# Creating Roles

## Ansible Galaxy

Ansible has a built in tool `ansible-galaxy` which allows us to quickly create a set of folders and files that are needed in the creation of an Ansible role.

Simply run `ansible-galaxy init rolename --offline` and a folder will be created within your current directory that contains the basic structure of an Ansible role. Within this directory, we can easily pick and choose which components we will need for our role.

## Creating NGINX Roles

To begin, we will create a simple role for installing and configuring a simple nginx server. Navigate within your role, which we will assume is simply called `nginx-role`

### Define Tasks

Within `nginx-role/tasks/main.yml` we include the following -

```
---
# tasks file for /etc/ansible/roles/nginx
- import_tasks: install.yml
- import_tasks: configure.yml
- import_tasks: service.yml
```

This task assumes that within the `nginx-role/tasks/` directory we also have the files `install.yml`, `configure.yml`, and `service.yml` - See the below snippets for examples of how these files could look, depending on your scenario.
Within the `nginx-role/tasks/` directory, create the following files -

Create a `nginx-role/tasks/install.yml` task for installing nginx and any other required packages if needed

```
---
- name: Install nginx Package
apt: name=nginx state=latest
```

Create a `nginx-role/tasks/configure.yml` task for templating various configuration files needed to configure an nginx webserver

```
---
- name: Copy nginx configuration file
  template: src=files/nginx.conf dest=/etc/nginx/nginx.conf
- name: Copy index.html file
  template: src=files/index.html dest=/var/www/html
  notify:
    - restart nginx
```

Create a task `nginx-role/tasks/service.yml` for starting the nginx service

```
---
- name: Start and enable nginx service
  service: name=nginx state=restarted enabled=yes
```

Now we have defined all the tasks that Ansible needs to carryout in order to create a new nginx host. All thats left to do is ensure that the tasks we created above have all the resources we said would be available when the role is ran on a host.

## Define Handlers

In the tasks above, notice the `notify: -restart nginx` within configure.yml. Here, we have declared that this task makes changes that require nginx to be restarted in order to be applied. So, we create the handler task below to carry out the `restart nginx` task that we have notified of our changes.
To set this up, create the following `nginx-role/handlers/main.yml` configuration

```
---
# handlers file for /etc/ansible/roles/nginx
- name: restart nginx
  service: name=nginx state=restarted
```

## Define Templates / Files

Ansible will need to refer to the templates / files we declared in the above tasks -
Add them within the `nginx-role/files/` directory

Create the following `nginx-role/files/nginx.conf`

```
user www-data;
worker_processes auto;
pid /run/nginx.pid;

events { }
```

```
http {
  include mime.types;

  # Basic Server Configuration
  server {
    listen 80;
    server_tokens off;
    server_name {{ domain_name }};

    location / {
      root {{ nginx_root_dir }};
      index {{ index_files }};
    }

    # Uncomment to pass for SSL
    #return 301 https://$host$request_uri;
  }
}
```

Then we can create a custom template at `nginx-role/files/index.html` for our landing page to verify things are working.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Welcome to nginx!</title>
    <style>
      body {
          width: 35em;
          margin: 0 auto;
          font-family: Tahoma, Verdana, Arial, sans-serif;
      }
    </style>
 </head>
  <body>
    <h1>Klips!</h1>
    <p>If you see this page, the nginx web server is successfully installed and working. Further configuration is required.</p>
    <p>For online documentation and support please refer to
```

```
    <a href="http://nginx.org/">nginx.org</a>.<br/>
    Commercial support is available at
    <a href="http://nginx.com/">nginx.com</a>.</p>
    <p><em>Thank you for using nginx.</em></p>
  </body>
</html>
```

## Define Variables / Defaults

Last, we need to define the Ansible defaults we referenced in the above configurations `{{ variable_name }}` is a variable within Ansible, these can be used to create roles that can be used dynamically or easily reconfigured and reapplied to different scenarios.
Create the following `main.yml` file in `nginx-role/defaults`

```
---
# defaults file for /etc/ansible/roles/nginx
#
domain_name: "localhost"
nginx_root_dir: "/var/www/html/"
index_files: "index.html index.htm"
```

Ansible has a wide range of variables, or facts, that it collects on the hosts within its inventory. To see a complete list of all the facts available for a host, run the following

```
ansible hostname -m setup
```

This will print a ton of information, all of which is available for use within ansible templates by calling a variable corresponding to the fact name. For example, if we wanted the fact `ansible_hostname` and `ansible_fqdn`, we call them as `{{ ansible_hostname }}` or `{{ ansible_fqdn }}`. When these variables are ran within a playbook, ansible will insert the values of these variables depending on the host the task is running on.

# Using Ansible Roles

That's it! Now all we need to do is create an inventory / hosts file and run a playbook using our new role -

Create your ansible host file at `/etc/ansible/hosts` with the relevant information for your environment

```
# This is the default ansible 'hosts' file.
#
# It should live in /etc/ansible/hosts
```

```
[group]
www.domain.com
sub.domain.com:22
0.0.0.0
127.0.0.1:22

[othergroup]
sub.domain.com:22
127.0.0.1:22

[nginx-server]
sub.domain.com:22
```

Create the playbook `/etc/ansible/nginx.yml` to kick off our role using the role information and groups entered above within the `hosts` file.

```
---
- hosts: nginx-server
  become: yes
  roles:
  - nginx
```

Now from within `/etc/ansible/`, simply run `ansible-playbook nginx.yml` and our tasks configured above will be carried out on the server defined in the `hosts` file above.

If you are testing using SSL, be sure to use the `--dry-run` argument until your configurations are tested and working correctly.

```
sudo certbot certonly -d domain.com -d www.domain.com --dry-run --standalone --agree-tos -m some-email@domain.com
```

---

Revision #12
Created 30 August 2019 07:13:22 by Shaun Reed
Updated 18 July 2020 10:36:48 by Shaun Reed