Configuration Management with Ansible: Using Ansible

- Configuration Management with Ansible: Using Ansible
  - Overview
  - Step 1: Writing a role to add our users, first try
    - Designing the "users" role
    - The "users" tasks
    - Adding the role to the playbook & running
  - Step 2: Writing an NTPD role
    - Designing the "ntp" role
    - Setting the timezone
    - Installing the ntp package
    - Creating `ntp.conf`
    - Enabling/starting the ntp services
    - Setting up defaults
    - Creating an ntpd handler
    - Putting it all together
  - Step 3: Writing a role to add our users, take two
    - (Re)designing the users role
    - The "users" and "authorized_keys" modules
    - Using dictionaries & loops

Overview

In the previous part, we learned to use Ansible to deploy a cluster. In this part, we will add some functionality to our cluster with new Ansible roles.

Step 1: Writing a role to add our users, first try

Designing the "users" role

You probably already noticed that one of the things that our ansible does not do is set up our individual user accounts. Let's get started by making a role to do this. There are several ways to go about this, but let's start with the simplest to implement first.

One we can create the user accounts is to have ansible provide the whole of the `{passwd,group,shadow}` files directly to the master. There are some obvious pitfalls to this, but it should get us started. Note for instance that storing our `shadow` file in our config repo with password hashes is probably not extremely secure. Let's do it this way to start out anyway.

As we saw in the last part, we can find roles for our repo in `$HOME/ansible/roles`. We'll be creating a new one of these. It's a good idea to give it a name that is simple yet descriptive. We'll call our role `users`.

We can start by making the directory for the role as well as it's `tasks` folder. The `tasks` folder is where our plays live for our role. By default ansible will load the `main.yml` file under `tasks`, but the `main.yml` file could, for instance, load other YAML files of tasks.

```
[lowell@te-cm roles]$ mkdir -p users/tasks
[lowell@te-cm roles]$ cd users/tasks/
```

The "users" tasks

We'll need to create our tasks file. It will have to have three tasks in it, one for each of the files we intend to copy to the master.

Ansible tasks are performed by ansible modules. Our first job is figuring out which module to use. There are a lot of them available. Take a look at The Ansible Module Index. This is your source for what modules available, along with documentation on how to use them.

We're trying to copy files to the master and put them in place (with appropriate ownership, permissions, etc). The module that makes sense for this is the "copy" module, see copy module documentation. Using the "copy" module we can write our three simple tasks:

```
---
- name: Setup /etc/passwd
  copy:
    src: passwd
    dest: /etc/passwd
    owner: root
    group: root
    mode: '0644'

- name: Setup /etc/group
  copy:
    src: group
    dest: /etc/group
    owner: root
    group: root
    mode: '0644'

- name: Setup /etc/shadow
  copy:
    src: shadow
    dest: /etc/shadow
    owner: root
    group: root
    mode: '0000'
```

Note that we have assigned specific ownership and permissions to each file. This prescriptive definition is at the heart of why we use configuration management. Ansible will try to ensure that all of these aspects are correct every time it runs.

Also, notice that we gave each task a descriptive name. There are a lot of naming conventions people use for Ansible, but it is universally agreed that any best practice gives a short, descriptive name for what the play does.

Now that we have the `main.yml` file setup, we need to actually provide the files to copy in. Just like `tasks` is a special folder for plays in the role, the `files` directory can contain files to be provided through modules like the "copy" module. We just need a `passwd`, `group` and `shadow` file to put here.

First, make the `files` directory, then we can `scp` the necessary files from our master:

```
[lowell@te-cm tasks]$ cd $HOME/ansible/roles/users/
[lowell@te-cm users]$ mkdir files
[lowell@te-cm users]$ cd files
[lowell@te-cm files]$ scp root@te-master:/etc/{passwd,group,shadow} .
passwd
100% 1500   564.5KB/s   00:00
group
100%  644   259.8KB/s   00:00
shadow
100%  932   159.1KB/s   00:00
[lowell@te-cm files]$ ls
group  passwd  shadow
```

Our role is now complete. If we update these files, those updates will propagate to our master.

Adding the role to the playbook & running

We have a role now, but we need it to actually be added to our playbook for it to work. We do this by editing `site.yml`. We can add a line to `site.yml` like the one below:

```
...
4     - { role: common, tags: [ 'common', 'base' ] }
5     - { role: users, tags: [ 'users', 'base' ] }
6     - { role: cluster_lan, tags: [ 'cluster_lan', 'base' ] }
...
```

That's all there is to it. We can now try out our play by running:

NOTE: This is a bit dangerous. Be very careful not to accidentally mangle these files, or you may have to work hard to get back into your master.

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

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

TASK [Gathering Facts]
*********************************************************************************
ok: [te-master]
```

```
TASK [users : Setup /etc/passwd]
*******************************************************************
ok: [te-master]

...
PLAY RECAP
*********************************************************************************
*********
te-master                  : ok=4    changed=0    unreachable=0
failed=0
```

It seems to have run fine, but nothing changed (because the files should be the same). Let's add a group, "testers", to `roles/users/files/group` to try it out:

```
...
cgred:x:994:
tss:x:59:
testers:x:65533:
```

Now run it again:

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

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

TASK [users : Setup /etc/group]
*******************************************************************
changed: [te-master]

...
PLAY RECAP
*********************************************************************************
*********
te-master                  : ok=4    changed=1    unreachable=0
failed=0
```

Ok, it reported that we got a change. Let's see if it took. We can ssh to the master and check:

```
[lowell@te-cm ansible]$ ssh root@te-master tail -n3 /etc/group
cgred:x:994:
tss:x:59:
testers:x:65533:
```

That worked!

But, this is clearly less than ideal. For one, we have to specify the ***whole*** `{passwd,group,shadow}` files. Also, we have to store secrets in our repository, which doesn't seem like a good idea.

Step 2: Writing an NTPD role

Designing the "ntp" role

You may have noticed that our cluster does not have the NTPD service defined. This is an important service, and we'll want it for a fully configured system. We can write a play that will create the service.

First, we need to create the role and tasks folder:

```
[lowell@te-cm roles]$ mkdir -p ntp/tasks
[lowell@te-cm roles]$ cd ntp/tasks
```

Let's make a list of the things we'll need to do to get NTP running:

1. NTP needs a correct timezone; we should probably set that first
2. We need to make sure the package is installed
3. We need to write its config: `/etc/ntp.conf`
4. We need to enable the `ntpd` and `ntpdate` services

Setting the timezone

Let's start with the timezone task. Looking through the module list we see that there's a timezone module. That will make the first item easy. We can start our `main.yml` with:

```
---
- name: set timezone
  timezone:
    name: "America/Denver"
```

That's easy enough, but it would be nice if we didn't have to hard-code the timezone. Fortunately, we don't. We can make the timezone a variable. All we have to do is change this to:

```
---
- name: set timezone
  timezone:
    name: "{{ ntp_timezone }}"
```

Now ansible will set the timezone to the value of the variable `ntp_timezone`. That variable can be set in a lot of places, like the `host_vars` or the `group_vars`.

Installing the ntp package

Moving on to the second item, we can use the package module. The package module provides a way to install packages that isn't package-manager specific. There are also ways to specifically use, say, `yum`, but we don't need any special features, so this module will work. Our play for this is relatively simple:

```
- name: install ntpd
  package:
    name: ntp
    state: present
```

The format of this is pretty straight forward. We give it the name of the package, `ntp`, and say that we want it to be `present`. That's it.

Creating `ntp.conf`

Next we need to configure NTP. A very simple NTP config might look like this:

```
driftfile /var/lib/ntp/drift
restrict default nomodify notrap nopeer noquery
restrict 127.0.0.1
restrict ::1
restrict 172.16.0.0 mask 255.255.255.0 nomodify notrap
server 10.0.52.146 prefer
server 127.127.1.0
fudge  127.127.1.0 stratum 10
includefile /etc/ntp/crypto/pw
keys /etc/ntp/keys
disable monitor
```

We could use the "copy" module and copy this file in as we did with the files in Step 1. But, it would be nice to be more flexible. It would be nice to not fully specify the internal network (`172.16.0.0/24`), or the server (`10.0.52.146`) and instead provide those as variables.

To achieve this, we can use the template module. The template module works a bit like copy, but it will fille Jinja2 templates out instead of just copying simple files. The Jinja2 template can use variables much like the variables used in ansible plays themselves. We can use:

```
driftfile /var/lib/ntp/drift
restrict default nomodify notrap nopeer noquery
restrict 127.0.0.1
restrict ::1
restrict {{ ntp_allow_net }} mask {{ ntp_allow_netmask }} nomodify notrap
server {{ ntp_server }} prefer
server 127.127.1.0
fudge  127.127.1.0 stratum 10
includefile /etc/ntp/crypto/pw
keys /etc/ntp/keys
disable monitor
```

Note the specific values are now variable values, like `{{ ntp_allow_net }}`. We can specify these in the `host_vars`, `group_vars`, and other places.

The play for this would be:

```
- name: update ntpd.conf
  template:
    src: ntp.conf.j2
    dest: /etc/ntp.conf
    owner: root
    group: root
    mode: 0444
```

As we can see, this looks a lot like `copy`.

The template itself lives in the `ntp/templates` folder. Make this directory and create the file `ntp.conf.j2` with the template contents above.

Enabling/starting the ntp services

The last of our requires is enabling the `ntpd` and `ntpdate` services. We can use the systemd module. This module lets us manage the various states of systemd services.

```
- name: enable ntp service
  systemd:
    name: ntpdate
    enabled: yes
    state: started

- name: enable ntpdate service
  systemd:
    name: ntpdate
    enabled: yes
```

We simply tell systemd to make sure those two services are enabled, and that they've been started. Note that we only want to tell systemd to enable `ntpdate`, but not start it. `ntpdate` is intended to be run on startup only.

Setting up defaults

If we tried to run our plays without defining the variables we use, they would fail. It would be nice to make sure the variables are at least always defined. We can do this in the `ntp/defaults` folder. Create that folder, and add a file called `main.yml` with these contents:

```
---
ntp_timezone: "America/Denver"
ntp_server: "10.0.52.146"
```

```
ntp_allow_net: "172.16.0.0"
ntp_allow_netmask: "255.255.255.0"
```

This not only makes sure that some sensible defaults are assigned and the play won't fail but is a place we can look to see what variables the role uses at a glance.

Creating an ntpd handler

There's one last step to making this role complete. We would really like to make sure that any time the `ntp.conf` gets updated `ntpd` gets restarted. This can be achieved with what ansible calls `handlers`. Handlers are just plays, but they don't run by default. If a play in the `tasks` folder gets marked as a change, and it has a specification `notify: <handler_name>`, then the handler will get run at the very end of the ansible run. The handler will only get run once no matter how many times it is called.

Handlers live in the `ntp/handlers` folder. Let's create a file `main.yml` in that folder and add the following:

```
---
- name: restart ntpd
  systemd:
    name: ntpd
    state: restarted
```

As you can see, we've used the systemd module. This module lets us manage the various states of systemd services.

Now, let's modify our `ntp.conf` play from above and add a `notify:`:

```
- name: update ntpd.conf
  template:
    src: ntp.conf.j2
    dest: /etc/ntp.conf
    owner: root
    group: root
    mode: 0444
  notify: restart ntpd
```

Notice that we need `restart ntpd` to be exactly the value of `notify:` as well as the `name:` of the handler.

Putting it all together

Our role should now be complete. The filesystem layout should look like this:

```
[lowell@te-cm roles]$ tree ntp
ntp
|-- defaults
|   `-- main.yml
```

```
|-- handlers
|   `-- main.yml
|-- tasks
|   `-- main.yml
`-- templates
    `-- ntp.conf.j2

4 directories, 4 files
```

We can enable the role in the playbook. Edit the `site.yml` and add it there:

```
...
    - { role: cluster_lan, tags: [ 'cluster_lan', 'base' ] }
    - { role: ntp, tags: [ 'ntp', 'base' ] }
    - { role: mellanox, tags: [ 'mellanox' ] }
...
```

Now, let's run it:

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

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

TASK [Gathering Facts]
*********************************************************************************
ok: [te-master]

TASK [ntp : set timezone]
*********************************************************************************
ok: [te-master]
...
```

Let's login to the master and make sure it's actually working:

```
[lowell@te-cm ansible]$ ssh root@te-master
Last login: Thu Jun 13 00:40:29 2019 from 172.16.1.252
tp[root@te-master ~]# ntpq
ntpq> lpeer
     remote           refid      st t when poll reach   delay   offset
jitter
==============================================================================
====
 te-hyperv       .INIT.          16 u   41   64    0    0.000    0.000
0.000
```

```
*LOCAL(0)        .LOCL.          10 l   40   64    1    0.000    0.000
0.000
ntpq>
```

Looks like it's working.

Step 3: Writing a role to add our users, take two

(Re)designing the users role

In Step 1 we wrote a simple "users" module, but it's not very flexible, secure or maintainable. Let's try a better way now that we've learned a couple more tools. We can look at the module list, and notice there's a module called "users" that can do a lot of what we need. Moreover, it might be nice to auto-setup public key authentication rather than pass around password hashes. There's an "authorized_keys" module for that.

In general, it's a good idea to use modules that are built for our task rather than run commands (see command module) or directly manipulate/copy files. This will ultimately be more portable and future-proof since it leverages the work of developers that are actively maintaining the module.

To get a clean slate, let's move the old role out of the way and create a new one:

```
[lowell@te-cm roles]$ mv users users-simple
[lowell@te-cm roles]$ mkdir -p users/tasks
```

The "users" and "authorized_keys" modules

In our `tasks/main.yml` file, we can add an arbitrary user named "joe" and add an authorized key with a play like this (feel free to substitute real information for a user):

```
---
- name: Add the user 'joe', make admin
  user:
    name:   joe
    shell:  /bin/bash
    groups: wheel
    append: yes

- name: Set authorized key for joe
  authorized_key:
    user: joe
    state: present
    key: |
      ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQD5jy8UVjx2etSeDMnpC91t30R0Z8dqmSOAoxnmynNpjb
RiMrZHYe1jvBS+2ERD0D9aZu2iOfHsc9cq4LzdyKMrYIhvdVfqMTQI68FVE2ffKghpwT0IRfYn
+3sjLc/NxH7Pnra+IzXk81BntISkjmqp7wavDMH6k3Dw8kTYTaedD+gnpUiQRa5EunQaO2xrPx
Er2XFv+KmY+qBiagIDg1W/rQbDxDxpE5QmBOM3EFM7zvv+DUp3CiRsBKS04Q9edCxp/ceSK0mD
```

```
XbVZtDW9Op2b58IcKOOupQXXt6ax0nzn7CksglvaTkzIxZMNTw+pc1XfzQPcfit8+k1QJf5NOH
L5 lowell@te-cm
```

We already have our role enabled. Let's give it a try:

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

TASK [users : Add the user 'joe', make admin]
*********************************************
changed: [te-master]

TASK [users : Set authorized key for joe]
*********************************************************
changed: [te-master]

...
te-master                  : ok=3    changed=2    unreachable=0
failed=0
```

Now, let's see if it worked.

```
[lowell@te-cm ansible]$ ssh root@te-master
Last login: Thu Jun 13 05:27:49 2019 from 172.16.1.252
[root@te-master ~]# id joe
uid=1001(joe) gid=1001(joe) groups=1001(joe),10(wheel)
[root@te-master ~]# cat /home/joe/.ssh/authorized_keys
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQD5jy8UVjx2etSeDMnpC91t30R0Z8dqmSOAoxnmynNpjb
RiMrZHYe1jvBS+2ERD0D9aZu2iOfHsc9cq4LzdyKMrYIhvdVfqMTQI68FVE2ffKghpwT0IRfYn
+3sjLc/NxH7Pnra+IzXk81BntISkjmqp7wavDMH6k3Dw8kTYTaedD+gnpUiQRa5EunQaO2xrPx
Er2XFv+KmY+qBiagIDg1W/rQbDxDxpE5QmBOM3EFM7zvv+DUp3CiRsBKS04Q9edCxp/ceSK0mD
XbVZtDW9Op2b58IcKOOupQXXt6ax0nzn7CksglvaTkzIxZMNTw+pc1XfzQPcfit8+k1QJf5NOH
L5 lowell@te-cm
```

Looks like it did.

Using dictionaries & loops

Of course, this is a very static play. It would be much nicer if we could configure our users and keys like we configure everything else.

We can use "dictionaries" and "loops" in ansible to do this. We've already seen dictionaries. Our cluster node definitions are dictionaries.

It often helps to design an example dictionary entry first, then develops the play from that. We need to be able to specify in each entry:

- the username
- any add-on groups (like wheel)
- the ssh-key

In a more complete role we might also allow for specifying things like the shell too, but let's keep this simple.

Any example dictionary might look like:

```
cluster_users:
  joe:
    groups: wheel
    sshkey: |
      ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQD5jy8UVjx2etSeDMnpC91t30R0Z8dqmSOAoxnmynNpjb
RiMrZHYe1jvBS+2ERD0D9aZu2iOfHsc9cq4LzdyKMrYIhvdVfqMTQI68FVE2ffKghpwT0IRfYn
+3sjLc/NxH7Pnra+IzXk81BntISkjmqp7wavDMH6k3Dw8kTYTaedD+gnpUiQRa5EunQaO2xrPx
Er2XFv+KmY+qBiagIDg1W/rQbDxDxpE5QmBOM3EFM7zvv+DUp3CiRsBKS04Q9edCxp/ceSK0mD
XbVZtDW9Op2b58IcKOOupQXXt6ax0nzn7CksglvaTkzIxZMNTw+pc1XfzQPcfit8+k1QJf5NOH
L5 lowell@te-cm
```

Go ahead and add a dictionary like this to your `inventories/host_vars/<hostname>` host vars file. You should use real information for your users.

We could then define all of the users we want in this dictionary. How do we design the `tasks.yml`? We'll start with the answer, then explain it:

```
---
- name: Add the users
  user:
    name:   "{{ item.key }}"
    shell:  /bin/bash
    groups: "{{ item.value.groups }}"
    append: yes
  loop: "{{ query('dict', cluster_users|default({})) }}"

- name: Setup authorized keys
  authorized_key:
    user: "{{ item.key }}"
    state: present
    key: "{{ item.value.sshkey }}"
  loop: "{{ query('dict', cluster_users|default({})) }}"
```

The `loop:` parameter tells ansible that the play should be run once for each result of the expression it contains. The value of each iteration will be set in `item` (note: you *can* rename this, but it's often unnecessary). To get the key of the dictionary (i.e. the username), we use `"{{ item.key }}"`. To get a particular value, we use `"{{ item.value.<name> }}"`.

Let's run our role:

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

TASK [users : Add the users]
*********************************************************************
changed: [te-master] => (item={'key': u'cluening', 'value': {u'sshkey':
u'ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQD5jy8UVjx2etSeDMnpC91t30R0Z8dqmSOAoxnmynNpjb
RiMrZHYe1jvBS+2ERD0D9aZu2iOfHsc9cq4LzdyKMrYIhvdVfqMTQI68FVE2ffKghpwT0IRfYn
+3sjLc/NxH7Pnra+IzXk81BntISkjmqp7wavDMH6k3Dw8kTYTaedD+gnpUiQRa5EunQaO2xrPx
Er2XFv+KmY+qBiagIDg1W/rQbDxDxpE5QmBOM3EFM7zvv+DUp3CiRsBKS04Q9edCxp/ceSK0mD
XbVZtDW9Op2b58IcKOOupQXXt6ax0nzn7CksglvaTkzIxZMNTw+pc1XfzQPcfit8+k1QJf5NOH
L5 lowell@te-cm\n', u'groups': u'wheel'}})

TASK [users : Setup authorized keys]
*********************************************************
changed: [te-master] => (item={'key': u'cluening', 'value': {u'sshkey':
u'ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQD5jy8UVjx2etSeDMnpC91t30R0Z8dqmSOAoxnmynNpjb
RiMrZHYe1jvBS+2ERD0D9aZu2iOfHsc9cq4LzdyKMrYIhvdVfqMTQI68FVE2ffKghpwT0IRfYn
+3sjLc/NxH7Pnra+IzXk81BntISkjmqp7wavDMH6k3Dw8kTYTaedD+gnpUiQRa5EunQaO2xrPx
Er2XFv+KmY+qBiagIDg1W/rQbDxDxpE5QmBOM3EFM7zvv+DUp3CiRsBKS04Q9edCxp/ceSK0mD
XbVZtDW9Op2b58IcKOOupQXXt6ax0nzn7CksglvaTkzIxZMNTw+pc1XfzQPcfit8+k1QJf5NOH
L5 lowell@te-cm\n', u'groups': u'wheel'}})

PLAY RECAP
*********************************************************************************
*********
te-master                  : ok=3    changed=2    unreachable=0
failed=0
```

In my case, I added the "cluening" user. I gave it the ssh key owned by the current user, so I should be able to ssh to that user on the master. Let's try it out:

```
[lowell@te-cm ansible]$ ssh cluening@te-master
Configuring SSH for cluster access
[cluening@te-master ~]$ cat .ssh/authorized_keys
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQD5jy8UVjx2etSeDMnpC91t30R0Z8dqmSOAoxnmynNpjb
RiMrZHYe1jvBS+2ERD0D9aZu2iOfHsc9cq4LzdyKMrYIhvdVfqMTQI68FVE2ffKghpwT0IRfYn
+3sjLc/NxH7Pnra+IzXk81BntISkjmqp7wavDMH6k3Dw8kTYTaedD+gnpUiQRa5EunQaO2xrPx
Er2XFv+KmY+qBiagIDg1W/rQbDxDxpE5QmBOM3EFM7zvv+DUp3CiRsBKS04Q9edCxp/ceSK0mD
XbVZtDW9Op2b58IcKOOupQXXt6ax0nzn7CksglvaTkzIxZMNTw+pc1XfzQPcfit8+k1QJf5NOH
L5 lowell@te-cm
...
```

We now have a much safer, more secure, and more sustainable way to add our users and give them access. This is hardly complete but will work for our purposes. For instance, it doesn't handle ***deleting*** unspecified users.

Make sure you commit all of these changes to your git repo.