Understanding Core Components of Ansible

Introducing Ansible core components

- **Ansible Configuration Files** - Defines various default values to be used by Ansible. Almost all default values can be overwritten in ansible ad-hoc commands and playbooks.
- **Ansible Inventories** - Defines hosts and groups of hosts on which ansible operates.
- **Ansible Modules** - Module is small python program which is designed to execute specific task on ansible managed nodes or local node.
- **Ansible Variables** - Variables used in playbook .
- **Ansible Facts** - Represent data about remote systems gathered by Ansible which can be used in playbooks as variables. Ansible facts are also very important for conditional playbook execution.
- **Ansible Plays** - Ansible play basically defines target hosts and specific task(s) to be executed on target hosts. Ansible play is written in YAML.
- **Ansible Playbooks** - List of ansible plays

Ansible Config file

Different Ansible settings can be configured using Ansible configuration file named as **ansible.cfg**.

When installing ansible using package manager (**dnf or yum**) , configuration file is found at path **/etc/ansible** which is system wide directory.

Ansible configuration file can be present at multiple locations as described below and they are searched in same order.

ANSIBLE_CONFIG	Environment Variable if set
ansible.cfg	In current working Directory
~/.ansible.cfg	User's home directory
/etc/ansible/ansible.cfg	System wide directory

Ansible searches for configuration file in the above order. Config file found first is considered and others are simply ignored.

Some Important Default Settings of Ansible

Below is list of important defaults which are used by Ansible if we don't define them in **ansible.cfg** file considered by Ansible.

[Defaults]

Directive Name	**Default Value/Behavior**	**Remarks**
inventory	/etc/ansible/hosts	Ansible looks for this **default inventory file** if **inventory** directive is not used in config file to set some other path.
remote_port	22	By default, **SSH port 22** is used to connect to remote nodes unless **remote_port** directive is set to some other port in config file.
forks	5	By default , Ansible starts 5 forks unless **forks** directive is set to some other value.
roles_path	/etc/ansible/roles	Ansible looks for roles in this directory if **roles_path** directive is not used in config file to define some other roles path(s).
gathering	implicit	By default, Facts are always gathered unless we set this directive to **explicit** in config file.
gather_subset	all	By default, **all facts** (All subsets) are gathered.
module_name	command	Default module is **command** module unless we set this directive to some other module.

[privilege_escalation]

Directive Name	**Default Value/Behavior**	**Remarks**
become	False	By Default, privilege escalation is disabled unless we set **become** directive to **yes or true** (Yes or True).
become_method	sudo	By default, **become_method** is **sudo**.
become_user	root	By default , privileges are escalated to **root user**.
become_ask_pass	False	By default, Ansible does not ask for **privilege escalation password** in case it is needed (case where privilege escalation is possible only with password)

Example of ansible config file (ansible.cfg)

We can define different settings using ansible config file. For example,

- Location of ansible inventory file
- Location of roles directory
- User to be used to connect to remote machines.
- Enabling/Disabling privilege escalation and many more settings....

Sample config file:

```
vim /home/ansible/tasks/ansible.cfg
[defaults]
inventory = /home/ansible/tasks/nodes
roles_path = /home/ansible/tasks/roles
remote_user = ansible

[privilege_escalation]
become = yes
become_user = root
become_method = sudo
become_ask_pass = False
```

Ansible Inventories

- Ansible inventory file defines the hosts to be managed by ansible. Different managed hosts can be grouped together in different groups.
- Ansible ad-hoc commands and playbooks are executed on inventory hosts defined on command line and in playbook, respectively.
- Host and group specific variables are also defined in inventory file.
- Ansible inventory file default path is defined in ansible configuration file.
- Ansible inventory file can be defined in INI,YAML or JSON format. Most used format is INI format.
- Ansible inventory file name is not a standard name, you can define any file in ansible config file to be used as ansible inventory e.g. **hosts** or **inventory** or **mhosts**.

Example of Ansible Inventory file in INI format

vim /home/ansible/tasks/nodes

```
[hgroup1]
host1
host2
192.168.99.10
host1.example.com
host[3:5]

[hgroup2]
webserver.example.com
db.example.com
```

Ansible Host and Group variables(Example)

vim /home/ansible/tasks/nodes

```
[hgroup1]
host1 ansible_user=ansible ansible_port=555
host2
192.168.99.10
host1.example.com
host[3:5]

[hgroup2]
webserver.example.com
db.example.com

[hgroup2:vars]
ansible_user=ansible
```

Organizing Host and Group Variables

In previous lecture, We understood how we can specify host and group vars in inventory file. In this lecture we will learn how we can organize host and group variables by putting them in separate files for specific host and group.

Organizing variables is a cleaner approach when you have many hosts (or host groups) in inventory file.

To organize them, You must create directories with name **host_vars** and **group_vars** at the same path where inventory file is present.

To define host vars :

Create file with same as that of host (for example **mhost1**) under **host_vars** directory and specify host vars specific to that host in this file.

For Example:

vim /etc/ansible/hosts/**host_vars/mhost1** **(Valid extensions are .yml,.yaml, .json and no extension)**

```
---
ansible_user: ansible
ansible_port: 555
:wq
```

To define group vars:

Create file with same name as that of group name and specify group vars in this file.

Note: You must create directories with standard names **host_vars** and **group_vars** to specify host and group specify variables.

Also, you must create these directories at path where inventory file is present.

Ansible Modules

Modules (also referred to as “task plugins” or “library plugins”) are discrete units of code that can be used from the command line or in a playbook task. Ansible executes each module, usually on the remote target node, and collects return values.

Most used modules are:

- ping
- setup
- yum
- yum_repository
- service
- package
- systemd
- cron
- lvg
- lvol
- parted
- template
- fetch

Ansible Modules

- register
- user
- group
- authorized_key
- lineinfile
- firewalld
- nmcli
- sefcontext
- selinux
- command
- file
- filesystem
- mount
- debug

And many more……

Ansible ad-hoc Commands

An ad-hoc command is nothing but command we type on terminal to do some action quickly and usually once. We don't want to save for later. For example -To Shutdown/Restart remote host or to quickly verify some configuration.

Syntax of ad-hoc command is :

ansible [pattern] -m [module] -a "[module options]"

Examples of some ad-hoc commands are shown below:

ansible mhost1 -m command -a "cat /etc/hosts"	To display **hosts** file of **mhost1**
ansible mhost2 -m copy -a "src=/etc/hosts dest=/tmp/hosts_bkp"	To copy **hosts** file to **/tmp/hosts_bkp** file on **mhost2**
ansible mhost3 -m file -a "path=/tmp/test state=touch"	To create **test** file on **mhost3**
ansible mhost4 -m user -a "name=ansible state=present"	To create user **ansible** on **mhost4**

We must keep in mind ad-hoc command would be successfully executed if user we are using has permissions to perform that action. For example, normal user cannot create user ,so we must execute command to create user as root user.

If default **remote_user** is normal user as defined in **ansible.cfg**, the use --**become** flag to execute command on remote host with **sudo** provided remote host must be configured to allow to use **sudo** for this user.

ansible mhost4 -m user -a "name=ansible state=present" --become

Ansible plays and playbooks

At basic level, Ansible playbook describes some configuration steps that need to be enforced on remote host.

Ansible playbooks are written in YAML format which is human read able and this makes ansible easy to use and understand than other programming languages.

We can think of modules as tools ,playbooks as instruction manuals and inventory hosts as raw material.

So essentially a playbook contains list of plays where each play defines instructions to perform some task on remote host.

We can write a playbook comprising different plays, each play with different target(s) and set of tasks.

A play may contain below sections :

hosts	To define target of play and other directives like remote_user ,become, gather_facts etc
vars	To define variables
tasks	To define list of tasks to be executed
handlers	To define some task to be executed on successful change due to some other task in tasks section
roles	Roles to be included in the play

During this course , we will create ansible playbooks to configure remote systems, create users ,installing packages etc.

Ansible Playbook format example in YAML

```
---
- hosts: mhost1
  become: True
  tasks:
          - name: Adding firewall rule
            firewalld:
                    port: 80/tcp
                    state: enabled
                    permanent: yes
            notify: Reload firewall
  handlers:
          - name: Reload firewall
            systemd:
                   name: firewalld
                   state: reloaded
...
```

Ansible Variables

- Ansible playbooks are written in YAML. YAML supports dictionary variables, list variables and complex variables using dictionaries and lists both.
- Valid variable names should start with letter and can contain only underscore.
- Dictionary variable maps keys to values.

Dictionary variable example:

```
vars:
    key: value
```

We can use defined variables in playbooks using **Jinja2 templating system**.

For example, we can reference the variable defined above in playbook with jinja2 expression **"{{ key }}".**

We will display the variable using **debug** module in simple playbook to understand this.

We will discuss more about jinja2 expressions for conditionals in separate section which is very important topic from exam point of view.

Ansible Variables

Named Dictionary variable example:

```
vars:
    dict:
        key1: value1
        key2: value2
```

Jinja2 expressions to reference variable in playbook.

“{{ dict }}” - To display dictionary variable.

“{{ dict.key1 }}” - To display value mapped to key1 using dot notation.

“{{ dict[‘key1’] }}” - To display value mapped to key1 using bracket notation.

We will display the variable using **debug** module in simple playbook to understand this.

Ansible Variables

Named List variable example:

```
vars:
    list:
        - item1
        - item2
        - item3
```

Jinja2 expressions to reference variable in playbook.

"{{ list }}" - To display list variable.

"{{ list[0] }}" - To display first item in the list.

"{{ list[1] }}" - To display second item in the list.

We will display the variable using **debug** module in simple playbook to understand this.

Ansible Variables

Dictionary+list mixed variable example:

```
vars:
     users:
          - name: mark
            age: 35
          - name: lisa
            age: 30
```

Jinja2 expressions to reference variable in playbook.

“{{ users }}” - To display list variable.

“{{ users[0] }}” - To display first item in the list.

“{{ users[0].name }}” - To display name in first list item.

“{{ users[0][‘name’] }}” - To display name in first list item.

We will display the variable using **debug** module in simple playbook to understand this.

Ansible Variables

Variables defined in YAML file:

```
 vars_files:
        - vars_file.yml


vim vars_file.yml
---
users:
        - name: mark
          age: 35
        - name: lisa
          age: 30
...
```

Jinja2 expressions to reference variable in playbook.

“{{ users }}” - To display list variable.

“{{ users[0] }}” - To display first item in the list.

“{{ users[0].name }}” - To display name in first list item.

We will display the variable using **debug** module in simple playbook to understand this.

Ansible Variables

Variable provided during playbook execution:

```
vars_prompt:
            - name: var
              prompt: Enter variable
              private: no
```

Jinja2 expressions to reference variable in playbook.

"{{ var }}" - To display variable.

We will display the variable using **debug** module in simple playbook to understand this.

Ansible facts

Ansible facts are data/information related to your remote systems like operating systems, IP addresses assigned on different interfaces, hostname, disks, filesystems and more.

You can access this data in the **ansible_facts** variable. By default, you can also access some Ansible facts as top-level variables with the **ansible_ prefix.**

We can display ansible facts using setup module using ansible ad-hoc command:

ansible managed_host -m setup

Please keep in mind normal users can not display complete information of remote host(s). Execute command as **root** user to display all information about remote systems for example information about disks.

Filtering Ansible facts

In previous lecture, We displayed Ansible facts of remote node, but we did not use any filtering options .To filter Ansible Facts ,We can use **gather_subset** and **filter** parameters. We can use them using **-a option** with setup module.

Examples of using 'gather_subset' directive

To gather Network related facts:	ansible target_host(s) -m setup -a **"gather_subset=network"** --become
To gather hardware related facts:	ansible target_host(s) -m setup -a **"gather_subset=hardware"** --become
To gather Virtual environment related facts:	ansible target_host(s) -m setup -a **'gather_subset=virtual,!min'** --become

Default is all (All facts).

To gather facts except network facts:	ansible target_host(s) -m setup -a **'gather_subset=!network'** --become

Examples of using 'filter' directive

ansible target_host(s) -m setup -a **"filter=ansible_devices"** --become

ansible target_host(s) -m setup -a **"filter=ansible_lvm"** --become

ansible target_host(s) -m setup -a **"filter=ansible_*name"** --become

Note 1: When using **! Symbol** to filter facts, always use single quotes ('single quotes') . With double quotation marks it will not work.

Note 2: Minimum default facts are always gathered with each subset of facts until we exclude them (!min)

Ansible Documentation

Ansible documentation is available on website **https://docs.ansible.com/**

Using **ansible-doc** command line tool, We can conveniently check information about modules ,various directives that can be used and with some playbook examples.

For example, To display all information about **copy** module, execute below command.

ansible-doc copy

In similar way you can check information about other modules, but you must know exact name of module.

To display all modules ,execute **ansible-doc -l**