ANSIBLE

History :

- Michael Dehhan developed Ansible and the ansible project began in feb 2012
- Ansible was taken over by Red-Hat
- Ansible supports in Linux for all the flavors. i.e., RHEL, Debian, CentOS, Oracle Linux

We can use this tool, whether our servers are in On-Premise (or) in the cloud

- Ansible is a simple open-source IT engine which automates application deployment
- It is a server configuration management tool
- For Server management we are using ansible
- Ansible is Orchestration, Security and Compliance
- It uses YAML Scripting Language which works on Key-Value pair
- So, here for server configure we're using YAML
- Ansible dependency "Python". It is used python for back end
- Ansible GUI is called as Ansible Tower. It is paid version
- Instead of this we use playbooks for this
- Ansible is having greater community

For Creation of infrastructure also we're using ansible

Configuration Management:

It is a process where we maintain infrastructure and software changes through automation and the goal is to maintain them in a desired state.

- It saves lots of time and it makes it easy to fix defects if any

Why we're using Ansible ?

1. While managing the multiple servers it's hard to keep their configuration identical. If you have multiple servers which needs to configure the same setup in all

- while doing the one to one server their might be a chances to miss some configuration steps in some servers

Eg: We are having 10 servers. In this 10 servers we have to set up the tomcat. Generally, tomcat setup contains lot of steps. So, it's little bit hard. For single servers we can do it. If 10 servers we do manually means may be we will miss some steps. so, tomcat fails.

- That's why automation tools comes into play.
- The automation tools: Ansible, chef, puppet and SaltStack all are based on same principle

"Describe the desired state of the System"

2. Ansible uses plain SSH

- That means no need to install the agents/dependencies on client machines. Eg: Slave (or) Tomcat etc..,
- But other automation tools like Chef/Puppet needs to install agent/dependency on client machine when we need to perform a task

3. Ansible is light weight, relative easy to use deployment speed here. When we compared to others

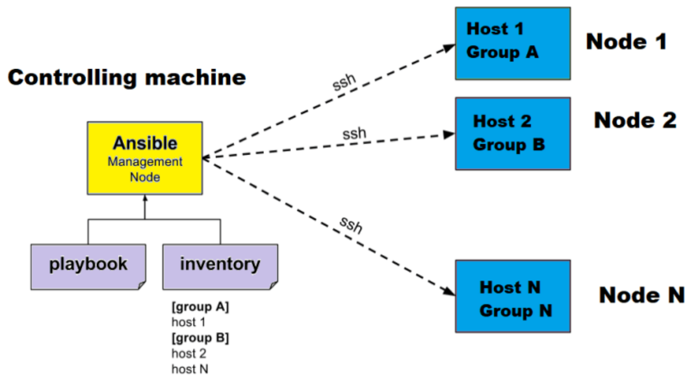
4. Ansible doesn't track the dependencies

5. Here, master & slave we will connect in 2 ways

- Using standard SSH Commands
- Using paramiko module which provides a python interface to SSH2.
    - But, it is very hard, high end configuration
- So, in real-life companies are using SSH connection via standard SSH Commands

Ansible WorkFlow

Using YAML Scripting we're writing playbooks



Eg: Suppose, in Host servers we have to install 10 servers. In each server inside we have to install maven

- So, go to ansible server, here we write the playbook
- And we execute the playbook in ansible server
- Automatically, on that host servers maven installed

Inventory :

The ansible inventory file defines the hosts and groups of hosts upon which commands, modules, and tasks in playbooks operate

(or)

The ansible inventory file is a text file consisting of host names and IP addresses of remote/slave servers that stores the information in "ansible.cfg" file and target system where the ansible script will be executed

- Default inventory path : /etc/ansible/hosts

2 types of Inventory

1. Static Inventory
   - It is called default inventory
   - We can manually create this file, which will include the group name and all the hosts that you want to run the Ansible playbooks against
   - stored the data in the form of an YAML file

2. Dynamic Inventory

   - Dynamic Inventory comes in handy when your Ansible inventory fluctuates overs time, with different management nodes spinning up and shutting down frequently.
   - Dynamic Inventory has two ways of retrieving the management nodes, inventory scripts and plugins.

We can see the differences between above inventories in below categories

- [ ] **Maintenance:** Static inventory requires manual updates, therefore making it more error prone and time consuming for your team. However, Dynamic Inventory utilizes the plugin and adjusts the management nodes automatically
- [ ] **Flexibility:** Static inventory 'groups' management nodes manually, but dynamic inventory is able to create groups and assign management nodes to them automatically, therefore increasing flexibility
- [ ] **Scalability**: For the long term, its always best to prepare for scalability. Dynamic Inventories will allow for your team to be ahead of the game and ready for any scaling up that is needed in the future.
- [ ] **Real-Time Accuracy:** Static Inventory can easily become out of sync with management nodes that are no longer running. Dynamic Inventory will always give you real-time data based on your management nodes

Ansible Master Server Setup:

Step -1 : Launch 5 instances and named as master, slave1 to slave4

Step -2 : Open master server, Here we're doing setup in master server

1. Install Ansible → amazon-linux-extras install ansible2 -y
   - Here, ansible doesn't track the dependencies
   - That's why if we install ansible, that ansible only installed
2. Install Dependency - Python → yum install python python-pip python-level -y

   So upto 2nd step ansible setup is done

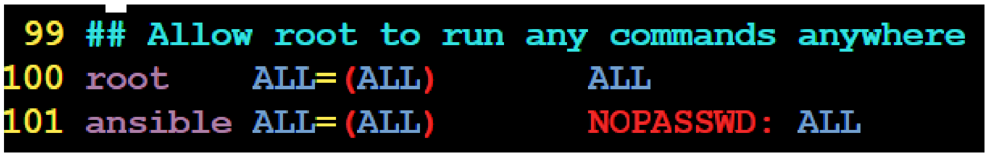
3. Add the Ansible user → useradd ansible

   - Here, we can add any name but we have to maintain same name for every server

4. Set the password for the user → passwd ansible

5. Give the root powers to ansible user

   - Open #visudo in terminal
   - Go to 100th line and copy the root line and make it as ansible. For better understanding do same like below image

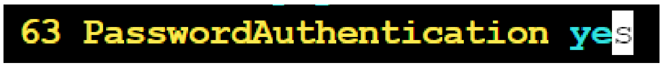


```
 99 ## Allow root to run any commands anywhere
100 root    ALL=(ALL)       ALL
101 ansible ALL=(ALL)       NOPASSWD: ALL
```

   - After write above data save and exit use :wq

6. Password Authentication

   - Go to #vi /etc/ssh/sshd_config
   - Go to 63rd line and give password authentication -yes, After the changes do save and exit



```
63 PasswordAuthentication yes
```

   - Because when we're login into slave server it will asks password for that reason we are using password authentication that's the reason we gave yes

7. Restart the sshd → systemctl restart sshd

   - Here, we changed the sshd configuration data.
   - So, whenever if we change any data in the configuration file we need to restart it

So, upto 7 steps we successfully setup the master server

Ansible Slave Server Setup:

- So, In Slave servers we no need to install ansible (or) python. Just we can add the user Ansible
- So, In master setup perform 3 to 7 steps in slave servers. So, this steps will be done in all slave servers

So, right now in all slave servers we did server configuration

Now, we have to do Ansible configuration

Ansible Configuration Setup:

So, here take Slave-1, Slave -2 as a group and Slave-3, Slave-4 as another group

If we are forming as a groups means the use is .

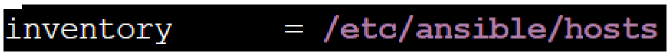
Suppose, I want to perform a task/action at a time in both slave-1 and slave-2. Usually we do manually server to server. But here, using groups I can directly mention the group names instead of IP address

So, we have to define the groups in ansible path i.e., → cd /etc/ansible

Now, go to master server, where ansible is installed

Step -3 :

1. Go to this path → cd /etc/ansible
2. vim ansible.cfg
   - Inside remove '#' in inventory like in shown below. So, right now inside hosts we're defining the slave server IP address

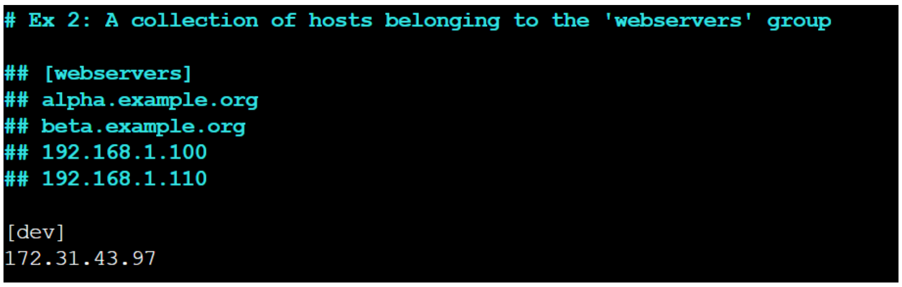


```
inventory      = /etc/ansible/hosts
```

   - Remove '#' in #sudo_user = root
   - Because we're using non-root user
   - After performing above changes do save and exit

Step -4 :

1. Go to this path → vi /etc/ansible/hosts and create the groups here
2. Here, we add our slave server public (or) private IP address with the group name like given below
   - So, like this we can add multiple groups and inside we can add multiple slave server IP address



```
# Ex 2: A collection of hosts belonging to the 'webservers' group

## [webservers]
## alpha.example.org
## beta.example.org
## 192.168.1.100
## 192.168.1.110

[dev]
172.31.43.97
```

3. But preferred one is "private IP" because every time we stop and start the server public IP will change. So, again we have to change the configuration. So, that's the reason we take private IP because it's constant

Step -5 :

Now, we need to connect slave servers with master for that we need keys. Because we launch servers without key-pair. So, we don't have a pem file

- Login into ansible user
  - Go to home directory and perform → su - ansible

Step -6 :

Here, we're generating the keys

- command is → ssh-keygen
  - click on enter..
- Check the keys whether it's present/not
  - command is → ls -al → cd .ssh → you're having keys

Step -7 :

Here, we have to copy the keys in slave servers

- command is → ssh-copy-id ansible@slavePrivateIP
- click on enter
- Give the ansible user password (check step-2, here 4th point you have the answer)

So, in this step we send keys in slave server

Step -8 :

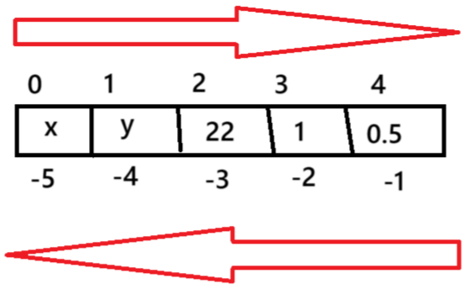
Now, we can directly login into slave server inside the master i.e.., connect with slave-1

- command is → ssh ansible@slavePrivateIP
- Now, check the privateIP and compare
- If you want to come to master → exit

So, this is the way we can connect with slave servers through step-7&8. So, like this connect upto 4 slaves

Ansible Host Pattern :

An Ansible pattern can refer to a single host, an IP address, an inventory group, a set of groups, or all hosts in your inventory.



Here, we are having list, So we're having 2 values

- Forward Indexing values → starts from 0, 1, 2, 3, 4,.......
  - If you want to know the index → n-1
- Reverse Indexing values → starts from -1, -2, -3, -4, ...
  - If you want to know the index → -n

Here, we are checking the connection is established (or) not. Before we executing playbooks it is important step which we need to do

- This command will shows you the how many servers are connected with master
  - ansible all --list-hosts
- Now, I want to see only particular group servers IP address
  - ansible groupName --list-hosts
- So, coming to our servers. we are having 4 slave servers. So, here i want to see first 2 servers
  - ansible all[0:1] --list-hosts
- Now, In the group I want to see the 1st slave server IP address

  - ansible dev[0] --list-hosts

So, here we are getting these servers through host pattern

So, overall we are getting the slave servers IP address list

- I want to check whether my server is active/inactive
    - ansible all -m ping   (or) ping SlavePrivateIPAddress
  - we got success
  - Here, ping is used to check the server network connections
- Here, we are having 2 states
  - reachable state → connection perfect
  - non-reachable state → connection lost

Note: Make sure that, perform the commands in ansible user

PlayBooks

- An Ansible playbook is an organized unit of scripts that defines the tasks involved in managing a system configuration using the automation tool Ansible
- Playbooks in ansible are written in YAML Language
- It is human readable
- For the configuration files frequently we're using playbooks
- You can write codes consists of vars, tasks, handlers, files, templates and roles
- Each playbook is composed of one (or) more modules in a list
  - ie., Inside a playbook we can do multiple tasks by using multiple modules

Why we're using ansible playbook

- Very simple to set up and use: No special coding skills are necessary to use Ansible's playbooks (more on playbooks later).
- Powerful: Ansible lets you model even highly complex IT workflows.
- Flexible: You can orchestrate the entire application environment no matter where it's deployed

Playbooks are mainly divided into 3 sections

1. Target Section

   It defines host against which Playbooks task has to be executed

   If you take any playbooks target section won't change

2. Variable Section

   Here, we're defining variables/values

3. Task Section

   It defines actions you are performing

The difference between Target and Task Section is

☐ Target - Where you are going to do
☐ Task - what you are going to do

We have to write the PLAYBOOK in MASTER SERVER because here ansible exists

Creating a Playbook → vim file.yaml (or) vi file.yml

Every Playbook starts with “---” ends with “...”

Sample code for PlayBook

☐ Target Section

- hosts: dev → in this hosts we’re giving group names. and we can multiple groups Eg: all, dev[0] - for single slave

  user: ansible

  become: yes → if the user is non-root. So, this become code line gives the root powers

  connection: ssh

Above code is all about target section. In a playbook target section won’t change

☐ Task Section

  tasks:

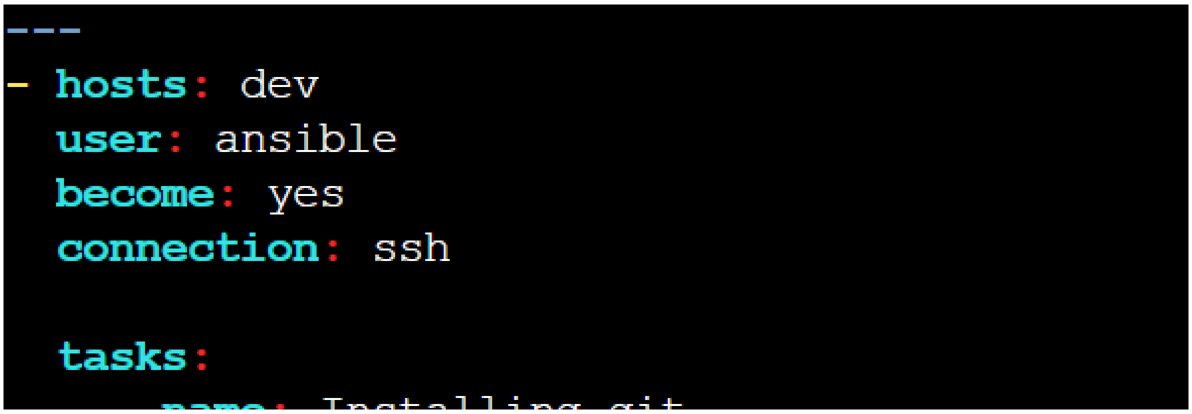
   - name: Installing git → every new action we have to give like “- name”

     action: yum name=git state=present → action is module

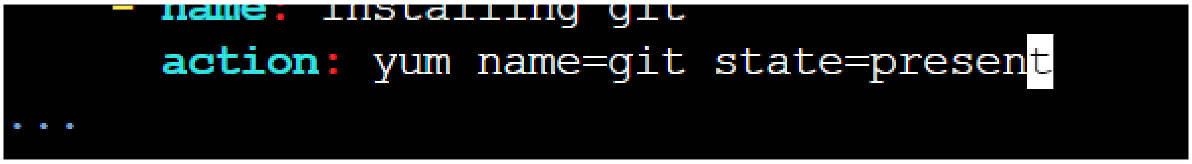
Above code is all about task section. In a playbook we can write so many tasks

Note:

- you must & should follow indentation. Actually when you’re writing the code, it will automatically follows indentation. If you miss single void space, your playbook won’t execute
- If you’re having YAML Scripting syntax errors, check in YAMLlint in browser



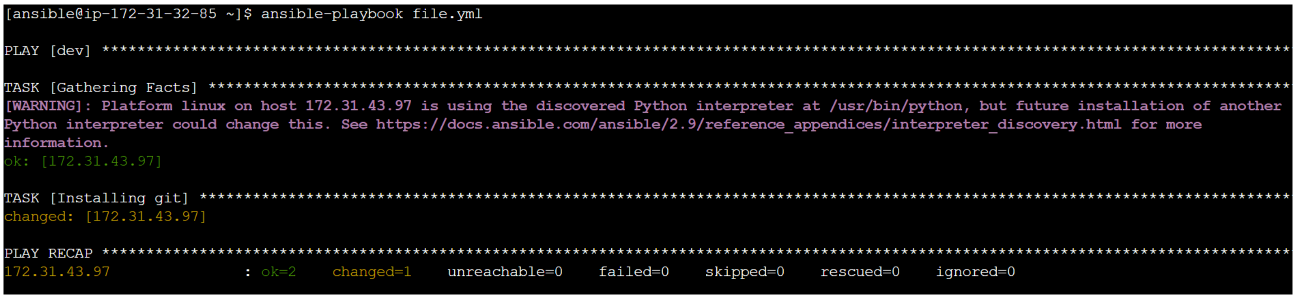
```
---
- hosts: dev
  user: ansible
  become: yes
  connection: ssh

  tasks:
    - name: Installing git
```



```
    - name: installing git
      action: yum name=git state=present
...
```

In this playbook we're installing this git

Now, Execute the Playbook in master → ansible-playbook file.yml



```
[ansible@ip-172-31-32-85 ~]$ ansible-playbook file.yml

PLAY [dev] ***************************************************************************************************************

TASK [Gathering Facts] ***************************************************************************************************
[WARNING]: Platform linux on host 172.31.43.97 is using the discovered Python interpreter at /usr/bin/python, but future installation of another
Python interpreter could change this. See https://docs.ansible.com/ansible/2.9/reference_appendices/interpreter_discovery.html for more
information.
ok: [172.31.43.97]

TASK [Installing git] ****************************************************************************************************
changed: [172.31.43.97]

PLAY RECAP ***************************************************************************************************************
172.31.43.97               : ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

we will get like this that means we successfully executed playbook.

- So, here we can see the task name and description
- And under the task whatever the tasks we're performing in ansible it will perform in the given slave server IP address
- OK=2 → server reachable state & task completed means it will shows like this
- Changed =1 → Here, we install the single task, that's why it is showing 1
- So, go to slave server and check whether the git is installed (or) not

In ansible playbooks we're having 4 states

1. present = install
2. absent = uninstall
3. latest = update
4. restarted = restart

Here, In Tasks we are having 3 types of modules to install tools

1. action → Eg: action: yum name=git state=present
2. yum → Eg: yum: name=git state=present
3. command → Eg: command: yum install git -y
   - This is used to write hard commands like amazon-linux-extras etc..,

Ansible Setup through root user

Previously, we did ansible setup by creating one ansible user like that, but here without creating any user, we're doing ansible setup by root user

Ansible Master Server Setup:

Step -1 : Launch 5 instances and named as master, slave1 to slave4

Step -2 : Open master server, Here we're doing setup in master server

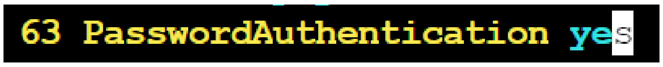
1. Install Ansible → amazon-linux-extras install ansible2 -y
    - Here, ansible doesn't track the dependencies
    - That's why if we install ansible, that ansible only installed
2. Install Dependency - Python → yum install python python-pip python-level -y

   So upto 2nd step ansible setup is done

3. Set the password for the root user → passwd root

4. Password Authentication

    - Go to #vi /etc/ssh/sshd_config
    - Go to 38th line and give the permit root login "YES"
    - Go to 63rd line and give password authentication -yes, After the changes do save and exit



```
63 PasswordAuthentication yes
```

    - Because when we're login into slave server it will asks password for that reason we are using password authentication that's the reason we gave yes

5. Restart the sshd → systemctl restart sshd

    - Here, we changed the sshd configuration data.
    - So, whenever if we change any data in the configuration file we need to restart it

So, upto 7 steps we successfully setup the master server

Ansible Slave Server Setup:

- So, In Slave servers we no need to install ansible (or) python.
- No need to add any user, because we're doing in root user
- So, In master setup perform 3 to 5 steps in slave servers. So, this steps will be done in all slave servers

So, right now in all slave servers we did server configuration

Now, we have to do Ansible configuration

Ansible Configuration Setup:

So, here take Slave-1, Slave -2 as a group and Slave-3, Slave-4 as another group

If we are forming as a groups means the use is .

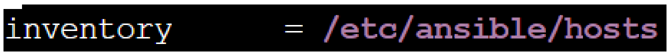
Suppose, I want to perform a task/action at a time in both slave-1 and slave-2. Usually we do manually server to server. But here, using groups I can directly mention the group names instead of IP address

So, we have to define the groups in ansible path i.e., → cd /etc/ansible

Now, go to master server, where ansible is installed

Step -3 :

1. Go to this path → cd /etc/ansible
2. vim ansible.cfg
   - Inside remove '#' in inventory like in shown below. So, right now inside hosts we're defining the slave server IP address

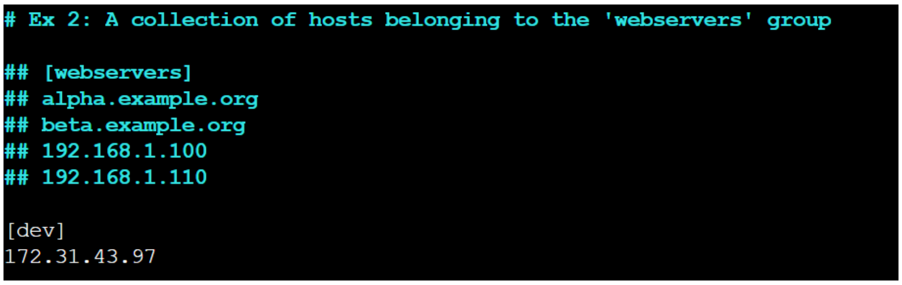


```
inventory      = /etc/ansible/hosts
```

   - After performing above changes do save and exit

Step -4 :

1. Go to this path → vi /etc/ansible/hosts and create the groups here
2. Here, we add our slave server public (or) private IP address with the group name like given below
   - So, like this we can add multiple groups and inside we can add multiple slave server IP address



```
# Ex 2: A collection of hosts belonging to the 'webservers' group

## [webservers]
## alpha.example.org
## beta.example.org
## 192.168.1.100
## 192.168.1.110

[dev]
172.31.43.97
```

3. But preferred one is "private IP" because every time we stop and start the server public IP will change. So, again we have to change the configuration. So, that's the reason we take private IP because it's constant

Step -5 :

Now, we need to connect slave servers with master for that we need keys. Because we launch servers without key-pair. So, we don't have a pem file

Here, we're generating the keys in root user

- command is → ssh-keygen
  - click on enter..
- Check the keys whether it's present/not
  - command is → ls -al → cd .ssh → you're having keys

Step -6 :

Here, we have to copy the keys in slave servers

- command is → ssh-copy-id root@slavePrivateIP
- click on enter
- Give the ansible user password (check step-2, here 3rd point you have the answer)

So, in this step we send keys in slave server

Step -7 :

Now, we can directly login into slave server inside the master i.e.., connect with slave-1

- command is → ssh root@slavePrivateIP
- Now, check the privateIP and compare
- If you want to come to master → exit

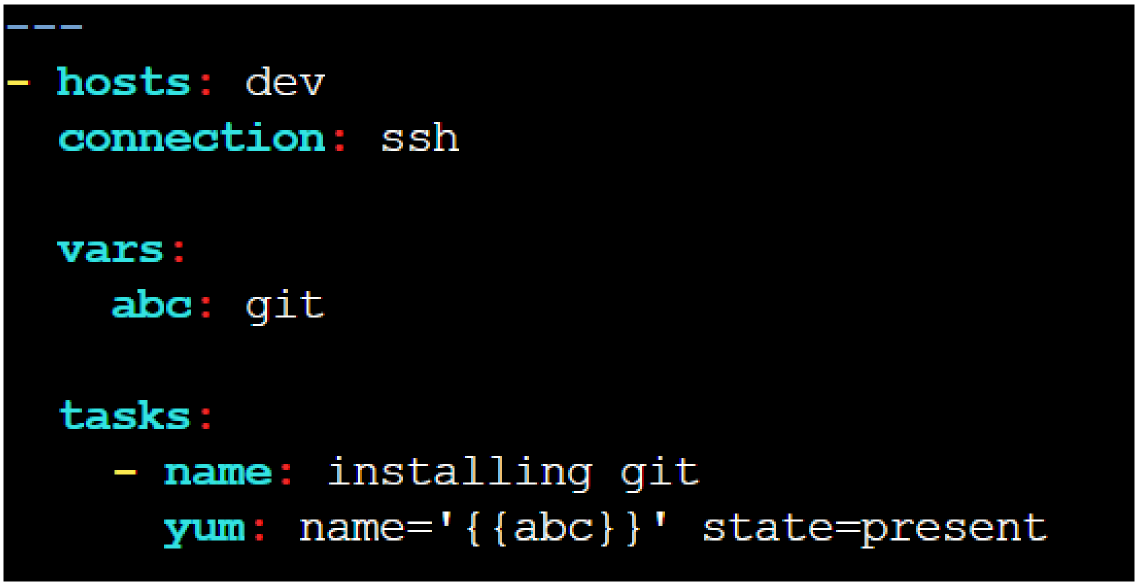
Variable Section

In Playbooks we define variables in multiple ways. Generally 3 ways

1. Direct way
2. Runtime variable declaration
3. File-passing a variable in different files

☐ Direct way

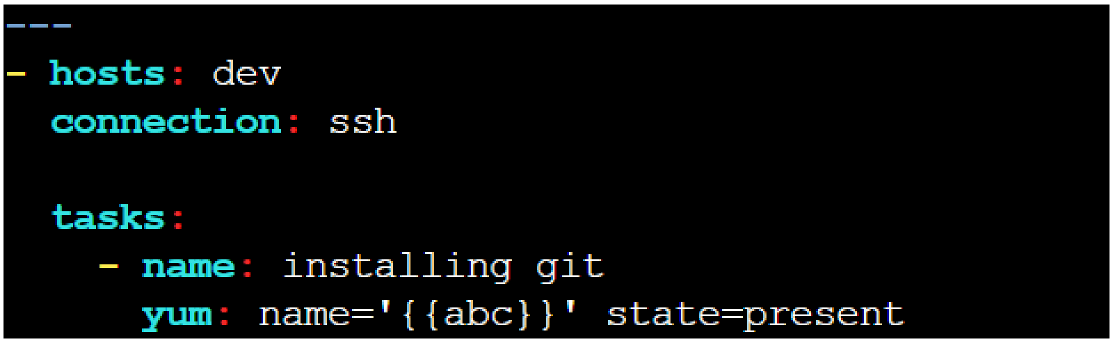
Here, we're defining variables in playbook directly



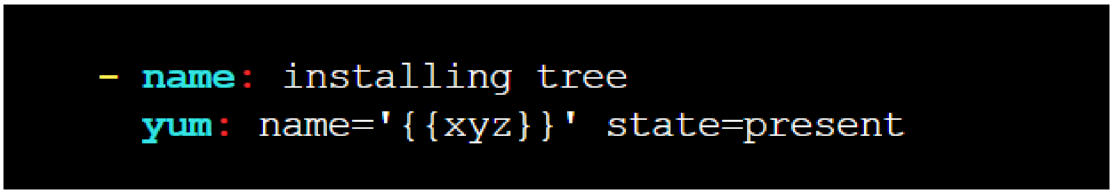
```yaml
---
- hosts: dev
  connection: ssh

  vars:
    abc: git

  tasks:
    - name: installing git
      yum: name='{{abc}}' state=present
```

☐ Runtime variable declaration

- Here, when we run the playbook we assign variables instead of going into the file.
- So, in above code, remove 'vars' section and keep tasks section as it is
- Here, we can declare multiple variables in runtime



```yaml
---
- hosts: dev
  connection: ssh

  tasks:
    - name: installing git
      yum: name='{{abc}}' state=present
```


```
- name: installing tree
  yum: name='{{xyz}}' state=present
```

To run a Single variable

- ansible-playbook filename.yml --extra-vars “abc=git”

To run a Multiple variable

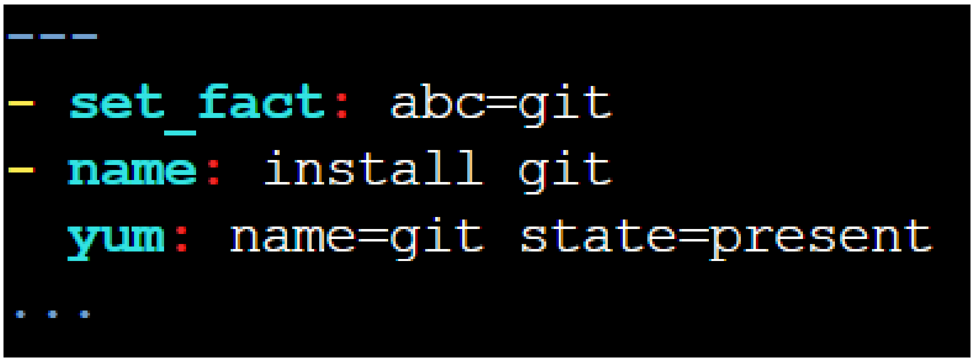
- ansible-playbook filename.yaml --extra-vars “abc=git xyz=maven”

This is the way we can run multiple variables using above command

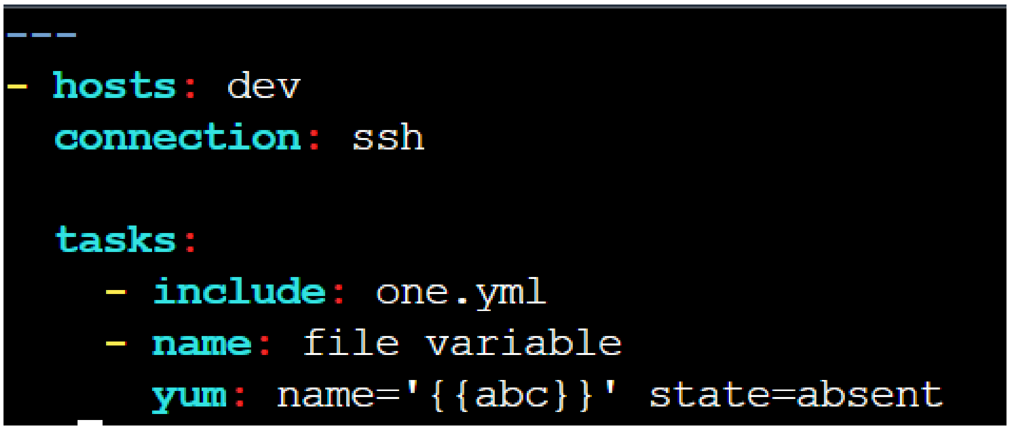
☐ File-passing a variable in different files

Here, we’re creating 2 files. In one file we are declaring variable, In another file we’re accessing the variable

- So, that means in one file we’re writing playbook. And in another file we’re defining a variable
- In file1 we write below code
    - whenever if you want to assign a variable value to another file means we use “set_fact”


```
---
- set_fact: abc=git
- name: install git
  yum: name=git state=present
...
```

- In 2nd file, write below code
    - Inside the tasks we are including the variable file name


```
---
- hosts: dev
  connection: ssh

  tasks:
    - include: one.yml
    - name: file variable
      yum: name='{{abc}}' state=absent
```

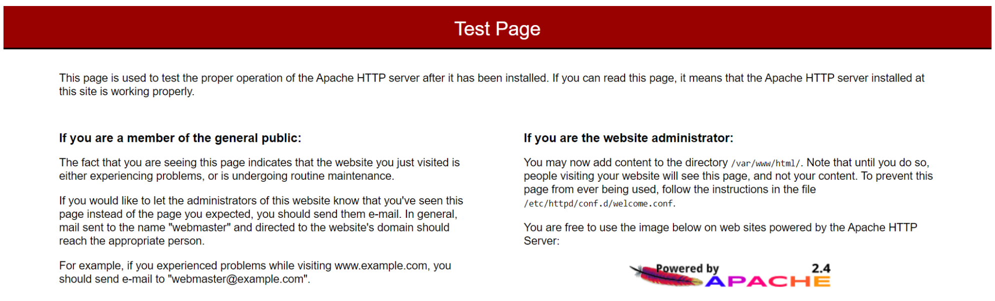
- So ,overall from this above codes what exactly we’re doing is in 1st file we’re installing git and in 2nd file we’re uninstalling git

HTTPD

- HTTPD stands for Hypertext Transfer Protocol daemon. It is a service
- It usually is the main software part of an HTTP server.
- So we can call HTTPD as a web server
- Some commonly used implementations are: Apache HTTP Server
- Apache HTTPD is an HTTP Server daemons which handles the requests
- It is designed to be run as a standalone daemon process
- Port number is 80
- How to access httpd → publicIP:80

HTTPD Setup is very easy

1. Install the httpd in master server → yum install httpd -y
2. We have to start the HTTPD Service → systemctl start httpd
3. Go to Browser and access httpd with the port number of 80, you get below image



Test Page

This page is used to test the proper operation of the Apache HTTP server after it has been installed. If you can read this page, it means that the Apache HTTP server installed at this site is working properly.

If you are a member of the general public:

The fact that you are seeing this page indicates that the website you just visited is either experiencing problems, or is undergoing routine maintenance.

If you would like to let the administrators of this website know that you've seen this page instead of the page you expected, you should send them e-mail. In general, mail sent to the name "webmaster" and directed to the website's domain should reach the appropriate person.

For example, if you experienced problems while visiting www.example.com, you should send e-mail to "webmaster@example.com".

If you are the website administrator:

You may now add content to the directory `/var/www/html/`. Note that until you do so, people visiting your website will see this page, and not your content. To prevent this page from ever being used, follow the instructions in the file `/etc/httpd/conf.d/welcome.conf`.

You are free to use the image below on web sites powered by the Apache HTTP Server:

This is the way we have to do the HTTPD setup

DEPLOY APPLICATION IN HTTPD

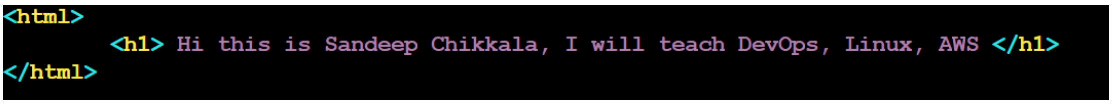
Now, we're deploying a HTML based application in HTTPD in Ansible. So we have to do this in Master ie., where ansible installed

- HTTPD contains Default Path → /var/www/html

So, inside html we have to write the source code

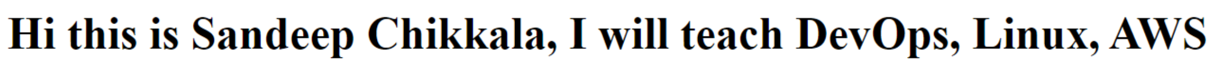
First go to HTTPD Default path

a. cd /var/www/html
b. Inside html folder, create a file named as "index.html"
c. Copy one sample code and paste inside that index.html file



```
<html>
        <h1> Hi this is Sandeep Chikkala, I will teach DevOps, Linux, AWS </h1>
</html>
```

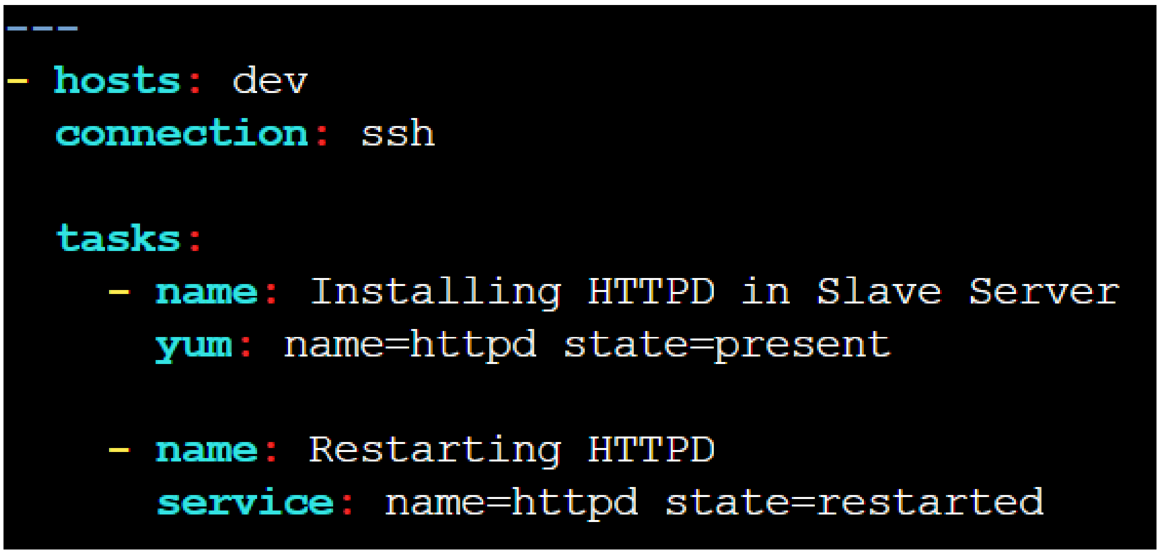
Now, go to Web Server and refresh, you can see the output like this



Hi this is Sandeep Chikkala, I will teach DevOps, Linux, AWS

Install HTTPD in Slave Servers

For install HTTPD in Slave Servers we need to write playbook



```
---
- hosts: dev
  connection: ssh

  tasks:
    - name: Installing HTTPD in Slave Server
      yum: name=httpd state=present

    - name: Restarting HTTPD
      service: name=httpd state=restarted
```

For restarting purpose, we have to use service module

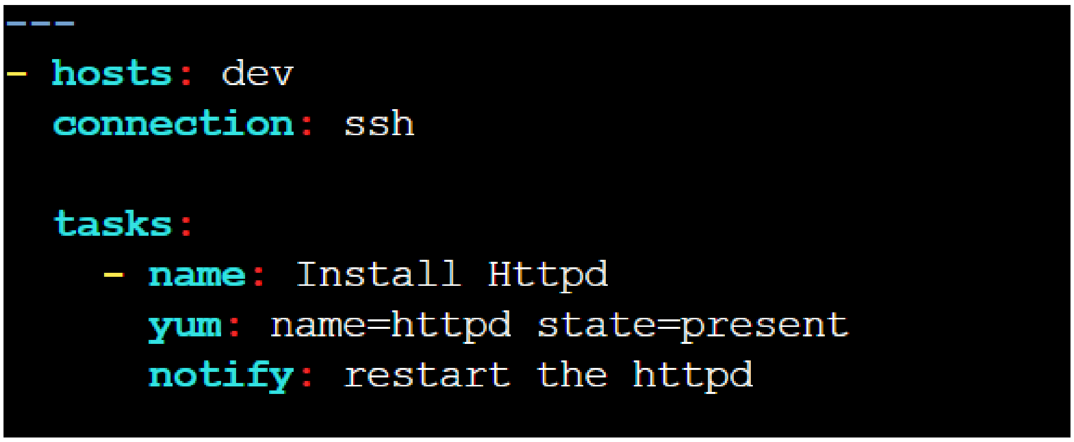
Run the code and check with slave server IP address

Handlers

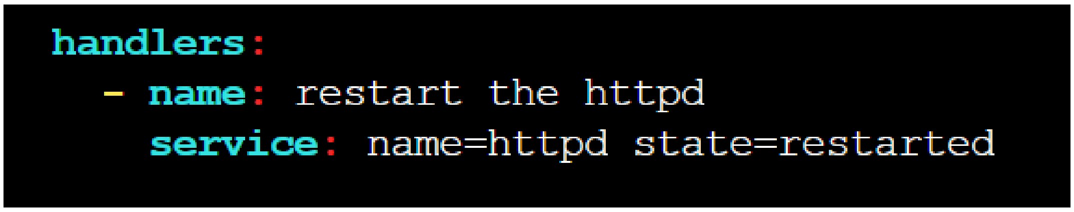
Handlers is same as the task but it will run when they called by a notifier in another task

Through handlers, we're sending a notification from one task to another task

- Installing HTTPD using handlers in a playbook



```
---
- hosts: dev
  connection: ssh

  tasks:
    - name: Install Httpd
      yum: name=httpd state=present
      notify: restart the httpd
```



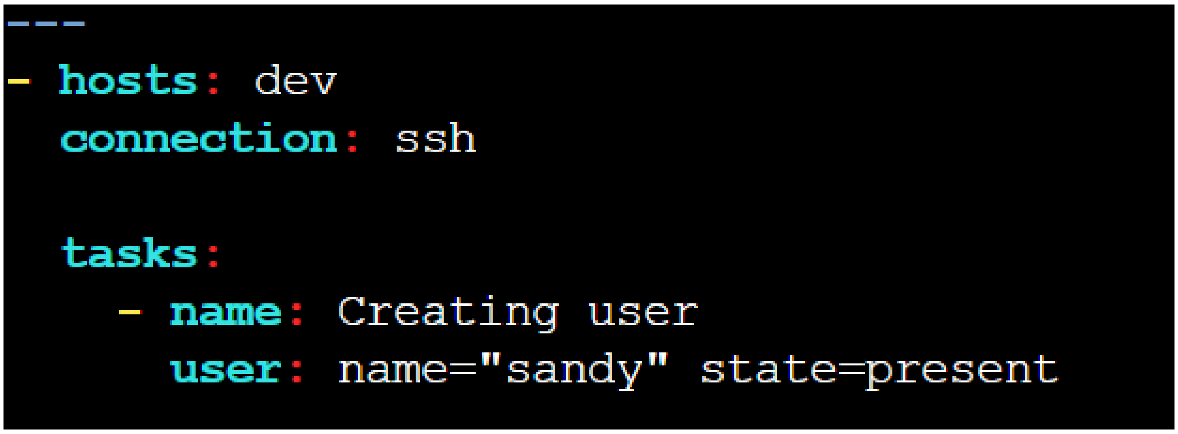
```
handlers:
  - name: restart the httpd
    service: name=httpd state=restarted
```

- So, here inside notify we're writing a message. Here, that message should be same as inside handlers name
- If it's same handlers will execute success, otherwise it throws error

The difference between Handlers & Playbooks is

- Playbooks are executed in sequential order eg: 1,2,3.......
- But Handlers are also same but when we gave notify. Eg: 1,2,3,H,4... like that
  - So, here after 2nd task I gave notify. So, 1st ,2nd execute then after handler again 3,4 will execute
- Overall execution is different

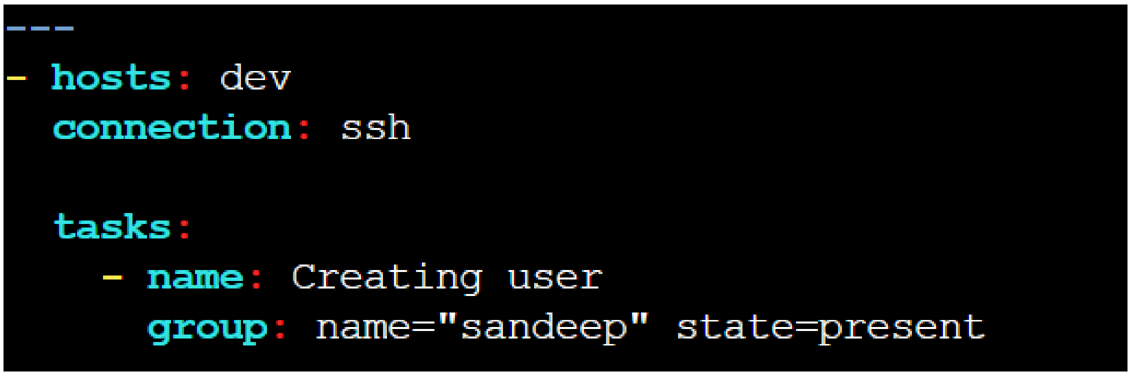
Create User through Playbook



```
---
- hosts: dev
  connection: ssh

  tasks:
    - name: Creating user
      user: name="sandy" state=present
```

Execute this playbook and check the user in slave server → cat /etc/passwd

Create Group through Playbook

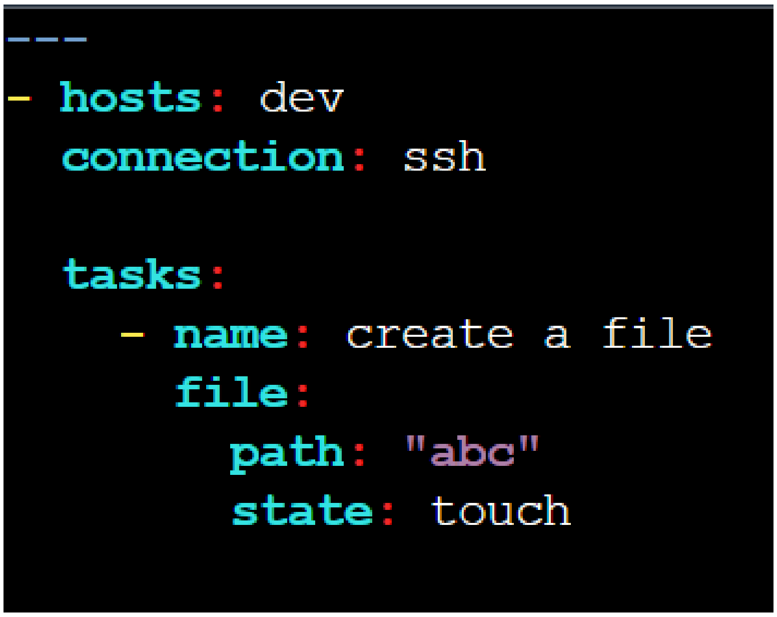


```
---
- hosts: dev
  connection: ssh

  tasks:
    - name: Creating user
      group: name="sandeep" state=present
```

Execute this playbook and check the group in slave server → cat /etc/group

Creating the files to remote/slave server

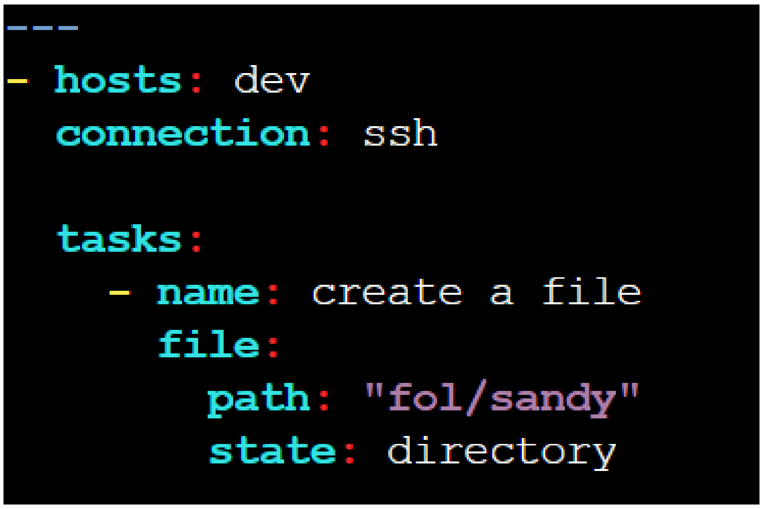
So, here we are using file module



```
---
- hosts: dev
  connection: ssh

  tasks:
    - name: create a file
      file:
        path: "abc"
        state: touch
```

Here, we are creating a "abc" file in slave server root directory

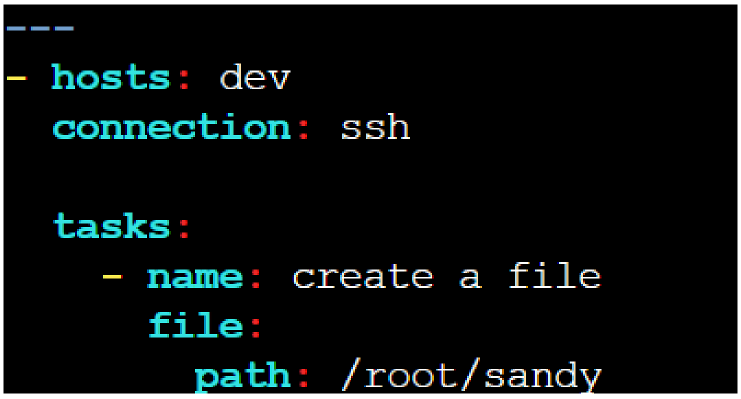
Creating the folders to remote/slave server



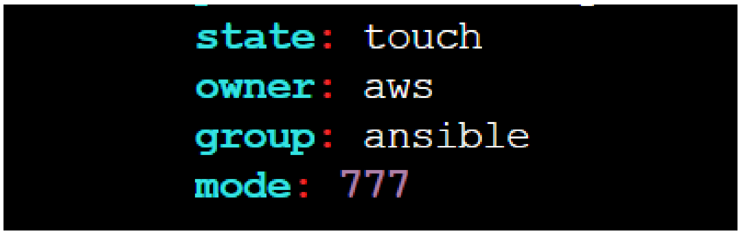
```
---
- hosts: dev
  connection: ssh

  tasks:
    - name: create a file
      file:
        path: "fol/sandy"
        state: directory
```

Here, we are creating the folders. In the path we created a "sandy" folder in particular folder

Creating the custom file, with changing owners and permissions to slave server



```
---
- hosts: dev
  connection: ssh

  tasks:
    - name: create a file
      file:
        path: /root/sandy
```

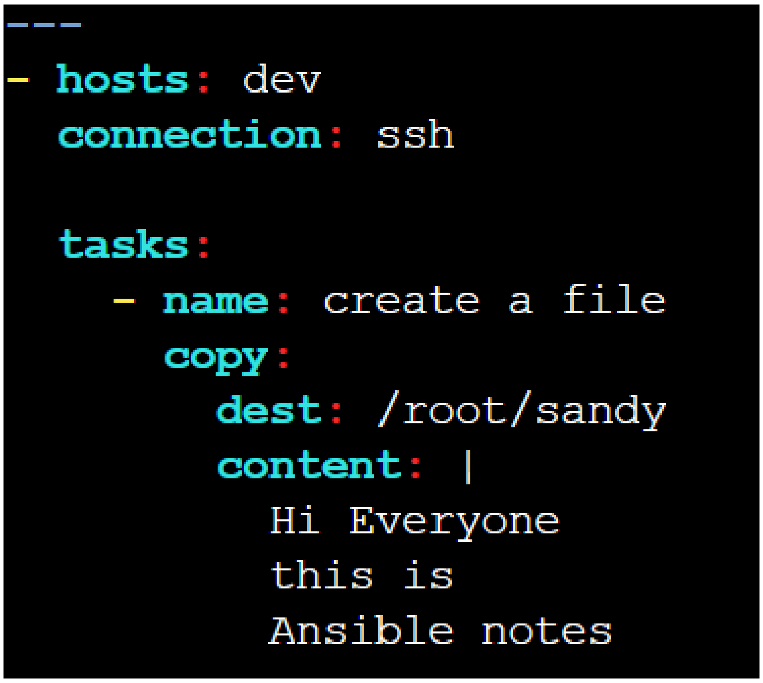

```
        state: touch
        owner: aws
        group: ansible
        mode: 777
```

Here, we are creating the files based on our requirements by changing the owners and permissions

Insert data into a file

- Here, we are using "copy" module.
- It will sends the data from master server to slave server
- Overall, Directly file copied from server to server

Now, write the data for existing file

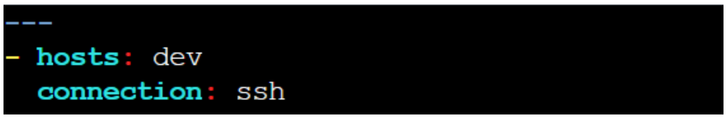

```
---
- hosts: dev
  connection: ssh

  tasks:
    - name: create a file
      copy:
        dest: /root/sandy
        content: |
          Hi Everyone
          this is
          Ansible notes
```

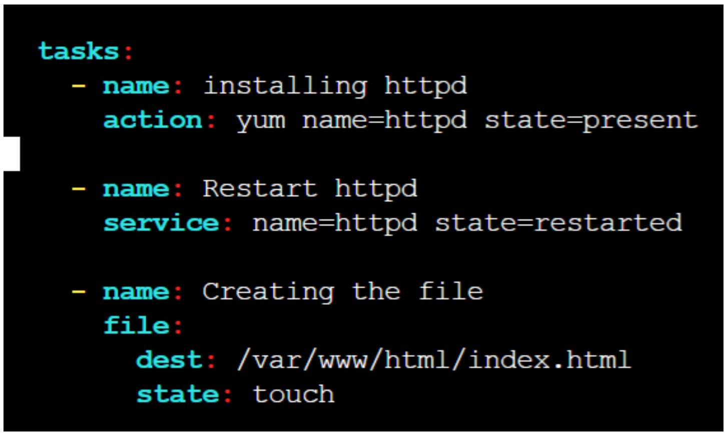
- So, here 'dest' means path of the file, Content used to insert some data into a file
- | (pipe symbol) write means, Inside the content how the data we wrote, we can see the data same like this.
- If we remove pipe symbol means, we can see the data in a single line

Note :

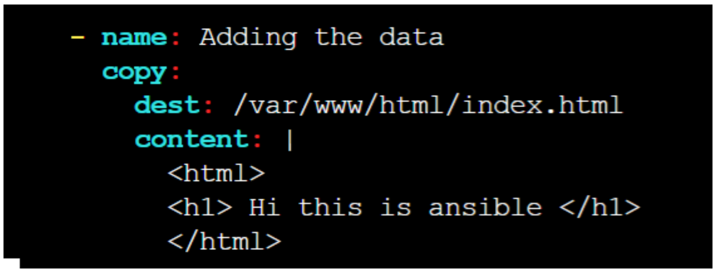
1. We can't create folder & file at a time like(file inside folder)
2. But 1st create a folder, then after execution we can create a file in that same playbook
3. Simultaneously folder and file we can create using multiple tasks

Creating a web server and insert the data in a file


```
---
- hosts: dev
  connection: ssh
```



```
tasks:
  - name: installing httpd
    action: yum name=httpd state=present

  - name: Restart httpd
    service: name=httpd state=restarted

  - name: Creating the file
    file:
      dest: /var/www/html/index.html
      state: touch
```



```
  - name: Adding the data
    copy:
      dest: /var/www/html/index.html
      content: |
        <html>
        <h1> Hi this is ansible </h1>
        </html>
```

After Completion of the code. Now, check in slave and go to index.html and check in browser

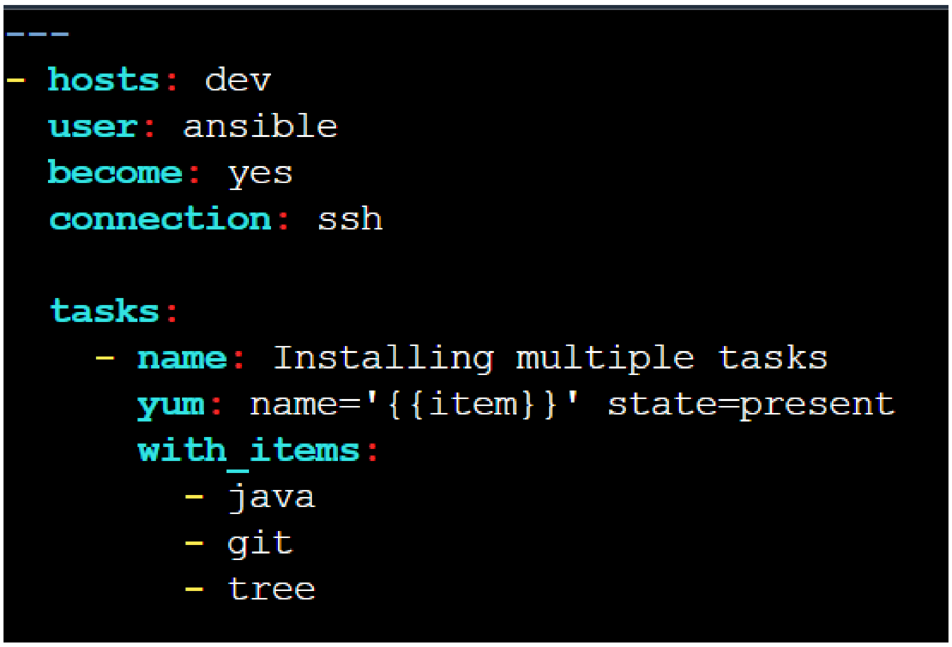
This is the way we can add the data in web server file through playbook

Loops

If we want to perform multiple actions in ansible we used Loops

Example -1 :

1. Create a multiple users for slave in master. If we want to create 3 (or) 4 users means we can do manually, but what if we want to create 100 users. So, that's why here we are using loops

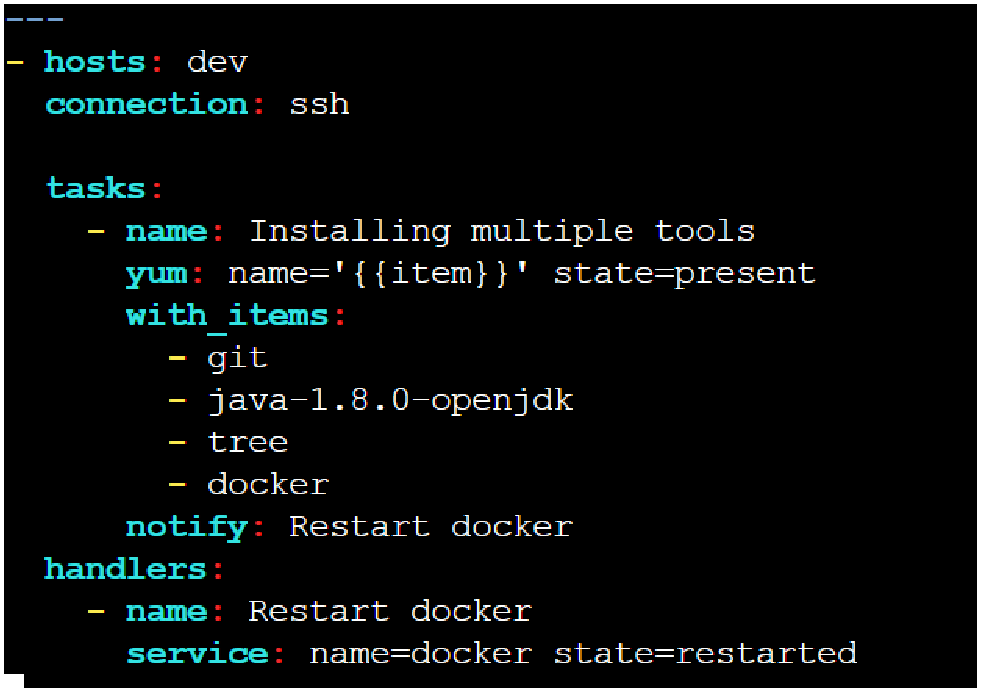


```
---
- hosts: dev
  user: ansible
  become: yes
  connection: ssh

  tasks:
    - name: Installing multiple tasks
      yum: name='{{item}}' state=present
      with_items:
        - java
        - git
        - tree
```

So, we defined the pre-defined variable '{{item}}' . For better understanding see the below code

2. Using Loops we can install multiple tools

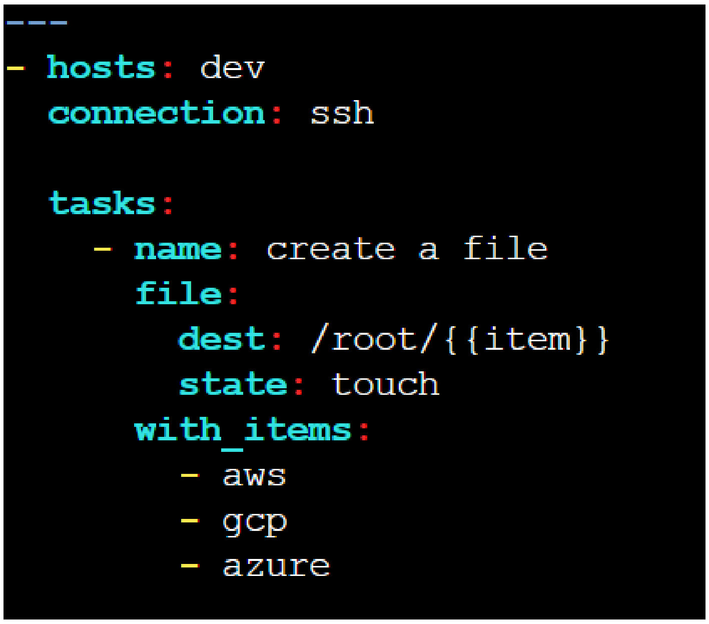
- So, right now using Loops and handlers how to install multiple tools



```
---
- hosts: dev
  connection: ssh

  tasks:
    - name: Installing multiple tools
      yum: name='{{item}}' state=present
      with_items:
        - git
        - java-1.8.0-openjdk
        - tree
        - docker
      notify: Restart docker
  handlers:
    - name: Restart docker
      service: name=docker state=restarted
```

- Execute the playbook and check in slave server

3. Create multiple files inside a folders/directories

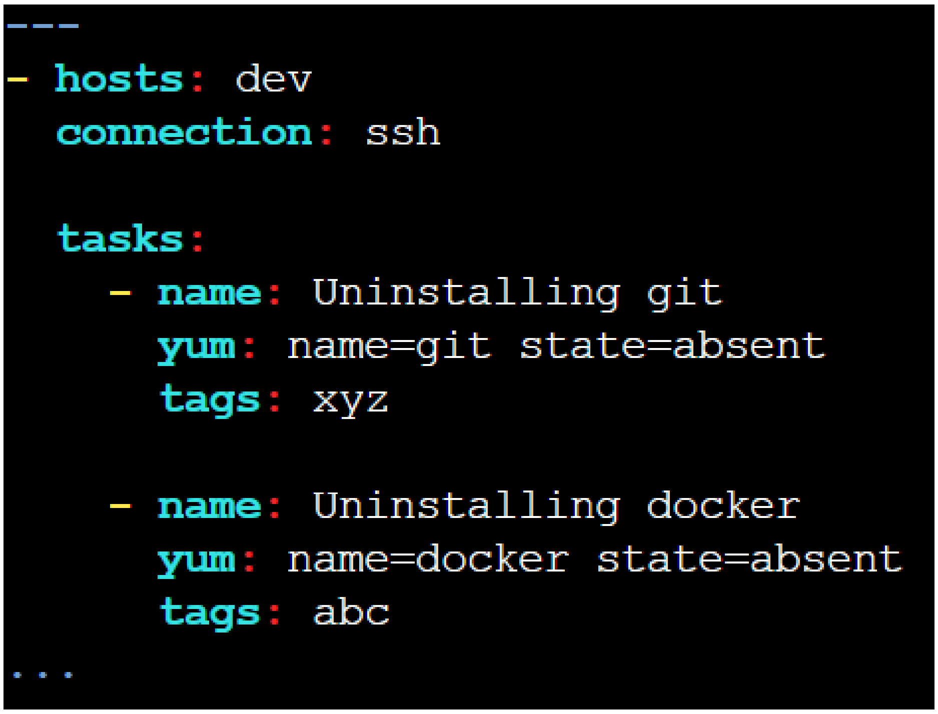


```
---
- hosts: dev
  connection: ssh

  tasks:
    - name: create a file
      file:
        dest: /root/{{item}}
        state: touch
      with_items:
        - aws
        - gcp
        - azure
```

So, like this we can install multiple tasks based on our requirement

Tags

- Tags are used to Skip the tasks
- We can perform action/tags in Runtime
- It won't work on loops because that contains multiple tasks



```
---
- hosts: dev
  connection: ssh

  tasks:
    - name: Uninstalling git
      yum: name=git state=absent
      tags: xyz

    - name: Uninstalling docker
      yum: name=docker state=absent
      tags: abc
...
```

So, in the above code we gave "tags" for every task.

- [ ] Now, I don't want to execute 1st task ie.., git task. So, the command is
    - ansible-playbook filename.yml --skip-tags "xyz"
- [ ] I want to skip multiple tasks in playbook
    - ansible-playbook filename.yml --skip-tags "xyz,abc"

When, exactly tags are using means, I want to create a file. When we execute playbook first time, file will created. Second time run means fail, because already file created. So, that situations we can skip the task using tags concept.

Roles

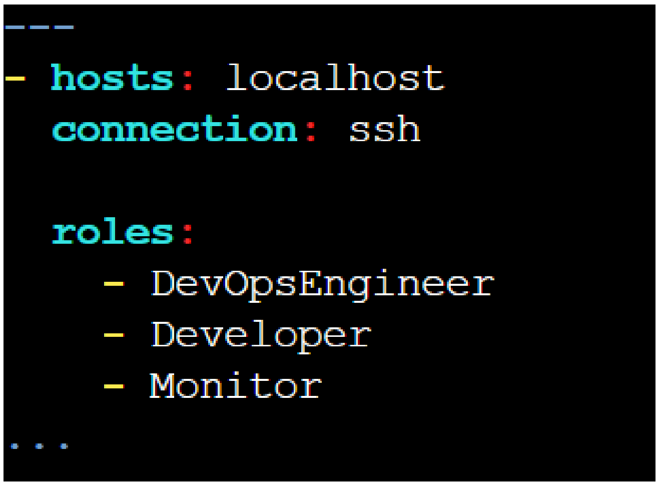
- Roles let **you automatically load related vars, files, tasks, handlers, and other Ansible** artifacts based on a known file structure
- Roles provide a framework for fully independent, or interdependent collections of variables, tasks, files, templates, and modules.
- In Ansible, the role is the primary mechanism for breaking a playbook into multiple files.
- This simplifies writing complex playbooks, and it makes them easier to reuse.

So, here we are creating/defining roles

Step -1 :

1. Create folder → mkdir roles → cd roles
2. Inside roles folder, create file name called main.yml → touch main.yml

3. Inside the main.yml file we are defining roles → vi main.yml



```
---
- hosts: localhost
  connection: ssh

  roles:
     - DevOpsEngineer
     - Developer
     - Monitor
...
```

4. Inside roles, we have to create folders ie.., In main.yml Inside the roles whatever the names we gave, with the same names we have to create folders

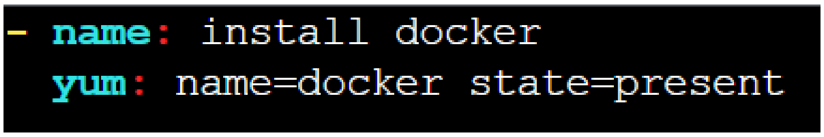
5. Creating the folders, command is → mkdir DevOpsEngineer Developer Monitor

Step -2 :

- Go to individual folder and create tasks for every folder
- Inside the tasks folder we write the playbook with the filename is "main.yml".
- Here In this "main.yml" file, no need to mention target section. Because it's already mentioned in "main.yml" inside roles folder.
- So, directly we can write the playbooks
- Use the same file name for all folders

Perform the below commands inside roles folder

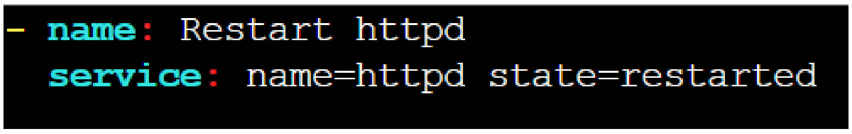
1. cd DevOpsEngineer → mkdir tasks → cd tasks → vi main.yml → save the code & exit



```
- name: install docker
  yum: name=docker state=present
```

- Here, we're installing a docker and we have to restart. So, for that purpose we can create another folder named as handlers. Inside handlers we create and we write the service code

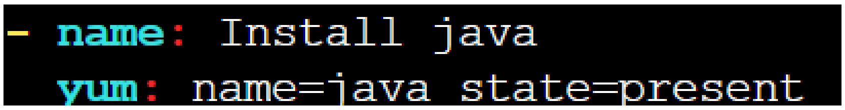
☐ cd DevOpsEngineer → mkdir handlers → cd handlers → vi main.yml → save the code & exit



```
- name: Restart httpd
  service: name=httpd state=restarted
```

So, we wrote tasks for DevOpsEngineer role

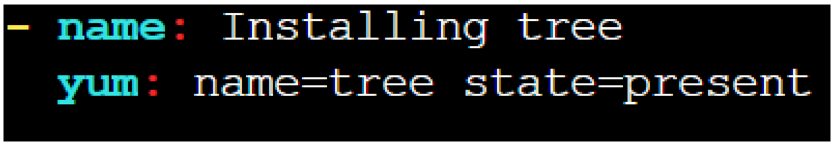
2. cd Developer → mkdir tasks → cd tasks → vi main.yml → save the code & exit



```
- name: Install java
  yum: name=java state=present
```

Here, we wrote tasks for Developer role

3. cd Monitor → mkdir tasks → cd tasks → vi main.yml → save the code & exit



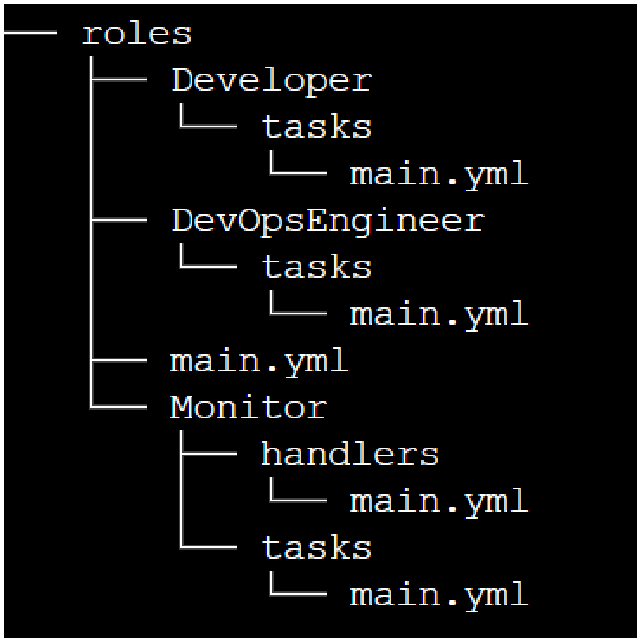
```
- name: Installing tree
  yum: name=tree state=present
```

Here, we wrote tasks for Monitor role

Step -3 :

Go to roles folder → cd roles

- After wrote the tasks for all the roles. Come to the roles folder ie.., cd roles
- Now, execute the main.yml file in roles folder → ansible-playbook main.yml
  - In roles → Inside main.yml file whatever the order we gave, In the same order the roles inside tasks will execute
  - After execute everything. We can see the roles file structure, using "tree" command

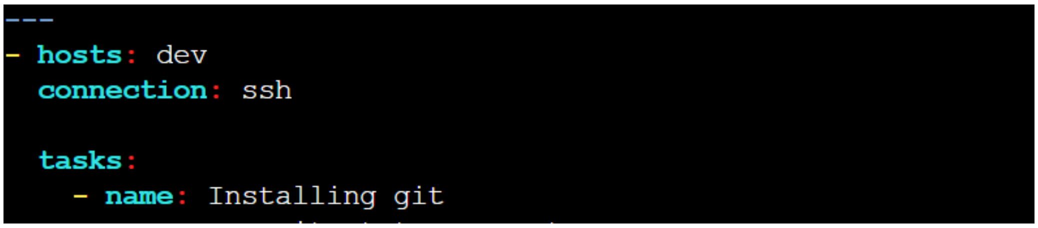


```
── roles
    ├── Developer
    │   └── tasks
    │       └── main.yml
    ├── DevOpsEngineer
    │   └── tasks
    │       └── main.yml
    ├── main.yml
    └── Monitor
        ├── handlers
        │   └── main.yml
        └── tasks
            └── main.yml
```

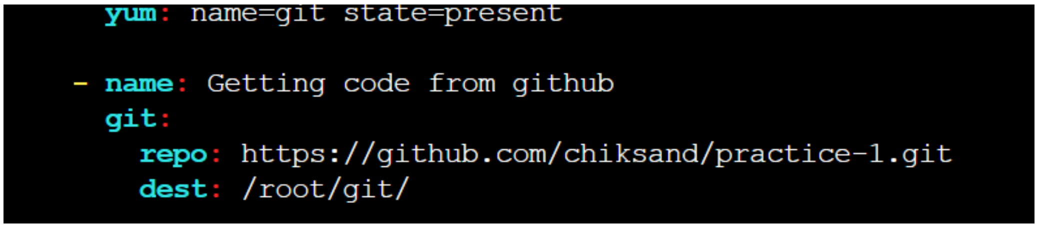
This is the way we are having file structure in roles

Getting the git repo in slave servers

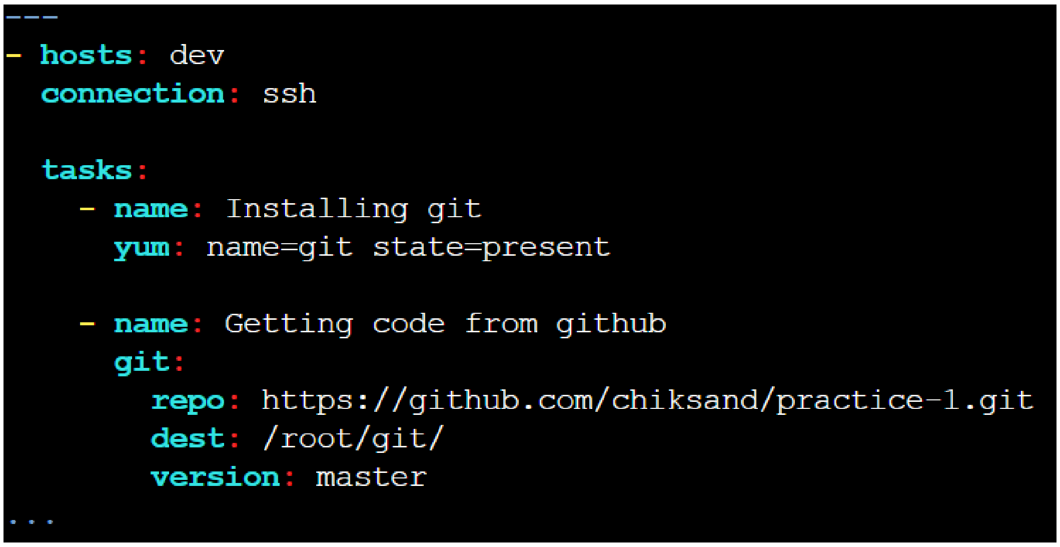
For Public repo:



```
---
- hosts: dev
  connection: ssh

  tasks:
    - name: Installing git
```

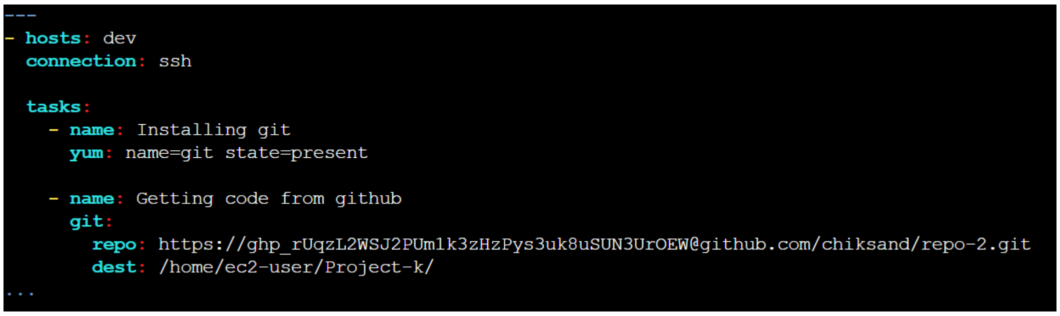

```
    yum: name=git state=present

  - name: Getting code from github
    git:
      repo: https://github.com/chiksand/practice-1.git
      dest: /root/git/
```

Get the repo for particular branch


```
---
- hosts: dev
  connection: ssh

  tasks:
    - name: Installing git
      yum: name=git state=present

    - name: Getting code from github
      git:
        repo: https://github.com/chiksand/practice-1.git
        dest: /root/git/
        version: master
...
```

Here, we are using "version" to get the particular branch repo

For Private repo:


```
---
- hosts: dev
  connection: ssh

  tasks:
    - name: Installing git
      yum: name=git state=present

    - name: Getting code from github
      git:
        repo: https://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@github.com/chiksand/repo-2.git
        dest: /home/ec2-user/Project-k/
...
```

Here, we are using "github-token" for getting the private repo. In playbook In repo module we paste the github token

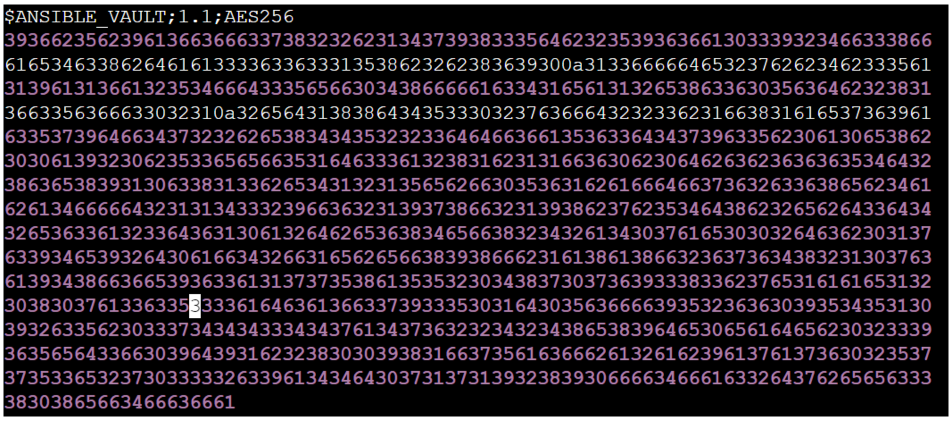
Syntax in repo: https://github-token@github.com/username/reponame.git

Ansible-vault

- Ansible Vault is a feature of ansible that allows you to keep sensitive data such as passwords or keys in encrypted files, rather than as plaintext in playbooks or roles.
- These vault files can then be distributed or placed in source control.
- Here, we are doing encryption for files. So, it converts file data into numbers format.

Encrypt the file:

- ansible-vault encrypt filename
- Create a password
  - Note: Don't forgot the password. If you forgot you'll lose all the data. We can't retrieve the password.
- Now, open the file and check, you'll see the below image



```
$ANSIBLE_VAULT;1.1;AES256
39366235623961366366633738323262313437393833356462323539363661303339323466333866
6165346338626461613333633633313538623262383639300a313366666646532376262346233561
31396131366132353466643335656630343866666163343165613132653863363035636462323831
3663356366633032310a32656431383864343533303237636664323233623166383161653736396 1
63353739646634373232626538343435323233646466366135363364343739633562306130653862
30306139323062353365656635316463336132383162313166363062306462636236363635346432
38636538393130633831336265343132313565626630353631626166646637363263363865623461
62613466666643231313433323966363231393738663231393862376235346438623265626433643 4
32653633613233643631306132646265363834656638323432613430376165303032646362303137
63393465393264306166343266316562656638393866623161386138663236373634383231303763
61393438663665393633613137373538613535323034383730373639333833623765316161653132
30383037613363353333361646361366337393335303164303563666663935323636303935343531 30
39326335623033373434343334343761343736323234323438653839646530656164656230323339
36356564336630396439316232383030393831663735616366626132616239613761373630323537
37353365323730333332633961343464303731373139323839306666346661633264376265656333
38303865663466636661
```

Decrypt the file:

- ansible-vault decrypt filename
- Asking password

So, this is the way we can encrypt and decrypt the file in Ansible-vault

Adhoc Commands

- Ansible adhoc commands are CLI commands used for simple and one-time tasks.
- One-time tasks include checking whether the nodes are reachable over SSH, shutting down all nodes, and so on.
- Overall, without writing a playbook you want to perform a task with single command we use adhoc commands
- If you want to perform tasks Individual, quick, (or) speed we use adhoc
- Inside the master server we're performing the adhoc commands

Syntax: ansible groupname -a "command"

- ansible dev -a "yum install git -y"

Modules

- A module is a reusable, standalone script that Ansible runs on your behalf, either locally or remotely.
- Here, also we are writing single line of module
- In adhoc commands, modules are different and they are having different syntax
- Eg of modules: yum, action, copy, etc..,

Syntax: ansible groupname -b -m yum -a "pkg=git state=present"

-b → become yes (when you create any user use this, For some tasks it's not mandatory)

-m → It represents module

yum → In this place we can write the type of module

-a → attribute

pkg → package name

Eg-1: Install tree

- ansible dev -b -m yum -a “pkg=tree state=present”

Eg-2: Copy a file from master to slave

- ansible dev -b -m copy -a “src=filename dest=/home/filename”

Difference between playbook and module (or) ansible ?

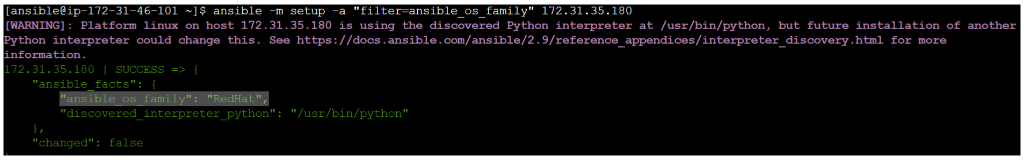
- We can store the playbooks and reuse it.
- But in modules/Adhoc commands it is not possible to reuse because it is single-line commands
- Modules can be written in any language, Ansible only ships with python and powershell modules.

Eg: When we restart the server, this commands will be not there history is clean. So, for that case, we are using playbooks

Ansible Setup Modules

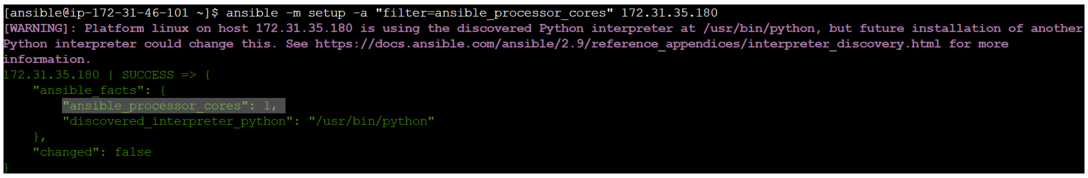
Based on the server states we’re performing the below commands

- ansible_os_family - We can get OS name like RedHat, ubuntu, etc..,



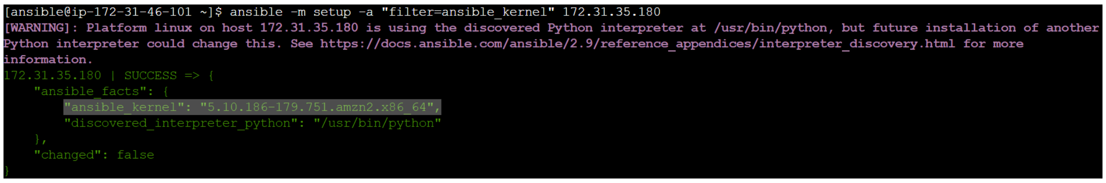
```
[ansible@ip-172-31-46-101 ~]$ ansible -m setup -a "filter=ansible_os_family" 172.31.35.180
[WARNING]: Platform linux on host 172.31.35.180 is using the discovered Python interpreter at /usr/bin/python, but future installation of another
Python interpreter could change this. See https://docs.ansible.com/ansible/2.9/reference_appendices/interpreter_discovery.html for more
information.
172.31.35.180 | SUCCESS => {
    "ansible_facts": {
        "ansible_os_family": "RedHat",
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false
```

- ansible_processor_cores - We can get no.of CPU cores



```
[ansible@ip-172-31-46-101 ~]$ ansible -m setup -a "filter=ansible_processor_cores" 172.31.35.180
[WARNING]: Platform linux on host 172.31.35.180 is using the discovered Python interpreter at /usr/bin/python, but future installation of another
Python interpreter could change this. See https://docs.ansible.com/ansible/2.9/reference_appendices/interpreter_discovery.html for more
information.
172.31.35.180 | SUCCESS => {
    "ansible_facts": {
        "ansible_processor_cores": 1,
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false
}
```

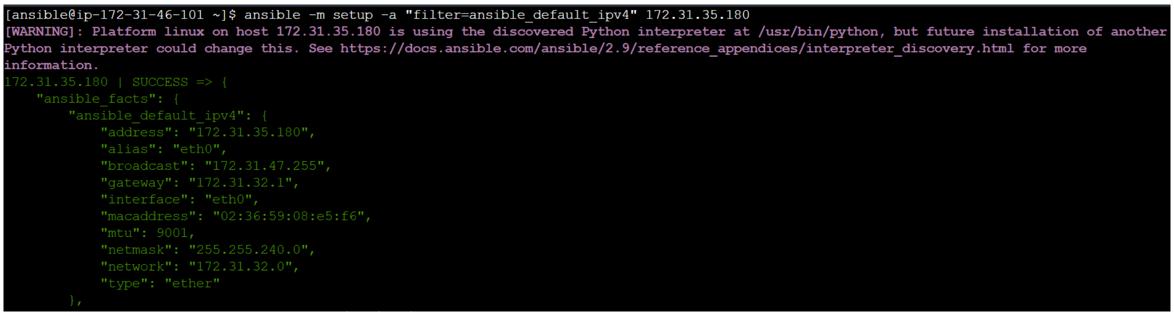
- ansible_kernel - We can get servers on the kernel version



```
[ansible@ip-172-31-46-101 ~]$ ansible -m setup -a "filter=ansible_kernel" 172.31.35.180
[WARNING]: Platform linux on host 172.31.35.180 is using the discovered Python interpreter at /usr/bin/python, but future installation of another
Python interpreter could change this. See https://docs.ansible.com/ansible/2.9/reference_appendices/interpreter_discovery.html for more
information.
172.31.35.180 | SUCCESS => {
    "ansible_facts": {
        "ansible_kernel": "5.10.186-179.751.amzn2.x86_64",
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false
}
```

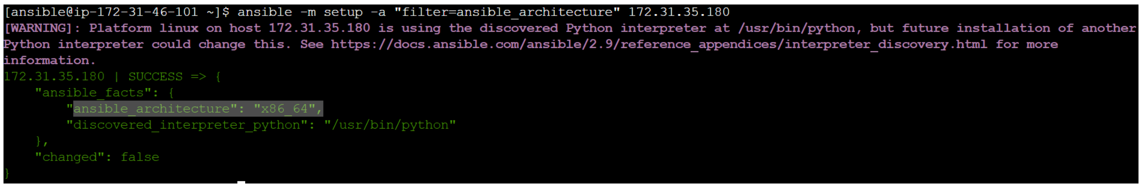
- ansible_devices - We can get connected devices information (ram, memory, etc..,)

- ansible_default_ipv4 - We can get IP Mac address, Gateway



```
[ansible@ip-172-31-46-101 ~]$ ansible -m setup -a "filter=ansible_default_ipv4" 172.31.35.180
[WARNING]: Platform linux on host 172.31.35.180 is using the discovered Python interpreter at /usr/bin/python, but future installation of another
Python interpreter could change this. See https://docs.ansible.com/ansible/2.9/reference_appendices/interpreter_discovery.html for more
information.
172.31.35.180 | SUCCESS => {
    "ansible_facts": {
        "ansible_default_ipv4": {
            "address": "172.31.35.180",
            "alias": "eth0",
            "broadcast": "172.31.47.255",
            "gateway": "172.31.32.1",
            "interface": "eth0",
            "macaddress": "02:36:59:08:e5:f6",
            "mtu": 9001,
            "netmask": "255.255.240.0",
            "network": "172.31.32.0",
            "type": "ether"
        },
```

- ansible_architecture - We can get 64 BIT (or)) 32 BIT



```
[ansible@ip-172-31-46-101 ~]$ ansible -m setup -a "filter=ansible_architecture" 172.31.35.180
[WARNING]: Platform linux on host 172.31.35.180 is using the discovered Python interpreter at /usr/bin/python, but future installation of another
Python interpreter could change this. See https://docs.ansible.com/ansible/2.9/reference_appendices/interpreter_discovery.html for more
information.
172.31.35.180 | SUCCESS => {
    "ansible_facts": {
        "ansible_architecture": "x86_64",
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false
}
```

Eg-1: Check the slave servers OS family

- ansible -m setup -a "filter=ansible_os_family" serverPrivateIPAddress

Eg-2: Get server information device

- ansible -m setup -a "filter=ansible_devices" privateIPAddress
- So, here we get the device info like memory,ram, etc..,

Mostly here we are using "ansible_os_family"

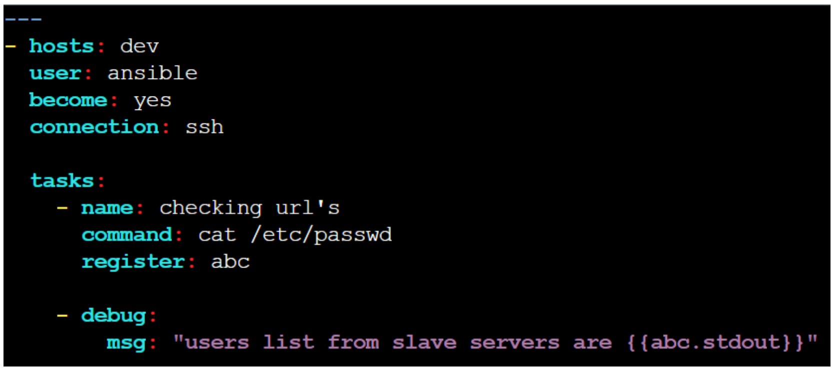
DeBug

If we are performing any task, you want to see the output in the runtime. We are using debug

Ansible debug is used to get the logs from the slave server while performing the action. So, for that case, we are using 'debug' mode

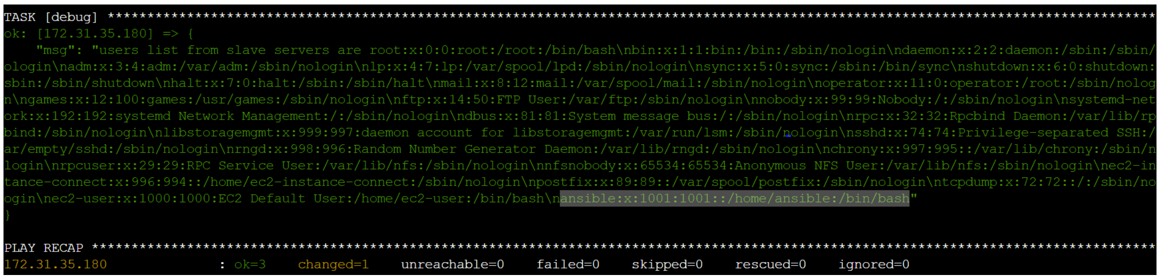
Eg: If we write the playbook and execute the code, usually we can see the output in slave server. But using debug 'run' we can see the output in master

In Debug 'register' is used to store the data. If we want to print the store data we use debug module ie.., using stdout. "msg" is the module is used to print



```
---
- hosts: dev
  user: ansible
  become: yes
  connection: ssh

  tasks:
    - name: checking url's
      command: cat /etc/passwd
      register: abc

    - debug:
        msg: "users list from slave servers are {{abc.stdout}}"
```

So, it will print in JSON Format = {”abc” = x }



```
TASK [debug] ******************************************************************************************************************************************
ok: [172.31.35.180] => {
    "msg": "users list from slave servers are root:x:0:0:root:/root:/bin/bash\nbin:x:1:1:bin:/bin:/sbin/nologin\ndaemon:x:2:2:daemon:/sbin:/sbin/
ologin\nadm:x:3:4:adm:/var/adm:/sbin/nologin\nlp:x:4:7:lp:/var/spool/lpd:/sbin/nologin\nsync:x:5:0:sync:/sbin:/bin/sync\nshutdown:x:6:0:shutdown:
sbin:/sbin/shutdown\nhalt:x:7:0:halt:/sbin:/sbin/halt\nmail:x:8:12:mail:/var/spool/mail:/sbin/nologin\noperator:x:11:0:operator:/root:/sbin/nolog
n\ngames:x:12:100:games:/usr/games:/sbin/nologin\nftp:x:14:50:FTP User:/var/ftp:/sbin/nologin\nnobody:x:99:99:Nobody:/:/sbin/nologin\nsystemd-net
ork:x:192:192:systemd Network Management:/:/sbin/nologin\ndbus:x:81:81:System message bus:/:/sbin/nologin\nrpc:x:32:32:Rpcbind Daemon:/var/lib/rp
bind:/sbin/nologin\nlibstoragemgmt:x:999:997:daemon account for libstoragemgmt:/var/run/lsm:/sbin/nologin\nsshd:x:74:74:Privilege-separated SSH:/
ar/empty/sshd:/sbin/nologin\nrngd:x:998:996:Random Number Generator Daemon:/var/lib/rngd:/sbin/nologin\nchrony:x:997:995::/var/lib/chrony:/sbin/n
login\nrpcuser:x:29:29:RPC Service User:/var/lib/nfs:/sbin/nologin\nnfsnobody:x:65534:65534:Anonymous NFS User:/var/lib/nfs:/sbin/nologin\nec2-in
tance-connect:x:996:994::/home/ec2-instance-connect:/sbin/nologin\npostfix:x:89:89::/var/spool/postfix:/sbin/nologin\ntcpdump:x:72:72::/:/sbin/no
ogin\nec2-user:x:1000:1000:EC2 Default User:/home/ec2-user:/bin/bash\nansible:x:1001:1001::/home/ansible:/bin/bash"
}

PLAY RECAP ********************************************************************************************************************************************
172.31.35.180              : ok=3    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

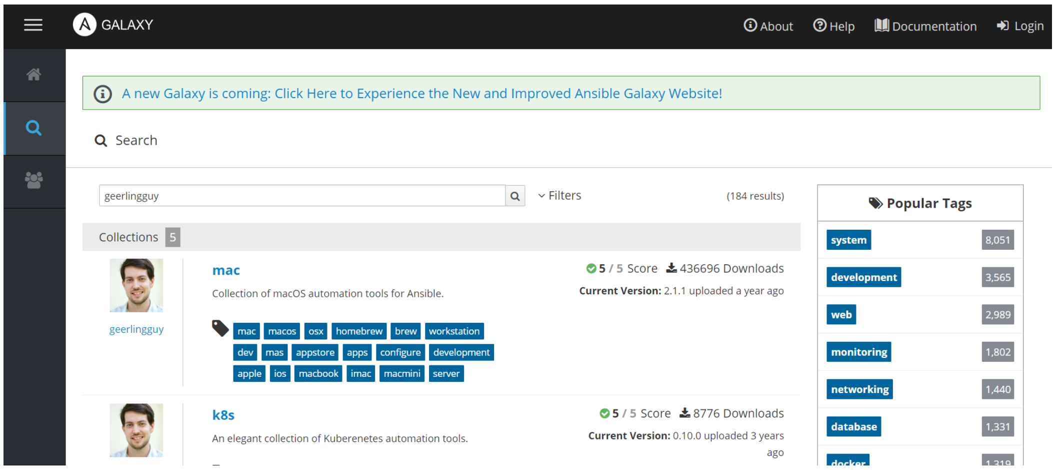
So, same like we can get server IP address and everything we can use with the ansible setup modules

Ansible - GALAXY

- We are using ansible to write the roles. We need to do centralize the roles, for that purpose we’re using ansible galaxy
- Ansible Galaxy is a free online repository that contains a vast collection of Ansible roles, modules and playbooks
- It is a centralized hub for Ansible automation content ie., open-source, community-driven
- It is maintained by RedHat Company
- Ansible Galaxy allows users to search, download and share Ansible roles, modules, and playbooks with the wider ansible community

So, here Go to browser → Search Ansible galaxy → we got interface

- It is Galaxy interface we don’t need to do any account creation
- Here, Go to system, and click on search, “Geerlingguy” he is very famous for writing the playbooks
- Once, you got the data it will be showing like below image



GALAXY — About · Help · Documentation · Login

A new Galaxy is coming: Click Here to Experience the New and Improved Ansible Galaxy Website!

Search

geerlingguy — Filters — (184 results)

Collections 5

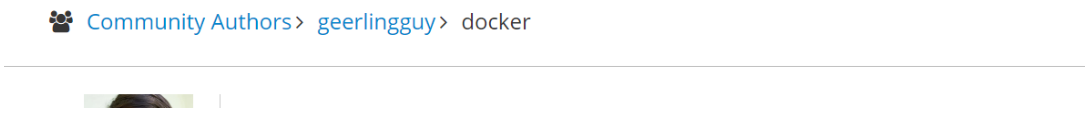
mac — 5 / 5 Score — 436696 Downloads
Collection of macOS automation tools for Ansible.
Current Version: 2.1.1 uploaded a year ago
geerlingguy
mac · macos · osx · homebrew · brew · workstation · dev · mas · appstore · apps · configure · development · apple · ios · macbook · imac · macmini · server

k8s — 5 / 5 Score — 8776 Downloads
An elegant collection of Kuberenetes automation tools.
Current Version: 0.10.0 uploaded 3 years ago

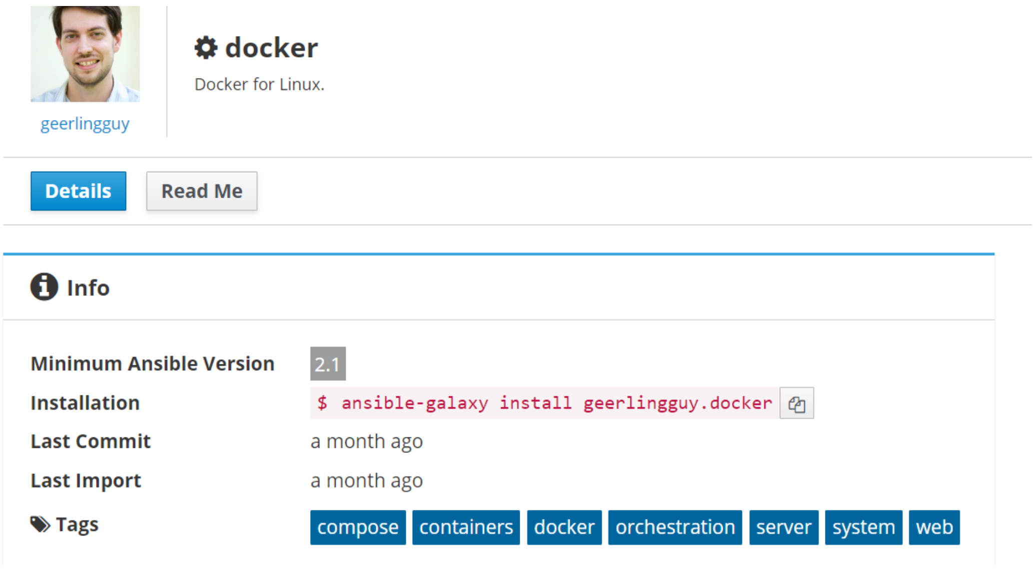
Popular Tags

system	8,051
development	3,565
web	2,989
monitoring	1,802
networking	1,440
database	1,331
docker	1,319

- click on any roles and open



Community Authors › geerlingguy › docker



docker

Docker for Linux.

Info

Minimum Ansible Version	2.1
Installation	`$ ansible-galaxy install geerlingguy.docker`
Last Commit	a month ago
Last Import	a month ago
Tags	compose containers docker orchestration server system web

- Here, copy that installation linke and paste in terminal. So, everything is download your terminal

I want to check in CLI

- So, in master server perform this command
  - ansible-galaxy search package name/python
- So,you want to get particular author playbooks/roles in CLI
  - ansible-galaxy search elasticsearch --author alikins/authorname
  - Now, playbooks came
    - Install the above playbook
    - ansible-galaxy install package/playbook name

So, overall others written roles for that downloading purpose we're using ansible-galaxy. It is a collection of playooks, modules, roles, adhoc commands, etc.., which is used to share with others