1. Ansible Lab 1 - Installation and Inventory file basics

Every beginning is difficult, and I hope you made it with the installations. The first lab is intended to set up the vagrant infrastructure you'll need to learn Ansible.

As shown in the image above, our Ansible control will come from ansible-control through which we will be able to ssh and orchestrate the other machines.

1.1. Create the Vagrantfile

In this document, we will go slowly and create all the files and configurations in order. We will start with the Vagrant file. Create file named Vagrantfile with:

touch Vagrantfile

Text

Description automatically generated

I init the vagrant so that a vagrant file can be created and than changed the content with the given one.

Then edit with your favorite editor such as vim, nano etc. The code shown below is in the Vagrantfile and is intended to provision the virtual machines we will need for these labs.

Text

Description automatically generated

This Vagrantfile defines an array of servers with their respective hostname, IP address, and SSH port. It then loops through each machine in the array and defines a Vagrant machine with the specified hostname, box, and network settings.

The network settings for each machine include a public network interface.

1.2. Create VMs using vagrant and ssh to our control server

The following commands will setup Vagrant and connect to ansible-control server

**vagrant up --provider virtualbox**

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**vagrant ssh ansible-control**

Graphical user interface, text, application

Description automatically generated

1.3. Create the /vagrant/hosts\_file

With the executed commands, create the /vagrant/hosts\_file file:

**nano /vagrant/hosts\_file**

Import the following content and save the file:

192.168.7.101 loadbalancer

192.168.7.102 db01

192.168.7.103 web01

192.168.7.104 web02

192.168.7.105 ansible-control

1.4. Copy /vagrant/hosts\_file to /etc/hosts

The below command will copy hosts file on ansible-control

**sudo cp /vagrant/hosts\_file /etc/hosts**

Graphical user interface, text

Description automatically generated

1.5. Install Ansible

Since we are now connected to the ansible-control machine we can install Ansible with:

**sudo apt update**

**sudo apt install ansible -y**

Graphical user interface, text

Description automatically generated

1.6. Create an inventory file named hosts

Create the directory structure for the labs with the command:

**mkdir ansible-lab{1..5}**

Create the hosts file with the command:

nano /vagrant/ansible-lab1/hosts

add the following content:

[control]

ansible-control

[proxy]

loadbalancer

[webservers]

web01

web02

[database]

db01

[webstack:children]

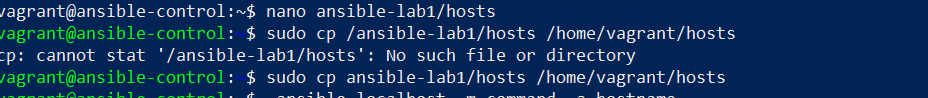
proxy

webservers

database

With this command you will copy the pre-defined hosts file:

**sudo cp /vagrant/ansible-lab1/hosts /home/vagrant/hosts**

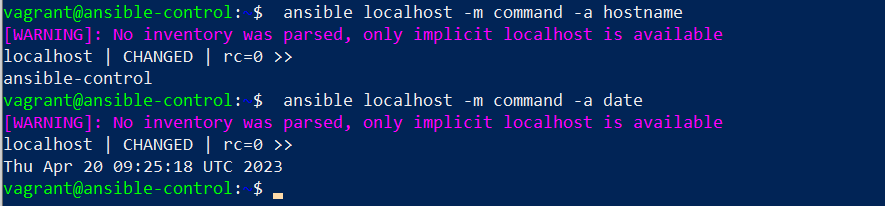


1.7. Test out a command.

Now it is time to test if the setup works. You can do that we the following ad-hoc commands:

**ansible localhost -m command -a hostname**

**ansible localhost -m command -a date**



1.8. Generate SSH Keys and copy to hosts

Run the below commands to create a SSH key and copy to all servers

**ssh-keygen**

**ssh-copy-id localhost**

**ssh-copy-id web01 && ssh-copy-id web02 && ssh-copy-id loadbalancer && ssh-copy-id db01**

*NOTE:* Type yes to confirm and If it prompt for password it is the SSH pass which is: vagrant

Graphical user interface, text

Description automatically generated

Text

Description automatically generated

1.9. Test running ad-hoc commands to all hosts

Now we can run an ad-hoc command to the webstack group that we created and set

ansible webstack -i /home/vagrant/hosts -m command -a hostname

Text

Description automatically generated

You just set up a hostfile, installed ansible on ansible-control, generated an ssh key, and copied it to all virtual machines. Well done, your lab infrastructure is ready now.

2. Ansible Lab 2 - Ad HOC tasks and Modules

This lab is intended to introduce ad hoc tasks and modules

First you can sync the current ansible-lab state to the newly create lab structure:

**rsync -WaP /vagrant/ansible-lab1/ /vagrant/ansible-lab2/**

**cd /vagrant/ansible-lab2/**

Graphical user interface, text, application

Description automatically generated

and move to the second lab directory.

An Ansible ad hoc command uses the /usr/bin/ansible command-line tool to automate a single task on one or more managed nodes. ad hoc commands are quick and easy, but they are not reusable. So why learn about ad hoc commands? ad hoc commands demonstrate the simplicity and power of Ansible. The concepts you learn here will port over directly to the playbook language.

2.1. Ansible ad hoc command for checking the uptime of the hosts

In this example, we are going to know the uptime of the hosts. Ansible provides two major modules to run the command over the host group or on the remote server.

Which one to pick is not a big confusion if you know what they and their capabilities are. Here are the commands you can use to get the uptime. All three commands would yield you the same results.

**ansible all -i /home/vagrant/hosts -m command -a uptime**

**ansible all -i /home/vagrant/hosts -m shell -a uptime**

**ansible all -i /home/vagrant/hosts -a uptime**

as you could have already figured out -m is the module and -a should contain the command it should run which goes as an argument to command and shell.

Text

Description automatically generated

2.2. Check the free memory or memory usage of hosts using ansible ad hoc command.

The following ansible ad hoc command would help you get the free memory of all the hosts in the host groups.

As you can see we are running the free -m command on the remote hosts and collecting the information.

**ansible all -a "free -m" -i /home/vagrant/hosts**

Graphical user interface

Description automatically generated

2.3. Update and upgrade all machines

Now that you're comfortable with a few ad hoc commands, you can move on to updating and upgrading your infrastructure. Because it's always good to be on the latest and greatest versions.

**ansible all -i /home/vagrant/hosts -m command -a 'sudo apt update'**

**ansible all -i /home/vagrant/hosts -m command -a 'sudo apt upgrade -y'**

Text

Description automatically generated

2.4. Use APT module to install services

In the next example, we will use the APT module. With it, we will install an Apache web server in the webservers group and a MySQL server in the database group.

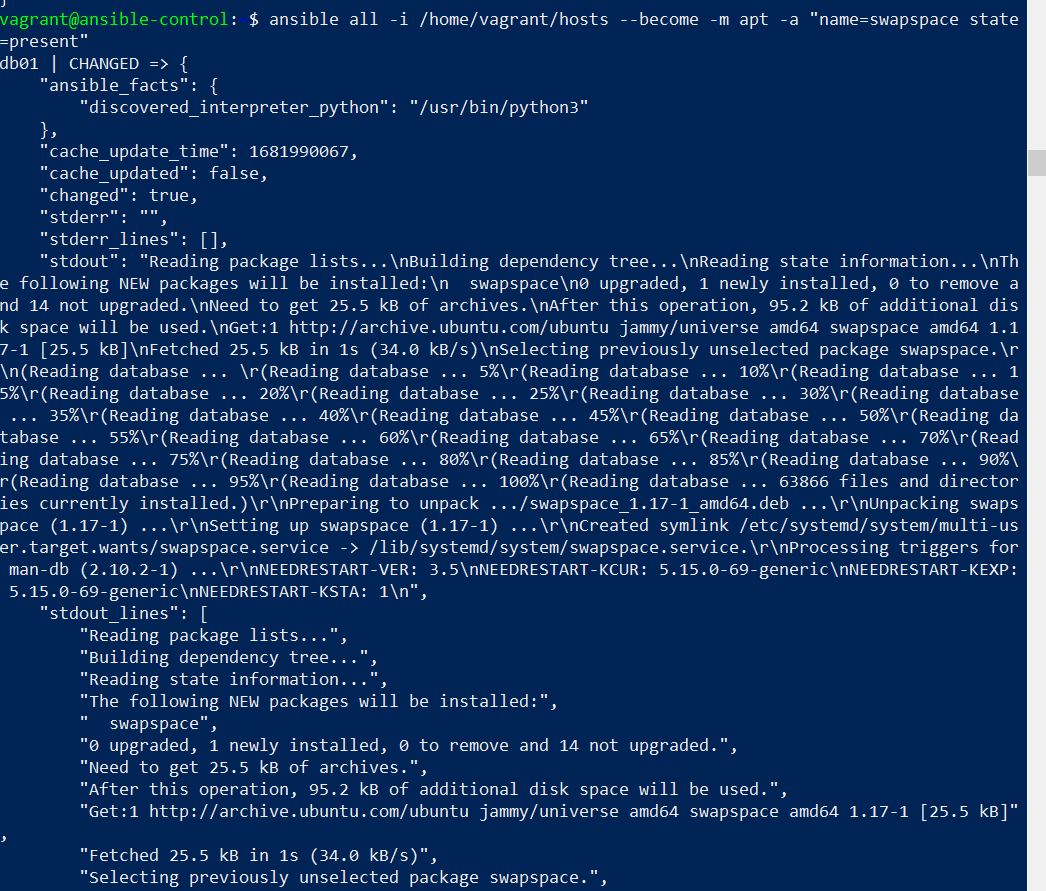
The important thing to note is that the packages will only be installed in the specific group that is specified in the command itself. If all is used, then that command will be executed on all defined machines.

**ansible all -i /home/vagrant/hosts --become -m apt -a "update\_cache=yes"**

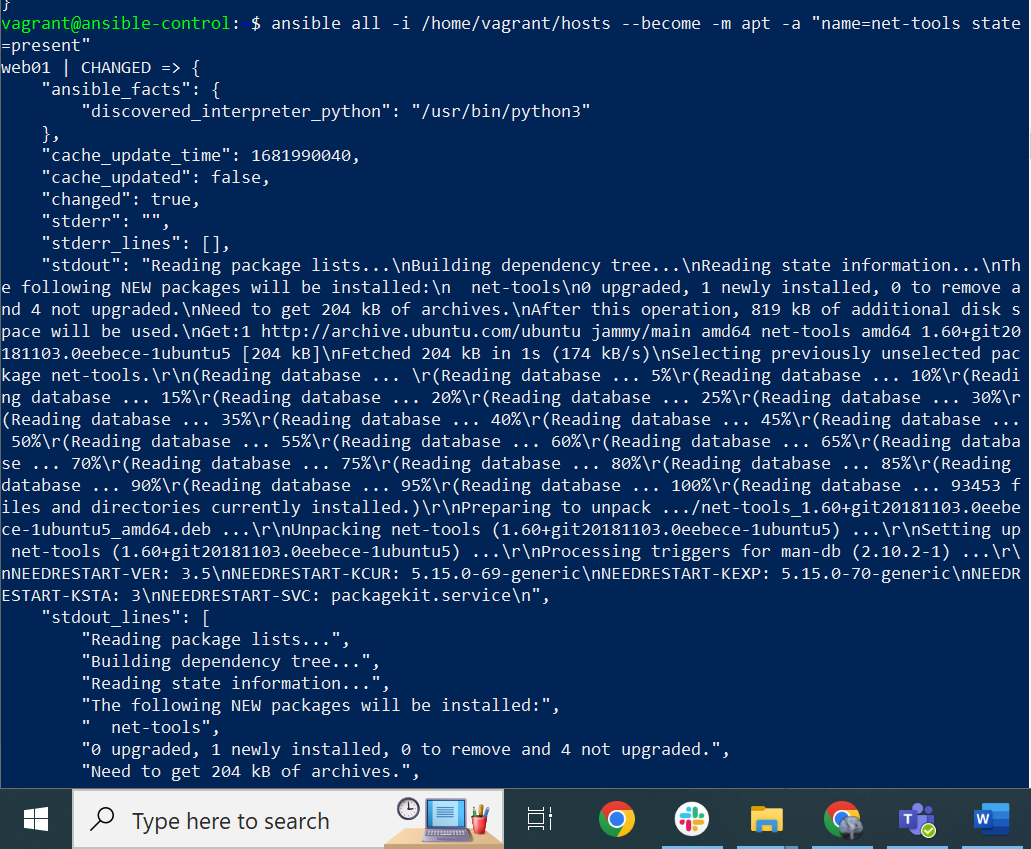
Text

Description automatically generated

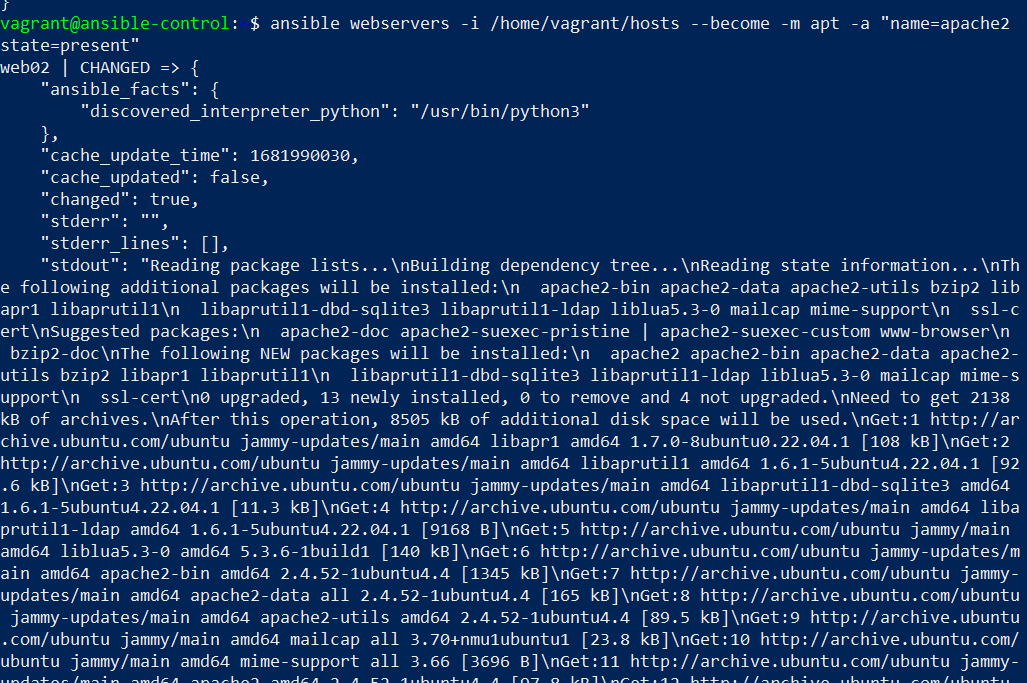
**ansible all -i /home/vagrant/hosts --become -m apt -a "name=swapspace state=present"**



**ansible all -i /home/vagrant/hosts --become -m apt -a "name=net-tools state=present"**



**ansible webservers -i /home/vagrant/hosts --become -m apt -a "name=apache2 state=present"**



**ansible database -i /home/vagrant/hosts --become -m apt -a "name=mysql-server state=present"**

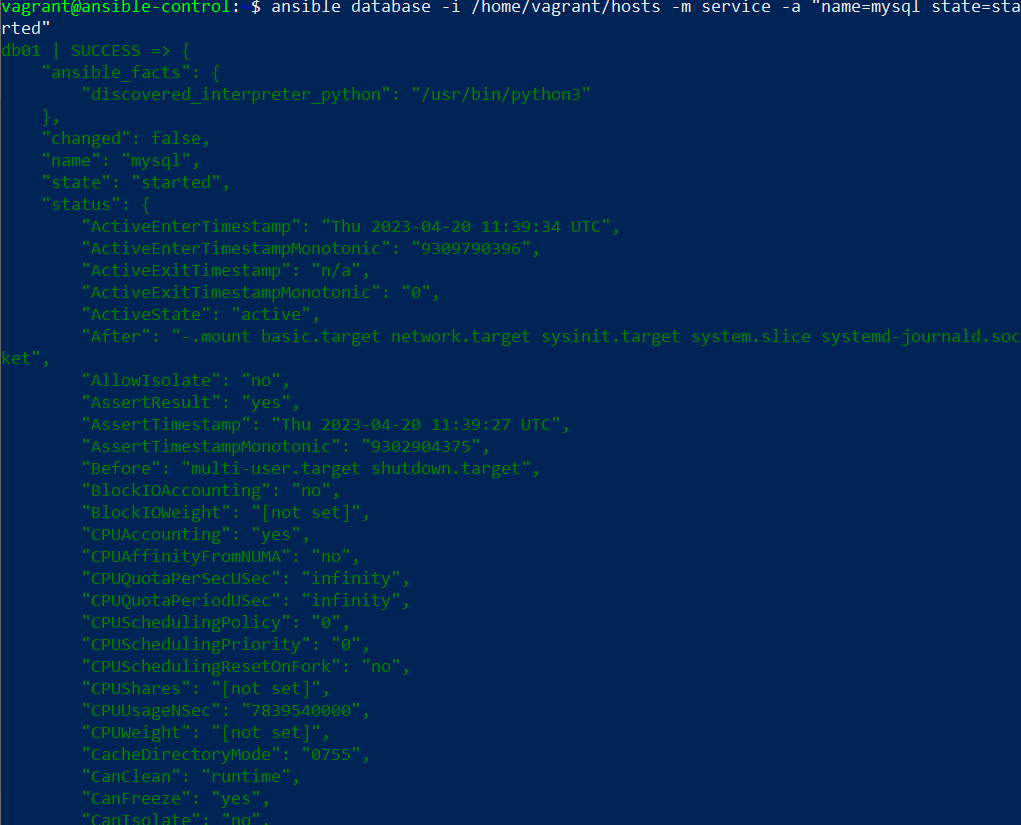
Text

Description automatically generated

2.5. Use service module to manage services

The service module is quite useful for checking the status or restarting all services. In the next two examples, you will learn how to check if your MySQL server is active and running and then restart it.

**ansible database -i /home/vagrant/hosts -m service -a "name=mysql state=started"**



**ansible database --become -i /home/vagrant/hosts -m service -a "name=mysql state=restarted"**

Text

Description automatically generated

2.6. Use ansible to reboot webstack

By executing the following ad hoc command, you will restart the virtual machines from the webstack group.

**ansible webstack -i /home/vagrant/hosts --become -a "reboot --reboot"**

Text

Description automatically generated

It is quite normal to receive a message "Failed to connect to the host via ssh: System is going down" because after executing the command our connection is interrupted because the machine itself is restarted.

These few examples were just an introduction to what can be done with Ansible and ad hoc commands.

Ansible Lab 3 - Playbooks, Templates and Handlers

Now, sync the current ansible-lab state to the newly create lab structure:

rsync -WaP /vagrant/ansible-lab2/ /vagrant/ansible-lab3/

cd /vagrant/ansible-lab3/

and move to the third lab directory.

In this lab, we will define playbook named playbook1.yml that is intended for installing the Apache2 web server. With this playbook, we will try several scenarios to see how a YAML file is structured in which variables, tasks, and handlers are defined. Jinja templates (web template engine for Python) are used, which we will copy to your web01 and web02, that is, to the group of webservers.

Text

Description automatically generated

1. Exam the playbook and look over the details of the YAML file

Navigate to ansible-lab3 with the following command:

**cd /vagrant/ansible-lab3/**

Create the playbook with:

**nano playbook1.yml**

Add the following:

---

# This playbook is written in YAML and is used to automate the deployment of an Apache web server on a group of hosts. The playbook does the following:

# The hosts section specifies the group of hosts on which the playbook will be executed. In this case, it is set to "webservers", meaning the playbook will be executed on all the hosts in the "webservers" group.

# The become section is set to "yes", meaning that the playbook will run with administrative privileges.

# The vars section defines three variables: http\_port, https\_port, and html\_welcome\_msg. These variables can be used in the playbook to configure the Apache web server.

- hosts: webservers

become: yes

vars:

http\_port: 8000

https\_port: 4443

html\_welcome\_msg: "Hello Scalefocus Academy!"

# The tasks section contains a list of tasks that the playbook will execute.

# The first task "ensure apache is at the latest version" uses the apt module to update the Apache2 package to the latest version.

tasks:

- name: ensure apache is at the latest version

apt:

name: apache2

state: latest

# The second task "write the apache2 ports.conf config file" uses the template module to generate a configuration file for Apache. This configuration file (ports.conf) will specify the ports that Apache should listen on. The template for the file is stored in the "templates" directory and has a .j2 file extension, indicating that it is a Jinja2 template. The template file will be written to the "/etc/apache2/ports.conf" location on the target host.

- name: write the apache2 ports.conf config file

template:

src: templates/ports.conf.j2

dest: /etc/apache2/ports.conf

notify:

- restart apache

# The third task "write a basic index.html file" uses the template module to generate a basic HTML file that will be served by the Apache web server. The template for the file is stored in the "templates" directory and has a .j2 file extension. The template file will be written to the "/var/www/html/index.html" location on the target host.

- name: write a basic index.html file

template:

src: templates/index.html.j2

dest: /var/www/html/index.html

notify:

- restart apache

# The fourth task "ensure apache is running" uses the service module to start the Apache2 service if it is not already running.

- name: ensure apache is running

service:

name: apache2

state: started

# The handlers section contains a single handler "restart apache" that uses the service module to restart the Apache2 service. This handler is notified by the second and third tasks to restart the Apache service whenever a change is made to the ports.conf or index.html files.

handlers:

- name: restart apache

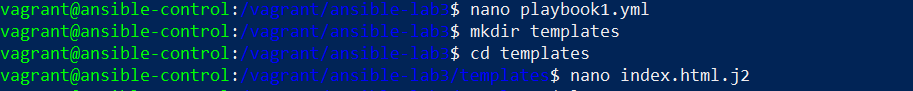
service:

name: apache2

state: restarted

listen: "restart apache"

As you can see, there are comments left on the playbook itself explaining all the components and tasks in the playbook.



2. Check the Templates

Two Jinja templates have been added to the playbook1.yml file, which you should create abd look at before running this playbook.

**mkdir /vagrant/ansible-lab3/templates**

**touch /vagrant/ansible-lab3/templates/index.html.j2**

**touch /vagrant/ansible-lab3/templates/ports.conf.j2**

creat and edit the index.html.j2

**nano /vagrant/ansible-lab3/templates/index.html.j2**

add the following HTML content:

<html>

<h1>{{ html\_welcome\_msg }} You have reached the {{ansible\_facts['nodename']}} server.</h1>

</html>

creat and edit the index.html.j2

**nano /vagrant/ansible-lab3/templates/ports.conf.j2**

add the following Apache2 configuration file:

# If you just change the port or add more ports here, you will likely also

# have to change the VirtualHost statement in

# /etc/apache2/sites-enabled/000-default.conf

Listen {{ http\_port }}

<IfModule ssl\_module>

Listen {{ https\_port }}

</IfModule>

<IfModule mod\_gnutls.c>

Listen {{ https\_port }}

</IfModule>

# vim: syntax=apache ts=4 sw=4 sts=4 sr noet

The index.html.j2 is a simple HTML file with a message defined in the playbook. The ports.conf.j2 template is intended as a configuration file for your new Apache servers. That is, it defines the HTTP and HTTPS ports of this service.





3. Run the playbook

Now that we know what the playbook does, it's time to run it. This time we will use the ansible-playbook instead of ansible as before.

**ansible-playbook -i /home/vagrant/hosts playbook1.yml**

Table

Description automatically generated with low confidence

4. Test connectivity to servers

In this situation, it is best to use the curl command to check if your newly installed Apache servers are available and responsive with the templates you added.

curl web01:8000

curl web02:8000

Graphical user interface, text

Description automatically generated

In summary, this playbook will deploy an Apache web server on a group of hosts and configure it to listen on the specified ports and serve a basic HTML welcome message.

Ansible Lab 4 - Re-usable playbooks, import\_tasks, Roles and Ansible Galaxy

Now, sync the current ansible-lab state to the newly create lab structure:

**rsync -WaP /vagrant/ansible-lab3/ /vagrant/ansible-lab4/**

**cd /vagrant/ansible-lab4/**

and move to the fourth lab directory.

Graphical user interface, text, application

Description automatically generated

We are going to take the playbook that we are working on and sort of enhance it and make it a lot leaner. We can see that we're getting a lot of code in it and we only have our web servers set up yet.

1. Re-usable playbooks, import\_tasks

Therefore following the best practices we will rewrite the code from our defined playbook1.yml from ansible-lab3 by doing the following:

1. We will move tasks to a tasks subfolder

Create new directory tasks and new yml file apache2\_install. Then edit the playbook1.yml and move the tasks code to the newly created file.

**mkdir tasks ; touch tasks/apache2\_install.yml**

1. Move handler to handler subfolder

Do the same as the tasks.

mkdir handlers ; touch handlers/main.yml

After we move around our code, we will use the import\_tasks with the path of our new files. It should looks like this:

- hosts: webservers

become: yes

vars:

http\_port: 8000

https\_port: 4443

html\_welcome\_msg: "Hello Scalefocus Academy!"

tasks:

- import\_tasks: tasks/apache2\_install.yml

handlers:

- import\_tasks: handlers/main.yml

1. Run the playbook (optional for a test)

ansible-playbook -i hosts -K playbook1.yml

If everything was green, then we can move on to one more topic in this lab, Ansible Roles and Ansible Galaxy.

A picture containing timeline

Description automatically generated

2. Ansible Roles and Ansible Galaxy

Ansible Roles are a way to organize your playbooks and tasks into reusable units. A role is essentially a collection of related tasks, files, templates, and variables that can be easily shared and used across multiple playbooks.

Roles are useful for several reasons:

1. Reusability: You can use roles in multiple playbooks and projects, which saves time and effort.
2. Modularity: Roles make it easy to break down complex playbooks into smaller, more manageable units.
3. Organization: Roles provide a structured way to organize your playbooks and tasks, which makes it easier to understand and maintain them.
4. Collaboration: Roles can be shared with other teams or the community, which fosters collaboration and sharing of best practices.

In short, Ansible Roles provide a way to organize and reuse your Ansible code, making it more efficient, modular, and maintainable.

[ansible-galaxy](https://galaxy.ansible.com/) is a repository of reusable Ansible roles and collections that are contributed by the community. It provides a central location where Ansible users can search for, download, and use existing roles, rather than starting from scratch.

Using Ansible Galaxy is straightforward. You can search for roles and collections on the Ansible Galaxy website, or you can use the ansible-galaxy command-line tool to search, download, and install roles and collections.

Overall, Ansible Galaxy provides a convenient and efficient way to find, share, and use Ansible roles and collections, making it an essential resource for any Ansible user.

Since our current status is to use imported tasks, the next step is to familiarize ourselves with the roles. The current tree in our directory on this point is the following:

Our goal is to change our playbook1.yml so it uses the Role functionality to setup apache2. We will proceed with the following:

1. Use ansible-galaxy to create Apache2 webserver role scaffolding

**cd /vagrant/ansible-lab4/**

**ansible-galaxy init roles/apache2**

Graphical user interface, text

Description automatically generated

1. Move the tasks to the roles\webserver folder

**mv tasks/apache2\_install.yml roles/apache2/tasks/**

**mv handlers/main.yml roles/apache2/handlers/main.yml**

**mv templates/ roles/apache2/**

**rmdir tasks/ handlers/**

Graphical user interface, text

Description automatically generated

edit the roles/apache2/tasks/main.yml and add the following:

---

# tasks file for roles/webservers

- include: apache2\_install.yml

there is one more thing to do and that is to edit the current state of our playbook1.yml. We will change the task to roles and the playbook should look like this:

- hosts: webservers

become: yes

vars:

http\_port: 8000

https\_port: 4443

html\_welcome\_msg: "Hello Scalefocus Academy!"

roles:

- apache2

So far we created Apache2 Role, and move tasks/handlers/templates to new roles/apache2 structure. Modified the roles/apache2/tasks/main.yml and playbook1.yml. After these changes, our directory structure will look like this:

1. Run the playbook

**ansible-playbook -i hosts -K playbook1.yml**

A picture containing timeline

Description automatically generated

We didn’t edit apache\_install.yml so it’s empty.

Next, let's create another role with name 'common' and add it to 'web', 'loadbalancer' and 'database' servers.

1. Use ansible-galaxy to create 'common' and nginx role scaffolding

ansible-galaxy init roles/common

ansible-galaxy init roles/nginx

1. Setup 'common' role tasks/main.yml and tasks/install\_tools.yml

Let's define the task for our new 'common' role:

**nano roles/common/tasks/install\_tools.yml**

add the following:

- name: "Install Common packages"

apt: name={{ item }} state=latest

with\_items:

- net-tools

- tree

- python3-pip

- name: Install pymysql python package

pip:

name: pymysql

after the creation of this task, we will add an include in the main.yml task:

**nano roles/common/tasks/main.yml**

and enter the include:

---

# tasks file for roles/common

- include: install\_tools.yml

1. Setup the nginx role

Let's create the tasks files:

**nano roles/nginx/tasks/install\_packages.yml**

import the following and save the file:

- name: "Install Nginx packages"

apt:

name: nginx

state: present

another task to configure the Nginx:

nano roles/nginx/tasks/configure\_nginx.yml

add the content:

- name: Deploy Nginx sites configuration

template: src=mysite.j2 dest="/etc/nginx/sites-enabled/mysite"

notify: restart nginx

- name: Remove defaults

file: path="/etc/nginx/sites-enabled/default" state=absent

and let's include the new tasks that we just created into the main.yml:

**nano roles/nginx/tasks/main.yml**

add the following:

---

# tasks file for roles/nginx

- include: install\_packages.yml

- include: configure\_nginx.yml

Save.

Next is to configure the template and the handler:

**mkdir roles/nginx/templates**

**nano roles/nginx/templates/mysite.j2**

paste the following:

upstream webservers {

server 192.168.7.103:8000;

server 192.168.7.104:8000;

}

server {

listen 80;

location / {

proxy\_pass http://webservers;

}

}

Let's configure the Nginx handler so we can try what we've done.

**nano roles/nginx/handlers/main.yml**

add the following code that will restart the Nginx:

---

# handlers file for roles/nginx

- name: restart nginx

service: name=nginx state=restarted

Finally, let's update the playbook1.yml file:

# This playbook consists of two plays.

# The first play targets the hosts with the tag "webservers".

- hosts: webservers

become: yes

vars:

http\_port: 8000

https\_port: 4443

html\_welcome\_msg: "Hello Scalefocus Academy!"

roles:

- common

- apache2

# The second play targets the hosts with the tag "proxy".

- hosts: proxy

become: yes

roles:

- common

- nginx

Huh done. For now, we've configured the new roles and before we try our updated two-part playbook, our directory structure looks like this:

Graphical user interface, text

Description automatically generated

1. Run the playbook and test

**ansible-playbook -i hosts -K playbook1.yml**

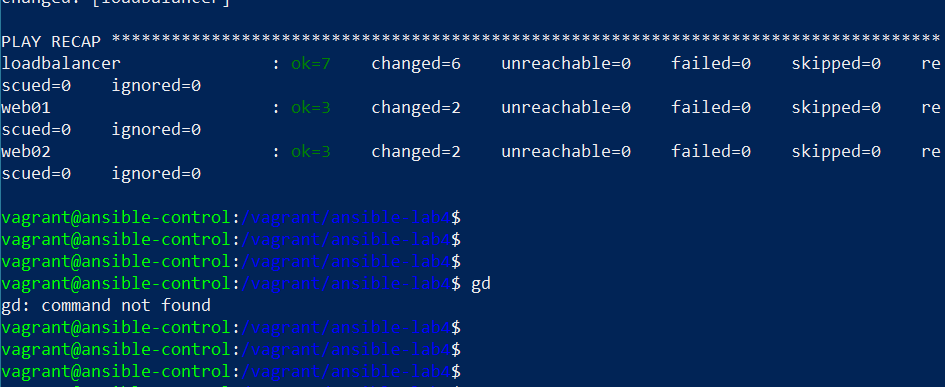
If everything is as it should be, we will get the desired yellow color.

PLAY RECAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

loadbalancer : ok=6 changed=5 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

web01 : ok=6 changed=1 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

web02 : ok=6 changed=1 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0



Also I you can test the Nginx Loadbalancer with the following command:

for i in {1..10}; do curl loadbalancer; done

You should get an interesting result :)

Text

Description automatically generated

# Ansible Lab 5 - Variables, Ansible Vault

Before we make any changes sync the current ansible-lab state to the newly create lab structure:

**rsync -WaP /vagrant/ansible-lab4/ /vagrant/ansible-lab5/**

**cd /vagrant/ansible-lab5/**

and move to the fifth lab directory.

Graphical user interface, text, application

Description automatically generated



In this lab, we can add the MySQL role and get a database up and running, with the appropriate configuration.

1. Create mysql role using ansible-galaxy.

**ansible-galaxy init roles/mysql**



1. Create tasks, handlers and templates for new mysql role.

We will create the installation task:

**nano roles/mysql/tasks/install\_mysql.yml**

and enter the following:

- name: Install MySQL server

apt:

name: mysql-server

update\_cache: yes

- name: Installing python module MySQL-python

pip:

name: PyMySQL

- name: Ensure mysql-server is running

service:

name: mysql

state: started



We will create the configuration tasks:

**nano roles/mysql/tasks/setup\_mysql.yml**

and paste:

- name: Create my.cnf configuration file

template: src=templates/my.cnf.j2 dest=/etc/mysql/conf.d/mysql.cnf

notify: restart mysql

- name: Configure MySQL server to listen on all interfaces

lineinfile:

path: /etc/mysql/mysql.conf.d/mysqld.cnf

regexp: '^bind-address'

line: 'bind-address = 0.0.0.0'

notify:

- restart mysql

- name: Configure MySQL root password

mysql\_user:

login\_user: root

login\_password: "{{ mysql\_root\_password }}"

user: root

password: "{{ mysql\_root\_password }}"

host\_all: yes

host: "%"

update\_password: always

login\_unix\_socket: /var/run/mysqld/mysqld.sock

- name: Restart the mysql-server

service:

name: mysql

state: restarted

- name: "Create new database {{ mysql\_database }}"

mysql\_db:

name: "{{ mysql\_database }}"

state: present

login\_user: root

login\_password: "{{ mysql\_root\_password }}"

- name: "Create new user {{ mysql\_user }} with all database privileges"

mysql\_user:

name: "{{ mysql\_user }}"

password: "{{ mysql\_password }}"

priv: "{{ mysql\_database }}.\*:ALL"

host: '%'

state: present

login\_user: root

login\_password: "{{ mysql\_root\_password }}"



After we define the MySQL role tasks we need to import them in the main.yml file:

**nano roles/mysql/tasks/main.yml**

and include the created files:

---

# tasks file for roles/mysql

- include: install\_mysql.yml

- include: setup\_mysql.yml

Next is the template config:



mkdir roles/mysql/templates

nano roles/mysql/templates/my.cnf.j2



paste the following:

[mysql]

bind-address = 0.0.0.0

Finally, create the restart handler:

**nano roles/mysql/handlers/main.yml**



insert the following code:

---

# handlers file for roles/mysql

- name: restart mysql

service:

name: mysql

state: restarted

listen: "restart mysql"

1. Set our vars and encrypt the important data

In this step, we will move the variables that were declared in the playlist itself to a separate directory and encrypt the MySQL password with the Ansible vault.

**mkdir vars**

**nano vars/main.yml**

add the variables:

---

http\_port: 8000

https\_port: 4443

html\_welcome\_msg: "Hello Scalefocus Academy!"

mysql\_user: simple\_user

mysql\_password: "{{ vaultMySqlPassword }}"

mysql\_root\_password: "{{ vaultMySqlRootPassword }}"



As you can notice mysql\_password is declared as a variable. That's because we have the plan to protect and encrypt it with Ansible Vault. For that, we will create another file:

**nano vars/vault.yml**

the info inside should look similar to this:

**vaultMySqlPassword: "secretMySQLPassword"**

**vaultMySqlRootPassword: "secretRootMySQLPassword"**

after you save the file you can encrypt by using the ansible-vault command:

**ansible-vault encrypt vars/vault.yml**

it will prompt you for an encryption password, so make sure you know what you enter.

Text

Description automatically generated

1. Modify the playbook1.yml and add our new play

**nano playbook1.yml**

after adding the third play the playbook should look like this:

- hosts: webservers

become: yes

vars\_files:

- vars/main.yml

roles:

- common

- apache2

- hosts: proxy

become: yes

roles:

- common

- nginx

- hosts: database

become: yes

vars\_files:

- vars/main.yml

- vars/vault.yml

vars\_prompt:

- name: mysql\_database

prompt: Please enter the database name

private: no

roles:

- common

- mysql

1. Run playbook
2. Timeline

   Description automatically generated

Now, it is time to run our playbook1.yml and see if our MySQL server is provisioned and setup correctly.

**ansible-playbook -i /home/vagrant/hosts -K playbook1.yml --ask-vault-password**

Graphical user interface

Description automatically generated with medium confidence

1. Test a mysql connection to database

Connect to the remote server with the following command and verify that the user simple\_user has been created:

**mysql -h 192.168.7.102 -u simple\_user -p**

Graphical user interface, text, application

Description automatically generated

type the simple\_user password. When it connects to the mysql shell list the databases to verify that the database is created.

**mysql> show databases;**

Graphical user interface, text

Description automatically generated with medium confidence