# Ansible

## History of automation

Perspective

1. Bash Scripting(linux task automation) / Batch Scripting (windows)

2. PERL /Python / Ruby 🡪programming languages

3. Powershell 🡪 window specific🡪 vmware related automation etc

4. Puppet (dsl 🡪 domain specific language)(ruby)=> pull model (configuration management tool) 🡪 config the cluster of web server

Ex: you have cluster of web server, from that cluster of web server you have configuration, so almost all the web servers in cluster are exactly similar configuration, package, service status etc. but some time we used to make changes to those servers and they use to go out of sync. Some configuration file changes (server goes down etc) are suppose to manged though a centralized server (puppet servers). Puppet server as the configuration of all your servers and server is apply like law and all the machines where we want to manage configuration will have the agent that will regularly query to the puppet server about its configuration and everything used to be in the right state

5. Salt Stack(python) => pull (to execute command on remotely)

Both combination of puppet and salt stack was very famous, general we run peppet though salt stack( initial set up though salt stack and configuration though puppet.it was like combination of multiple automation tools)

6. Chef => pull

Chef came with more manageability and more power to the user like writing ruby code and more template, the graphical user interface (we can generate reports etc). but along with this feature it came with more complications. They are to many moving parts in chef. You have server, client, workstation.

Puppet and chef is not about, it is matter of usecase and inclination towards it.

7. Ansible(python) => push

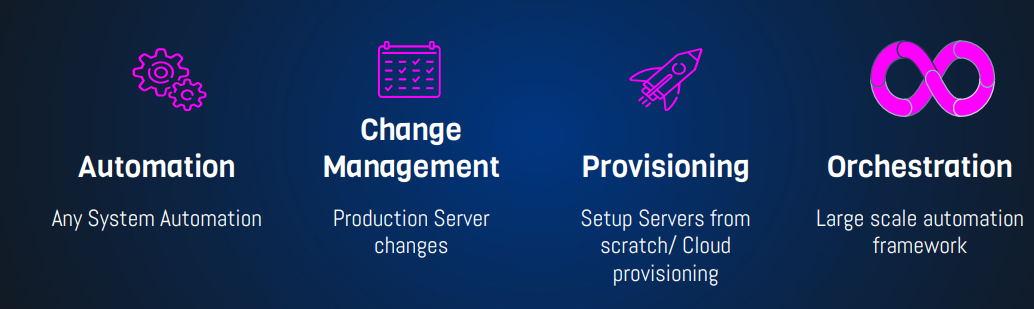
It is called a control machine in generate.

After Ansible came(again it configuration management tool developed by Mitchael Dohan. Ansible written in python and out of all , ansible is the most simplest automation tool). It was acquires by Red Hat then. Now we have lot of the things in ansible 🡪 ansible tower, enterprise versions etc. ansible came with an idea of simplicity, having a very simple code to manage your huge infrastructure . at first it focus in Linux machines and still windows automation also possible through ansible, cloud automation, then network tool automation, database automation. so much of integration came with Ansible now(being simple it become very powerful automation tool)

8. Terraform(cloud automation)

We can do cloud automation in ansible but terraform is more cloud specific automation tool

## Use cases



Ansible playbooks

## Ansible is Simple

### No Agents

### 🡪 in puppet , slat stack , you need to install agents in all those server but ansible have no agents

Target machines/Services are accessed by SSH, winrm & API

### No Databases

YAML, INI & Texts

### No Complex Setup

It’s just a Python Library

### No Residual Software

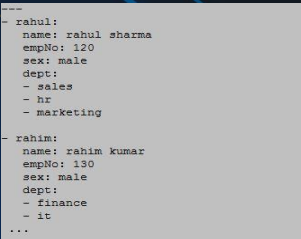
Push Python package

Execute

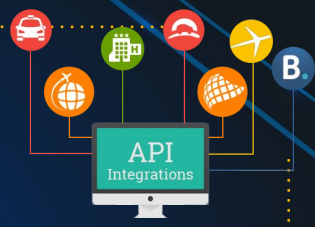
Return Output

# YAML

* No Programming 🡪 written in ymal format
* Structured
* Easy to Read & Write



## API

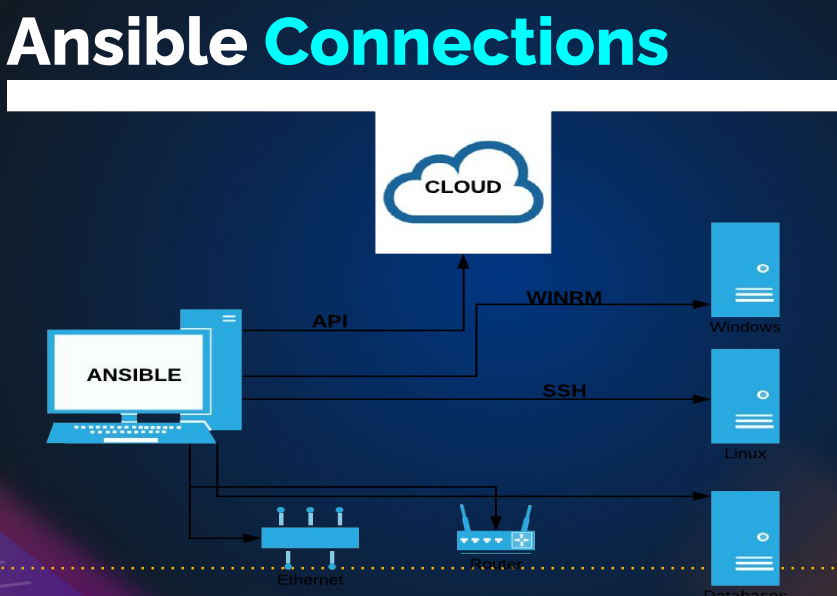


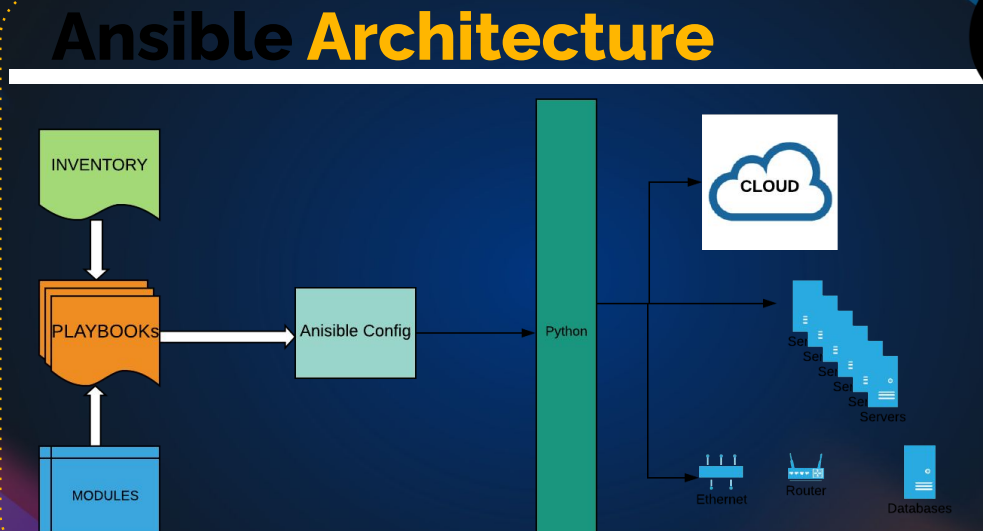
URL/Restful Calls(e:g Cloud)

Shell Commands

Scripts

Ansible has lot of modules to do the particular task, apart from system modules it has a API based like, you can launch ec2 instance by using ansible ansible use some API based tools to connect. (for aws **boto** that used API) . we can also use exiting shell command and window PowerShell commands from ansible

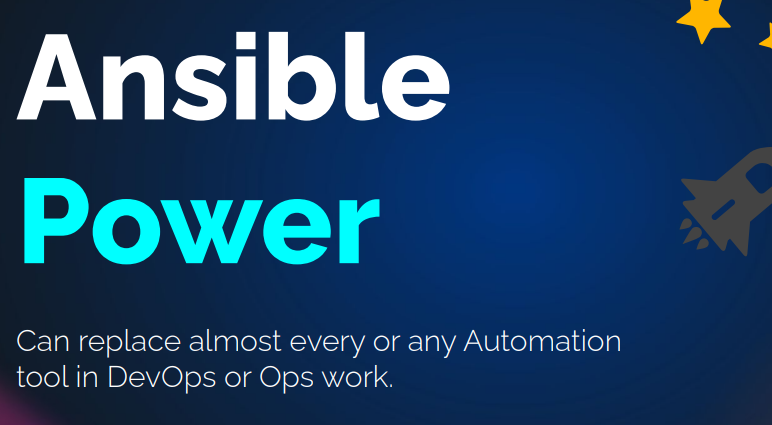




We have to create a inventory files. Inventory file we have the target machine information(like ip address, user name, password etc)

Ansible as lot of inbuild modules like installing packages, restarting service or taking snapshot of EBS volume. Etc(more then 1000’s modules)

In python🡪 group of function to create a modules🡪 group of modules to create python packages



# SET UP ANSIBLE & INFRA

Create a four EC2 instance in AWS:

* One is ubuntu instances got control server(for ansible)
* Other 3 is centos9 for the web servers(web01,web02,web03)

Target machines/Services are accessed by **SSH, winrm & API**

When you SSH the instances though git bash all the fingerprint stored in

$cat ~/.ssh/known\_hosts

// We can remove it by $cat /dev/null > ~/.ssh/known\_hosts

Similar ansible work in the same by without any agents

Ansible installation:

We can google it

$ sudo apt update

$ sudo apt install software-properties-common

$ sudo add-apt-repository --yes --update ppa:ansible/ansible

$ sudo apt install ansible

# INVENTORY & PING MODULES

## Write inventory

They are two way to write INI and YMAL Format

Document 🡪 <https://docs.ansible.com/ansible/latest/inventory_guide/intro_inventory.html>

Mkdir -r vprofile/exercise1

Vim inventory

all:

hosts:

web01:

ansible\_host: 172.31.42.193

ansible\_user: ec2-user

ansible\_ssh\_private\_key\_file: clientkey.pem

Vim clientkey.pem

Copy the client.pem(it genareated in ec2) key to clientkey.pem(ansible will use the key to access the server)

cd /etc/ansible/

mv ansible.cfg ansible.cfg\_backup

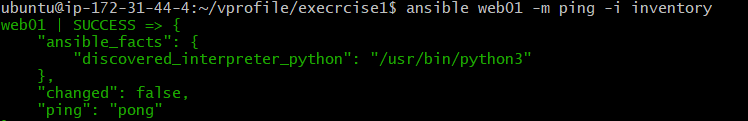
vim ansible.cfg 🡪 copy the command “ansible-config init --disabled -t all > ansible.cfg”

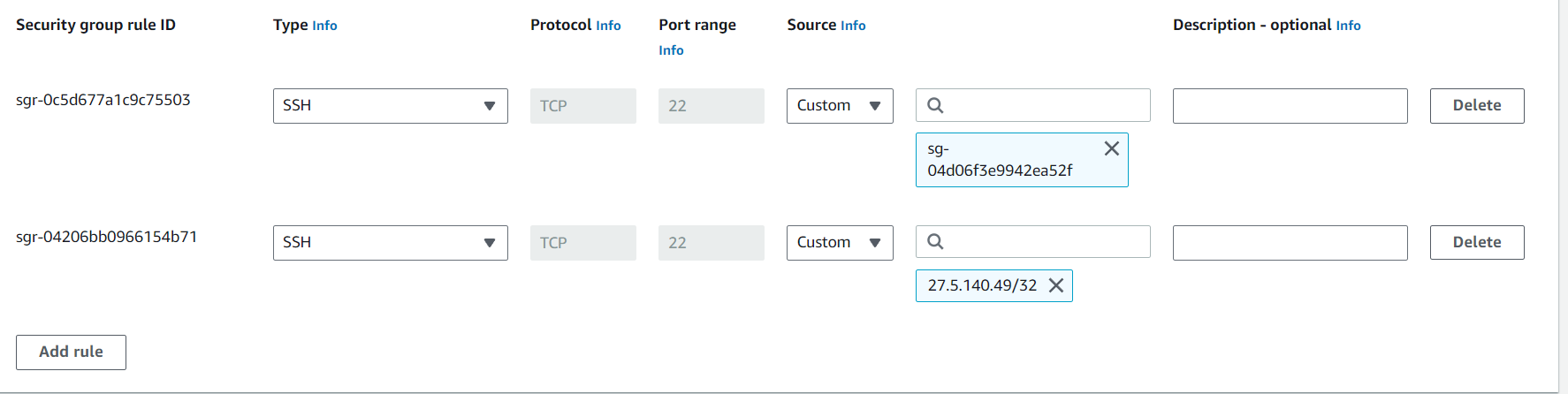
ansible-config init --disabled -t all > ansible.cfg > ansible.cfg

vim ansible.cfg

uncommont and change to false to host\_key\_checking=false

change the permission to chmod 400 ansible.cfg is needed





Sg group 🡪 one is myip

* Second one 22 port to control-sg

Ansible web01 -m ping -I inventory ( -m =’module’ -I =’information’)

# INVENTORY PART 2

In this we will know about group in inventory file

all:

hosts:

web01:

ansible\_host: 172.31.42.193

ansible\_user: ec2-user

ansible\_ssh\_private\_key\_file: clientkey.pem

web02:

ansible\_host: 172.31.37.109

ansible\_user: ec2-user

ansible\_ssh\_private\_key\_file: clientkey.pem

db01:

ansible\_host: 172.31.42.3

ansible\_user: ec2-user

ansible\_ssh\_private\_key\_file: clientkey.pem

children:

webservers:

hosts:

web01:

web02:

dbservers:

hosts:

db01:

dc\_oregen:

children:

webservers:

dbservers:

Ansible **webservers** -m ping -i inventory

Ansible **dbservers** -m ping -i inventory

Ansible **dc\_oregen** -m ping -i inventory

Ansible **all** -m ping -i inventory

Ansible **‘\*’** -m ping -i inventory

Ansible **web\*** -m ping -i inventory

Note: variable of hosts level 🡪 high priority

Variable of group level 🡪 lower priority compare to hosts level

all:

hosts:

web01:

ansible\_host: 172.31.42.193

web02:

ansible\_host: 172.31.37.109

db01:

ansible\_host: 172.31.42.3

children:

webservers:

hosts:

web01:

web02:

dbservers:

hosts:

db01:

dc\_oregen:

children:

webservers:

dbservers:

vars:

ansible\_user: ec2-user

ansible\_ssh\_private\_key\_file: clientkey.pem

Variable is placed at dc\_oregon group level so (it first check the hosts level then it come to group level)

# YMAL & JSON

1. **Syntax**:
   * **Python**: Python uses a specific syntax to define data structures. For example, dictionaries are represented using curly braces, lists using square brackets, and key-value pairs using colons.
   * **YAML**: YAML (YAML Ain't Markup Language) uses indentation to represent data structures. It's designed to be human-readable and uses spaces and colons to indicate structure.
   * **JSON**: JSON (JavaScript Object Notation) uses a more strict and minimalistic syntax. It uses curly braces to define objects and square brackets for arrays, with key-value pairs separated by colons.
2. **Human-Readability**:
   * **Python**: Python data structures are not inherently human-readable, as they are meant to be executed as code.
   * **YAML**: YAML is designed to be human-readable, making it easy for people to write and understand.
   * **JSON**: JSON is moderately human-readable, but its primary focus is on being easy for machines to parse.
3. **Data Types**:
   * **Python**: Python data structures support a wide range of data types and can represent complex objects and custom classes.
   * **YAML**: YAML has support for common data types, but it may not be as flexible as Python for custom data types.
   * **JSON**: JSON supports a limited set of data types, including strings, numbers, booleans, arrays, and objects.
4. **Comments**:
   * **Python**: Python allows for comments using the '#' symbol.
   * **YAML**: YAML allows comments using the '#' symbol as well.
   * **JSON**: JSON does not support comments, making it less suitable for documentation within the data.
5. **Extensibility**:
   * **Python**: Python data structures can be easily extended to include custom methods and behavior.
   * **YAML**: YAML is not designed for extensibility and is primarily focused on data serialization.
   * **JSON**: JSON does not support custom methods or behavior; it's mainly a data interchange format.
6. **Usage**:
   * **Python**: Used for programming and defining data structures within Python code.
   * **YAML**: Often used for configuration files, data serialization, and human-readable structured data exchange.
   * **JSON**: Widely used for data interchange between systems, including web APIs and configuration data for applications.

python json ymal

name: John

age: 30

city: New York

{

"name": "John",

"age": 30,

"city": "New York"

}

data = {

"name": "John",

"age": 30,

"city": "New York"

}

Ex:2

Python:

person = {

"name": "Alice",

"age": 25,

"address": {

"street": "123 Main St",

"city": "Wonderland"

},

"interests": ["reading", "programming"]

}

PYTHON JSON

{

"name": "David",

"age": 28,

"city": "San Francisco"

}

[

{

"name": "Alice",

"age": 30

},

{

"name": "Bob",

"age": 22

}

]

YMAL

server:

port: 8080

hostname: example.com

ssl: true

database:

name: mydb

username: user

password: secret

books:

- title: "The Hobbit"

author: "J.R.R. Tolkien"

- title: "To Kill a Mockingbird"

author: "Harper Lee"

people = [

{"name": "Bob", "age": 30},

{"name": "Charlie", "age": 22}

]

<https://docs.ansible.com/ansible/latest/reference_appendices/YAMLSyntax.html>

# Ad Hoc Commands

<https://docs.ansible.com/ansible/latest/command_guide/intro_adhoc.html>

## [Why use ad hoc commands?](https://docs.ansible.com/ansible/latest/command_guide/intro_adhoc.html#id4)

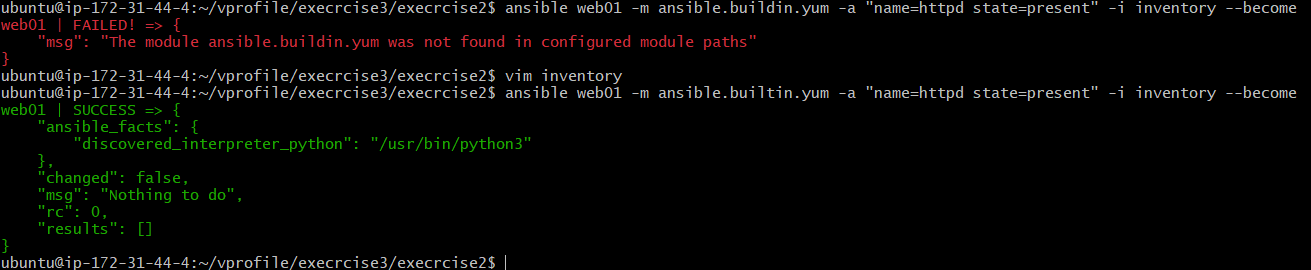
ad hoc commands are great for tasks you repeat rarely. For example, if you want to power off all the machines in your lab for Christmas vacation, you could execute a quick one-liner in Ansible without writing a playbook. An ad hoc command looks like this:

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

The -a option accepts options either through the key=value syntax or a JSON string starting with { and ending with } for more complex option structure. You can learn more about [patterns](https://docs.ansible.com/ansible/latest/inventory_guide/intro_patterns.html#intro-patterns) and [modules](https://docs.ansible.com/ansible/6/user_guide/modules.html#working-with-modules) on other pages.

## [Use cases for ad hoc tasks](https://docs.ansible.com/ansible/latest/command_guide/intro_adhoc.html#id5)

ad hoc tasks can be used to reboot servers, copy files, manage packages and users, and much more. You can use any Ansible module in an ad hoc task. ad hoc tasks, like playbooks, use a declarative model, calculating and executing the actions required to reach a specified final state. They achieve a form of idempotence by checking the current state before they begin and doing nothing unless the current state is different from the specified final state.



**To install the service in servers**:

ansible web01 -m ansible.builtin.yum -a "name=httpd state=present" -i inventory –become

ansible webservers -m ansible.builtin.yum -a "name=httpd state=present" -i inventory –become

ansible webservers -m ansible.builtin.yum -a "name=httpd state=absent" -i inventory –become

difference b/w scripting and configuration management:

configuration management tools maintain the state of the target 🡪idempotent

configuration management tools are idempotent(it the target is in the different state then only it can apply the changes, same state it not apply the changes(scripts and command is not going to fail)

**services commands**

ansible webservers -m ansible.builtin.service -a "name=httpd state= started enabled=yes" -i inventory –become

ansible web01 -m ansible.builtin.service -a "name=httpd state=started enabled=yes" -i inventory --become

ansible web01 -m ansible.builtin.service -a "src=index.html dest=/var/www/html/index.html" -i inventory –become

# Playbook & modules

🡪Ansible playbook/playbooks is similar to the bash script/python script like that in ansible playbook.

🡪Playbook is the collection of play’s

Single play with single tast:

-host: websrvgrp

Tasks:

-yum:

Name: httpd

State: present

Example:

---

- name: Copy File to Remote Server

hosts: your\_target\_server

become: yes # To escalate privileges using sudo (if necessary)

tasks:

- name: Copy a File

copy:

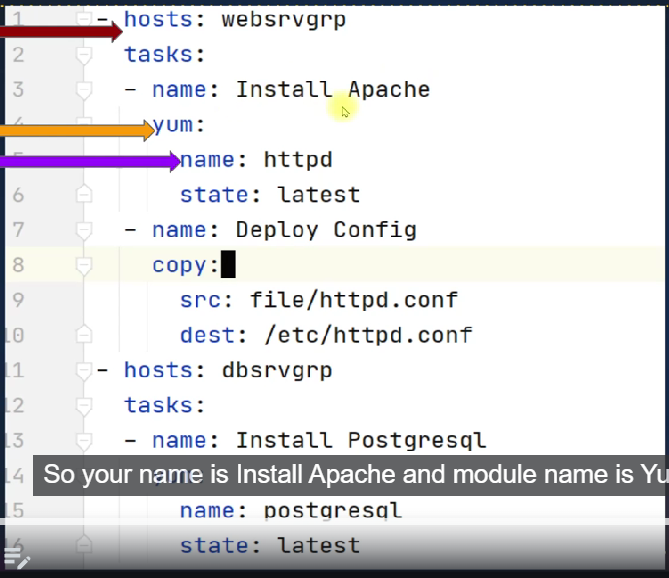
src: /path/to/local/file.txt # Local file path

dest: /path/to/remote/ # Remote directory path

owner: your\_remote\_user # Remote user

group: your\_remote\_group # Remote group

mode: 0644 # Permissions for the copied file



YUM

Playbooks:

---

- name: webserver setup

hosts: webservers

become: yes

tasks:

- name: install httpd

ansible.builtin.yum:

name: httpd

state: present

- name: start service

ansible.builtin.service:

name: httpd

state: started

enabled: yes

- name: DBserver setup

hosts: dbservers

become: yes

tasks:

- name: install mariadb-server

ansible.builtin.yum:

name: mariadb-server

state: present

$ ansible '\*' -m ansible.builtin.yum -a "name=httpd state=absent" -i inventory --become

$ansible-playbook -i inventory web-db.yaml

Playbooks vs adhoc commands

## Debugging:

In the playbook run command they are 4 levels of debugging in each level in see more information(with max level(4th level) you can see the keys, login key etc)

$ ansible-playbook -i inventory web-db.yaml -v

ansible-playbook -i inventory web-db.yaml -vv

ansible-playbook -i inventory web-db.yaml -vvv

ansible-playbook -i inventory web-db.yaml -vvvv

it can be used to find logical error

## syntax error

normally we run the command if any syntax error is it though the error

but we can also use

ansible-playbook -i inventory –syntax-check

## test run/dry run

ansible-playbook -I inventory -C

* It is called as dry run

## Document ansible

<https://docs.ansible.com/ansible/latest/>

<https://docs.ansible.com/ansible/latest/collections/index_module.html>

# MODULES – FIND, USE, TROUBLESHOOT & REPEAT

DOCUMENT – ANSIBLE MODULE INDEX: <https://docs.ansible.com/ansible/2.8/modules/modules_by_category.html>

TASK:

* Copying a file to control machine to web server
* To add database and create database users

Being devops, our job will be majorly managing files, their configuration of file could be an archive, it could be an artifact, a text file, configuration file, scripts . we deal a lot with files.

File modules will help to execute those tasks.

---

- name: webserver setup

hosts: webservers

become: yes

tasks:

- name: install httpd

ansible.builtin.yum:

name: httpd

state: present

- name: start service

ansible.builtin.service:

name: httpd

state: started

enabled: yes

- name: Copy file

ansible.builtin.copy:

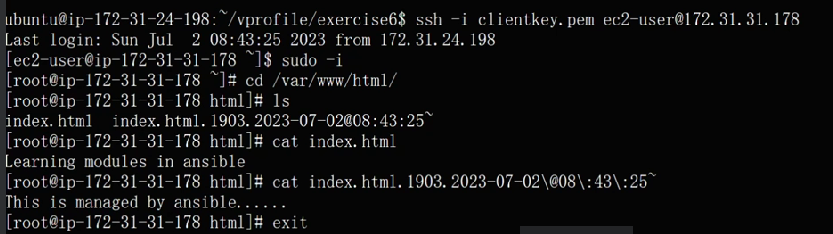
src: files/index.html

dest: /var/www/html/index.html

backup: yes

copy command simple it uses SCP command in the linux.

It will take the backup for index.html



2) create a database and add the user

---

- name: DBserver setup

hosts: dbservers

become: yes

tasks:

- name: install mariadb-server

ansible.builtin.yum:

name: mariadb-server

state: present

- name: mariadb-server

ansible.builtin.service:

name: mariadb

state: started

enabled: yes

- name: install PyMySQL

ansible.builtin.yum:

name: python3-PyMySQL

state: present

- name: Create database user with name 'bob' and password '12345' with all database privileges

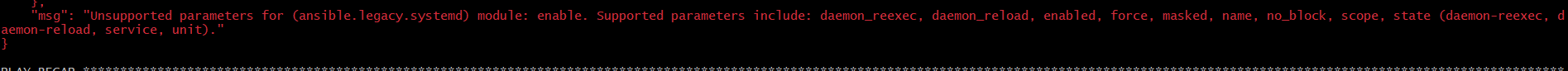
mysql\_user:

name: accounts

state: present

login\_unix\_socket: /var/lib/mysql/mysql.sock

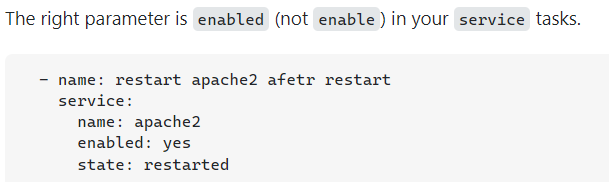
## Errors:



"msg": "Unsupported parameters for (ansible.legacy.systemd) module: enable. Supported parameters include: daemon\_reexec, daemon\_reload, enabled, force, masked, name, no\_block, scope, state (daemon-reexec, daemon-reload, service, unit)."

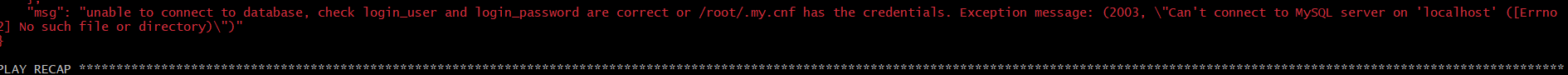
}

Check the syntax us correctly



2) "msg": "unable to connect to database, check login\_user and login\_password are correct or /root/.my.cnf has the credentials. Exception message: (2003, \"Can't connect to MySQL server on 'localhost' ([Errno 2] No such file or directory)\")"

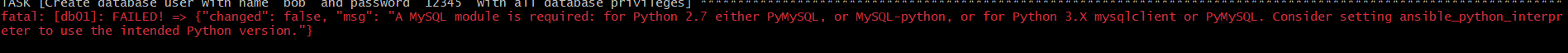
}



Add the sock

login\_unix\_socket: /var/lib/mysql/mysql.sock

3) fatal: [db01]: FAILED! => {"changed": false, "msg": "A MySQL module is required: for Python 2.7 either PyMySQL, or MySQL-python, or for Python 3.X mysqlclient or PyMySQL. Consider setting ansible\_python\_interpreter to use the intended Python version."}

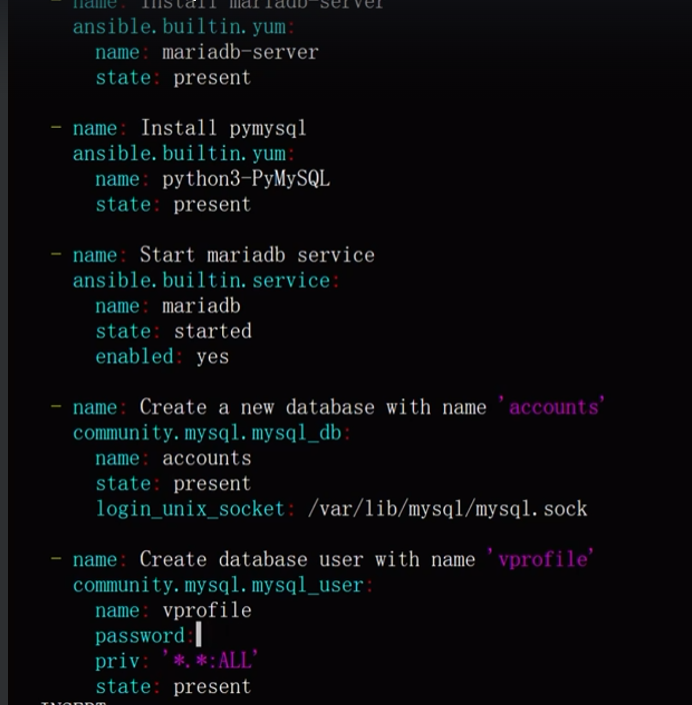


Install the dependencies

Note:

Always took for error and search the document (if you going to the community some time you will get the help🡪 they post common error etc)

<https://docs.ansible.com/ansible/latest/collections/community/mysql/index.html>



Sorry I missed last login\_unix\_socket: /var/lib/mysql/mysql.sock in the vprofile database

This is🡪 community.mysql.mysql\_user

# Ansible configuration setting

<https://docs.ansible.com/ansible/latest/installation_guide/intro_configuration.html>

config files is configuration description etc

* Normally ansible as default configuration

Order of ansible Config

1. Ansible config (environment variable if set)
2. Ansible.cfg (in the current directory)
3. ~/.ansible.cfg (in the home directory)
4. /etc/ansible/ansible.cfg 🡪 global ansible configuration

In the inventory & ping modules 🡪 we have generated the directed to the ansible.cfg,

In that they so many config

## Create a config file for logs

[defaults]

host\_key\_checking = False

inventory = ./inventory

forks = 5

log\_path = /var/log/ansible.log

[privilege\_escalation]

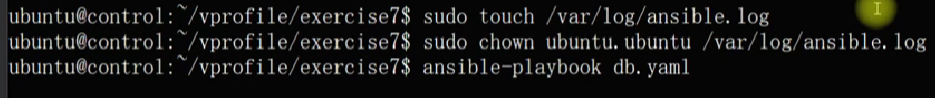
become=True

become\_method=sudo

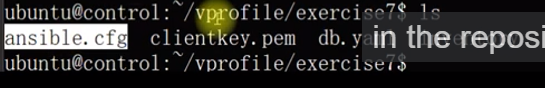
#become\_user=root

become\_ask\_pass=False

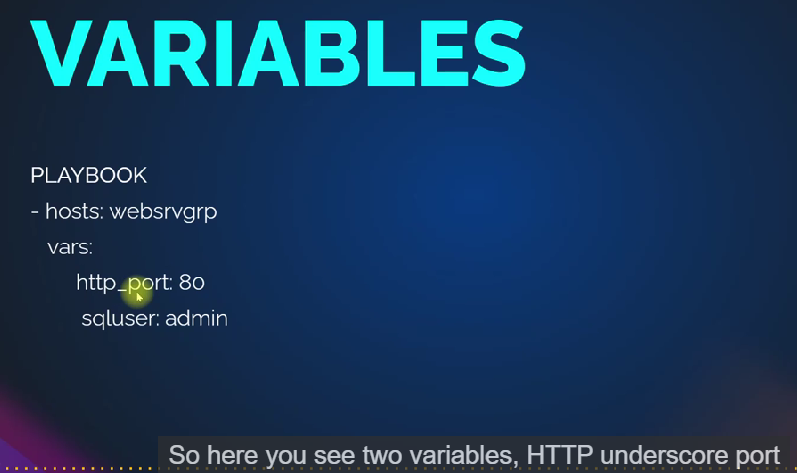
Change the permission in it ask any



Ansible.cfg(if you specify any) file need to present in the repository



# Variables



We can define the variable at play book 🡪 vars:

We can also seen the variable in inventory file , we can define the file group\_vars

Roles also they are two file vars and defaults



Now, those were our own custom variables.We are defining them. Ansible has also its own variables.

The majority of its variable gets generated from the setup module.

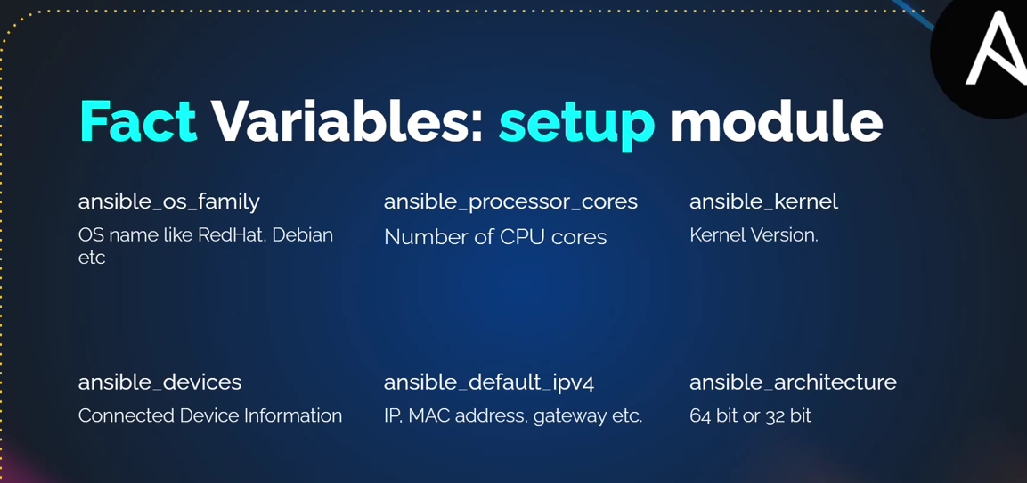
Now, you don't need to run setup module.When you execute the playbook,

the first task that gets executed, gathering facts,

that task will run the setup module

and generate the fact variables.

Example:





## Variables 🡪 playbook

---

- name: DBserver setup

hosts: dbservers

become: yes

vars:

dbname: electric

dbuser: current

dbpass: tesla

tasks:

- debug:

msg: "this is {{dbname}}"

- debug:

msg: dbuser

- name: install mariadb-server

ansible.builtin.yum:

name: mariadb-server

state: present

- name: mariadb-server

ansible.builtin.service:

name: mariadb

state: started

enabled: yes

- name: install PyMySQL

ansible.builtin.yum:

name: python3-PyMySQL

state: present

- name: Create database user with name 'bob' and password '12345' with all database privileges

community.mysql.mysql\_user:

name: "{{dbname}}"

state: present

login\_unix\_socket: /var/lib/mysql/mysql.sock

- name: Create database user 'vprofile'

community.mysql.mysql\_user:

name: "{{dbname}}"

password: "{{dbpass}}"

priv: '\*.\*: ALL'

state: present

login\_unix\_socket: /var/lib/mysql/mysql.sock

register: dbout

- name: print the dbout

debug:

var: dbout

## debug module

it is use for troubleshooting purpose only.

But you can we can print messages, variable(but playbook is very verbose)

# Group & host variables:

In this we learn about inventory variables 🡪 groups , hosts, common variable and priority.

mkdir group\_vars

vim group\_vars/all

dbname: sky

dbuser: pilot

dbpass: aircraft

if you run the play-book 🡪 ansible-play-book -I inventory web-db.yaml

but it will take variable from yaml file because🡪 yaml file variable have high priority (if you command the yaml file variable then it will go to the group\_vars/all

example:

- name: Understanding precedence of variables

  hosts: all

  become: yes

  gather\_facts: False

  vars:

    USRNM: playuser

    COMM: variable from playbook

  tasks:

    - name: create a user

      user:

        name: "{{USRNM}}"

        comment: "{{COMM}}"

      register: USROUT

    - debug:

        var: USROUT.name

    - debug:

        var: USROUT.comment

- name:understanding preceduce of variable

hosts: all

become: yes

vars:

USRNM: playuser

COMM: variable from playbook

tasks:

-name: create a user

user:

name: "{{USRNM}}"

comments: "{{COMM}}"

register: USROUT

- debug:

var: USROUT.name

- debug:

var: USROUT.comment

mkdir group\_vars

vim group\_vars/all

USRNM: common user

COMM: variable from groupvars\_all files

Like before playbook as higher priority so need to command in playbook the group\_vars/all will get the variable and execute it

Similarly :

vim group\_vars/webservers

USRNM: webservers

COMM: variable from group\_vars/webservers file

Now web01 and web02 will take the variable from group\_vars/webservers

Similarly:

mkdir host\_vars

vim host\_vars/host

USRNM: web02

COMM: variable from host\_vars/host file

Ansible-playbook -I inventory web-db.yaml

We we can see the web02 🡪 takes host\_vars/host

Web01🡪 take group\_vars/webservers

Db01 🡪 normally taken from group\_vars/all

## Priority :

Playbook inside variable has high priority

Host\_vars/host 🡪 next priority

Group\_vars/group 🡪 next priority

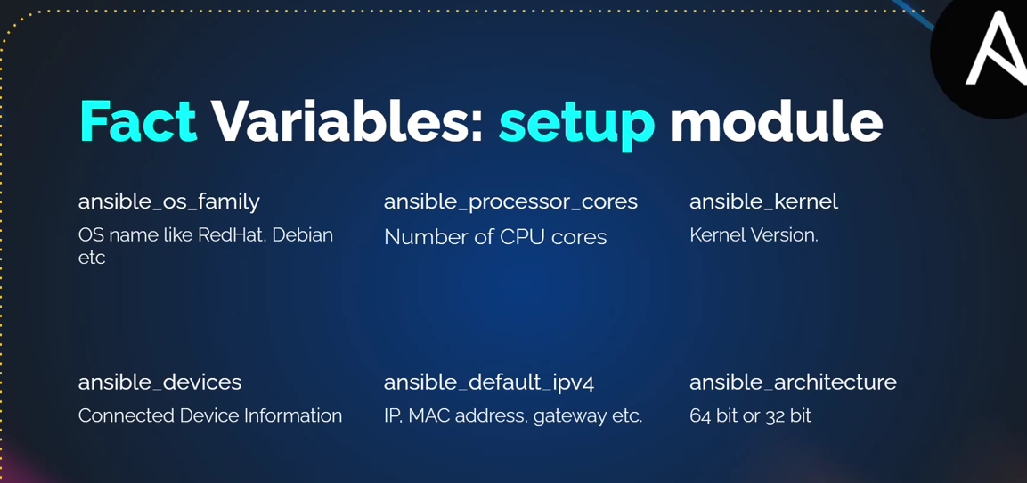
Group\_vars/all 🡪 next priority

but if you declare the variables in command line(cli) then all it take variable from command only 🡪 which makes the highest priority 🡪but it is use for testing purposes

ansible-playbook -e USRNM: cliuser -e COMM: cliuser -I inventory web-db.yaml

search 🡪 ansible variable : <https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_variables.html>

# FACT VARIABLE

Fact variable is run time variable, they get execute when setup model gets executed

When ever play-book execute by default, then ansible execute **gathering facts** task for all the host.

Gathering facts execute the module called setup. This module collects information in host in JSON format in only in runtime

We can disable by given 🡪 gather\_facts: False