HANDS ON ANSIBLE

INTORDUCTION

Wikipedia

***Ansible** is software that automates software provisioning, configuration management, and application deployment*

WHAT IS ANSIBLE

- ***CHANGE MANAGEMENT***
- ***PROVISIONING***
- ***AUTOMATION***
- ***ORCHESTRATION***

CHANGE MANAGEMENT

Define a "System State"

Doing any kind of configuration changes or system changes.

It can be vary from doing small stuff like editing ssh config file or a big task as installing or configuring a web server.

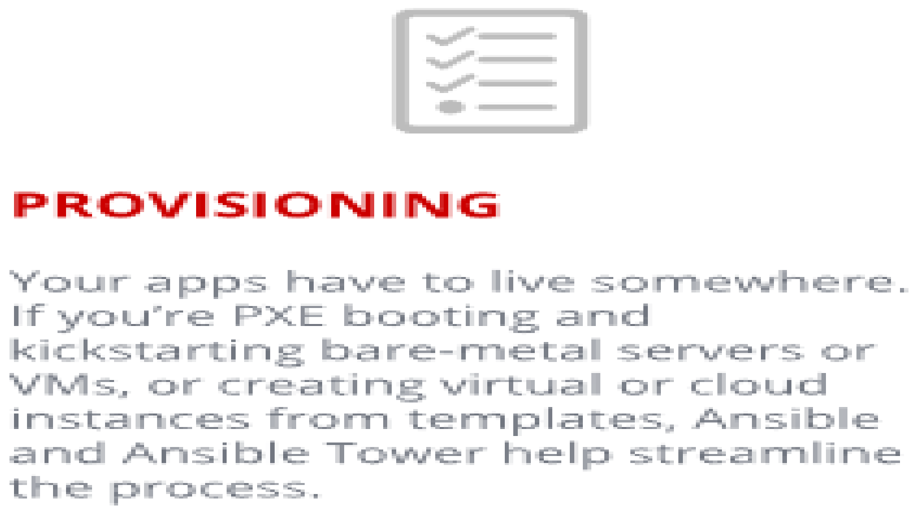
PROVISIONING

Prepare a system to make it ready

Building servers in any kind of environment , it can vary from your bare metal physical box to virtual machines or any kind of cloud environment on demand in very less time.

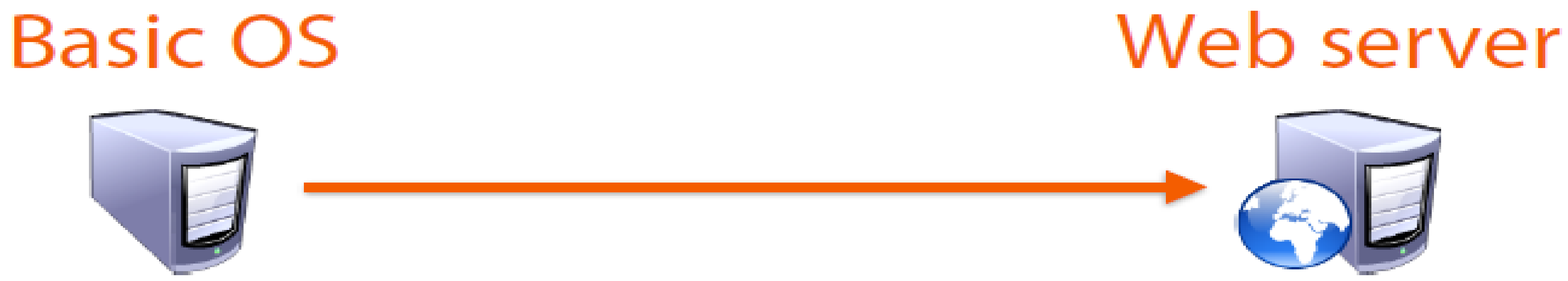
Examples

- *Make an FTP Server*
- *Make an Email Server*
- *Make a DB Server*



PROVISIONING

Your apps have to live somewhere. If you're PXE booting and kickstarting bare-metal servers or VMs, or creating virtual or cloud instances from templates, Ansible and Ansible Tower help streamline the process.



1. Install web software
2. Copy configurations
3. Copy web files
4. Install security updates
5. Start web service

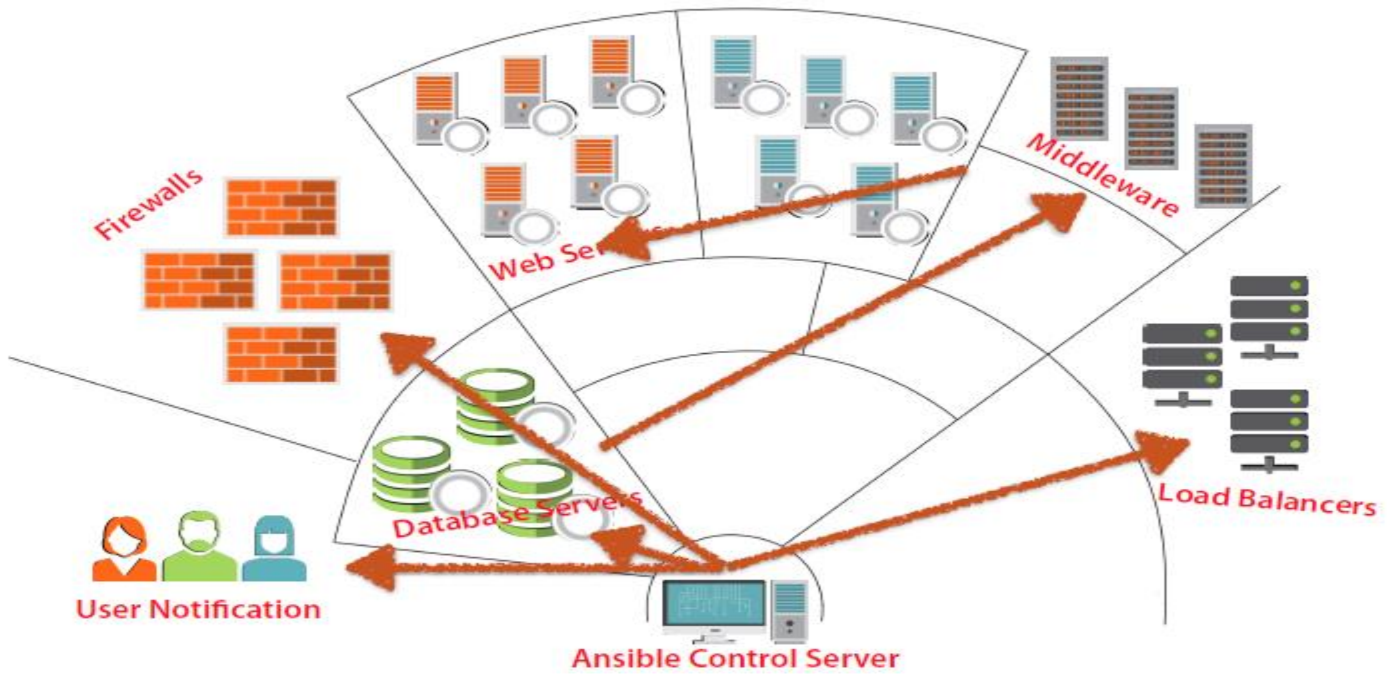
AUTOMATION

Define tasks to be executed automatically

A set of ordered task to be executed in orderly fashion which include making decision and running ad-hoc tasks

ORCHESTRATION

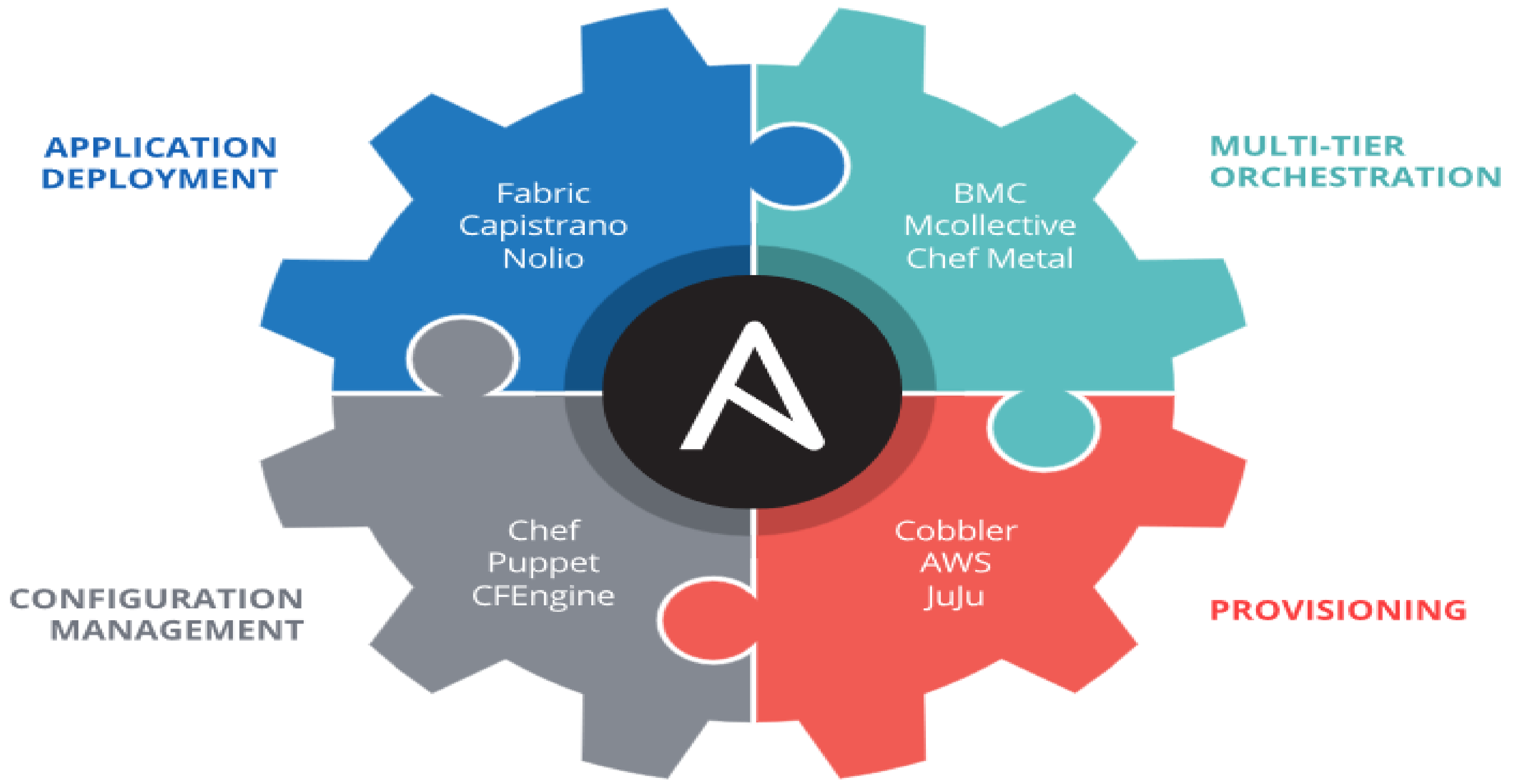
Coordinates automation BETWEEN systems



ORCHESTRATION

Configurations alone don't define your environment. You need to define how multiple configurations interact and ensure the disparate pieces can be managed as a whole. Out of complexity and chaos, Ansible brings order.

Why Ansible?


APPLICATION DEPLOYMENT
Fabric
Capistrano
Nolio
MULTI-TIER ORCHESTRATION
BMC
Mcollective
Chef Metal
CONFIGURATION MANAGEMENT
Chef
Puppet
CFEngine
PROVISIONING
Cobbler
AWS
JuJu

In market we have many automation tools as mentioned below.

For configuration management (Puppet, Chef, cfengine)

Server deployment (Capistrano, Fabric)

Ad-hoc task execution (Func,plain SSH)

With ***ANSIBLE*** *you will be getting all the above functionality*

What makes it so different?

- **It’s clean!**

-> No agents

-> No database

-> No residual software

-> No complex upgrades

YAML

- **Ansible Execution**

-> No programming required

-> Not a markup language

-> Easy to read and write

-> Human readable automation

Built-in security

-> Uses SSH

-> Root / Sudo usage

-> Encrypted vault

Easy to extend

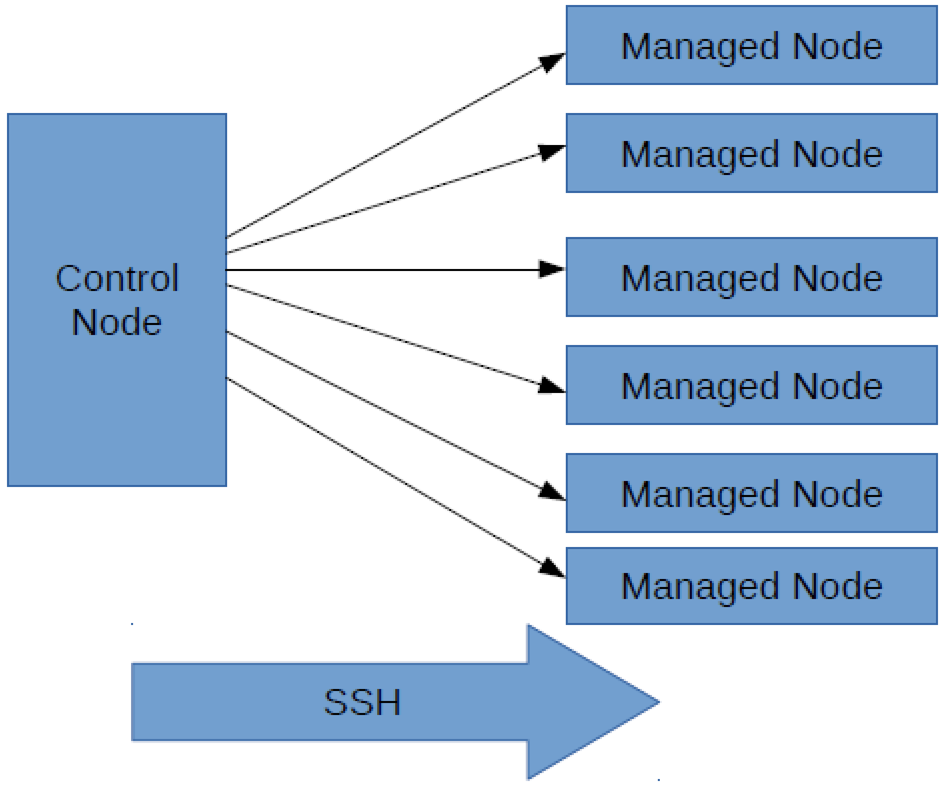
-> Shell Commands

-> Scripts

-> Ansible-Galaxy

Architecture and Process Flow

Ansible Architecture



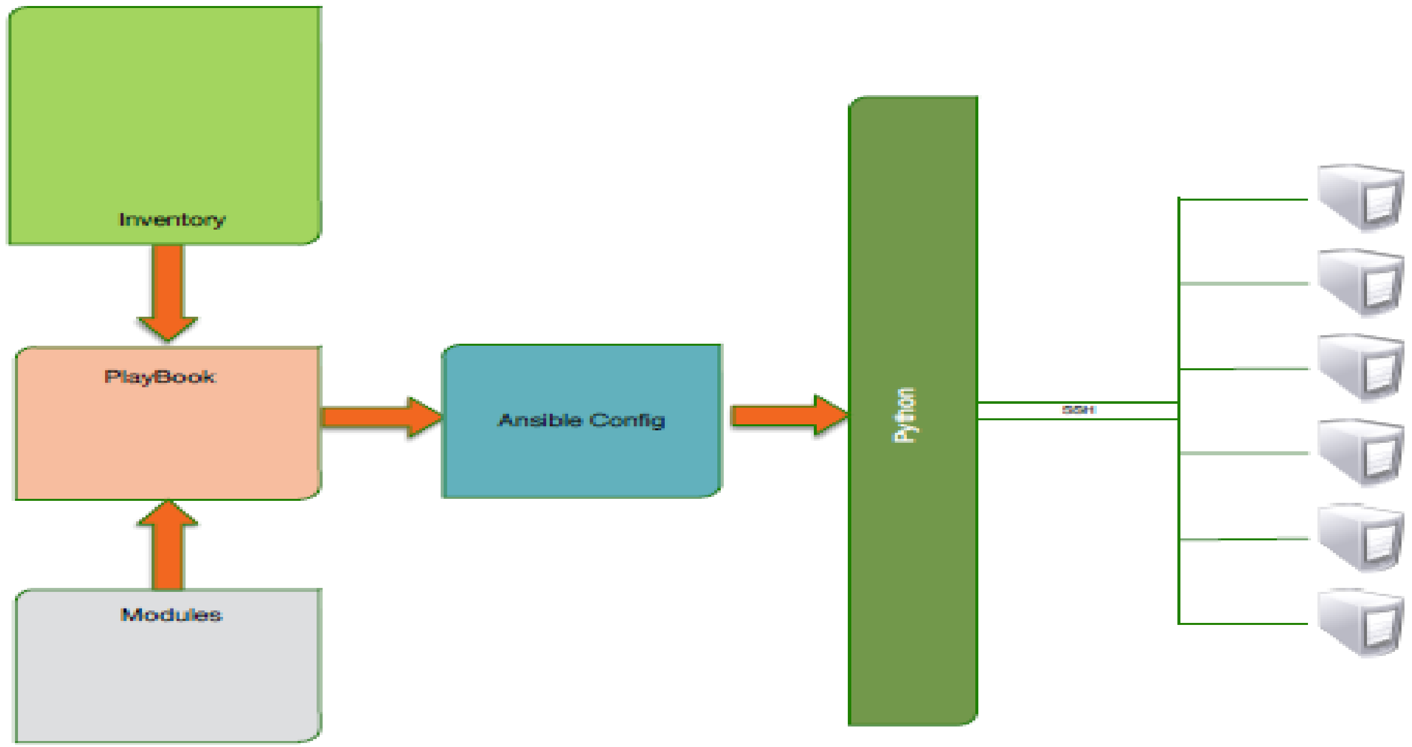
Requirements for Control Node

- *Ansible software is installed on the control node.*
- *A machine acting as a control node must have Python 2.6+.*
- *Red Hat Enterprise Linux 6 or 7 will run Ansible software.*
- *Windows is not supported for the control node at this time*.

Requirements on Managed Node

- *SSH must be installed and configured to allow incoming connections.*
- *Managed hosts must have Python 2.4 or later installed.*
- *The python-simple json package must also be installed on Red Hat Enterprise Linux 5 managed hosts. It is not required on Red Hat Enterprise Linux 6 and 7 managed hosts, since Python 2.5 (and newer versions) provide its functionality by default.*

Process Flow



1. Inventory

Its the list the of "Managed Hosts" and their connection configuration.

2. Modules

A programmed unit of work to be done.

***Core Modules**: There are around 500+ core modules that comes bundled with Ansible*

***Custom Modules**: Users can extend Ansible's functionality by writing custom modules, which are written in Python typically, but can also be written in Ruby, Python, shell etc.*

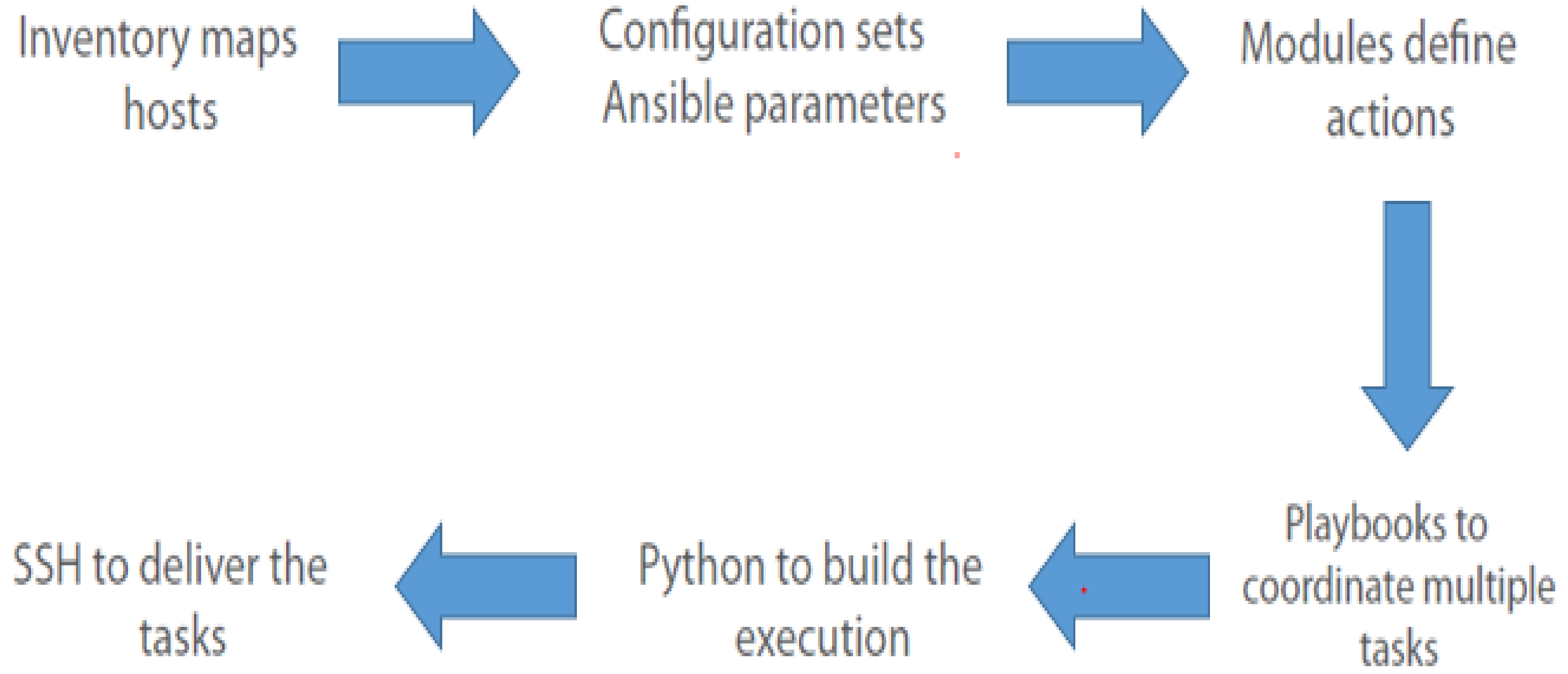
3. Playbooks

Ansible playbooks are files written in YAML syntax that define the modules, with arguments, to apply to managed nodes. They declare the tasks that need to be performed.

4. Configuration

Ansible have its own config which manages the behavior of ansible

***Default file**: /etc/ansible/ansible.cfg*


Inventory maps hosts
Configuration sets Ansible parameters
Modules define actions
Playbooks to coordinate multiple tasks
Python to build the execution
SSH to deliver the tasks

Ansible Connection Plugins

- ***Control Persist:*** *A feature that improves Ansible performance by eliminating SSH connection overhead, when multiple SSH commands are executed in succession.*
- *RHEL7 uses default ssh which have "ControlPersist" feature.*
- ***Paramiko:*** *Is a python implementation of ssh with ControlPersist feature, can be used on RHEL6.(python-paramiko.noarch)*
- ***Local:*** *This plugin is used to connect and manage "control node" itself.*
- ***winrm:*** *Connection plugin used for managing windows machines.*
- ***Docker***: *Ansible 2, we have docker plugin as well, which can connect from Docker host to containers.*

OUR LAB ENVIRONMENT

ANSIBLE CONTROL SERVER	**xpwinno1vscor.xavient.com**	**10.5.2.235**
MANAGED NODE1	**node1.xavient.com**	**10.5.2.233**
MANAGED NODE2	**node2.xavient.com**	**10.5.2.234**

How To Access Our Lab Environment

User ID- xavient (You can switch to root from this account)

Password- 1@34567b

*For Demo we will be using user **ansible** to perform all our task.*

Please create your userids with password less authentication for practice.

Installing Ansible

- *Ansible is required to be installed only on control node, unlike Puppet or Chef.*
- *Python3 is not used by ansible as of now.*
- *Ansible is not a part of RHEL repo (as of now). Try GitHub or EPEL Repos.*
- *Officially we can get ansible from www.ansible.com*

Install Ansible (Debian)

```
$ sudo apt-get install ansible
```

Install Ansible (CentOS)

```
$ sudo yum install epel-release
$ sudo yum install ansible
```

Ansible Inventory and Configuration

Inventory Features

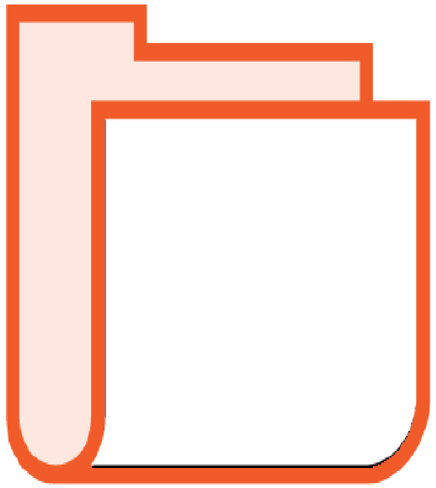
- Behavioral Parameters
- Groups
- Groups of Groups
- Assign Variables
- Scaling out using multiple files
- Static/Dynamic

Inventory File

```
[web]
Node1.example.com ansible_ssh_user=ansible ansible_ssh_pass=233
Node2.exmple.com ansible_python_interpreter=/usr/bin/python
[cluster:children]
web
[cluster:vars]
ansible_ssh_user=ansible
ansible_ssh_pass=#######
ntp-server=26.6.6.5
```

Scaling-out Inventory Files

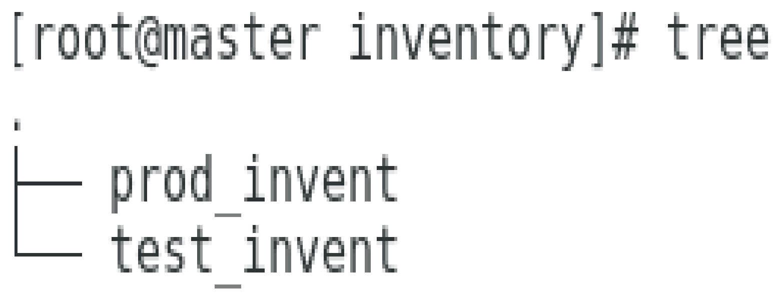
We can have multiple inventories files in directory



Using Directories

Can use to break-out long-running inventory files.

Very useful when dealing with large environments.



```
[root@master inventory]# tree
.
├── prod_invent
└── test_invent
```

Static/Dynamic Inventories

- ***Static Inventory:*** defined in an INI-like text file, in which

  each section defines one group of hosts (a host group)
- ***Dynamic Inventory:*** dynamically generated from various

  sources. public/private cloud providers, an LDAP database or CMDB.
- ***/etc/ansible/hosts*** is the default inventory

Configuration Settings Order-of-Operations

$ANSIBLE_CONFIG

$PWD/ansible.cfg

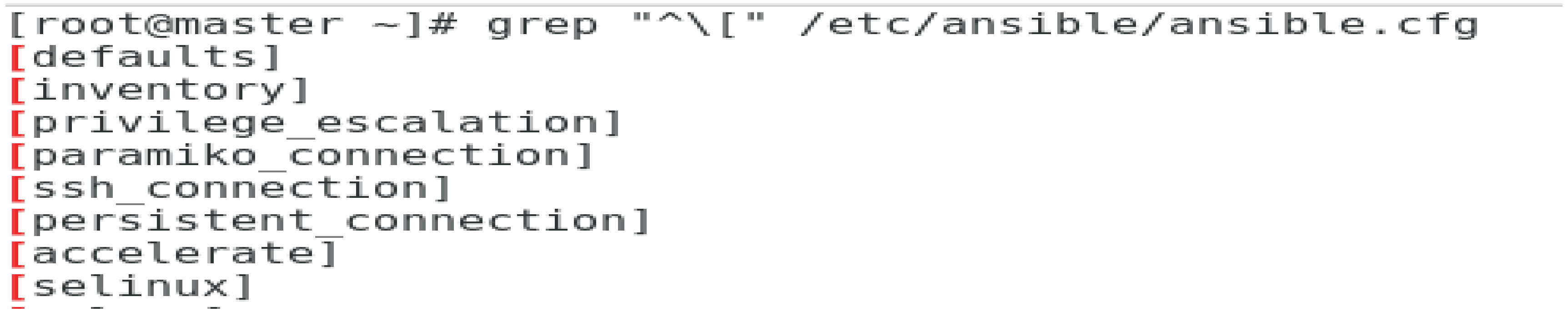
$HOME/.ansible.cfg

/etc/ansible/ansible.cfg

Configuration files are not merged
First one wins!

Configuration File

/etc/ansible/ansible.cfg consist of several sections



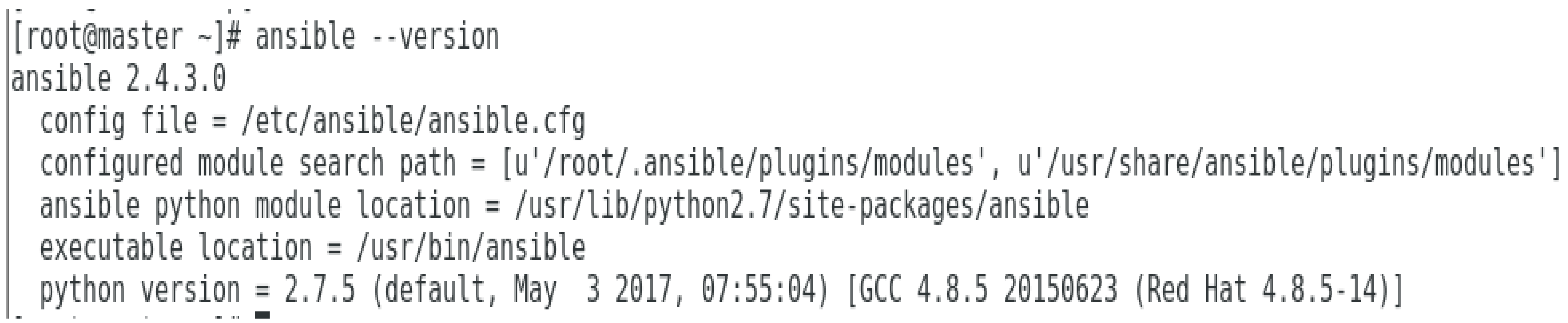
```
[root@master ~]# grep "^\[" /etc/ansible/ansible.cfg
[defaults]
[inventory]
[privilege_escalation]
[paramiko_connection]
[ssh_connection]
[persistent_connection]
[accelerate]
[selinux]
```

Before – Doing Actual Things !!

- *Before start doing actual things some setup has to be prepared.*
- *Remote User should be planned.*
- *SSH-Key based authentication (for password less auth)*
- *Either Privileges has to be given to the user or not.*
- *If yes, privilege escalation will be password less ?*

Running First Ansible Command

- $ ansible –version



```
[root@master ~]# ansible --version
ansible 2.4.3.0
  config file = /etc/ansible/ansible.cfg
  configured module search path = [u'/root/.ansible/plugins/modules', u'/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python2.7/site-packages/ansible
  executable location = /usr/bin/ansible
  python version = 2.7.5 (default, May  3 2017, 07:55:04) [GCC 4.8.5 20150623 (Red Hat 4.8.5-14)]
```

Running Ad-hoc Command

Intro

- *Ansible Ad-Hoc commands enable you to perform tasks on remote nodes without having to write a playbook. They are very useful when you simply need to do one or two things quickly and often, to many remote nodes*

EXAMPLES

```
ansible all -m ping
ansible web -m command -a "uptime“
ansible all -m setup
ansible web -m yum -a "name=httpd state=present" -b
```

Ansible command line options

Setting	Command-line option
inventory	-i
remote_user	-u
become	--become –b
become_method	--become-method
become_user	--become-user
become_ask_pass	--ask-become-pass -k

Ansible Playbooks

PLAY

A order set of task which should be run against hosts selected from inventory

PLAYBOOK

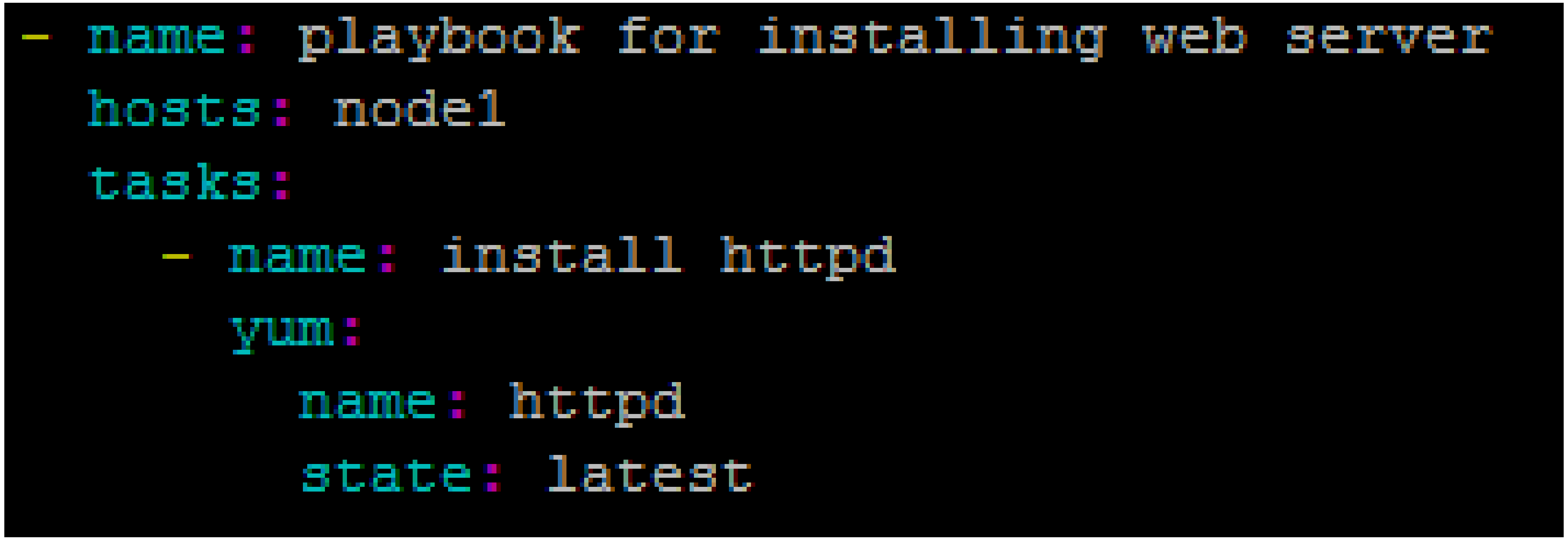
A playbook is a text file that contains a list one or more plays to run in order. It is saved with extension .yml

It primarily uses indentation with space characters to indicate the structure of its data.

Playbooks have three keys

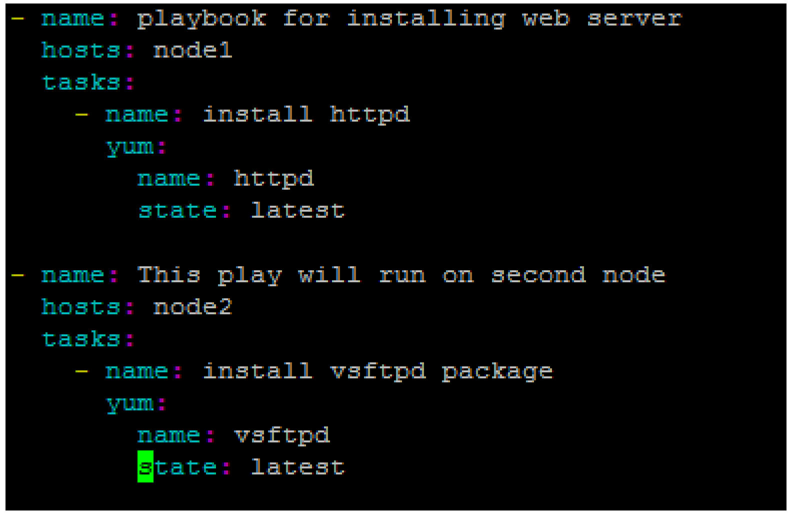
1. name: It is optional but recommended to document the playbook.
2. hosts: Servers on which tasks are executed
3. tasks: Actual work to perform on managed node

Format of Playbook



```
- name: playbook for installing web server
  hosts: node1
  tasks:
    - name: install httpd
      yum:
        name: httpd
        state: latest
```

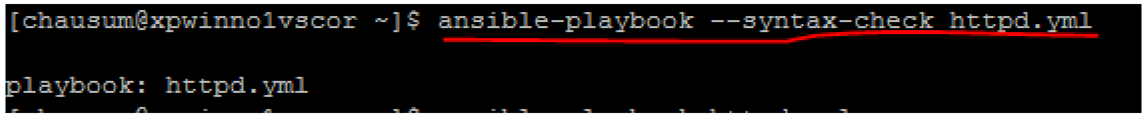
Playbook can have multiple plays



```
- name: playbook for installing web server
  hosts: node1
  tasks:
    - name: install httpd
      yum:
        name: httpd
        state: latest

- name: This play will run on second node
  hosts: node2
  tasks:
    - name: install vsftpd package
      yum:
        name: vsftpd
        state: latest
```

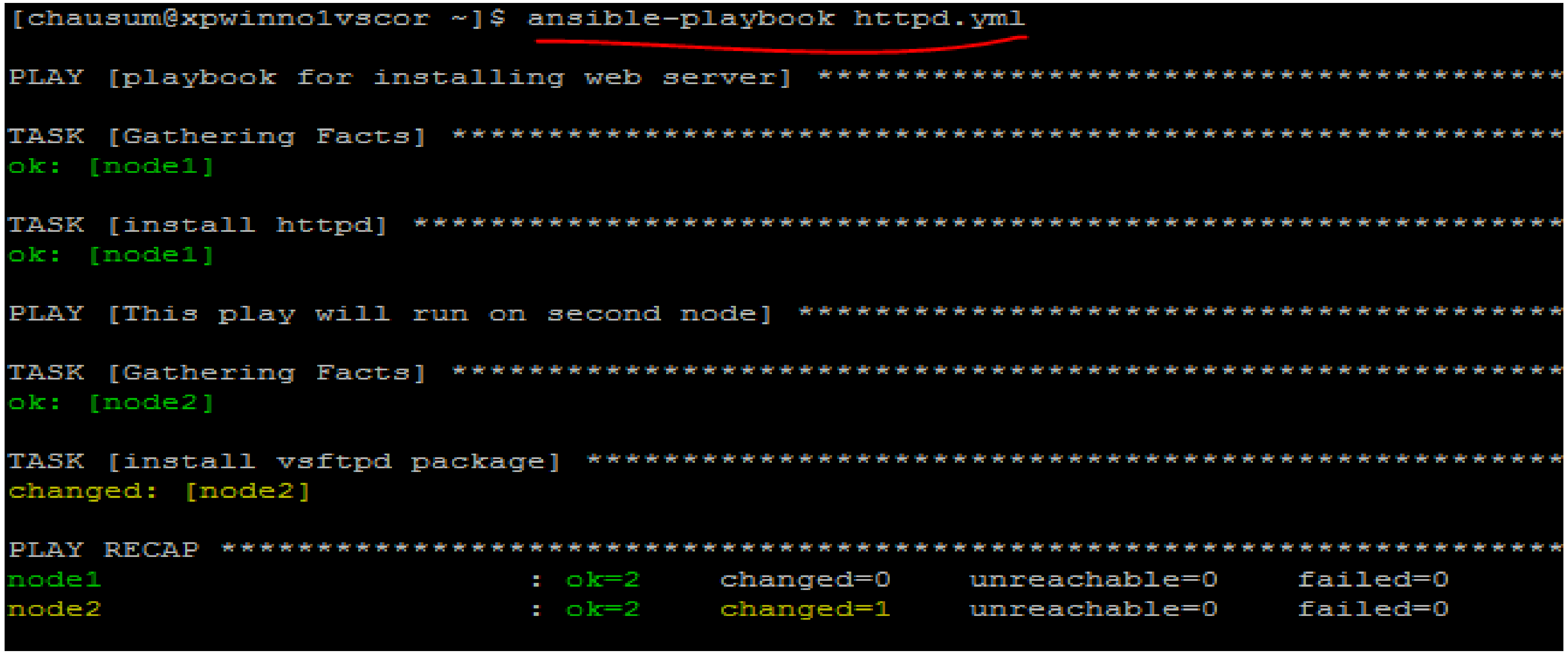
The order in which the plays are and tasks are listed in playbooks is important because ansible runs them in same order.

Syntax-check For Playbooks



```
[chausum@xpwinno1vscor ~]$ ansible-playbook --syntax-check httpd.yml

playbook: httpd.yml
```

Running Playbooks



```
[chausum@xpwinno1vscor ~]$ ansible-playbook httpd.yml

PLAY [playbook for installing web server] ****************************************

TASK [Gathering Facts] ***********************************************************
ok: [node1]

TASK [install httpd] *************************************************************
ok: [node1]

PLAY [This play will run on second node] *****************************************

TASK [Gathering Facts] ***********************************************************
ok: [node2]

TASK [install vsftpd package] ****************************************************
changed: [node2]

PLAY RECAP ***********************************************************************
node1                      : ok=2    changed=0    unreachable=0    failed=0
node2                      : ok=2    changed=1    unreachable=0    failed=0
```

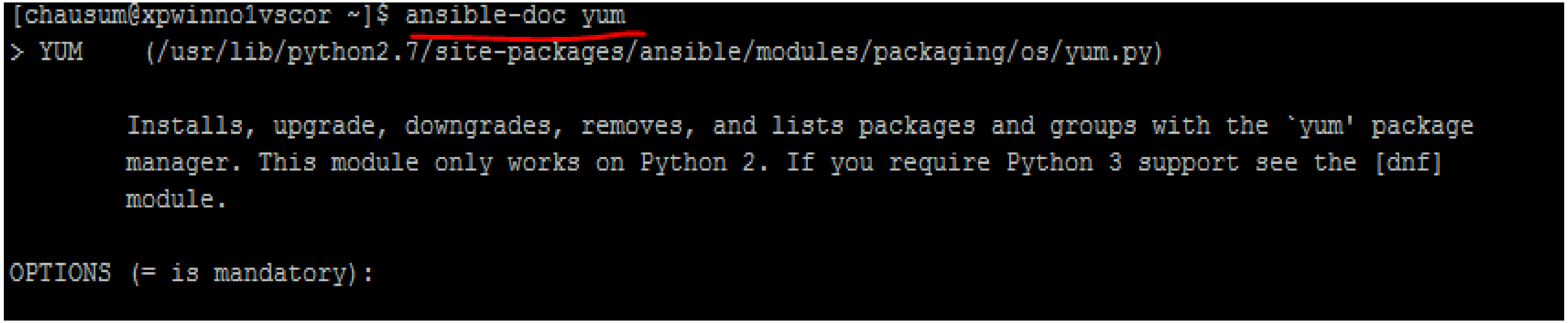
Modules For Tasks

- **To Check for the list of modules**

[chausum@xpwinno1vscor ~]$ ansible-doc -l | wc -l

1652

- **To find module info**



```
[chausum@xpwinno1vscor ~]$ ansible-doc yum
> YUM    (/usr/lib/python2.7/site-packages/ansible/modules/packaging/os/yum.py)

        Installs, upgrade, downgrades, removes, and lists packages and groups with the `yum' package
        manager. This module only works on Python 2. If you require Python 3 support see the [dnf]
        module.

OPTIONS (= is mandatory):
```

Simple Playbook Demo

1. *Use yum module to install httpd and firewalld*
2. *Ensure firewalld service is enabled and started*
3. *Ensure firewalld is configured to allow http connection*
4. *Ensure httpd service is started and enabled*
5. *Ensure managed hosts /var/www/html/index.html file consist of content "Welcome to Ansible World"*

MANAGING VARIABLES AND INCLUSIONS

Variables can be used to store value that can be used throughout files in entire ansible project

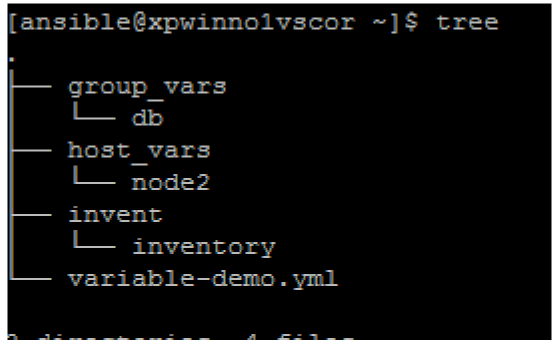
Defining Variables

Variables can be defined in variety of places in ansible configuration.

It can be simplified to three basic scope levels:

Global Scope	***Variable set from command line or in ansible configuration***
Play Scope	*Variable set in the play*
Host Scope	*Variable set on host group or individual hosts*

DEMO

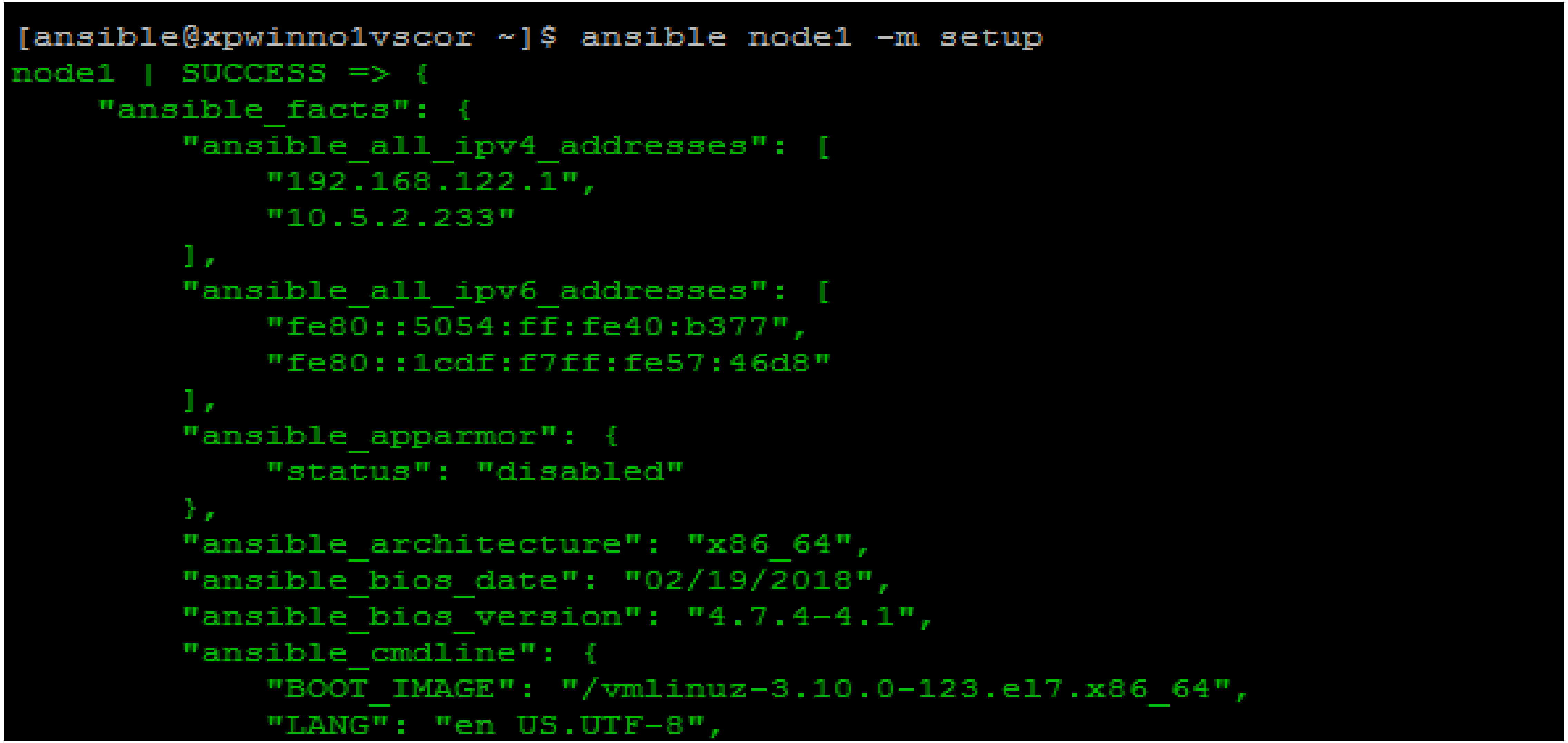


```
[ansible@xpwinno1vscor ~]$ tree
.
├── group_vars
│   └── db
├── host_vars
│   └── node2
├── invent
│   └── inventory
└── variable-demo.yml
```

Managing Facts

Ansible facts are variable that are automatically discovered by ansible on a managed hosts

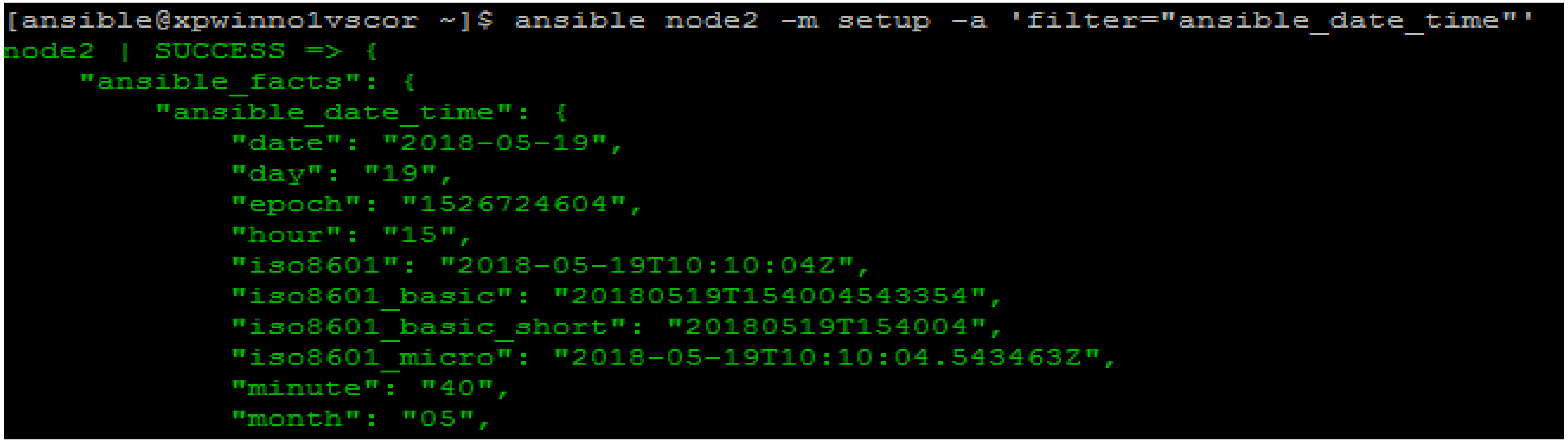
Facts are collected by using setup module



```
[ansible@xpwinno1vscor ~]$ ansible node1 -m setup
node1 | SUCCESS => {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "192.168.122.1",
            "10.5.2.233"
        ],
        "ansible_all_ipv6_addresses": [
            "fe80::5054:ff:fe40:b377",
            "fe80::1cdf:f7ff:fe57:46d8"
        ],
        "ansible_apparmor": {
            "status": "disabled"
        },
        "ansible_architecture": "x86_64",
        "ansible_bios_date": "02/19/2018",
        "ansible_bios_version": "4.7.4-4.1",
        "ansible_cmdline": {
            "BOOT_IMAGE": "/vmlinuz-3.10.0-123.el7.x86_64",
            "LANG": "en_US.UTF-8",
```

Fact Filters

You can use filter in order to limit the results when gathering facts from managed node



```
[ansible@xpwinno1vscor ~]$ ansible node2 -m setup -a 'filter="ansible_date_time"'
node2 | SUCCESS => {
    "ansible_facts": {
        "ansible_date_time": {
            "date": "2018-05-19",
            "day": "19",
            "epoch": "1526724604",
            "hour": "15",
            "iso8601": "2018-05-19T10:10:04Z",
            "iso8601_basic": "20180519T154004543354",
            "iso8601_basic_short": "20180519T154004",
            "iso8601_micro": "2018-05-19T10:10:04.543463Z",
            "minute": "40",
            "month": "05",
```

Custom Facts

You can create custom facts which are stored locally on each managed node2.

By defaults setup loads custom facts from files and scripts on managed hosts /etc/ansible/facts.d dir. The name of file should be end with .fact in order to be used.

Custom facts are stored by setup in ansible_local variable

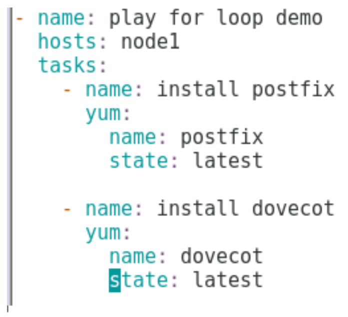
IMPLEMENTING TASK CONTROL

SIMPLE LOOPS

- *Loops iterates a task over a list of items.*
- *The **with_items** key is added to the task and takes as a value the list of items over which the task should be iterated.*
- *The loop variable **item** holds the current value being used for this iteration*

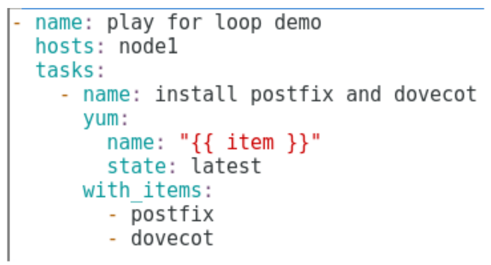
EXAMPLE

Without Loop



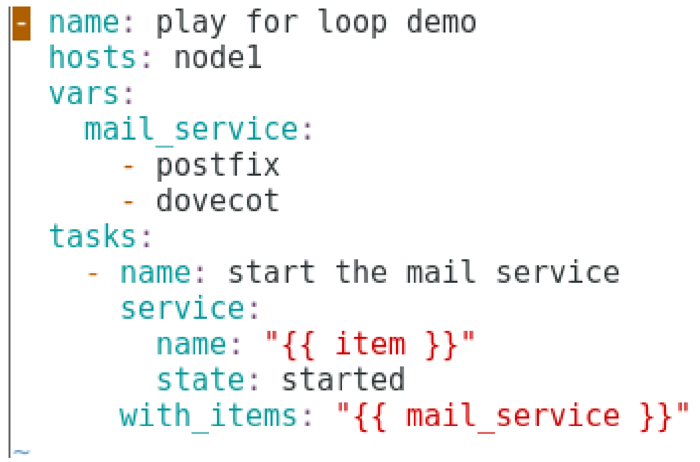
```
- name: play for loop demo
  hosts: node1
  tasks:
    - name: install postfix
      yum:
        name: postfix
        state: latest

    - name: install dovecot
      yum:
        name: dovecot
        state: latest
```

With Loop



```
- name: play for loop demo
  hosts: node1
  tasks:
    - name: install postfix and dovecot
      yum:
        name: "{{ item }}"
        state: latest
      with_items:
        - postfix
        - dovecot
```

With_items can be provided by variable also


```
- name: play for loop demo
  hosts: node1
  vars:
    mail_service:
      - postfix
      - dovecot
  tasks:
    - name: start the mail service
      service:
        name: "{{ item }}"
        state: started
      with_items: "{{ mail_service }}"
~
```

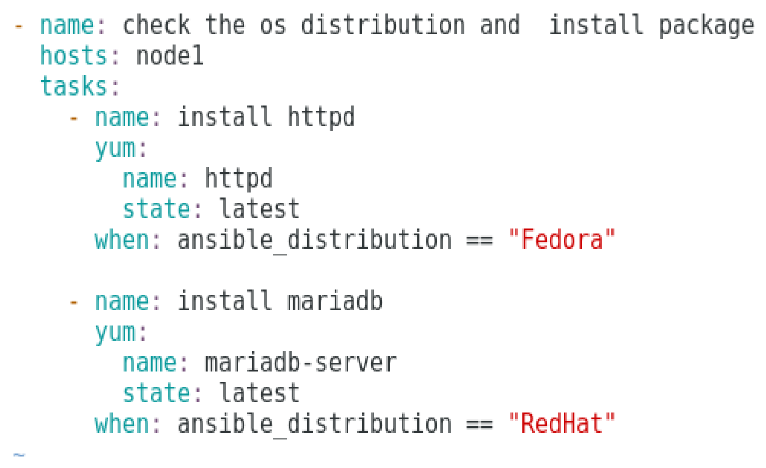
Running Tasks Conditionally

- *Ansible uses conditionals to execute tasks or play when certain conditions are met .*
- *Playbook variable ,registered variables and ansible facts can be tested with conditionals*

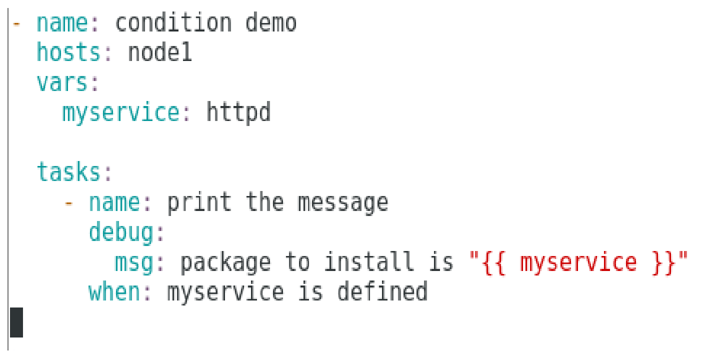
Ansible WHEN Statement

- *When statement is used to run task conditionally.*
- *It takes as a value the condition to check . If the condition is met , the tasks run. If the condition is not met, the task is skipped*

EXAMPLE



```
- name: check the os distribution and  install package
  hosts: node1
  tasks:
    - name: install httpd
      yum:
        name: httpd
        state: latest
      when: ansible_distribution == "Fedora"

    - name: install mariadb
      yum:
        name: mariadb-server
        state: latest
      when: ansible_distribution == "RedHat"
~
```



```
- name: condition demo
  hosts: node1
  vars:
    myservice: httpd

  tasks:
    - name: print the message
      debug:
        msg: package to install is "{{ myservice }}"
      when: myservice is defined
```

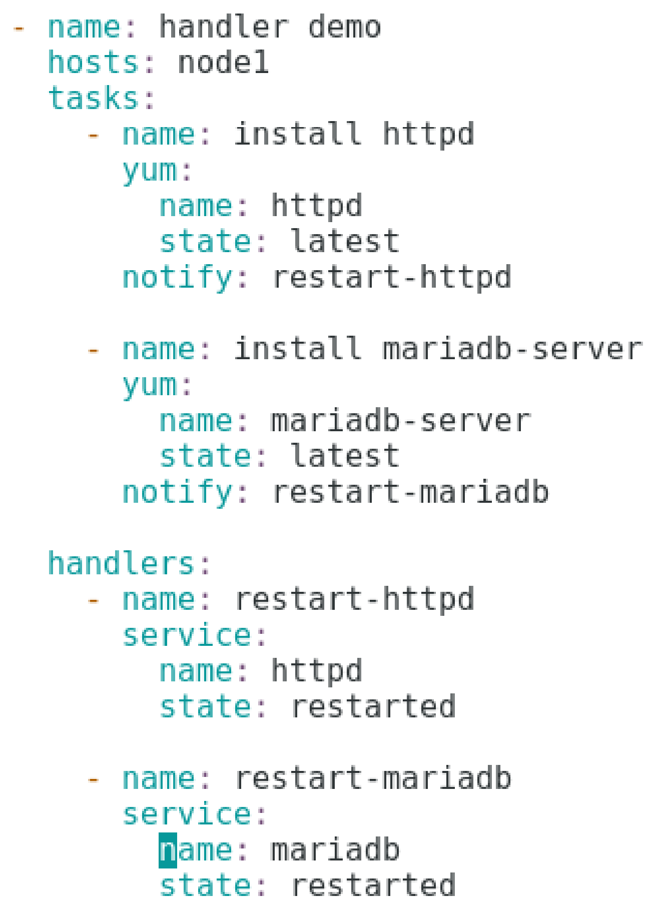
EXAMPLE-CONTINUED

```yaml
- name: check if remote path exist or not
  hosts: node1
  tasks:
    - name: check /opt/rh/remote direcotry is present
      stat:
        path: /opt/rh/remote
      register: result

    - name: if the dir is not present make it
      file:
        path: /opt/rh/remote
        state: directory
      when: result.stat.exists == false
```

Ansible Handlers

- *Handlers are the tasks that respond to a notification triggered by other task.*
- *Each handlers has globally-unique name and is triggered at the end of block of tasks in playbook*
- *If one or more tasks notify the handlers , It will run exactly once after all the other tasks in the play have completed*

EXAMPLE



```yaml
- name: handler demo
  hosts: node1
  tasks:
    - name: install httpd
      yum:
        name: httpd
        state: latest
      notify: restart-httpd

    - name: install mariadb-server
      yum:
        name: mariadb-server
        state: latest
      notify: restart-mariadb

  handlers:
    - name: restart-httpd
      service:
        name: httpd
        state: restarted

    - name: restart-mariadb
      service:
        name: mariadb
        state: restarted
```

IMPLEMENTING TAGS

- *Sometime it is useful to be able to run a particular tasks in a playbook. Tags can be applied as a text label in order to allow this.*
- *Tagging only require that the **tags** keyword be used*

```
# ansible-playbook main.yml --tags 'webserver'
#ansible-playbook main.yml –skip-tags 'webserver'
```

Handling Errors

Normally , when a task fails ansible immediately aborts the rest of the play on that hosts, skipping all subsequent tasks.

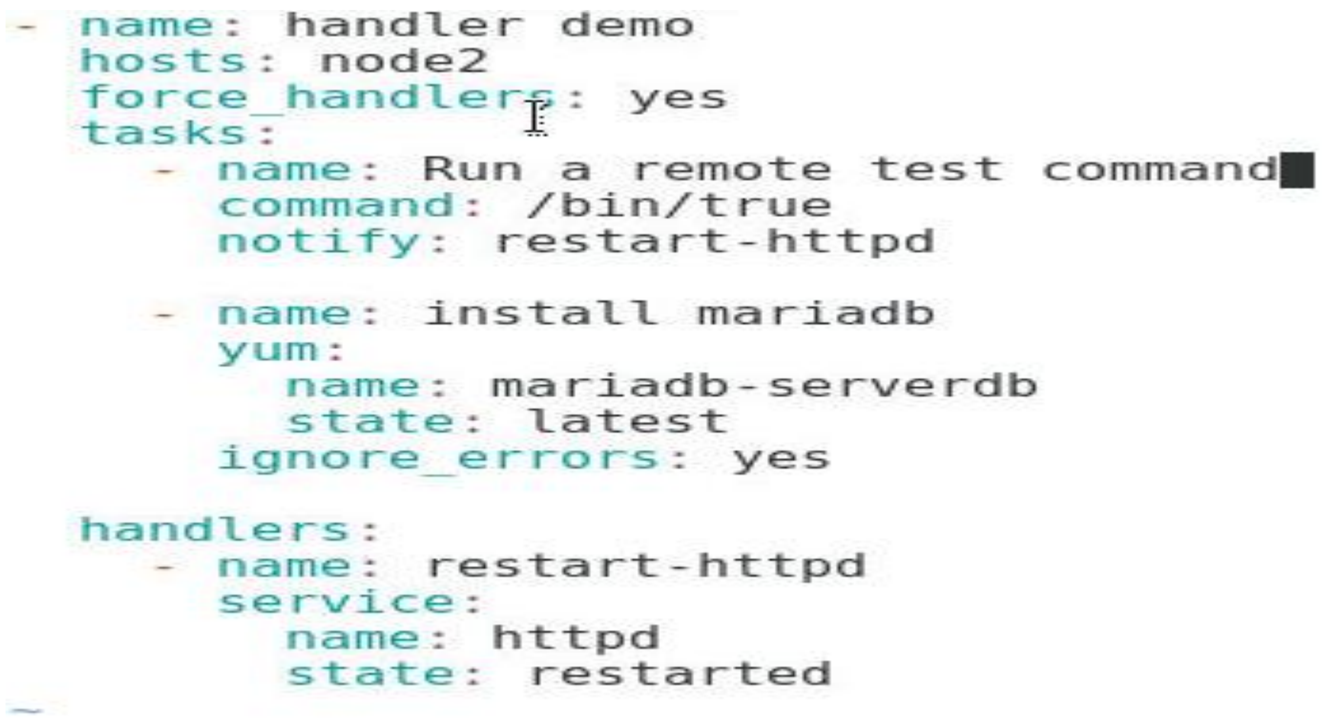
This behavior can be overridden by ignoring failed tasks.

To do so, the ***ignore_errors*** *keyword needs to be used in a task.*

Forcing Execution Of Handlers After Task Failure

*Normally if a task fails and the play aborts on that host, any handler which has been notified by earlier tasks in play will not run. If you set the **force_handlers: yes** directive on the play , then notified handlers will be called even if the play aborted because a later task failed*

Example



```
- name: handler demo
  hosts: node2
  force_handlers: yes
  tasks:
    - name: Run a remote test command
      command: /bin/true
      notify: restart-httpd

    - name: install mariadb
      yum:
        name: mariadb-serverdb
        state: latest
      ignore_errors: yes

  handlers:
    - name: restart-httpd
      service:
        name: httpd
        state: restarted
~
```

JINJA2 TEMPLATES

INTRO

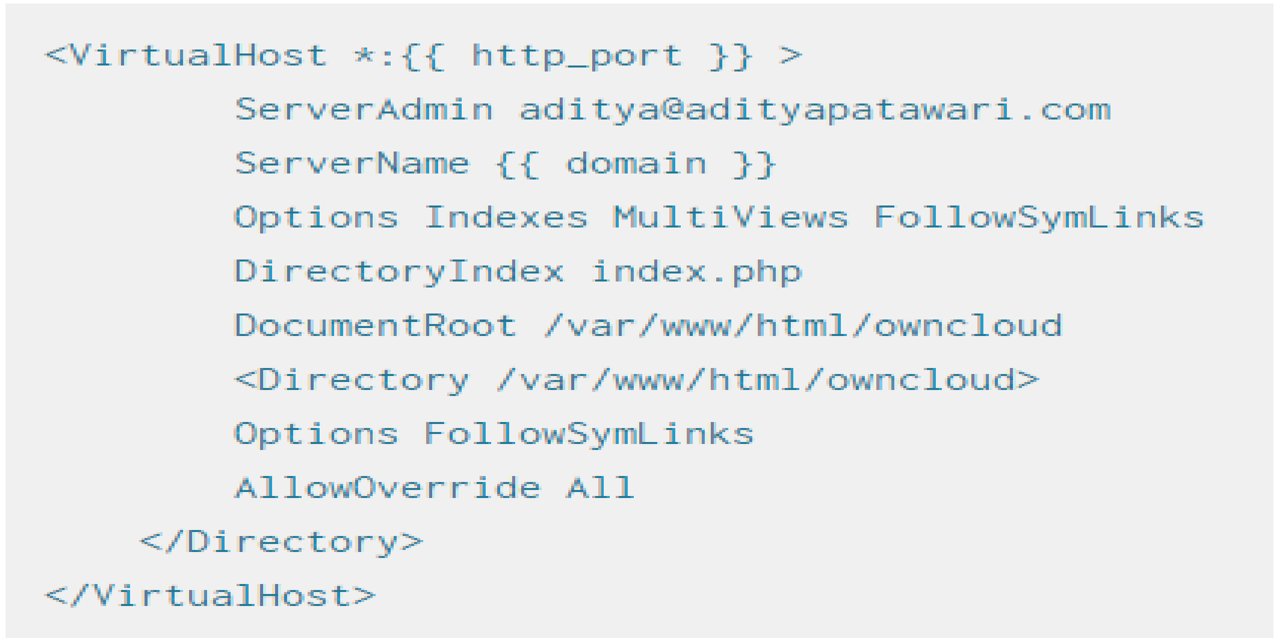
Managing configurations of multiple servers and environments are one of the significant uses of Ansible. But these configuration files may vary for each remote servers or each cluster. But apart from some few parameters, all other settings will be same.

Creating static files for each of these configurations is not an efficient solution. And It will take a lot more time and every time a new cluster is added you will have to add more files. So if there is an efficient way to manage these dynamic values it would be beneficial. This is where Ansible template modules come into play.

Cont.

Ansible uses [Jinja2]() templates which ends with .j2 extension. Based on variables, these templates fill the blanks and generate configurations. These configurations are then uploaded to the target server.

Example template:



```
<VirtualHost *:{{ http_port }} >
        ServerAdmin aditya@adityapatawari.com
        ServerName {{ domain }}
        Options Indexes MultiViews FollowSymLinks
        DirectoryIndex index.php
        DocumentRoot /var/www/html/owncloud
        <Directory /var/www/html/owncloud>
        Options FollowSymLinks
        AllowOverride All
    </Directory>
</VirtualHost>
```

Template module

# Example from Ansible Playbooks

```
- template:
    src: /mytemplates/foo.j2
    dest: /etc/file.conf
    owner: bin
    group: wheel
    mode: 0644
```

ANSIBLE-VAULT

INTRO

Ansible Vault is a feature that allows users to encrypt values and data structures within Ansible projects. This provides the ability to secure any sensitive data that is necessary to successfully run Ansible plays but should not be publicly visible, like passwords or private keys. Ansible automatically decrypts vault-encrypted content at runtime when the key is provided

How To Manage Sensitive Files with ansible-vault

*The **ansible-vault** command is the main interface for managing encrypted*

content within Ansible. This command is used to initially encrypt files

and is subsequently used to view, edit, or decrypt the data.

VAULT TASKS

Creating New Encrypted Files	**ansible-vault create foo.yml**
Editing Encrypted Files	**ansible-vault edit foo.yml**
Rekeying Encrypted Files	**ansible-vault rekey foo.yml**
Decrypting Encrypted Files	**ansible-vault decrypt foo.yml**
Viewing Encrypted Files	**ansible-vault view foo.yml**

VAULT USAGE IN PLAYBOOK

Providing Vault Passwords

To be prompted for a vault password, use the --ask-vault-pass cli option

ansible-playbook --ask-vault-pass site.yml

To specify a vault password in a text file ‘dev-password’, use the --vault-password-file option

ansible-playbook --vault-password-file dev-password site.yml

ANSIBLE ROLES

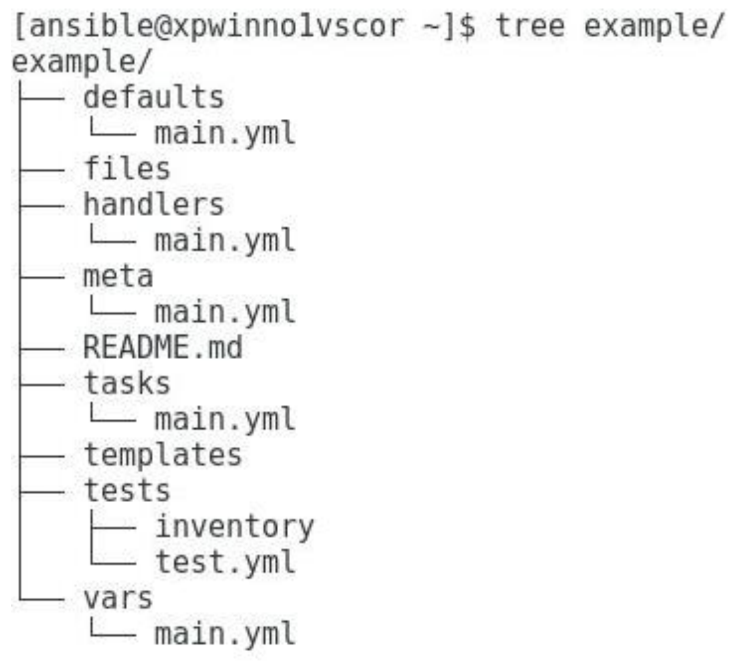
INTRODUCTION

- *Ansible roles allow admins to organize playbooks into separate, smaller playbooks and files.*
- *It provide a way to load tasks, handlers and variables from external files.*
- *Static files and templates can also be associated and referenced by a role.*

ANSIBLE ROLE STRUCTURE

- *Ansible roles functionality is defined by it directory structure*
- *Top level directory defines name of role itself*
- *Some of directory contain YAML files, named main.yml*
- *The files and templates subdirectory can contain objects referenced by the YAML file*

EXAMPLE

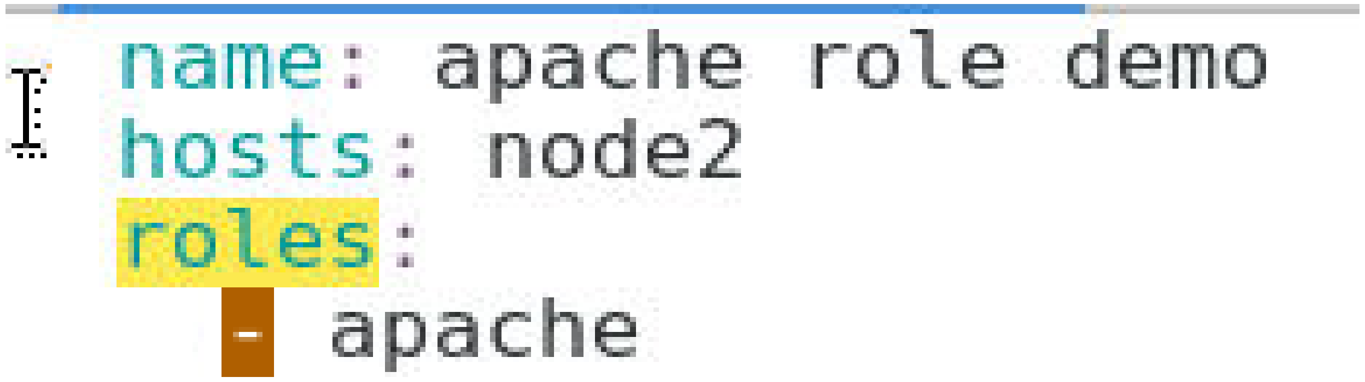


```
[ansible@xpwinnolvscor ~]$ tree example/
example/
├── defaults
│   └── main.yml
├── files
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── README.md
├── tasks
│   └── main.yml
├── templates
├── tests
│   ├── inventory
│   └── test.yml
└── vars
    └── main.yml
```

ROLE SUBDIRECTORIES

SUBDIRECTORY	FUNCTION
defaults	main.yml in this dir contain the default values of roles variables that can be overwritten when role is used
files	This contain static file that are referenced by roles tasks
handlers	main.yml in this dir contain the roles handlers definition
meta	main.yml in this dir contain information about roles, include author, license etc
tasks	main.yml in this dir contain roles tasks definitions
templates	This dir contain jinja2 template that are referenced by roles
tests	This dir can contain an inventory and test.yml playbook that can be used to test the role
vars	The main.yml file in this dir defines the roles variable values

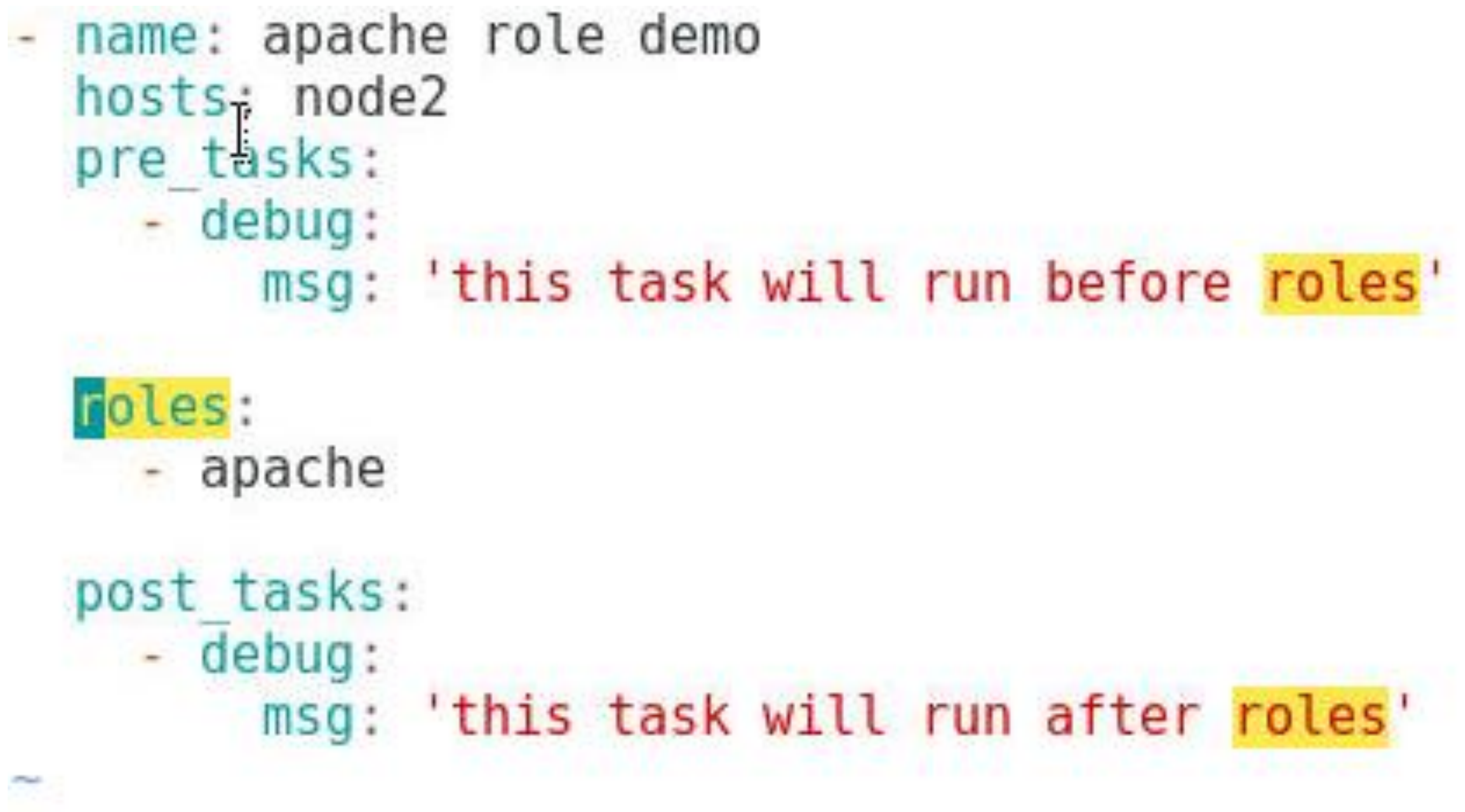
USING ANSIBLE ROLES IN PLAYBOOK



```
name: apache role demo
hosts: node2
roles:
    - apache
```

CONTROLLING ORDER OF EXECUTION

- *Normally the tasks of roles execute before the tasks of the playbooks that use them.*
- *Ansible provides a way of overriding this default behavior with keywords **pre_tasks** and **post_tasks***



```
- name: apache role demo
  hosts: node2
  pre_tasks:
    - debug:
        msg: 'this task will run before roles'

  roles:
    - apache

  post_tasks:
    - debug:
        msg: 'this task will run after roles'
~
```

ANSIBLE GALAXY

- *Ansible galaxy (https://galaxy.ansible.com) is a public library of ansible roles written by variety of ansible admins and users.*

ANSIBLE-GALAXY COMMAND LINE TOOL

Search for ansible role

```
ansible-galaxy search 'nginx' --platforms el
```

Get info of the role

```
ansible-galaxy info  maruina.nginx
```

Install the role

```
ansible-galaxy install maruina.nginx
```

Create role in offline mode

```
Ansible-galaxy init --offline role-name
```

Additional Information

- *https://docs.ansible.com/*
- *https://www.ansible.com/resources/webinars-training*
- *https://github.com/ansible/lightbulb*

THANK YOU