# 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

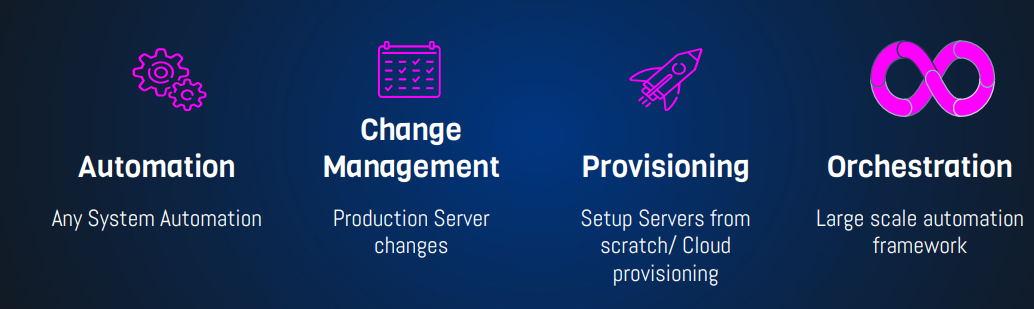
It is called a control machine in generate.

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

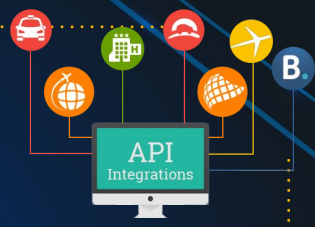
Return Output

# YAML

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



## API

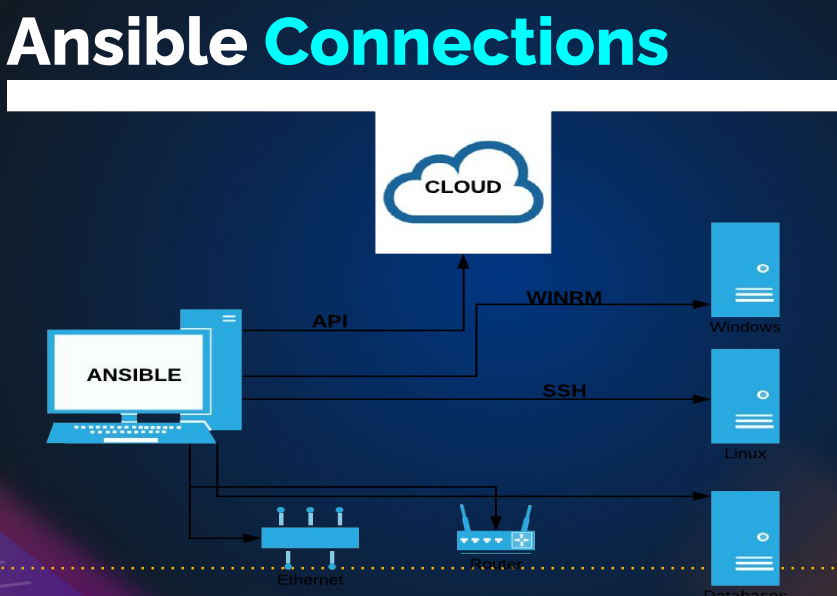


URL/Restful Calls(e:g Cloud)

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

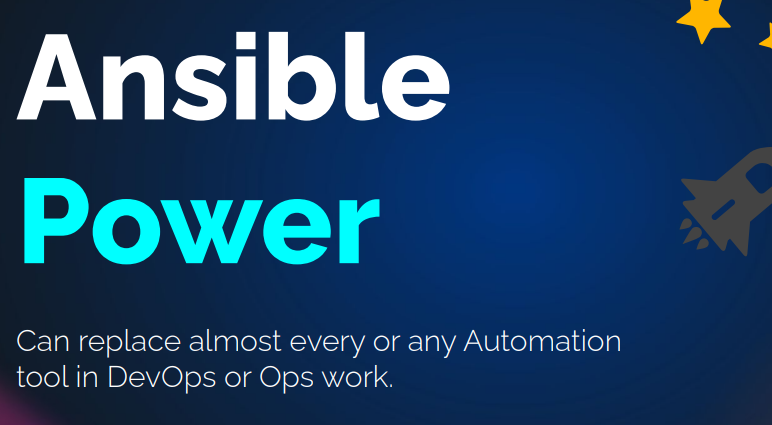




We have to create a inventory files. Inventory file we have the target machine information(like ip address, user name, password etc)

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

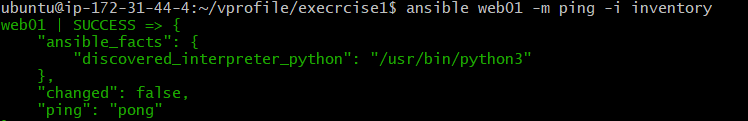
vim ansible.cfg 🡪 copy the command “ansible-config init --disabled -t all > ansible.cfg”

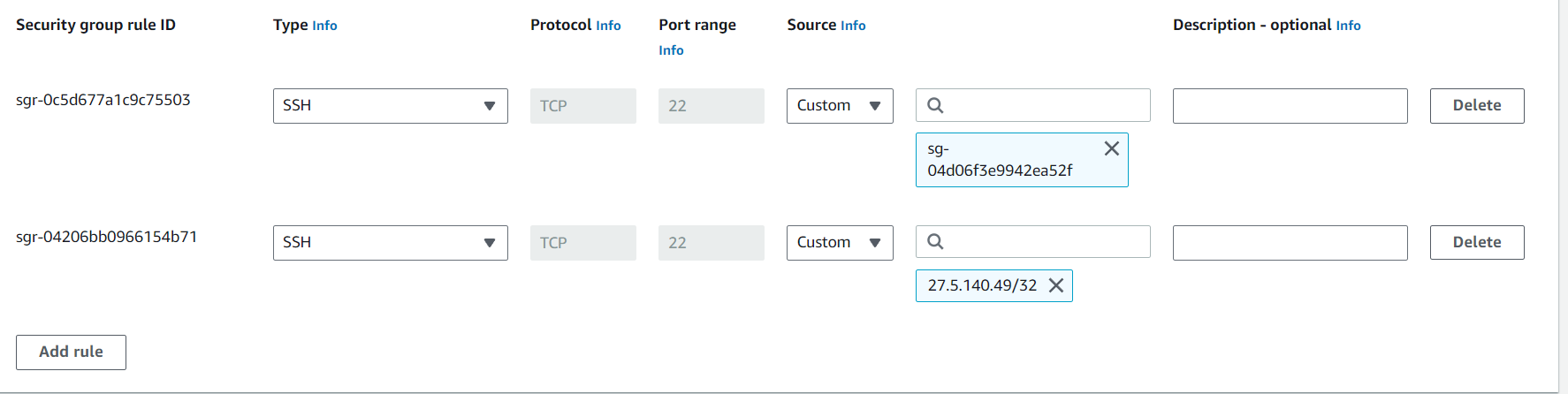
ansible-config init --disabled -t all > ansible.cfg > ansible.cfg

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:

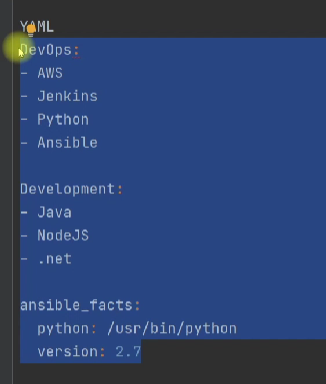
ansible\_user: ec2-user

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"

}

data = {

"name": "John",

"age": 30,

"city": "New York"

}

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)

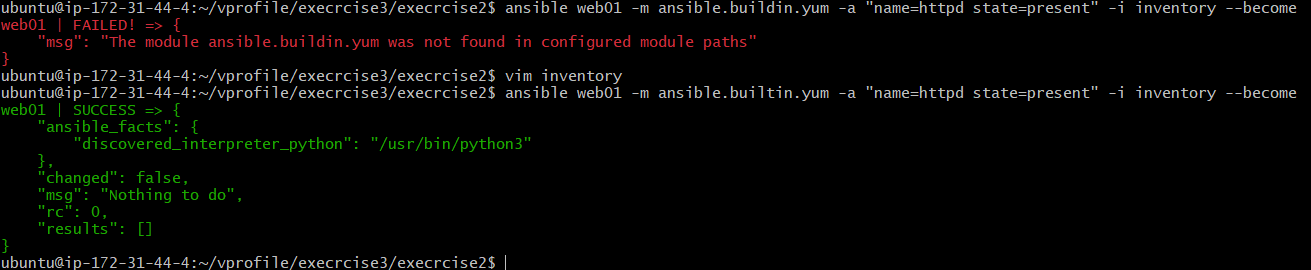
ad hoc commands are great for tasks you repeat rarely. For example, if you want to power off all the machines in your lab for Christmas vacation, you could execute a quick one-liner in Ansible without writing a playbook. An ad hoc command looks like this:

$ 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 # (--become is give sudo)

ansible web01 -m ansible.builtin.copy -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

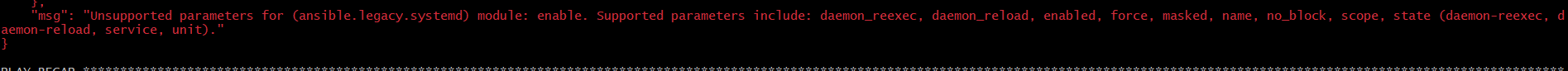
mysql\_user:

name: accounts

state: present

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

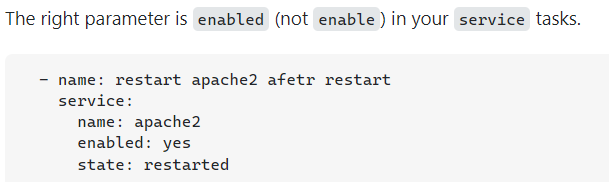
## Errors:



"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)."

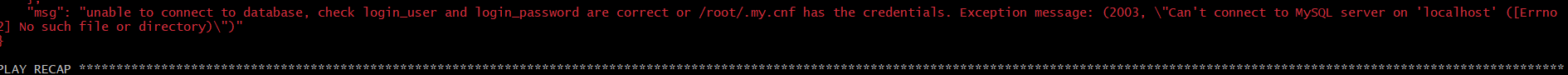
}

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)\")"

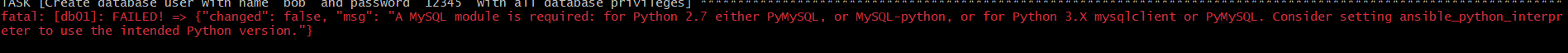
}



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."}

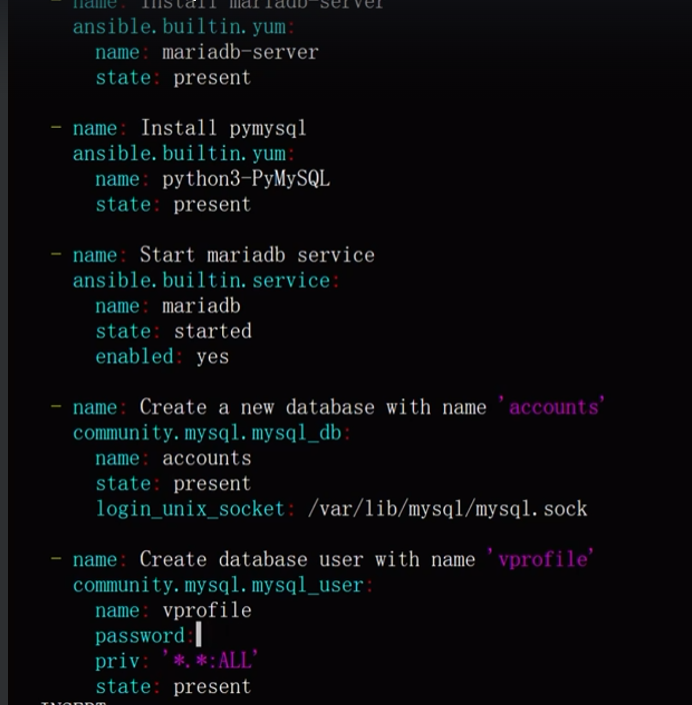


Install the dependencies

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) 🡪 repository specific for the project
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

## I Create a config file

[defaults]

host\_key\_checking = False

inventory = ./inventory

forks = 5

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

[privilege\_escalation]

become=True

become\_method=sudo

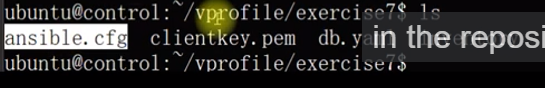
#become\_user=root

become\_ask\_pass=False

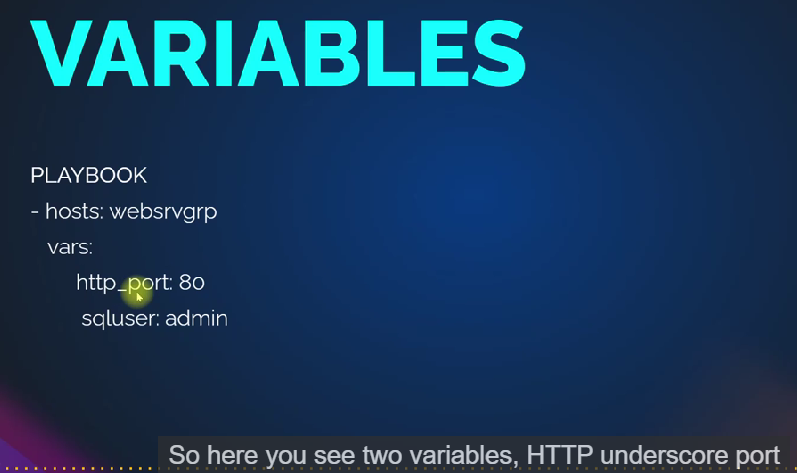
Change the permission in it ask any



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.

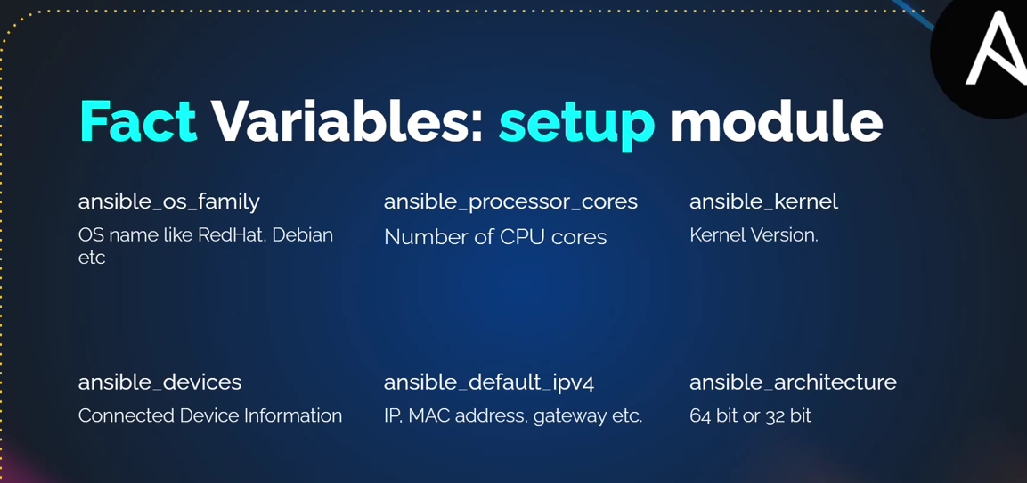
Now, you don't need to run setup module.When you execute the playbook,

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 #--> host = web01 , web02 , db01

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

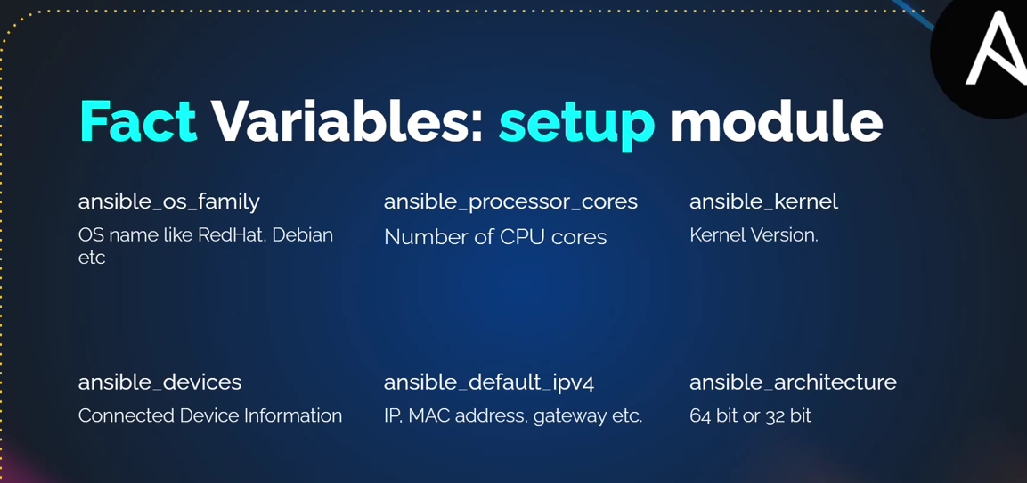
Group\_vars/all 🡪 next priority

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

Ansible -m setup web01 -i inventory(if you didn’t create config file(ansible.cfg)

You will get this error

[WARNING]: provided hosts list is empty, only localhost is available. Note that the implicit localhost does not match 'all'

[WARNING]: Could not match supplied host pattern, ignoring: dbservers

We can see ansible fact with dict, key & value , list etc

Vim facts.yaml

---

- name : print facts

hosts: all

tasks:

- name: print os name

debug:

var: ansible\_distribution

---

- name : print facts

hosts: all

# gather\_facts: false

tasks:

- name: print os name

debug:

var: ansible\_distribution

🡪create a new ec2 instance(ubuntu)🡪 name(web01)

But now ansible all -m ping -I inventory 🡪 will failed due to because we have centos instances we declare the ansible\_user in group wise

Now declare ansible\_user: ubuntu in hosts(which as high priority compare to groups)

all:

hosts:

web01:

ansible\_host: 172.31.42.193

web02:

ansible\_host: 172.31.37.109

web03:

ansible\_host: 172.31.37.36

ansible\_user: ubuntu

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

## we have use some fact variable

we can use ansible web01 -m setup -i inventory 🡪 to know fact variables

---

- name : print facts

hosts: all

#gather\_facts: false

tasks:

- name: print os name

debug:

var: ansible\_distribution

- name: print selinux mode

debug:

var: ansible\_selinux.mode

- name: print RAM memory

debug:

var: ansible\_memory\_mb.real.free

~

# Decision making

Provisioning server

1. NTP service on multi OS
2. User & Groups
3. Configuration files
4. Decision Making
5. Loops
6. Templates
7. Handles
8. Ansible Roles

🡪For provisioning ntp server

---

- name: Provisioning Server

  hosts: all

  become: yes

  tasks:

    - name: Install NTP on Centos

      yum:

        name: "{{item}}"

        state: present

      when: ansible\_distribution == "CentOS"

      loop:

        - ntp

        - wget

        - git

        - zip

        - unzip

    - name: Install NTP on Ubuntu

      apt:

         name: "{{item}}"

         state: present

         update\_cache: yes

      when: ansible\_distribution == "Ubuntu"

      loop:

        - ntp

        - wget

        - git

        - zip

        - unzip

    - name: Start & enable NTP on Redhat

      service:

        name: ntpd

        state: started

        enabled: yes

      when: ansible\_os\_family == "RedHat"

    - name: Start & enable NTP on Debian

      service:

        name: ntp

        state: started

        enabled: yes

      when: ansible\_os\_family == "Debian"

    - name: Add group

      group:

         name: devops

         state: present

    - name: Add users

      user:

          name: "{{item}}"

          state: present

          groups: devops

      loop: "{{usernames}}"

for decision making 🡪 in ansible will when if we need execute only centos machine (then we need to point centos only)

tasks:

    - name: Install NTP on Centos

      yum:

        name: chrony

        state: present

      when: ansible\_distribution == "CentOS"

    - name: Install NTP on Ubuntu

      apt:

         name: ntp

         state: present

         update\_cache: yes

      when: ansible\_distribution == "Ubuntu"

update\_cache : yes 🡪 first it going run apt update then install

to search ansible condition document:

<https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_conditionals.html>

# LOOPS

Example:

We can install multi tasks loop with name variable “{{item}}” (item is the standard team when add loop ansible try to find the item in insert the each package

tasks:

    - name: Install NTP on Centos

      yum:

        name: "{{item}}"

        state: present

      when: ansible\_distribution == "CentOS"

      loop:

        - ntp

        - wget

        - git

        - zip

        - unzip

    - name: Install NTP on Ubuntu

      apt:

         name: "{{item}}"

         state: present

         update\_cache: yes

      when: ansible\_distribution == "Ubuntu"

      loop:

        - ntp

        - wget

        - git

        - zip

        - unzip

In search 🡪 ansible loop document

<https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_loops.html>

he explain a lot in document try to read it

# File, copy & template modules

<https://docs.ansible.com/ansible/2.8/modules/list_of_files_modules.html>

you need to work is inside doc and how to use it

for example:

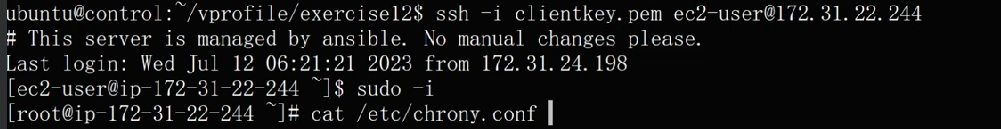
if you this file copy module

**-** name**:** Banner file

copy**:**

content**:** '#Thisserver is managed by Ansible. nomanual changes please'

dest**:** /etc/motd



## Templates

Login to the web01 (centos machine) 🡪 cat /etc/chrony.cong (copy all the content)

Make a folder in control machine

mkdir Templates

vim Templates/ntpconf\_centos

* Paste the all the content from /etc/chorny.conf

Similar

Login to the web01 (ubuntu machine) 🡪 cat /etc/ntp.cong (copy all the content)

Make a folder in control machine

mkdir Templates

vim Templates/ntpconf\_ubuntu

* Paste the all the content from /etc/ntp.conf

Now add the template configuration in play-book

**-** name**:** deploy ntp conf in centos

Template**:**

**src: Templates/ntpconf\_centos**

**dest: /etc/chrony.conf**

**backup: yes**

**when:ansible\_distribution == “CentOS”**

**-** name**:** deploy ntp conf in ubuntu

template**:**

**src: Templates/nptconf\_ubuntu**

**dest: /etc/ntp.conf**

**when: ansible\_distribution == “Ubuntu”**

**RESTART THE CHORNY & NTP SERVICES**

**-** name**:** restart the service on centos

service**:**

**name: chronyd**

**state: restarted**

**enable: yes**

**when: ansible\_distribution == “CentOS”**

**-** name**:** restart the service on Ubuntu

service**:**

**name: ntpd**

**state: restarted**

**enable: yes**

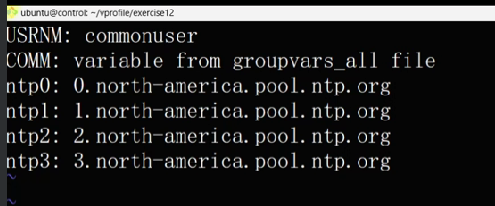
**when: ansible\_distribution == “Ubuntu”**

## why templates instead of copy modules?

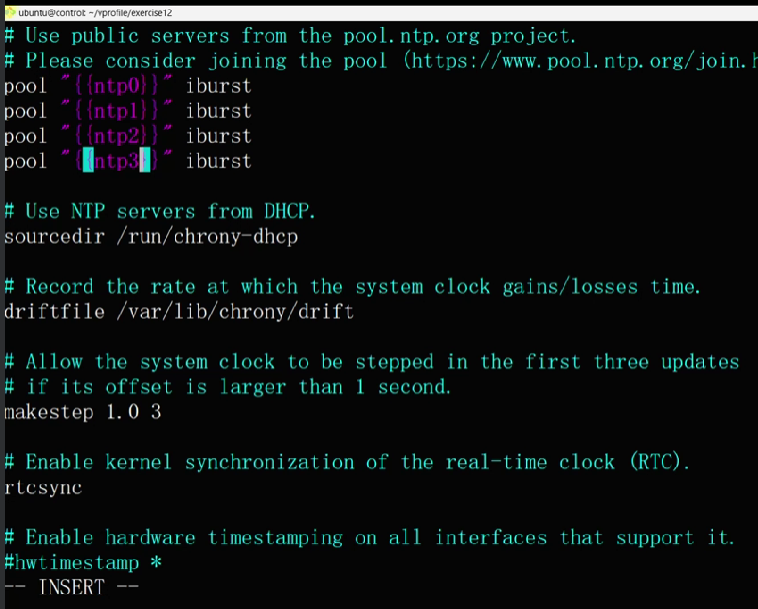
Copy modules just dump the content in server but template is intelligent, it read the template if any genja template is present. ( variable, condition is called a genja tool template)

Vim group\_vars/all

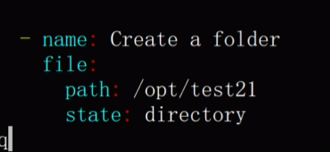
We can any ntp server in any location



Also Change in the templates/ntconf\_centos



Add one task to create a folder



**But when you run the play-book , it will create folder and also it will restart the service which is not good in production environment we can use handler/when condition to overcome it**

# Handlers

Seem like task and handler are same, but handler will not be executed util they are notified.

- name: Deploy NTP conf file for RedHat

template:

src: templates/ntp\_redhat.conf.j2

dest: /etc/ntp.conf

when: ansible\_os\_family == "RedHat"

notify:

- Restart NTP on Redhat

- name: Deploy NTP conf file for Debian

template:

src: templates/ntp\_debian.conf.j2

dest: /etc/ntp.conf

when: ansible\_os\_family == "Debian"

notify:

- Restart NTP on Debian

- name: Dir for dev data

file:

path: /opt/devdata

state: directory

mode: 0775

handlers:

- name: Restart NTP on Redhat

service:

name: ntpd

state: restarted

enabled: yes

when: ansible\_os\_family == "RedHat"

- name: Restart NTP on Debian

service:

name: ntp

state: restarted

enabled: yes

when: ansible\_os\_family == "Debian"

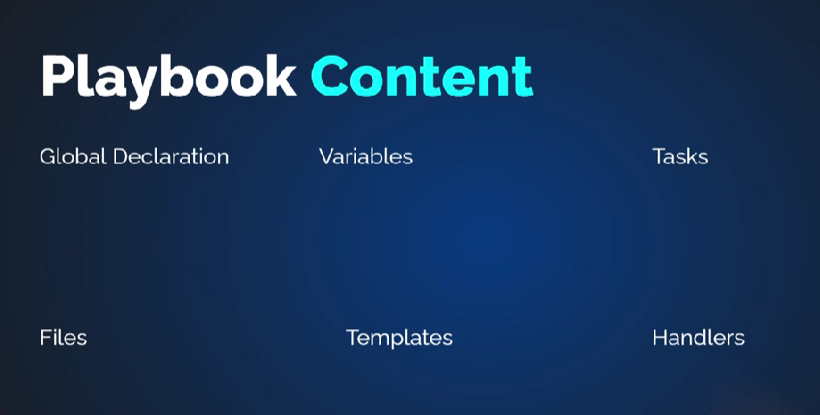
note: handle is for many task 🡪 mostly handler in dormant state 🡪 when notify is execute it will redirect to handler.

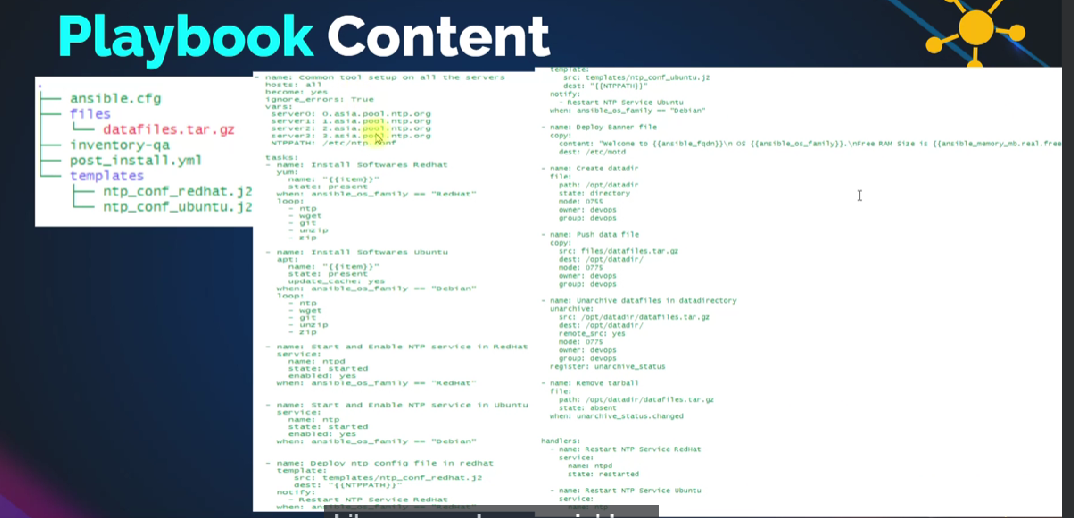
### Handlers Document link : <https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_handlers.html>

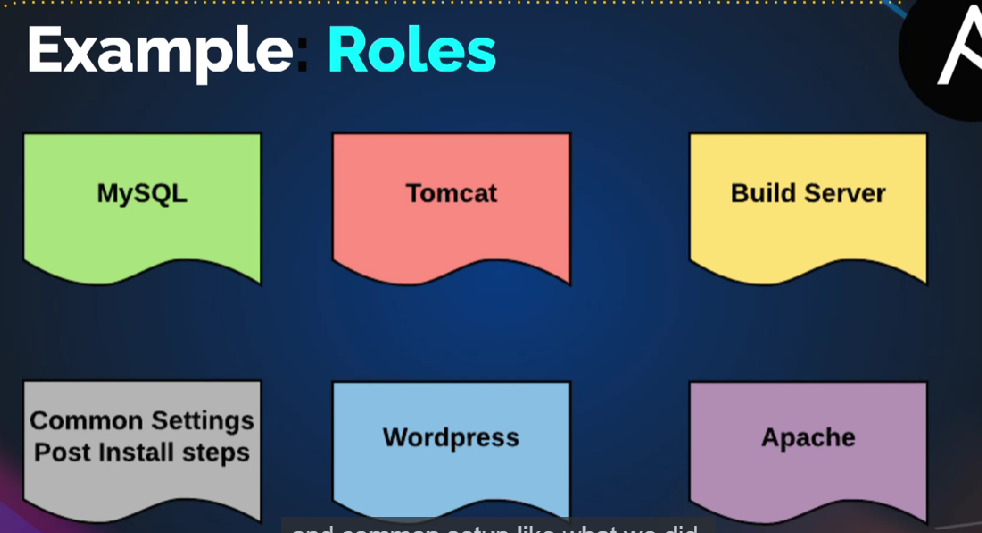
# Roles

Already we have lot of the thing in play-book/ ansible to manage this we can use roles(which really simplify and segregate all the content)

* It used for reusability



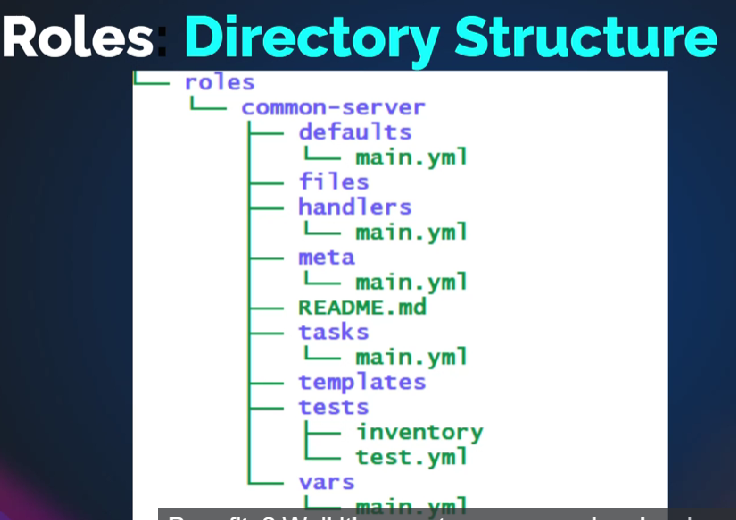




We may some server mysql , tomcat

Some use common role like ntp etc

First we need identify the role then we can distribute account it



Adding few more task in play-book: to make some complex

* Name: dump

Copy:

src: files/file.txt

Dest: /etc/myfile.txt

Or

* Name: dump

Copy:

src: “{{mydir}}”

Dest: /etc/myfile.txt

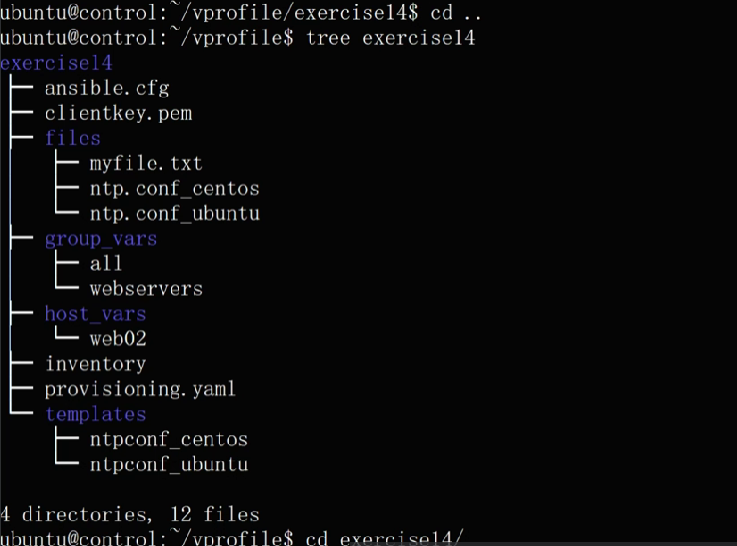
Declare the variable in outside of the play-book or top of the playbook (global variable)

Vars:

mydir: files/file.txt

run the play-book(ansible-playbook (playbook name) )

🡪 apt install tree



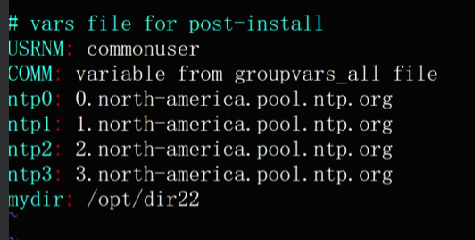
mkdir roles

cd roles

ansible-galaxy init post-install

Vim roles/pst-install/vars/main.yml

Past the vars and ntp



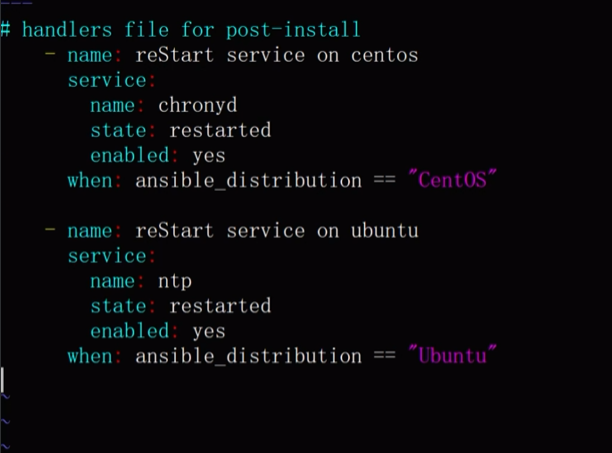
And remove group\_vars and host\_vars

rm -rf group\_vars host\_vars

ls

cp files/\* roles/post-install/files

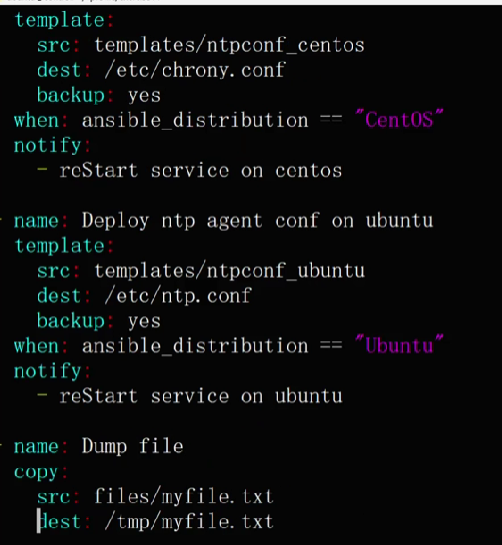
vim roles/post-install/handlers/main.yml



Remove space in %s/~ //

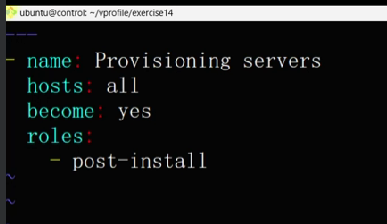
This is important 🡪 it execute first them handler etc will called

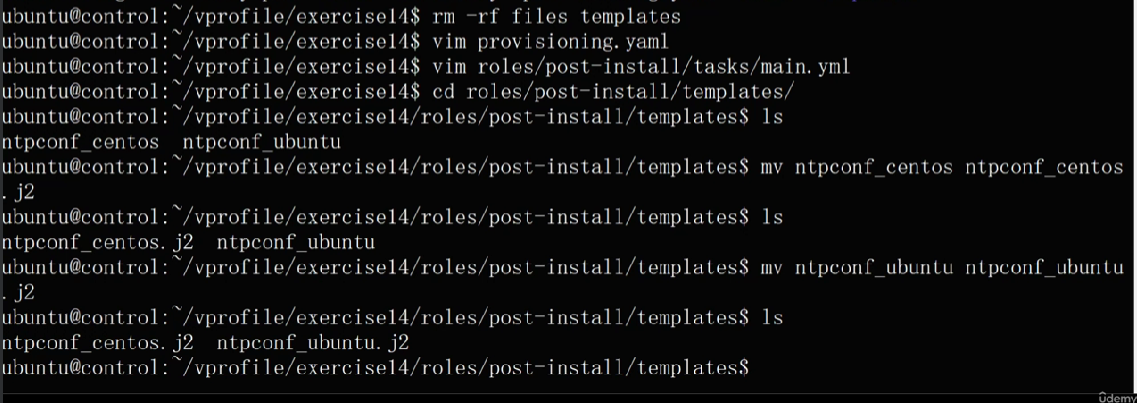
vim roles/post-install/tasks/main.yml

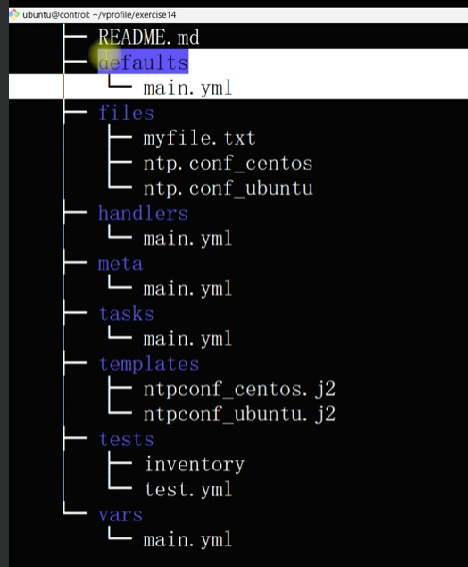


Totally 63 lines search in exercise

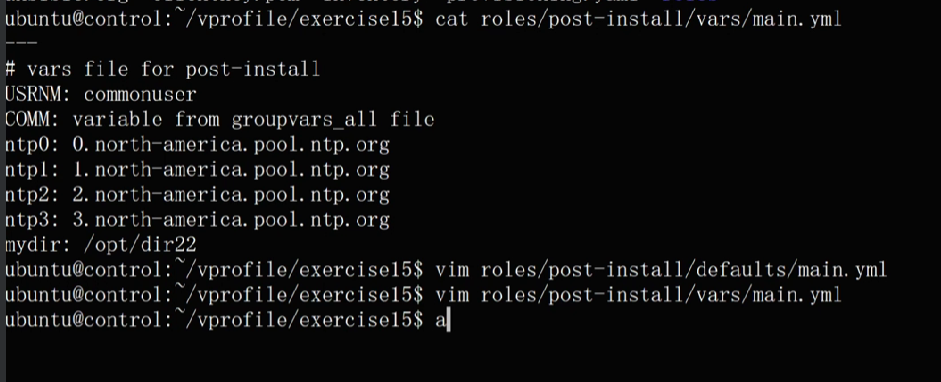
Vim play-book





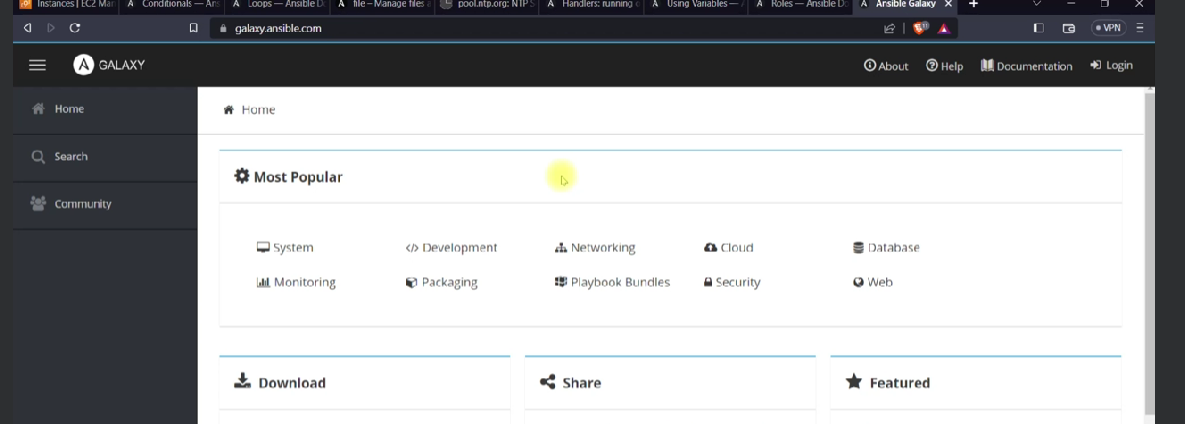


We can place variable in defaults also (it is low priority)



We can use play-book to override it like this







# Ansible for AWS

### Document

<https://docs.ansible.com/ansible/latest/collections/amazon/aws/index.html>

## perquisites:

# Authentication[ℑ](https://docs.ansible.com/ansible/latest/collections/amazon/aws/docsite/aws_ec2_guide.html#authentication)

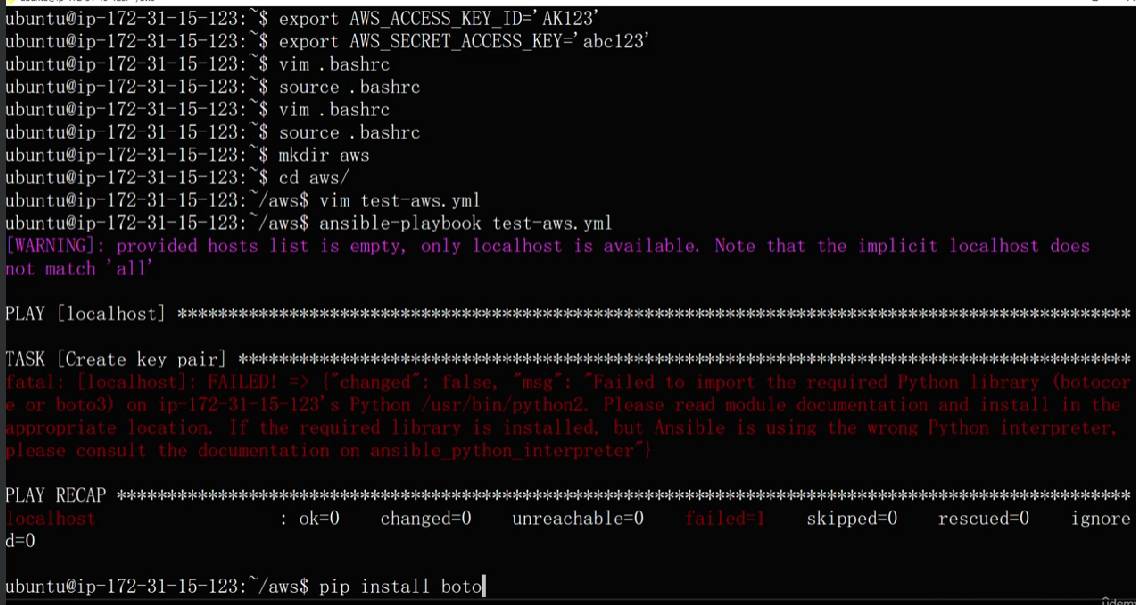
If your Ansible controller is not in AWS, authentication is handled by either specifying your access and secret key as ENV variables or inventory plugin arguments.

For environment variables:

export **AWS\_ACCESS\_KEY\_ID=**'AK123' 🡪 this are not real keys

export **AWS\_SECRET\_ACCESS\_KEY=**'abc123'

create IAM use with attached policy of administrator 🡪security🡪access key 🡪 download and put the original keys



First install pip

Sudo apt install python3-pip -y

Pip (press tab tab)

Pip pip3 pip3.10

Pip3.10 install boto3

## yaml file

---

- name: Setup keypair and ec2 instance

hosts: localhost

tasks:

- name: create a new ec2 key pair, returns generated private key

ec2\_key:

name: sample

region: us-east-2

register: keyout

when we create a key in iam user(it is key pair public and private key)🡪but public is available in iam user

we can store the key by using register : keyput followed by debug vars’s

* if key is already present (we can’t able to see the private key
* frist delete the public and run the yaml files so that we can see the private key in debug session

# - debug:

# var: keyout

- name: save private key content

copy:

content: "{{keyout.key.private\_key}}"

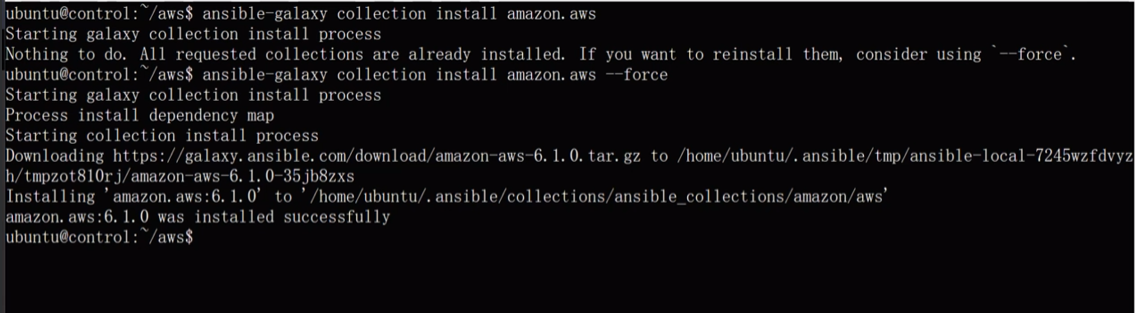
dest: ./sample-key.pem

when: keyout.changed

copy module will save the private key in ./sample-key.pem

install the ansible-galary collection for aws

ansible-galaxy collection install amazon.aws



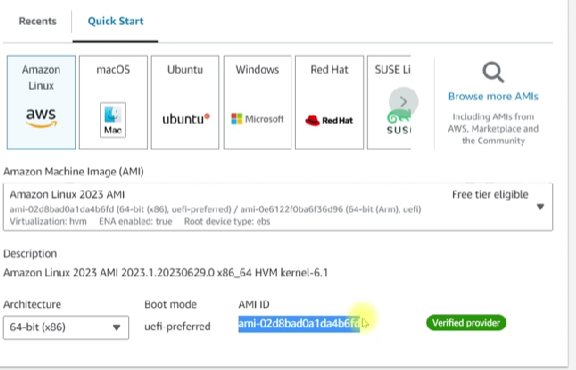
- name: test ec2 instance

ec2:

key\_name: sample

instance\_type: t2.micro

image: ami-016b213e65284e9c9 -🡪 this machine ami number



wait: yes

instance\_tags:

Name: vpro-web01

project: vprofile

exact\_count: 1 🡪 count should be 1 become if you not keep exact\_count the every time . you run playbook 1 instance will create

count\_tag:

Name: vpro-web01

project: vprofile

region: us-east-2

### Ansible for AWS Management: Lecture Summary

**Overview:** In this lecture, we learn how to use Ansible for managing AWS services by utilizing various AWS modules provided by Ansible. We will cover the necessary prerequisites, authentication setup, and creating a playbook for AWS tasks such as creating a key pair and launching EC2 instances.

**Key Concepts:**

1. **AWS Authentication:**
   * Ansible requires AWS credentials to manage AWS services.
   * Create an IAM user with necessary permissions (e.g., AdministratorAccess).
   * Generate access keys for the IAM user and export them as environment variables (AWS\_ACCESS\_KEY\_ID and AWS\_SECRET\_ACCESS\_KEY).
2. **Environment Setup:**
   * Export the AWS credentials using the export command or add them to your ~/.bashrc file to persist them across sessions.
   * Example:

bash

Copy code

export AWS\_ACCESS\_KEY\_ID=<your\_access\_key>

export AWS\_SECRET\_ACCESS\_KEY=<your\_secret\_key>

1. **Creating a Basic Playbook:**
   * Create a directory for your AWS playbook and create a YAML file (e.g., test-AWS.yml).
   * Define the playbook structure with hosts, gather\_facts, and tasks.
2. **Tasks in Playbook:**
   * **Creating a Key Pair:**
     + Use the ec2\_key module to create a key pair.
     + Save the private key using the copy module.
     + Handle the case where the key pair already exists by checking if the task changed and registering the output.
   * **Launching an EC2 Instance:**
     + Use the ec2\_instance module to launch an EC2 instance.
     + Ensure correct parameters such as image\_id, instance\_type, key\_name, and region are provided.
     + Optionally, wait for the instance state to become running using the wait parameter.

**Example Playbook:** Here’s a condensed example of what the playbook might look like:

yaml

Copy code

---

- name: Manage AWS EC2

hosts: localhost

gather\_facts: false

tasks:

- name: Create key pair

ec2\_key:

name: sample

region: us-west-2

register: key\_out

- name: Save private key

copy:

content: "{{ key\_out.key.private\_key }}"

dest: ./sample.pem

when: key\_out.changed

- name: Launch EC2 instance

ec2\_instance:

name: my-ec2-instance

key\_name: sample

instance\_type: t2.micro

image\_id: ami-0abcdef1234567890

region: us-west-2

wait: true

exact\_count: 1

**Common Issues and Tips:**

* Ensure boto3 and other required Python libraries are installed (pip install boto3).
* Double-check AWS region, AMI IDs, and other resource identifiers for accuracy.
* Utilize Ansible documentation for detailed module usage and parameters.

**Conclusion:** This lecture provided a practical guide to managing AWS resources using Ansible, from setting up authentication to creating key pairs and launching EC2 instances. By understanding and using the relevant Ansible modules, you can automate and manage your AWS infrastructure effectively.