Ansible Workshop - Exercises

Basics

Get to know Ansible and learn to write your first Ansible Playbooks.

Bonus exercises

You have finished the lab already. But it doesn't have to end here. We prepared some slightly more advanced bonus labs for you to follow through if you like. So if you are done with the labs and still have some time, here are some more labs for you:

Bonus Lab 1: Prepare infrastructure

Ansible uses SSH to communicate with Linux nodes, the recommended method is to use *SSH-Keys* and not use a password to connect to the managed nodes.
It is also advisable to use a dedicated user for automation on all managed nodes. In our exercises this user will be called `ansible`.

Let's break the initially working (password-less) SSH-connection in the lab environment and establish a new one with the service user `ansible`.

Download a script using the next command. Copy the command by clicking the *copy* button on the right of the code block:

```
wget -q https://raw.githubusercontent.com/TimGrt/prepare-redhat-demo-system/master/break-ssh.sh
```

After downloading the script to your home directory, execute it:

```
[student@ansible-1 ~]$ wget -q https://raw.githubusercontent.com/TimGrt/prepare-redhat-demo-
system/master/break-ssh.sh
[student@ansible-1 ~]$ sh break-ssh.sh
[student@ansible-1 ~]$
```

No output is good output. Now we can configure the SSH connection the way it want.

Success

The goal is to be able to communicate from *ansible-1* as `student` to the `ansible` user on all 3 managed nodes.

We will need the (already present) SSH **public key** of user `student` on *ansible-1* (use your own, not this one!):

```
[student@ansible-1 ~]$ cat .ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCFeZ0j9HODBeDzP5aV5mkrsIGY1mvHTLjbCZIeHNpldIGETKflG6W0/
...
```

Hint

If you want to create your own SSH-Key-Pair, use this command:

```
ssh-keygen
```

Next, SSH to the *ec2-user* on *node1*.

```
[student@ansible-1 ~]$ ssh ec2-user@node1
[ec2-user@node1 ~]$
```

Your are now on *node1*. Switch to the *root* user and create a new user `ansible` (with a home directory). After creating the user, switch to the `ansible` user:

```
[ec2-user@node1 ~]$ sudo su - root
Last login: Sun Apr 17 08:36:53 UTC 2022 on pts/0
[root@node1 ~]# useradd ansible
[root@node1 ~]# su - ansible
[ansible@node1 ~]$
```

Ensure that you are the *ansible* user, we need to create the (hidden) `.ssh` directory and the `authorized_keys` file in it. The `authorized_keys` file houses the **public** key of user *student* on the *ansible-1* host, copy the key to the file (press *i* in *vi* to enter the *insert mode*):

```
[ansible@node1 ~]$ mkdir .ssh
[ansible@node1 ~]$ vi .ssh/authorized_keys
```

Now we have to set the correct permissions, the `.ssh` directory needs *0700*, the `authorized_keys` file needs *0600*.

```
[ansible@node1 ~]$ chmod 0700 .ssh
[ansible@node1 ~]$ chmod 0600 .ssh/authorized_keys
```

Good! We now have established a service user for our automation. The user must be able to do *root-like* tasks e.g. installing and starting services, therefore he needs sudo permissions. Switch back to the *root* user by entering `exit`, you are still on *node1*.

> **Warning**
>
> Clobbering the sudoers file is one of the fastest ways to make your host unusable. Whenever you deal with suoders files, use `visudo`!

As the *root* user, create a new file under `/etc/sudoers.d`:

```
[root@node1 ~]$ visudo -f /etc/sudoers.d/automation
```

Copy the following line which enables the `ansible` user to use password-less sudo (use the copy button of the code block again):

```
ansible ALL=(ALL) NOPASSWD:ALL
```

We can check if the `ansible` user has the required permissions:

```
[root@node1 ~]# sudo -l -U ansible
Matching Defaults entries for ansible on node1:
    !visiblepw, always_set_home, match_group_by_gid, always_query_group_plugin, env_reset,
env_keep="COLORS DISPLAY HOSTNAME HISTSIZE KDEDIR LS_COLORS", env_keep+="MAIL PS1 PS2 QTDIR
USERNAME LANG LC_ADDRESS LC_CTYPE", env_keep+="LC_COLLATE LC_IDENTIFICATION LC_MEASUREMENT
LC_MESSAGES",
    env_keep+="LC_MONETARY LC_NAME LC_NUMERIC LC_PAPER LC_TELEPHONE", env_keep+="LC_TIME LC_ALL
LANGUAGE LINGUAS _XKB_CHARSET XAUTHORITY", secure_path=/sbin\:/bin\:/usr/sbin\:/usr/bin,
!requiretty

User ansible may run the following commands on node1:
    (ALL) NOPASSWD: ALL
```

Log out of *node1* (ensure that you are back on your ansible master node *ansible-1*, run `exit` twice) and try to log in to *node1* with the *ansible* user:

```
[student@ansible-1 ~]$ ssh ansible@node1
[ansible@node1 ~]$
```

Failure

If password-less SSH is not working, check the permissions of the `.ssh` folder and the `authorized_keys` file on the target host!

Success

Repeat the steps above for *node2* and *node3*!

Once you can reach all managed nodes password-less (and sudo-permissions are set, you will need this later), we can start to do some Ansible stuff like executing this Ad-hoc command:

```
[student@ansible-1 ~]$ ansible web -m ping
```

We got an error, all three nodes aren't reachable?! But manually, we can reach all nodes via SSH!
Observing the error message we can see what the problem is, Ansible tries to us reach all hosts as the *student* user. We established the service user *ansible* for that, we must instruct Ansible to use that user. By default, Ansible will use the user that is executing the *ansible* commands.

Open the Ansible inventory file, either by clicking the *lab_inventory* folder and the *hosts* file in the VScode explorer or on the terminal.
Create a new variable section (with *:vars*) for the *web* group and set the *ansible_user=ansible* variable:

```
[web]
node1 ansible_host=<X.X.X.X>
node2 ansible_host=<Y.Y.Y.Y>
node3 ansible_host=<Z.Z.Z.Z>

[web:vars]
ansible_user=ansible

[control]
ansible-1 ansible_host=44.55.66.77
```

All hosts in the *web* group will now use the *ansible* user for the SSH connection. Try with the ad hoc command again:

```
[student@ansible-1 ~]$ ansible web -m ping
node2 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false,
    "ping": "pong"
}
node3 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false,
    "ping": "pong"
}
node1 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false,
    "ping": "pong"
}
```

Success! All three nodes are reachable, we get a *pong* back, we proved that we can establish a SSH connection and that the node(s) have a usable Python interpreter.

Try to run the same ad hoc command against the *control* group.

```
[student@ansible-1 ~]$ ansible control -m ping
```

An error again?? Although being on the same host, Ansible tries to open an SSH connection. Adjust the inventory file again and set the `ansible_connection` variable for the *ansible-1* host:

```
[web]
node1 ansible_host=<X.X.X.X>
node2 ansible_host=<Y.Y.Y.Y>
node3 ansible_host=<Z.Z.Z.Z>

[web:vars]
ansible_user=ansible

[control]
ansible-1 ansible_host=44.55.66.77 ansible_connection=local
```

With `ansible_connection=local` (on host-level) Ansible uses the **local** Python interpreter, which is fine for our Ansible master node. Now the ad hoc command succeeds:

```
[student@ansible-1 ~]$ ansible control -m ping
ansible-1 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false,
    "ping": "pong"
}
```

Bonus Lab 2: Ad Hoc Commands

Create a new user "testuser" on `node1` and `node3` with a comment using an ad hoc command, make sure that it is not created on `node2`!

- Find the parameters for the appropriate module using `ansible-doc user` (leave with `q`)
- Use an Ansible ad hoc command to create the user with the comment "Test D User"
- Use the "command" module with the proper invocation to find the *userid*
- Delete the user and its directories, then check that the user has been deleted

Tip

Remember privilege escalation...

Solution

Your commands could look like these:

```
[student@ansible-1 ansible-files]$ ansible-doc -l | grep -i user
[student@ansible-1 ansible-files]$ ansible-doc user
[student@ansible-1 ansible-files]$ ansible node1,node3 -m user -a "name=testuser comment='Test D
User'" -b
[student@ansible-1 ansible-files]$ ansible node1,node3 -m command -a " id testuser" -b
[student@ansible-1 ansible-files]$ ansible node2 -m command -a " id testuser" -b
[student@ansible-1 ansible-files]$ ansible node1,node3 -m user -a "name=testuser state=absent
remove=yes" -b
[student@ansible-1 ansible-files]$ ansible web -m command -a " id testuser" -b
```

Bonus Lab 3: Templates and Variables

You have learned the basics about Ansible templates, variables and handlers. Let's combine all of these.

Instead of editing and copying `httpd.conf` why don't you just define a variable for the listen port and use it in a template? Here is your job:

- Define a variable `listen_port` for the `web` group with the value `8080` and another for `node2` with the value `80` using the proper files.
- Copy the `httpd.conf` file into the template `httpd.conf.j2` that uses the `listen_port` variable instead of the hard-coded port number.
- Write a Playbook that deploys the template and restarts Apache on changes using a handler.
- Run the Playbook and test the result using `curl`.

Tip

Remember the `group_vars` and `host_vars` directories? If not, refer to the chapter Using variables.

Solution

Define the variable. Add this line to `group_vars/web`:

```
listen_port: 8080
```

Add this line to `host_vars/node2`:

```
listen_port: 80
```

Prepare the template:

- Copy `httpd.conf` to `httpd.conf.j2`
- Edit the "Listen" directive in `httpd.conf.j2` to make it look like this:

```
Listen {{ listen_port }}
```

Create a playbook called `apache_config_tpl.yml`:

```
---
- name: Apache httpd.conf deployment
  hosts: web
  become: true
  tasks:
    - name: Create Apache configuration file from template
      ansible.builtin.template:
        src: httpd.conf.j2
        dest: /etc/httpd/conf/httpd.conf
        mode: "0644"
      notify:
          - Restart_apache
  handlers:
      - name: Restart_apache
        ansible.builtin.service:
          name: httpd
          state: restarted
```

First run the playbook itself, then run curl against `node1` with port `8080` and `node2` with port `80`.

```
[student@ansible-1 ansible-files]$ ansible-playbook apache_config_tpl.yml
[...]
[student@ansible-1 ansible-files]$ curl http://18.195.235.231:8080
<body>
<h1>This is a development webserver, have fun!</h1>
</body>
[student@ansible-1 ansible-files]$ curl http://35.156.28.209:80
<body>
<h1>This is a production webserver, take care!</h1>
</body>
```