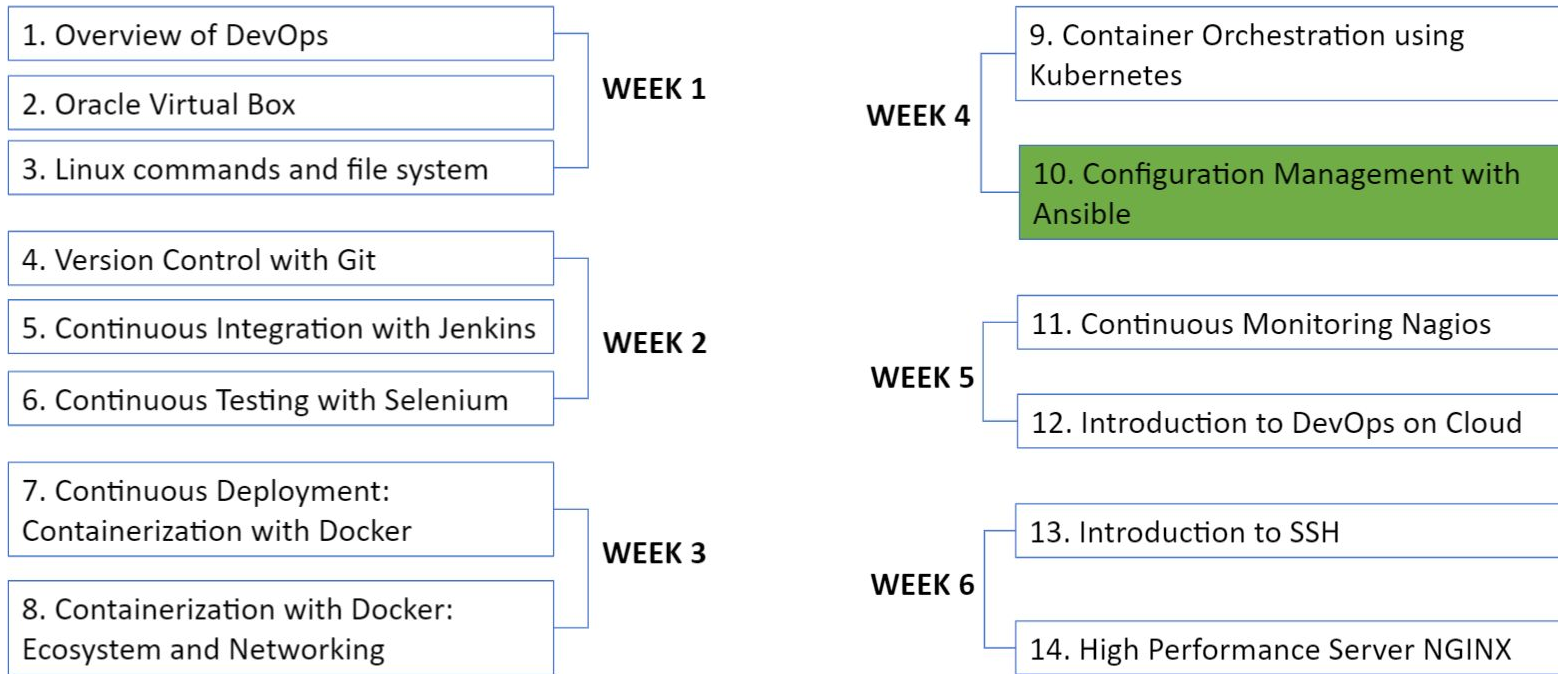


A N S I B L E

Configuration Management with Ansible

Week 4

Course Outline
Module 10



Topics

- Introduction to Ansible
- Ansible Architecture
- Setting up Ansible
- Ad-hoc commands in Ansible
- Ansible Playbook
- Roles in Ansible
- Variables in Ansible
- Deploy a Docker Container



Objective

Upon completion of this module, you should be able to:

- Install Ansible on your machine
- Understand Ansible architecture
- Write Ansible playbooks
- Execute different ad-hoc commands using Ansible
- Implement a project using Ansible



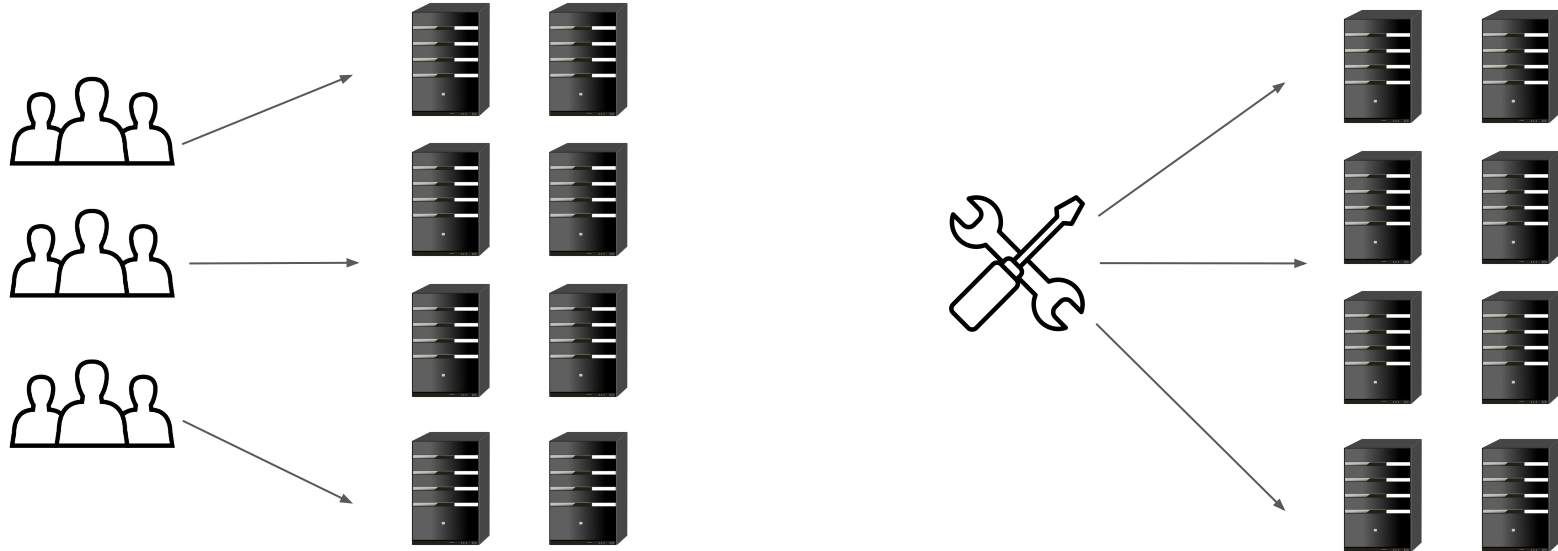
What is a Configuration Management System?

- Configuration management is a systems engineering process for establishing consistency of a product's attributes throughout its life.
- In the technology world, configuration management is an IT management process that tracks individual configuration items of an IT system. An IT asset may represent a piece of software, or a server, or a cluster of servers.

Why is configuration management important?

- Configuration management helps engineering teams build robust and stable systems through the use of tools that automatically manage and monitor updates to configuration data.
- When managing hundreds or thousands of servers consisting of different types, groups and flavours, it is essential to maintain consistency of their respective environments.
- It is extremely difficult, inaccurate, expensive and time consuming to maintain the state of the server individually. However collectively with the use of a Configuration management tool it is easy to maintain a healthy state of all servers.

Configuration Management

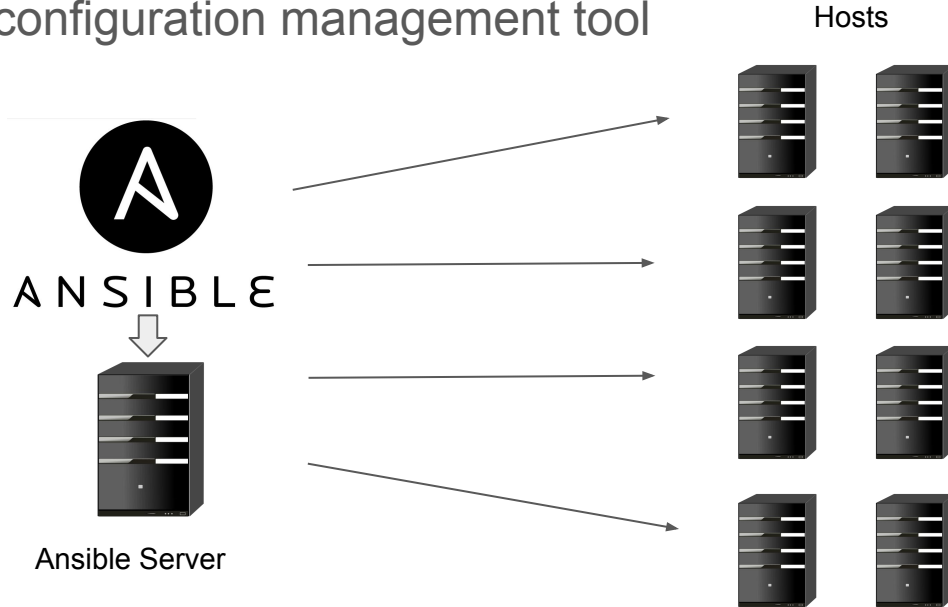


Configuration Management Tools



What is Ansible

Ansible is a deployment automation tool which uses push approach to achieve its objective, by managing all the servers through one single machine running the Ansible configuration management tool



How is Push different from Pull Approach



PUSH	PULL
Push approach does not require agents set up on individual nodes like pull does	Pull works in a master slave architecture which requires agents set up on all slave nodes
Push based systems are completely Synchronous as you can see the changes made instantaneously and can fix the system if changes cause problems	Systems using pull architecture can scale quite easily which is not the case with push model

Why Ansible?

Simple

Ansible uses YAML(Yet Another Mark-up Language) which is very simple and readable. No special coding skills are required to write in YAML



Agentless

Ansible uses OpenSSH and WinRM as transport to achieve automation. It is completely agentless, therefore making it a more efficient and secure option.



Powerful

Ansible can be used to manage entire infrastructure of an application. It can orchestrate the entire application and environment lifecycle no matter where it is deployed.



Accessible

If someone automates something in Ansible, everyone on the team now knows how it is done.



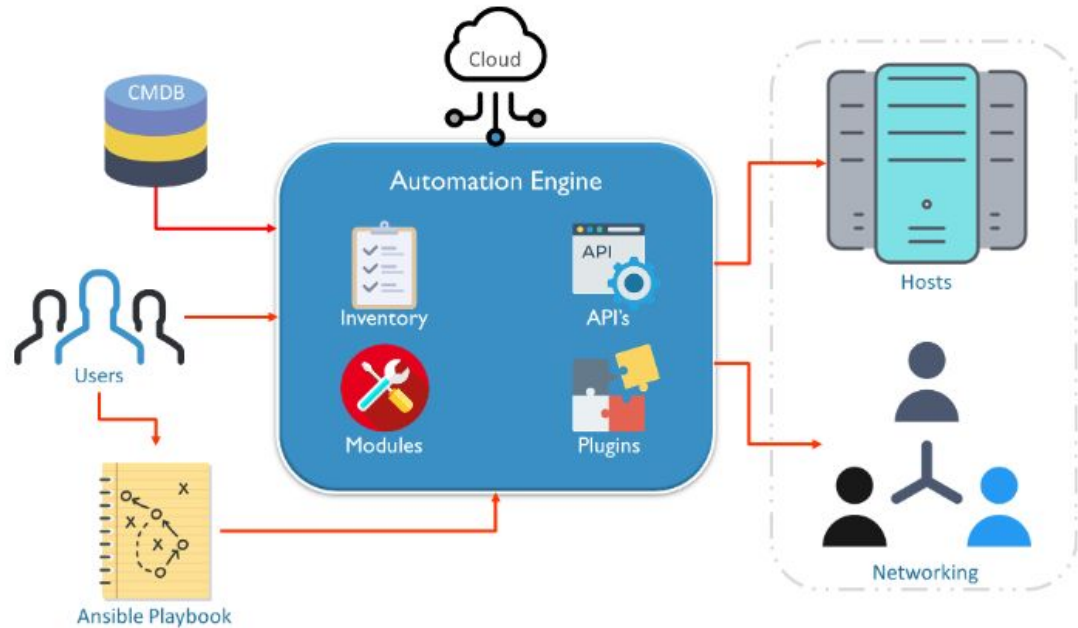
Ansible Vs Other Tools

- Ansible is much easier to work with compared to other tools like Chef, Puppet and Saltstack
- Ansible does not require Agent to be setup on individual hosts or nodes
- Ansible is simple enough for new users and integrates with other tools
- Ansible also support pull architecture if required

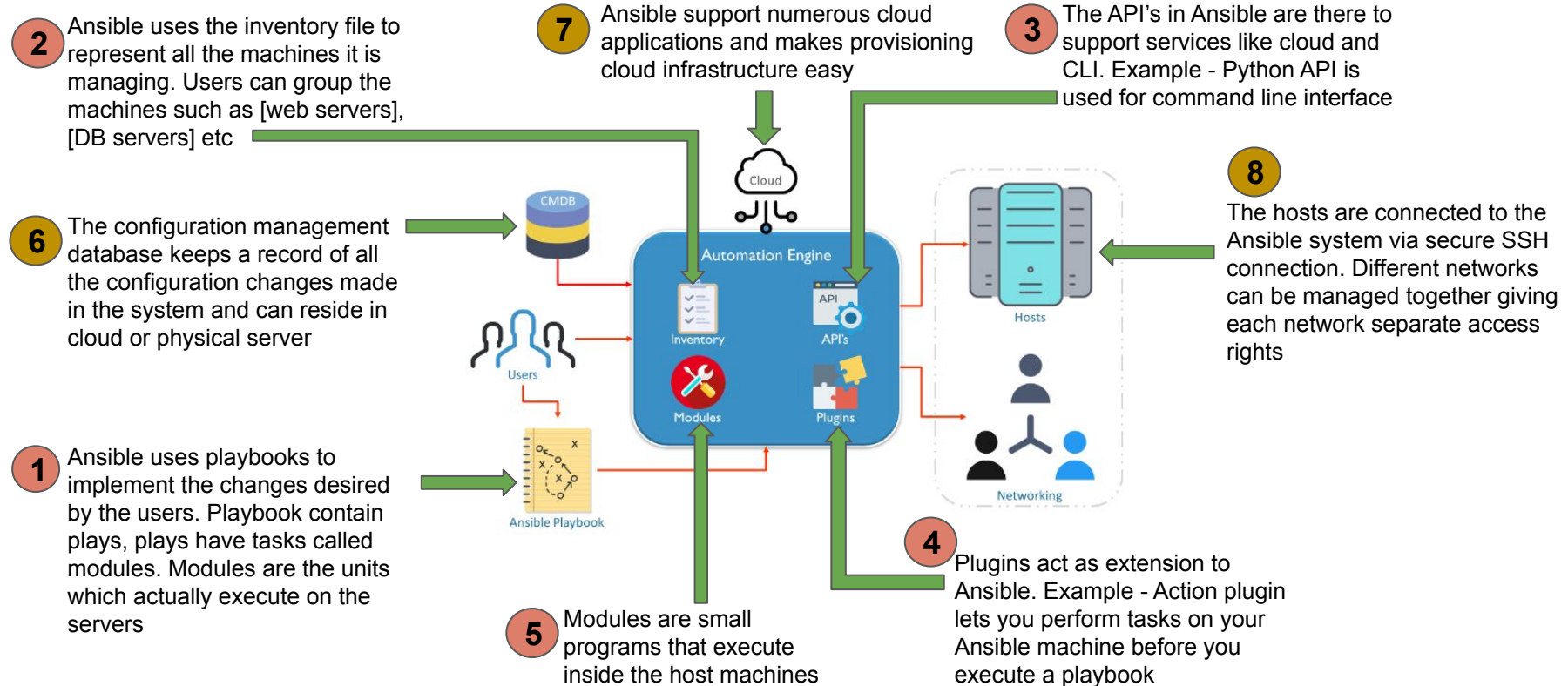


Ansible Architecture

Ansible connects to the nodes and pushes out small programs called “Modules”. These modules bring the system to the desired state. Ansible uses SSH to execute these modules and then removes them when finished.



Architecture Components



Ansible Playbook

YAML File

Playbook Name

Host

Tasks

Modules

(Example: apt, service,
yum, file, ping, user etc)

Top 100 Ansible Modules

<https://mike42.me/blog/2019-01-the-top-100-ansible-modules>


```
---  
- name: Playbook  
  hosts: webserver  
  become: yes  
  become_user: root  
  tasks:  
    - name: ensure apache is at the latest version  
      yum:  
        name: httpd  
        state: latest  
    - name: ensure apache is running  
      service:  
        name: httpd  
        state: started
```

Installing Ansible

- Spin up 3 EC2 (Ubuntu 20.x) instance if using AWS or create 3 VMs using Oracle VM Box and then connect to your machines
- Name your VMs as AnsibleServer, Host1 and Host2
- Login to all machines and change to root user: `sudo -i`
- Run command `sudo apt-get update`
- On your AnsibleServer install Ansible:
 - `apt-get install ansible`
- Next need to configure password less SSH:
 - First need to create a SSH key pair on AnsibleMaster: `ssh-keygen` (press enter for all options)
 - To copy the public key type command: `cat .ssh/id_rsa.pub` and select the key to copy
 - In all the host nodes type command: `vim .ssh/authorized_keys` and paste by right clicking
 - From AnsibleServer ssh to all the nodes once then logout: `ssh xx.xx.xx.xx`

```
ubuntu@ip-172-31-10-31:~$ ssh ip-172-31-11-115
The authenticity of host 'ip-172-31-11-115 (172.31.11.115)' can't be establish
ECDSA key fingerprint is SHA256:UD/1R38iWl2fV/qzh9HyjrNJzvdvZmeUVQLsJjRFzY0.
Are you sure you want to continue connecting (yes/no/[fingerprint])? Yes
```


Configuring Host File

- As root user the host file is under `/etc/ansible/hosts`
- Now you can edit and save the file using vim
 - `vim /etc/ansible/hosts`
- Copy and paste the IP addresses of the hosts at the bottom of the file in this format and save and exit vim
 - `ip-172-31-11-115` 

```
# Here's another example of host r
# leading 0s:

#db-[99:101]-node.example.com
ip-172-31-15-229
ip-172-31-15-102
ip-172-31-11-115
```

Ad-hoc Commands & Modules in Ansible

- Example 1: Use the ping module to check if a host can be pinged

- `ansible ip-172-31-11-115 -m ping`



```
root@ip-172-31-10-226:~# ansible ip-172-31-11-138 -m ping
ip-172-31-11-138 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
```

Module

- Example 2: Create a new user by using the user module

- `ansible ip-172-31-12-148 -m user -a "name=demouser state=present"`



```
root@ip-172-31-10-226:~# ansible ip-172-31-12-148 -m user -a "name=demouser state=present"
ip-172-31-12-148 | CHANGED => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": true,
  "comment": "",
  "create_home": true,
  "group": 1001,
  "home": "/home/demouser",
  "name": "demouser",
  "shell": "/bin/sh",
  "state": "present",
  "system": false,
  "uid": 1001
}
```


Attributes: Values



Ad-hoc Commands & Modules in Ansible (cont)

- Check if demouser has been created in host

- `id demouser`



```
root@ip-172-31-12-148:~# id demouser
uid=1001(demouser) gid=1001(demouser) groups=1001(demouser)
```

- Install a package on the host machine using apt module

- `ansible ip-172-31-12-148 -m apt -a "name=finger state=present update_cache=true"`
 - Check on the host machine if finger was installed by typing finger
 - To run a module on all the hosts use "all" instead of IP address
 - `ansible all -m apt -a "name=finger state=present update_cache=true"`



```
root@ip-172-31-12-148:~# finger
Login      Name      Tty      Idle   Login Time   Office      Office Phone
ubuntu     Ubuntu    *pts/0           Jul  8 00:57 (47.185.195.136)
```

- Google list of all ansible modules

Create and Execute a Playbook

- Change directory to /etc/ansible: `cd /etc/ansible`
- Use vim editor to write a playbook: `vim demoplaybook.yaml`
- Type this command to execute the play: `ansible-playbook demoplaybook.yaml`

```

- name: demo play
  hosts: all
  tasks:
    - name: install package ntp
      apt:
        name: ntp
        state: present
    - name: create a file
      copy:
        dest: /tmp/fileFromAnsible
        content: "Hello World!"

```



```

PLAY [demo play] *****

TASK [Gathering Facts] *****
ok: [ip-172-31-12-148]
ok: [ip-172-31-11-138]

TASK [install package ntp] *****
changed: [ip-172-31-12-148]
changed: [ip-172-31-11-138]

TASK [create a file] *****
changed: [ip-172-31-12-148]
changed: [ip-172-31-11-138]

PLAY RECAP *****
ip-172-31-11-138      : ok=3    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
ip-172-31-12-148      : ok=3    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

```

- Check host for changes



```

root@ip-172-31-12-148:~# apt install ntp
Reading package lists... Done
Building dependency tree
Reading state information... Done
ntp is already the newest version (1:4.2.8p12+dfsg-3ubuntu4.20.04.1).
0 upgraded, 0 newly installed, 0 to remove and 39 not upgraded.
root@ip-172-31-12-148:~# cat /tmp/filefromserver
Master Academyroot@ip-172-31-12-148:~#

```

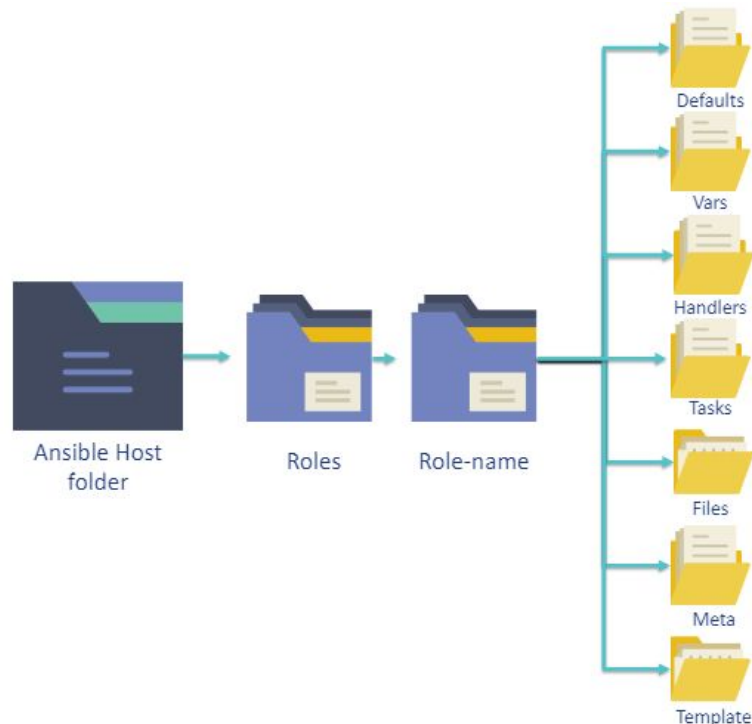
Grouping Inventory

- Grouping of inventory is a general practice to isolate hosts or machines into different categories. If we have 100s of hosts we can group them as webserver, database etc. Grouping allows Ansible to push different configurations to different types of hosts in production environment.
- Inside the inventory file (/etc/ansible/hosts) host machines can be grouped
 - vim /etc/ansible/hosts
 - Type [webserver] & [database]: follow the example below

```
#db-[99:101]-node.example.com
[webserver]
ip-172-31-12-148
[database]
ip-172-31-11-138
-- INSERT --
```

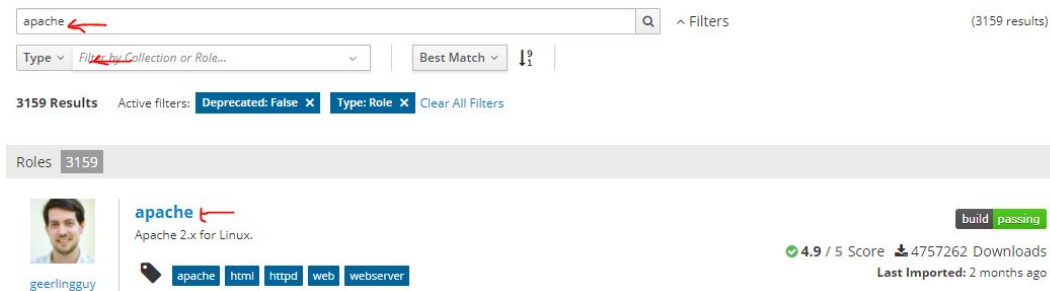
Ansible Roles

- Overtime working with Ansible a user may create hundreds of playbooks, variables, templates etc. Roles allow users to group this logic into an organized manner making reusability and sharing of Ansible structure easier.
- Roles uses directories to structure and group all the playbooks, variables, templates, tasks, handlers, files and defaults.
- This collected logic can be grouped in any way the user wants, for example you group server specific roles together.
- These roles can then be used inside playbooks and even as in-line commands



Creating a Role

- To create a role a tool called Ansible-galaxy comes installed with ansible.
- In order to create a new role it must be initialized, once initialized the folders will be created. The defaults folder contains a file called main.yml which contains all the default variables to be used by the role.
- For example a requirement has come in to install 10 apache webserver. There could be different types of machines like ubuntu, centos, suse and all of them will require different configurations and setup parameters. You can write a playbook but need to define the logic which can be complicated. This problem can be solved by using Roles.
- Roles for different task (as stated above) can be found on galaxy.ansible.com
 - Browse to galaxy.ansible.com
 - Click “Search” and type apache and filter type for roles then click on apache



The screenshot shows the Ansible Galaxy search interface. The search bar contains 'apache' with a red arrow pointing to it. Below the search bar, there are filters for 'Type' (set to 'Role') and 'Best Match'. The results show 3159 roles. The first result is 'apache' by 'geerlingguy', which is highlighted. It has a 'build passing' status, a 4.9/5 score, 4757262 downloads, and was last imported 2 months ago. The role is described as 'Apache 2.x for Linux' and has tags for 'apache', 'html', 'httpd', 'web', and 'webserver'.

Creating Roles (cont)

- Next copy the instruction and run it on your Ansible Server
 - `cd /etc/ansible` (change dir to ansible if not already in)
 - `ansible-galaxy install geerlingguy.apache`
 - `cd ~/.ansible/roles/geerlingguy.apache/` (After installation go to the folder where role was installed)
 - Inside this folder type "ll" to list all folders and go inside tasks folder and you will find main.yml
- Next change directory to /etc/ansible and create a new playbook and only include the installed role
 - `vim apacheRolePlaybook.yaml`
 - `ansible-playbook apacheRolePlaybook.yaml`
- Open browser and type webserver public IP and port 80
 - Example: 3.91.174.225:80

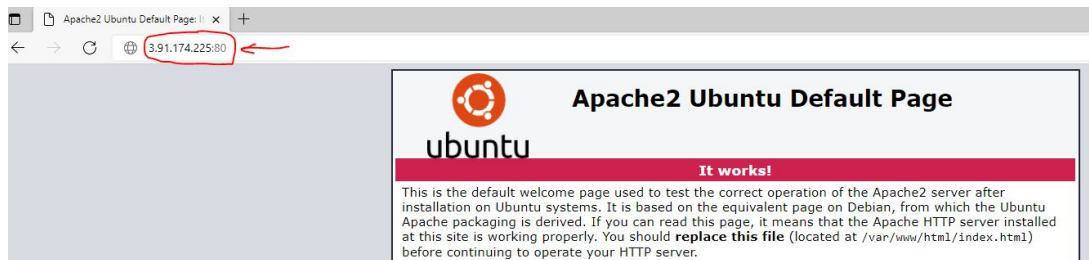
Minimum Ansible Version

2.4

Installation

\$ ansible-galaxy install geerlingguy.apache

```
- name: Install Apache Webserver
  hosts: webserver
  roles:
    - role: geerlingguy.apache
```





Create Custom Role

- It is also possible to create your own role
 - Make a directory inside /etc/ansible: `mkdir roles`
 - Change directory to roles: `cd roles`
 - Run command: `ansible-galaxy role init myrole`
 - A new directory called myrole will be created
 - Change directory to myrole and list all folders
 - Change directory to tasks and open main.yml
 - Create a task inside main.yml using vim
 - Now go to /etc/ansible and create new playbook
 - Run playbook useCustomRole.yaml

```
# Tasks file for myrole
- name: create a file
  copy:
    dest: /tmp/filefromserver
    content: "Master Academy Role!"
```

```
- name: using my custom role
  hosts: database
  roles:
    - role: myrole
```

```
root@ip-172-31-10-226:/etc/ansible# cd roles
root@ip-172-31-10-226:/etc/ansible/roles# ansible-galaxy role init myrole
- Role myrole was created successfully.
root@ip-172-31-10-226:/etc/ansible/roles# ll
total 12
drwxr-xr-x  3 root root 4096 Jul  8 21:28 ./
drwxr-xr-x  3 root root 4096 Jul  8 21:24 ../
drwxr-xr-x 10 root root 4096 Jul  8 21:28 myrole/
root@ip-172-31-10-226:/etc/ansible/roles# cd myrole
root@ip-172-31-10-226:/etc/ansible/roles/myrole# ll
total 48
drwxr-xr-x 10 root root 4096 Jul  8 21:28 ./
drwxr-xr-x  3 root root 4096 Jul  8 21:28 ../
-rw-r--r--  1 root root  539 Jul  8 21:28 .travis.yml
-rw-r--r--  1 root root 1328 Jul  8 21:28 README.md
drwxr-xr-x  2 root root 4096 Jul  8 21:28 defaults/
drwxr-xr-x  2 root root 4096 Jul  8 21:28 files/
drwxr-xr-x  2 root root 4096 Jul  8 21:28 handlers/
drwxr-xr-x  2 root root 4096 Jul  8 21:28 meta/
drwxr-xr-x  2 root root 4096 Jul  8 21:28 tasks/
drwxr-xr-x  2 root root 4096 Jul  8 21:28 templates/
drwxr-xr-x  2 root root 4096 Jul  8 21:28 tests/
drwxr-xr-x  2 root root 4096 Jul  8 21:28 vars/
root@ip-172-31-10-226:/etc/ansible/roles/myrole# cd task
-bash: cd: task: No such file or directory
root@ip-172-31-10-226:/etc/ansible/roles/myrole# cd tasks
root@ip-172-31-10-226:/etc/ansible/roles/myrole/tasks# vim main.yml
```

Ansible Variables

- Ansible is not a full-fledged programming language, but it does have several programming language features, and one of the most important of these is variable substitution.
- To define a variable in YAML we use “vars” and use {{ }} curly braces to insert the variable
- Here is a simple example of the usefulness of variables in Ansible:
 - Open the demoplaybook.yaml file and edit it as follows
 - Execute the demoplaybook.yaml

Variable Declared

Variable Used

```
---
- name: demo play
  hosts: all
  vars:
    file_content: "Hello World!"
  tasks:

    - name: create 1st file
      copy:
        dest: /tmp/filefromserver1
        content: "{{ file_content }}"

    - name: create 2nd file
      copy:
        dest: /tmp/filefromserver2
        content: "{{ file_content }}"

    - name: create 3rd file
      copy:
        dest: /tmp/filefromserver3
        content: "{{ file_content }}"

    - name: create 4th file
      copy:
        dest: /tmp/filefromserver4
        content: "{{ file_content }}"
```



Deploying a Docker Container using Ansible

- We can use Ansible to deploy docker containers across several hosts
- We will use a docker image from docker hub “mateors/webapp”
- Create a new playbook called dockerplaybook.yaml
- On the host machine run docker ps

```
root@ip-172-31-12-148:~# docker ps
CONTAINER ID   IMAGE                COMMAND
83b543e7d81e   mateors/webapp:beta  "dockerapp"
root@ip-172-31-12-148:~#
```

- Check on web browser by typing
 - <host-public-ip:8180/webapp>

```
---
- name: Deploy Docker image in webserver
  hosts: webserver
  vars:
    package_name: "docker.io"
  tasks:
    - name: Install Docker
      apt:
        name: "{{ package_name }}"
        state: present
        update_cache: true
    - name: Start Docker Service
      service:
        name: docker
        state: started

- name: Run webapp docker container
  hosts: webserver
  vars:
    image_name: "mateors/webapp:beta"
  tasks:
    - name: Run Container
      command: "docker run -d -p 8180:8180 {{ image_name }}"
```

Thank You