Configuration Management with Ansible

Overview

Step 1: Setting up and exploring your git/Ansible environment

Your git workflow

We are providing an existing Ansible git repository that will be the base of your configuration management for you cluster. Our first step is to access the "CM" (configuration management server).

The CM lives at `10.0.52.146`. You may recognize this as also being our repo server for our yum repos.

You should be able to login with:

```
$ ssh <username>@10.0.52.146
Password: ChangeMe123
[lowell@te-cm ~]$
```

If you take a look at your groups:

```
[lowell@te-cm ~]$ groups
lowell cscnsi group1
```

We will use the `cscnsi` group to access files that you need. Your `group[0-9]` is the identifier for your particular group. ***You will be working on configuration management as a group.*** Note that you do not have `sudo` access on this system.

You also have access to a group-shared folder:

```
[lowell@te-cm ~]$ ls -l /home/share
total 0
drwxrws--- 2 root group1 6 Jun 12 11:50 group1
drwxrws--- 2 root group2 6 Jun 12 11:50 group2
drwxrws--- 2 root group3 6 Jun 12 11:50 group3
drwxrws--- 2 root group4 6 Jun 12 11:50 group4
```

You will use these folders to host your ***bare*** ansible repos that you will collaborate with.

The master repo that you will base your cluster on is located at `/data/ansible`. We will ***bare*** clone the repo into our group folders:

```
[lowell@te-cm ~]$ cd /home/share/group1
[lowell@te-cm group1]$ git clone --bare /data/ansible
Cloning into bare repository 'ansible.git'...
done.
[lowell@te-cm group1]$ ls -l
total 0
drwxrwsr-x 7 lowell group1 138 Jun 12 12:31 ansible.git
```

Now, as our individual users, we should make our own clone of this repo in our own folder.

```
[lowell@te-cm group1]$ cd
[lowell@te-cm ~]$ git clone /home/share/group1/ansible.git ansible
Cloning into 'ansible'...
done.
[lowell@te-cm ~]$ ls -l
total 0
drwxrwxr-x  5 lowell lowell 85 Jun 12 12:33 ansible
```

Now would be a good time to set your git global config:

```
[lowell@te-cm ansible]$ git config --global user.email lowell@lanl.gov
[lowell@te-cm ansible]$ git config --global user.name "J. Lowell Wofford"
```

Your workflow will be:

1. Pull down the latest version of your own git repo with `git pull`
2. Make changes you need
3. Push your changes (to `/home/share/group*/ansible`)
4. (maybe) run your ansible changes

Each group will now be working with their own shared ansible repository.

Navigating your ansible repo

Let's look inside our local clone of the ansible repo.

```
[lowell@te-cm ansible]$ ls -l
total 4
drwxrwxr-x  4 lowell lowell  54 Jun 12 12:33 inventories
drwxrwxr-x 19 lowell lowell 284 Jun 12 12:33 roles
-rw-rw-r--  1 lowell lowell 540 Jun 12 12:33 site.yaml
```

These files have the following purpose:

- `inventories` defines what hosts we know about and any host/group variables in our configuration.
- `roles` is where all of our ansible roles live.
- `site.yaml` is our playbook.

Let's look at each of these.

`inventories`

Under `inventories` we see:

```
[lowell@te-cm ansible]$ cd inventories/
[lowell@te-cm inventories]$ ls -l
total 4
drwxrwxr-x 2 lowell lowell 62 Jun 12 12:33 group_vars
drwxrwxr-x 2 lowell lowell 23 Jun 12 12:33 host_vars
-rw-rw-r-- 1 lowell lowell 63 Jun 12 12:33 hosts
```

- `host_vars` is where host specific variables live. This is where you will do most of your work.
- `group_vars` provide handy variables that are common across all of our clusters. Values set here can be overriden by `host_vars`.
- `hosts` is the listing of hosts in our CM. It also binds hosts to groups, which then map to files under `group_vars`.

Let's take a look at `hosts`:

```
[lowell@te-cm inventories]$ cat hosts
[masters]
ex-master

[warewulf]
ex-master

[cscnsi]
ex-master
```

Taking a look at `group_vars`:

```
[lowell@te-cm inventories]$ ls -l group_vars/
total 16
-rw-rw-r-- 1 lowell lowell  132 Jun 12 12:33 cms
-rw-rw-r-- 1 lowell lowell 1643 Jun 12 12:33 cscnsi
-rw-rw-r-- 1 lowell lowell  104 Jun 12 12:33 masters
-rw-rw-r-- 1 lowell lowell  200 Jun 12 12:33 warewulf
```

We can see that the names of these files map to groups in `hosts`. Variables set in here will be defined for any host that is a member of that group. Let's take a look at one file. Take a look at the `cscnsi` file under `group_vars`. Many of these settings should be recognizable.

Under `host_vars`:

```
[lowell@te-cm inventories]$ ls -l host_vars/
total 4
-rw-rw-r-- 1 lowell lowell 1830 Jun 12 12:33 ex-master
```

We see there is a file that maps in name to values in `hosts`. The provided `ex-master` is intended to function as an example for your own configuration.

`roles`

Under roles we find all of the roles we have defined:

```
[lowell@te-cm inventories]$ cd ../roles
[lowell@te-cm roles]$ ls
cluster_lan  common  hosts     mellanox  mount-hyperv-data
ohpc_dev_components  powerman  serve-repos  warewulf
cm_pkgs      conman  iptables  modules   munge             opensm
repos     slurm
```

We can see there are quite a few of these. A production system might have quite a few more still. The roles tend to be named in a useful way that says what they do. Having a `common` role is a common convention where tasks that all systems are expected to run go.

We will explore roles in a later exercise where we will write our own.

`sites.yml`

If we look inside `sites.yml` we see:

```
- hosts: masters
  roles:
```

```
    - { role: common, tags: [ 'common', 'base' ] }
    - { role: cluster_lan, tags: [ 'cluster_lan', 'base' ] }
    - { role: mellanox, tags: [ 'mellanox' ] }
    - { role: powerman, tags: [ 'powerman' ] }
    - { role: conman, tags: [ 'conman' ] }
    - { role: slurm, tags: [ 'slurm' ] }
    - { role: warewulf, tags: [ 'warewulf' ] }
    - { role: ohpc_dev_components, tags: [ 'ohpc_dev_components', 'pe' ] }
    - { role: modules, tags: [ 'modules', 'pe' ] }
    - { role: iptables, tags: [ 'iptables' ] }
```

This is a playbook. It links hosts in the group `masters` to this sequence of roles. We have also asigned `tags` to each role. This will later give as a convenient way to run ansible with only specific roles.

Step 2: Setting your specific host vars

The first thing you need to do is let me know what your hostname and IP are. I will need to add entries to `/etc/hosts` on the CM that will map your master hostname to the correct IP.

Once your entry is entered, try pinging your master by hostname:

```
[lowell@te-cm ~]$ ping te-master
PING te-master (172.16.1.254) 56(84) bytes of data.
64 bytes from te-master (172.16.1.254): icmp_seq=1 ttl=64 time=0.957 ms
64 bytes from te-master (172.16.1.254): icmp_seq=2 ttl=64 time=0.510 ms
```

This is important because Ansible will need to be able to access your system by hostname.

Setting your host vars

This repository has everything you need for a base operating system, but you have to tell it about your system specifics.

The following should only be done by one user; follow along/work together in `tmux`

Go to `$HOME/ansible/inventories/host_vars`. We need to make a copy of `ex-master` to your master's hostname. In the examples I will be using the hostname `te-master`.

```
[lowell@te-cm ~]$ cd $HOME/ansible/inventories/host_vars
[lowell@te-cm host_vars]$ ls
ex-master
[lowell@te-cm host_vars]$ git -v ex-master te-master
'ex-master' -> 'te-master'
```

Now edit this file to have the specific settings for your cluster.

You will need to set the following values at least:

- `cluster_prefix` is the prefix you want to give your nodes. Up until now we have called them `n<number>`. In this case the prefix is 'n'. It can be whatever you like.
- `cluster_sms_hostname` is the hostname of your master. This should be the same as the name of the file your editing.
- `cluster_compute_nodes` is a "dictionary" of information about yoru compute nodes. You will need to set both `mac:` and `bmc_mac:` with your `em1` MAC addresses and BMC MAC addresses respectively.
- For `cluster_compute_nodes` make sure the node names (e.g. `ex01`) match your `cluster_prefix`.

You will want to SSH to your currently installed master to get your MAC addresses.

On your master, to get the `em` MAC addresses:

```
[root@te-master ~]# wwsh node list
NAME                GROUPS              IPADDR              HWADDR
================================================================================
======
n01                 compute             172.16.0.1
18:66:da:ea:34:7c
...
```

To get just the MAC addresses:

```
[root@te-master ~]# wwsh node list | grep compute | awk '{print $NF}'
18:66:da:ea:34:7c
...
```

To get the BMC MAC addresses:

```
[root@te-master ~]# grep host /etc/warewulf/dhcpd-template.conf
host n01-bmc { hardware ethernet 18:66:da:68:3e:40; fixed-address
...
```

Make sure these are sequential. If they are you can get just a list of the BMC MAC addresses:

```
[root@te-master ~]# grep host /etc/warewulf/dhcpd-template.conf | awk
'{print $6}' | sed -e 's/;//'
18:66:da:68:3e:40
```

Save the file. Now commit it to git.

```
[lowell@te-cm ansible]$ git status
# On branch master
# Untracked files:
```

```
#   (use "git add <file>..." to include in what will be committed)
#
#   inventories/host_vars/te-master
nothing added to commit but untracked files present (use "git add" to
track)
```

```
[lowell@te-cm ansible]$ git commit -m "added te-master host variables"
[master 1a11af9] added te-master host variables
 1 file changed, 86 insertions(+)
 create mode 100644 inventories/host_vars/te-master
```

We should see that we're one commit ahead of our origin:

```
[lowell@te-cm ansible]$ git status
# On branch master
# Your branch is ahead of 'origin/master' by 1 commit.
#   (use "git push" to publish your local commits)
#
nothing to commit, working directory clean
```

Let's not push just yet.

We also need to add our hostname to the `inventories/hosts` file. We can remove `ex-master` while we're at it:

```
[masters]
te-master

[warewulf]
te-master

[cscnsi]
te-master
```

Now let's get that committed to git:

```
[lowell@te-cm ansible]$ git status
# On branch master
# Your branch is ahead of 'origin/master' by 2 commits.
#   (use "git push" to publish your local commits)
#
nothing to commit, working directory clean
```

```
[lowell@te-cm ansible]$ git push
...
Counting objects: 13, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (9/9), done.
Writing objects: 100% (9/9), 1.23 KiB | 0 bytes/s, done.
Total 9 (delta 3), reused 0 (delta 0)
To /home/share/group1/ansible.git
   403ff07..718eb3b  master -> master
```

```
[lowell@te-cm ansible]$ git status
# On branch master
nothing to commit, working directory clean
```

Our origin should now be up-to-date.

One of the other users should `git pull` on their clone and make sure they see the changes.

Step 3: Re-install the master

We now need to go re-install our master to have a clean slate.

IMPORTANT: Make sure you have all of your MAC addresses before we do this!

Step 4: Deploy with Ansible

Setup SSH keys

Every user on `te-cm` should generate SSH keys. Run:

```
[lowell@te-cm ~]$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/lowell/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/lowell/.ssh/id_rsa.
Your public key has been saved in /home/lowell/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:3tVMkpmfyxky7P5BDxCvXqzgjCs3Y0hPBo+qtD7srNw lowell@te-cm
The key's randomart image is:
+---[RSA 2048]----+
|            .    |
|             *   |
|            * o  |
|      .    . X . |
|       +S . * @  |
|      o.+= = B * |
| ..  o =o + + = .|
```

```
|.o+.. o *. .   . |
|.=*E   +.o  ...  |
+----[SHA256]-----+
```

Now, grab the contents of `$HOME/.ssh/id_rsa.pub` for each user:

```
[lowell@te-cm ~]$ cat $HOME/.ssh/id_rsa.pub
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQD5jy8UVjx2etSeDMnpC91t30R0Z8dqmSOAoxnmynNpjb
RiMrZHYe1jvBS+2ERD0D9aZu2iOfHsc9cq4LzdyKMrYIhvdVfqMTQI68FVE2ffKghpwT0IRfYn
+3sjLc/NxH7Pnra+IzXk81BntISkjmqp7wavDMH6k3Dw8kTYTaedD+gnpUiQRa5EunQaO2xrPx
Er2XFv+KmY+qBiagIDg1W/rQbDxDxpE5QmBOM3EFM7zvv+DUp3CiRsBKS04Q9edCxp/ceSK0mD
XbVZtDW9Op2b58IcKOOupQXXt6ax0nzn7CksglvaTkzIxZMNTw+pc1XfzQPcfit8+k1QJf5NOH
L5 lowell@te-cm
```

SSH to the master using your `admin` account. Become root and add these keys to `/root/.ssh/authorized_keys`. Make sure the permissions on `/root/.ssh/authorized_keys` are set to `0600`.

```
$ ssh admin@master
$ sudo -i
# ssh-keygen
...
# vi $HOME/.ssh/authorized_keys
(add your keys, line-by-line)
# chmod 0600 $HOME/.ssh/authorized_keys
```

Your users should now be able to ssh from the CM to your master using `ssh root@<hostname>` without a password. We will use this for running ansible.

Running ansible

We will use the `ansible-playbook` command to run ansible. We will do this *in* your $HOME/ansible directory as yourself. You should be careful not to have more than one person running ansible at once.

The options we will use for `ansible-playbook` are:

```
ansible-playbook -u root -i inventories/hosts -l <hostname> -t <tags>
site.yml
```

The `-u root` option tells ansible to ssh to the master as `root`.

If we don't specify `-l` ansible will run on all hosts.

If we don't specify `-t` ansible will run on all defined roles in the `site.yml`.

It's a good idea to run just the "common" tag until everything is working.

Try running on the common tag:

```
[lowell@te-cm ansible]$ ansible-playbook -u root -i inventories/hosts -l
te-master -t common site.yaml

PLAY [masters]
*********************************************************************************
***************

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

...(lots of output)

PLAY RECAP
*********************************************************************************
***************
te-master                  : ok=15   changed=7    unreachable=0
failed=0
```

If all went well, the summary should say `failed=0`.

We should now be able to fully deploy our master:

```
[lowell@te-cm ansible]$ ansible-playbook -u root -i inventories/hosts -l
te-master site.yaml | tee $HOME/ansible.log
...(huge amount of input; much time)
```

Note:

- We didn't specify `-t`, so this will run the whole recipe.
- We `tee`'ed output to `$HOME/ansible.log` so we can review the output later.

In the ideal world, you would have a perfectly deployed system at this point. Unfortunately, there is a known bug in the warewulf role where we probably need to run it twice. Let's run just that role again:

```
[lowell@te-cm ansible]$ ansible-playbook -u root -i inventories/hosts -l
te-master -t warewulf site.yaml | tee $HOME/ansible-warewulf.log
...(huge amount of input; much time)
```

Our system should now be deployed with only a couple of missing pieces. We will fix those mising pieces (passwd files, ntp, etc) next time by writing our own extension roles.

Check our master

Let's check our master to see what's working.

Check:

- Are the right services running? (dhcpd, xinetd, httpd, powerman, conman, slurmctld)
- What does `wwsh node list` say?
- What does `pm -q` say?
- Can you `conman` to a node?
- Can you boot a node?
- Do the interfaces look ok on the node? `eth0`? `ib0`?
- Can you run IB fabric tests on the node? Does everything look OK?
- Does Slurm look ok? What does `sinfo` say? Do you need to resume node(s)?
- Can you do an `salloc` to get to a node?
- Can you see `lmod` packages on a node with `module avail`?

If all of this checks out, you're good to go! Go ahead and boot all of your nodes.

Next time we'll learn more about how to write our own ansible roles to configure more.

Challenge: Install `fortune-mod` on your nodes using ansible.