**Introduction**

Roles are a robust feature of Ansible that facilitate reuse and further promote modularization of configuration, but Ansible roles are often overlooked in lieu of straightforward playbooks for the task at hand. The good news is that **Ansible roles are simple to get set up and allow for complexity when necessary**

**The anatomy of an Ansible role**

The concept of an Ansible role is simple; it is a group of variables, tasks, files, and handlers that are stored in a standardized file structure.

---

- hosts: apache

sudo: yes

vars:

http\_port: 80

domain: example.com

tasks:

- name: install apache2

apt: name=apache2 update\_cache=yes state=latest

- name: enabled mod\_rewrite

apache2\_module: name=rewrite state=present

notify:

- restart apache2

- name: apache2 listen on port {{ http\_port }}

lineinfile: dest=/etc/apache2/ports.conf regexp="^Listen " line="Listen {{ http\_port }}" state=present

notify:

- restart apache2

- name: apache2 virtualhost on port {{ http\_port }}

lineinfile: dest=/etc/apache2/sites-available/000-default.conf regexp="^<VirtualHost \\*:" line="<VirtualHost \*:{{ http\_port }}>"

notify:

- restart apache2

- name: create virtual host file

template: src=virtualhost.conf dest=/etc/apache2/sites-available/{{ domain }}.conf

- name: a2ensite {{ domain }}

command: a2ensite {{ domain }}

args:

creates: /etc/apache2/sites-enabled/{{ domain }}.conf

notify:

- restart apache2

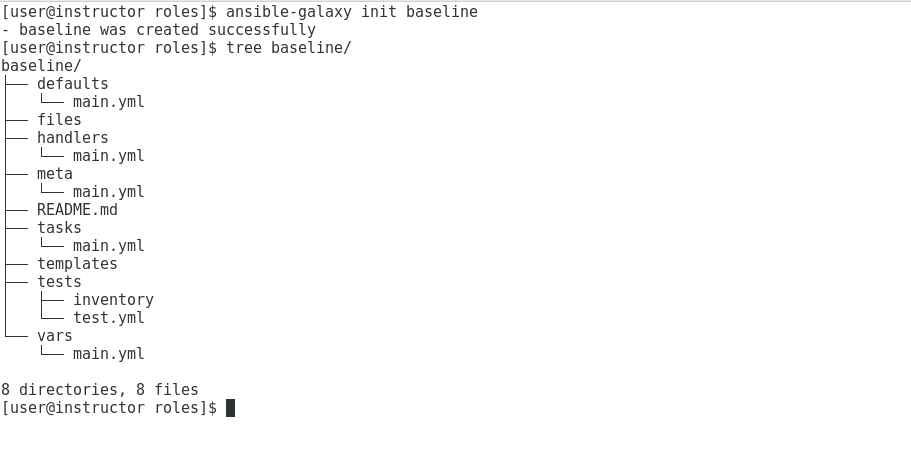
handlers:

- name: restart apache2

service: name=apache2 state=restarted

The most complicated part of a role is recalling the directory structure, but there is help. The built-in *ansible-galaxy* command has a subcommand that will create our role skeleton for us.

Simply use ansible-galaxy init <ROLE\_NAME> to create a new role in your present working directory. You will see here that several directories and files are created within the new role:



The number of files and directories may appear intimidating, but they are fairly straightforward. Most directories contain a *main.yml* file; Ansible uses each of those files as the entry point for reading the contents of the directory (except for files, templates, and test). You have the freedom to branch your tasks and variables into other files within each directory. But when you do this, you must use the ***include*** directive in a directory’s *main.yml* to have your files utilized as part of the role. We will take a closer look at this after a brief rundown of each directory’s purpose.

The **defaults directory** is designated for variable defaults that take the lowest precedence. Put another way: If a variable is defined nowhere else, the definition given in *defaults/main.yml* will be used.

The ***files* and *templates*** directories serve a similar purpose. They contain affiliated files and Ansible templates (respectively) that are used within the role. The beautiful part about these directories is that Ansible does not need a path for resources stored in them when working in the role. Ansible checks them first. You may still use the full path if you want to reference files outside of the role, however, best practices suggest that you keep all of the role components together.

The **handlers** directory is used to store Ansible handlers. If you aren’t familiar, Ansible handlers are simply tasks that may be flagged during a play to run at the play’s completion. You may have as many or as few handlers as are needed for your role.

The **meta** directory contains **authorship information** which is useful if you choose to publish your role on [galaxy.ansible.com](http://galaxy.ansible.com). The meta directory may also be used to define role dependencies. As you may suspect, a role dependency allows you to require that other roles be installed prior to the role in question.

The ***README****.md* file is simply a README file in markdown format. This file is essential for roles published to [galaxy.ansible.com](http://galaxy.ansible.com) and, honestly, the file should include a general description of how your role operates even if you do not make it publicly available.

The ***task*** directory is where most of your role will be written. This directory includes all the tasks that your role will run. Ideally, each logically related series of tasks would be laid out in their own files, and simply included through the *main.yml* file in the *tasks* directory.

The ***test*** directory contains a sample inventory and a *test.yml* playbook. This may be useful if you have an automated testing process built around your role. It can also be handy as you are constructing your role but use of it is not mandatory.

The last directory created is the ***vars*** directory. This is where you create variable files that define necessary variables for your role. The variables defined in this directory are meant for role internal use only. It is a good idea to namespace your role variable names, to prevent potential naming conflicts with variables outside of your role. For example, if you needed a variable named ***config\_file*** in your baseline playbook, you may want to name your variable ***baseline\_config\_file***, to avoid conflicts with another possible ***config\_file*** variable defined elsewhere.

**Creating a simple role**

As stated earlier, Ansible roles can be as complex or as simple as you need. Sometimes, it is helpful to start simple and iterate into a more complex role as you shore up the base functionality. Let’s try that, and define a role called *base\_httpd* that installs httpd with a simple configuration and a very simple website.

To get started, we will need to create our role. We could create each directory and file by hand, but it is far simpler to let *ansible-galaxy* do the grunt work for us by simply running ansible-galaxy init httpd:

$ sudo ansible-galaxy -f init httpd

Next, we can create our simple web page in the files directory. For our academic purposes, we can create a file named *index.html* containing some tried and true sample text:

$ sudo touch httpd/files/index.html

$ sudo vi httpd/files/index.html

$ cat httpd/files/index.html

We will create a template of our *httpd.conf* file by copying an existing one from a fresh install of httpd. Let’s take this opportunity to define a couple of default variables in our role. We will do a default listening port of 80 and a LogLevel that will default to warn. We can do this by adding an entry to *defaults/main.yml*:

$ sudo yum install httpd

$ sudo cp /etc/httpd/conf/httpd.conf /etc/ansible/roles/httpd/templates/httpd.conf.j2

$ sudo vi /etc/ansible/roles/httpd/templates/httpd.conf.j2

Find below variables and update the values accordingly

Listen {{ httpd\_listen\_port }}

LogLevel {{ httpd\_loglevel }}

$ sudo vi httpd/defaults/main.yml

#Add the below variables

httpd\_listen\_port: 80

httpd\_loglevel: warn

You will notice here the template file has the *.j2* extension as custom, and I have used grep to highlight where we have customized the template by replacing the default values in *httpd.conf* with Ansible variables. Then I show where the variables are defined in *defaults/main.yml*.  **Using defaults instead of vars here is preferred**, as it allows for later customization without having to change the actual role.

Now that we have our simple website and configuration, we will need to create the tasks to bring our webserver to life. For the sake of example, I will isolate our httpd setup to its own yaml file in *tasks/httpd.yml* and then include that file into the *tasks/main.yml*:

$ sudo vi httpd/tasks/main.yml

#add this below lines

---

# tasks file for httpd

- include: httpd.yml

$ sudo vi httpd/tasks/httpd.yml

#add below lines

---

#Tasks file for httpd

- yum: name=httpd state=present

- template: src=httpd.conf.j2 dest=/etc/httpd/conf/httpd.conf

- copy: src=index.html dest=/var/www/html/index.html

notify:

- restart httpd

Edit the handlers file

$ sudo vi httpd/handlers/main.yml

---

# handlers file for httpd

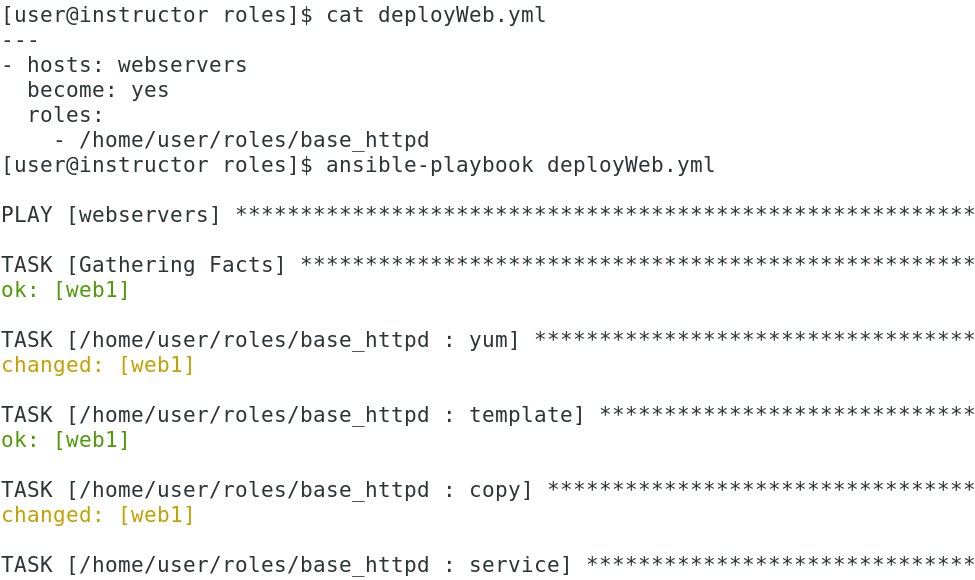
- name: restart httpd

service: name=httpd state=started enabled=yes

We use the yum, template, copy, and service modules to install, configure, and start our webserver. It is worth noting here that I reference the *httpd.yml*, *httpd.conf.j2*, and *index.html* files without full paths as they are stored within the role.

**Deploying the role**

Most of the hard work was completed when we constructed the role itself. Deployment of the role, by comparison, is easy. We only need to set up a simple playbook and pull in the role using the appropriate keyword:



We could easily add typical tasks after the role deployment, or we could also deploy additional roles by simply adding them to the list. Also, we can override the default variables we configured using the same variable names as shown below: