Ansible Plays and playbooks ,Conditional Execution

- **Know how to work with commonly used Ansible modules**
- **Use variables to retrieve the results of running a command**
- **Use conditionals to control play execution**
- **Configure error handling**
- **Create playbooks to configure systems to a specified state**

Jinja2 templating Language : Jijna2 Filters

While talking about **Ansible Variables**, We already used **Jinja2 templating language** to form dynamic expressions to reference and access variables in playbooks.

We used jinja2 expression **"{{ variable_name }}"** multiple times to access and reference variable and Ansible facts.

Also we used **jinja2 filter** e.g. **password_hash('sha512')** to encrypt passwords.

Templating happens on the Ansible controller before the task is sent and executed on the target machine. This is done to minimize the requirements on the target (jinja2 is only required on the controller) and to be able to pass the minimal information needed for the task, so the target machine does not need a copy of all the data that the controller has access to.

Now we will discuss about more **jinja2 filters.**

Jinja2 filters

Jinja2 filters to format data :

Using Jinja2 filter we can format data from one data structure to other e.g. from JSON to YAML.

"{{ ansible_facts | to_yaml }}"

For human readable output:

We can use : **"{{ ansible_facts | to_nice_yaml }}"**

Defaulting Undefined variable :

Using this filter ,we can print default value for some variable which does not exist. Jinja2 provides **default** filter for this purpose.

For example , If we will try to display size of some partition which does not exist, It will give some error. Using default filter ,We can display some default value like **Does not exist** or **NA** . So by using this filter, we can **configure error handling**.

Example:

"{{ ansible_facts['devices']['sda']['partitions']['sda12']['size'] | default('Block device does not exist') }}"

Jinja2 filters

List Filters:

To get minimum value of list of items :

"{{ list | min }}"

Example: "{{ [1, 2, 2, 3, 3, 4, 5] | min }}"

In similar way, to get maximum value :

"{{ [1, 2, 3, 3, 4, 5] | max }}"

To get unique set from a list :

"{{ [1, 2, 2, 3, 3, 4, 5] | unique }}"

There are many more jinja2 filters but for exam we don't need to know all of them. Most important filter is **default** filter which we will use for error handling which is on list of objectives of RHCE Exam.

Jinja2 template Statements : Conditionals

The When Statement:

We can use **when** clause when we need to skip a particular host for some task e.g. when we need to install some package on some nodes and want to skip some others.

We can do this using **when** statement using **raw jinja2 expression** (without curly braces).

For example, To display some variable on **mhost1** and skip for all others.

```
---
-
hosts: all
gather_facts: True
tasks:
        - name: Displaying variable
          debug:
                msg: "{{ ansible_facts['fqdn'] }}"
          when: ansible_facts['hostname'] == "mhost1"
...
```

We can specify multiple conditions that all need to be True(**Logical AND**).

```
---
-
hosts: all
gather_facts: True
tasks:
        - name: Displaying variable
          debug:
                msg: "{{ ansible_facts['fqdn'] }}"
          when:
                - ansible_facts['hostname'] == "mhost1"
                - ansible_facts['os_family'] == "RedHat"
...
```

We can specify multiple conditions using **logical OR**, where one condition needs to be True.

```
---
-
hosts: all
gather_facts: True
tasks:
        - name: Displaying variable
          debug:
                msg: "{{ ansible_facts['fqdn'] }}"
          when: ansible_facts['hostname'] == "mhost1" or ansible_facts['distribution_major_version'] == "7"
...
```

We can also combine multiple conditions using **logical AND** and **logical OR.**

```
---
-
hosts: all
gather_facts: True
tasks:
        - name: Displaying variable
          debug:
                msg: "{{ ansible_facts['fqdn'] }}"
          when: (ansible_facts['hostname'] == "mhost1" and ansible_facts['distribution_major_version'] == "8") or
(ansible_facts['hostname'] == "mhost2" and ansible_facts['distribution_major_version'] == "7")
…
```

Jinja2 template Statements : Loops

Loops are used to repeat some task multiple times based on some condition. Important example of using loop is to create multiple users with details defined as list items in some dictionary.

Ansible offers two keyworks to create loops : **loop** and **with_<lookup>.**

loop is best choice for simple loops, so most of times we will be using **loop** keyword for creating loops.

Standard loops using list of items:

Example:

```
---
-
 hosts: mhost1
 become: True
 tasks:
          - name: Adding users
            user:
                    name: "{{ item }}"
                    state: present
            loop:
                     - user1
                     - user2
...
```

Example: User details defined in file **details.yml** as **list of items within dictionary**.

```
---
-
 hosts: mhost1
 become: True
 vars_files: details.yml
 tasks:
          - name: Adding users
            user:
                    name: "{{ item.username }}"
                    state: present
            loop: "{{ users }}"
...
```

Sample file **details.yml**:

```
---
users:
          - username: user1
            department: HR
          - username: user2
            department: Testing
...
```

Jinja2 template Statements : Loops and Conditionals

For some cases we need to combine when statement with loop. For such cases when statement will be executed for each loop item.

Example:

```
---
-
 hosts: mhost1
 tasks:
          - name: Displaying item
            command: echo "{{ item }}"
            loop: [1, 2, 3, 4, 5, 6, 7, 8]
            when: item > 5
...
```

Example:

```
---
-
 hosts: mhost1
 become: True
 vars_files: details.yml
 tasks:
          - name: Adding users
            user:
                    name: "{{ item.username }}"
                    state: present
            loop: "{{ users }}"
            when: item.department == "HR"
...
```

Sample file details.yml:

```
---
users:
          - username: user1
            department: HR
          - username: user2
            department: Testing
...
```

Magic Variables : Information About Ansible + Remote Nodes

Information About Ansible inventory hosts and groups ,the directories for playbooks ,python version ,roles path etc. is available on Ansible control node and stored in special variables called **Magic variables.**

Magic variable names are reserved. We are not allowed to use variable with same names.

Below is list of commonly used **Magic variables**:

inventory_hostname : The inventory name for the 'current' host being iterated over in the play

ansible_play_batch : List of active hosts in the current play run. Failed/Unreachable hosts are not considered 'active'.

ansible_play_hosts: Same as ansible_play_batch

group_names: List of groups the current host is part of

groups: A dictionary/map with all the groups in inventory and each group has the list of hosts that belong to it.

hostvars :

A dictionary/map with all the hosts in inventory and variables assigned to them.

This variable stores information about all remote nodes(Ansible Facts) in dictionaries with same name as hostname and all other information about nodes related to ansible.

This is very useful variable that enables us to access and use variables of other remotes nodes on any node. We will use this variable to configure hosts file to serve as Local DNS.

We can display magic variables using **vars** variable. It stores all the information about Ansible operations and ansible facts of remote node(s) if **gather_facts** is set to True (Default).

“{{ vars }}” = Information stored in Ansible (Magic variables) + “{{ ansible_facts }}” for hosts with facts gathered

Task. Create a playbook with name 'condition.yml' to execute below tasks.

- Configure MOTD on **webservers** nodes as **"Welcome to webserver node\n".**
- Configure MOTD on **prod1** node as **"Welcome to mhost1\n".**

```
---
-
 hosts:
          - webservers
          - prod1
 become: True
 gather_facts: True
 tasks:
            - name: Configuring MOTD on webserver nodes
              copy:
                    content: "Welcome to webserver node\n"
                    dest: /etc/motd
              when: inventory_hostname in groups['webservers']

            - name: Configuring MOTD on mhost1
              copy:
                    content: "Welcome to mhost1\n"
                    dest: /etc/motd
              when: ansible_facts['hostname'] == 'mhost1'
...
```

Task. Create a playbook with name 'packages.yml' to execute below tasks.

- To install software package **samba** on **webservers nodes**.
- To install software package **nfs-utils** on **prod nodes**.

```
---
-
 hosts: all
 become: True
 gather_facts: False
 tasks:
            - name: Installing package samba on webservers nodes
              yum:
                    name: samba
                    state: present
              when: inventory_hostname in groups['webservers']

            - name: Installing package nfs-utils on prod nodes
              yum:
                    name: nfs-utils
                    state: present
              when: inventory_hostname in groups[ 'prod']
...
```

Task. Create a playbook with name 'firewall_config.yml' to execute below tasks.

- Configure **webservers nodes** to accept inbound traffic for **ntp** and **https** services.
- Configure **prod nodes** to accept traffic on port range **400-404/tcp**.
- Firewall configs should be persistent. Reload firewall to make changes effective.

```
---
-
 hosts: all
 become: True
 gather_facts: False
 tasks:
          - name: Configuring firewall on webservers nodes
            firewalld:
                  service: "{{ item }}"
                  state: enabled
                  permanent: yes
            loop:
                    - ntp
                    - https
            when: inventory_hostname in groups['webservers']
            notify: Reload firewall
          - name: Configuring firewall on prod nodes
            firewalld:
                  port: 400-404/tcp
                  state: enabled
                  permanent: yes
            when: inventory_hostname in groups[ 'prod']
            notify: Reload firewall
 handlers:
          - name: Reload firewall
            service:
                  name: firewalld
                   state: reloaded
...
```

Task. Create a YAML file with name 'userdetails.yml' to contain below information.

- Define List variable with name **users** to define list of dictionary items under this.
- Each list item defines **username ,department and age** of user as per below table.
- Also create one more YAML file **password.yml** to store password information only about users as **key: value** pairs.

username	department	age	key: password
lisa	Software developer	32	lisa_password: lisa_pass
bob	Testing	38	bob_password: bob_pass
lara	HR	28	lara_password: lara_pass

userdetails.yml

```
---
users:
        - username: lisa
          department: Software developer
          age: 32
        - username: bob
          department: Testing
          age: 38
        - username: lara
          department: HR
          age: 28
…
```

password.yml

```
---
lisa_password: lisa_pass
bob_password: bob_pass
lara_password: lara_pass
…
```

Task. Create a playbook 'create_users.yml' with below conditions.

- Create user on **webservers nodes** when user's department is **Software developer** and assign supplementary group **testing** to user.
- Create user on **prod nodes** when user's department is **Testing** and assign supplementary group **networks** to user.
- Create user on **all managed nodes** when user's department is HR.
- Set password for different users as defined in **password.yml** file and password must be encrypted.

```
---
 hosts: all
 become: True
 gather_facts: False
 vars_files:
                - userdetails.yml
                - password.yml
 tasks:
         - name: Creating user on webservers nodes
           user:
                name: "{{ item.username }}"
                password: "{{ lisa_password | password_hash('sha512') }}"
                groups: testing
           when: "item.department == 'Software developer' and inventory_hostname in groups['webservers']"
           loop: "{{ users }}"
         - name: Creating user on prod nodes
           user:
                name: "{{ item.username }}"
                password: "{{ bob_password | password_hash('sha512')  }}"
                groups: networks
           when: "item.department == 'Testing' and inventory_hostname in groups['prod']"
           loop: "{{ users }}"
         - name : Creating user on all managed nodes
           user:
                name: "{{ item.username }}"
                password: "{{ lara_password | password_hash('sha512') }}"
           when: item.department == 'HR'
           loop: "{{ users }}"
...
```

Task. Create a playbook 'vgroup.yml' to create volume group.

- Create logical partition of size **1 GiB(1024 MiB)** on all **webservers** nodes and of size **600 MiB** on **prod1** node.
- Create volume group with name **vgroup** using this partition.

Note: We must install package **lvm2** on all Managed nodes before starting this task. Package is missing because we installed Managed nodes with Minimum install option.

```
---
-
 hosts: webservers,prod1
 become: True
 gather_facts: True
 tasks:
          - name: Read device information
            parted: device=/dev/sda unit=MiB
            register: sda_info
          - name: Creating logical partition
            parted:
                        device:  /dev/sda
                        number: “6”
                        part_type: logical
                        part_start: “{{ sda_info.partitions[4].end  + 1 }}MiB”
                        part_end: “{{ sda_info.partitions[4].end  + 1025 }}MiB”
                        flags: [ lvm ]
                        state: present
            when: inventory_hostname in groups[‘webservers’]
          - name: Creating logical partition
            parted:
                        device:  /dev/sda
                        number: “6”
                        part_type: logical
                        part_start: “{{ sda_info.partitions[4].end  + 1 }}MiB”
                        part_end: “{{ sda_info.partitions[4].end  + 601 }}MiB”
                        flags: [ lvm ]
                        state: present
            when: inventory_hostname in groups[‘prod1’]
          - name: Creating volume group
            lvg:
                  vg: vgroup
                  pvs: /dev/sda6
…
```

Task. Create a playbook 'logvol.yml' to create logical volume with name 'lvm' on managed nodes meeting below conditions.

- Create logical volume of size **800 MiB** if volume group **vgroup** has sufficient free space(greater than 800 MiB).
- Create logical volume of size **500 MiB** if volume group has free space less than **800 MiB.**
- Message **'Vol group does not exist'** should be displayed if volume group is not created on node.

```
---
-
 hosts: all
 become: Yes
 gather_facts: True
 tasks:
        - name: Creating logical volume of size 800 MiB
          lvol:
                vg: vgroup
                lv: lvm
                size: 800m
          when: "'vgroup' in ansible_facts['lvm']['vgs'] and ansible_facts['lvm']['vgs']['vgroup']['free_g'] > '0.79'"
        - name: Creating logical volume of size 500 MiB
          lvol:
                vg: vgroup
                lv: lvm
                size: 500m
          when: "'vgroup' in ansible_facts['lvm']['vgs'] and ansible_facts['lvm']['vgs']['vgroup']['free_g'] < '0.79'"
        - name: Displaying message when Volume group does not exist
          debug:
                msg: "Vol group does not exist"
          when: "'vgroup' not in ansible_facts['lvm']['vgs']"
...
```

Task. Create a playbook 'volume.yml' to create logical volume with name 'vol' on managed nodes meeting below conditions.

- Create logical volume to use all remaining size on volume group **vgroup.**
- Message **'VG does not exist'** should be displayed if volume group is not created on node.

```
---
-
 hosts: all
 become: Yes
 gather_facts: True
 tasks:
          - name: Creating logical volume
            lvol:
                  vg: vgroup
                  lv: vol
                  size: 100%FREE
            when: "'vgroup' in ansible_facts['lvm']['vgs']"
          - name: Displaying message
            debug:
                  msg: "VG does not exist"
            when: "'vgroup' not in ansible_facts['lvm']['vgs']"
...
```

Task. Create a playbook 'mount_vol.yml' to mount logical volume 'vol' created in previous task.

- Create filesystem **xfs** on volume.
- Use mount point as **/volume/lvm and** mount should be persistent.

```
---
-
 hosts: all
 become: Yes
 gather_facts: True
 tasks:
        - name: Creating filesystem
          filesystem:
                dev: /dev/vgroup/vol
                fstype: xfs
          when: "'vol' in ansible_facts['lvm']['lvs']"
        - name: Mounting logical volume
          mount:
                src: /dev/vgroup/vol
                path: /volume/lvm
                fstype: xfs
                state: mounted
          when: "'vol' in ansible_facts['lvm']['lvs']"
...
```