Lab 7 – Variables & Inclusions

Introduction

In this lab you will learn how to use variables to create interface descriptions, set MTU values, and ensure the ports are up on a Nexus device.

Please refer to the **Ansible-Pod-Info.docx** file for information on connecting to your Ansible host.

Don't forget to change the XX in your inventory file based on your Pod information.

1. Managing Variables

1.1 Ansible has a concept of variables which can be used to store values and reuse throughout files or playbooks in your ansible project. It can help simplify overall the creation and maintenance of a project as well.

In our case variables will be used for the interface descriptions and for the MTU values.

We can define variables in a few different places in an Ansible project.

1. **Global Scope** - Variables that are set from the CLI or Ansible config.
2. **Play Scope** - Variables are set in the actual playbook.
3. **Host Scope** - Variables set on host groups or individual hosts by the inventory, fact gathering, etc.

In our example we will be working with Play Scope, we will have our variables specified in the **group_vars** folder and in a subfolder called **nxos** since that will be our group we will be running against.

Ensure you are in the proper Lab 7 folder on the CentOS box.

```
cd ~/ansible_labs/lab7-variables-inclusions
```

1.2 Now let's create our playbook file and group vars structure and specify our variables in our **group_vars/nxos/nxos.yaml** file.

So first let's create our directory structure and files we will need:

1. Make the folder structure **group_vars/nxos**
2. Create an inventory file specifying your Nexus switch in a group called **nxos**.
3. Create an ansible.cfg file referencing the above inventory file and to skip host name checking.
4. Create a file called **nxos.yaml** inside the group_vars/nxos folder.
5. Specify the below values in the **nxos.yaml** file, these are used for our NXOS module and our user values from vault and 2 values we will use in our playbook to create interface descriptions and mtu values.
    a. **ansible_connection:** network_cli
    b. **ansible_network_os:** nxos
    c. **ansible_user:** "{{ n9k_user }}"
    d. **ansible_ssh_pass:** "{{ n9k_pw }}"
    e. transport: nxapi
    f. int_desc: "Core uplink Jumbo MTU"
    g. int_mtu: 9216
    h. interface: Ethernet1/2
6. Using what you learned in the vault lab setup the **n9k_user** and **n9k_pw** in your vault file and set them to the values in your pod information sheet. These are considered variables and we are using the {{ }} to specify our variable from our vault file. This is considered using Jinja which we will be going over shortly.
7. Create a playbook called **int_mtu_example.yaml** (under folder lab7-variables-inclusions) targeting the host group **nxos** with the below supplied task to change the interface description:

```
- name: Change Interface Desc and MTU

  nxos_interface:

    name: "{{ interface }}"

    description: "{{ int_desc }}"

    mtu: "{{ int_mtu }}"

    admin_state: up
```

Now we should be able to run our playbook and then see if our description, MTU and state were applied.

```
ansible-playbook int_mtu_example.yaml --ask-vault
```

We should have output similar to below:

```
Vault password:


PLAY [Variable Example] ************************************
*********************************************************
***


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

ok: [n9k-standalone-XX.localdomain]


TASK [Change Interface Desc and MTU] ***********************
*********************************************************
***

changed: [n9k-standalone-XX.localdomain]
```

```
PLAY RECAP *********************************************************
*********************************************************************

n9k-standalone-XX.localdomain : ok=2    changed=1    unreach
able=0    failed=0
```

Login to your switch and verify the changes were made.

1.3 Based off what you have already learned, refactor this playbook to debug and show the output as well as assert if the changes were not made successfully. Remember you need to check that the MTU, and description were changed.

HINT: Here is how you would assert for the MTU:

```
{ that: "ansible_net_interfaces['Ethernet1/2']['mtu'] == \"{
{ int_mtu }}\""}
```

After we have refactored our playbook we should be able to re-run it and see new results.

```
ansible-playbook int_mtu_example.yaml --ask-vault
```

We should get output now similar to below:

```
Vault password:



PLAY [Variable Example] ********************************************
*********************************************************************
```

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

ok: [n9k-standalone-XX.localdomain]


TASK [Change Interface Desc and MTU] *********************
*************************************************************
***

ok: [n9k-standalone-XX.localdomain]


TASK [nxos_facts] ****************************************
*************************************************************
***

ok: [n9k-standalone-XX.localdomain]


TASK [Get MTU Value] *************************************
*************************************************************
***

ok: [n9k-standalone-XX.localdomain] => {

    "ansible_net_interfaces['Ethernet1/2']['mtu']": "9216"

}


TASK [Verify MTU is set to "9216"] ***********************
*************************************************************
***
```

```
ok: [n9k-standalone-XX.localdomain] => {

    "changed": false,

    "msg": "All assertions passed"

}



TASK [Get "Ethernet1/2" description] ************************
*****************************************************************
***

ok: [n9k-standalone-XX.localdomain] => {

    "ansible_net_interfaces['Ethernet1/2']['description']":
"Core uplink Jumbo MTU"

}



TASK [Verify "Ethernet1/2" description matches "Core uplink
Jumbo MTU"] *************************************************
***

ok: [n9k-standalone-XX.localdomain] => {

    "changed": false,

    "msg": "All assertions passed"

}
```

```
PLAY RECAP *********************************************************
*********************************************************************
***

n9k-standalone-XX.localdomain : ok=7    changed=0    unreach
able=0    failed=0
```

2. Managing Inclusions

As an administrator if you are working with long or complex playbooks you might want the option to separate these files to be able to divide up the tasks and the lists of variables into smaller pieces to manage. We can use inclusions to do this.

2.1 First off let's create a folder called inclusions inside our existing lab7 working directory. We will also copy our group_vars folder from the previous section so we don't have to specify any of that again.

```
mkdir inclusions && cd inclusions

cp -R ../group_vars .

cp ../ansible.cfg .

cp ../inventory .
```

2.2 We will be creating a task file, a variable file, and a playbook file. The variable file will include variables used in the playbook. The task file will define what required tasks we will be running.

a. Create a directory called **tasks** and change into that directory.

```
mkdir tasks && cd tasks
```

b. In the tasks directory, create the **change_vlan.yml** task file. Define the **task** to **create** a **new vlan 120**; use the vlan_number **variable** for the vlan number. If you get stuck creating the playbook the command below will create the playbook.

```
cat > change_vlan.yml <<EOF
```

```
---

- name: Adding Vlan "{{ vlan_number }}"

  nxos_vlan:

    vlan_id: "{{ vlan_number }}"

EOF
```

c. Change back into the main project directory. Create a directory named **vars** and change into that directory.

```
cd ..

mkdir vars && cd vars
```

d. In the **vars** directory, create the **variables.yml** variables file. The file defines the **vlan_number** variable in YAML format.

```
cat > variables.yml <<EOF

---

vlan_number: 120

EOF
```

e. Change back to the top-level project directory for the playbook.

```
cd ..
```

2.3 Now we are going to create and edit the main playbook. We will step through each section and discuss what it does and at the end you can copy the **playbook.yml** file output to create it if you so choose.

The playbook **imports the tasks** as well as the **variables**; and it creates the **vlan_number** we specified.

a. Start with a name for the playbook, then add the **nxos** host group.

```
---

- name: Setup Vlan

  hosts: nxos
```

b. Here we define the first task, which uses the **include_vars** module to import extra variables in the playbook. The variables are used by other tasks in the playbook. Include the **variables.yml** variable file created previously.

```
tasks:

  - name: Include the variables from the YAML file

    include_vars: vars/variables.yml
```

c. Here we define the **second task** which uses the include module to include the base **change_vlan.yml** playbook.

```
- name: Include the change_vlan file and set the variables

  include: tasks/change_vlan.yml
```

2.4 Before running the playbook, verify its syntax is correct by running **ansible-playbook —-syntax-check**.

Copy

```
ansible-playbook --syntax-check playbook.yml --ask-vault
```

Output:

```
playbook: playbook.yml
```

2.5 Run the playbook using the **ansible-playbook** command. Watch the output as Ansible starts by including the **change_vlan.yml** playbook and running its tasks, then keeps executing the tasks defined in the main playbook.

```
ansible-playbook playbook.yml --ask-vault
```

Output:

```
Vault password:



PLAY [Setup Vlan] ********************************************
*************************************************************
***



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

ok: [n9k-standalone-XX.localdomain]



TASK [Include the variables from the YAML file] ************
*************************************************************
***

ok: [n9k-standalone-XX.localdomain]
```

```
TASK [Adding Vlan "120"] ***********************************
*************************************************************
***

changed: [n9k-standalone-XX.localdomain]



PLAY RECAP *************************************************
*************************************************************
***

n9k-standalone-XX.localdomain : ok=3    changed=1    unreach
able=0    failed=0
```

2.6 SSH to your nexus switch and ensure the vlan 120 now shows up.

Copy

```
show vlan
```

Output:

```
VLAN Type         Vlan-mode

---- -----        ----------

1    enet         CE

100  enet         CE

101  enet         CE

102  enet         CE

120  enet         CE
```

```
1001 enet         CE

2000 enet         CE
```

3. Knowledge Check - Setup Vlan on Arista EOS

Now based on what we have learned you will setup a new playbook called **eos_vlan.yaml** and use the **group_vars** method of supplying the variables to the playbook.

The vlan you should be configuring is Vlan 120 to match up with the VLAN we created on the switch.

Remember you will need to create a new vault file as well as a new group folder and file under the group_vars section.

HINTS:

- You may need to use the ansible_become, ansible_become_method, and ansible_become_pass variables.
- **EOS Connection Docs - https://docs.ansible.com/ansible/2.6/network/user_guide/platform_eos.html?highlight=eos**
- **EOS VLAN Docs - https://docs.ansible.com/ansible/2.4/eos_vlan_module.html**

Version Control Commit

Now add all your new files to your repository and push it up, reference the GitHub lab if you get stuck or ask for help.