the software-properties-common package

```
$ sudo apt install software-properties-common
```

**Step 3)** Install ansible personal package archive

```
$ sudo apt-add-repository ppa:ansible/ansible
```

**Step 4)** Install ansible

```
$ sudo apt update  
$ sudo apt install ansible
```

## Ansible ad-hoc commands

One of the simplest ways Ansible can be used is by using ad-hoc commands. These can be used when you want to issue some commands on a server or a bunch of servers. Ad-hoc commands are not stored for future uses but represent a fast way to interact with the desired servers.

For this Ansible tutorial, a simple two servers hosts file will be configured, containing host1 and host2.

You can make sure that the hosts are accessible from the ansible server by issuing a ping command on all hosts.

```
[root@ansible-server test_ansible]# ansible -i hosts all -m ping  
host1 | SUCCESS => {  
    "changed": false,  
    "ping": "pong"  
}  
host2 | SUCCESS => {  
    "changed": false,  
    "ping": "pong"  
}
```

```
[root@ansible-server test_ansible]# ansible -i hosts all -m ping
host1 | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
host2 | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
```

Explanation:

1. Status of the command, in this case, SUCCESS
2. Host on which the command ran
3. The command issued via the -m parameter, in this case, ping
4. With the -i parameter, you can point to the hosts file.

You can issue the same command only on a specific host if needed.

```
[root@ansible-server test_ansible]# ansible -i hosts all -m ping --limit host2
host2 | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
```

```
[root@ansible-server test_ansible]# ansible -i hosts all -m ping --limit host2
host2 | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
```

Explanation:

1. --limit parameter can be used to issue commands only on specific hosts in the host's file
2. Name of the host as defined in the inventory file

If you need to copy a file to multiple destinations rapidly, you can use the copy module in ansible which uses SCP. So the command and its output look like below:

```
[root@ansible-server test_ansible]# ansible -i hosts all -m copy -a "src=/root/test_ansi
ble/testfile dest=/tmp/testfile"
host1 | SUCCESS => {
```

```
    "changed": true,
    "checksum": "da39a3ee5e6b4b0d3255bfef95601890afd80709",
    "dest": "/tmp/testfile",
    "gid": 0,
    "group": "root",
    "md5sum": "d41d8cd98f00b204e9800998ecf8427e",
    "mode": "0644",
    "owner": "root",
    "size": 0,
    "src": "/root/.ansible/tmp/ansible-tmp-1562216392.43-256741011164877/source",
    "state": "file",
    "uid": 0
  }
host2 | SUCCESS => {
    "changed": true,
    "checksum": "da39a3ee5e6b4b0d3255bfef95601890afd80709",
    "dest": "/tmp/testfile",
    "gid": 0,
    "group": "root",
    "md5sum": "d41d8cd98f00b204e9800998ecf8427e",
    "mode": "0644",
    "owner": "root",
    "size": 0,
    "src": "/root/.ansible/tmp/ansible-tmp-1562216392.6-280302911361278/source",
    "state": "file",
    "uid": 0
  }
}
```

```
[root@ansible-server test_ansible]# ansible -i hosts all -m copy -a "src=/root/test_ansible/testfile dest=/tmp/testfile"
host1 | SUCCESS => {
  "changed": true,
  "checksum": "da39a3ee5e6b4b0d3255bfef95601890afd80709",
  "dest": "/tmp/testfile",
  "gid": 0,
  "group": "root",
  "md5sum": "d41d8cd98f00b204e9800998ecf8427e",
  "mode": "0644",
  "owner": "root",
  "size": 0,
  "src": "/root/.ansible/tmp/ansible-tmp-1562165182.37-113050681175629/source",
  "state": "file",
  "uid": 0
}
host2 | SUCCESS => {
  "changed": true,
  "checksum": "da39a3ee5e6b4b0d3255bfef95601890afd80709",
  "dest": "/tmp/testfile",
  "gid": 0,
  "group": "root",
  "md5sum": "d41d8cd98f00b204e9800998ecf8427e",
  "mode": "0644",
  "owner": "root",
  "size": 0,
  "src": "/root/.ansible/tmp/ansible-tmp-1562165182.57-108259253389109/source",
  "state": "file",
  "uid": 0
}
```

Explanation:

1. Copy module defined
2. Module arguments, in this case, are source absolute path and destination absolute path.
3. Ansible command output reflecting the success of the copy command and other details like the sha1 or md5 checksums for file integrity check and metadata like owner, size, or permissions.

It is effortless to have a package installed on a bunch of servers. Ansible has several modules that interact with used installers, like yum, apt, dnf, etc.

In the next example, you will find out how to install a package via the yum module on two Centos hosts.

```
[root@ansible-server test_ansible]# ansible -i hosts all -m yum -a 'name=ncdu state=present'
host1 | SUCCESS => {
  "changed": true,
  "msg": "",
  "rc": 0,
  "results": [
```

```

"Loaded plugins: fastestmirror\nLoading mirror speeds from cached hostfile\n * base:
mirror.netsite.dk\n * elrepo: mirrors.xservers.ro\n * epel: fedora.mirrors.telekom.ro
\n * extras: centos.mirrors.telekom.ro\n * remi-php70: remi.schlundtech.de\n * remi-s
afe: remi.schlundtech.de\n * updates: centos.mirror.iphh.net\nResolving Dependencies\n
--> Running transaction check\n--> Package ncdu.x86_64 0:1.14-1.el7 will be install
ed\n--> Finished Dependency Resolution\n\nDependencies Resolved\n\n=====
=====
\n Package                                Arch
\n
\nVersion                                Repository                                Size\n=====
\nInstalling:\n ncdu
\n
\nx86_64                                1.14-1.el7                                epel                                51 k\n\nTransaction Summa
ry\n=====
\nInstall 1 Package\n\nTotal download size: 51 k\nInstalled size: 87 k\nDownloading p
ackages:\nRunning transaction check\nRunning transaction test\nTransaction test succe
eded\nRunning transaction\n Installing : ncdu-1.14-1.el7.x86_64
\n
\n1/1 \n Verifying : ncdu-1.14-1.el7.x86_64
\n
\n1/1 \n\nInstalled:\n ncdu.x86_64 0:1.14-1.el7
\n\nComplete!\n"
]
}
host2 | SUCCESS => {
  "changed": true,
  "msg": "",
  "rc": 0,
  "results": [
    "Loaded plugins: fastestmirror\nLoading mirror speeds from cached hostfile\n
* base: mirror.netsite.dk\n * elrepo: mirrors.leadhosts.com\n * epel: mirrors.nav.ro\n
\n * extras: centos.mirrors.telekom.ro\n * remi-php70: mirrors.uni-ruse.bg\n * remi-sa
fe: mirrors.uni-ruse.bg\n * updates: centos.mirror.iphh.net\nResolving Dependencies\n
--> Running transaction check\n--> Package ncdu.x86_64 0:1.14-1.el7 will be installe
d\n--> Finished Dependency Resolution\n\nDependencies Resolved\n\n=====
=====
\n Package                                Arch
\n
\nVersion                                Repository                                Size\n=====
\nInstalling:\n ncdu
\n
\nx86_64                                1.14-1.el7                                epel                                51 k\n\nTransaction Summar
y\n=====
\nInstall 1 Package\n\nTotal download size: 51 k\nInstalled size: 87 k\nDownloading pa
ckages:\nRunning transaction check\nRunning transaction test\nTransaction test succee
ded\nRunning transaction\n Installing : ncdu-1.14-1.el7.x86_64
\n
\n1/1 \n Verifying : ncdu-1.14-1.el7.x86_64
\n
\n1/1 \n\nInstalled:\n ncdu.x86_64 0:1.14-1.el7
\n\nComplete!\n"
]
}

```



```

1 [root@ansible-server test_ansible]# ansible -i hosts all -m yum -a 'name=ncdu state=present'
2 host2 | SUCCESS => {
3   "changed": true,
   "msg": "",
   "rc": 0,
   "results": [
     "Loaded plugins: fastestmirror\nLoading mirror speeds from cached hostfile\n * base: mirror.netsite.dk\n * elrepo: mirrors.lead
     hosts.com\n * epel: fedora.mirrors.telekom.ro\n * extras: centos.mirrors.telekom.ro\n * remi-php70: mirrors.uni-ruse.bg\n * remi-safe:
     mirrors.uni-ruse.bg\n * updates: centos.mirror.ipph.net\nResolving Dependencies\n--> Running transaction check\n--> Package ncdu.x86_6
     4 0:1.14-1.el7 will be installed\n--> Finished Dependency Resolution\n\nDependencies Resolved\n\n-----
       \n Package          Arch          Version              Repository      Size\n-----
       \nInstalling:\n ncdu              x86_64         1.14-1.el7
     epel              51 k\n\nTransaction Summary\n-----
     \nInstall 1 Package\n\nTotal download size: 51 k\nInstalled size: 87 k\nDownloading packages:\nRunning transaction check\nRunning trans
     action test\nTransaction test succeeded\nRunning transaction\n Installing : ncdu-1.14-1.el7.x86_64
     1/1 \n Verifying : ncdu-1.14-1.el7.x86_64
     1/1 \n\nInstalled:\n ncdu.x86_64 0:1.14-1.el7
     ]
   }
   host1 | SUCCESS => {
     "changed": true,
     "msg": "",
     "rc": 0,
     "results": [
       "Loaded plugins: fastestmirror\nLoading mirror speeds from cached hostfile\n * base: mirror.netsite.dk\n * elrepo: mirrors.xser
       vers.ro\n * epel: fedora.mirrors.telekom.ro\n * extras: centos.mirrors.telekom.ro\n * remi-php70: remi.schlundtech.de\n * remi-safe: re
       mi.schlundtech.de\n * updates: centos.mirror.ipph.net\nResolving Dependencies\n--> Running transaction check\n--> Package ncdu.x86_64
       0:1.14-1.el7 will be installed\n--> Finished Dependency Resolution\n\nDependencies Resolved\n\n-----
       \n Package          Arch          Version              Repository      Size\n-----
       \nInstalling:\n ncdu              x86_64         1.14-1.el7
     epel              51 k\n\nTransaction Summary\n-----
     \nInstall 1 Package\n\nTotal download size: 51 k\nInstalled size: 87 k\nDownloading packages:\nRunning transaction check\nRunning transac
     tion test\nTransaction test succeeded\nRunning transaction\n Installing : ncdu-1.14-1.el7.x86_64
     1/1 \n Verifying : ncdu-1.14-1.el7.x86_64
     1/1 \n\nInstalled:\n ncdu.x86_64 0:1.14-1.el7
     ]
   }
 }

```

Explanation:

1. Yum module is used in this example
2. - It defines the module arguments, and in this case, you will choose the name of the package and its state. If the state is absent, for example, the package will be searched and if found, removed
3. When colored in yellow, you will see the output of the ansible command with the state changed, meaning in this case, that the package was found and installed.
4. Status of the yum install command issued via ansible. In this case the package ncdu.x86\_64 0:1.14-1.el7 was installed.

Of course, all of the yum installer options can be used via ansible, including update, install, latest version, or remove.

In the below example the same command was issued to remove the previously installed ncdu package.

```

[root@ansible-server test_ansible]# ansible -i hosts all -m yum -a 'name=ncdu state=absent'
host1 | SUCCESS => {
  "changed": true,
  "msg": "",
  "rc": 0,

```

```

    "results": [
        "Loaded plugins: fastestmirror\nResolving Dependencies\n--> Running transaction check\n--> Package ncdu.x86_64 0:1.14-1.el7 will be erased\n--> Finished Dependency Resolution\n\nDependencies Resolved\n\n=====
=====
\n Package                Arch                Version
Repository                Size\n=====
\nRemoving:\n ncdu                x86_64                1.14-1.
el7                @epel                87 k\n\nTransaction Summary\n=====
\nRemove  1 Package\n\nInstalled size: 87 k\nDownloading packages:\nRunning transaction check\nRunning transaction test\nTransaction test succeeded\nRunning transaction\n Erasing      : ncdu-1.14-1.el7.x86_64
1/1 \n Verifying  : ncdu-1.14-1.el7.x86_64
1/1 \n\nRemoved:\n ncdu.x86_64 0:1.14-1.el7
\n\nComplete!\n"
    ]
}
host2 | SUCCESS => {
    "changed": true,
    "msg": "",
    "rc": 0,
    "results": [
        "Loaded plugins: fastestmirror\nResolving Dependencies\n--> Running transaction check\n--> Package ncdu.x86_64 0:1.14-1.el7 will be erased\n--> Finished Dependency Resolution\n\nDependencies Resolved\n\n=====
=====
\n Package                Arch                Version
Repository                Size\n=====
\nRemoving:\n ncdu                x86_64                1.14-1.
el7                @epel                87 k\n\nTransaction Summary\n=====
\nRemove  1 Package\n\nInstalled size: 87 k\nDownloading packages:\nRunning transaction check\nRunning transaction test\nTransaction test succeeded\nRunning transaction\n Erasing      : ncdu-1.14-1.el7.x86_64
1/1 \n Verifying  : ncdu-1.14-1.el7.x86_64
1/1 \n\nRemoved:\n ncdu.x86_64 0:1.14-1.el7
\n\nComplete!\n"
    ]
}

```

```
[root@ansible-server test_ansible]# ansible -i hosts all -m yum -a 'name=ncdu state=absent'
```

```
host1 | SUCCESS => {
  "changed": true,
  "msg": "",
  "rc": 0,
  "results": [
    "Loaded plugins: fastestmirror\nResolving Dependencies\n--> Running transaction check\n--> Package ncdu.x86_64 0:1.14-1.el7 will be erased\n--> Finished Dependency Resolution\n\nDependencies Resolved\n\n-----\n\nPackage Arch Version Repository Size\n-----\n\nRemoving: ncdu x86_64 1.14-1.el7 @epel\n\n87 k\n\nTransaction Summary\n-----\n\nRemove 1 Package\n\nInstalled size: 87 k\nDownloading packages:\nRunning transaction check\nRunning transaction test\nTransaction test succeeded\nRunning transaction\nErasing : ncdu-1.14-1.el7.x86_64 1/1\nVerifying : ncdu-1.14-1.el7.x86_64 1/1\nRemoved: ncdu.x86_64 0:1.14-1.el7\n\nComplete!\n"]
  }
}

host2 | SUCCESS => {
  "changed": true,
  "msg": "",
  "rc": 0,
  "results": [
    "Loaded plugins: fastestmirror\nResolving Dependencies\n--> Running transaction check\n--> Package ncdu.x86_64 0:1.14-1.el7 will be erased\n--> Finished Dependency Resolution\n\nDependencies Resolved\n\n-----\n\nPackage Arch Version Repository Size\n-----\n\nRemoving: ncdu x86_64 1.14-1.el7 @epel\n\n87 k\n\nTransaction Summary\n-----\n\nRemove 1 Package\n\nInstalled size: 87 k\nDownloading packages:\nRunning transaction check\nRunning transaction test\nTransaction test succeeded\nRunning transaction\nErasing : ncdu-1.14-1.el7.x86_64 1/1\nVerifying : ncdu-1.14-1.el7.x86_64 1/1\nRemoved: ncdu.x86_64 0:1.14-1.el7\n\nComplete!\n"]
  }
}
```

Explanation:

1. The output of the yum command shows that the package was removed.

Another useful and essential feature that ansible uses to interact with the client's server is to gather some facts about the system. So, it fetches hardware, software, and versioning information from the system and stores each value in a variable that can be later on used.

If you need detailed information about the systems to be modified via ansible, the next command can be used. The setup module gathers facts from the system variables.

```
[root@ansible-server test_ansible]# ansible -i hosts all -m setup
```

## Ansible Playbooks

**Ansible Playbooks** are the way of sending commands to remote systems through scripts. Ansible playbooks are used to configure complex system environments to increase flexibility by executing a script to one or more systems. Ansible playbooks tend to be more of a configuration language than a programming language.

Ansible playbook commands use YAML format, so there is not much syntax needed, but indentation must be respected. Like the name is saying, a playbook is a collection of plays.

Through a playbook, you can designate specific roles to some hosts and other roles to other hosts. By doing so, you can orchestrate multiple servers in very diverse scenarios, all in one playbook.

To have all the details precise before continuing with Ansible playbook examples, we must first define a task. These are the interface to ansible modules for roles and playbooks.

Now, let's learn Ansible playbook through an example with one playbook with one play, containing multiple tasks as below:

```
---  
  
- hosts: group1  
  tasks:  
    - name: Install lldpad package  
      yum:  
        name: lldpad  
        state: latest  
    - name: check lldpad service status  
      service:  
        name: lldpad  
        state: started
```

```

---
1
- hosts: group1
  tasks:
2   name: Install lldpad package
    yum:
      name: lldpad
      state: latest
    - name: check lldpad service status
      service:
3      name: lldpad
      state: started
~

```

In the above Ansible playbook example, the group1 of hosts in the host's file is targeted for lldpad package installation using the yum module and afterward the service lldpad created after the installation is then started using the service module used mostly to interact with systemd ensemble.

Explanation:

1. Group of hosts on which the playbook will run
2. Yum module is used in this task for lldpad installation
3. The service module is used to check if the service is up and running after installation

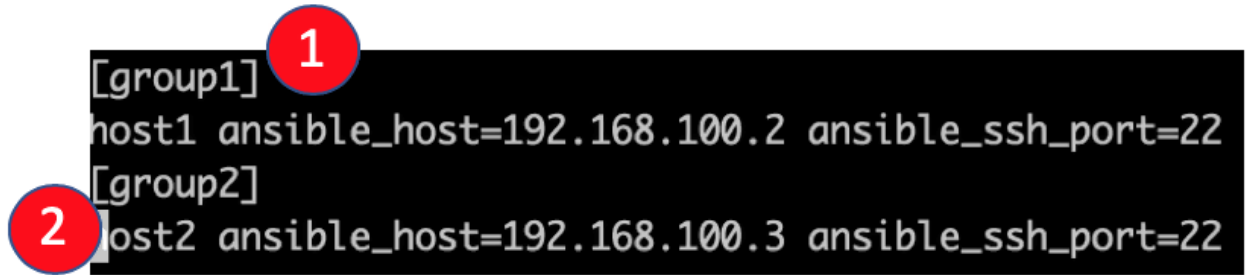
Each ansible playbook works with an inventory file. The inventory file contains a list of servers divided into groups for better control for details like [IP Address](#) and SSH port for each host.

The inventory file you can use for this Ansible playbook example looks like below. There are two groups, named group1 and group2 each containing host1 and host2 respectively.

```

[group1]
host1 ansible_host=192.168.100.2 ansible_ssh_port=22
[group2]
host2 ansible_host=192.168.100.3 ansible_ssh_port=22

```



```
[group1]
host1 ansible_host=192.168.100.2 ansible_ssh_port=22
[group2]
host2 ansible_host=192.168.100.3 ansible_ssh_port=22
```

Explanation:

1. Group name
2. Hostname, with IP address and ssh port, in this case, the default one, 22.

Another useful Ansible playbook example containing this time two plays for two hosts is the next one. For the first group of hosts, group1, selinux will be enabled. If it is enabled, then a message will appear on the screen of the host.

For the second group of hosts, httpd package will be installed only if the `ansible_os_family` is RedHat and `ansible_system_vendor` is HP.

`Ansible_os_family` and `ansible_system_vendor` are variables gathered with `gather_facts` option and can be used like in this conditional example.

```
---

- hosts: group1
  tasks:
    - name: Enable SELinux
      selinux:
        state: enabled
        when: ansible_os_family == 'Debian'
        register: enable_selinux

    - debug:
        msg: "Selinux Enabled. Please restart the server to apply changes."
        when: enable_selinux.changed == true

- hosts: group2
  tasks:
    - name: Install apache
      yum:
        name: httpd
```

```
state: present
when: ansible_system_vendor == 'HP' and ansible_os_family == 'RedHat'
```

```
---
- hosts: group1
  tasks:
    - name: Enable Selinux
      selinux:
        state: enabled
      1 when: ansible_os_family == 'Debian'
      2 register: enable_selinux

    - debug:
      msg: "Selinux Enabled. Please restart the server to apply changes."
      3 when: enable_selinux.changed == true

- hosts: group2
  tasks:
    - name: Install apache
      yum:
        name: httpd
        state: present
      4 when: ansible_system_vendor == 'HP' and ansible_os_family == 'RedHat'
```

Explanation:

1. Example of the when clause, In this case, when OS type is Debian. The ansible\_os\_family variable is gathered via gather\_facts functionality.
2. The task output is registered for future use, with its name enable\_selinux
3. Another example of the when clause. In this case, a message will be displayed for the host user if the SELinux was indeed enabled before.
4. Another example of the when clause consisting of two rules

Besides tasks, there are also some particular tasks called handlers. Handlers must have a unique name throughout the playbook. These work in the same way as a regular task but a handler can be notified via a notifier.

If a handler is not notified during the run of the playbook, it will not run. However, if more than one task notifies a handler, this will run only once after all the tasks are finished.

In the example shown below, you can see how a specific task has a notify section which calls upon another task. If the output of the first task is changed, then a handler task will be called.

The best example is to have a configuration file changed and afterward restart that specific service.

```
---  
  
- hosts: group2  
  tasks:  
    - name: sshd config file modify port  
      lineinfile:  
        path: /etc/ssh/sshd_config  
        regexp: 'Port 28675'  
        line: '#Port 22'  
      notify:  
        - restart sshd  
  handlers  
    - name: restart sshd  
      service: sshd  
        name: sshd  
        state: restarted
```

In this case, if the first task, "sshd config file modify port" is changed, meaning that if the port is not 28675 in the first place, then it will be modified and the task will notify the handler with the same name to run, and it will restart the sshd service.



```

- hosts: group2
  tasks:
    - name: sshd config file modify port
      lineinfile:
        path: /etc/ssh/sshd_config
        regexp: 'Port 28675'
        line: '#Port 22'
      1 notify:
        - restart sshd
  handlers
    2 - name: restart sshd
      service: sshd
      name: sshd
      state: restarted

```

Explanation:

1. Example of a notifier
2. Example of a handler

## Ansible Roles

When dealing with extensive playbooks, it is easier to split the tasks into roles. This also helps in reusing the roles in the future. Roles are a collection of tasks, which can be moved from one playbook to another, can be run independently but only through a playbook file.

Roles are stored in separate directories and have a particular directory structure.

```
[root@ansible-server test2]# tree
```

```
.
|-- role1
|   |-- defaults
|   |   `-- main.yml
|   |-- handlers
|   |   `-- main.yml
|   |-- meta
|   |   `-- main.yml
|   |-- README.md
|   |-- tasks
|   |   `-- main.yml
|   |-- tests
|   |   |-- inventory
|   |   `-- test.yml
|   `-- vars
|       `-- main.yml
```

7 directories, 8 files

The yml file in the defaults directory contains a list of default variables that are to be used along with the playbook. The handlers directory is used to store handlers. The meta-directory is supposed to have information about the author and role dependencies. In the tasks directory, there is the main yml file for the role.

The tests directory contains a sample yml playbook file and a sample inventory file and is mostly used for testing purposes before creating the actual role.

The vars directory contains the yml file in which all the variables used by the role will be defined. The directory templates and the directory files should contain files and templates that will be used by the tasks in the role.

To create the directory tree for a role, you should use the following command with the last parameter, the role name:

```
[root@ansible-server test2]# ansible-galaxy init role1
```

Ansible also works well with templates. As a language for templating, it uses Jinja2.

In the next example, you will find out how a basic jinja2 template looks like and use it in a role.

At the run time, depending on, let's say in which datacenter your server is located, you can select from more than one nameservers, each corresponding to a datacenter, using the variable "resolver\_ip\_addresses."

```
{% for resolver in resolver_ip_addresses %}  
nameserver {{ resolver }}  
{% endfor %}  
  
options timeout:1  
options attempts:5  
options rotate
```

In this example, in the playbook directory, there are defined some variables, including a variable named `resolver_ip_addresses` with different values depending on the datacenter.

```
- name: Set resolver for server  
  template:  
    src: dns.j2  
    dest: /etc/resolv.conf  
    group: root  
    owner: root  
    mode: "0644"  
    tag: resolver
```

The image shows a terminal-style screenshot of an Ansible task. The task is `- name: Set resolver for server`, where `server` is highlighted in yellow. Below it is the `template:` section with several parameters. Three red circles with white numbers are used as callouts:   
1. Points to `src: dns.j2`.   
2. Points to `dest: /etc/resolv.conf`.   
3. Points to `mode: "0644"`.   
The text is color-coded: `name` and `template` are cyan, `src` and `dest` are green, `group` and `owner` are blue, and `mode` is red.

```
- name: Set resolver for server  
  template:  
    src: dns.j2  
    dest: /etc/resolv.conf  
    group: root  
    owner: root  
    mode: "0644"
```

Explanation:

1. Name of the template to be used. Template is located in `templates` dir in the role path
2. Destination path of the filename to be replaced with the template, on the client-side.
3. Permissions of the destination file

The roles tasks can also have a tag field, which has a name attributed. More than one task can share the same tag. When running an ansible playbook, you can specify the tag as well, so those tasks will be executed.

## Ansible Case Study

In this section, we will analyze a Case study of an essential ansible playbook that has three roles. The purpose of this is to give a practical example of what we talked about before. Some of the examples used before in this Ansible playbook tutorial will be adapted and used in this playbook.

Below is the directory structure of the playbook. The Yaml file that will be used will be p4.yml.

```
[root@ansible-server test_ansible]# ls -lrth
total 16K
-rw-r--r--. 1 root root  0 Jul  3 10:13 testfile
-rw-r--r--. 1 root root 203 Jul  3 13:30 p1.yml
-rw-r--r--. 1 root root 125 Jul  3 15:00 hosts
-rw-r--r--. 1 root root 488 Jul  3 16:40 p2.yml
-rw-r--r--. 1 root root 188 Jul  4 17:33 p4.yml
drwxr-xr-x. 5 root root  47 Jul  4 17:35 roles
[root@ansible-server test_ansible]# cd roles
[root@ansible-server roles]# ls -lrth
total 12K
drwxr-xr-x. 9 root root 4.0K Jul  4 12:52 httpd
drwxr-xr-x. 9 root root 4.0K Jul  4 13:55 selinux
drwxr-xr-x. 9 root root 4.0K Jul  4 16:54 resolver
```

The playbook has three roles, one called resolver that sets a specific nameserver on the servers by copying a file from the server to the /etc/resolv.conf destination. Another one is called httpd, and it installs the httpd package with yum module, and the third one enables SELinux and notifies the logged user to reboot the system. Each role was created using ansible-galaxy command.

Resolver role, main.yml task:

```
---  
- name: Set resolver for server  
  copy:  
    src: resolv.conf  
    dest: /etc/resolv.conf  
    group: root  
    owner: root  
    mode: "0644"  
  tags: resolver  
# tasks file for resolver
```

Httpd role, main.yml task:

```
---  
- name: Install httpd  
  yum:  
    name: httpd  
    state: present  
    tags: httpd  
# tasks file for apache
```

Selinux role, main.yml task:

```

---
- name: Enable Selinux
  selinux:
    state: enabled
  when: ansible_os_family == 'Debian'
  register: enable_selinux
  tags: selinux

- debug:
  msg: "Selinux Enabled. Please restart the server to apply changes."
  when: enable_selinux.changed == true

# tasks file for copyfile

```

Below is the p4.yml playbook defined. It will run on all hosts if not otherwise specified in the command line, it will run as root user on port 22 (SSH), it will gather facts before running the roles, and it will run all three roles mentioned before. Each role can be run independently by specifying the tag in the ansible-playbook command line with the `-t` parameter.

```

---

- hosts: all
  user: root
  port: 22
  gather_facts: True
  roles:
    - { role: selinux, tags: selinux }
    - { role: httpd, tags: httpd }
    - { role: resolver, tags: resolver }

```

Running the p4.yml playbook on two hosts and interpreting the output. The same command can be run with the `-check` parameter for a dry-run. In case you want to use password authentication, use `-k` parameter.

```

[root@ansible-server test_ansible]# ansible-playbook -i hosts p4.yml 1
PLAY [all] *****

TASK [Gathering Facts] *****
ok: [host1]
ok: [host2]

TASK [selinux : Enable Selinux] *****
skipping: [host1] 2
skipping: [host2]

TASK [selinux : debug] *****
skipping: [host1]
skipping: [host2]

TASK [httpd : Install httpd] *****
ok: [host1] 3
ok: [host2]

TASK [resolver : Set resolver for server] *****
changed: [host1] 4
changed: [host2]

PLAY RECAP *****
host1      : ok=3    changed=1    unreachable=0    failed=0
host2      : ok=3    changed=1    unreachable=0    failed=0

```

Explanation:

1. Ansible-playbook command that runs p4.yml
2. Playbook skips SELinux role because it is already enabled.
3. Ansible found that httpd package is already installed, so it returns ok.
4. Resolver was set, and role resolver got status changed.

## Ansible Commands Cheat Sheet

Install EPEL repo on Centos/RHEL systems

```
[root@ansible-server ~]# sudo yum install epel-release
```

Install ansible package on Centos/RHEL systems

```
[root@ansible-server ~]# sudo yum install -y ansible
```

Perform an update to the packages on Debian/Ubuntu systems

```
$ sudo apt update
```

Install the software-properties-common package on Debian/Ubuntu systems

```
$ sudo apt install software-properties-common
```

Install ansible personal package archive on Debian/Ubuntu systems

```
$ sudo apt-add-repository ppa:ansible/ansible
```

Install ansible on Debian/Ubuntu systems

```
$ sudo apt update  
$ sudo apt install ansible
```

Issue a ping command on all servers defined in the inventory file named hosts

```
[root@ansible-server test_ansible]# ansible -i hosts all -m ping
```

Issue a ping command only on host2

```
[root@ansible-server test_ansible]# ansible -i hosts all -m ping --limit host2
```

Copy the file "testfile" on all hosts in the inventory file

```
[root@ansible-server test_ansible]# ansible -i hosts all -m copy -a "src=/root/test_a  
nsible/testfile dest=/tmp/testfile"
```

Install ncdp package on all hosts

```
[root@ansible-server test_ansible]# ansible -i hosts all -m yum -a 'name=ncdp state=p  
resent'
```

Remove ncdp package on all hosts

```
[root@ansible-server test_ansible]# ansible -i hosts all -m yum -a 'name=ncdp state=a  
bsent'
```

Build the directory structure for role named role1.

```
[root@ansible-server test2]# ansible-galaxy init role1
```

Dry-run p4.yml playbook

```
[root@ansible-server test_ansible]# ansible-playbook -i hosts p4.yml --check
```

Run p4.yml playbook with password authentication for all hosts

```
[root@ansible-server test_ansible]# ansible-playbook -i hosts p4.yml -k
```