1. Could you construct a simple playbook to install Nginx on a server?

The playbook file would be:

- hosts: stagingwebservers

gather\_facts: False

vars:

- server\_port: 8080

tasks:

- name: install nginx

apt: pkg=nginx state=installed update\_cache=true

- name: serve nginx config

template: src=../files/flask.conf dest=/etc/nginx/conf.d/

notify:

- restart nginx

handlers:

- name: restart nginx

service: name=nginx state=restarted

- name: restart flask app

service: name=flask-demo state=restarted

...

In the above playbook, we are fetching all hosts of stagingwebservers group for executing these tasks. The first task is to install Nginx and then configure it. We are also taking a flask server for reference. In the end, we also defined handlers so that in case the state changes it will restart Nginx. After executing the above playbook we can verify whether Nginx is installed or not.

ps waux | grep nginx

2. What is the significance of the " notation? And how may variables or dynamic variable names be interpolated?

In most other cases you should always use the brackets, even if previouslly you could use variables without specifying (like with\_ clauses), as this made it hard to distinguish between an undefined variable and a string.

Another rule is ‘moustaches don’t stack’. We often see this:

**{{** somevar\_**{{**other\_var**}}** }}

The above DOES NOT WORK, if you need to use a dynamic variable use the hostvars or vars dictionary as appropriate:

**{{** hostvars**[**inventory\_hostname**][**'somevar\_' **+** other\_var**]** **}}**

3. What is the difference between an Ansible role and a playbook role?

Role is a set of tasks and additional files to configure host to serve for a certain role.

Playbook is a mapping between hosts and roles.

Roles are a way to group tasks together into one container. You could have a role for setting up MySQL, another one for setting up Postfix etc.

A playbook defines what is happening where. This is the place where you define the hosts (hostgroups, see below) and the roles which will be applied to those hosts.

[databases] and the other entries in your inventory are hostgroups. Hostgroups define a set of hosts a play will run on.

A play is a set of tasks or roles (or both) inside a playbook. In most cases (and examples) a playbook will contain only one single play. But you can have as many as you like. That means you could have a playbook which will run the role postfix on the hostgroup mail\_servers and the role mysql on the hostgroup databases:

- hosts: mail\_servers

roles:

- postfix

- hosts: databases

roles:

- mysql

4. How can I write a multi-task Ansible handler in Ansible?

In your handler file, chain the different steps together using notify.

- name: Restart conditionally

debug: msg=Step1

changed\_when: True

notify: Restart conditionally step 2

- name: Restart conditionally step 2

debug: msg=Step2

changed\_when: True

notify: Restart conditionally step 3

- name: Restart conditionally step 3

debug: msg=Step3

Then refer to it from a task with notify: Restart conditionally.

Note that you can only notify to handlers below the current one. So for example, Restart conditionally step 2 can't notify Restart conditionally.

5. What are Ansible Vaults and how do you use them?

Ansible Vault is a feature of ansible that allows you to keep sensitive data such as passwords or keys in encrypted files, rather than as plaintext in playbooks or roles. These vault files can then be distributed or placed in source control.

To enable this feature, a command line tool - [ansible-vault](https://docs.ansible.com/ansible/2.8/cli/ansible-vault.html" \l "ansible-vault) - is used to edit files, and a command line flag ([**--ask-vault-pass**](https://docs.ansible.com/ansible/2.8/cli/ansible-playbook.html#cmdoption-ansible-playbook-ask-vault-pass), [**--vault-password-file**](https://docs.ansible.com/ansible/2.8/cli/ansible-playbook.html#cmdoption-ansible-playbook-vault-password-file) or [**--vault-id**](https://docs.ansible.com/ansible/2.8/cli/ansible-playbook.html#cmdoption-ansible-playbook-vault-id)) is used. Alternately, you may specify the location of a password file or command Ansible to always prompt for the password in your ansible.cfg file. These options require no command line flag usage.

6. How can I use Ansible to create encrypted files?

Ansible has a very simple ad-hoc command for this

ansible all -i localhost, -m debug -a "msg={{ 'mypassword' | password\_hash('sha512', 'mysecretsalt') }}"

We can also use the Passlib library of Python, e.g

python -c "from passlib.hash import sha512\_crypt; import getpass; print(sha512\_crypt.using(rounds=5000).hash(getpass.getpass()))"

On top of this, we should also avoid storing raw passwords in playbook or host\_vars, instead, we should use integrated methods to generate a hash version of a password.

7. What is Ansible Tower, exactly?

Ansible Tower is an enterprise-level solution by RedHat. It provides a web-based console and REST API to manage Ansible across teams in an organization. There are many features such as

Workflow Editor – We can set up different dependencies among playbooks, or running multiple playbooks maintained by different teams at once

Real-Time Analysis – The status of any play or tasks can be monitored easily and we can check what’s going to run next

Audit Trail – Tracking logs are very important so that we can quickly revert back to a functional state if something bad happens.

Execute Commands Remotely – We can use the tower to run any command to a host or group of hosts in our inventory.

There are other features also such as Job Scheduling, Notification Integration, CLI, etc.

8. What are the benefits of the Ansible Tower?

Ansible Tower **allows you to control access to who can access what, even allowing sharing of SSH credentials without someone being able to transfer those credentials**.

Ansible Tower is like Ansible at a more enterprise level. It is a web-based solution for managing your organization with an easy user interface that provides **a dashboard with all of the state summaries of all the hosts.** **And allows quick deployments, and monitors all configurations**.

9. What is the role of Ansible in the Continuous Delivery pipeline? Explain.

Release early & often but keep it simple

We come from a long history of building software the "release early, release often" way. If release often is an ideal, continuous application delivery may be nirvana.

To do it right, automation is key - but so is simplicity. Your team needs the tools that enable quick turnaround, requiring human intervention only when necessary. That's what Ansible does with one of the easiest paths to continuous delivery in the industry.

Rolling updates. Zero Downtime.

Ansible provides true multi-tier, multi-step orchestration. Ansible's push-based architecture allows very fine-grained control over operations, able to orchestrate configuration of servers in batches, all while working with load balancers, monitoring systems, and cloud or web services. Slicing thousands of servers into manageable groups and updating them 100 at a time is incredibly simple, and can be done in a half page of automation content.

Call your play

Ansible let you define "plays", which select a particular group of hosts and assign tasks to execute or roles for them to fulfill. The order in which these plays run, and the hosts on which they run, is tightly controlled by Ansible. For example, you might migrate a database schema and flush the caching servers prior to updating application servers by running one simple play. This is far better than just blasting out orders to servers.

Power plus simplicity, across your environment

**STAGE & TEST**

Your Ansible inventory can be easily split to slice your environment up into different groups of machines. You can then easily test your plays with a staging machine and if tests pass, that can then be instantly run against production if you so choose.

**BEYOND JUST SERVERS**

We don’t stop at just servers. Ansible can work with [networks](https://www.ansible.com/network-automation?hsLang=en-us), load balancers, monitoring systems, web services and other devices that might need touching during a rolling update. For example, you can add or remove servers from your load balancing pool and disable monitoring alerts for each machine that is being updated.

Simple Integration

Ansible fits into any existing development practice and can be integrated into any workflow by utilizing the [Red Hat® Ansible® Tower API](https://www.ansible.com/tower-editions?hsLang=en-us) and the command-line interface. One common way to use Ansible is by calling it from a continuous integration (CI) system upon a successful application build:

* The CI asks Ansible to run a playbook that deploys a staging environment with the application.
* When the stage tests pass, it might then be asked to run a production deployment.
* Ansible can check out your artifacts from version control on each machine, or pull artifacts from the CI server, or from a package mirror.

10. Using Ansible, how do you build a LAMP stack and deploy a webpage?

Ansible playbook provides an alternative to manually running through the procedure outlined in our guide on [How To Install Linux, Apache, MySQL and PHP (LAMP) on Ubuntu 18.04](https://www.digitalocean.com/community/tutorials/how-to-install-linux-apache-mysql-php-lamp-stack-ubuntu-18-04).

Running this playbook will perform the following actions on your Ansible hosts:

1. Install aptitude, which is preferred by Ansible as an alternative to the apt package manager.
2. Install the required LAMP packages.
3. Create a new Apache VirtualHost and set up a dedicated document root for that.
4. Enable the new VirtualHost.
5. Disable the default Apache website, when the **disable\_default** variable is set to true.
6. Set the password for the MySQL **root** user.
7. Remove anonymous MySQL accounts and the test database.
8. Set up UFW to allow HTTP traffic on the configured port (80 by default).
9. Set up a PHP test script using the provided template.

Once the playbook has finished running, you will have a web PHP environment running on top of Apache, based on the options you defined within your configuration variables.

## How to Use this Playbook

The first thing we need to do is obtain the LAMP playbook and its dependencies from the [do-community/ansible-playbooks](https://github.com/do-community/ansible-playbooks) repository. We need to clone this repository to a local folder inside the Ansible Control Node.

In case you have cloned this repository before while following a different guide, access your existing ansible-playbooks copy and run a git pull command to make sure you have updated contents:

1. cd ~/ansible-playbooks
2. git pull

Copy

If this is your first time using the do-community/ansible-playbooks repository, you should start by cloning the repository to your home folder with:

1. cd ~
2. git clone https://github.com/do-community/ansible-playbooks.git
3. cd ansible-playbooks

Copy

The files we’re interested in are located inside the lamp\_ubuntu1804 folder, which has the following structure:

lamp\_ubuntu1804

├── files

│ ├── apache.conf.j2

│ └── info.php.j2

├── vars

│ └── default.yml

├── playbook.yml

└── readme.md

Here is what each of these files are:

* files/info.php.j2: Template file for setting up a PHP test page on the web server’s root
* files/apache.conf.j2: Template file for setting up the Apache VirtualHost.
* vars/default.yml: Variable file for customizing playbook settings.
* playbook.yml: The playbook file, containing the tasks to be executed on the remote server(s).
* readme.md: A text file containing information about this playbook.

We’ll edit the playbook’s variable file to customize the configurations of both MySQL and Apache. Access the lamp\_ubuntu1804 directory and open the vars/default.yml file using your command line editor of choice:

1. cd lamp\_ubuntu1804
2. nano vars/default.yml

Copy

This file contains a few variables that require your attention:

vars/default.yml

---

mysql\_root\_password: "mysql\_root\_password"

app\_user: "sammy"

http\_host: "your\_domain"

http\_conf: "your\_domain.conf"

http\_port: "80"

disable\_default: **true**

Copy

The following list contains a brief explanation of each of these variables and how you might want to change them:

* mysql\_root\_password: The desired password for the **root** MySQL account.
* app\_user: A remote non-root user on the Ansible host that will be set as the owner of the application files.
* http\_host: Your domain name.
* http\_conf: The name of the configuration file that will be created within Apache.
* http\_port: HTTP port for this virtual host, where 80 is the default.
* disable\_default: Whether or not to disable the default website that comes with Apache.

Once you’re done updating the variables inside vars/default.yml, save and close this file. If you used nano, do so by pressing CTRL + X, Y, then ENTER.

You’re now ready to run this playbook on one or more servers. Most playbooks are configured to be executed on every server in your inventory, by default. We can use the -l flag to make sure that only a subset of servers, or a single server, is affected by the playbook. We can also use the -u flag to specify which user on the remote server we’re using to connect and execute the playbook commands on the remote hosts.

To execute the playbook only on server1, connecting as sammy, you can use the following command:

1. ansible-playbook playbook.yml -l server1 -u sammy

Copy

You will get output similar to this:

Output

PLAY [all] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TASK [Gathering Facts] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ok: [server1]

TASK [Install prerequisites] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ok: [server1] => (item=aptitude)

...

TASK [UFW - Allow HTTP on port 80] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [server1]

TASK [Sets Up PHP Info Page] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [server1]

RUNNING HANDLER [Reload Apache] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [server1]

RUNNING HANDLER [Restart Apache] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [server1]

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

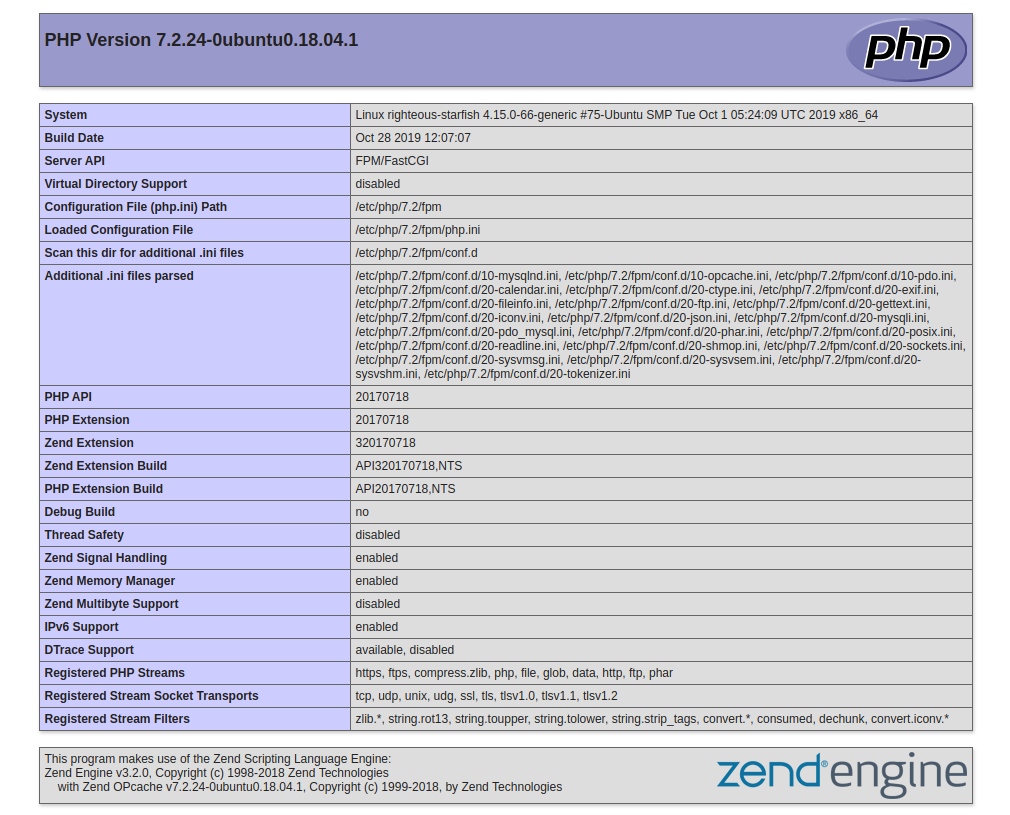
server1 : ok=15 changed=11 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

**Note**: For more information on how to run Ansible playbooks, check our [Ansible Cheat Sheet Guide](https://www.digitalocean.com/community/tutorials/how-to-use-ansible-cheat-sheet-guide).

When the playbook is finished running, go to your web browser and access the host or IP address of the server, as configured in the playbook variables, followed by /info.php:

http://server\_host\_or\_IP/info.php

You will see a page like this:



Because this page contains sensitive information about your PHP environment, it is recommended that you remove it from the server by running an rm -f /var/www/info.php command once you have finished setting it up.

## The Playbook Contents

You can find the LAMP server setup featured in this tutorial in the [lamp\_ubuntu1804](https://github.com/do-community/ansible-playbooks/tree/master/apache_ubuntu1804) folder inside the [DigitalOcean Community Playbooks](https://github.com/do-community/ansible-playbooks) repository. To copy or download the script contents directly, click the **Raw** button towards the top of each script.

The full contents of the playbook as well as its associated files are also included here for your convenience.

### vars/default.yml

The default.yml variable file contains values that will be used within the playbook tasks, such as the password for the MySQL **root** account and the domain name to configure within Apache.

vars/default.yml

---

mysql\_root\_password: "mysql\_root\_password"

app\_user: "sammy"

http\_host: "your\_domain"

http\_conf: "your\_domain.conf"

http\_port: "80"

disable\_default: **true**

Copy

### files/apache.conf.j2

The apache.conf.j2 file is a [Jinja 2](https://jinja.palletsprojects.com/en/2.10.x/) template file that configures a new Apache VirtualHost. The variables used within this template are defined in the vars/default.yml variable file.

files/apache.conf.j2

<VirtualHost \*:{{ http\_port }}>

ServerAdmin webmaster@localhost

ServerName {{ http\_host }}

ServerAlias www.{{ http\_host }}

DocumentRoot /var/www/{{ http\_host }}

ErrorLog ${APACHE\_LOG\_DIR}/error.log

CustomLog ${APACHE\_LOG\_DIR}/access.log combined

<Directory /var/www/{{ http\_host }}>

Options -Indexes

</Directory>

<IfModule mod\_dir.c>

DirectoryIndex index.php index.html index.cgi index.pl index.xhtml index.htm

</IfModule>

</VirtualHost>

### files/info.php.j2

The info.php.j2 file is another Jinja template, used to set up a test PHP script in the document root of the newly configured LAMP server.

files/info.php.j2

<?php

phpinfo();

### playbook.yml

The playbook.yml file is where all tasks from this setup are defined. It starts by defining the group of servers that should be the target of this setup (all), after which it uses become: true to define that tasks should be executed with privilege escalation (sudo) by default. Then, it includes the vars/default.yml variable file to load configuration options.

---

- hosts: all

become: **true**

vars\_files:

- vars/default.yml

tasks:

- name: Install prerequisites

apt: name={{ item }} update\_cache=yes state=latest force\_apt\_get=yes

loop: [ 'aptitude' ]

#Apache Configuration

- name: Install LAMP Packages

apt: name={{ item }} update\_cache=yes state=latest

loop: [ 'apache2', 'mysql-server', 'python3-pymysql', 'php', 'php-mysql', 'libapache2-mod-php' ]

- name: Create document root

file:

path: "/var/www/{{ http\_host }}"

state: directory

owner: "{{ app\_user }}"

mode: '0755'

- name: Set up Apache virtualhost

template:

src: "files/apache.conf.j2"

dest: "/etc/apache2/sites-available/{{ http\_conf }}"

notify: Reload Apache

- name: Enable new site

shell: /usr/sbin/a2ensite {{ http\_conf }}

notify: Reload Apache

- name: Disable default Apache site

shell: /usr/sbin/a2dissite 000-default.conf

when: disable\_default

notify: Reload Apache

# MySQL Configuration

- name: Sets the root password

mysql\_user:

name: root

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

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

- name: Removes all anonymous user accounts

mysql\_user:

name: ''

host\_all: yes

state: absent

login\_user: root

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

- name: Removes the MySQL test database

mysql\_db:

name: test

state: absent

login\_user: root

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

# UFW Configuration

- name: "UFW - Allow HTTP on port {{ http\_port }}"

ufw:

rule: allow

port: "{{ http\_port }}"

proto: tcp

# PHP Info Page

- name: Sets Up PHP Info Page

template:

src: "files/info.php.j2"

dest: "/var/www/{{ http\_host }}/info.php"

handlers:

- name: Reload Apache

service:

name: apache2

state: reloaded

- name: Restart Apache

service:

name: apache2

state: restarted