Ansible Control Node, Inventory & Managed Nodes

By: Er. Vikas Nehra (M. Tech, B. Tech), Experience: 15 + Years

Session - 10 Agenda:

1. Ansible Variables:

(a) Playbook Variables

(b) Command Line Variables

(c) Inventory Variables

(b) Command Line Variables:

You can define variables when you run your playbook by passing variables at the command line using the --extra-vars (or -e) argument.

Let's create a playbook to add a user and apply the password.

```
$ mkdir playbooks ; cd playbooks
$ vim command_variable.yml
---
- name: Command Line Variable Example
  hosts: RHEL
  become: true
  tasks:
     - name: Adding user and applying the password
       user:
           name: "{{ user_name }}"
           state: present
           password: "{{ password }}"
...
```

Let's check the playbook for errors.

```
$ ansible-playbook command_variable.yml --syntax-check
```

Let's execute the playbook and provide the variable values on the command line itself.

```
$ ansible RHEL -m command -a 'cat /etc/passwd'
$ ansible-playbook command_variable.yml -e user_name=vikas -e password=redhat
```

Let's verify the changes.

```
$ ansible RHEL -m command -a 'cat /etc/passwd'
$ ansible RHEL -m command -a 'sudo cat /etc/shadow'
```

Similarly, we can use other variable values to create different user accounts.

```
$ ansible-playbook command_variable.yml -e user_name=ashish -e password=abc12
```

If we set the variable values using playbook variables and also pass the command level variable at the same time.

```
$ vim command_variable2.yml
---
- name: Command Line Variable Example
  hosts: RHEL
  become: true
  vars:
       user_name: nehra
       password: redhat
  tasks:
      - name: Adding user and applying the password
        user:
             name: "{{ user_name }}"
             state: present
             password: "{{ password }}"
...
```

And execute the playbook with command line variables then, playbook variables will not be used. Command line varaibles will override the playbook variable values because their priority is higher.

```
$ ansible RHEL -m command -a 'cat /etc/passwd'
$ ansible-playbook command_variable2.yml -e user_name=rahul -e password=redhat
```

Let's verify the changes.

```
$ ansible RHEL -m command -a 'cat /etc/passwd'
$ ansible RHEL -m command -a 'sudo cat /etc/shadow'
```

But if we will not provide the command line variables then ansible will use playbook variable values.

```
$ ansible-playbook command_variable2.yml
```

Let's verify the changes.

```
$ ansible RHEL -m command -a 'cat /etc/passwd'
$ ansible RHEL -m command -a 'sudo cat /etc/shadow'
```

The passwords stored in the /etc/shadow file are not encrypted using these methods. They are stored in /etc/shadow file in plain text format which is not recommended.

Let's make appropriate changes in the playbook file, so that the passwords stored in the /etc/shadow file are encrypted.

```
$ vim command_variable2.yml
---
- name: Command Line Variable Example
  hosts: RHEL
  become: true
  vars:
        user_name: nehra
        password: redhat
  tasks:
       - name: Adding user and applying the password
         user:
              name: "{{ user_name }}"
              state: present
              password: "{{ password | password_hash ('sha512') }}"
...
```

Let's execute the playbook again and check the password.

```
$ ansible-playbook command_variable3.yml -e user_name=greta -e password=redhat
```

Let's verify the changes.

```
$ ansible RHEL -m command -a 'cat /etc/passwd'
$ ansible RHEL -m command -a 'sudo cat /etc/shadow'
```

Variable With Prompt:

There is a problem in the above-mentioned method, the user running this command must know the variables defined in the playbook and must mention the same on the command line.

If the user will use some other variable name playbook will not execute.

```
$ ansible-playbook command_variable.yml -e username=amit -e passwrd=redhat
```

The above command will not work and will throw an error, because of the use of different variables at the command line which are not present in the playbook file.

So, the best practice is to use the variable prompt option.

```
$ vim prompt.yml
---
- name: Command Line Variable With Prompt Example
  hosts: RHEL
  become: true
  vars_prompt:
```

```
            - name: user_name
              prompt: "please mention the username"
              private: no
            - name: password
              prompt: "please set the password"
              private: yes
  tasks:
      - name: Adding user and applying the password
        user:
            name: "{{ user_name }}"
            state: present
            password: "{{ password | password_hash ('sha512') }}"
...
```

$ Let's execute the playbook & test.

```
$ ansible-playbook prompt.yml
```

Let's verify the changes.

```
$ ansible RHEL -m command -a 'cat /etc/passwd'
$ ansible RHEL -m command -a 'sudo cat /etc/shadow'
```

(c) Inventory Variables:

You can store variable values that relate to a specific host or group in inventory. To start with, you may add variables directly to the hosts and groups in your main inventory file. We use inventory variables to use host specific values in ansible.

There are two categories of Inventory Variables:

i. *Host Scope Variables*
ii. *Group Scope Variables*

i. Host Scope Variables:

Let's create a playbook for creating the following directories on the managed nodes:
RHEL: /tmp/testdir
Ubuntu: /tmp/newdir

```
$ sudo cp /etc/ansible/hosts /etc/ansible/hosts.bkp
$ sudo vim /etc/ansible/hosts
[Ubuntu]
node2 dir=/tmp/newdir
```

```
[RHEL]
node1 dir=/tmp/testdir
node3 dir=/tmp/testdir
```

```
$ vim inventory_variable.yml
---
- name: Inventory Variable Example
  hosts: all
  become: true
  tasks:
        - name: Creating directories.
          file:
              path: "{{ dir }}"
              state: directory
        - debug:
              msg: "{{ dir }}"
...
```

Let’s execute the playbook and test the working.

```
$ ansible-playbook inventory_variable.yml
```

Let’s verify the changes.

```
$ ansible RHEL -m command -a 'ls -ld /tmp/testdir'
$ ansible Ubuntu -m command -a 'ls -ld /tmp/newdir'
```

ii. Group Scope Variables:

```
$ sudo vim /etc/ansible/hosts
[Ubuntu]
node2

[RHEL]
node1
node3

[RHEL:vars]
dir=/tmp/testdir1
```

Let’s execute the playbook and test the working.

```
$ ansible-playbook inventory_variable.yml
```

Let’s verify the changes.

```
$ ansible RHEL -m command -a 'ls -ld /tmp/testdir1'
```

Restore the original inventory file back.

```
$ sudo rm /etc/ansible/hosts
$ sudo mv /etc/ansible/hosts.bkp /etc/ansible/hosts
```

If there are a large number of variables present in your inventory file, it will not only make your inventory file larger and complex but it will create a lot of problems in managing the inventory file too.

To overcome this problem, we can create separate directories and files as shown below.

```
$ sudo mkdir /etc/ansible/host_vars
$ sudo mkdir /etc/ansible/group_vars
$ sudo vim /etc/ansible/host_vars/node2
dir: /tmp/newdir2
$ sudo vim /etc/ansible/group_vars/RHEL
dir: /tmp/testdir2
```

Let’s execute the playbook and test the working.

```
$ ansible-playbook inventory_variable.yml
```

Let’s verify the changes.

```
$ ansible RHEL -m command -a 'ls -ld /tmp/testdir2'
$ ansible node2 -m command -a 'ls -ld /tmp/newdir2'
```

Thanks