Lab 9

Using Galaxy Roles

During this lab you will improve the playbooks with nginx configuration and use a role to provision the front-end and load-balancing machines.

In a previous lab you already noticed both of these have to setup nginx and for sure there are thousands of playbooks that require the same. So you assume there must be ready to use role available for this purpose:

- Use the command line tool to search for an appropriate role for nginx `ansible-galaxy search nginx`
- most likely you received more than 1000 results, try to narrow on the platform (try compatible platforms such as "Fedora" or "EL") `ansible-galaxy search nginx --platforms=EL`
- Most likely still many results, try adding some tags. We also need load balancing `ansible-galaxy search nginx --platforms=EL --galaxy-tags=load balancer`
- You should see a handful of results.
- Let's have a bit more information on the one named *geerlingguy.nginx* `ansible-galaxy info geerlingguy.nginx`

After looking at the info you decide to use this one, but also check the Galaxy web console https://galaxy.ansible.com/ (https://galaxy.ansible.com/) and search for the same. You should be able to locate https://galaxy.ansible.com/geerlingguy/nginx (https://galaxy.ansible.com/geerlingguy/nginx)

Glance over the README and notice it really seems exactly what we need.

Install

Let's install this role:

- Use the CLI to install it

```
$ ansible-galaxy install geerlingguy.nginx
- downloading role 'nginx', owned by geerlingguy
- downloading role from https://github.com/geerlingguy/ansible-role-nginx
/archive/2.6.0.tar.gz
- extracting geerlingguy.nginx to /home/student/.ansible/roles/geerlingguy.nginx
- geerlingguy.nginx (2.6.0) was installed successfully
```

Notice the role has been installed under `~/.ansible/`

Open the `galaxy-roles/front-end.yaml` . When you open the file, you'll notice some contents is already available. It should look familiar as it should resemble a play you created earlier on. Some differences:

- We removed the installation, configuration and restarting of nginx
- We introduced a variable named `www_dir` as the directory for the html/js files (notice how this variable is used in the three tasks)

It will be your responsibility to use the role we chose `geerlingguy.nginx` . You want to open the role's readme while you go through these steps: https://github.com/geerlingguy/ansible-role-nginx/blob/master/README.md (https://github.com/geerlingguy/ansible-role-nginx/blob/master/README.md).

We could make an exact copy of the configuration we've created earlier, but we just take some of the more

important ones and add some others.

Start by applying the role to our hosts `import_role:` with a nested `name`

http config

- You will be setting variables, add the yaml to support that

```
- import_role:
    name: geerlingguy.nginx
  vars:
```

- Recall we removed the version from our server's response header, we'll do the same using the role. The role provides a standard variable for this, try and find it `nginx_server_tokens: "off"`
- We also set the `max_body_size` to `20m` , do the same using an available role variable `nginx_client_max_body_size: "20m"`
- Not all properties we defined earlier have corresponding role variables. We won't define those, but would you be able to find a way to define additional http configuration `nginx_extra_http_options`
- Use this option to set our expiration map:

```
map $sent_http_content_type $expires {
  default                    off;
  text/html                  epoch;
  text/css                   max;
  application/javascript     max;
  ~image/                    max;
}
```

server blocks (vhost definitions)

Let's continue and define the variables for the server definition:

- however first, make sure the default host is removed `nginx_remove_default_vhost: true`
- Then add a variable to override the vhost configuration `nginx_vhosts`
- Add a single host
  - listens on port 80
  - name of the server `localhost`
  - the server's root is our `www_dir` variable
  - make sure you add the following additional server configuration `extra_parameters` :

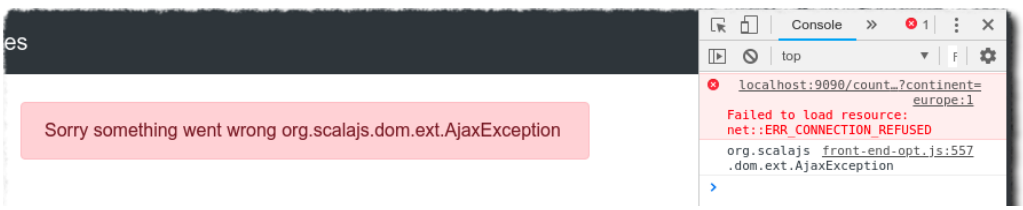
```
expires $expires;
```

Test

Ready to test? You probably want to make sure you have a clean slate for machine-3:

```
machines$ vagrant snapshot restore machine-3 initial
```

Let's test, as before the we will see an error as we are *bypassing* the load-balancer, but as before seeing the following page on http://10.20.1.3/index.html (http://10.20.1.3/index.html) means you have been successful



Also check your headers

```
$ curl -I http://10.20.1.3/index.html
```

Open the `galaxy-roles/load-balancer.yaml` . This should be similar as how you left this file during an earlier exercise. We have just removed all the nginx installation/configuration. Again you will use the same role as you've used for the front-end to configure the load-balancer. This way you can see how to use a role and apply it to different needs.

Make sure you import the role after setting up the selinux permissions (if this would have been ansible 2.2, how would you have accomplished this by using `pre_tasks`)

- As before, turn off the server tokens
- also make sure that the default server configuration is removed

You will need to define two upstream servers `nginx_upstreams:` :

- Define one named 'front-end' `name`
- set its upstream servers `servers` using a jinja template (similar as you created in `worldstack/solutions/files/upstream.conf.j2`). The syntax should resemble:

```
{
     "srv1.example.com",
     "srv2.example.com weight=3",
     "srv3.example.com"
}
```

- Define a similar one for `rest-servers` (no looping possibility for setting these variables)

Then you can define the server/vhosts. These will differ from each other. Let's first define the front-end:

- Is should listen to port `80`
- use `web` for the server-name (normally this would be a domain name, but we need it in order to make sure the configuration files are named different)
- Make sure the server configuration includes the following (it resembles what you defined yourself in an earlier lab)

```
location / {
    proxy_pass http://front-end;
}
error_page   500 502 503 504  /50x.html;
   location = /50x.html {
     root   /usr/share/nginx/html;
 }
```

And for the rest-servers:

- The port is `9090`
- for the server name, use `api`
- and make sure it the server has the following configuration (notice we don't bother with the 50x error mapping, but we do care about forwarding te proxy headers)

```
location / {
    proxy_pass http://rest-servers;
    proxy_set_header        X-Real-IP          $remote_addr;
    proxy_set_header        X-Forwarded-Port   9090;
    proxy_set_header        X-Forwarded-Host   $host:9090;
    proxy_set_header        X-Forwarded-Server $host;
    proxy_set_header        X-Forwarded-Proto  http;
}
```

Done! Let's test it:

First reset your machine so you feel more confident about the result:

```
machines$ vagrant snapshot restore machine-2 initial
```

Then run your playbook and test if your application is available through the load-balancer: http://10.20.1.2/