**How to Use Ansible Roles to Abstract your Infrastructure Environment**

[Ubuntu](https://www.digitalocean.com/community/tags/ubuntu)[Configuration Management](https://www.digitalocean.com/community/tags/configuration-management)[Ansible](https://www.digitalocean.com/community/tags/ansible)

* **By**[**Justin Ellingwood**](https://www.digitalocean.com/community/users/jellingwood)**and [Jamon Camisso](https://www.digitalocean.com/community/users/jamonation)**

**UpdatedJanuary 3, 2020 221.1kviews**

**Introduction**

Ansible carries out tasks on servers that it manages. These tasks invoke built-in and community maintained Ansible modules using small snippets of YAML for each task.

As the number and variety of systems become more complex, it makes sense to group tasks together into [Ansible playbooks](https://www.digitalocean.com/community/tutorials/how-to-create-ansible-playbooks-to-automate-system-configuration-on-ubuntu). Using playbooks eliminates the need to run many individual tasks on remote systems, instead letting you configure entire environments at once with a single file.

However, playbooks can become complex when they are responsible for configuring many different systems with multiple tasks for each system, so Ansible also lets you organize tasks in a directory structure called a **Role**. Roles also allow you to collect templates, static files, and variables along with your tasks in one structured format.

**What is an Ansible Role?**

Roles are a level of abstraction on top of tasks and playbooks that let you structure your Ansible configuration in a modular and reusable format. As you add more and more functionality and flexibility to your playbooks, they can become unwieldy and difficult to maintain. Roles allow you to break down a complex playbook into separate, smaller chunks that can be coordinated by a central entry point. For example, in this tutorial the entire playbook.yml that we will work with looks like this:

Example Playbook

---

- hosts: all

become: true

roles:

- apache

vars:

doc\_root: /var/www/example

The entire set of tasks to configure an Apache web server will be contained in the apache role that we will create. The roll will define all the tasks that need to be completed to install Apache

Organizing your Ansible setup into roles allows you to reuse common configuration steps between different types of servers. roles rely on a known directory structure and file name conventions to automatically load files that will be used within the play.

In general, the idea behind roles is to allow you to share and reuse tasks using a consistent structure, while making it easy to maintain them without duplicating tasks for all your infrastructure.

**Creating a Role**

To create an Ansible role you will need a specifically laid out directory structure. Roles always need this directory layout so that Ansible can find and use them.

To get started, let’s create a directory called roles. Ansible will look here when we want to use our new role in a playbook later in this tutorial.

cd ~

mkdir roles

cd roles

Within this directory, we will define. Each role that we will create requires its own directory. We are going to take the below playbook and turn it into a reusable Ansible role.

For reference, this is the playbook from that tutorial:

playbook.yml

---

- hosts: all

become: true

vars:

doc\_root: /var/www/example

tasks:

- name: Update apt

apt: update\_cache=yes

- name: Install Apache

apt: name=apache2 state=latest

- name: Create custom document root

file: path={{ doc\_root }} state=directory owner=www-data group=www-data

- name: Set up HTML file

copy: src=index.html dest={{ doc\_root }}/index.html owner=www-data group=www-data mode=0644

- name: Set up Apache virtual host file

template: src=vhost.tpl dest=/etc/apache2/sites-available/000-default.conf

notify: restart apache

handlers:

- name: restart apache

service: name=apache2 state=restarted

First, let’s create an Apache directory for our role and populate it with the required directories:

mkdir apache

cd apache

Next we’ll create the required set of sub-directories that will let Ansible know that it should use the contents as a role. Create these directories using the mkdir command:

mkdir defaults files handlers meta templates tasks vars

These directories will contain all of the code to implement our role. Many roles will only use one or a few of these directories depending on the complexity of the tasks involved. When you are writing your own roles, you may not need to create all of these directories.

Here is a description of what each directory represents:

* **defaults**: This directory lets you set default variables for included or dependent roles. Any defaults set here can be overridden in playbooks or inventory files.
* **files**: This directory contains static files and script files that might be copied to or executed on a remote server.
* **handlers**: All handlers that were in your playbook previously can now be added into this directory.
* **meta**: This directory is reserved for role metadata, typically used for dependency management.. For example, you can define a list of roles that must be applied before the current role is invoked.
* **templates**: This directory is reserved for templates that will generate files on remote hosts. Templates typically use variables defined on files located in the vars directory, and on host information that is collected at runtime.
* **tasks**: This directory contains one or more files with tasks that would normally be defined in the tasks section of a regular Ansible playbook. These tasks can directly reference files and templates contained in their respective directories within the role, without the need to provide a full path to the file.
* **vars**: Variables for a role can be specified in files inside this directory and then referenced elsewhere in a role.

If a file called main.yml exists in a directory, its contents will be automatically added to the playbook that calls the role. However, this does not apply to files and templates directories, since their contents need to be referenced explicitly.

**Turning a Playbook Into a Role**

Now that you are familiar with what each directory in an Ansible role is used for, we’ll turn the Apache playbook into a role to organize things better.

We should already have the roles/apache2/{subdirectories} structure set up from the last section. Now, we need to create some YAML files to define our role.

**Creating the Tasks main.yml File**

We’ll start with the tasks subdirectory. Move to that directory now:

cd ~/roles/apache/tasks

We need to create a main.yml file in this directory. We will populate it with the entire contents of the Apache playbook and then edit it to only include tasks.

nano main.yml

The file should look like this when you begin:

main.yml

---

- hosts: all

become: true

vars:

doc\_root: /var/www/example

tasks:

- name: Update apt

apt: update\_cache=yes

- name: Install Apache

apt: name=apache2 state=latest

- name: Create custom document root

file: path={{ doc\_root }} state=directory owner=www-data group=www-data

- name: Set up HTML file

copy: src=index.html dest={{ doc\_root }}/index.html owner=www-data group=www-data mode=0644

- name: Set up Apache virtual host file

template: src=vhost.tpl dest=/etc/apache2/sites-available/000-default.conf

notify: restart apache

handlers:

- name: restart apache

service: name=apache2 state=restarted

We only want to keep the first --- line and the lines in the tasks section that are highlighted. We can also remove the extraneous spaces to the left of our tasks. We will also add a new section to enable an Apache module called modsecurity that we will configure later in this tutorial. After these changes, our new ~/roles/apache/tasks/main.yml file will look like this:

main.yml

---

- name: Update apt

apt: update\_cache=yes

- name: Install Apache

apt: name=apache2 state=latest

- name: Create custom document root

file: path={{ doc\_root }} state=directory owner=www-data group=www-data

- name: Set up HTML file

copy: src=index.html dest={{ doc\_root }}/index.html owner=www-data group=www-data mode=0644

- name: Set up Apache virtual host file

template: src=vhost.tpl dest=/etc/apache2/sites-available/000-default.conf

notify: restart apache

Now the tasks file is easier to follow and understand because it only contains the actual steps that will be performed when we use the Apache role.

Note how the copy and template lines use src=index.html and src=vhost.tpl respectively to reference files in our role, without any preceding path. The directory structure of our role allows referencing files and templates directly by their name, and Ansible will find them automatically for us.

Make sure to save and close the file when you are finished editing it.

**Creating the Handlers main.yml File**

Now that we have the bulk of the playbook in the tasks/main.yml file, we need to move the handlers section into a file located at handlers/main.yml.

First cd into the handlers subdirectory in our role:

* cd ~/roles/apache/handlers

Again, open the file in your text editor and paste the entire contents of the original playbook.yml:

* nano main.yml

The parts that we need to keep are highlighted again:

playbook.yml

1. ---
2. - hosts: all
3. become: true
4. vars:
5. doc\_root: /var/www/example
6. tasks:
7. - name: Update apt
8. apt: update\_cache=yes
9. - name: Install Apache
10. apt: name=apache2 state=latest
11. - name: Create custom document root
12. file: path={{ doc\_root }} state=directory owner=www-data group=www-data
13. - name: Set up HTML file
14. copy: src=index.html dest={{ doc\_root }}/index.html owner=www-data group=www-data mode=0644
15. - name: Set up Apache virtual host file
16. template: src=vhost.tpl dest=/etc/apache2/sites-available/000-default.conf
17. notify: restart apache
18. handlers:
19. - name: restart apache
20. service: name=apache2 state=restarted

Remove the whitespace from before the handlers also. In the end, the file should look like this:

---

- name: restart apache

service: name=apache2 state=restarted

Save and close the file when you are finished.

**Adding Files and Templates**

Now that we have tasks and handlers in place, the next step is to make sure there is an index.html file and a vhost.tpl template so that Ansible can find and place them on our remote servers. Since we referenced these files in the tasks/main.yml file, they need to exist or Ansible will be unable to run the role properly.

First, create the index.html file in the ~/roles/apache/files directory:

* cd ~/roles/apache/files
* nano index.html

Paste the following into the editor, then save and close it:

<html>

<head><title>Configuration Management Hands On</title></head>

<h1>This server was provisioned using <strong>Ansible</strong></h1>

</html>

Next we’ll edit the vhost.tpl template. Change to the templates directory and edit the file with nano:

* cd ~/roles/apache/templates
* nano vhost.tpl

Paste these lines into the editor, then save and close it:

<VirtualHost \*:80>

ServerAdmin webmaster@localhost

DocumentRoot {{ doc\_root }}

<Directory {{ doc\_root }}>

AllowOverride All

Require all granted

</Directory>

</VirtualHost>

**The Meta directory**

If our role depended on another role, we could add a file in the meta directory called main.yml. This file might specify that this role depends on a role called “apt”. In the Apache role that we have created we do not require any dependencies. However, in the hypothetical case of requiring another role like “apt”, the file at ~/roles/apache/meta/main.yml might look like this:

---

dependencies:

- apt

This would ensure that the “apt” role is run before our Apache role. Creating dependencies like this is useful with more complex roles that require other pieces of software or configuration to be in place before running the actual role.

**The Vars directory**

We said earlier that there is a “vars” directory that can be used to set variables for our role. While it is possible to configure default parameters for a role through a vars/main.yml file, this is usually not recommended for smaller roles.

The reason for not using the “vars” directory is that it makes the details of your configuration reside within the roles hierarchy. A role is mostly generic tasks and dependencies, whereas variables are configuration data. Coupling the two makes it harder to reuse your role elsewhere.

Instead, it is better to specify configuration details outside of the role so that you can easily share the role without worrying about exposing sensitive information. Also, variables declared within a role are easily overridden by variables in other locations. It is much better to place variable data in playbooks that are used for specific tasks.

However, the “vars” directory is still worth mentioning here because it is useful with more complex roles. For example, if a role needs to support different Linux distributions, specifying default values for variables can be useful to handle different package names, versions, and configurations.

**Including other files**

Sometimes when you create roles with lots of tasks, dependencies, or conditional logic, they will become large and difficult to understand. In situations like this you can split tasks out into their own files and include them in your tasks/main.yml.

For example, if we had an additional set of tasks to configure TLS for our Apache server, we could separate those out into their own file. We could call the file tasks/tls.yml and include it like this in the tasks/main.yml file:

. . .

tasks:

- include: roles/apache/tasks/tls.yml

**Create a Skeleton Playbook**

Now that we have configured our role structure, we can use it with a minimal playbook compared to the monolithic version at the beginning of this tutorial.

To create a minimal playbook that includes our Apache role, cd out of the role directory (our home directory in this example). Now we can create a playbook file:

* cd ~
* nano playbook.yml

Once you have the file open, paste the following then save and close the file:

---

- hosts: all

become: true

roles:

- apache

vars:

- doc\_root: /var/www/example

if we have multiple role requirements, we can now list them under the roles section in our playbook and they will run in the order they appear.

For instance, if we had roles to set up a WordPress server using Apache and MySQL, we might have a playbook that looks like this:

---

- hosts: wordpress\_hosts

become: true

roles:

- apache

- php

- mysql

- wordpress

vars:

- doc\_root: /var/www/example

This playbook structure allows us to be very succinct about what we want a server to look like. Finally, since playbooks call roles, the command to run is exactly the same as if it all lived in a single file:

* ansible-playbook playbook.yml

Output

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

TASK [Gathering Facts] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [64.225.15.1]

TASK [apache : Update apt] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [64.225.15.1]

TASK [apache : Install Apache] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [64.225.15.1]

TASK [apache : Create custom document root] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [64.225.15.1]

TASK [apache : Set up HTML file] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [64.225.15.1]

TASK [apache : Set up Apache virtual host file] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [64.225.15.1]

RUNNING HANDLER [apache : restart apache] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [64.225.15.1]

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

64.225.15.1 : ok=7 changed=5 unreachable=0 failed=0

You could also call the playbook.yml file apache.yml for example, to make the name of the file reflect the role(s) that it contains.

**Ansible Galaxy**

A tutorial about Ansible roles would not be complete without exploring the resources available via [Ansible Galaxy](https://galaxy.ansible.com/docs/). The searchable [Galaxy](https://galaxy.ansible.com/home) is a repository of user contributed roles that you can add to playbooks to accomplish various tasks without having to write them yourself.

For example, we can add a useful Apache module called mod\_security2 to our playbook to configure Apache with some extra security settings. We will use an Ansible Galaxy role called [apache\_modsecurity](https://galaxy.ansible.com/apolloclark/apache_modsecurity). To use this role, we’ll download it locally and then include it in our playbook.

First let’s get familiar with the ansible-galaxy tool. We will search the Galaxy using the tool and then choose a role from the list that is returned from our search command:

* ansible-galaxy search "PHP for RedHat/CentOS/Fedora/Debian/Ubuntu"

The search command will output something like the following:

Output

Found 21 roles matching your search:

Name Description

---- -----------

alikins.php PHP for RedHat/CentOS/Fedora/Debian/Ubuntu.

bpresles.php PHP for RedHat/CentOS/Fedora/Debian/Ubuntu.

entanet\_devops.ansible\_role\_php PHP for RedHat/CentOS/Fedora/Debian/Ubuntu.

esperdyne.php PHP for RedHat/CentOS/Fedora/Debian/Ubuntu.

fidanf.php PHP for RedHat/CentOS/Fedora/Debian/Ubuntu.

frogasia.ansible-role-php PHP for RedHat/CentOS/Fedora/Debian/Ubuntu.

geerlingguy.php PHP for RedHat/CentOS/Fedora/Debian/Ubuntu.

icamys.php PHP for RedHat/CentOS/Fedora/Debian/Ubuntu.

jhu-sheridan-libraries.php PHP for RedHat/CentOS/Fedora/Debian/Ubuntu.

jibsan94.ansible\_php PHP for RedHat/CentOS/Fedora/Debian/Ubuntu.

KAMI911.ansible\_role\_php PHP for RedHat/CentOS/Fedora/Debian/Ubuntu.

monsieurbiz.geerlingguy\_php PHP for RedHat/CentOS/Fedora/Debian/Ubuntu.

nesh-younify.ansible-role-php PHP for RedHat/CentOS/Fedora/Debian/Ubuntu.

net2grid.php PHP for RedHat/CentOS/Fedora/Debian/Ubuntu.

thom8.ansible-role-php PHP for RedHat/CentOS/Fedora/Debian/Ubuntu.

v0rts.php PHP for RedHat/CentOS/Fedora/Debian/Ubuntu.

vahubert.php PHP for RedHat/CentOS/Fedora/Debian/Ubuntu.

Vaizard.mage\_php PHP for RedHat/CentOS/Fedora/Debian/Ubuntu.

viasite-ansible.php PHP for RedHat/CentOS/Fedora/Debian/Ubuntu.

vvgelder.ansible-role-php PHP for RedHat/CentOS/Fedora/Debian/Ubuntu.

(END)

Ansible will use the less command to output the search results if there are many results, which will block your terminal until you press q to exit. This is useful for when the search results are extensive and you need to paginate across them, which you can do by pressing space.

We will pick the role geerlingguy.php for our playbook. If you would like to read more about the roles returned by your search results, you can visit the [Galaxy search page](https://galaxy.ansible.com/search) and paste in the role name that you’d like to learn more about.

To download a role for use in our playbook, we use the ansible-galaxy install command:

* ansible-galaxy install geerlingguy.php

When you run that command you should see output like this:

Output

- downloading role 'php', owned by geerlingguy

- downloading role from https://github.com/geerlingguy/ansible-role-php/archive/3.7.0.tar.gz

- extracting geerlingguy.php to /home/sammy/.ansible/roles/geerlingguy.php

- geerlingguy.php (3.7.0) was installed successfully

Now we can add the role to our playbook.yml file:

---

- hosts: all

become: true

roles:

- apache

- geerlingguy.php

vars:

- doc\_root: /var/www/example

- php\_default\_version\_debian: "7.2"

By placing the role after our apache role, we ensure Apache is set up and configured on remote systems before any configuration for the geerlingguy.php role place. We could also include mysql, and wordpress roles in any order we choose depending on how we want remote servers to behave.

Running ansible-playbook playbook.yml with the added Galaxy role will result in output like the following:

Output

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

TASK [Gathering Facts] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [64.225.15.1]

TASK [apache : Update apt] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [64.225.15.1]

TASK [apache : Install Apache] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [64.225.15.1]

TASK [apache : Install modsecurity] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [64.225.15.1]

TASK [apache : Create custom document root] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [64.225.15.1]

TASK [apache : Set up HTML file] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [64.225.15.1]

TASK [apache : Set up Apache virtual host file] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [64.225.15.1]

TASK [geerlingguy.php : Include OS-specific variables.] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [64.225.15.1]

TASK [geerlingguy.php : Define php\_packages.] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [64.225.15.1]

. . .

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

64.225.15.1 : ok=37 changed=15 unreachable=0 failed=0

(END)

**Conclusion**

Ansible roles are an excellent way to structure and define what your servers should look like. It is worth learning how to use them even if you could rely solely on playbooks for each of your servers. If you plan on using Ansible extensively, roles will keep your host-level configuration separate from your task, and ensure your Ansible code is clean and readable. Most importantly, roles allow you to easily reuse and share code, and to implement your changes in controlled and modular fashion.