Ansible & System Automation

Ansible concepts
These concepts are common to all uses of Ansible. You need to understand them to use Ansible for any kind of automation. This basic introduction provides the background you need to follow the rest of the User Guide.

Control node
Any machine with Ansible installed. You can run commands and playbooks, invoking /usr/bin/ansible or /usr/bin/ansible-playbook, from any control node. You can use any computer that has Python installed on it as a control node - laptops, shared desktops, and servers can all run Ansible. However, you cannot use a Windows machine as a control node. You can have multiple control nodes.

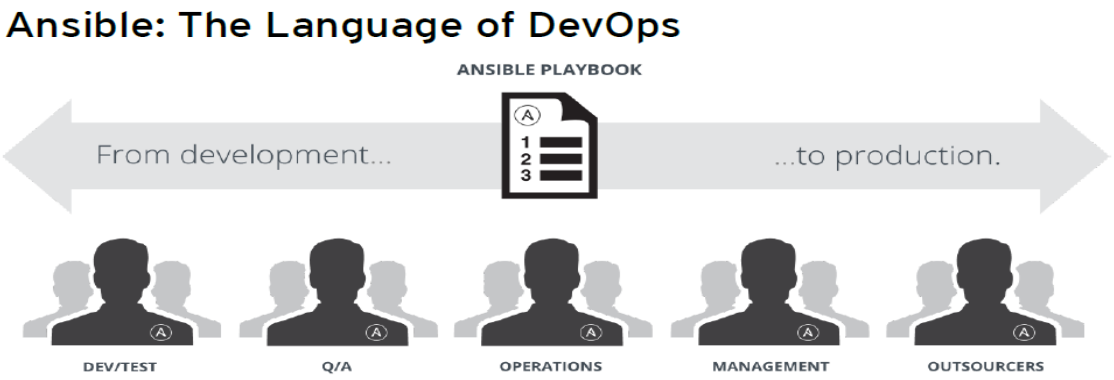
Managed nodes
The network devices (and/or servers) you manage with Ansible. Managed nodes are also sometimes called "hosts". Ansible is not installed on managed nodes.

Inventory
A list of managed nodes. An inventory file is also sometimes called a "hostfile". Your inventory can specify information like IP address for each managed node. An inventory can also organize managed nodes, creating and nesting groups for easier scaling. To learn more about inventory, see the Working with Inventory section.

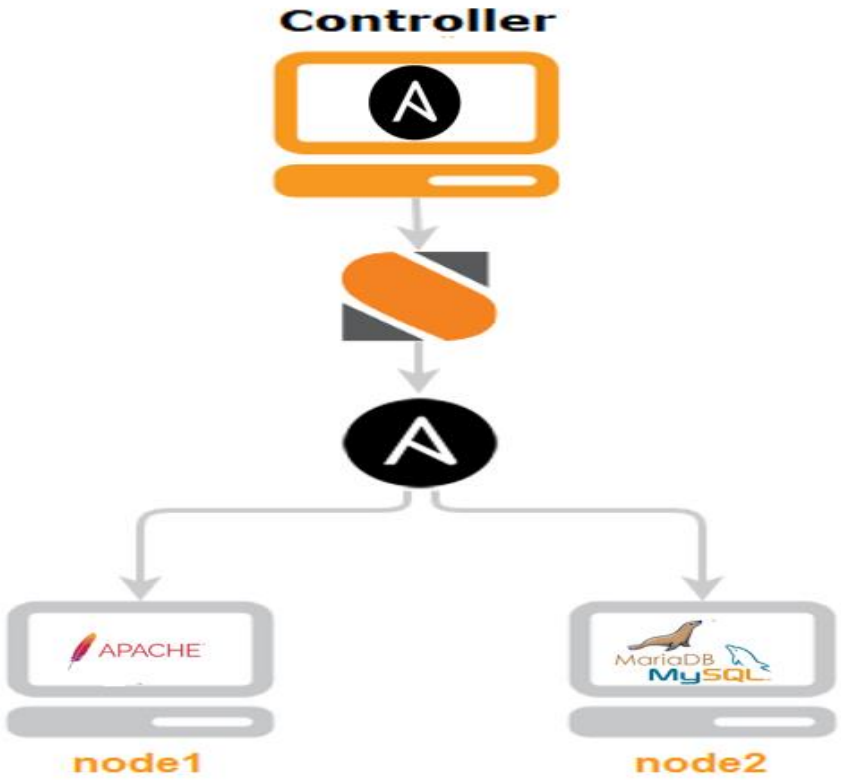
Modules
The units of code Ansible executes. Each module has a particular use, from administering users on a specific type of database to managing VLAN interfaces on a specific type of network device. You can invoke a single module with a task or invoke several different modules in a playbook. For an idea of how many modules Ansible includes, look at the list of all modules.

Tasks
The units of action in Ansible. You can execute a single task once with an ad-hoc command.

Playbooks
Ordered lists of tasks, saved so you can run those tasks in that order repeatedly. Playbooks can include variables as well as tasks. Playbooks are written in YAML and are easy to read, write, share and understand. To learn more about playbooks, see About Playbooks.



Our Objective:



Configuring Connections:

Ansible needs to know how to communicate with its managed hosts. One of the most common reasons to change the configuration file is in order to control what methods and users Ansible will use to administer managed hosts. Some of the information needed includes:

- Where the inventory is that lists the managed hosts and host groups
- Which connection protocol to use to communicate with the managed hosts (by default, SSH),and whether a non-standard network port is needed to connect to the server.
- Which remote user to use on the managed hosts; this could be **root** or it could be an unprivileged user
- If the remote user is unprivileged, Ansible needs to know whether it should try to escalate privileges to **root** and how to do it (for example, by using **sudo**).
- Whether or not to prompt for an SSH password or **sudo** password to log in or gain privileges

Privilege Escalation:

For security and auditing reasons, Ansible might need to connect to remote hosts as a nonprivileged
user before escalating privileges to get administrative access as **root**. This can be set up in the **[privilege_escalation]** section of the Ansible configuration file.

To enable privilege escalation by default, set the directive **become = true** in the configuration file. Even if this is set by default, there are various ways to override it when running ad hoc commands or Ansible Playbooks. (For example, there might be times when you want to run a task or play that does not escalate privileges.)

The **become_method** directive specifies how to escalate privileges. Several options are available, but the default is to use **sudo**.

Likewise, the **become_user** directive specifies which user to escalate to, but the default is **root**.

If the **become_method** mechanism chosen requires the user to enter a password to escalate privileges, you can set the **become_ask_pass = true** directive in the configuration file.

The following example **ansible.cfg** file assumes that you can connect to the managed hosts as **someuser** using SSH key-based authentication, and that **someuser** can use **sudo** to run commands as **root** without entering a password:

```
[defaults]
inventory = ./inventory
remote_user = someuser
ask_pass = false

[privilege_escalation]
become = true
become_method = sudo
become_user = root
become_ask_pass = false
```

Environment Creation:

```
# yum install net-tools vim bash-completion wget -y
# hostnamectl set-hostname ansible-master
# vim /etc/hosts
        192.168.0.104 ansible-master
        192.168.0.105 ansible-worker1
        192.168.0.106 ansible-worker2
 # cat /etc/hosts
 # ifconfig
```

Ansible Installation & Validation:

```
# yum install epel-release -y
# yum install python -y
# yum install ansible -y
# yum list installed python
# yum list installed ansible
# ansible --version
```

```
# grep "^\[" /etc/ansible/ansible.cfg
[defaults]
[inventory]
[privilege_escalation]
[paramiko_connection]
[ssh_connection]
[persistent_connection]
[accelerate]
[selinux]
```

- Most of the settings in the configuration file are grouped under the **[defaults]** section.
- The **[privilege_escalation]** section contains settings for defining how operations that require escalated privileges are executed on managed hosts.
- The **[paramiko_connection]**,**[ssh_connection]**, and **[accelerate]** sections contain settings for optimizing connections to managed hosts.
- The **[selinux]** section contains settings for defining how SELinux

Validation of name Resolution:

```
# ping anasible-worker1
# ping ansible-worker1
# ping ansible-worker2
# ping ansible-master
```

Password Less SSH Login:

```
# ssh-keygen
# ssh-copy-id root@ansible-worker1
# ssh-copy-id root@ansible-worker2
# ssh-copy-id localhost
# ssh ansible-worker1
# ssh ansible-worker2
```

Inventory File:

```
# cd /etc/ansible/
# vim host
```

```
[myself]
 localhost

 [webservers]
ansible-worker1

 [dbservers]
```

```
 ansible-worker2

[infrastructure:children]
webservers
dbservers
myself
```

Host Checking with ansible command:

```
# ansible myself --list-hosts
# ansible webservers --list-hosts
# ansible dbservers --list-hosts
# ansible infrastructure --list-hosts
```

Ping validation with Ansible command:

```
# ansible dbservers -m ping
# ansible webservers -m ping
# ansible myself -m ping
# ansible infrastructure -m ping
```

Ansible Ad-Hoc Command validation:

```
# ansible webservers -m command -a "cat /etc/hosts"
# ansible dbservers -m command -a "cat /etc/hosts"
# ansible localhost -m command -a "cat /etc/hosts"
# ansible infrastructure -m command -a "cat /etc/hosts"

# ansible webservers -m command -a "df -HT"
# ansible dbservers -m command -a "df -HT"
# ansible localhost -m command -a "df -HT"
# ansible infrastructure -m command -a "df -HT"

# ansible webservers -m command -a "free -mh"
# ansible dbservers -m command -a "free -mh"
# ansible localhost -m command -a "free -mh"
# ansible infrastructure -m command -a "free -mh”

# ansible webservers -m command -a "getenforce"
# ansible dbservers -m command -a "getenforce"
# ansible localhost -m command -a "getenforce "
# ansible infrastructure -m command -a "getenforce "

# ansible webservers -m command -a "ifconfig"
# ansible dbservers -m command -a "cat/etc/motd"
```

Input Test in a file by Ansible Ad-HOC command:

```
# ansible infrastructure -b -m lineinfile -a 'dest=/etc/motd line="This server is managed by Ansible"'

# ansible infrastructure -m command -a "cat /etc/motd"
```

Directory Create by Ansible Ad-HOC command:

```
# ansible webservers -m file -a "dest=/opt/devops/ state=directory"
```

==========

LAB: Web Server Installation and configuration by ansible playbook.

vim webserver.yaml

```
---
- name: play to setup web server
  hosts: webservers
  tasks:
  - name: latest httpd version installed
    yum:
     name: httpd
     state: latest

  - name: httpd is started
    service:
     name: httpd
     state: started
     enabled: true

  - name: test html page is installed
    copy:
     content: "Welcome to the example.com intranet!\n"
     dest: /var/www/html/index.html
  - name: Enabled in firewalld
    firewalld:
     service: http
     permanent: true
     state: enabled
     immediate: yes
...
```

```
[root@ansible-master ansible]# cat webserver.yml
---
- name: play to setup Apache web server
  hosts: webservers
  tasks:
  - name: latest httpd version installed
    yum:
     name: httpd
     state: latest

  - name: httpd is started
    service:
     name: httpd
     state: started
     enabled: true

  - name: test html page is installed
    copy:
     content: "Welcome to the example.com intranet!\n"
     dest: /var/www/html/index.html
  - name: Enabled in firewalld
    firewalld:
     service: http
     permanent: true
     state: enabled
     immediate: yes
...
```

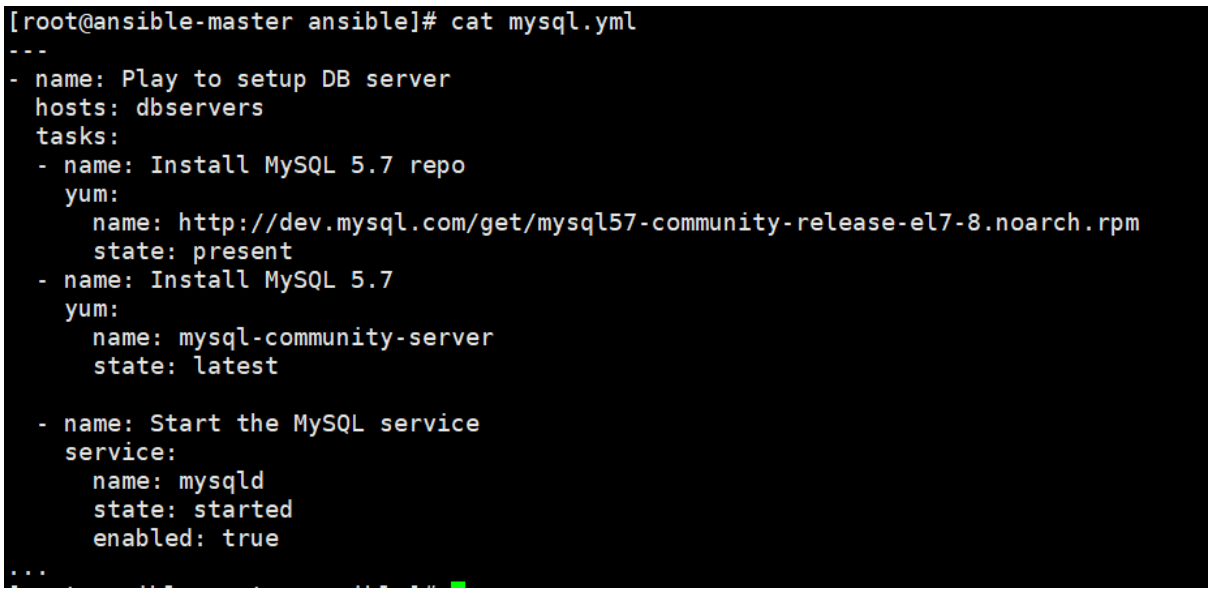
ansible-playbook --syntax-check webserver.yml
ansible-playbook -C webserver.yml
ansible-playbook webserver.yaml

LAB: - Database Server Installation and configuration by ansible playbook.

vim mysql.yml

```
---
- name: Play to setup DB server
  hosts: dbservers
  tasks:
  - name: Install MySQL 5.7 repo
    yum:
      name: http://dev.mysql.com/get/mysql57-community-release-el7-
8.noarch.rpm
      state: present
  - name: Install MySQL 5.7
    yum:
      name: mysql-community-server
      state: latest

  - name: Start the MySQL service
    service:
      name: mysqld
      state: started
      enabled: true
...
```



ansible-playbook --syntax-check mysql.yml
ansible-playbook -C mysql.yml
ansible-playbook mysql.yaml