

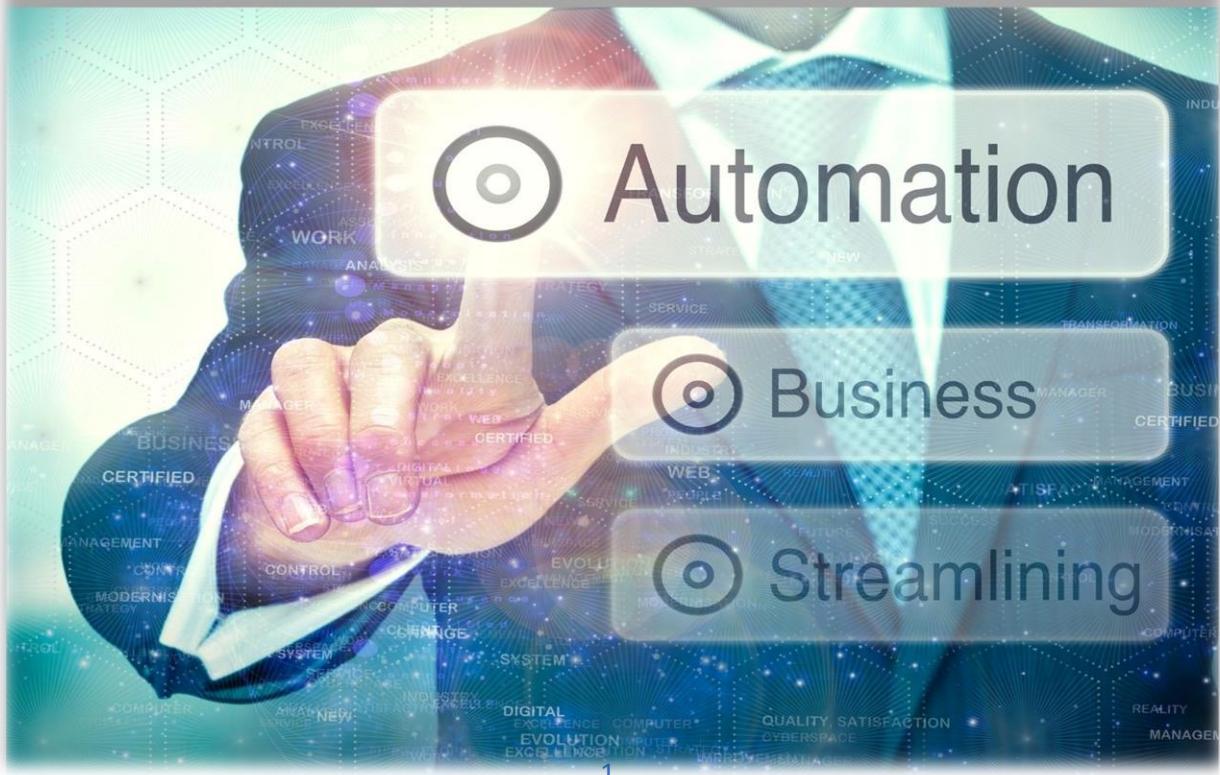
Mastering Ansible

for DevOps Engineering

A Free Guide



English Book



1

Instructor: Fariborz Fallahzadeh

Email Address: fariborz.fallahzadeh@gmail.com

Table of Contents

History of Automation

Use Cases of Ansible

Introduction to YAML Format

Introduction to API Capability

Introduction to Ansible Connection

Introduction to Ansible Architecture

How to Set Up Ansible and Infrastructure

How to Install Ansible on the Control Machine

Introduction to Creating an Inventory

How to Group Targets in the Inventory

How to Use Variables in the Inventory File

Introduction to Ad Hoc Command

Introduction to Playbook and Module

How to Create a Playbook

How to Find and Use Modules

History of Automation

Before DevOps became popular and adopted by companies, automation was already in use. This automation was performed on Windows operating systems through Batch Scripting and on Linux operating systems through Bash Scripting.

After OS automation, there was a need to automate databases, virtualization, and cloud environments as well. This required a different kind of automation that was not limited to operating systems. At this point, programming languages like Python, Ruby, Perl, and others entered the market. Among these languages, Python gained more popularity due to its vast library support and is now widely used in the field of automation.

In addition to these programming languages, Microsoft introduced PowerShell, which is used for automating Windows and other network-related tasks such as virtualization and more.

Then, Puppet was introduced as a Configuration Management Tool. Puppet is not actually a tool for automation, but rather for managing configurations. For example, when you have a large number of servers or a cluster of servers, you can use Puppet to centrally configure them. In other words, an agent is installed on each server that continuously communicates with Puppet, and Puppet executes its instructions in the form of configurations through its agents on the servers. However, when DevOps was introduced, Puppet started being used as a tool for automation as well.

In addition to Puppet, other tools like Salt Stack were introduced, which is a simple tool for executing commands on remote machines. Combining Puppet and Salt provides us with many capabilities. We can run the Puppet agent through Salt Stack, requiring an initial setup via Salt Stack followed by configuration through Puppet.

After tools like Puppet and Salt Stack, Chef was introduced. Puppet is generally a configuration tool and not a programming language.

Puppet is built using the Ruby language, and Salt Stack is built using Python, but there is no need to write Python or Ruby code, as they each have their own specific language.

Puppet uses a Domain Specific Language (DSL).

Chef was introduced with greater manageability and includes many templates and features. It has a graphical user interface and a good reporting system. In addition to these capabilities, Chef can operate on various parts of the network, such as servers and clients.

Then, Ansible was introduced as a Configuration Management Tool and Scripting Language, created by Michael DeHaan.

Ansible is written in Python and is one of the simplest automation languages. It was acquired by Red Hat, and now there are many features available in Ansible.

Ansible was introduced with the goal of simplification and can manage a large infrastructure with simple code. Initially, Ansible focused on the Linux operating system, and later it was extended to support Windows automation, cloud automation, networking automation, and database automation.

Then, another tool called Terraform was introduced by HashiCorp, which is used for cloud automation.

Use Cases of Ansible

Ansible is used in the following areas:

- Any System Automation
- Change Management
- Provisioning
- Orchestration

Ansible is a simple and clean tool compared to other automation tools and does not require any agents like Chef, Puppet, or Salt Stack. Ansible uses SSH for managing Linux operating systems and winRM or API modules for Windows operating systems. Therefore, based on the type of target, it uses a specific communication module.

Ansible does not use any database. It uses YAML or text format, and its output is in JSON format.

Setting up Ansible is simple, as it comes in the form of a Python library, and you can install it using the following command.

```
pip install ansible
```

You can create Ansible code using a Playbook. A Playbook can execute a series of Python scripts on the target and then provide you with the output.

Introduction to YAML Format

You don't need to write code in a programming language for automation; instead, you can use YAML. Ansible scripts or Ansible playbooks are written in the form of a YAML file, which is easy to read and write.

The YAML format has less structure and complexity compared to other programming language formats, which is why Ansible is very easy to read and write.

Introduction to API Capability

Ansible has many modules for performing specific tasks. These modules are API-based, meaning that if you want to create an AWS EC2 instance using Ansible, it can use an API-based tool to communicate with it.

You can use a RESTful URL that can be called by Ansible.

You can also execute Shell commands or PowerShell commands through Ansible.

Introduction to Ansible Connection

Ansible can connect to targets using various methods. A target can be Windows, Linux, a cloud account, a database, or any network-connected device.

Ansible uses SSH to connect to any Linux system, so SSH must be running on the Linux operating systems.

To connect to Windows, Ansible uses winRM, which is a service available on Windows. You need to enable remote connection on Windows. In Windows PowerShell, you can enable remote connection so that you can connect to Windows through winRM.

To connect to clouds like AWS, Azure, Google, or any other type of cloud, you can use APIs.

Ansible acts as a control machine. In other words, Ansible is not a server because no service runs on it. However, tools like Puppet, Salt, or Chef are considered servers because each of them runs as a service on a server.

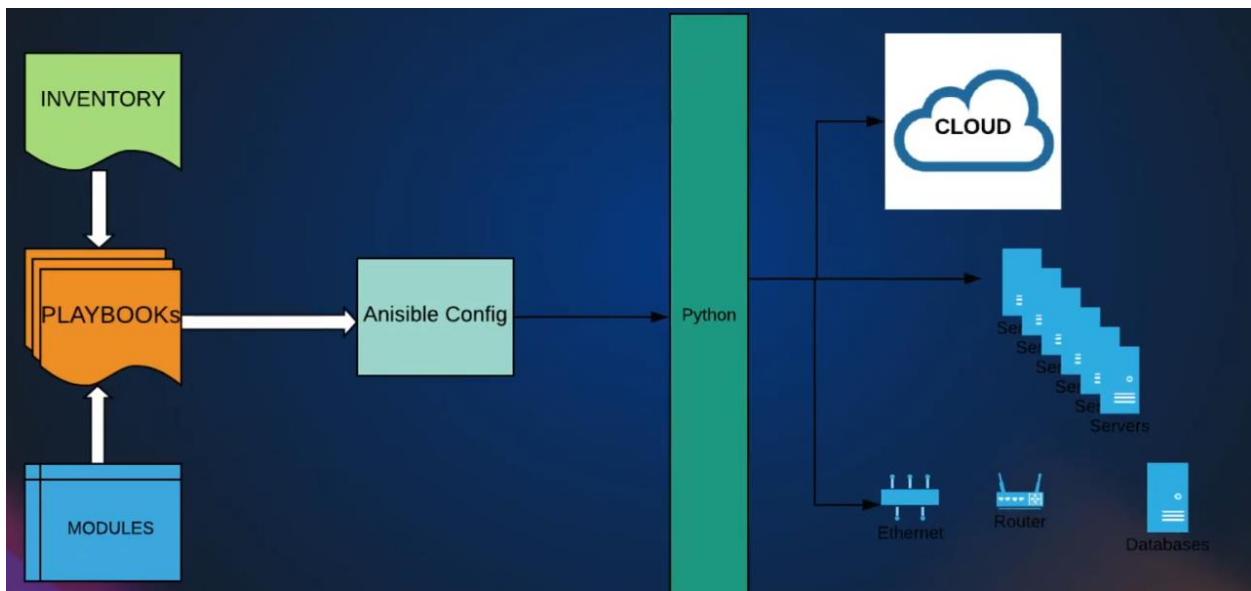
Ansible uses Python modules and scripts to connect to the target machine.

For connections made through SSH or winRM, Ansible sends Python scripts to the target, executes them, and returns the output.

For API tasks, the commands are executed on the control machine.

Introduction to Ansible Architecture

In this architecture, you need to create an inventory file. An inventory file contains information about the target machines, such as IP address, username, and password.



Ansible has many built-in modules, each used for specific tasks, such as installing a package, restarting a service, or taking a snapshot of an AWS EBS volume.

You can write a playbook. A playbook contains a task that uses module and inventory information to execute the task.

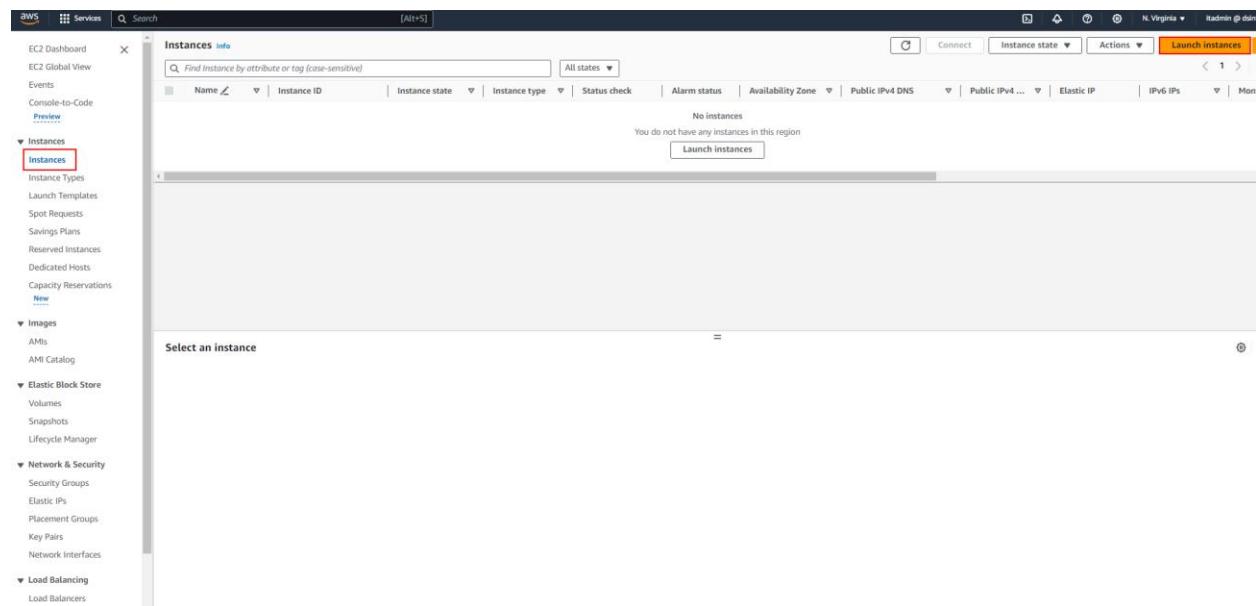
You can use Ansible configuration to create a Python script. When you execute it, Ansible runs a Python package on your destination.

How to Set Up Ansible and Infrastructure

In this section, we are going to set up an infrastructure for Ansible. We will use AWS Cloud infrastructure and create an Ubuntu EC2 instance to act as the control machine, on which Ansible will be installed. Additionally, we will create three EC2 instances with CentOS as the operating system—two of them will be used as web servers and the other one for the database. The setup of these three EC2 instances will be done using Ansible.

How to Create an Ansible Control EC2 Instance

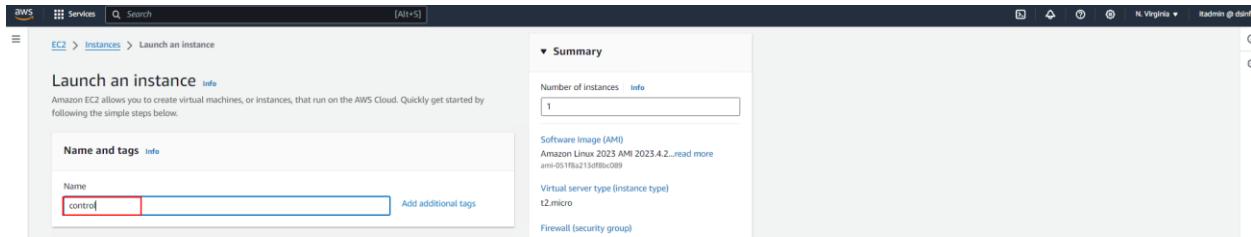
First, to create this instance, go to the Instances section and click on the **Launch Instance** button.



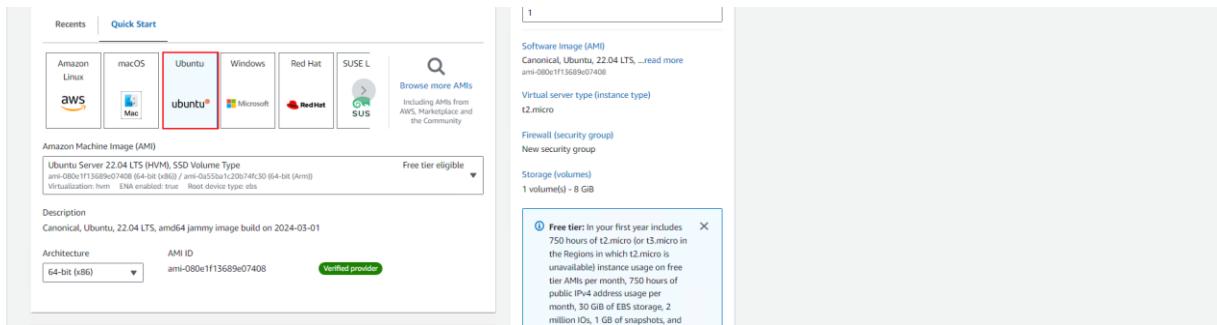
Instructor: Fariborz Fallahzadeh

Email Address: fariborz.fallahzadeh@gmail.com

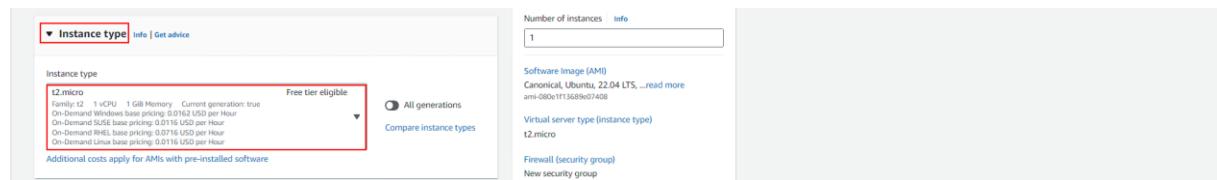
Specify a name for the instance.



Set the AMI type to Ubuntu.



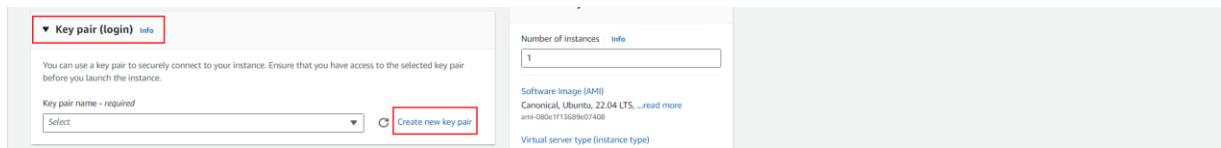
In the Instance Type section, set it to **t2.micro**.



Instructor: Fariborz Fallahzadeh

Email Address: fariborz.fallahzadeh@gmail.com

In the Key Pair section, click on the **Create New Key Pair** link.



At this stage, specify a name for the key pair, set the private key format to **PEM**, and then click on the **Create Key Pair** button.

The screenshot shows a 'Create key pair' dialog box. In the 'Key pair name' field, 'control-key' is entered. Under 'Key pair type', 'RSA' is selected. In the 'Private key file format', '.pem' is selected. A warning message at the bottom states: '⚠️ When prompted, store the private key in a secure and accessible location on your computer. You will need it later to connect to your instance. [Learn more](#)'.

Key pair name
control-key

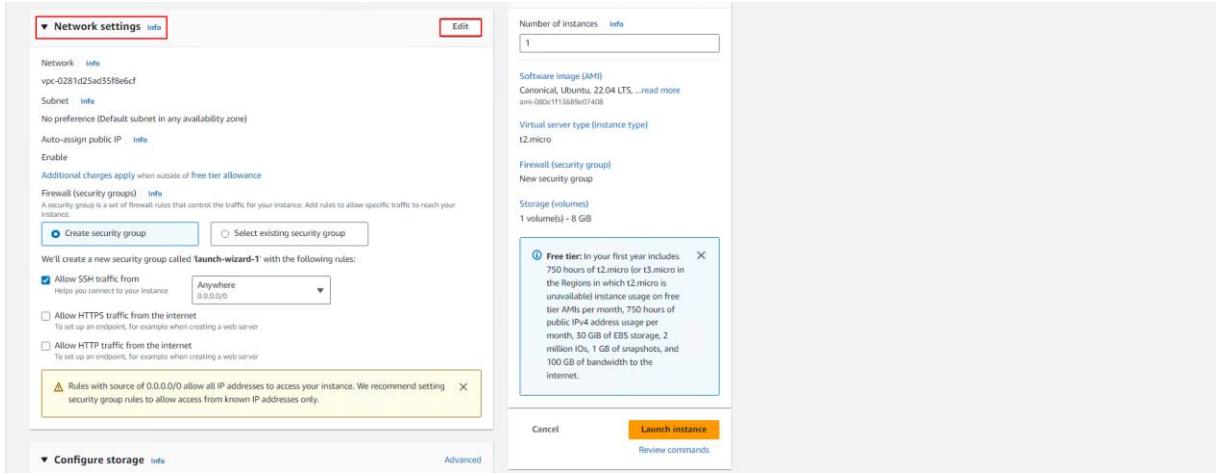
Key pair type
 RSA RSA encrypted private and public key pair
 ED25519 ED25519 encrypted private and public key pair

Private key file format
 .pem For use with OpenSSH
 .ppk For use with PuTTY

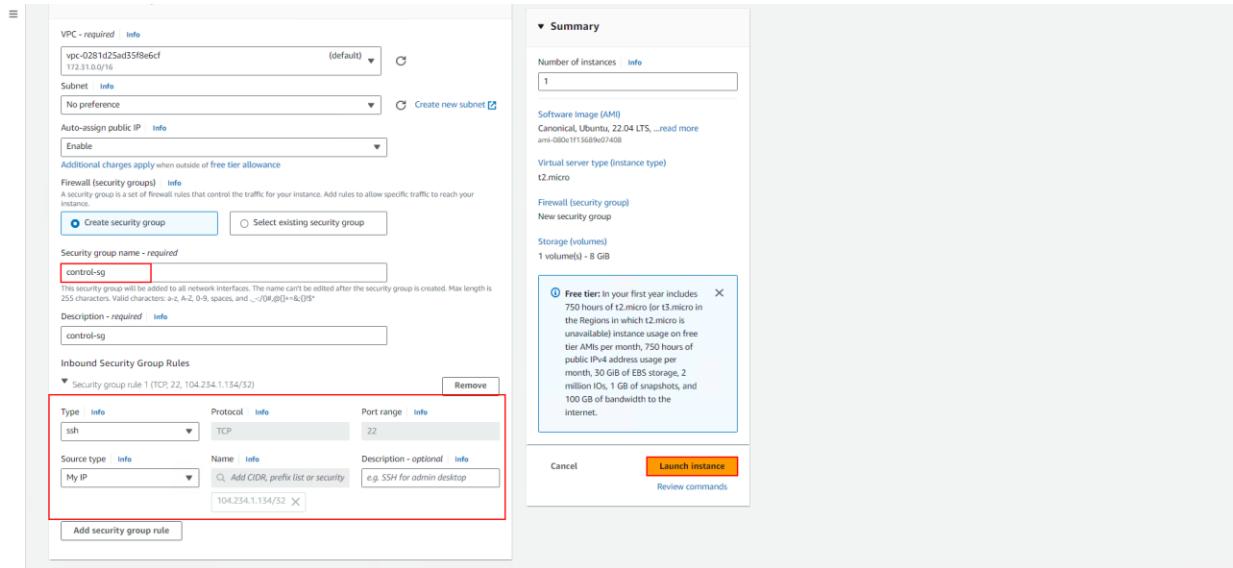
⚠️ When prompted, store the private key in a secure and accessible location on your computer. You will need it later to connect to your instance. [Learn more](#)

Cancel Create key pair

In the Network Settings section, click on the **Edit** button.



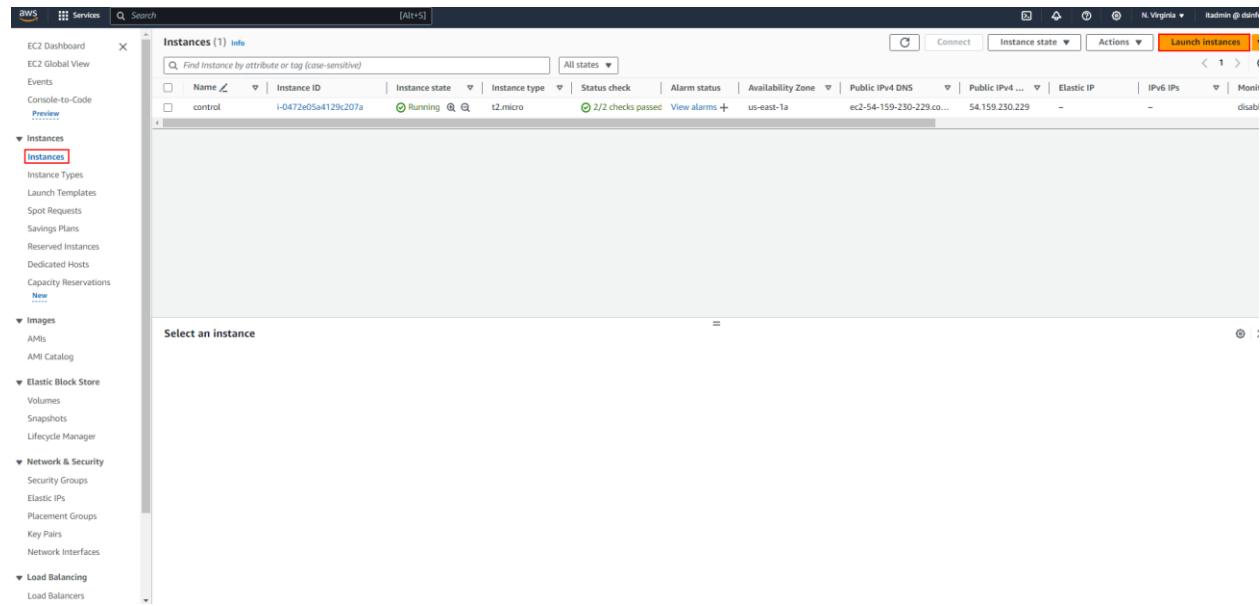
Then, specify a name for the security group and create an inbound rule to allow access to this instance via port 22. Finally, click on the **Launch Instance** button to create the instance.



How to Create 3 CentOS EC2 Instances

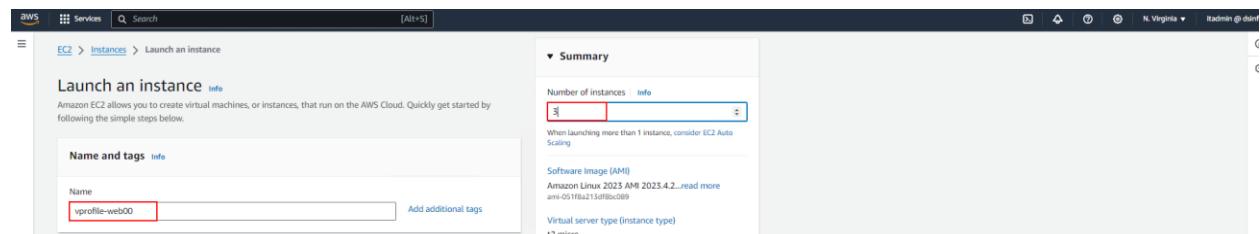
At this stage, you need to create 3 EC2 instances with CentOS as the operating system.

At this stage, go to the Instances section and click on the **Launch Instance** button.



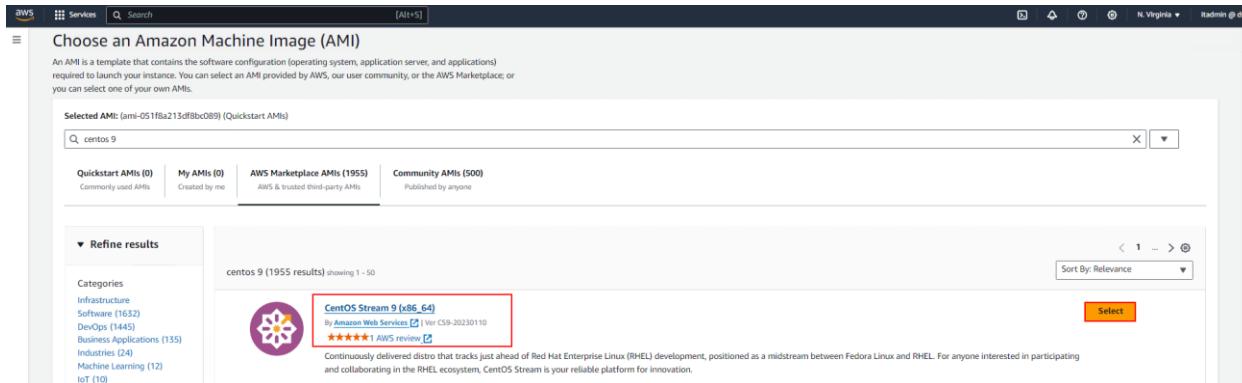
The screenshot shows the AWS EC2 Instances page. On the left, there's a navigation sidebar with sections like EC2 Dashboard, Services, and Instances (which is currently selected). The main area displays a table of instances. One instance is listed: 'control' (Instance ID: i-0472e05a4129c207a, State: Running, Type: t2.micro). The 'Launch instances' button is highlighted at the top right of the table.

At this stage, specify a name for the instance and set the **Number of Instances** to 3.



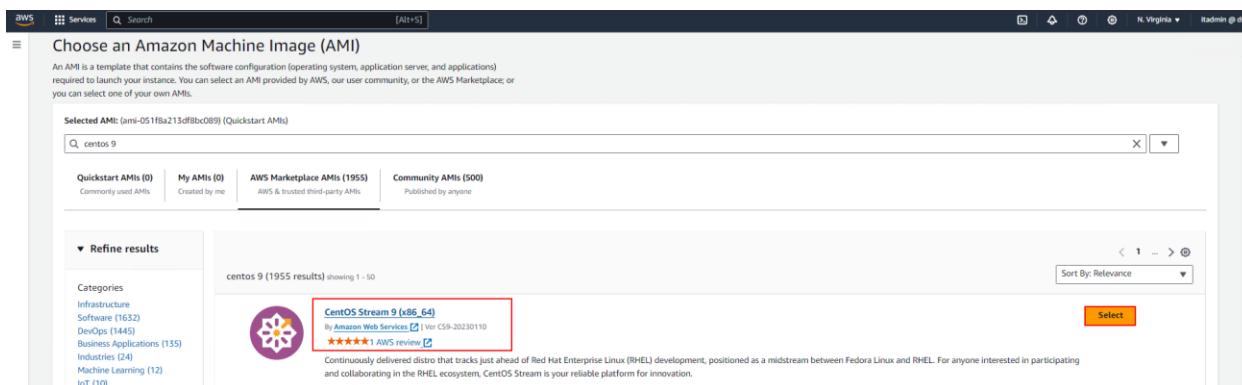
The screenshot shows the 'Launch an instance' wizard. In the 'Name and tags' section, the 'Name' field contains 'vprofile-web00'. In the 'Summary' section, the 'Number of instances' dropdown is set to '3'. Other settings shown include the 'Software Image (AMI)' as 'Amazon Linux 2023 AMI 2023.4.2...' and the 'Virtual server type (instance type)' as 't2.micro'.

At this stage, in the **AWS AMI Marketplace** section, you need to select **CentOS 9** as the operating system.



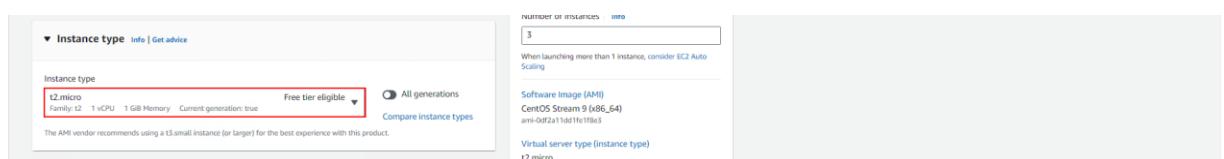
The screenshot shows the AWS AMI Marketplace search results for 'centos 9'. The search bar at the top contains 'centos 9'. Below it, there are four tabs: 'Quickstart AMIs (0)', 'My AMIs (0)', 'AWS Marketplace AMIs (1955)', and 'Community AMIs (500)'. The 'AWS Marketplace AMIs' tab is selected. On the left, a 'Refine results' sidebar shows categories like Infrastructure, Software, DevOps, Business Applications, Industries, Machine Learning, and IoT. The main results area shows 'centos 9 (1955 results)' with 'Showing 1 - 50'. A specific item, 'CentOS Stream 9 (x86_64)' by Amazon Web Services, is highlighted with a red box. It has a 5-star rating and a 'Select' button, which is also highlighted with a red box.

Then, at this stage, click on the **Subscribe Now** button.



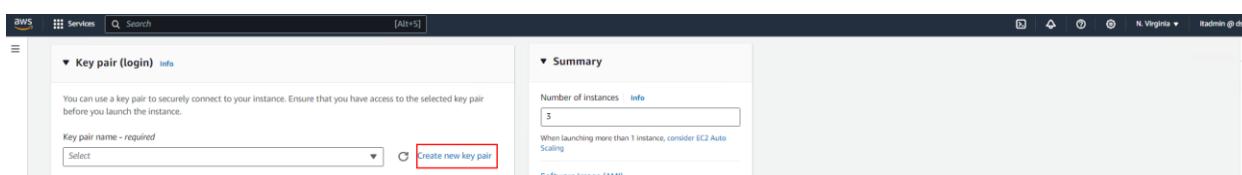
This screenshot is identical to the one above, showing the AWS AMI Marketplace search results for 'centos 9'. The 'CentOS Stream 9 (x86_64)' item is highlighted with a red box, and the 'Select' button is also highlighted with a red box.

At this stage, set the Instance Type to **t2.micro**.



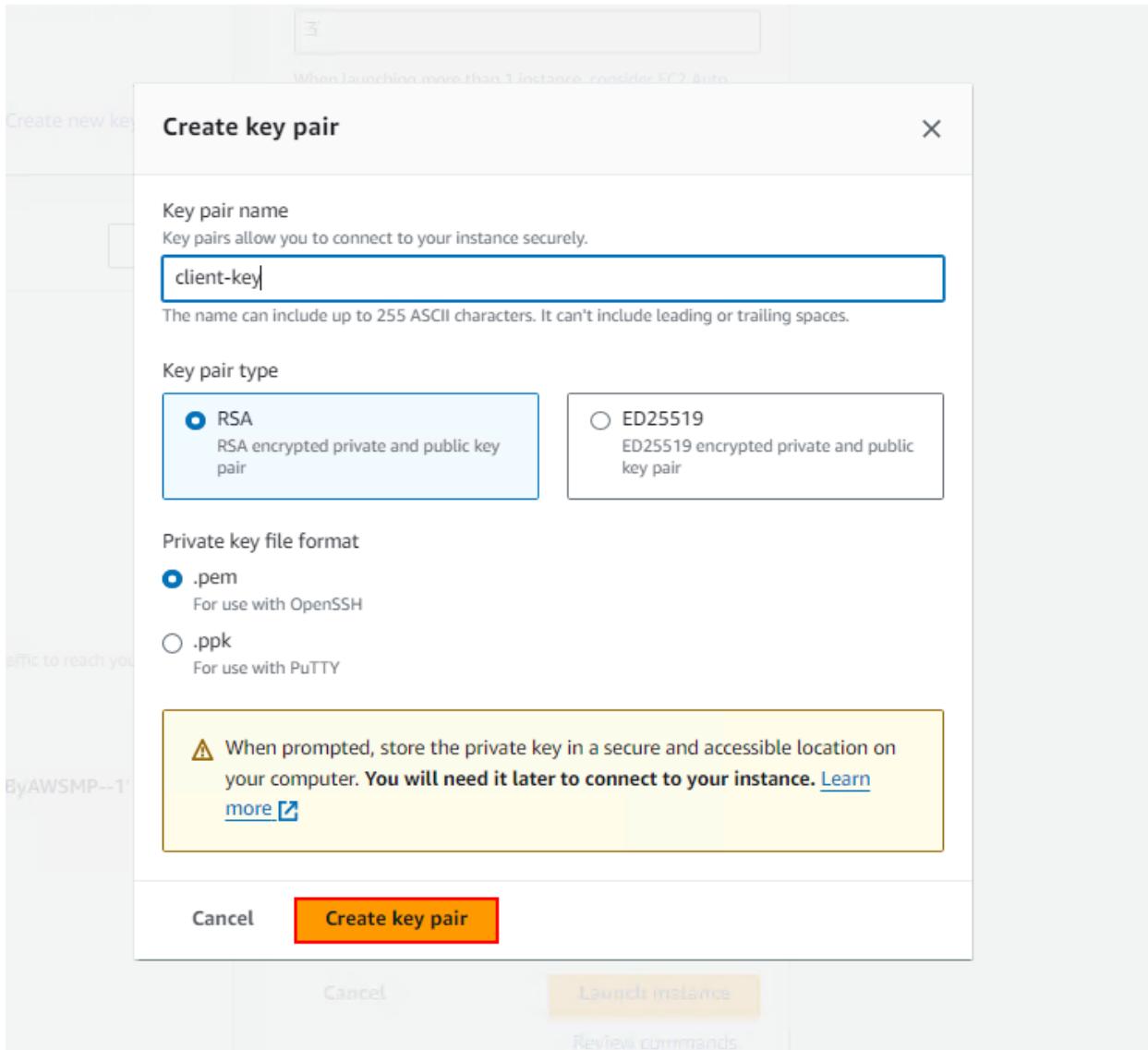
The screenshot shows the AWS instance configuration page. The 'Instance type' dropdown is set to 't2.micro'. The 'Number of instances' input field is set to '3'. The 'Software Image (AMI)' dropdown is set to 'CentOS Stream 9 (x86_64) ami-0f2a11d1ff1ff83'. The 'Virtual server type (instance type)' dropdown is set to 't2.micro'.

At this stage, in the **Key Pair** section, click on the **Create New Key Pair** button.

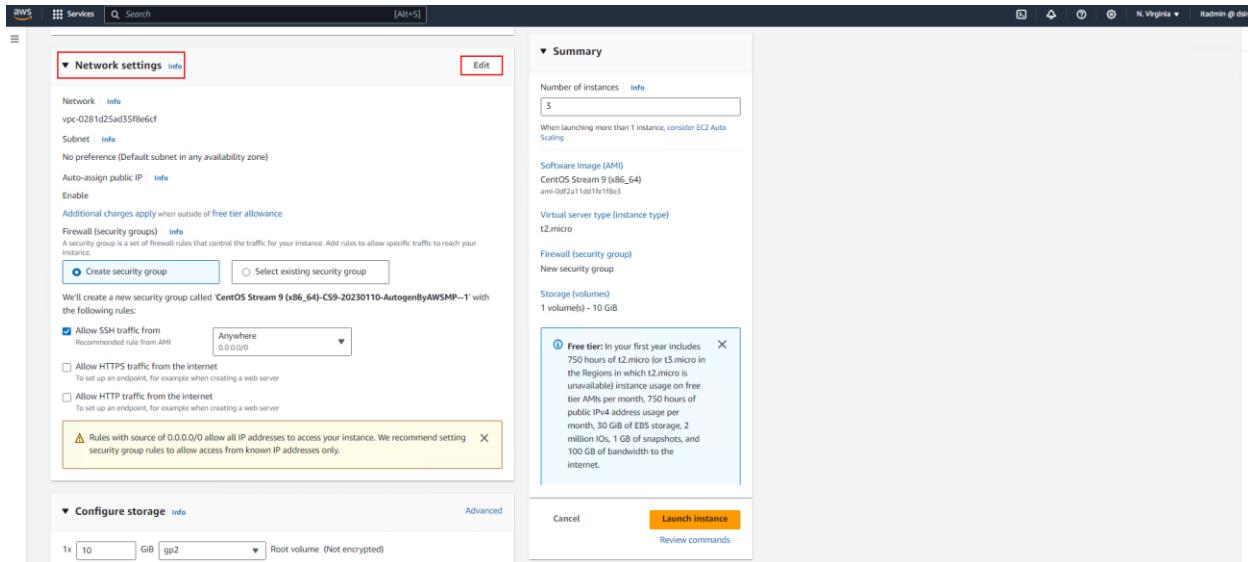


The screenshot shows the AWS instance configuration page. The 'Key pair (login)' dropdown is set to 'Select'. The 'Create new key pair' button is highlighted with a red box.

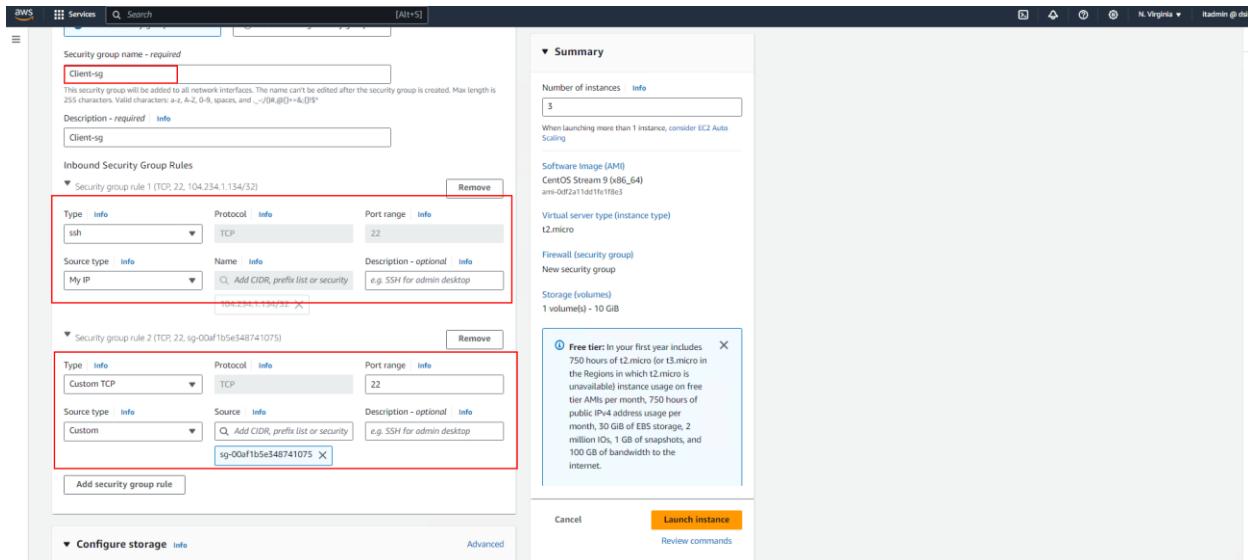
At this stage, specify a name for the key pair and then click on the **Create Key Pair** button.



In the **Network Settings** section, click on the **Edit** button.



At this stage, specify a name for the security group, then create two inbound rules—one for accessing the instances and another for allowing Ansible to access these instances. Finally, click on the **Launch Instance** button to create the instances.



At this stage, edit the name of the instance.

The screenshot shows the AWS EC2 Instances page. On the left, there's a navigation sidebar with various EC2-related options like EC2 Dashboard, Events, and Instances. The Instances section is expanded, showing sub-options like Instance Types, Launch Templates, and Capacity Reservations. The main area displays a table of four EC2 instances:

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv6 DNS	Elastic IP	IPv6 IPs
control	i-0472e05a4129c207a	Running	t2.micro	2/2 checks passed	View alarms	us-east-1a	ec2-54-159-230-229.co...	54.159.230.229	-	-
vprofile-web01	i-084f53ab4663965e	Running	t2.micro	Initializing	View alarms	us-east-1a	ec2-54-152-39-150.co...	54.152.39.150	-	-
vprofile-web02	i-0a8ca4228973445f7	Running	t2.micro	Initializing	View alarms	us-east-1a	ec2-52-55-157-66.com...	52.55.157.66	-	-
vprofile-db01	i-0e7852a29b8d01b60	Running	t2.micro	Initializing	View alarms	us-east-1a	ec2-54-172-10-134.co...	54.172.10.134	-	-

How to Install Ansible on the Control Machine

To install Ansible, you first need to SSH into the corresponding EC2 instance.

The screenshot shows a terminal window titled "Administrator@WIN-EN7125AN7JV MINGW64 ~/Downloads". The user is attempting to SSH into the EC2 instance "control" using the command:

```
$ ssh -i control-key.pem ubuntu@54.159.230.229
```

The terminal displays a warning about the authenticity of the host key fingerprint:

```
The authenticity of host '54.159.230.229 (54.159.230.229)' can't be established.  
ED25519 key fingerprint is SHA256:WnoVCXBHAO4HCGFz1g/27/npxUMph1G2mwDEUH7CVRY.  
This key is not known by any other names.  
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes  
Warning: Permanently added '54.159.230.229' (ED25519) to the list of known hosts
```

After confirming, the user is prompted for a password:

```
.
```

The terminal then shows the Ubuntu 22.04.4 LTS welcome message:

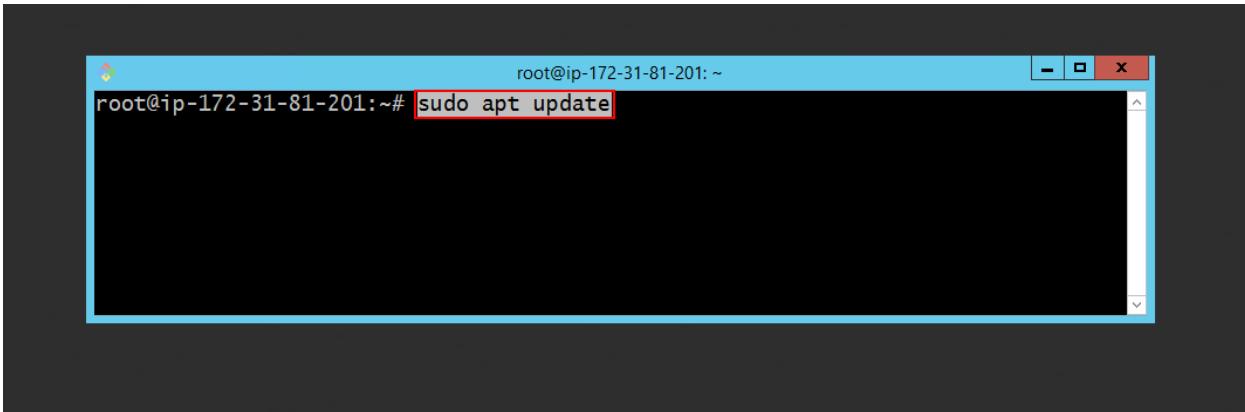
```
Welcome to Ubuntu 22.04.4 LTS (GNU/Linux 6.5.0-1014-aws x86_64)
```

System information as of Tue Apr 16 10:24:13 UTC 2024

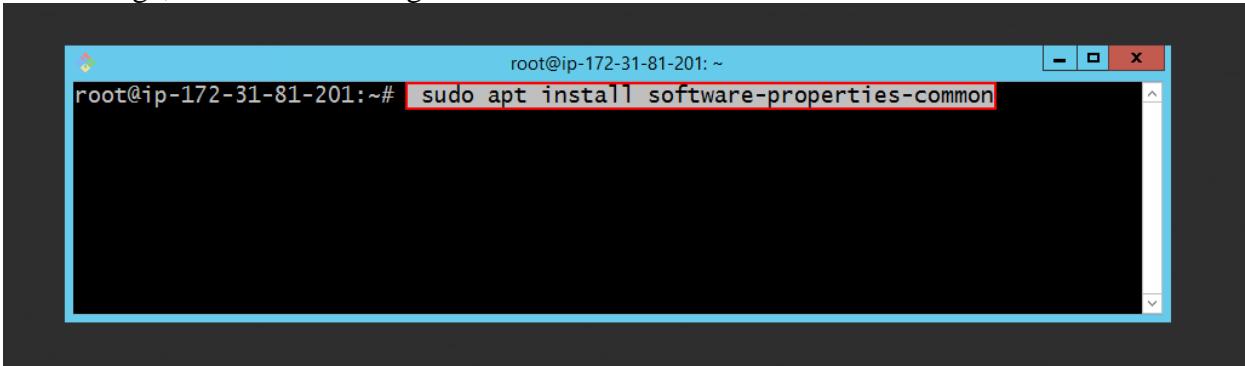
To install Ansible on the Ubuntu operating system, we use the steps provided on the following website.

https://docs.ansible.com/ansible/latest/installation_guide/installation_distros.html#installing-ansible-on-ubuntu

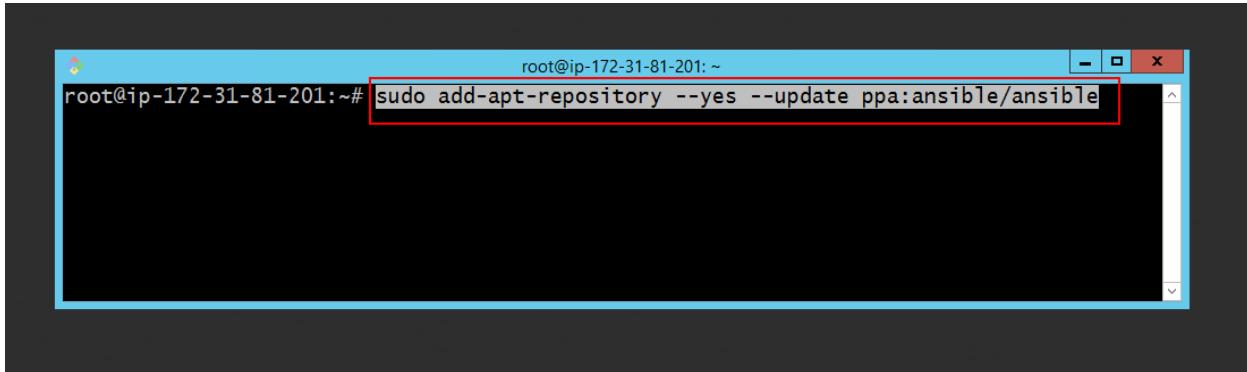
In the first step, update the repository once.



At this stage, enter the following command in the Linux terminal:

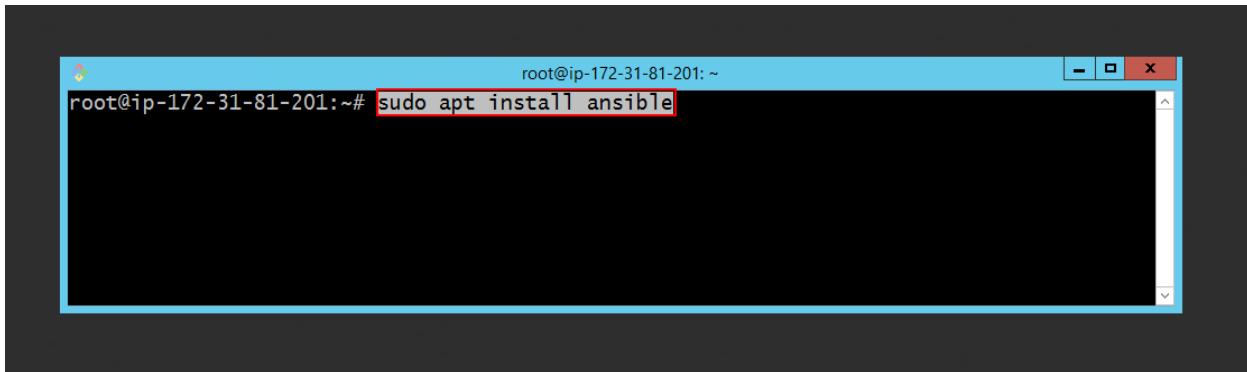


Then, you need to add the Ansible repository to the Ubuntu operating system repository using the following command.



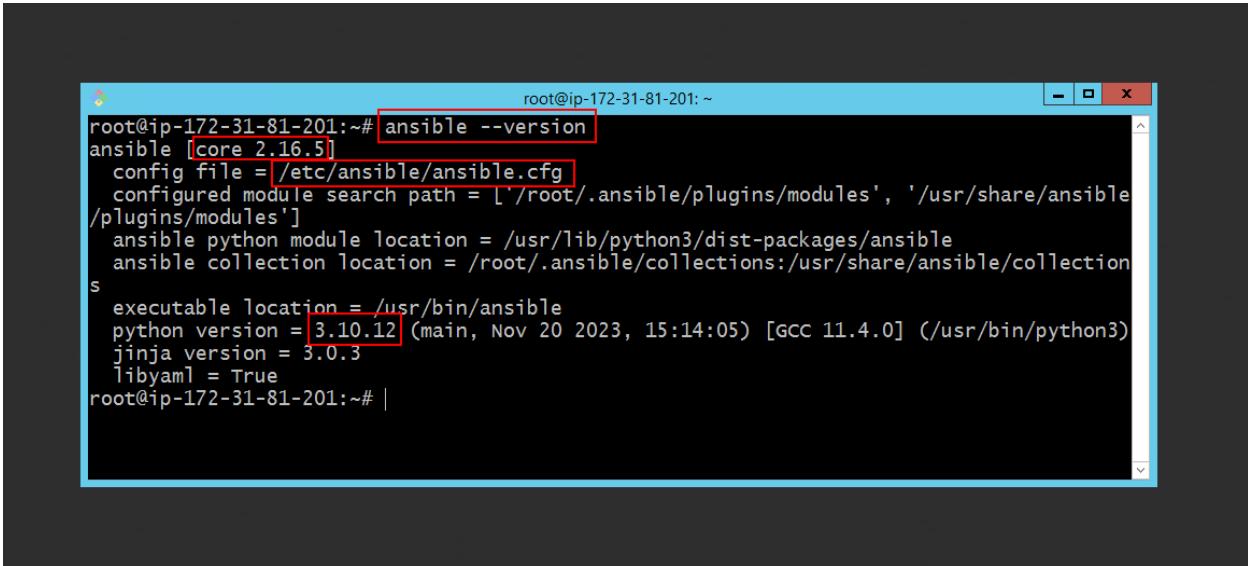
A screenshot of a terminal window titled "root@ip-172-31-81-201:~". The window contains a single line of text: "sudo add-apt-repository --yes --update ppa:ansible/ansible". A red rectangular box highlights this command line.

Finally, install Ansible using the following command.



A screenshot of a terminal window titled "root@ip-172-31-81-201:~". The window contains a single line of text: "sudo apt install ansible". A red rectangular box highlights this command line.

At this stage, check the Ansible version using the following command.



The screenshot shows a terminal window with a blue header bar containing the text "root@ip-172-31-81-201: ~". Below the header, the command "ansible --version" is run, and its output is displayed. The output includes the following information:

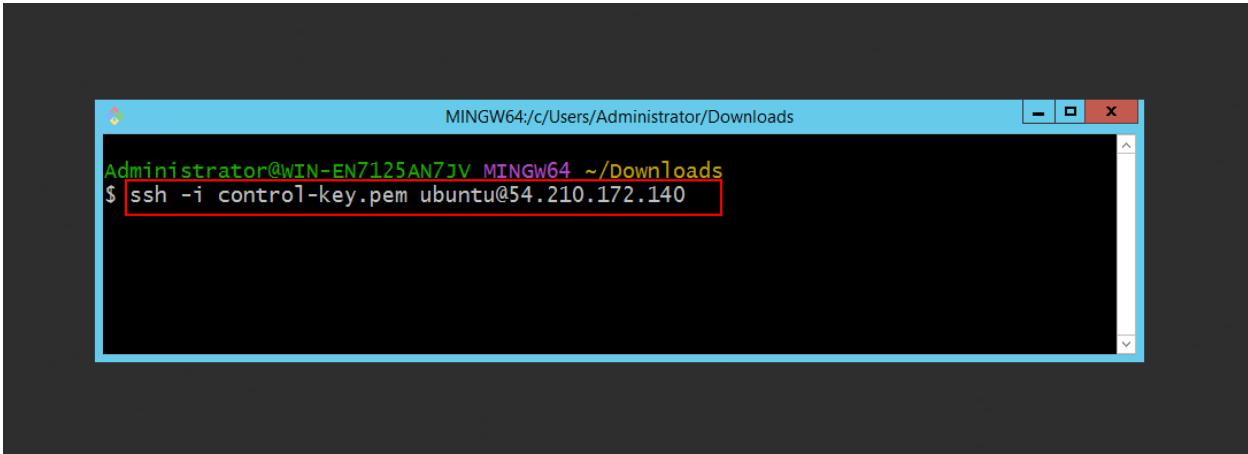
```
root@ip-172-31-81-201:~# ansible --version
ansible [core 2.16.5]
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/root/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  ansible collection location = /root/.ansible/collections:/usr/share/ansible/collection
  executable location = /usr/bin/ansible
  python version = 3.10.12 (main, Nov 20 2023, 15:14:05) [GCC 11.4.0] (/usr/bin/python3)
  jinja version = 3.0.3
  libyaml = True
root@ip-172-31-81-201:~# |
```

Introduction to Creating an Inventory

In Ansible, an inventory file is used to connect to targets. An inventory includes necessary information such as IP addresses, usernames, and passwords for connecting to the target machines.

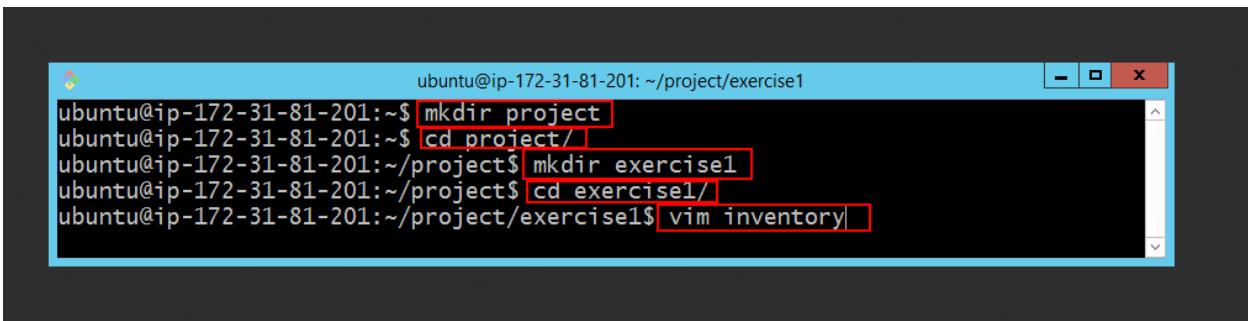
An inventory file can be written in two formats: INI or YAML.

To create the inventory file, SSH into your Ansible EC2 instance.



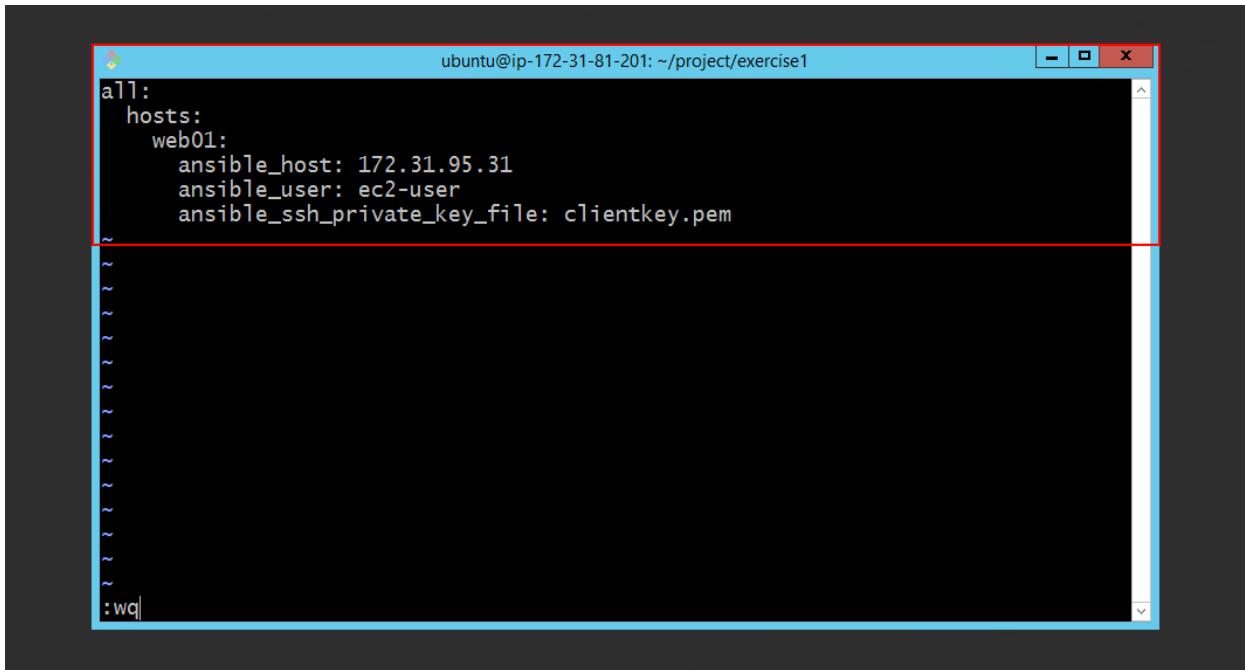
```
Administrator@WIN-EN7125AN7JV MINGW64 ~/Downloads
$ ssh -i control-key.pem ubuntu@54.210.172.140
```

At this stage, create a folder for the project, then create another folder named **Exercise1** for your practice. Inside this folder, use the VIM editor to create a file named **inventory**. You can choose any name for this file.



```
ubuntu@ip-172-31-81-201:~$ mkdir project
ubuntu@ip-172-31-81-201:~$ cd project/
ubuntu@ip-172-31-81-201:~/project$ mkdir exercise1
ubuntu@ip-172-31-81-201:~/project$ cd exercise1/
ubuntu@ip-172-31-81-201:~/project/exercise1$ vim inventory
```

Inside the inventory file, enter the information related to the target.



A screenshot of a terminal window titled "ubuntu@ip-172-31-81-201: ~/project/exercise1". The window contains the following text:

```
all:
  hosts:
    web01:
      ansible_host: 172.31.95.31
      ansible_user: ec2-user
      ansible_ssh_private_key_file: clientkey.pem
```

The text is highlighted with a red rectangle. The terminal window has a blue border and a black background. The bottom right corner of the terminal window shows a small vertical scroll bar. The bottom of the terminal window shows a command prompt with the text ":wq|".

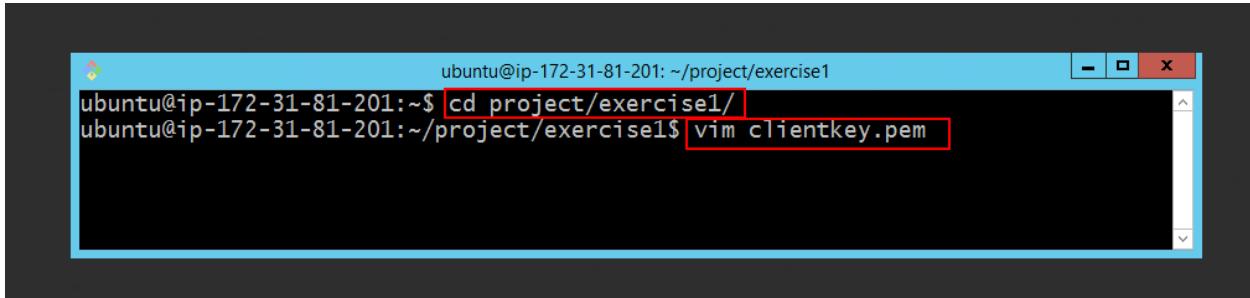
Exit from your control machine, and then use the following command to copy the contents of the private key file related to your target.

The screenshot shows a terminal window titled "MINGW64:/c/Users/Administrator/Downloads". The command \$ cat client-key.pem is run, displaying a long string of RSA private key data. A context menu is open over the terminal output, with the "Copy" option highlighted. The terminal prompt at the bottom is Administrator@WIN-EN7125AN7JV MINGW64 ~/Downloads \$ |

Then, SSH back into your control machine.

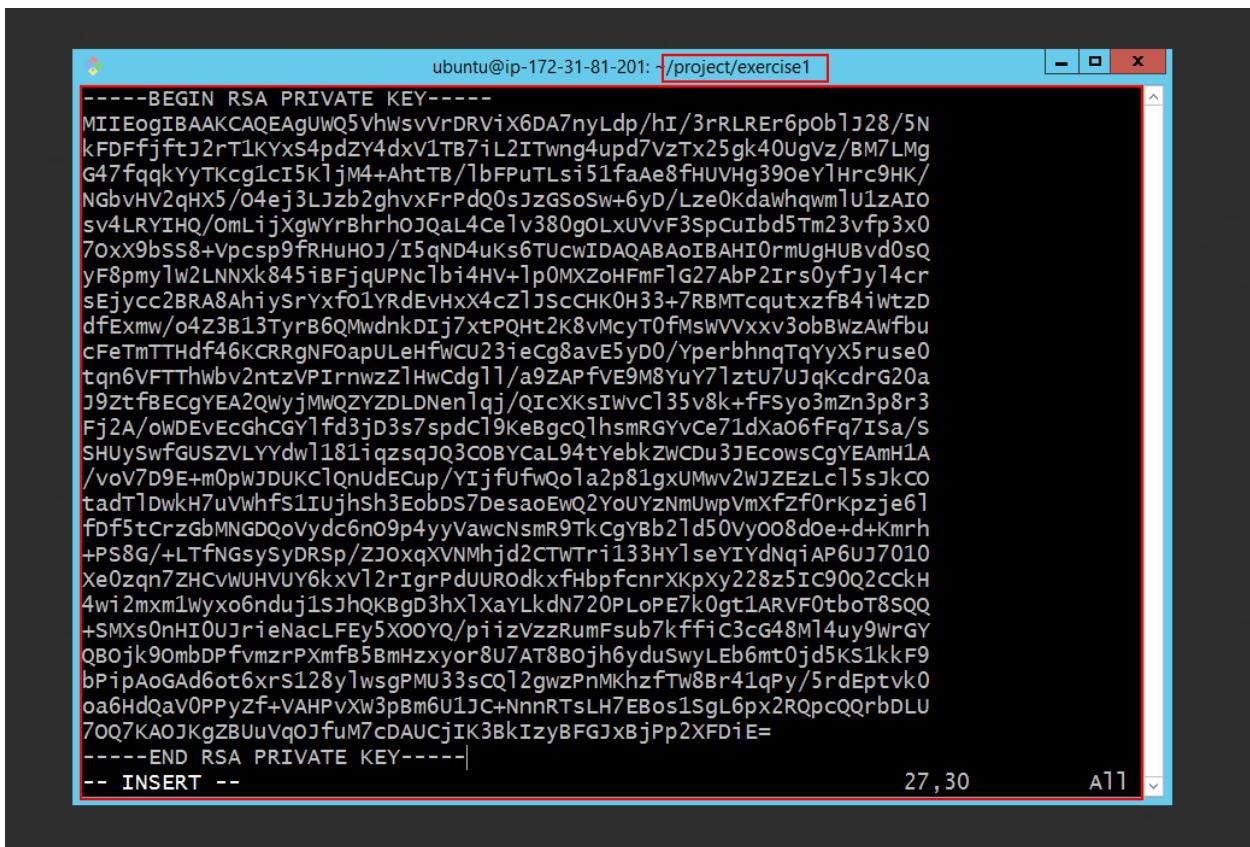
The screenshot shows a terminal window titled "MINGW64:/c/Users/Administrator/Downloads". The command \$ ssh -i kops-key.pem ubuntu@3.89.228.30 is run. The terminal prompt at the bottom is Administrator@WIN-EN7125AN7JV MINGW64 ~/Downloads \$ |

Go to the project directory, and then use the VIM editor to create a file with exactly the same name as the private key specified in the inventory file.



```
ubuntu@ip-172-31-81-201: ~/project/exercise1
ubuntu@ip-172-31-81-201:~$ cd project/exercise1/
ubuntu@ip-172-31-81-201:~/project/exercise1$ vim clientkey.pem
```

Then, paste the contents of the target's private key file that you copied in the previous steps into this file.



```
-----BEGIN RSA PRIVATE KEY-----
MIIEogIBAAKCAQEAgUwQ5vhWsvVrDRViX6DA7nyLdp/hI/3rRLER6pOb1j28/5N
kfDFffjftJ2rT1KYxs4pdZY4dxV1TB7iL2ITwng4upd7VzTx25gk40UgVz/BM7LMg
G47fqqkYyTKcg1cI5K1jM4+AhtTB/1bFPuTLSi51faAe8fHUVHg39OeY1Hrc9HK/
NGbvHV2qHX5/04ej3LJzb2ghvxFrPdQoSjZGSoS+6yD/Lze0KdawhqwmlU1zAIO
sv4LRYIHQ/OmLi jXgwYrBhrhOJQaL4Cevl380g0LxUVVF3SpcuIbd5Tm23vfp3x0
70xx9bSS8+Vpcsp9fRHuHOJ/I5qND4uks6TUCwIDAQABoIBAH0rmUgHUBvd0sQ
yF8pmj1w2LNXXk845iBFjqUPNC1bi4Hv+1p0MXzoHFmFlG27AbP2Irs0yfJy14cr
sEjycc2BRA8AhysiRyxfo1YRdEvHxx4cz1JScCHK0H33+7RBMTcqutxzfb4iWtzD
dfExmw/o4Z3B13TyrB6QMwdnkDIj7xtPQHt2K8vMcYTofMsVVxxv3obBWzAWfbu
cFeTmTTdf46KCRgNFOapuLEhFWCU43ieCg8avE5yD0/YperbhngTqYyx5ruse0
tqn6VFTThwbv2ntzVPIrnzz1HwCdgl1/a9ZAPfVE9M8YuV7lztU7UJqKcdrg20a
j9ZtfbECgYEAE0WyjMwQZYLDNen1qj/QIcXKsIwvC135v8k+fFsyo3mzn3p8r3
Fj2A/oWDEVEcGhCGY1fd3jd3s7spdC19KeBgcQ1hsrnGYvCe71dxa06FFq7Isa/S
SHuySwfGUSZVLYYdw1181iqzsqJQ3COBYCaL94tYebkZWCDU3JEcowsCgYEAmH1A
/voV7D9E+m0pwJDUKclQnudEcup/YIjfufwQola2p81gxUMwv2WJZEzLc15sjko
tadTlDwkH7uVwhfs1IUjhsh3EobDS7DesaoEwQ2YoUYzNmUwpVmxfzf0rkpzje61
fdf5tCrzGbMNGDQoVyd6n09p4yyVawcNsmR9TkCgyBb21d50Vyo08doe+d+Kmrh
+PS8G/+LTfNGsySyDRSp/Zj0xqVNMrhjd2CTwTr133HY1seYIYdNqiAP6Uj7010
Xe0zqn7ZHCvWUHVUY6kxv12rIgrPduRodkxfHbpfcnrXKpxy228z5ic90Q2cckh
4wi2mxm1wyx06nduj1sJhQKBgD3hx1xaYLkdN720PLoPE7k0gt1ARVF0tboT8sQQ
+SMXs0nHI0UjrieNacLFEy5XOOYQ/pizVzzRumFsub7kffic3cg48M14uy9wrgY
QBojk90mbDPfvmzrPxmfB5BmHzxyor8u7AT8Bojh6yduSwyLeb6mt0jd5ks1kkF9
bPipAoGAd6ot6xrs128ylwsgPMU33sCQ12gwzPnmkhzfTw8Br41qPy/5rdEptvk0
oa6HdQaV0PPyzf+VAHPVXw3pBm6U1JC+NnnRTsLH7EBos1SgL6px2RQpcQQrbDLU
70Q7KA0JkgZBuuvq0JfuM7cDAUCjIK3BkIzyBFGJxBjPp2XFDi=
-----END RSA PRIVATE KEY-----
-- INSERT --
```

27, 30

All

Then, at this stage, use the following command to test the connection to the target.

In the following command:

-web01

is the name of the target.

-m

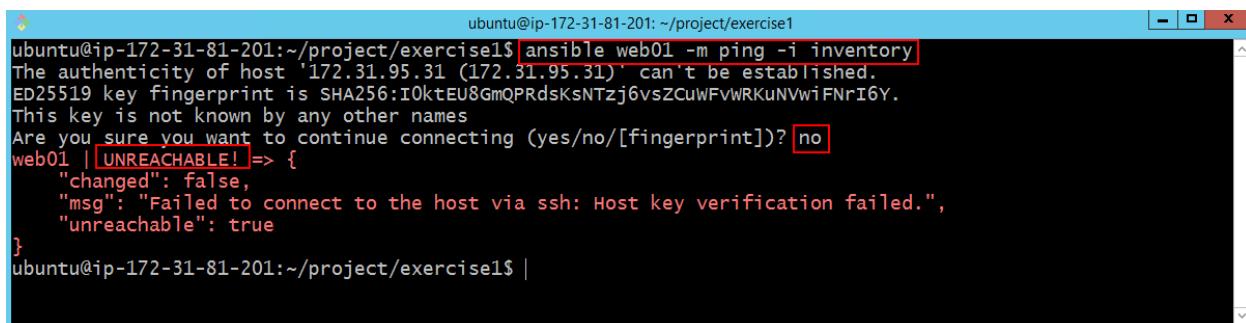
means module.

-Ping

is one of Ansible's modules used for SSH and does not function like a network ping.

-i

is used to specify the path to the inventory file.

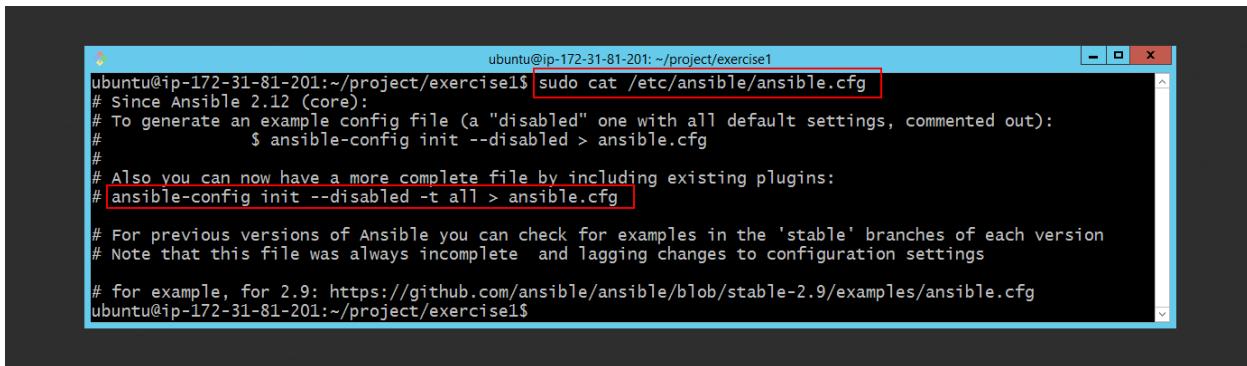


```
ubuntu@ip-172-31-81-201:~/project/exercise1$ ansible web01 -m ping -i inventory
The authenticity of host '172.31.95.31 (172.31.95.31)' can't be established.
ED25519 key fingerprint is SHA256:IOkteU8GmQPRdsKsNTzj6vsZCuWFvwRKuNVwiFNrI6Y.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? no
web01 | [UNREACHABLE!]=> {
    "changed": false,
    "msg": "Failed to connect to the host via ssh: Host key verification failed.",
    "unreachable": true
}
ubuntu@ip-172-31-81-201:~/project/exercise1$ |
```

As shown in the image above, when the above command is executed, it asks you to confirm the public key, which is not ideal because it interrupts the automation process. This default behavior should be changed so that no confirmation is requested when using the above command.

To change Ansible's default settings, you need to check the `ansible.cfg` file, which contains Ansible configurations. The path to this file is shown below, but by default, it doesn't contain any specific content.

You need to create an Ansible configuration to have more settings, which can be done using the `ansible-config` command.

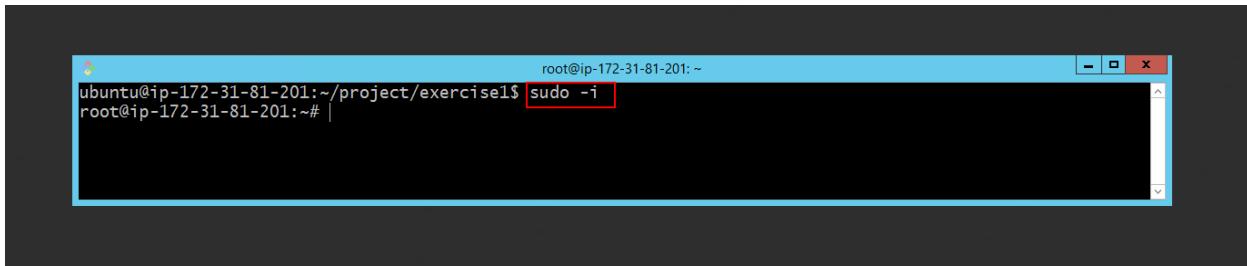


```
ubuntu@ip-172-31-81-201:~/project/exercise1$ sudo cat /etc/ansible/ansible.cfg
# Since Ansible 2.12 (core):
# To generate an example config file (a "disabled" one with all default settings, commented out):
#           $ ansible-config init --disabled > ansible.cfg
#
# Also you can now have a more complete file by including existing plugins:
# ansible-config init --disabled -t all > ansible.cfg

# For previous versions of Ansible you can check for examples in the 'stable' branches of each version
# Note that this file was always incomplete and lagging changes to configuration settings

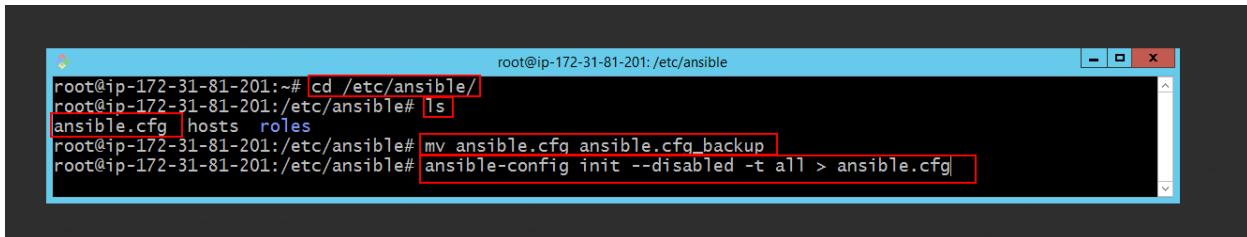
# for example, for 2.9: https://github.com/ansible/ansible/blob/stable-2.9/examples/ansible.cfg
ubuntu@ip-172-31-81-201:~/project/exercise1$
```

First, use the following command to enter the root user environment.



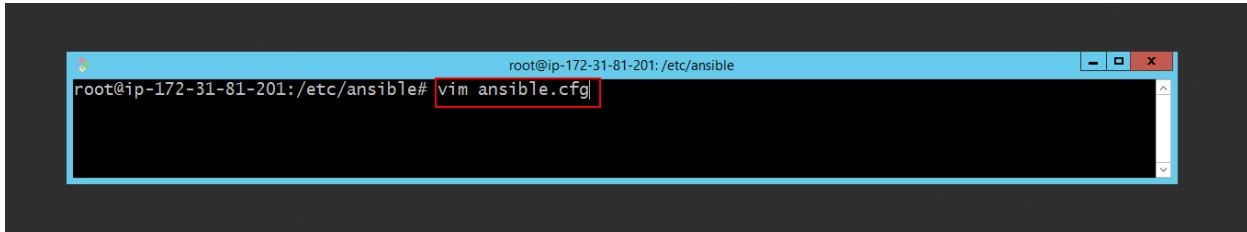
```
root@ip-172-31-81-201:~# sudo -i
root@ip-172-31-81-201:~# |
```

Then, go to the Ansible directory, and use the `mv` command to rename the original Ansible file—in other words, to create a backup of it. After that, use the `ansible-config init` command to generate a new Ansible configuration file with more settings.



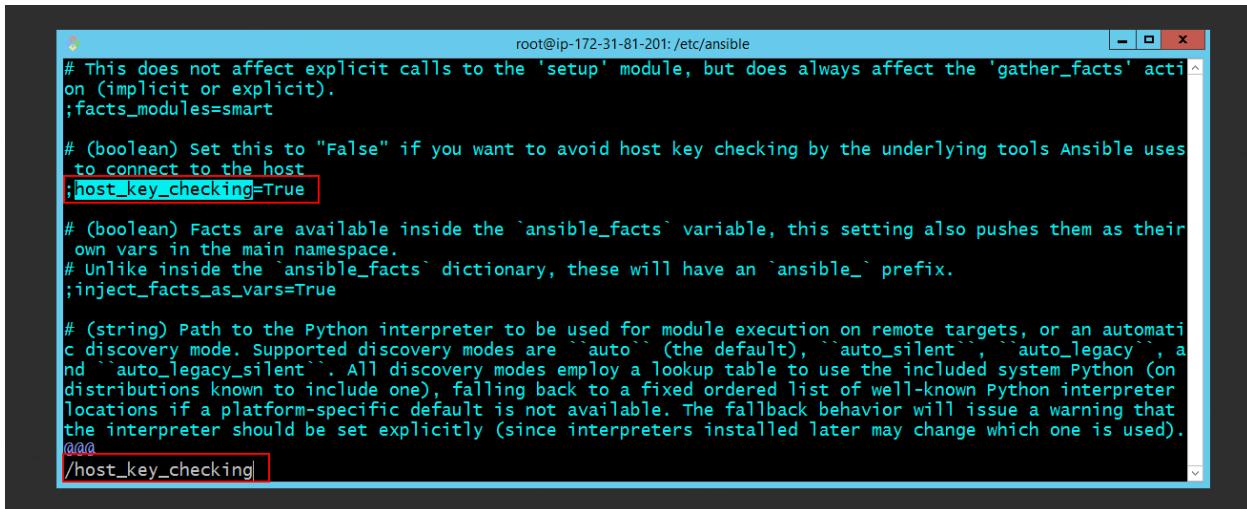
```
root@ip-172-31-81-201:~# cd /etc/ansible/
root@ip-172-31-81-201:/etc/ansible# ls
ansible.cfg hosts roles
root@ip-172-31-81-201:/etc/ansible# mv ansible.cfg ansible.cfg_backup
root@ip-172-31-81-201:/etc/ansible# ansible-config init --disabled -t all > ansible.cfg
```

At this stage, open the new `ansible.cfg` file using the VIM editor.



A screenshot of a terminal window titled "root@ip-172-31-81-201:/etc/ansible". The command "vim ansible.cfg" is entered in the terminal. The terminal has a blue header bar and a black body with white text. A red box highlights the command "vim ansible.cfg".

Inside the file, search for the variable `host_key_checking`.



A screenshot of a terminal window titled "root@ip-172-31-81-201:/etc/ansible". The file "ansible.cfg" is open in the vim editor. The configuration file contains several sections and variables. A red box highlights the line "`;host_key_checking=True`". The terminal has a blue header bar and a black body with white text. A red box highlights the line "`;host_key_checking=True`".

```
# This does not affect explicit calls to the 'setup' module, but does always affect the 'gather_facts' action (implicit or explicit).
;facts_modules=smart

# (boolean) Set this to "False" if you want to avoid host key checking by the underlying tools Ansible uses to connect to the host
;host_key_checking=True

# (boolean) Facts are available inside the `ansible_facts` variable, this setting also pushes them as their own vars in the main namespace.
# Unlike inside the `ansible_facts` dictionary, these will have an `ansible_` prefix.
;inject_facts_as_vars=True

# (string) Path to the Python interpreter to be used for module execution on remote targets, or an automatic discovery mode. Supported discovery modes are ``auto`` (the default), ``auto_silent``, ``auto_legacy``, and ``auto_legacy_silent``. All discovery modes employ a lookup table to use the included system Python (on distributions known to include one), falling back to a fixed ordered list of well-known Python interpreter locations if a platform-specific default is not available. The fallback behavior will issue a warning that the interpreter should be set explicitly (since interpreters installed later may change which one is used).
;interpreter=/usr/bin/python
```

At this stage, remove the ; at the beginning of the line to uncomment the instruction, then change its value from True to False. This way, Ansible will no longer ask for public key confirmation when connecting to the target. Finally, save the file.

```
root@ip-172-31-81-201:/etc/ansible
# This does not affect explicit calls to the 'setup' module, but does always affect the 'gather_facts' action
# (implicit or explicit).
;facts_modules=smart

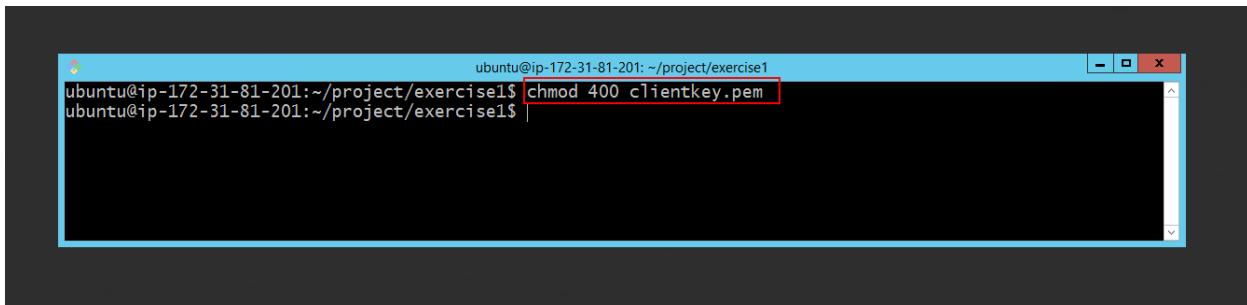
# (boolean) Set this to "False" if you want to avoid host key checking by the underlying tools Ansible uses
# to connect to the host
host_key_checking=False

# (boolean) Facts are available inside the `ansible_facts` variable, this setting also pushes them as their
# own vars in the main namespace.
# Unlike inside the `ansible_facts` dictionary, these will have an `ansible_` prefix.
;inject_facts_as_vars=True

# (string) Path to the Python interpreter to be used for module execution on remote targets, or an automatic
# discovery mode. Supported discovery modes are ``auto`` (the default), ``auto_silent``, ``auto_legacy``, and
# ``auto_legacy_silent``. All discovery modes employ a lookup table to use the included system Python (on
# distributions known to include one), falling back to a fixed ordered list of well-known Python interpreter
# locations if a platform-specific default is not available. The fallback behavior will issue a warning that
# the interpreter should be set explicitly (since interpreters installed later may change which one is used).
@@@
:wq
```

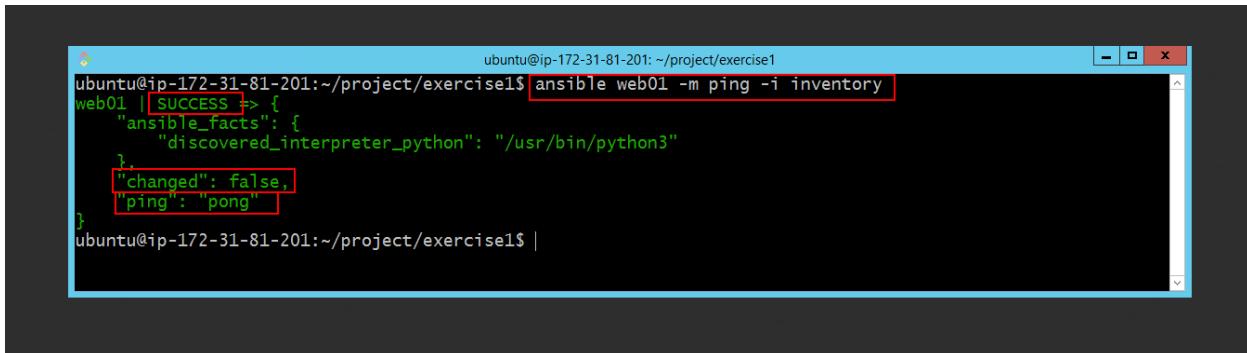
Then, exit the root user environment and use the command below again to connect to the target. However, as shown in the image below, you encounter a permission error when accessing the `clientkey.pem` file. This means Ansible cannot read the contents of this file. To fix this issue, you need to set the correct permission for the file.

Use the following command to change the permission of the `clientkey.pem` file.



A terminal window titled "ubuntu@ip-172-31-81-201: ~/project/exercise1\$". The command `chmod 400 clientkey.pem` is highlighted with a red box. The output shows the command was run successfully.

As shown in the image below, when the following command is executed, it returns **Success**, which means the connection between the control machine and the target machine was successfully established. The word **False** under the **Change** section indicates that no changes were made to the target, as only an SSH connection was made without modifying anything. In the **ping** module section, the word **pong** appears, which is the output of the ping module.

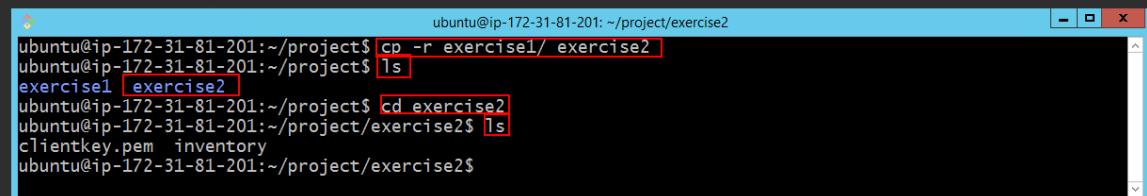


A terminal window titled "ubuntu@ip-172-31-81-201: ~/project/exercise1\$". The command `ansible web01 -m ping -i inventory` is highlighted with a red box. The output shows the command was run successfully, returning "Success" and "pong". The "changed": false part is also highlighted with a red box.

How to Group Targets in the Inventory

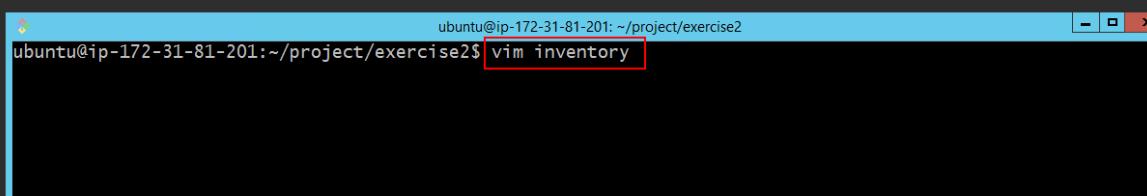
One of the things you can do inside the inventory is grouping the targets. By using grouping, you can easily apply certain configurations to one or multiple groups. For example, you can apply specific settings to a group of web servers or a group of databases, and so on.

To get familiar with grouping targets in the inventory, first make a copy of the previous project folder with a new name, and then enter the **exercise2** folder.



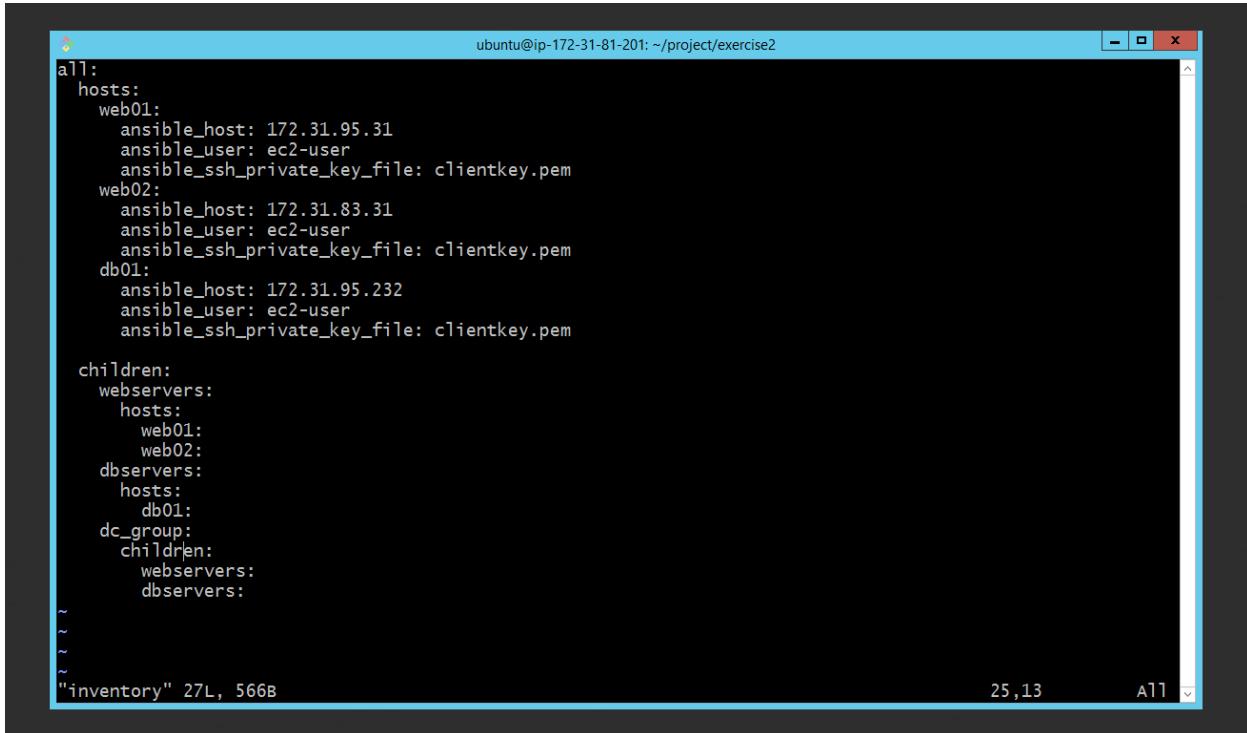
```
ubuntu@ip-172-31-81-201:~/project/exercise2$ cp -r exercise1/ exercise2
ubuntu@ip-172-31-81-201:~/project$ ls
exercise1 exercise2
ubuntu@ip-172-31-81-201:~/project$ cd exercise2
ubuntu@ip-172-31-81-201:~/project/exercise2$ ls
clientkey.pem inventory
ubuntu@ip-172-31-81-201:~/project/exercise2$
```

At this stage, open the **inventory** file using the VIM editor.



```
ubuntu@ip-172-31-81-201:~/project/exercise2$ vim inventory
```

As shown in the image below, we have created three groups in this inventory: **webservers**, **dbservers**, and **dc_group**. The **webservers** group contains the web servers, the **dbservers** group contains the database servers, and the **dc_group** group includes both the **webservers** and **dbservers** groups.



```
ubuntu@ip-172-31-81-201: ~/project/exercise2
all:
  hosts:
    web01:
      ansible_host: 172.31.95.31
      ansible_user: ec2-user
      ansible_ssh_private_key_file: clientkey.pem
    web02:
      ansible_host: 172.31.83.31
      ansible_user: ec2-user
      ansible_ssh_private_key_file: clientkey.pem
    db01:
      ansible_host: 172.31.95.232
      ansible_user: ec2-user
      ansible_ssh_private_key_file: clientkey.pem

  children:
    webservers:
      hosts:
        web01:
        web02:
    dbservers:
      hosts:
        db01:
    dc_group:
      children:
        webservers:
        dbservers:
~
~
~
~
"inventory" 27L, 566B
25,13          All
```

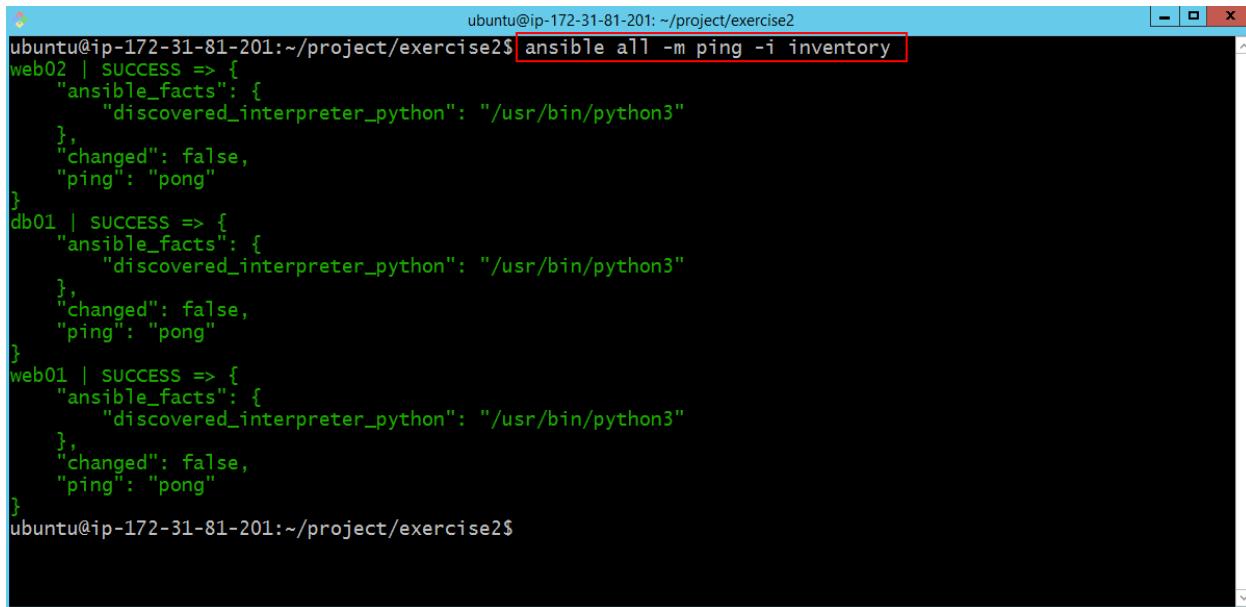
As shown in the image below, when the `ansible` command is used with the **webservers** group, the settings are applied only to the web server group. When used with the **dbservers** group, the settings are applied only to the database server group. And when the **dc_group** group is used in the `ansible` command, the settings are applied to both the **webservers** and **dbservers** groups.

The screenshot shows a terminal window on an Ubuntu system (version 17.2-31-81-201) with the following output:

```
ubuntu@ip-172-31-81-201:~/project/exercise2$ ansible webservers -m ping -i inventory
web01 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
web02 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
ubuntu@ip-172-31-81-201:~/project/exercise2$ ansible dbservers -m ping -i inventory
db01 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
ubuntu@ip-172-31-81-201:~/project/exercise2$ ansible dc_group -m ping -i inventory
db01 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
web02 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
web01 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
```

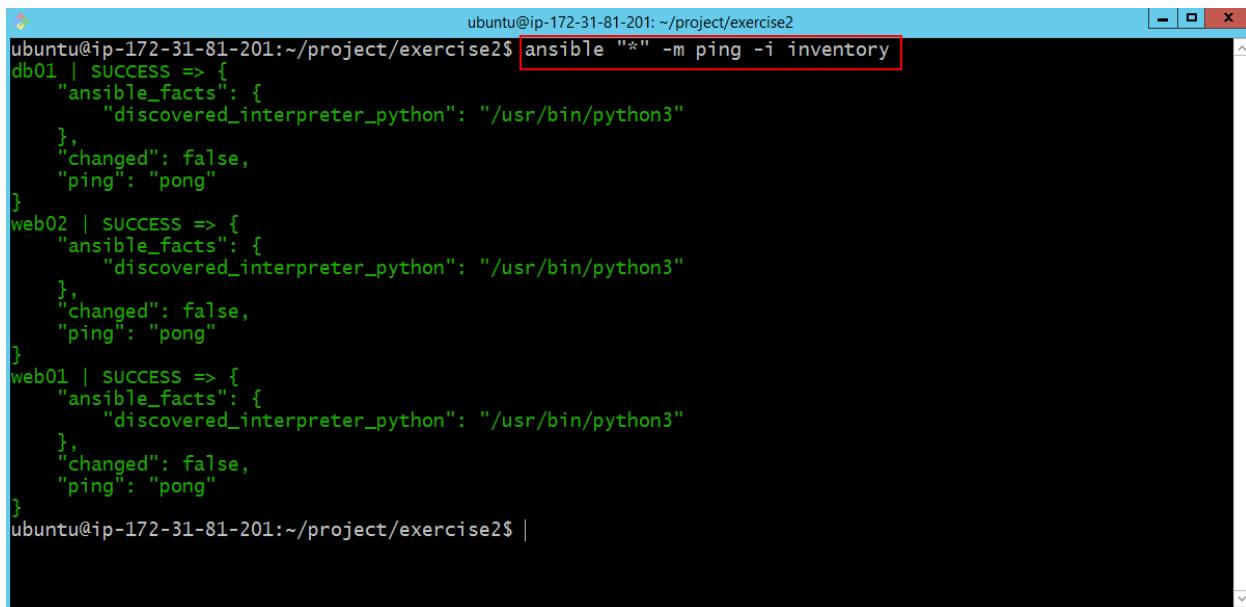
The terminal window has a light blue header bar and a black body. The command lines are in white, and the output is in green. The three `ansible` commands are highlighted with red boxes.

If you use **all** after the `ansible` command, the settings will be applied to all targets.



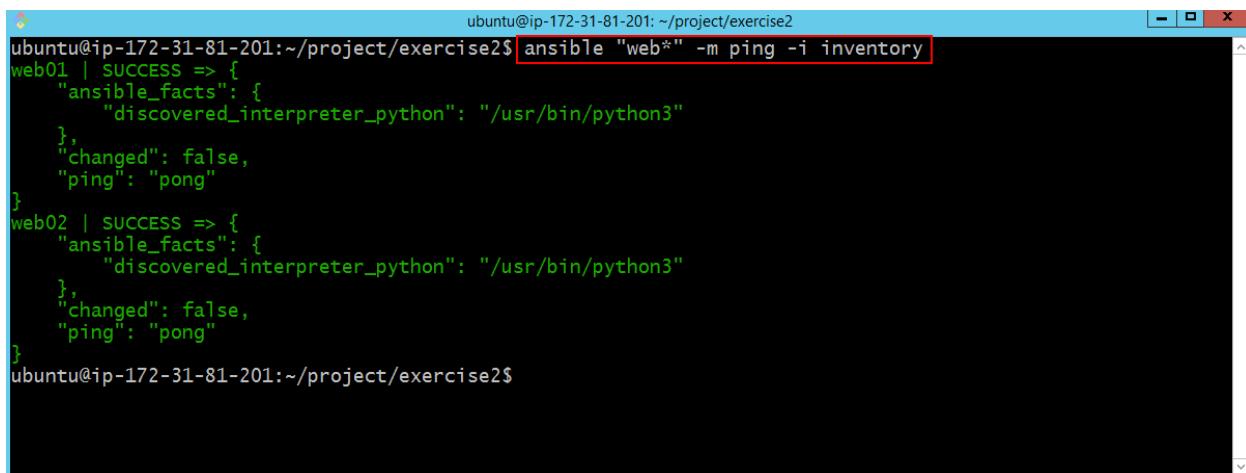
```
ubuntu@ip-172-31-81-201:~/project/exercise2$ ansible all -m ping -i inventory
web02 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
db01 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
web01 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
ubuntu@ip-172-31-81-201:~/project/exercise2$
```

As shown in the image below, you can also use ***** as a substitute for **all**, and in this case, the settings will be applied to all targets.



```
ubuntu@ip-172-31-81-201:~/project/exercise2$ ansible "*" -m ping -i inventory
db01 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
web02 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
web01 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
ubuntu@ip-172-31-81-201:~/project/exercise2$ |
```

You can also use regular expressions, like in the example below where * is used after the word web. This means the target includes all machines that start with web and are followed by any characters.

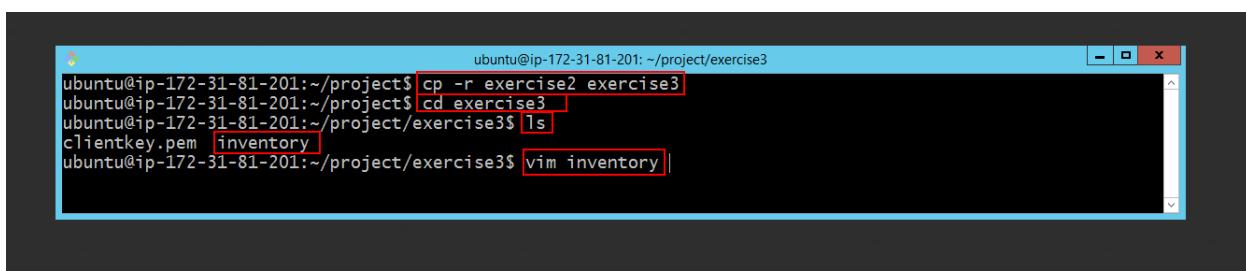


```
ubuntu@ip-172-31-81-201:~/project/exercise2$ ansible "web*" -m ping -i inventory
web01 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
web02 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
ubuntu@ip-172-31-81-201:~/project/exercise2$
```

How to Use Variables in the Inventory File

We can use variables for repetitive information. Variables defined at the host level have a higher priority than those defined at the group level. In this section, instead of defining the username and SSH private key under each host, we define them at the group level. Variables also help prevent repetition of information.

At this stage, make a copy of the **exercise2** folder with the name **exercise3**, then enter the new folder and edit it using the VIM editor.



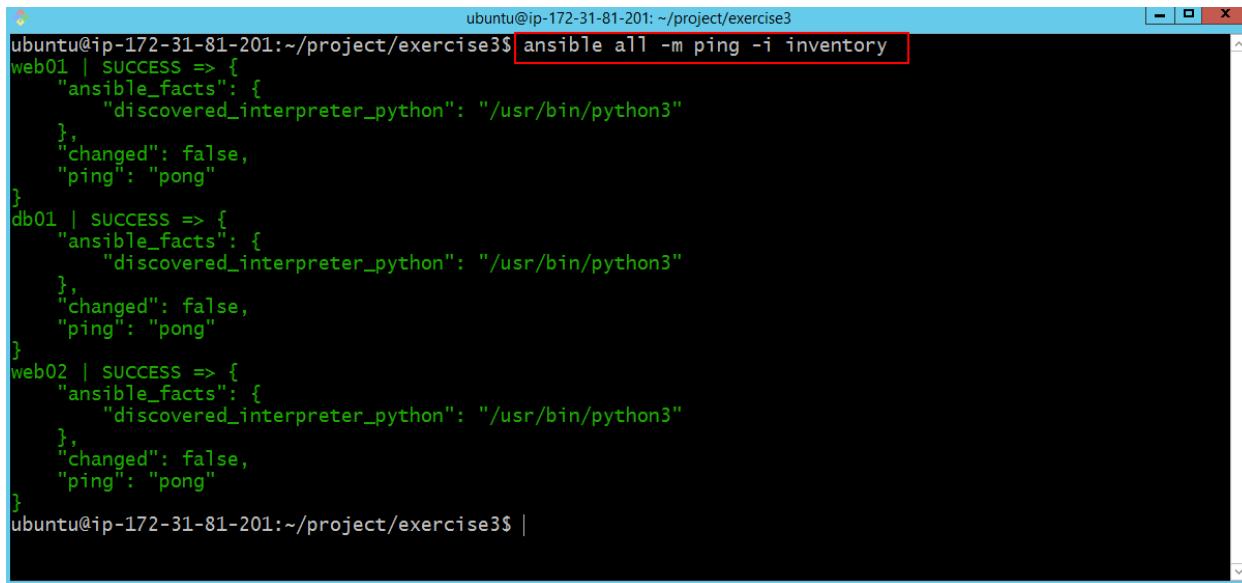
```
ubuntu@ip-172-31-81-201:~/project$ cp -r exercise2 exercise3
ubuntu@ip-172-31-81-201:~/project$ cd exercise3
ubuntu@ip-172-31-81-201:~/project/exercise3$ ls
clientkey.pem inventory
ubuntu@ip-172-31-81-201:~/project/exercise3$ vim inventory|
```

In the inventory file, remove the variables `ansible_user` and `ansible_ssh_private_key` from the host section, and define them under the `dc_group` in the `vars` section. Since this group includes

all the hosts, the values of these variables will apply to all of them, avoiding repetition and making the file structure simpler.

```
ubuntu@ip-172-31-81-201: ~/project/exercise3
all:
  hosts:
    web01:
      ansible_host: 172.31.95.31
    web02:
      ansible_host: 172.31.83.31
    db01:
      ansible_host: 172.31.95.232
  children:
    webservers:
      hosts:
        web01:
        web02:
    dbservers:
      hosts:
        db01:
    dc_group:
      children:
        webservers:
        dbservers:
  vars:
    ansible_user: ec2-user
    ansible_ssh_private_key_file: clientkey.pem
~
~
~
~
~
~
:wq|
```

At this stage, test the connection to the targets using the following command. As shown in the image below, all connections to the targets have been successfully established.



```
ubuntu@ip-172-31-81-201:~/project/exercise3$ ansible all -m ping -i inventory
web01 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
db01 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
web02 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
ubuntu@ip-172-31-81-201:~/project/exercise3$ |
```

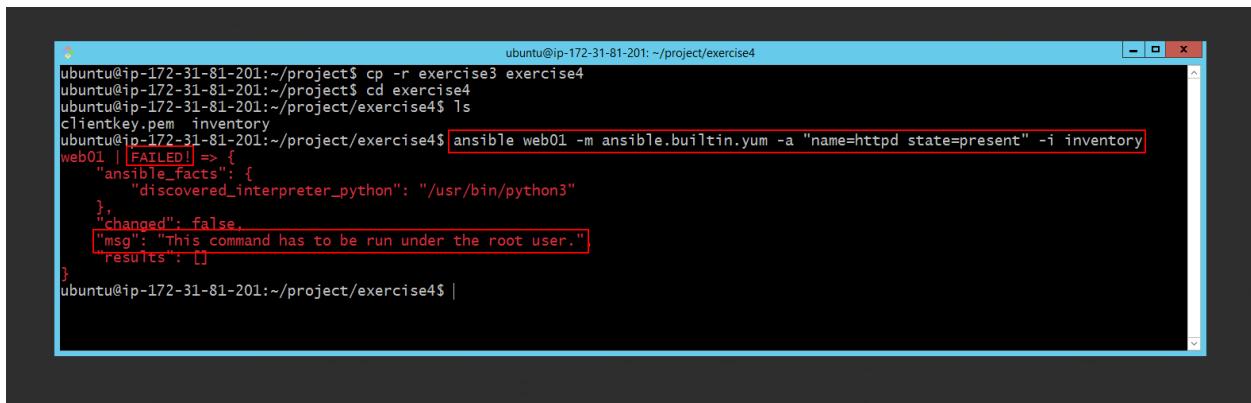
Introduction to Ad Hoc Command

In the previous section, we executed the ping module on the targets, and in this section, we intend to use Ad Hoc commands.

Ansible scripts are known as Ansible Playbooks. Before writing a Playbook, you can execute and test the commands intended for the Playbook in the form of Ad Hoc commands.

In other words, executing commands in the command line using the `ansible` command is known as an Ad Hoc command.

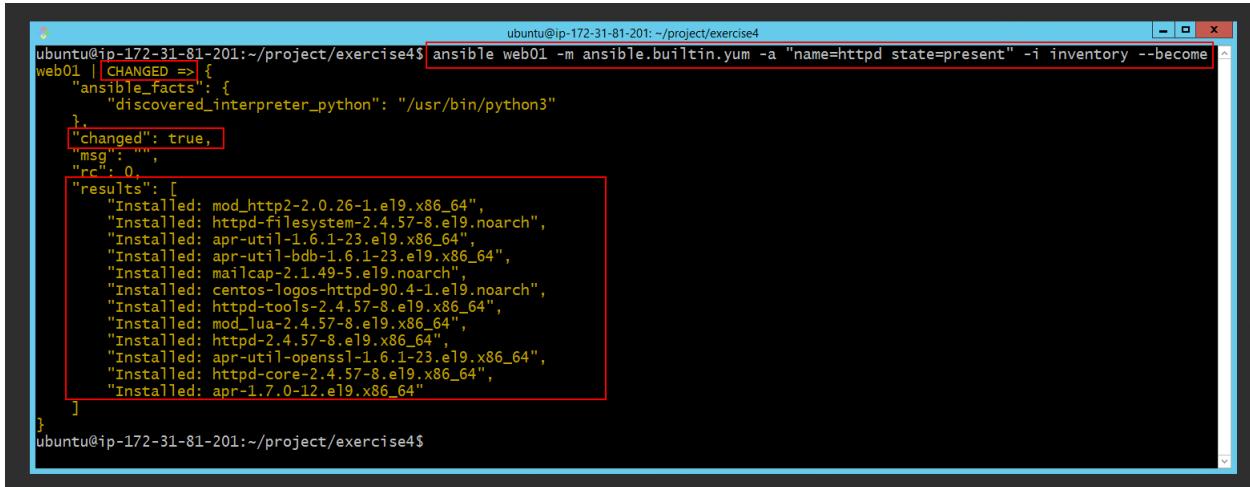
For example, suppose we want to install the `httpd` package on a CentOS Linux system. To do this, we use the `ansible.builtin.yum` module and then use the `-a` switch, which allows you to specify required parameters such as the package name and state. In this example, we use `present` as the state, which means that if the package is not installed on the target, it should be installed. However, as shown in the image below, when we run this Ad Hoc command, we encounter the following error, which means that the command must be executed on the destination system with root privileges.



The screenshot shows a terminal window titled "ubuntu@ip-172-31-81-201: ~/project/exercise4". The user has run several commands: "cp -r exercise3 exercise4", "cd exercise4", and "ls". Then, they run "ansible web01 -m ansible.builtin.yum -a \"name=httpd state=present\" -i inventory". The output shows an error for the "web01" host, indicating it failed because the command needs to be run under the root user. The error message is highlighted in red: "msg": "This command has to be run under the root user.".

```
ubuntu@ip-172-31-81-201:~/project$ cp -r exercise3 exercise4
ubuntu@ip-172-31-81-201:~/project$ cd exercise4
ubuntu@ip-172-31-81-201:~/project/exercise4$ ls
clientkey.pem  inventory
ubuntu@ip-172-31-81-201:~/project/exercise4$ ansible web01 -m ansible.builtin.yum -a "name=httpd state=present" -i inventory
web01 | [FAILED] => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "msg": "This command has to be run under the root user."
    "results": []
}
ubuntu@ip-172-31-81-201:~/project/exercise4$ |
```

To run the following command with root privileges, we use the `--become` switch.



```
ubuntu@ip-172-31-81-201:~/project/exercise4$ ansible web01 -m ansible.builtin.yum -a "name=httpd state=present" -i inventory --become
web01 | CHANGED => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": true,
  "msg": "",
  "rc": 0,
  "results": [
    "Installed: mod_http2-2.0.26-1.el9.x86_64",
    "Installed: httpd-filesystem-2.4.57-8.el9.noarch",
    "Installed: apr-util-1.6.1-23.el9.x86_64",
    "Installed: apr-util-bdb-1.6.1-23.el9.x86_64",
    "Installed: mailcap-2.1.49-5.el9.noarch",
    "Installed: centos-logos-httpd-90.4-1.el9.noarch",
    "Installed: httpd-tools-2.4.57-8.el9.x86_64",
    "Installed: mod_lua-2.4.57-8.el9.x86_64",
    "Installed: httpd-2.4.57-8.el9.x86_64",
    "Installed: apr-util-openssl-1.6.1-23.el9.x86_64",
    "Installed: httpd-core-2.4.57-8.el9.x86_64",
    "Installed: apr-1.7.0-12.el9.x86_64"
  ]
}
ubuntu@ip-172-31-81-201:~/project/exercise4$
```

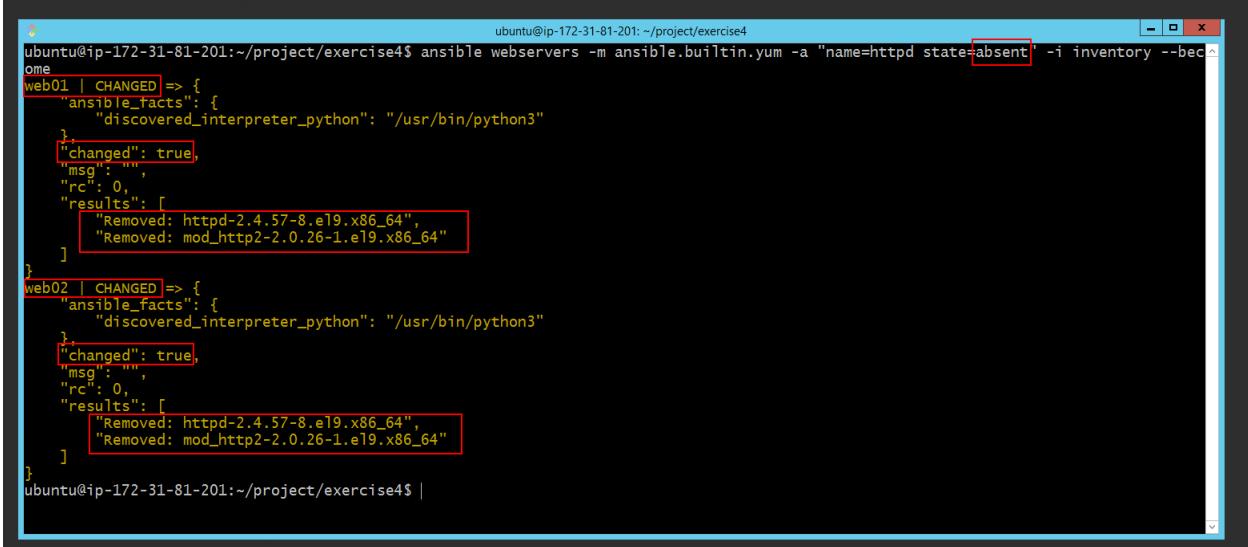
In the image below, pay attention to the word **changed**. Here you can see that Ansible can maintain configuration state. When you want to apply a configuration to a target, it first compares the configuration with the previous state. If the configurations are the same, it does not apply any changes to the target, and the value of **changed** will be **false**. However, if the changes differ from the current configuration state, Ansible applies the changes to the target, and the value of **changed** becomes **true**.

```
ubuntu@ip-172-31-81-201:~/project/exercise4$ ansible webservers -m ansible.builtin.yum -a "name=httpd state=present" -i inventory --become
web01 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "msg": "Nothing to do",
    "rc": 0,
    "results": []
}
web02 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": true,
    "msg": "",
    "rc": 0,
    "results": [
        "Installed: mod_http2-2.0.26-1.el9.x86_64",
        "Installed: httpd-filesystem-2.4.57-8.el9.noarch",
        "Installed: apr-util-1.6.1-23.el9.x86_64",
        "Installed: apr-util-bdb-1.6.1-23.el9.x86_64",
        "Installed: mailcap-2.1.49-5.el9.noarch",
        "Installed: centos-logos-httpd-90.4-1.el9.noarch",
        "Installed: httpd-tools-2.4.57-8.el9.x86_64",
        "Installed: mod_lua-2.4.57-8.el9.x86_64",
        "Installed: httpd-2.4.57-8.el9.x86_64",
        "Installed: apr-util-openssl-1.6.1-23.el9.x86_64",
        "Installed: httpd-core-2.4.57-8.el9.x86_64",
        "Installed: apr-1.7.0-12.el9.x86_64"
    ]
}
ubuntu@ip-172-31-81-201:~/project/exercise4$
```

As shown in the image below, because we are trying to apply the same settings again to the target, Ansible does not apply these changes to the target.

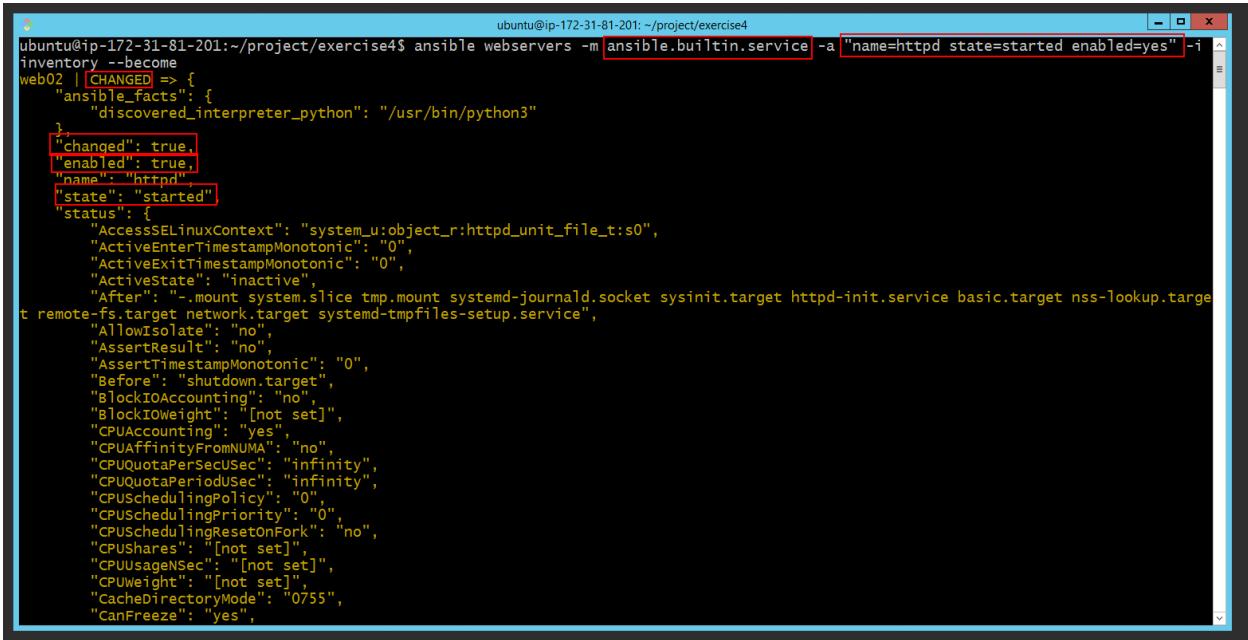
```
ubuntu@ip-172-31-81-201:~/project/exercise4$ ansible webservers -m ansible.builtin.yum -a "name=httpd state=present" -i inventory --become
web01 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "msg": "Nothing to do",
    "rc": 0,
    "results": []
}
web02 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "msg": "Nothing to do",
    "rc": 0,
    "results": []
}
ubuntu@ip-172-31-81-201:~/project/exercise4$
```

Now, at this stage, we change the state to **absent**. This state is used for uninstalling or removing configurations.



```
ubuntu@ip-172-31-81-201:~/project/exercise4$ ansible web01 -m ansible.builtin.yum -a "name=httpd state=absent" -i inventory --become
[...]
web01 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": true,
    "msg": "",
    "rc": 0,
    "results": [
        "Removed: httpd-2.4.57-8.el9.x86_64",
        "Removed: mod_http2-2.0.26-1.el9.x86_64"
    ]
}
[...]
web02 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": true,
    "msg": "",
    "rc": 0,
    "results": [
        "Removed: httpd-2.4.57-8.el9.x86_64",
        "Removed: mod_http2-2.0.26-1.el9.x86_64"
    ]
}
ubuntu@ip-172-31-81-201:~/project/exercise4$ |
```

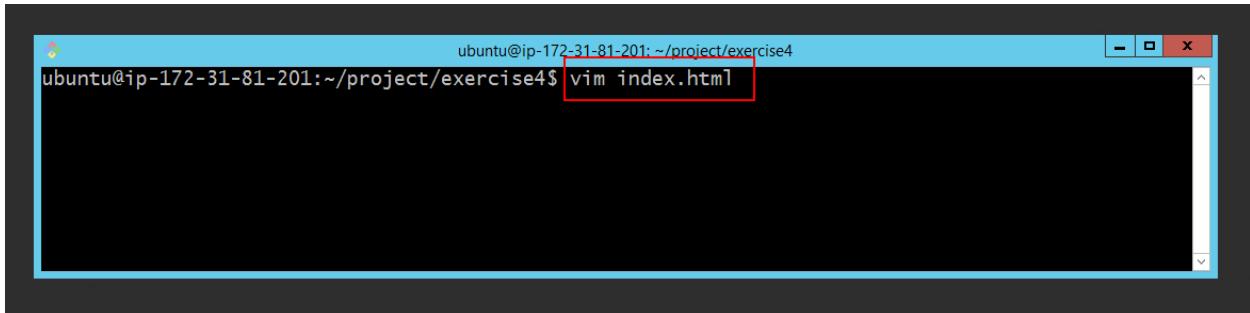
At this stage, we use another module called `ansible.builtin.service` for working with services. The `name` parameter specifies the name of the service, the `state` parameter defines the service status, and the `enabled` parameter is used to enable the service.



```
ubuntu@ip-172-31-81-201:~/project/exercise4$ ansible web02 -m ansible.builtin.service -a "name=httpd state=started enabled=yes" -i inventory --become
[...]
web02 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": true,
    "enabled": true,
    "name": "httpd",
    "state": "started",
    "status": {
        "AccessSelinuxContext": "system_u:object_r:httpd_unit_file_t:s0",
        "ActiveEnterTimestampMonotonic": "0",
        "ActiveExitTimestampMonotonic": "0",
        "ActiveState": "inactive",
        "After": "-.mount system.slice tmp.mount systemd-journald.socket sysinit.target httpd-init.service basic.target nss-lookup.target remote-fs.target network.target systemd-tmpfiles-setup.service",
        "AllowIsolate": "no",
        "AssertResult": "no",
        "AssertTimestampMonotonic": "0",
        "Before": "shutdown.target",
        "BlockIOAccounting": "no",
        "BlockIOWeight": "[not set]",
        "CPUAccounting": "yes",
        "CPUAffinityFromNUMA": "no",
        "CPUQuotaPerSecUsec": "infinity",
        "CPUQuotaPeriodUsec": "infinity",
        "CPUSchedulingPolicy": "0",
        "CPUSchedulingPriority": "0",
        "CPUSchedulingResetOnFork": "no",
        "CPUSHares": "[not set]",
        "CPUUsageSec": "[not set]",
        "CPUWeight": "[not set]",
        "CachedDirectoryMode": "0755",
        "CanFreeze": "yes",
        [...]
    }
}
ubuntu@ip-172-31-81-201:~/project/exercise4$ |
```

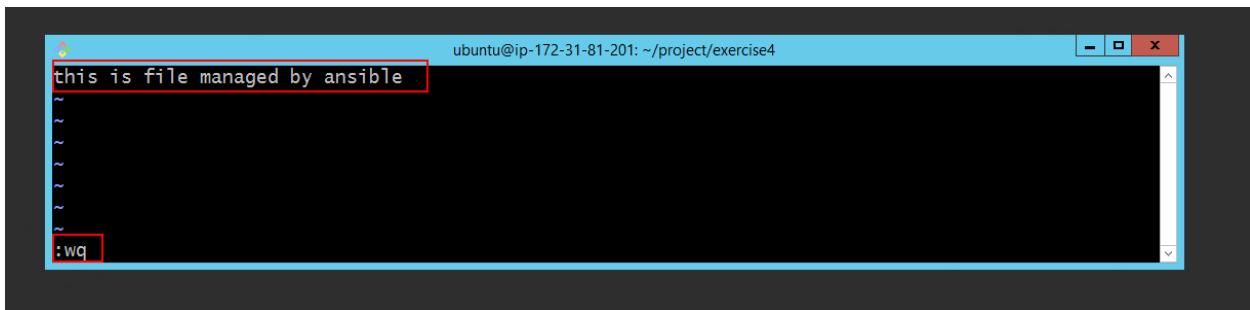
At this stage, we intend to create an HTML file on the control machine and then copy it to the target machine using Ansible modules.

Use the following command to create an HTML file.



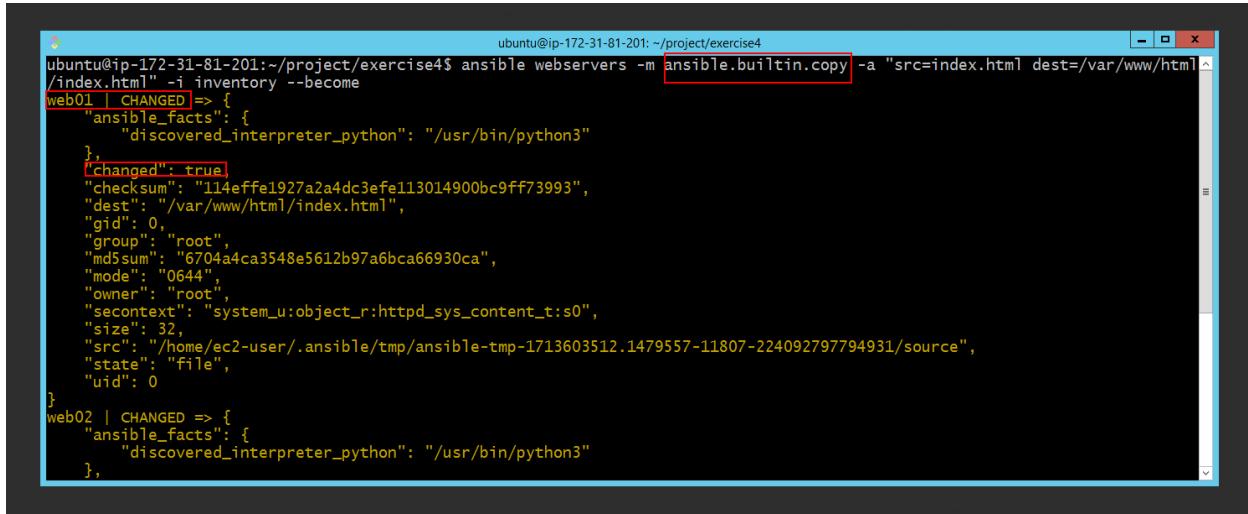
A screenshot of a terminal window on an Ubuntu system. The window title is "ubuntu@ip-172-31-81-201: ~/project/exercise4". The command "vim index.html" is typed into the terminal, with the entire command highlighted by a red box. The terminal has a dark background with light-colored text.

At this stage, write a test message inside the HTML file and then save it.



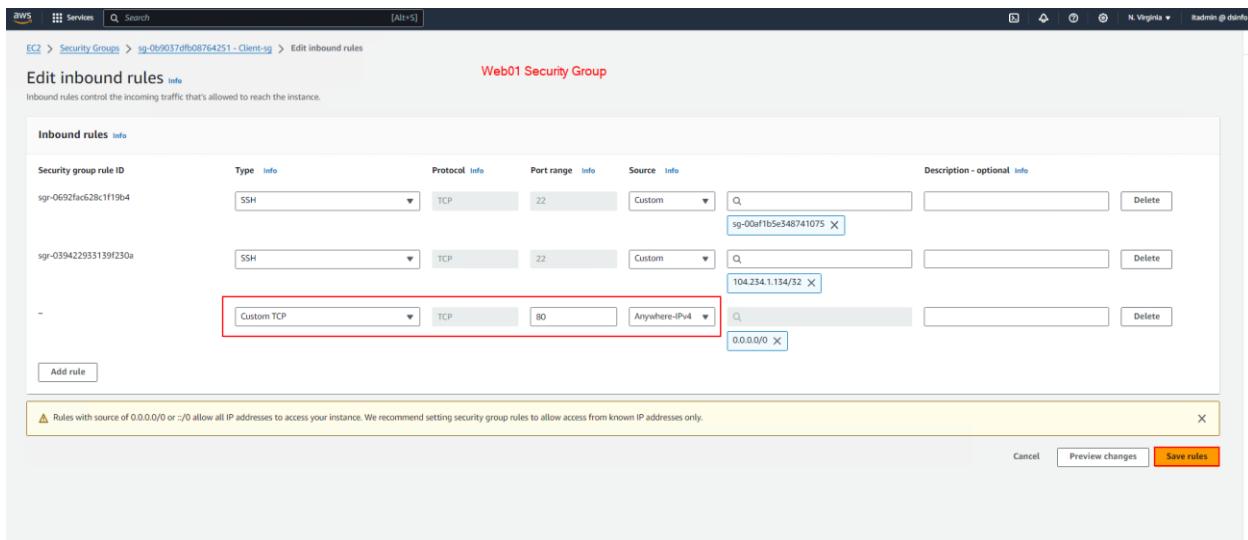
A screenshot of a terminal window on an Ubuntu system. The window title is "ubuntu@ip-172-31-81-201: ~/project/exercise4". Inside the terminal, the message "this is file managed by ansible" is displayed, followed by several blank lines. At the bottom, the command ":wq" is typed, with the entire command highlighted by a red box. The terminal has a dark background with light-colored text.

To copy a file from the Ansible machine to the target machine, you can use the `ansible.builtin.copy` module. In this module, you can use the `src` and `dest` parameters, which specify the source and destination paths for the file copy.



```
ubuntu@ip-172-31-81-201:~/project/exercise4$ ansible webservers -m ansible.builtin.copy -a "src=index.html dest=/var/www/html"
[web01 | CHANGED => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": true,
  "checksum": "114effe1927a2a4dc3efe113014900bc9ff73993",
  "dest": "/var/www/html/index.html",
  "gid": 0,
  "group": "root",
  "md5sum": "6704a4ca3548e5612b97a6bca66930ca",
  "mode": "0644",
  "owner": "root",
  "secontext": "system_u:object_r:httpd_sys_content_t:s0",
  "size": 32,
  "src": "/home/ec2-user/.ansible/tmp/ansible-tmp-1713603512.1479557-11807-224092797794931/source",
  "state": "file",
  "uid": 0
}
[web02 | CHANGED => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
}
```

To test whether the file has been copied to the target, simply open port 80 on the target. Note that the target is a web server and the HTML file has been copied to the web server's directory.



As shown in the image below, when we enter the target's IP address in the browser, we can see the contents of the HTML file.



Introduction to Playbook and Module

A Playbook is like a script, similar to a Bash script. A Playbook is a collection of plays and is written in YAML format. In a Playbook, the **hosts** section specifies the targets, which in the example below is the **webservers** group. The **tasks** section contains the actions to be performed on the targets. Then, a module can be used—in the example below, **yum** is the name of the module used in the Playbook. For each module, a series of options or arguments can be specified in the form of key-value pairs.

```
- hosts: websrvgrp
  tasks:
    - yum:
        name: httpd
        state: present
```

In a Playbook, we can have multiple plays, and for each play, multiple tasks can be defined.

In the example below, we have a Playbook that contains two plays. The first one includes the **webserver** group, and the second includes the **dbserver** group, each specified with `- hosts`.

```
1 - hosts: websrvgrp
2
3   tasks:
4     - name: Install Apache
5
6     yum:
7       name: httpd
8       state: latest
9
10    - name: Deploy Config
11      copy:
12        src: file/httpd.conf
13        dest: /etc/httpd.conf
14
15    - hosts: dbsrvgrp
16      tasks:
17        - name: Install Postgresql
18          yum:
19            name: postgresql
20            state: latest
```

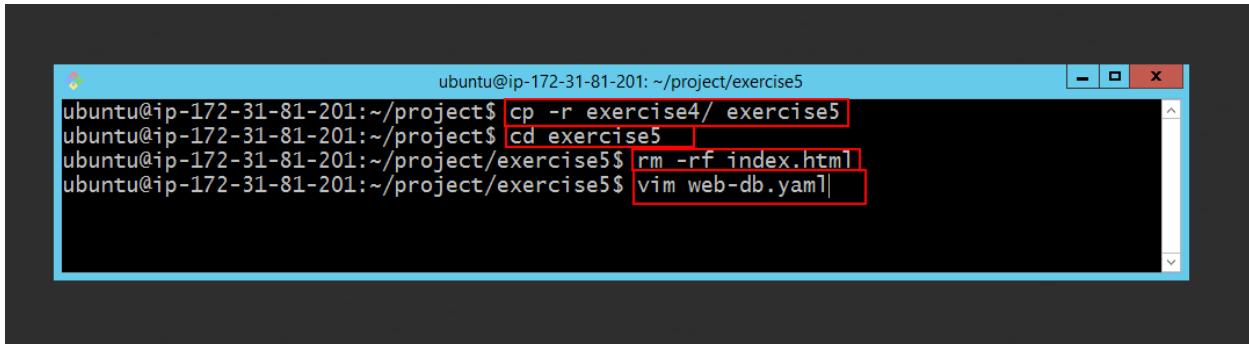
Each play in the example above contains a series of tasks. In the first play, which is related to the **webserver** group, there are two tasks.

Each task has a name specified by `name`. For example, the name of the first task is **Install Apache**, and the module used in this task is **yum**. Two options are defined for this module: `name` and `state`. This Playbook includes a three-level block structure. The first level is the **global area**, which contains `hosts` and `tasks`.

The second level is the **module name**, and the third level includes the **module options**.

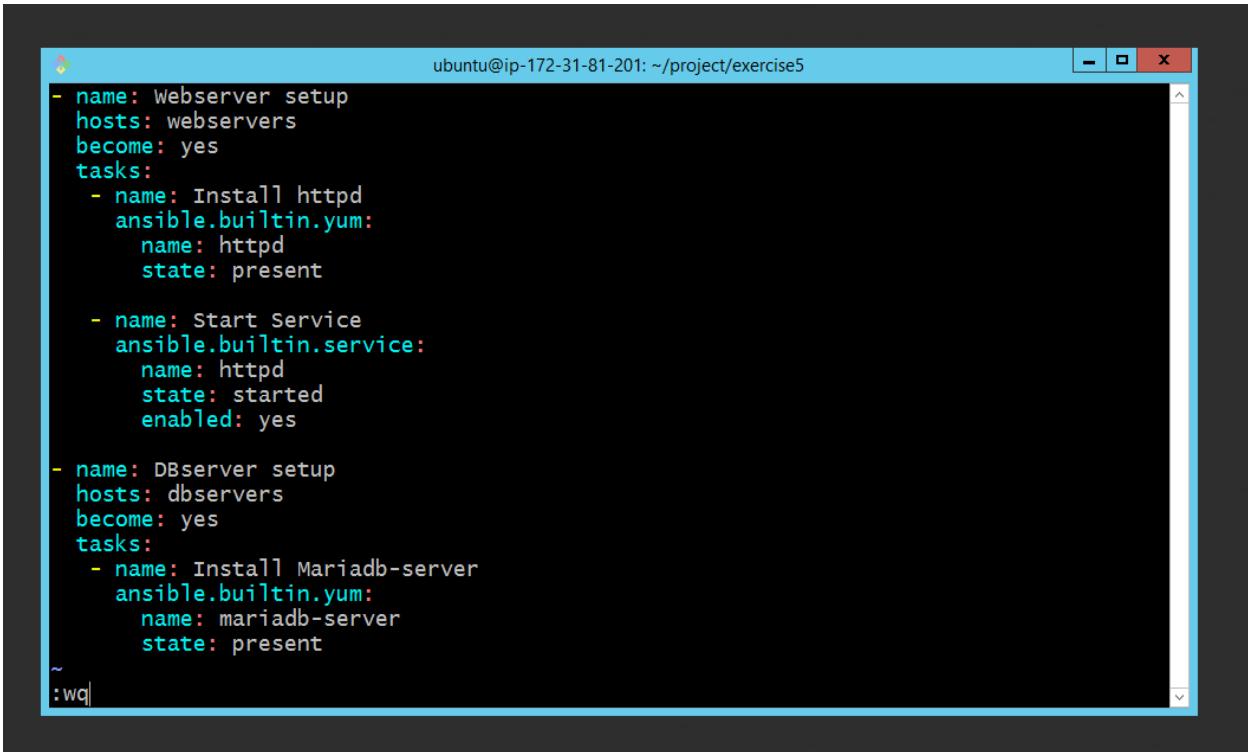
How to Create a Playbook

In the project directory, create a file with the `.yaml` extension. Just like the inventory file, you can choose any name for it.



```
ubuntu@ip-172-31-81-201:~/project/exercise5
ubuntu@ip-172-31-81-201:~/project$ cp -r exercise4/ exercise5
ubuntu@ip-172-31-81-201:~/project$ cd exercise5
ubuntu@ip-172-31-81-201:~/project/exercise5$ rm -rf index.html
ubuntu@ip-172-31-81-201:~/project/exercise5$ vim web-db.yaml|
```

Inside the file, type the script below. In the first play, it runs two tasks on the web server group: installing the `httpd` service and starting and enabling it. In the second play, it installs the database service on the `dbserver` group.



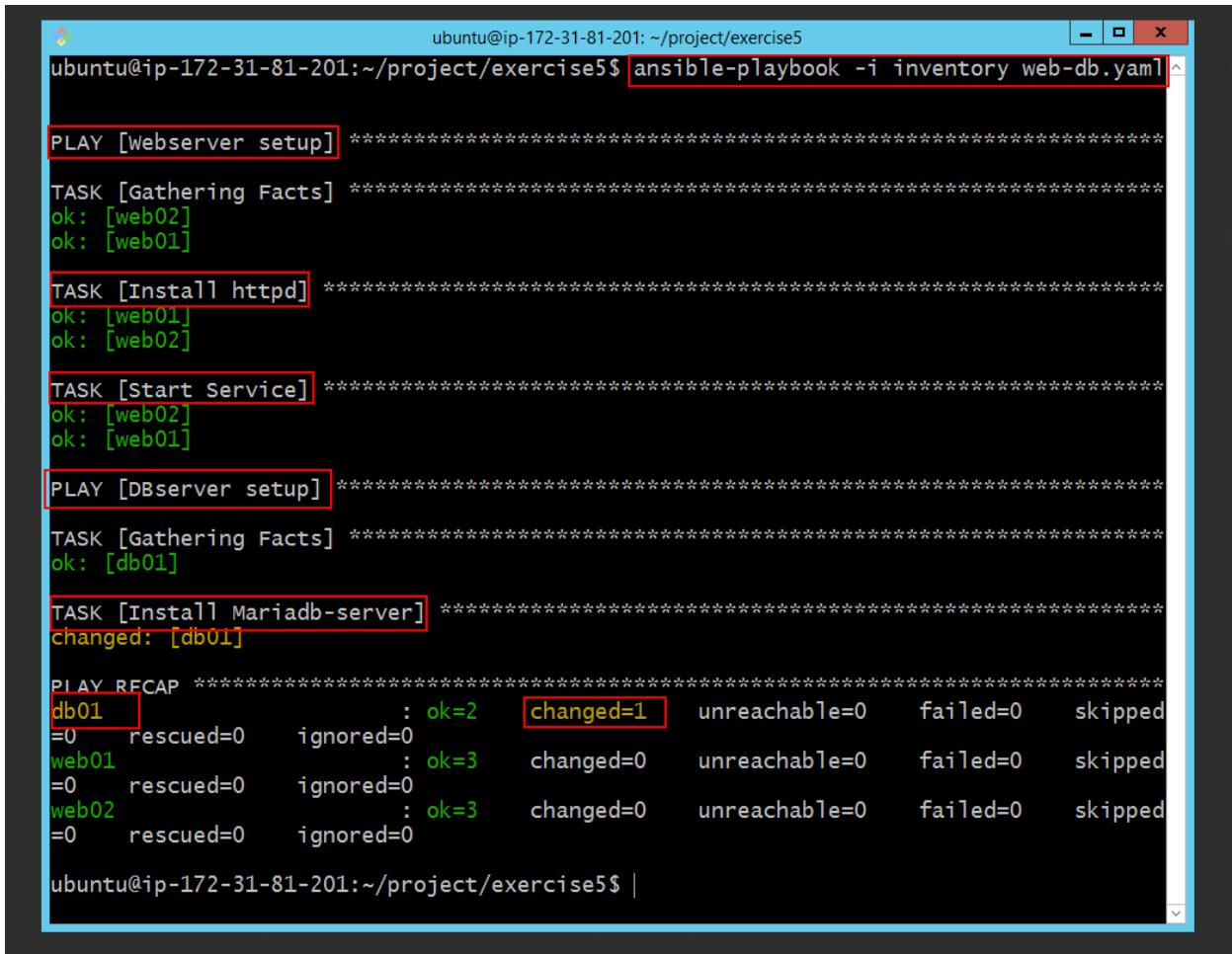
The screenshot shows a terminal window titled "ubuntu@ip-172-31-81-201: ~/project/exercise5". The terminal displays an Ansible playbook with two plays. The first play targets the "webservers" group and installs the "httpd" service. The second play targets the "dbservers" group and installs the "mariadb-server" service. The command ":wq" is visible at the bottom of the terminal window, indicating the user is about to save and quit.

```
- name: Webserver setup
  hosts: webservers
  become: yes
  tasks:
    - name: Install httpd
      ansible.builtin.yum:
        name: httpd
        state: present

    - name: Start Service
      ansible.builtin.service:
        name: httpd
        state: started
        enabled: yes

- name: DBserver setup
  hosts: dbservers
  become: yes
  tasks:
    - name: Install Mariadb-server
      ansible.builtin.yum:
        name: mariadb-server
        state: present
~ :wq|
```

Use the following command to run the Playbook.



The screenshot shows a terminal window on a Linux system (Ubuntu) with the command `ansible-playbook -i inventory web-db.yaml` highlighted in red. The output shows the execution of a playbook named `web-db.yaml`. The playbook consists of two main playbooks: `[Webserver setup]` and `[DBserver setup]`. The first play contains tasks for gathering facts and installing httpd on hosts `web01` and `web02`. The second play contains tasks for gathering facts and installing Mariadb-server on host `db01`. The final `PLAY RECAP` section summarizes the results for each host, showing counts for ok, changed, unreachable, failed, and skipped tasks.

```
ubuntu@ip-172-31-81-201:~/project/exercise5$ ansible-playbook -i inventory web-db.yaml

PLAY [Webserver setup] ****
TASK [Gathering Facts] ****
ok: [web02]
ok: [web01]

TASK [Install httpd] ****
ok: [web01]
ok: [web02]

TASK [Start Service] ****
ok: [web02]
ok: [web01]

PLAY [DBserver setup] ****
TASK [Gathering Facts] ****
ok: [db01]

TASK [Install Mariadb-server] ****
changed: [db01]

PLAY RECAP ****
db01 : ok=2    changed=1    unreachable=0    failed=0    skipped=0
      rescued=0    ignored=0
web01 : ok=3    changed=0    unreachable=0    failed=0    skipped=0
      rescued=0    ignored=0
web02 : ok=3    changed=0    unreachable=0    failed=0    skipped=0
      rescued=0    ignored=0

ubuntu@ip-172-31-81-201:~/project/exercise5$ |
```

In the output of the above command, there is a default task called **Gathering Facts**, which uses a module named **setup** to collect some information about the target.

If you add the `-v` switch at the end of the `ansible-playbook` command, it will provide detailed information during script execution, which is useful for troubleshooting.

```
ubuntu@ip-172-31-81-201:~/project/exercise$ ansible-playbook -i inventory web-db.yaml -v
Using /etc/ansible/ansible.cfg as config file

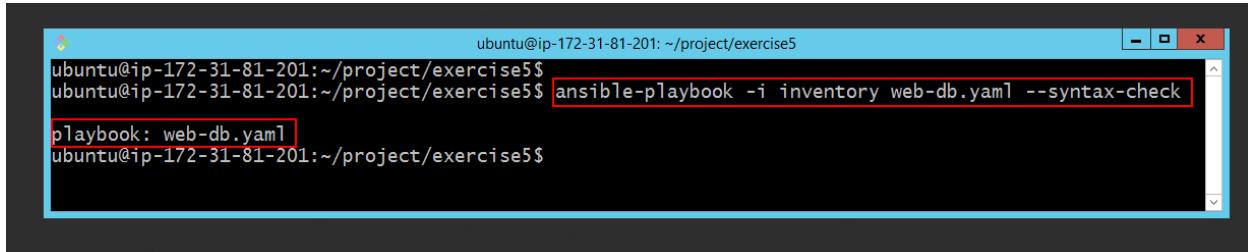
PLAY [Webserver setup] ****
TASK [Gathering Facts] ****
ok: [web02]
ok: [web01]

TASK [Install httpd] ****
ok: [web01] => {"changed": false, "msg": "Nothing to do", "rc": 0, "results": []}
ok: [web02] => {"changed": false, "msg": "Nothing to do", "rc": 0, "results": []}

TASK [Start Service] ****
ok: [web02] => {"changed": false, "enabled": true, "name": "httpd", "state": "started", "status": {"AccessSELinuxContext": "system_u:object_r:httpd_unit_file_t:s0", "ActiveEnterTimestamp": "Sun 2024-04-21 09:53:18 UTC", "ActiveEnterTimestampMonotonic": "14791717", "ActiveExitTimestampMonotonic": "0", "ActiveState": "active", "After": "network.target systemd-journald.socket .mount systemd-tmpfiles-setup.service remote-fs.target httpd-init.service nss-lookup.target basic.target sysinit.target tmp.mount system.slice", "AllowIsolate": "no", "AssertResult": "yes", "AssertTimestamp": "Sun 2024-04-21 09:53:18 UTC", "AssertTimestampMonotonic": "14457525", "Before": "multi-user.target shutdown.target", "BlockIOAccounting": "no", "BlockIOWeight": "[not set]", "CPUAccounting": "yes", "CPUAffinityFromNUMA": "no", "CPUQuotaPerSecUsec": "infinity", "CPUQuotaPeriodusec": "infinity", "CPUSchedulingPolicy": "0", "CPUSchedulingPriority": "0", "CPUSchedulingResetOnFork": "no", "CPUSHares": "[not set]", "CPUUsage_nsec": "1506000000", "CPUWeight": "[not set]", "CacheDirectoryMode": "0755", "CanFreeze": "yes", "CanIsolate": "no", "CanReload": "yes", "CanStart": "yes", "CanStop": "yes", "CapabilityBoundingSet": "cap_chown cap_dac_override cap_dac_read_search cap_fowner cap_fsetid cap_kill cap_setgid cap_setuid cap_setpcap cap_linux_immutable cap_net_bind_service cap_net_broadcast cap_net_admin cap_net_raw cap_ipc_lock cap_ip_c_owner cap_sys_module cap_sys_rawio cap_sys_chroot cap_sys_ptrace cap_sys_pacct cap_sys_admin cap_sys_boot cap_sys_nice cap_sys_resource cap_sys_time cap_sys_config cap_mknod cap_lease cap_audit_write cap_audit_control cap_setfcap cap_mac_override cap_mac_admin cap_syslog cap_wakelock cap_block_suspend cap_audit_read cap_perfmon cap_bpf cap_checkpoint_restore", "CleanResult": "success", "CollectMode": "inactive", "ConditionResult": "yes", "ConditionTimestamp": "Sun 2024-04-21 09:53:18 UTC", "ConditionTimestampMonotonic": "14457522", "ConfigurationDirectory": "M
```

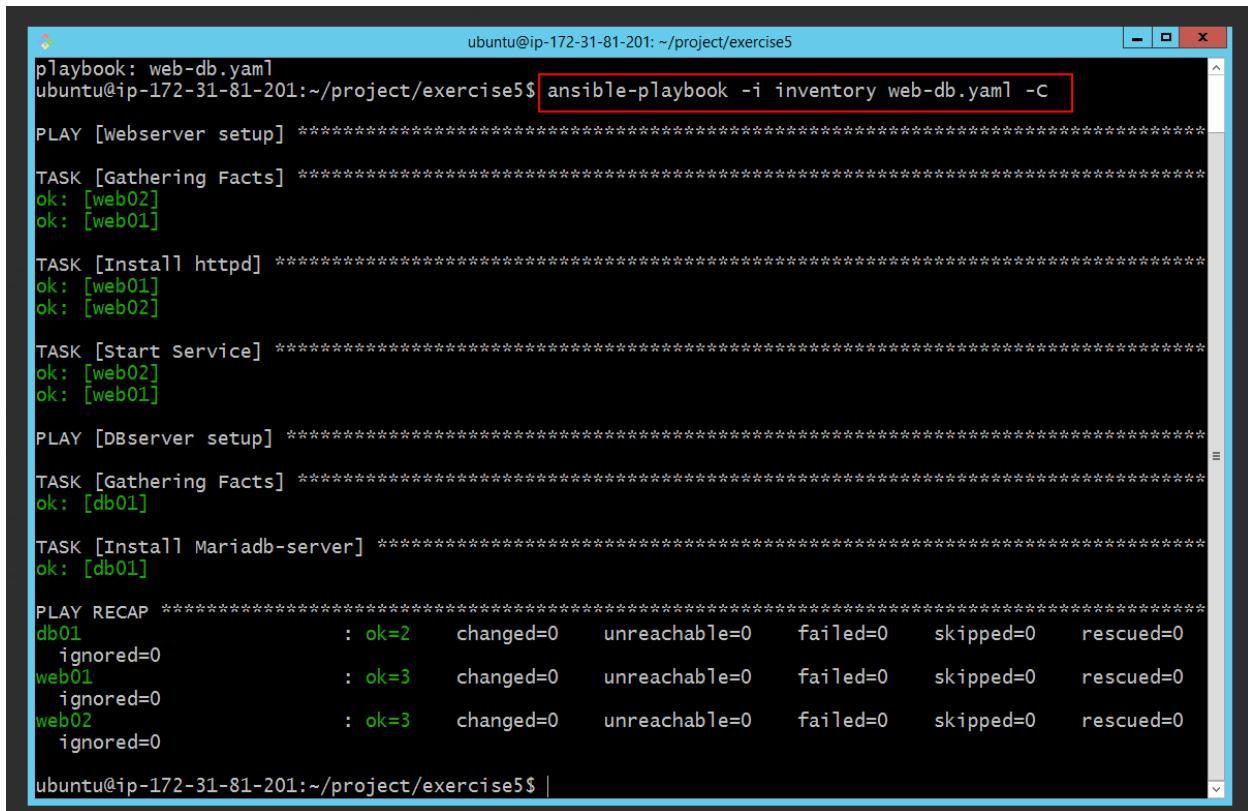
The `-v` switch stands for **verbose**, and you can use up to four `vs` in the command. The more `vs` you use, the more detailed the output will be.

To check the syntax of the YAML file, you can use the following switch:



```
ubuntu@ip-172-31-81-201:~/project/exercise5$ ansible-playbook -i inventory web-db.yaml --syntax-check
playbook: web-db.yaml
ubuntu@ip-172-31-81-201:~/project/exercise5$
```

One of the switches you can use with the `ansible-playbook` command is the `-c` switch, which tests the Playbook on the targets before actual execution. If there are any issues, it reports them in the output. This is known as a **dry run**, and it is recommended to use it in production environments.



```
playbook: web-db.yaml
ubuntu@ip-172-31-81-201:~/project/exercise5$ ansible-playbook -i inventory web-db.yaml -c
PLAY [Webserver setup] ****
TASK [Gathering Facts] ****
ok: [web02]
ok: [web01]

TASK [Install httpd] ****
ok: [web01]
ok: [web02]

TASK [Start Service] ****
ok: [web02]
ok: [web01]

PLAY [DBserver setup] ****
TASK [Gathering Facts] ****
ok: [db01]

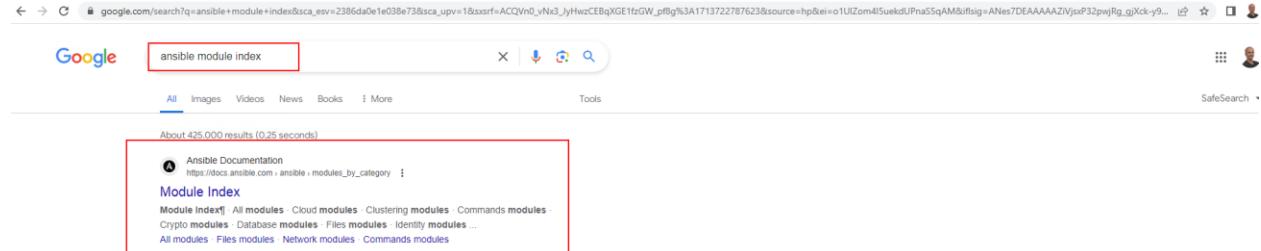
TASK [Install Mariadb-server] ****
ok: [db01]

PLAY RECAP ****
db01              : ok=2    changed=0    unreachable=0    failed=0    skipped=0    rescued=0
                   ignored=0
web01             : ok=3    changed=0    unreachable=0    failed=0    skipped=0    rescued=0
                   ignored=0
web02             : ok=3    changed=0    unreachable=0    failed=0    skipped=0    rescued=0
                   ignored=0
ubuntu@ip-172-31-81-201:~/project/exercise5$ |
```

How to Find and Use Modules

One of the tasks a DevOps specialist must be able to perform is finding and using modules based on the tasks in a Playbook.

To find modules, simply search the following phrase on Google:



In this section, we can view all Ansible modules categorized and organized.

The screenshot shows the Ansible Documentation website at docs.ansible.com/ansible/2.9/modules/modules_by_category.html. The left sidebar has a tree view of module categories. The main content area is titled "Module Index" and lists various module types. A red box highlights the list of module types: All modules, Cloud modules, Clustering modules, Commands modules, Crypto modules, Database modules, Files modules, Identity modules, Inventory modules, Messaging modules, Monitoring modules, Net Tools modules, Network modules, Notification modules, Packaging modules, Remote Management modules, Source Control modules, Storage modules, System modules, Utilities modules, Web Infrastructure modules, and Windows modules. Navigation buttons for "Previous" and "Next" are at the bottom.

For example, suppose we want to copy a file from the control machine to the web server machine, and on the other hand, we want to create a database and a database user on the DB server machine.

To copy a file, we can use the **Files** module, so we click on this module.

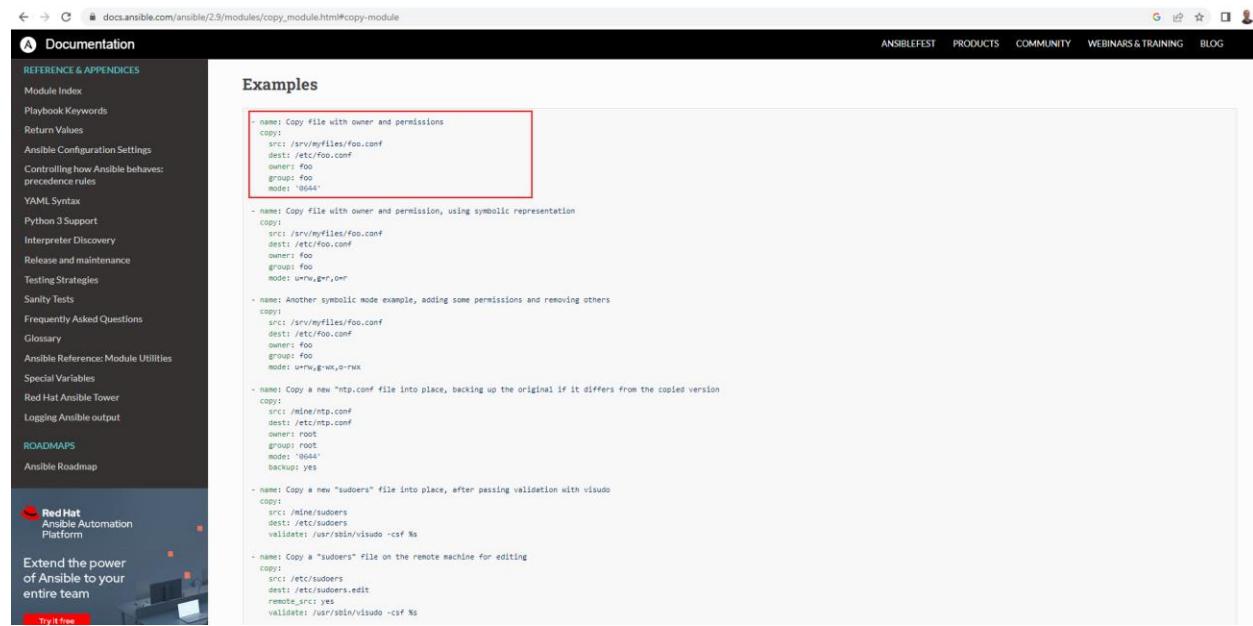
The screenshot shows the Ansible documentation website at docs.ansible.com/ansible/2.9/modules/modules_by_category.html. The page title is "Module Index". The left sidebar contains a tree view of module categories: All modules, Cloud modules, Clustering modules, Commands modules, Crypto modules, Database modules, Files modules (which is highlighted with a red box), Identity modules, Inventory modules, Messaging modules, Monitoring modules, Net Tools modules, Network modules, Notification modules, Packaging modules, Remote Management modules, Source Control modules, Storage modules, System modules, Utilities modules, Web Infrastructure modules, and Windows modules. Below the sidebar, there are sections for "CONTRIBUTING TO ANSIBLE" and "EXTENDING ANSIBLE". The main content area shows a list of all module categories again, with "Files modules" also highlighted by a red box. At the bottom of the content area are navigation buttons for "Previous" and "Next".

As shown in the image below, there are many file-related modules available in this section.

Since we want to copy a file from the control machine to the web server, we click on the following module: **copy**.

The screenshot shows the Ansible documentation website at docs.ansible.com/ansible/2.9/modules/list_of_files_modules.html. The page title is "Files modules". On the left, there's a sidebar with categories like "Identity modules", "Inventory modules", etc., and sections for "Contributing to Ansible", "Extending Ansible", and "Common Ansible Scenarios". The main content area lists various file-related modules with their descriptions. The "copy" module is highlighted with a red box. A note at the bottom states: "(D) This marks a module as deprecated, which means a module is kept for backwards compatibility but usage is discouraged. The module documentation details page may explain more about this rationale."

After opening the module, as you can see, there are various examples of file copying available.



The screenshot shows the Ansible documentation page for the 'copy' module. The left sidebar contains links to various Ansible resources like 'Module Index', 'Playbook Keywords', 'Return Values', etc. The main content area is titled 'Examples' and contains several code snippets demonstrating different ways to use the 'copy' module. One snippet is highlighted with a red border:

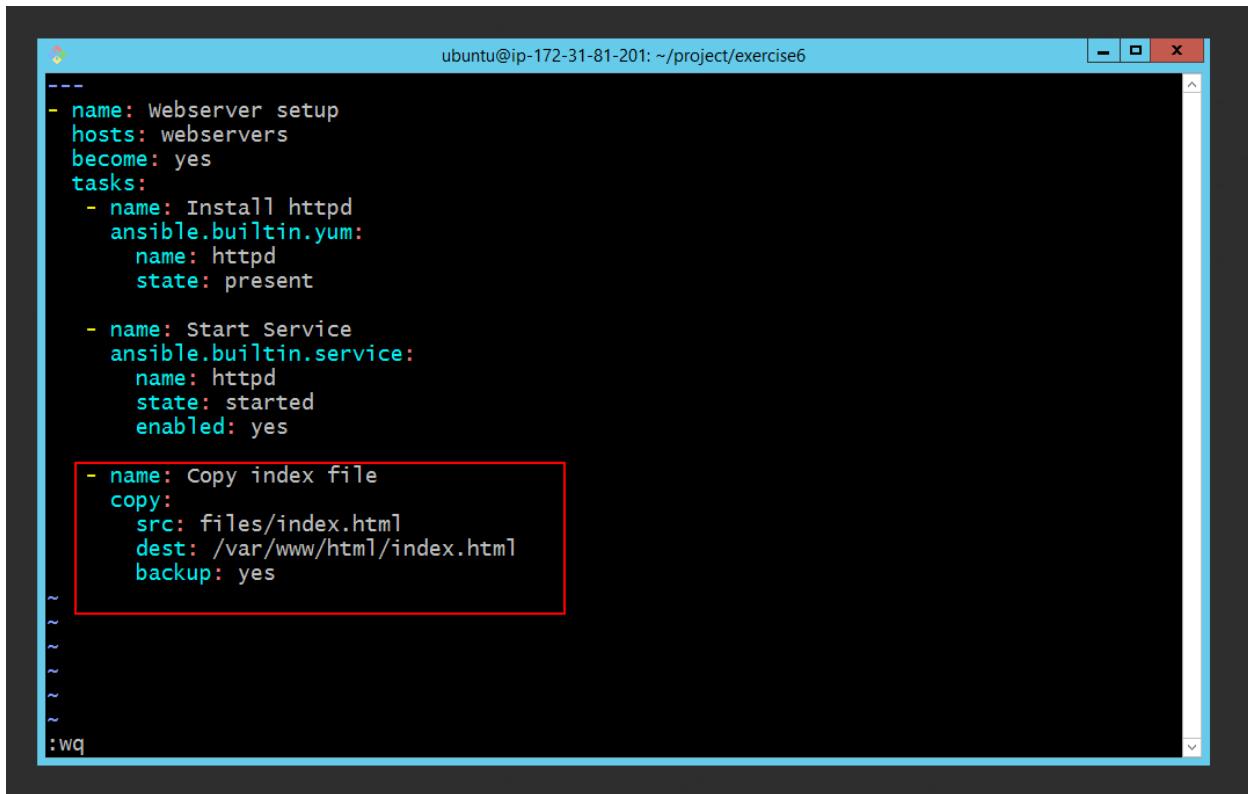
```
- name: Copy file with owner and permissions
  copy:
    src: /srv/myfiles/foo.conf
    dest: /etc/foo.conf
    owner: foo
    group: foo
    mode: '0644'
    backup: yes
```

At this stage, create a folder for the project and then open the Playbook file using the VIM editor.

In the Playbook file, create a new task to copy the `index.html` file from the control machine to the web server's directory. Use the **backup** option as well, which will first create a backup of the existing file at the destination if it exists, and then copy the new file.

Some options of a module are mandatory, and next to them, the word **Required** is listed, meaning they must be used. Other options are **Optional**, and you can use them if needed.

Add the following task to the Playbook and then save the file.



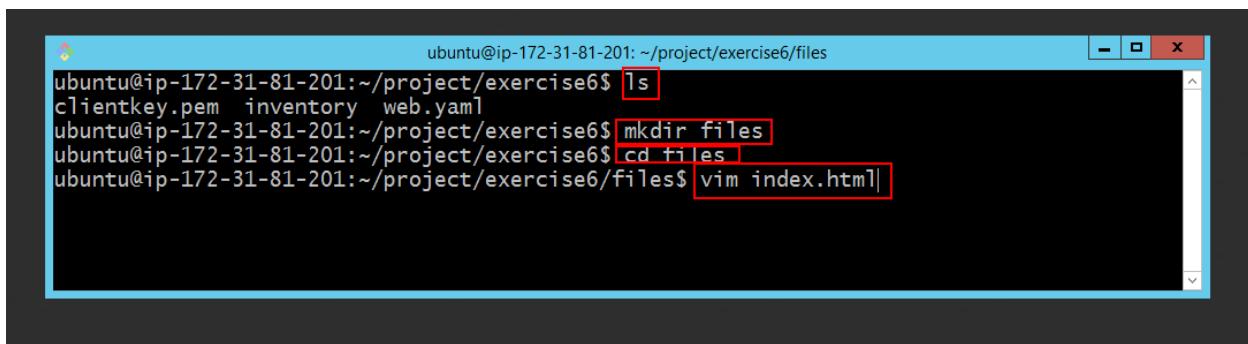
```
ubuntu@ip-172-31-81-201: ~/project/exercise6
---
- name: Webserver setup
  hosts: webservers
  become: yes
  tasks:
    - name: Install httpd
      ansible.builtin.yum:
        name: httpd
        state: present

    - name: Start Service
      ansible.builtin.service:
        name: httpd
        state: started
        enabled: yes

    - name: Copy index file
      copy:
        src: files/index.html
        dest: /var/www/html/index.html
        backup: yes
~
~
~
~
~
:wq
```

The terminal window shows an Ansible playbook named 'exercise6'. It contains three tasks: installing httpd via yum, starting the httpd service, and copying 'index.html' from the 'files' directory to the '/var/www/html' directory. The 'copy' task is highlighted with a red box. The command ':wq' at the bottom indicates the user is saving and exiting the editor.

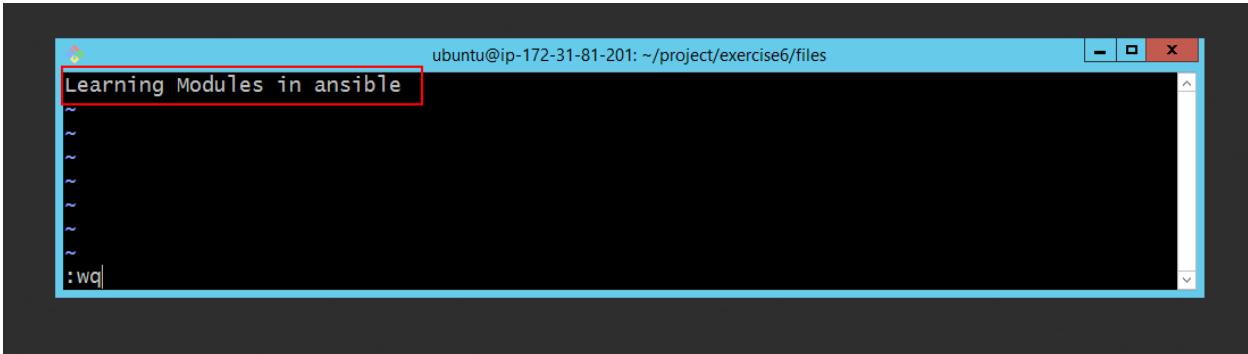
Now, since the `index.html` file needs to be copied from the control machine, you should copy the `files` folder and the `index.html` file to the control machine.



```
ubuntu@ip-172-31-81-201: ~/project/exercise6/files
ubuntu@ip-172-31-81-201:~/project/exercise6$ ls
clientkey.pem  inventory  web.yaml
ubuntu@ip-172-31-81-201:~/project/exercise6$ mkdir files
ubuntu@ip-172-31-81-201:~/project/exercise6$ cd files
ubuntu@ip-172-31-81-201:~/project/exercise6/files$ vim index.html|
```

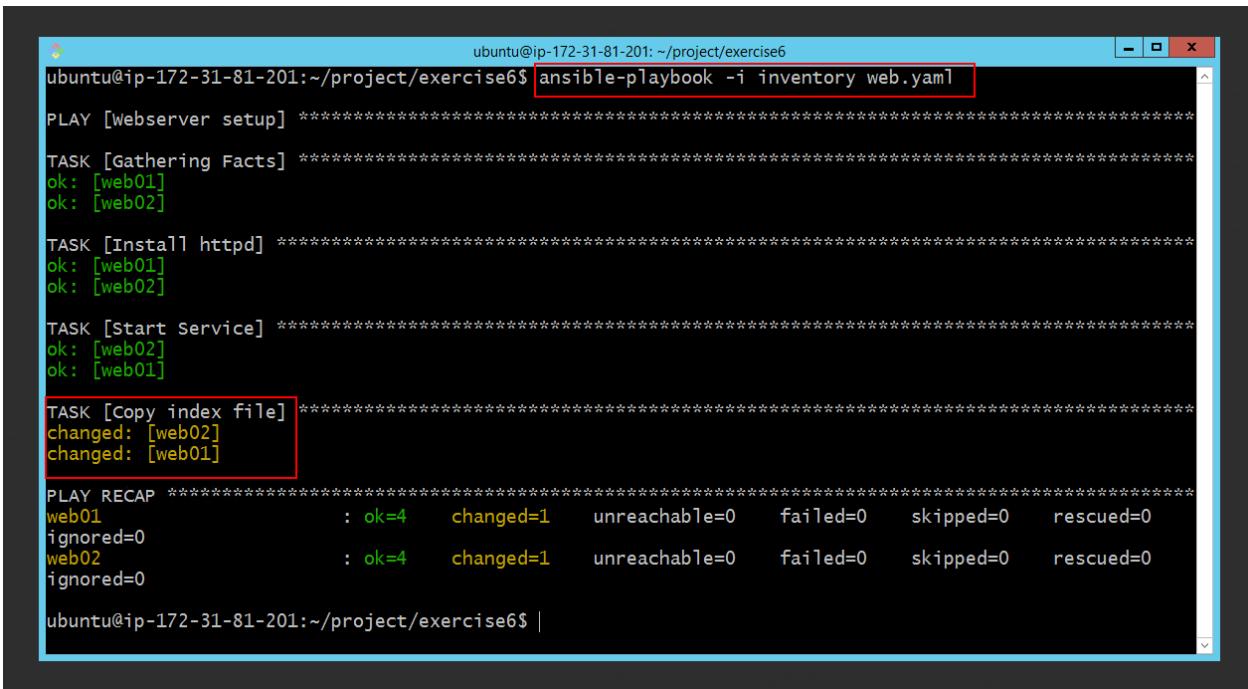
The terminal window shows the creation of a 'files' directory and navigating into it. The 'index.html' file is being edited with Vim, with the command 'vim index.html' highlighted by a red box.

At this stage, create some test content inside the `index.html` file and then save the file.



A screenshot of a terminal window titled "ubuntu@ip-172-31-81-201: ~/project/exercise6/files". The window contains a single line of text: "Learning Modules in ansible". Below this, there are several tilde (~) characters and the command ":wq" at the bottom. The entire terminal window is highlighted with a blue border.

At this stage, use the following command to test the Playbook.



A screenshot of a terminal window titled "ubuntu@ip-172-31-81-201: ~/project/exercise6\$". The command "ansible-playbook -i inventory web.yaml" is being run. The output shows the execution of the playbook, including tasks for gathering facts, installing httpd, starting services, and copying index files. A specific task, "TASK [Copy index file]", is highlighted with a red box, showing it changed files on both hosts. The final PLAY RECAP summary is also shown.

```
ubuntu@ip-172-31-81-201:~/project/exercise6$ ansible-playbook -i inventory web.yaml
PLAY [Webserver setup] ****
TASK [Gathering Facts] ****
ok: [web01]
ok: [web02]

TASK [Install httpd] ****
ok: [web01]
ok: [web02]

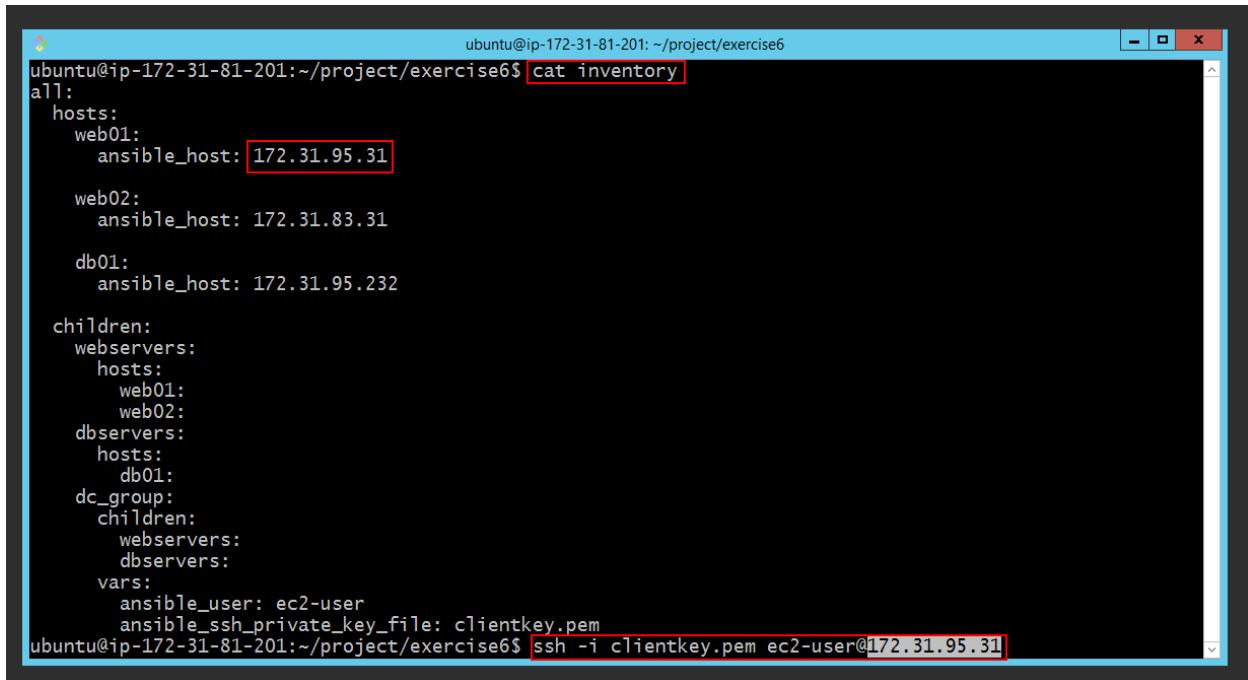
TASK [Start Service] ****
ok: [web02]
ok: [web01]

TASK [Copy index file] ****
changed: [web02]
changed: [web01]

PLAY RECAP ****
web01 : ok=4    changed=1    unreachable=0    failed=0    skipped=0    rescued=0
ignored=0
web02 : ok=4    changed=1    unreachable=0    failed=0    skipped=0    rescued=0
ignored=0

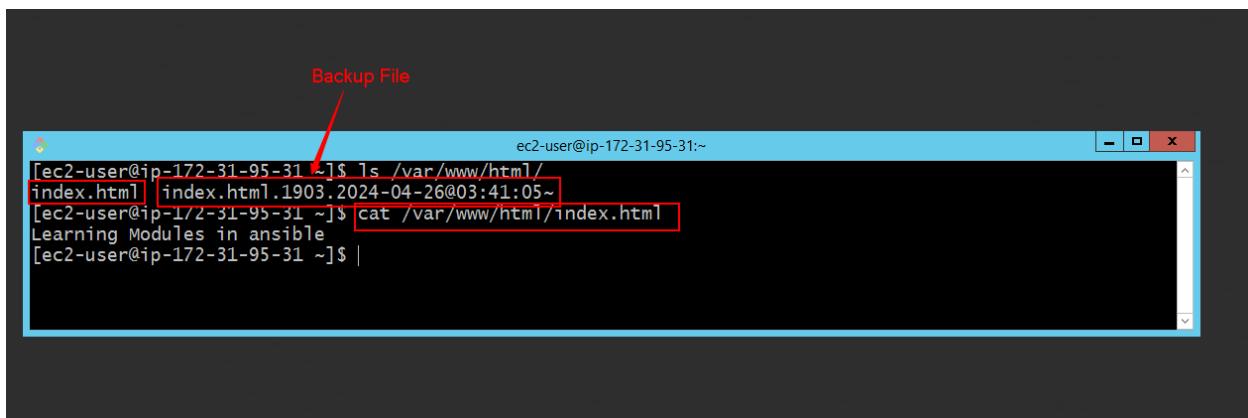
ubuntu@ip-172-31-81-201:~/project/exercise6$ |
```

At this stage, to ensure that the file has been copied to the web server, SSH into the web server.



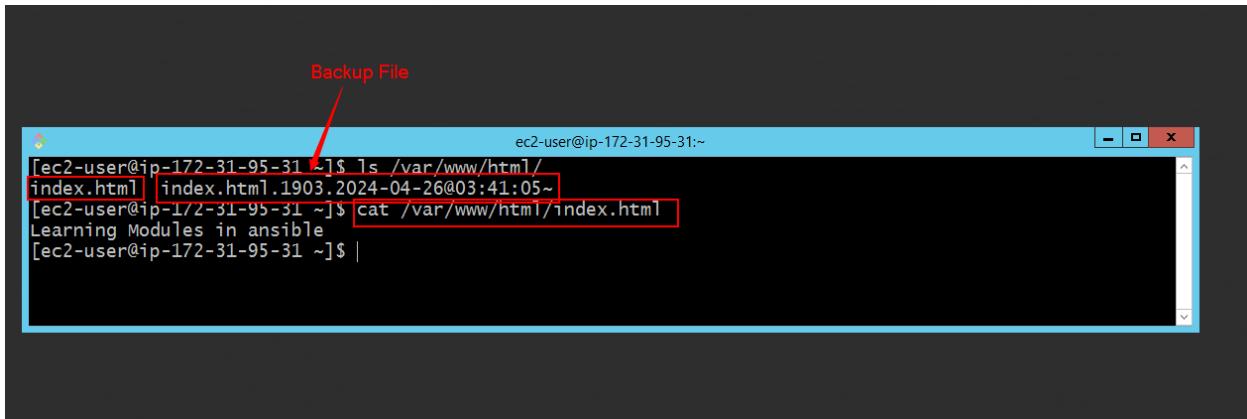
A screenshot of a terminal window titled "ubuntu@ip-172-31-81-201: ~/project/exercise6". The window shows the contents of an Ansible inventory file. A red box highlights the command "cat inventory". The inventory defines a group "all" with hosts "web01" and "web02", and a database host "db01". It also defines children groups "webservers" and "dbservers", and a variable group "dc_group". A red box highlights the command "ssh -i clientkey.pem ec2-user@172.31.95.31".

As shown in the image below, the file has been correctly placed in the web server's directory. Since the file already existed at this path, Ansible first created a backup of the existing file and then copied the new file.



A screenshot of a terminal window titled "ec2-user@ip-172-31-95-31:~". The window shows the command "ls /var/www/html/" followed by "index.html index.html.1903.2024-04-26@03:41:05~". A red arrow points from the text "Backup File" to the backup file name. The command "cat /var/www/html/index.html" is then run, displaying the content "Learning Modules in ansible".

At this stage, use the `exit` command to exit the web server environment and return to the Control Machine terminal.

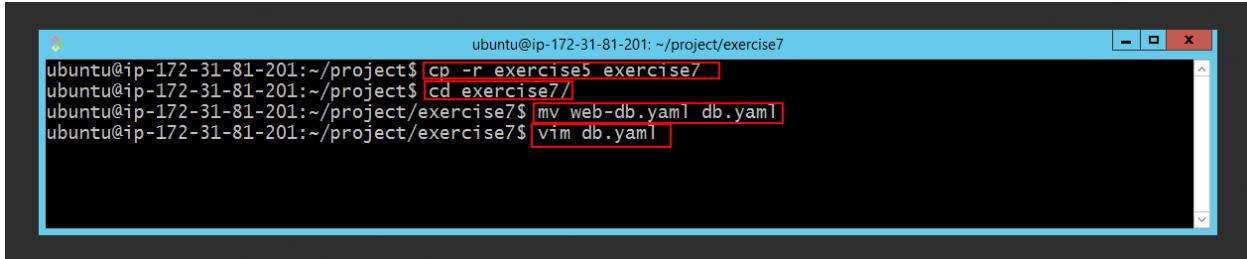


```
ec2-user@ip-172-31-95-31:~$ ls /var/www/html/
index.html index.html.1903.2024-04-26@03:41:05~
[ec2-user@ip-172-31-95-31 ~]$ cat /var/www/html/index.html
Learning Modules in ansible
[ec2-user@ip-172-31-95-31 ~]$ |
```

Note:

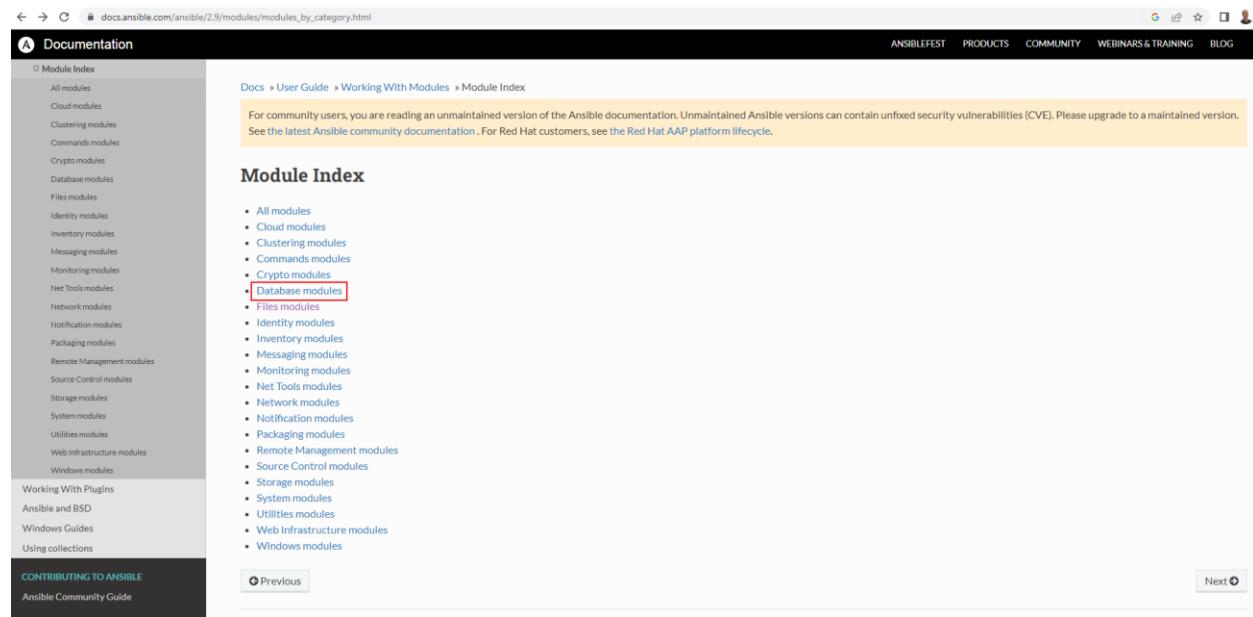
Some modules require dependencies.

To get familiar with these dependencies, we will create a new project in this stage. The goal is to use a module to install a database and then start its service. After that, we will create a database and a database user using a Playbook.



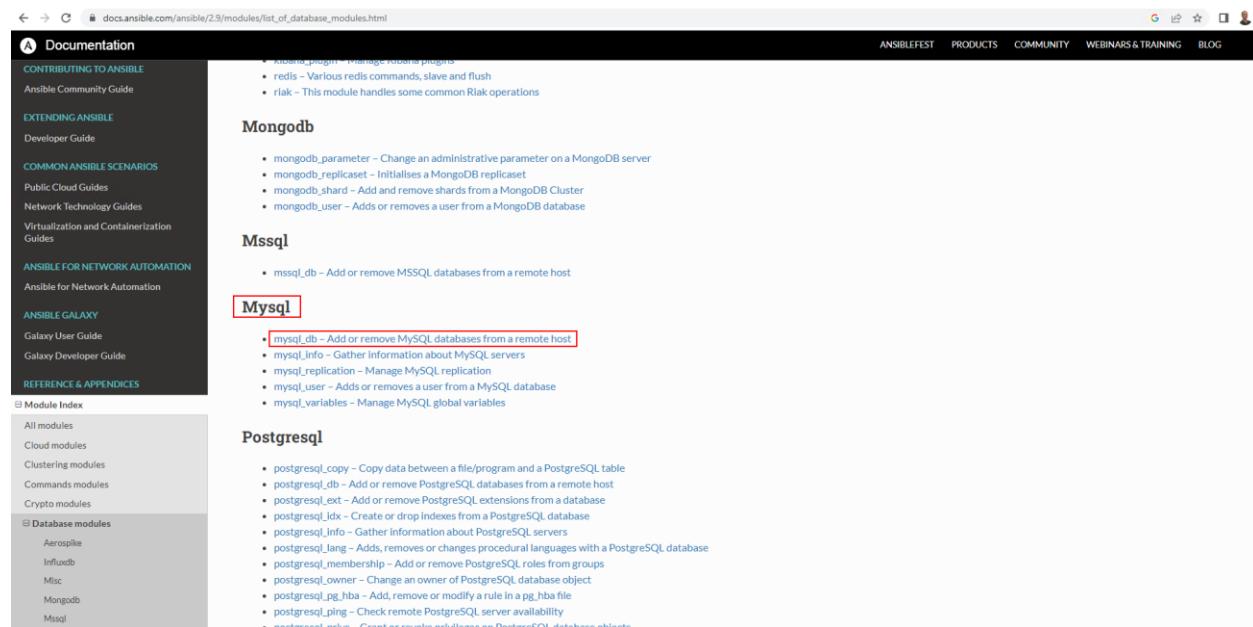
```
ubuntu@ip-172-31-81-201:~/project/exercise7$ cp -r exercise5 exercise7
ubuntu@ip-172-31-81-201:~/project$ cd exercise7/
ubuntu@ip-172-31-81-201:~/project/exercise7$ mv web-db.yaml db.yaml
ubuntu@ip-172-31-81-201:~/project/exercise7$ vim db.yaml |
```

To work with databases, click on the **Database Module** section.



The screenshot shows the Ansible documentation website at docs.ansible.com/ansible/2.9/modules/modules_by_category.html. The page title is "Module Index". The left sidebar contains a list of module categories, and the main content area shows a hierarchical list of modules under "Database modules", which is highlighted with a red box. The URL in the browser bar is docs.ansible.com/ansible/2.9/modules/modules_by_category.html.

Since we intend to create a database on MySQL, click on the following module: **mysql_db**.



The screenshot shows the Ansible documentation website at docs.ansible.com/ansible/2.9/modules/list_of_database_modules.html. The page title is "List of Database Modules". The left sidebar contains sections like "CONTRIBUTING TO ANSIBLE", "EXTENDING ANSIBLE", "COMMON ANSIBLE SCENARIOS", "ANSIBLE FOR NETWORK AUTOMATION", and "REFERENCE & APPENDICES". The main content area lists various database modules, including "MongoDB", "MySQL", and "PostgreSQL". The "mysql_db" module is highlighted with a red box. The URL in the browser bar is docs.ansible.com/ansible/2.9/modules/list_of_database_modules.html.

Instructor: Fariborz Fallahzadeh

Email Address: fariborz.fallahzadeh@gmail.com

In the **Example** section, you can see examples of how to create a database.

The screenshot shows the Ansible documentation for the MySQL module. The left sidebar contains links to various Ansible resources. The main content area has a heading 'Examples' with several code snippets highlighted by red boxes:

```
- name: Create a new database with name 'bobdata'
  mysql_db:
    name: bobdata
    state: present

# Copy database dump file to remote host and restore it to database 'my_db'
- name: Copy database dump file
  copy:
    src: dump.sql.bz2
    dest: /tmp

- name: Restore database
  mysql_db:
    name: my_db
    state: import
    targets: /tmp/dump.sql.bz2

- name: Dump multiple databases
  mysql_db:
    state: dump
    name: db_1,db_2
    targets: /tmp/dump.sql

- name: Dump multiple databases
  mysql_db:
    state: dump
```

As seen in this scenario, to create a database, you first need to ensure that the database is installed using a module. Then, you need to use another module to start the database service, and finally, the database can be created.

The screenshot shows a terminal window on an Ubuntu system with the command `ubuntu@ip-172-31-81-201: ~/project/exercise7`. It displays two Ansible playbooks. The first playbook installs MariaDB and starts the service, while the second playbook creates a database named 'accounts'. The code is highlighted with red boxes:

```
---
- name: DBserver setup
  hosts: dbservers
  become: yes
  tasks:
    - name: Install Mariadb-server
      ansible.builtin.yum:
        name: mariadb-server
        state: present

    - name: Start Mariadb Service
      ansible.builtin.service:
        name: mariadb
        state: started
        enabled: yes

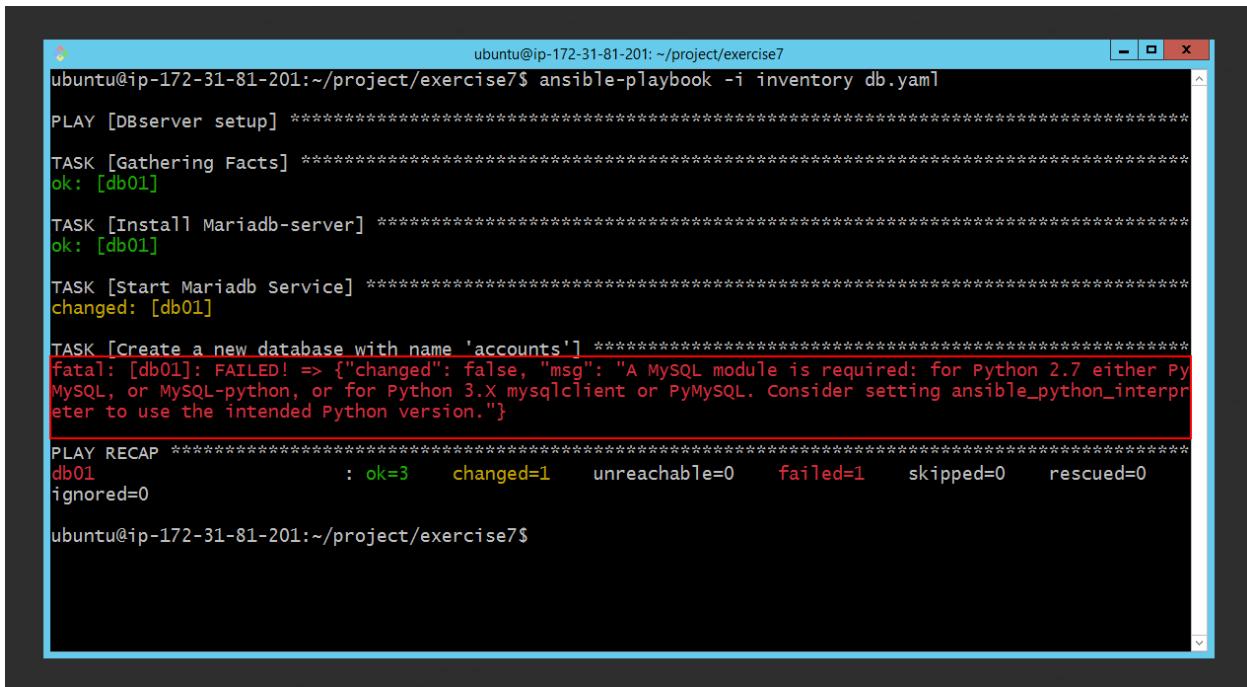
    - name: Create a new database with name 'accounts'
      mysql_db:
        name: accounts
        state: present
```

57

Instructor: Fariborz Fallahzadeh

Email Address: fariborz.fallahzadeh@gmail.com

At this stage, we run the Playbook. As you can see, during execution, it throws an error. The reason for this is that most Ansible modules require certain dependencies before they can be used, and most of them depend on Python.



```
ubuntu@ip-172-31-81-201:~/project/exercise7$ ansible-playbook -i inventory db.yaml
PLAY [DBserver setup] ****
TASK [Gathering Facts] ****
ok: [db01]

TASK [Install Mariadb-server] ****
ok: [db01]

TASK [Start Mariadb Service] ****
changed: [db01]

TASK [Create a new database with name 'accounts'] ****
fatal: [db01]: FAILED! => {"changed": false, "msg": "A MySQL module is required: for Python 2.7 either PyMySQL, or MySQL-python, or for Python 3.x mysqlclient or PyMySQL. Consider setting ansible_python_interpreter to use the intended Python version."}

PLAY RECAP ****
db01 : ok=3    changed=1    unreachable=0    failed=1    skipped=0    rescued=0    ignored=0

ubuntu@ip-172-31-81-201:~/project/exercise7$
```

Now, why did this happen during the execution of the Playbook?

We know that Ansible creates a Python script, which is uploaded to the DB server and then executed by the DB server. Some modules require specific dependencies before execution. For example, the MySQL module requires Python 2.7, and Python 2.7 needs the **PyMySQL** or **MySQL-python** module to connect to the database.

Before using a module in Ansible, always check the **Requirements** section of the module you want to use.

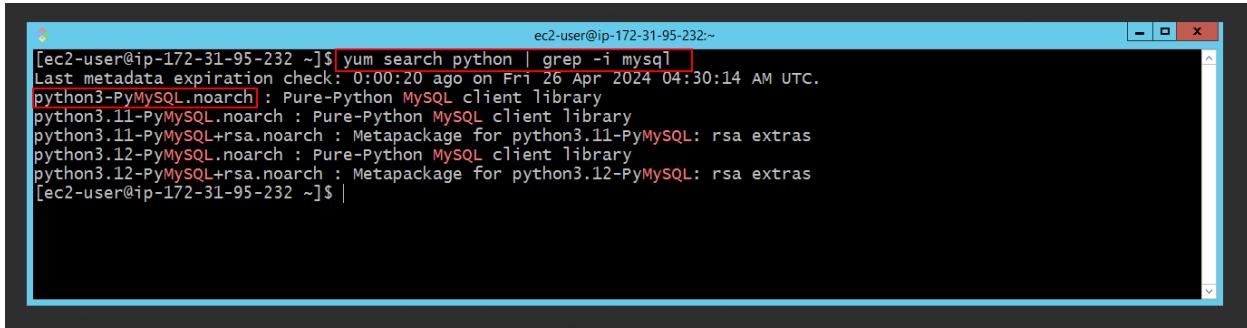
The screenshot shows the Ansible documentation website at docs.ansible.com/ansible/2.9/modules/mysql_db_module.html#mysql-db-module. The page title is "mysql_db – Add or remove MySQL databases from a remote host". The left sidebar contains links to various Ansible documentation sections like Installation, Using Ansible, Contributing, and Reference & Appendices. The main content area has a "Synopsis" section with bullet points, a "Requirements" section with a red border containing a note about dependencies, and a "Parameters" section with a table. The "Requirements" section notes that MySQLdb (Python 2.x), PyMySQL (Python 2.7 and Python 3.x), or mysql (command line binary) are required. The "Parameters" table includes a row for "ca_cert" with a comment: "The path to a Certificate Authority (CA) certificate. This option, if used, must specify the same certificate as used by the server."

To install the correct module on the DB server, first SSH into the DB server of the scenario.

The screenshot shows a terminal window on an Ubuntu system. The user has run the command `cat inventory`, which displays an Ansible inventory file. The file defines groups like "all", "web01", "web02", "db01", and "webservers", "dbservers", "dc_group", and "vars". It specifies host IP addresses and ansible_user. The user then runs the command `ssh -i clientkey.pem ec2-user@172.31.95.232` to connect to the database server.

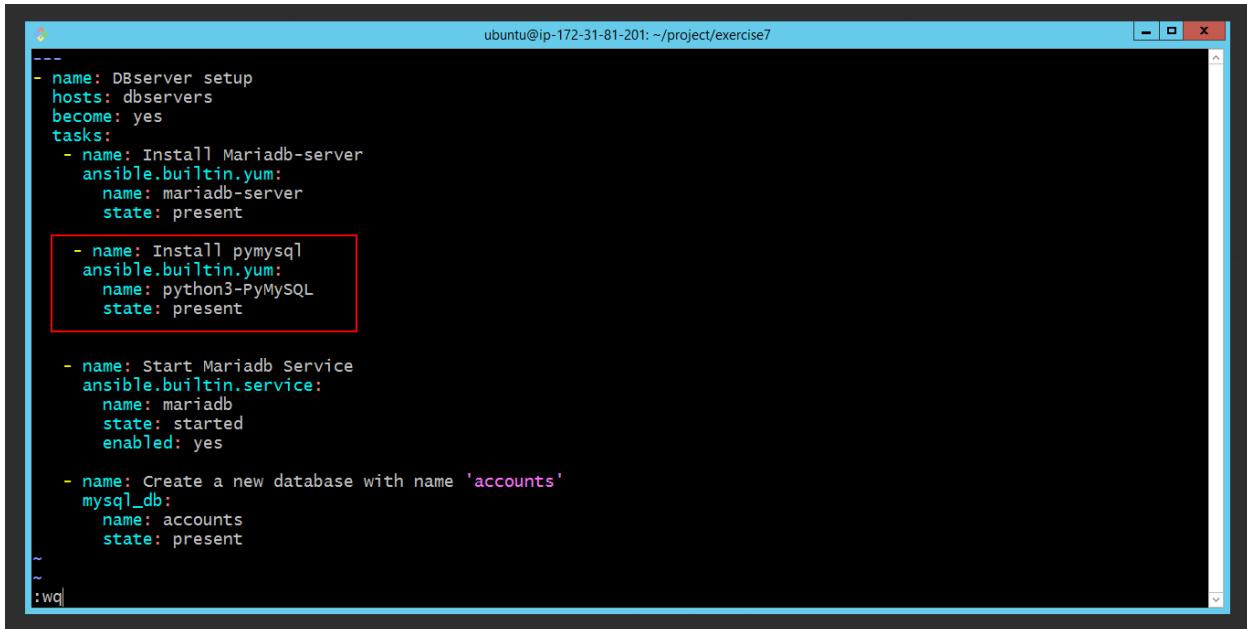
```
ubuntu@ip-172-31-81-201:~/project/exercise7$ cat inventory
all:
  hosts:
    web01:
      ansible_host: 172.31.95.31
    web02:
      ansible_host: 172.31.83.31
    db01:
      ansible_host: 172.31.95.232
  children:
    webservers:
      hosts:
        web01:
        web02:
    dbservers:
      hosts:
        db01:
    dc_group:
      children:
        webservers:
        dbservers:
  vars:
    ansible_user: ec2-user
    ansible_ssh_private_key_file: clientkey.pem
ubuntu@ip-172-31-81-201:~/project/exercise7$ ssh -i clientkey.pem ec2-user@172.31.95.232
```

At this stage, use the following command to search for the required dependency to connect to the MySQL database in the Linux repository.



```
ec2-user@ip-172-31-95-232 ~]$ yum search python | grep -i mysql
Last metadata expiration check: 0:00:20 ago on Fri 26 Apr 2024 04:30:14 AM UTC.
python3-PyMySQL.noarch : Pure-Python MySQL client library
python3.11-PyMySQL.noarch : Pure-Python MySQL client library
python3.11-PyMySQL+rsa.noarch : Metapackage for python3.11-PyMySQL: rsa extras
python3.12-PyMySQL.noarch : Pure-Python MySQL client library
python3.12-PyMySQL+rsa.noarch : Metapackage for python3.12-PyMySQL: rsa extras
[ec2-user@ip-172-31-95-232 ~]$
```

Then, in the Playbook file, use the **yum** module to add the Python package, so it can be installed on the DB server.



```
---
- name: DBserver setup
  hosts: dbservers
  become: yes
  tasks:
    - name: Install Mariadb-server
      ansible.builtin.yum:
        name: mariadb-server
        state: present

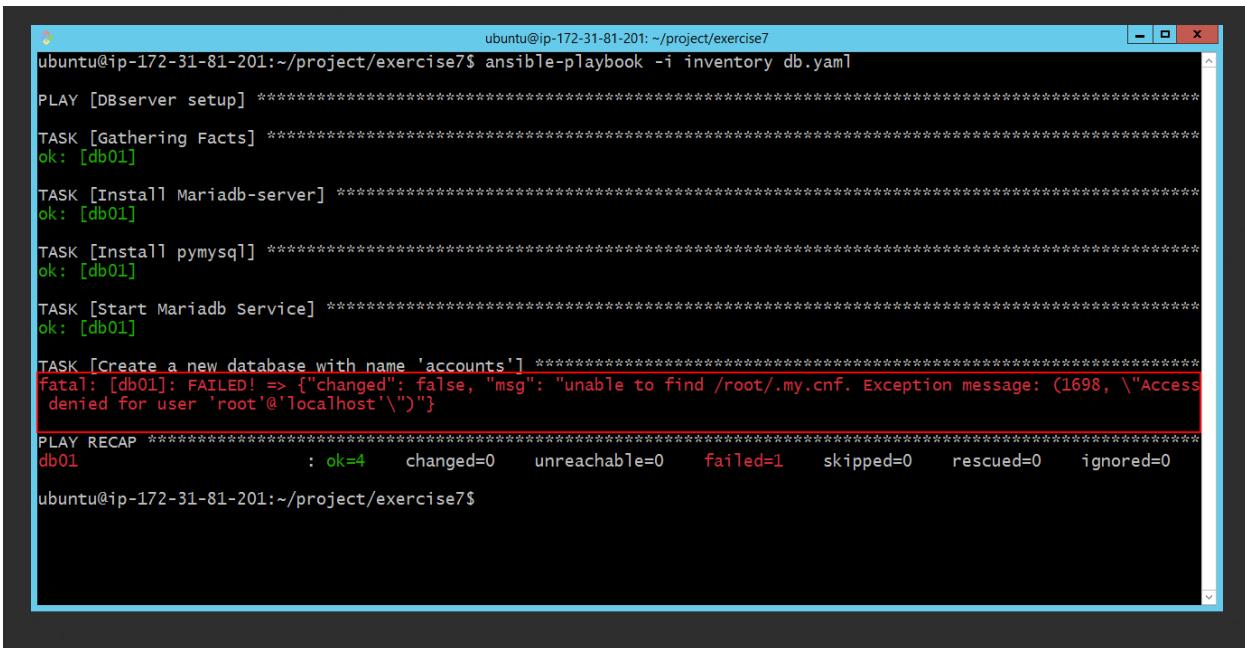
    - name: Install pymysql
      ansible.builtin.yum:
        name: python3-PyMySQL
        state: present

    - name: Start Mariadb Service
      ansible.builtin.service:
        name: mariadb
        state: started
        enabled: yes

    - name: Create a new database with name 'accounts'
      mysql_db:
        name: accounts
        state: present

~ :wq|
```

We test the Playbook again.



The screenshot shows a terminal window on an Ubuntu system. The command run is `ansible-playbook -i inventory db.yaml`. The output shows the execution of various tasks: DBserver setup, Gathering Facts, Install Mariadb-server, Install pymysql, Start Mariadb Service, and Create a new database with name 'accounts'. The 'Create a new database with name 'accounts'' task fails with the error: `Fatal: [db01]: FAILED! => {"changed": false, "msg": "unable to find /root/.my.cnf. Exception message: (1698, \"Access denied for user 'root' @'localhost'\\\")"}`. The final PLAY RECAP summary is: db01 : ok=4 changed=0 unreachable=0 failed=1 skipped=0 rescued=0 ignored=0.

```
ubuntu@ip-172-31-81-201:~/project/exercise7$ ansible-playbook -i inventory db.yaml
PLAY [DBserver setup] ****
TASK [Gathering Facts] ****
ok: [db01]

TASK [Install Mariadb-server] ****
ok: [db01]

TASK [Install pymysql] ****
ok: [db01]

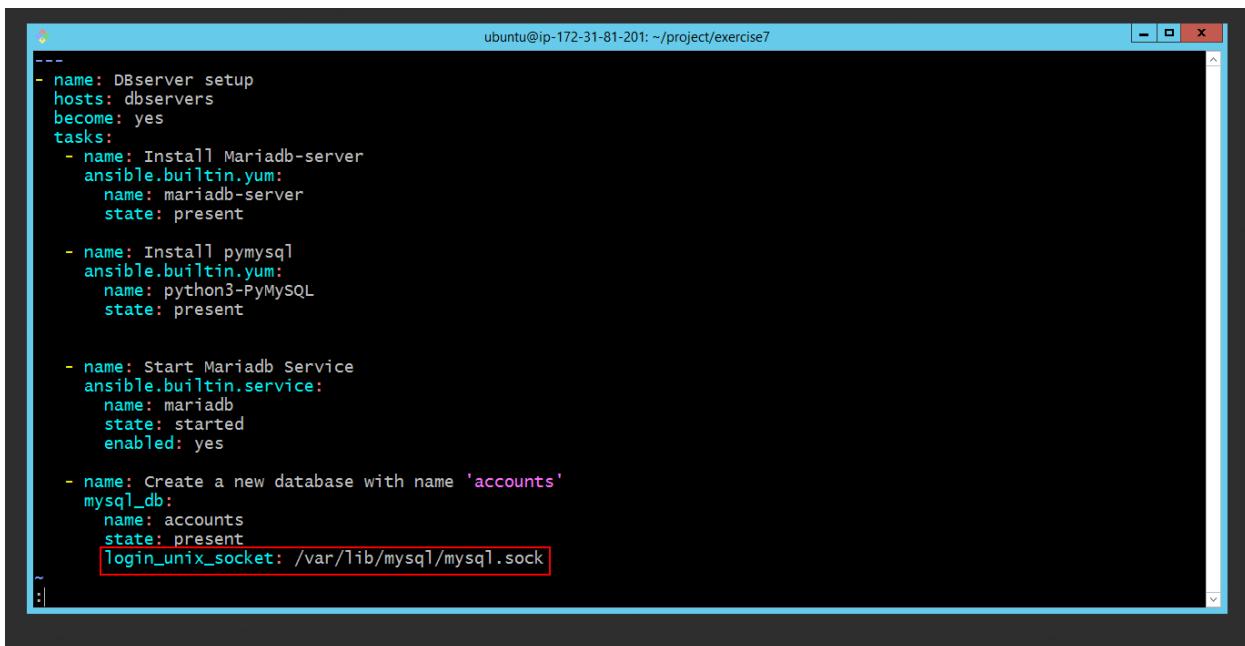
TASK [Start Mariadb Service] ****
ok: [db01]

TASK [Create a new database with name 'accounts'] ****
fatal: [db01]: FAILED! => {"changed": false, "msg": "unable to find /root/.my.cnf. Exception message: (1698, \"Access denied for user 'root' @'localhost'\\\")"}

PLAY RECAP ****
db01 : ok=4    changed=0   unreachable=0   failed=1    skipped=0   rescued=0   ignored=0

ubuntu@ip-172-31-81-201:~/project/exercise7$
```

As shown in the image above, after executing the Playbook, we encountered a new error. The reason for this is that the file socket path in the CentOS operating system is located elsewhere, and this needs to be corrected in the Playbook file.



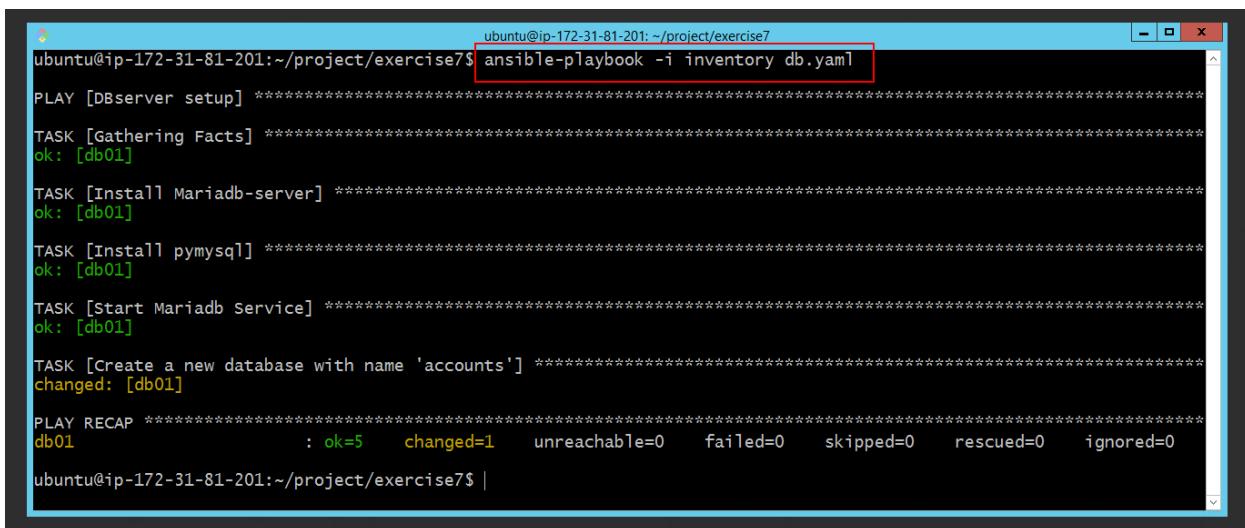
```
ubuntu@ip-172-31-81-201: ~/project/exercise7
---
- name: DBserver setup
  hosts: dbservers
  become: yes
  tasks:
    - name: Install Mariadb-server
      ansible.builtin.yum:
        name: mariadb-server
        state: present

    - name: Install pymysql
      ansible.builtin.yum:
        name: python3-PyMySQL
        state: present

    - name: Start Mariadb service
      ansible.builtin.service:
        name: mariadb
        state: started
        enabled: yes

    - name: Create a new database with name 'accounts'
      mysql_db:
        name: accounts
        state: present
        login_unix_socket: /var/lib/mysql/mysql.sock
```

We run the Playbook again. As shown in the image below, this time it runs without any errors.



```
ubuntu@ip-172-31-81-201:~/project/exercise7$ ansible-playbook -i inventory db.yaml
PLAY [DBserver setup] ****
TASK [Gathering Facts] ****
ok: [db01]
TASK [Install Mariadb-server] ****
ok: [db01]
TASK [Install pymysql] ****
ok: [db01]
TASK [Start Mariadb Service] ****
ok: [db01]
TASK [Create a new database with name 'accounts'] ****
changed: [db01]
PLAY RECAP ****
db01 : ok=5    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
ubuntu@ip-172-31-81-201:~/project/exercise7$ |
```

To find errors related to a module and how to resolve them, we can refer to the community discussions or forums related to that module.

For example, simply search for the following phrase on Google and click on the **Community MySQL** link.

A screenshot of a Google search results page. The search query is "mysql ansible module". The first result is a link to the Ansible Documentation for the MySQL collection, titled "Community MySQL". This link is highlighted with a red box. Below the link, there is a snippet of text describing the collection's purpose and some of its modules. Other search results for MySQL modules like mysql_db, mysql_info, mysql_query, mysql_role, mysql_user, and mysql_variables are listed below.

Then, click on the link for the specific module you are interested in.

A screenshot of the Ansible Community Documentation for the MySQL collection. The left sidebar shows various collection categories. The main content area is titled "MySQL collection for Ansible". It includes sections for "Author" (Ansible community), "Supported ansible-core versions" (2.9.10 or newer), and a "Plugin Index". The "Plugin Index" section lists several modules under the "community.mysql" namespace, with the "mysql_db" module being highlighted with a red box. Below the plugin index, there is a "See also" section listing other collections.

As seen in the **Community Module** section, here you can find examples of using the module without errors, exactly like the one we created in the Playbook.

The screenshot shows the Ansible Community Documentation page for the mysql_db module. The left sidebar contains links to various Ansible documentation sections. The main content area has three reference sections: mysqldump reference, CREATE DATABASE reference, and DROP DATABASE reference. Below these is an 'Examples' section containing several Ansible playbooks. One example is highlighted with a red box:

```
# If you encounter the "Please explicitly state intended protocol" error,
# use the login_unix_socket argument
- name: Create new database with name 'bobdata'
  community.mysql.mysql_db:
    name: bobdata
    state: present
    login_unix_socket: /run/mysqld/mysqld.sock

# Create new databases with names 'foo' and 'bar'
- name: Create new databases with names 'foo' and 'bar'
  community.mysql.mysql_db:
    name:
      - foo
      - bar
    state: present

# Copy database dump file to remote host and restore it to database 'my_db'
- name: Copy database dump file
  copy:
    src: dump.sql.bz2
    dest: /tmp

- name: Restore database
  community.mysql.mysql_db:
    name: my_db
    state: import
    target: /tmp/dump.sql.bz2

- name: Restore database ignoring errors
  community.mysql.mysql_db:
    name: my_db
    state: import
    target: /tmp/dump.sql.bz2
    force: true
```