**ANSIBLE**

**Ansible:** Ansible is an IT automation tool. It can configure systems, deploy software, and orchestrate more advanced IT tasks such as continuous deployments or zero downtime rolling updates.

1. What exactly it does or what can it automate?

**Configuration management:** configure system, Application deployment, configuration of os, implement security policy,

**Automation:** automate the repetitive tasks, deploy one app, we need db, web servers.

**Provisioning:** providing a service, ex:deploy some db server

**Orchestration**: Orchestrate more advanced tasks like continuous deployment, zero downtime rolling updates. Step by step process, we have to create web server, then db server and configure them, we need to follow the process, otherwise it does not work.

1. Why ansible?

Simple

Easy to use

Learning curve is so easy

Written in pure python libraries

Uses openssh protocol: more secure

Agentless architecture

Easy maintenance:

Uses yaml: non-programmers also can code,

1. Advantages of ansible?

Minimal in nature

Management systems should not impose additional dependencies on the environment

Consistent

**Secure**: because it openssh which more secure than any other protocol

Highly reliable

Low learning curve

**Idempotent:** if you execute the same playbook n number of times it will not change the state of the machine.

Ansible work on test and repair method.

**Ansible execution:**

* YAML (yet another markup language)
* No programming required
* Not a markup language
* Structured
* Easy to read and write

1. Why ansible is more powerful than chef or puppet?

Agent less architecture, it is purely build upon python, it uses the ssh communication between the server and node, it uses yaml language no coding skills required.

# [Installation](http://docs.ansible.com/ansible/latest/intro_installation.html#id8):

Ansible’s release cycles are usually about four months long.

## [Requirements](http://docs.ansible.com/ansible/latest/intro_installation.html#id11):

## Currently Ansible can be run from any machine with Python 2 (versions 2.6 or 2.7) or Python 3 (versions 3.5 and higher) installed.

## On the managed nodes, you need a way to communicate, which is normally ssh. By default this uses sftp. If that’s not available, you can switch to scp in ansible.cfg.

$ git clone git://github.com/ansible/ansible.git

$ cd ./ansible

$ make rpm

$ sudo rpm -Uvh ./rpm-build/ansible-\*.noarch.rpm

## Ubuntu

$ sudo apt-get update

$ sudo apt-get install software-properties-common

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

$ sudo apt-get update

$ sudo apt-get install ansible

## Occasionally you’ll encounter a device that doesn’t support SFTP. This is rare, but should it occur, you can switch to SCP mode in [Configuration file](http://docs.ansible.com/ansible/latest/intro_configuration.html).

## --ask-sudo-pass : f using sudo features and when sudo requires a password

**Adhoc-commands:**

>ansible all –m ping –u linga #as linga

>ansible all –m ping –u linga –sudo or –u linga –b #as linga, sudoing to root

>ansible all –m ping –u linga –sudo –sudo-user leo or –b –become-user leo #as linga sudoing to leo

>ansible all –a \*/bin/echo hello\*

> ansible localhost –m ping –e ‘ansible\_python\_interpreter=”/usr/bin/env python”’

## [Become](http://docs.ansible.com/ansible/latest/become.html#id2)

Ansible allows you to ‘become’ another user, different from the user that logged into the machine (remote user). This is done using existing privilege escalation tools, which you probably already use or have configured, like sudo, su, pfexec, doas, pbrun, dzdo, ksu and others.

**become**

set to ‘true’/’yes’ to activate privilege escalation.

**become\_user**

set to user with desired privileges — the user you ‘become’, NOT the user you login as. Does NOT imply *become: yes*, to allow it to be set at host level.

**become\_method**

(at play or task level) overrides the default method set in ansible.cfg, set to *sudo*/*su*/*pbrun*/*pfexec*/*doas*/*dzdo*/*ksu*

**become\_flags**

(at play or task level) permit to use specific flags for the tasks or role. One common use is to change user to nobody when the shell is set to no login. Added in Ansible 2.2.

To run a command as the apache user:

- name: Run a command as the apache user

command: somecommand

become: true

become\_user: apache

To do something as the nobody user when the shell is nologin:

- name: Run a command as nobody

command: somecommand

become: true

become\_method: su

become\_user: nobody

become\_flags: '-s /bin/sh'

[Dynamic Inventory](http://docs.ansible.com/ansible/latest/intro_dynamic_inventory.html):

Ansible can pull inventory information from dynamic sources, including cloud sources.

Ansible easily supports all of these options via an external inventory system. The contrib/inventory directory contains some of these already – including options for EC2/Eucalyptus, Rackspace Cloud, and OpenStack.

[Ansible Tower](http://docs.ansible.com/ansible/latest/tower.html) also provides a database to store inventory results that is both web and REST Accessible. Tower syncs with all Ansible dynamic inventory sources you might be using, and also includes a graphical inventory editor. By having a database record of all of your hosts, it’s easy to correlate past event history and see which ones have had failures on their last playbook runs.

How do we write a new one?

We just create a script or program that can print JSON in the right format when fed the proper arguments. You can do this in any language.

## [AWS EC2 External Inventory Script](http://docs.ansible.com/ansible/latest/intro_dynamic_inventory.html#id6)

If you use Amazon Web Services EC2, maintaining an inventory file might not be the best approach, because hosts may come and go over time, be managed by external applications, or you might even be using AWS autoscaling. For this reason, you can use the [EC2 external inventory](https://raw.github.com/ansible/ansible/devel/contrib/inventory/ec2.py) script.

You can use this script in one of two ways. The easiest is to use Ansible’s -i command line option and specify the path to the script after marking it executable:

>ansible -i ec2.py -u ubuntu us-east-1d -m ping

The second option is to copy the script to /etc/ansible/hosts and chmod +x it. You will also need to copy the [ec2.ini](https://raw.githubusercontent.com/ansible/ansible/devel/contrib/inventory/ec2.ini) file to /etc/ansible/ec2.ini. Then you can run ansible as you would normally.

To successfully make an API call to AWS, you will need to configure Boto (the Python interface to AWS). There are a [variety of methods](http://docs.pythonboto.org/en/latest/boto_config_tut.html) available, but the simplest is just to export two environment variables:

export AWS\_ACCESS\_KEY\_ID='AK123'

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

You can test the script by itself to make sure your config is correct:

cd contrib/inventory

./ec2.py --list

After a few moments, you should see your entire EC2 inventory across all regions in JSON.

If you use Boto profiles to manage multiple AWS accounts, you can pass --profile PROFILEname to the ec2.py script. An example profile might be:

[profile dev]

aws\_access\_key\_id = <dev access key>

aws\_secret\_access\_key = <dev secret key>

[profile prod]

aws\_access\_key\_id = <prod access key>

aws\_secret\_access\_key = <prod secret key>

You can then run ec2.py --profile prod to get the inventory for the prod account, although this option is not supported by ansible-playbook. You can also use the AWS\_PROFILE variable - for example: AWS\_PROFILE=prod ansible-playbook -i ec2.py myplaybook.yml

Since each region requires its own API call, if you are only using a small set of regions, you can edit the ec2.ini file and comment out the regions you are not using.

There are other config options in ec2.ini, including cache control and destination variables. By default, the ec2.ini file is configured for **all Amazon cloud services**, but you can comment out any features that aren’t applicable. For example, if you don’t have RDSor elasticache, you can set them to False

[ec2]

...

*# To exclude RDS instances from the inventory, uncomment and set to False.*

rds = False

# To exclude ElastiCache instances from the inventory, uncomment and set to False.

elasticache = False

...

At their heart, inventory files are simply a mapping from some name to a destination address. The default ec2.ini settings are configured for running Ansible from outside EC2 (from your laptop for example) – and this is not the most efficient way to manage EC2.

If you are running Ansible from within EC2, internal DNS names and IP addresses may make more sense than public DNS names. In this case, you can modify the destination\_variable in ec2.ini to be the private DNS name of an instance. This is particularly important when running Ansible within a private subnet inside a VPC, where the only way to access an instance is via its private IP address. For VPC instances, vpc\_destination\_variable in ec2.ini provides a means of using which ever [boto.ec2.instance variable](http://docs.pythonboto.org/en/latest/ref/ec2.html#module-boto.ec2.instance) makes the most sense for your use case.

The EC2 external inventory provides mappings to instances from several groups:

**Global**

All instances are in group ec2.

**Instance ID**

These are groups of one since instance IDs are unique. e.g. i-00112233 i-a1b1c1d1

**Region**

A group of all instances in an AWS region. e.g. us-east-1 us-west-2

**Availability Zone**

A group of all instances in an availability zone. e.g. us-east-1a us-east-1b

**Security Group**

Instances belong to one or more security groups. A group is created for each security group, with all characters except alphanumerics, converted to underscores (\_). Each group is prefixed by security\_group\_. Currently, dashes (-) are also converted to underscores (\_). You can change using the replace\_dash\_in\_groups setting in ec2.ini (this has changed across several versions so check the ec2.ini for details). e.g.security\_group\_default security\_group\_webservers security\_group\_Pete\_s\_Fancy\_Group

**Tags**

Each instance can have a variety of key/value pairs associated with it called Tags. The most common tag key is ‘Name’, though anything is possible. Each key/value pair is its own group of instances, again with special characters converted to underscores, in the format tag\_KEY\_VALUE e.g. tag\_Name\_Web can be used as is tag\_Name\_redis-master-001becomes tag\_Name\_redis\_master\_001 tag\_aws\_cloudformation\_logical-id\_WebServerGroup becomes tag\_aws\_cloudformation\_logical\_id\_WebServerGroup

When the Ansible is interacting with a specific server, the EC2 inventory script is called again with the --host HOST option. This looks up the HOST in the index cache to get the instance ID, and then makes an API call to AWS to get information about that specific instance. It then makes information about that instance available as variables to your playbooks. Each variable is prefixed by ec2\_. Here are some of the variables available:

* ec2\_architecture
* ec2\_description
* ec2\_dns\_name
* ec2\_id
* ec2\_image\_id
* ec2\_instance\_type
* ec2\_ip\_address
* ec2\_kernel
* ec2\_key\_name
* ec2\_launch\_time
* ec2\_monitored
* ec2\_ownerId
* ec2\_placement
* ec2\_platform
* ec2\_previous\_state
* ec2\_private\_dns\_name
* ec2\_private\_ip\_address
* ec2\_public\_dns\_name
* ec2\_ramdisk
* ec2\_region
* ec2\_root\_device\_name
* ec2\_root\_device\_type
* ec2\_security\_group\_ids
* ec2\_security\_group\_names
* ec2\_spot\_instance\_request\_id
* ec2\_state
* ec2\_state\_code
* ec2\_state\_reason
* ec2\_status
* ec2\_subnet\_id
* ec2\_tag\_Name
* ec2\_tenancy
* ec2\_virtualization\_type
* ec2\_vpc\_id

Both ec2\_security\_group\_ids and ec2\_security\_group\_names are comma-separated lists of all security groups. Each EC2 tag is a variable in the format ec2\_tag\_KEY.

To see the complete list of variables available for an instance, run the script by itself:

cd contrib/inventory

./ec2.py --host ec2-12-12-12-12.compute-1.amazonaws.com

Note that the AWS inventory script will cache results to avoid repeated API calls, and this cache setting is configurable in ec2.ini. To explicitly clear the cache, you can run the ec2.py script with the --refresh-cache parameter:

./ec2.py --refresh-cache

## [File Transfer](http://docs.ansible.com/ansible/latest/intro_adhoc.html#id9):

## Ansible can SCP lots of files to multiple machines in parallel.

To transfer a file directly to many servers:

$ ansible atlanta -m copy -a "src=/etc/hosts dest=/tmp/hosts"

The file module allows changing ownership and permissions on files. These same options can be passed directly to the copy module as well:

$ ansible webservers -m file -a "dest=/srv/foo/a.txt mode=600"

$ ansible webservers -m file -a "dest=/srv/foo/b.txt mode=600 owner=mdehaan group=mdehaan"

The file module can also create directories, similar to mkdir -p:

$ ansible webservers -m file -a "dest=/path/to/c mode=755 owner=mdehaan group=mdehaan state=directory"

As well as delete directories (recursively) and delete files:

$ ansible webservers -m file -a "dest=/path/to/c state=absent"

Demo: mon-24

1. Launch 3 instances : control system, node1, node2

Security group: ssh, source: anywhere

Download key.pem if required

Assign one elastic ip, attach to ansible control system

>mkdir ansible\_demo

>cp ~/Downloads/ansible\_demo.pem ansible\_demo/

>cd ansible\_demo

>ls –l

#i have to change the permissions to 400 then only I can able to connect to that instance.

>chmod 400 ansible.pem

>ssh –I ansible.pem centos@ip add

#now it is connected

We need ssh-key based authentication

Process:

1. For this I am using private ips of node1 & node2

Exit from connected ec2 and

>cat ansible.pem

>ssh –I ansible.pem ec2-user@node1 ip

>vi ansible.pem

Copy the pem file to node1, change the permission to 400

>cd ~/.ssh/

>ls

Authorized keys

>ssh-keygen

>ls

Create 2 more elastic ips for the 2 nodes.

>vi ~/.ssh/config

Ansible control system must be linux.

Functionality of config file:

>ssh node1

It will be able to connect to nodes by the above command, like that we are configuring.

>vi config

Host node1

HostName publicipadd of node1

User ec2-user

Port 22

Identityfile /home/reddy/Desktop/ansible/ansible.pem

Host node2

HostName publicipadd of node2

User ec2-user

Port 22

Identityfile /home/ec2-user/ansible/ansible.pem

Save fiel

>ssh node1

Connecting to node 1 and 2

ENI: Ethernet ports : elastic network interface

Teaming: in linux, round-robbin teaming,

In traditional servers we have one Ethernet for one server,

If it fails then the servel will fail, then will attach one more Ethernet to high availability, for reducing downtime,

Assume we have 2 ethernets in the same server, those 2 ethernets can be combined to make a team, then one will fail then the other will work.

In aws, here we are using virtual network servers.

7/25

**Ansible architecture:**

**Ansible control server:**

**Node:**

**Ansible components:**

**Inventory:**

The remote server details, ip address, host\_name,

**Playbook:**

What kind of task we will achieve,

Ex:lamp stack, linux is single play, mysql is single play.

Written in yaml.

Playbook contains no of plays.

**Modules:**

Each module have set of actions to perform. Two type:

1. Core modules: ex: copy, yum, apt,
2. Community supported modules(extras)

Modules are ansible commands. In general we call it as modules.

**Ansible config:**

It’s a global configuration file, which is used to change the configurations like,

Ex:how many parallel servers can be configured on remote server, these kind of configurations we can manage in this file. Each time you have to install in 2 servers, then next 2 like that.

**Python:**

Python will package the playbook, by using ssh the package will be transferred to remote server, the playbook can be compressed by python libraries, the package will be extracted at /tmp directory and then python will take care of execute the playbook on the remote server. And it will give feedback to the control machine using ssh connection.

One or more playbooks are packaged as a one package in the remote node.

Between the control server and remote nodes we are having the connectivity by using ssh.

The result will be formatted in json format.

The order of precedence: orchestration: step by step.

**Install ansible:**

**>**ansible all –I inventory –m ping

>ansible all –I inventory –a hostname

192.168.56.102 ansible\_user=reddy ansible\_ssh\_pwd="r2"

**Class-3 7/26**

**Default ansible files:**

* Rpm –qc ansible #what files are installed with ansible

/etc/ansible/ansible.cfg

/etc/ansible/hosts #notation will be inventory or hosts

We can create in 4 ways: **configuration settings order** of operations

1. $ANSIBLE\_CONFIG #give as environment variable

$ANSIBLE\_<configsetting> ex:export ANSIBLE\_FORKS=10

It will 10 operations parellely.

**Configuration parameters:**

1. **Forks**: Default set to 5: total no of parallel operations ansible exectes

Production recommendation: start at 20, and go up or down depending upon performance.

2.**log\_path**:

Default set to null: write information on ansible executions

Set path to log file: make sure all users of ansible has permissions to write

1. ./ansible.cfg #you can give in the present working directory: the best way to do it. We need to maintain the all the configurations for all servers, so here we need to maintain the cfg files for each server.
2. ~/.ansible.cfg #you can in home dir of the user
3. /etc/ansible/ansible.cfg #in the specified path

When you execute the ansible adhoc commands it will search in the above order.

Ansible will first look in the environment variables

**Ansible.cfg:** create one file add the log\_path, inventory under default group

**Inventory:** create one inventory file and give the host details

**>ansible all –**m ping

Here when you execute the adhoc cmd it will generate the log file

**Inventory features:**

**a)Behavioral parameters**: ssh username, pwd, private key file, port no etc.

**b)Groups**: test db, test servers, prod servers etc.

c)groups of groups:

d)**assign variables**: assign variables to groups, you want to pass the variables through inventory file for different version installation.

e**)scaling out using multiple files:** maintaining the multiple inventory files for each individual projects or servers.

f)**static/dynamic**: inventory may be static or dynamic.

Ex: inventory file

[db]

Db1.x.com ansible\_ssh\_user=linga ansible\_ssh\_port=22

Db2.x.com ansible\_ssh\_user=reddy ansible\_ssh\_port=22

Db3.x.com ansible\_python\_interpreter=/usr/bin/python3

[datacenter-east:children]

Db

[datacenter-east:vars]

Ansible\_ssh\_user=linga #these are used if we don’t mention in the db groups like db1,db2 have same values then we can mention here.

Ansible\_ssh\_port=22

Configure ec2 instance key file:

Copy the .pem file and

>vi ansible\_demo.pem

Paste here, change permission to 400

Now change the inventory file: ansible\_ssh\_private\_key\_file=./ansible\_demo.pem ansible\_ssh\_port=22

>ansible all –m ping

>ansible test-server –m ping

Inventory:

[datacenter:children] #group of groups, the keyword must be mentioned

Webserver

dbserver

>ansible datacenter –m ping

We can assign the variables for the groups

[datacenter:vars]

Ansible\_ssh\_user=reddy

Ansible\_ssh\_private\_key\_file=./ansible\_demo.pem

Ansibe\_ssh\_port=22

Remove the vars details in the top that is not required if we mention in the vars

Ansible has the enough knowledge to avoid the duplicate ips in the inventory file.

On the node: when you execute the adhoc command there will no log, bt for playbooks the will be there.

>/var/log/messages or secure #you can find the log files on the node

>tail –f /var/log/secure

Default parameters: inventory, library,

>ansible \*server –a hostname

>ansible dbserver:!webserver –a hostname #only execute on db server

>ansible webserver:dbserver –a hostname #

>ansible datacenter[0] –m ping #web

[0:1] #web and db

[0:2] #web,db,lb

**Class4: 07/31**

**How to create user and set password to the user?**

**>**ansible-doc user

Will give the user module details

Sha512 mechanism for password encryption.

**Playbook for user creation:**

---

- hosts: all

become: true

gather\_facts: no

vars:

password: ramakrishna2

tasks:

- name: creating test user

user:

name: test

comment: "this is a test user"

shell: /bin/bash

password: "{{ password | password\_hash('sha512') }}"

>ansible-playbook --ask-sudo-pass user\_creation.yml

to check whether the user is created or not in the remote server:

>id test

>su test

The above playbook is static one, bt we need dynamic playbook while we are running it should ask the username, shell etc.

>vi /etc/passwd

Hosts: webserver

Hosts: dbserver

We can create two plays for 2 different servers,

Shell is constant in all cases, so we can store it in vars and we call it.

**Including files:**

**Ex:**

Tasks:

* Include: wordpress.yml

Vars:

Sitename: my awesome site

* Include: loadbalancer.yml
* Include\_vars: variables.yml

Breaks up long playbooks. Use to add external variable files. Reuse other playbooks.

Register task output:

Grab output of task for another task

Useful to use tasks to feed data into other tasks

Useful to create custom errortrapping

Ex:

Tasks:

* Shell: /usr/bin/whoami #login user

Register: username

* File: path=/home/myfile.txt #changing the file ownership to username

Owner={{ username }}

**Debug module:**

Ex:

Tasks:

* Debug:
* Msg=”this host is {{ inventory\_hostname }} during execution
* Shell: /usr/bin/whoami

Register: username

* Debug: var=username

Useful to send output to screen during execution. Helps find problems.

**Prompting for input:**

Prompt user during execution

Ex:

* Hosts: web1

Vars\_prompt:

* Name: “sitename”

Prompt: “what is your new site name?”

Tasks:

* Debug: msg=”the name is {{ sitename }}”

Creates dynamic playbooks.

**Playbook handlers:**

Tasks with asynchronous execution . only runs tasks when notified. Tasks only notify when state=changed

Does not run until all playbook tasks have executed. Most common for restarting services to load changes(if changes are made)

>sudo systemctl status httpd

---

- hosts: all

become: yes

gather\_facts: no

tasks:

- name: installing apache

apt: name=apache2 update\_cache=yes state=latest

notify:

- restart apache

handlers:

- name: restart apache

service:

name: httpd

state: restarted

>vi php.yml

- name: installing php

Yum:

Name: php

State: latest

**Class-5: 8/1**

**Test:**

Write a playbook to execute any commands and display the output

Handelers are executed when the status is changed to false.

Apache playbook:

* Hosts: all

Become: true

Tasks:

* Name: install apache

Yum:

Name: httpd

State: latest

* Name: apache restart

Service:

Name: httpd

Stste: started

**How to create aws instances by using playbooks?**

Ec2-module

1.create one ami user: give programmatic policy roles

User able to create ec2-instance

a)create one user

add user: name: madhira

access type: y: programmatic access through api calls it will access to services: aws api(communication through) no ssh, we are going to create the ec2-instance

n: aws management control: user able to access control on console, eligible to login into console

b)attach exisiting policy: ec2fullaccess can do anything on aws ec2 instance dashboard

c)create user, download csv

access key and secrest access key are required to create ec2-instance through ansible.

1. Write playbook for ec2-instane creation

Ec2.yml

---

- name: create a sandbox instance

hosts: 127.0.0.1

gather\_facts: no

vars:

keypair: lingawebkey

instance\_type: t2.micro

image: ami-4fffc834

region: us-east-1

ackey: AKIAIFNLV5H6YCW4IVWA

sekey: b/3qdzNUJfkBuY7X2qmPAJnlA16nQyTDb6WlbCYR

tasks:

- name: lauch ec2-instance

ec2:

key\_name: "{{ keypair }}"

group: default

instance\_type: "{{ instance\_type }}"

image: "{{ image }}"

wait: true

region: "{{ region }}"

aws\_access\_key: "{{ ackey }}"

aws\_secret\_key: "{{ sekey }}"

register: ec2

- name: print all ec2 variables

debug: var=ec2

**when I am trying to connect to created ec2-instance its not connectin**g: reason

>ssh –I “ansible.pem” ec2-user@ip

in my playbook I did not mentioned the security groups so its not connecting

edit inbound: add rule

ssh 22 any where save

then its connecting

**conditional execution:**

use the clause “when to choose if task should run

ex:

tasks:

* Yum: name=httpd state=present

When: ansible\_os\_family==”RedHat”

* Apt: name=apache2 state=present

When: ansible\_os\_family==”Debian”

Use yum if os is redhat, uses apt if os is debian.

**Conditional clause based on output:**

Ex:

Tasks:

* Command: ls /path/doesn’t/exist

Register: result

Ignore\_errors: yes

* Debug: msg=”Failure”

When: result | failed

Choose when to execute tasks. Track whether previous task ran, searches json result for status.

Status options: success, failed, skipped.

**Templates:**

Uses jinja2 engine, insert variables into static files.

Creates and copies dynamic files. Deploy custom configurations.

**Class-6 08/2**

For python 2.7 – pip install boto

Python3 – pip install boto3

If somebody have the accesskey and secret key then they can create ec2 instances.

1. **Ansible vault**: to store the secret key and accesskey values.

Valuts are files which can be used to

**-e is used as flag**

Vault are in 5 modes, cmd:ansible vault

I want to deploy the ec2 on particular vpc. Then we need to mention the vpc id or particular subnet then provide the subnet id

For particular security group, you need to mention group: security group name

For elastic ip:

* Name: allocate a new elastic ip without associating it to anything

Ec2\_eip:

State: present

Region: “{{ region }}”

Aws\_access\_key: “{{ ackey }}”

Aws\_secret\_key: “{{ sekey }}”

Register: ec2

* Name: displaying created eip

Debug:

Msg: “allocated elastic ip is {{ eip.pulic\_ip }}”

In ec2 section:

Vpc\_subnet\_id: “{{ vpcid }}”

Register: ec2

* Name: associate new elasticip with created ec2 instance

Ec2\_eip:

Device\_id: “{{ item }}”

Region: “{{ region }}”

Aws\_access\_key: “{{ ackey }}”

Aws\_secret\_key: “{{ sekey }}”

With\_items: “{{ ec2.instance\_ids }}”

**Cmd to create ansible vault:**

>ansible-vault create ec2\_keys.yml

New vault pwd:

Open in the editor:

---

Ackey: accesskey

Sekey: secret key

Save it and

>vi ec2\_keys.yml

It shows the keys in encrypted format

>edit ec2\_keys.yml

Ask for the pwd, then it will show the real keys

In the ec2.yml we need to include the ec2\_keys.yml file

Tasks:

* Include\_vars: ec2\_keys.yml

>ansible-playbook ec2.yml –ask-vault-pass

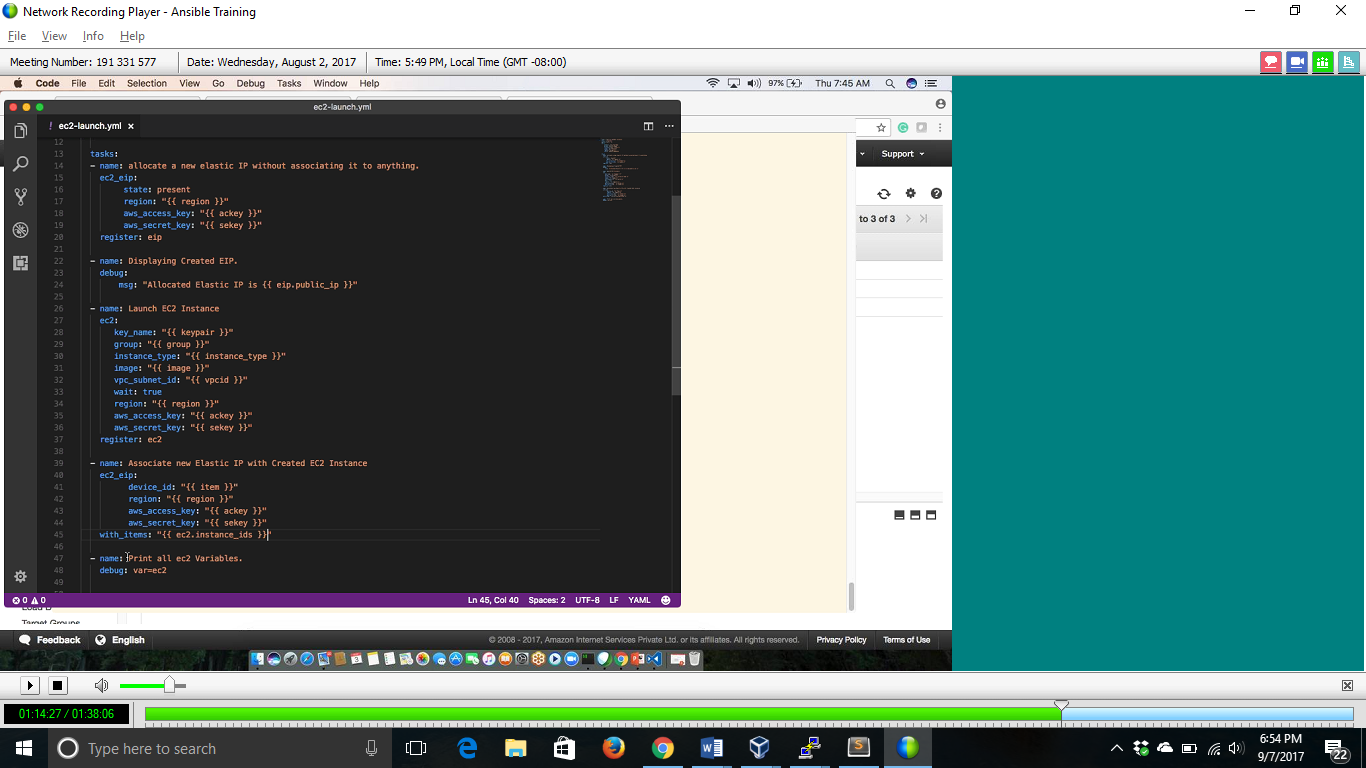
It ask the vault pwd then it will execute the playbook

>vi key.txt

Vaultpassword

Save it

>ansible-playbook ec2.yml –vault-password-file .key.txt



**How to recover the vault password?**

>ansible-vault rekey .key.txt #to set new password

**Tags module:**

Out of all tasks to execute only required tasks using tags we can achieve this.

>in the ec2.yml

Under the first task

Tags:

* Elasticip

Under second task

Tags:

* Ec2instance

>ansible-playbook ec2.yml –vault-password-file .key.txt –tags elasticip #to execute the particular task specify the tag under that task

>ansible-vault edit ec2\_keys.yml

How to delete the encryption?

>ansible-vault decrypt ec2\_keys.yml

After this command you can see the normal password

**Class-7 8/3**

**Using items in playbook:**

* Hosts: all

Become: true

Gather\_facts: no

Tasks:

* Name: installing {{ item }}

Yum: name={{ item }} state=latest

With\_items:

* Vim
* Tree

To copy a file from local to

Remote, if it already exists then it will change the permissions too, you need to mention everything in the playbook like the owner, permissions.

>ansible all –m copy –a “src=/home/centos/user\_creation.yml dest=/home/centos/ backup=yes”

If I want to change the ownership of the files, then what is the parameter to achieve that?

Two ways, 1. User module 2.we have a command module and use that and change the files permissions.

Difference between copy and fetch module?

We have to install apache on 2 servers when the server is ubutntu and when it is centos, use the when clause?

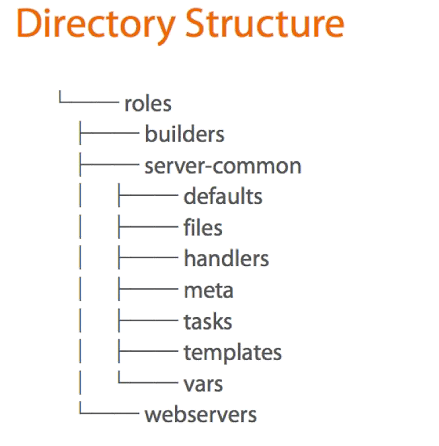
**ROLES:**

To configure the entire site we need roles, we can configure and install n number of things.

The role will be useful for other persons,

**Role ex**: wordpress, mysql, jboss, repository and server-common, build.

**Roles dir structure:**





In general templates will be in the jinja2 format.

Why templates?

I installed apache, we need to configure in apache.cfg. I have 100 servers then we have to configure all servers.using yum module I can install apache, but the configuration files, I created one config file and palce it template dir and it will replace the config file in /etc/../conf

Difference between copy and templates?

Copy will copy the files to the remote server, for changing content dynamically its not possible with the copy module we will use the templates.

If you mention the dynamic files or variables in the templates at the time of execution, it will prompt to the user to ask the on which port it want to run like that. We can do dynamic configurations with templates. The file extension should be ‘.j2’.

Tasks/main.yml

---

* Include: webservers.yml
* Include: dbservers.yml #can add includes to break-up long files

Site.yml

Primary file to include entire infrastructure. In parallel with handler’s folder.

We can include multiple playbooks

---

* Include: web.yml tags=web
* Include: db.yml tags=db

Use tags to define categories within your playbooks.

Tagging tasks:

Tasks:

Debug: msg=”this will only run on tag ‘debug’”

Tags:

* Debug

Debug: msg=”you can also use multiple tags”

Tags:

* Debug
* Ubercool

>ansible-playbook site.yml #executing roles

**Demo:**

LAMP tree structure

-ansible.cfg #configuration file

Ansible.log

Hosts #inventory file

Main.yml #main playbook

Roles

In roles we have different directory, these are called as role

Ex:epel-repo, httpd,mariadb, php, phpmyadmin, prereqs

Epel-repo: only we have tasks, other are not needed

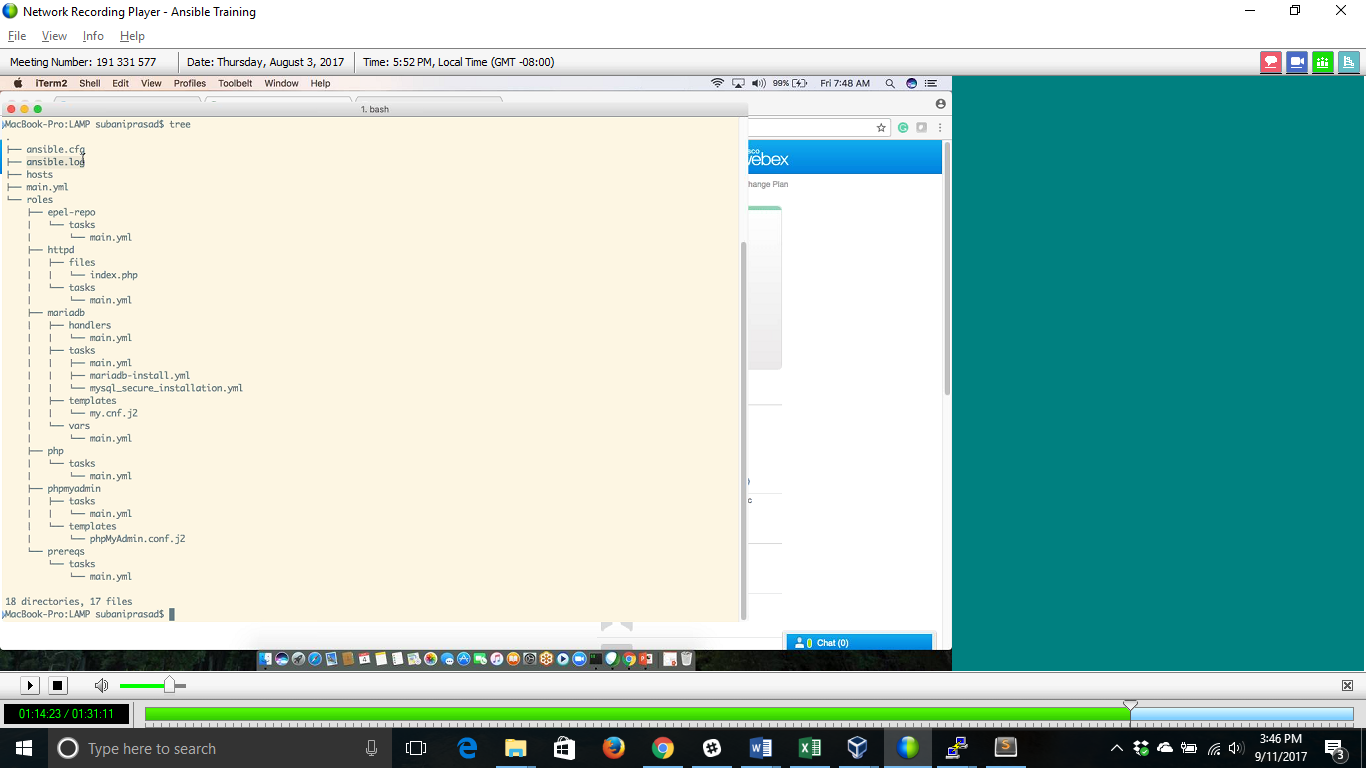
Httpd: files:index.php Tasks:mail.yml

Mariadb: handlers:main.yml tasks:main.yml, maria-install.yml, mysql\_secure\_installation.yml templates: my.cnf.j2

Vars: main.yml

Php: tasks:main.yml templates: phpmyadmin.conf.j2 #its a client tool to connect to mysql db, it’s a web-bsed ui

Prereqs: tasks:main.yml



How to create role from the playbook?

In the playbook cut the vars section and in the roles folder create a folder called varsand inside it create a task dir and in that folder create a file name main.yml in that you can paste the vars section. When the playbook is executing it will take the vars from the roles and execute it.

**Class-8 8/4**

**Ansible Playbooks:**

Playbooks in Ansible are written in YAML format. It is a human-readable data serialization language. It is commonly used for configuration files. It can also be used in many applications where data is being stored.

For Ansible, nearly every YAML file starts with a list. Each item in the list is a list of key/value pairs, commonly called a “hash” or a “dictionary”. So, we need to know how to write lists and dictionaries in YAML.

All members of a list are lines beginning at the same indentation level starting with a “- ” (dash and space). More complicated data structures are possible, such as lists of dictionaries or mixed dictionaries whose values are lists or a mix of both.

 e.g. For a list of departments in edureka:

departments:

- marketing

- sales

- solutions

- content writing

- support

- product

Now let me give you an example of a dictionary:

-USA

-continent: North America

-capital: Washington DC

-population: 319 million

**Hosts And Users:**

For each play in a playbook, you get to choose which machines in your infrastructure to target and which remote user to complete the tasks. To include hosts in Ansible inventory, we will be using the IP addresses of the host machines.

Generally the hosts are a list one or more groups or host patterns, separated by colons. The remote user is just the name of the user account.

**Variables:**

Ansible uses variables which are defined previously to enable more flexibility in playbooks and roles. They can be used to loop through a set of given values, access various information like the host name of a system and replace certain strings in templates with specific values.

Ansible already defines a rich set of variables, individual for each system. Whenever Ansible will run on a system, all facts and information about the system are gathered and set as variables.

But there is a *rule* for naming variables. Variable names should be letters, numbers, and underscores. Variables should always start with a letter. E.g. wamp\_21, port5 is valid variable names, whereas 01\_port, \_server are invalid.

**Tasks:**

Tasks allow you to break up bits of configuration policy into smaller files. Task includes pull from other files. Tasks in Ansible go with pretty much the English meaning of it.

E.g: Install <package\_name>, update <software\_name> etc.

**Handlers:**

Handlers are just like regular tasks in an Ansible playbook, but are only run if the Task contains a notify directive and also indicates that it changed something.

For example, if a config file is changed, then the task referencing the config file may notify a service restart handler.

Let me give you an example of a playbook which will start the Apache httpd server program:

---

- hosts: webservers

vars:

http\_port: 80

max\_clients: 200

remote\_user: root

tasks:

- name: ensure apache is at the latest version

yum: name=httpd state=latest

- name: write the apache config file

template: src=/srv/httpd.j2 dest=/etc/httpd.conf

notify:

- restart apache

- name: ensure apache is running (and enable it at boot)

service: name=httpd state=started enabled=yes

handlers:

- name: restart apache

service: name=httpd state=restarted

I hope the example will relate you to all the description of the playbook components that I have mentioned above. If it is still not clear to you, don’t worry all your doubts will be clear in the later part of this blog.

**Ansible – Modules**

Modules in Ansible are idempotent. From a RESTful service standpoint, for an operation (or service call) to be idempotent, clients can make that same call repeatedly while producing the same result. In other words, making multiple identical requests has the same effect as making a single request.

There are different types of modules in Ansible

* Core modules
* extras modules

**Core Modules**

These are modules that the core Ansible team maintains and will always ship with Ansible itself. They will also receive slightly higher priority for all requests than those in the “extras” repos.

The source of these modules is hosted by Ansible on GitHub in the Ansible-modules-core.

**Extras Modules**

These modules are currently shipped with Ansible, but might be shipped separately in the future. They are also mostly maintained by the Ansible community. Non-core modules are still fully usable, but may receive slightly lower response rates for issues and pull requests.

Popular “extras” modules may be promoted to core modules over time.

The source for these modules is hosted by Ansible on GitHub in the Ansible-modules-extras.

E.g: The one of the extras module in Remote Management Modules is ipmi\_power module, which is a power manger for the remote machines. It requires python 2.6 or later and pyghmi to run.

You can use this module by writing an adhoc command like the one I have written below:

ipmi\_power : name ="test.domain.com" user="localhost" password="xyz" state="on"

**Ansible - Return Values**

Ansible modules normally return a data structure that can be registered into a variable, or seen directly when output by the Ansibleprogram. Each module can optionally document its own unique return values.

Some examples of return values are:

* changed: returns with a boolean value whenever the task makes any change.
* failed: returns a boolean value, if the task is failed
* msg: it returns a string with a generic message relayed to the user.

**Ansible – AdHoc Commands**

Adhoc commands are simple one line command to perform some action. Running modules with Ansible commands are adhoc commands.

E.g:

ansible host -m netscaler -a "nsc\_host=nsc.example.com user=apiuser password=apipass"

The above adhoc command uses the netscaler module to disable the server. There are hundreds of modules available in Ansible from where you can refer to and write adhoc commands.

**Ansible – Hands On**

## ****Install Ansible:****

This blog will guide you to install Ansible in your **CentOS** machine in two easy steps.

You will also learn how to setup your node machines and I will show you an example of how to install Nginx in your node machine from your control machine.

Let’s begin with the installation now.

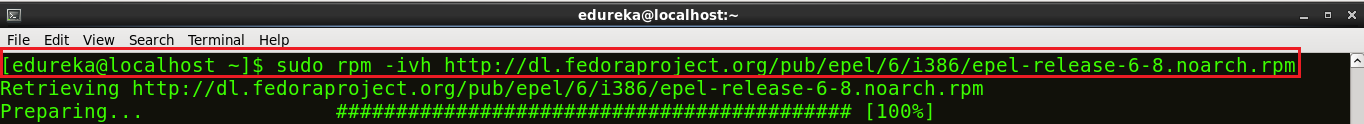
## ****Install Ansible on CentOS 6.8****

**Step 1:** Set**EPEL** Repository

**EPEL** (Extra Packages for Enterprise Linux) is open source and free community based repository project from Fedora team which provides high quality add-on software packages for Linux distribution including RHEL (Red Hat Enterprise Linux), CentOS, and Scientific Linux.

Ansible package is not available in the default yum repositories, so we will enable EPEL repository for CentOS 6.8 using the command below:

**sudo rpm -ivh http://dl.fedoraproject.org/pub/epel/6/i386/epel-release-6-8.noarch.rpm**

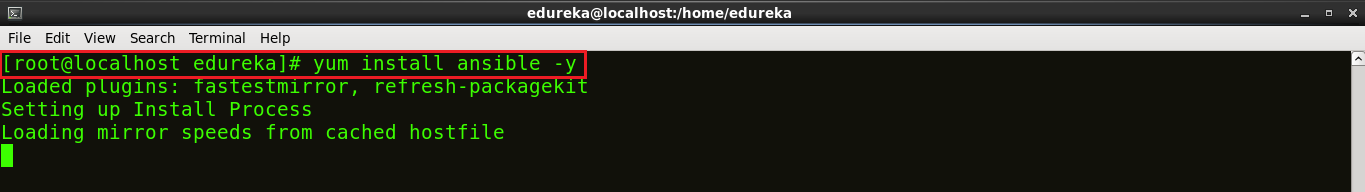


This will download all the necessary packages which will be required to install Ansible.

**Step 2:** Install Ansible

Now that your EPEL repository has been added, all you have to do now is install Ansible using the command below:

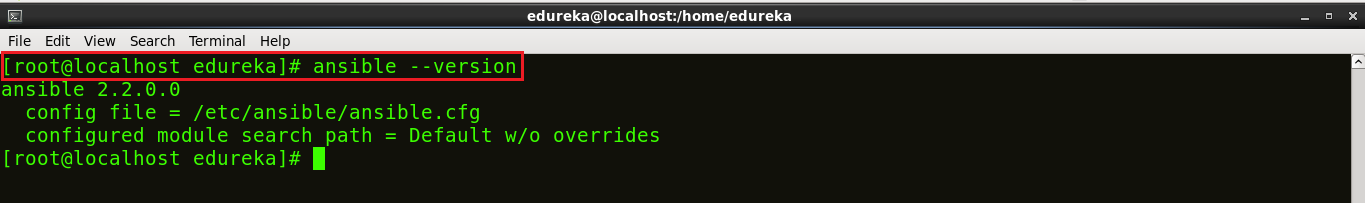
**yum install ansible -y**



And its done. I told you Ansible is easy! :D

Now, if you want to check out your version of Ansible which you have installed, you can use the command below:

**ansible –version**



You can see in the above snapshot that Ansible version 2.2.0.0 has been installed in your system

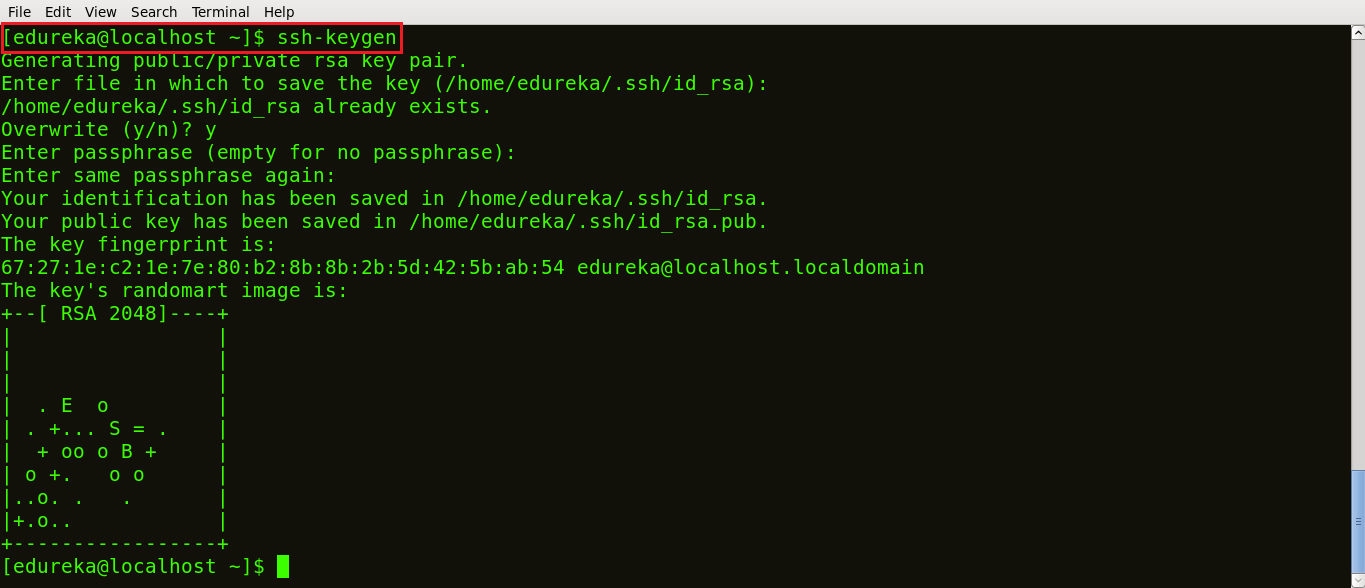
## ****Post Installation****

After installing Ansible you need to add servers which you want to manage through Ansible. For that I have created another CentOS VM which will act as my node machine.

The first task is to setup password-less SSH authentication with nodes on my Control machine.

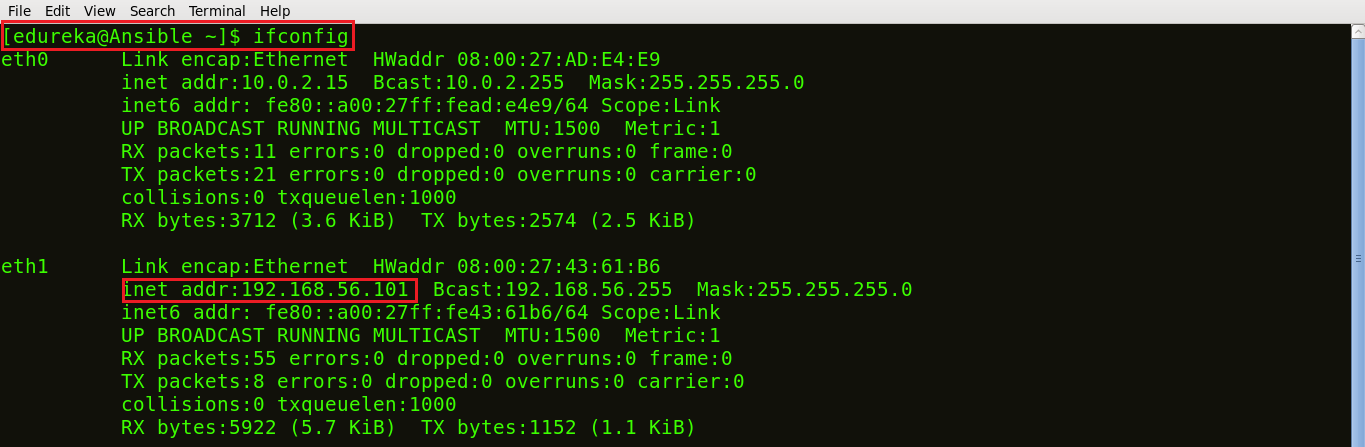
**Step 1:** Generate SSH key on the Ansible Control Machine. In order to do that, use the command below:

**ssh-keygen**



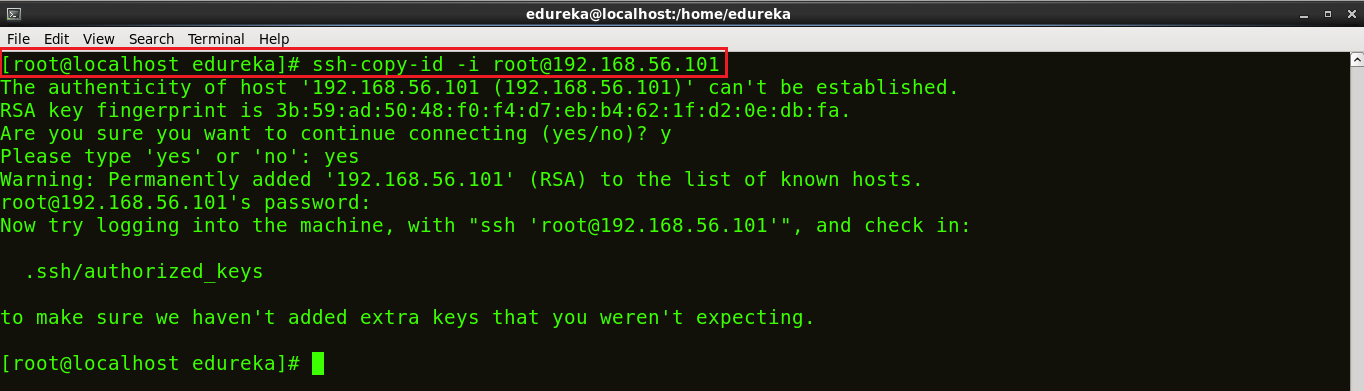
As you can see in the above snapshot a public SSH key has been generated.

**Step 2:** Now, check the IP addresses of your nodes because you will need to specify it in the Ansible inventory later. To do that, type in the command**ifconfig** on your node terminal.



**Step 3:** After the key is generated, the next task is to copy public key of Ansible server to its nodes. Use the command below:

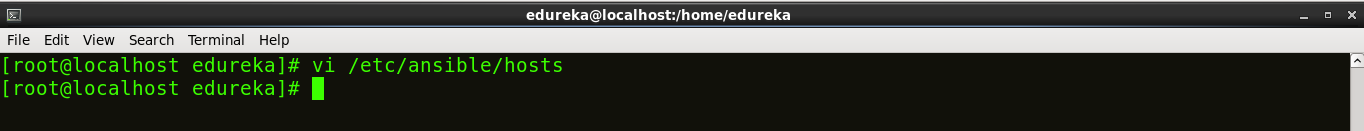
**ssh-copy-id -i root@<ip address of your node machine>**



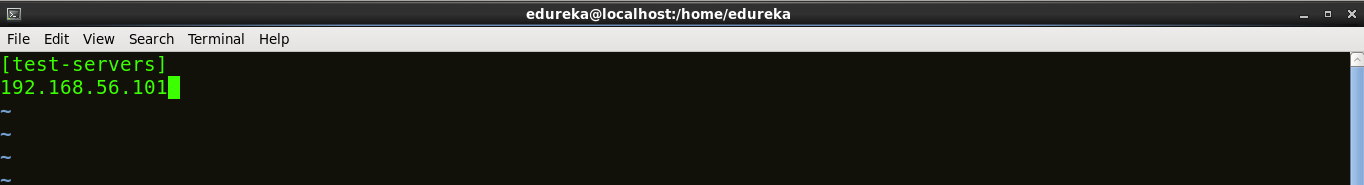
**Step 4:** Now, you can use any editor to write your inventory or specify the IP addresses of your nodes grouped under ‘test-servers’ (or whatever you want your group name to be). I am using the vi editor.

use the following command:

**vi /etc/ansible/hosts**



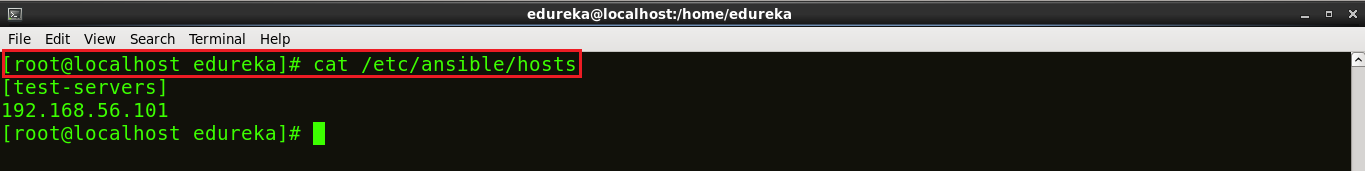
This will open a vi editor like the snapshot below:



After you have added the IP address under ‘test-servers’, save the file and then exit.

**Step 5:** You might want to check if the IP address of your host has been added or not. To sample output of the host files using the following command:

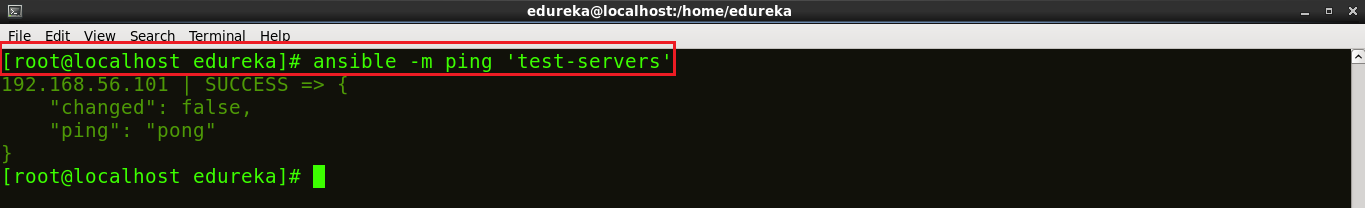
**cat /etc/ansible/hosts**



You can see the IP address of my host being displayed in the snapshot above.

**Step 6:** Let us now perform a simple ping operation to test the connectivity using Ansible. To do that, just type the following command:

**ansible -m ping ‘test-servers’**



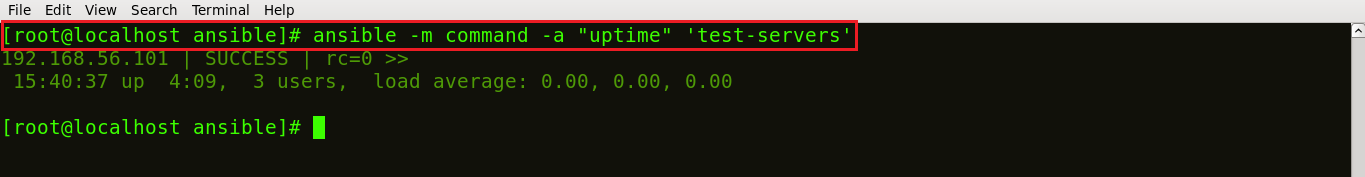
Now, that you have checked the connection with your host machines, you are ready to manage them with Ansible.

Let me show you few shell command examples using Ansible.

* **Check uptime of your node machines**

Uptime is a measure of the availability and working time of a computer. In order to check that use the following command:

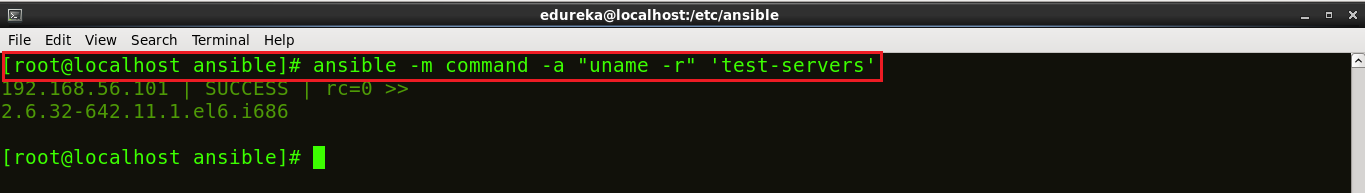
**ansible -m command -a “uptime” ‘test-servers’**



* **Check kernel version of your nodes**

It can be useful to know the version number of the kernel (i.e., the core of the operating system). Use the command below:

**ansible -m command -a “uname -r” ‘test-servers’**



Now, let us install Nginx into my node machine from my control machine using Ansible.

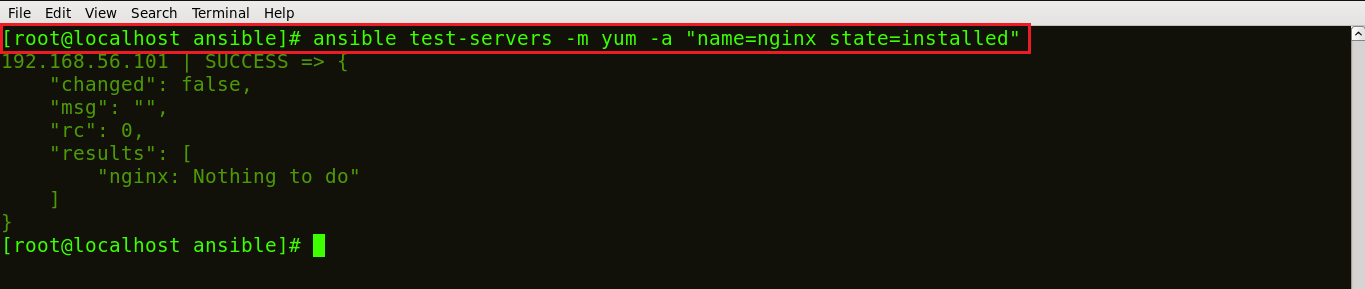
## ****Deploy Nginx Using Ansible****

**Nginx** is software to provide a web server. It can act as a reverse proxy server for TCP, UDP, HTTP, HTTPS, SMTP, POP3, and IMAP protocols, as well as a load balancer and an HTTP cache.

I am deploying Nginx using Ansible in just one node. You can deploy it in multiple nodes too using the same way. All you have to do is list out the IP addresses of the nodes under ‘test-servers’.

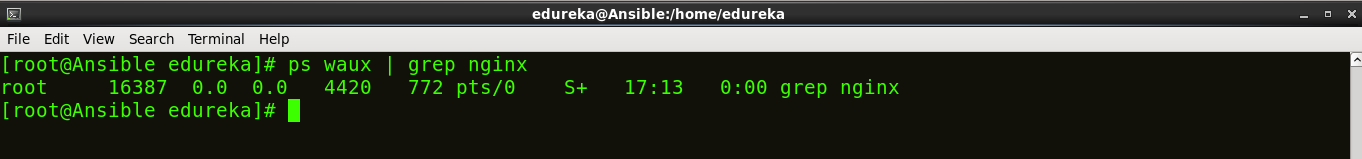
Use the following command in your control machine:

**ansible test-servers -m yum -a “name=nginx state=installed”**



Now to check if it is installed in your node machine, type the following command in your node:

**ps waux | grep nginx**



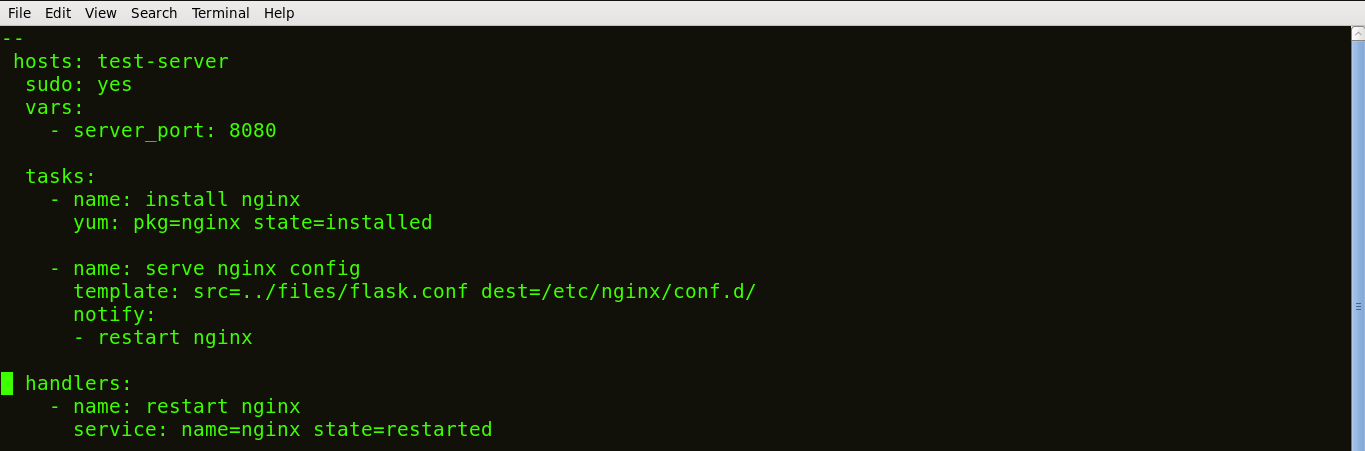
The above snapshot shows that few processes are running with the PID 16387 and 772 which indicates that Nginx has been installed and it is ready to get started.

I have used Ad hoc commands here to install Nginx on my nodes, but you can do the same using Ansible playbooks or using the pre-defined Ansible modules as well.

**Step 5:** Let us now write a playbook to install Nginx on the host machine. You can write your playbook in the vi editor. For that, simply create your playbook, using the command:

**vi <name of your file>.yml**

The below snapshot shows my playbook to install Nginx written in YAML format.



The tasks of a playbook are defined in YAML as a list of dictionaries and executed from top to bottom. If we have several hosts, then each task is tried for each host before moving on to the next one. Each task is defined as a dictionary that can have several keys, such as “name” or “sudo” which signify the name of the task and whether it requires sudo privileges.

A variable *server\_port* is set that listens on TCP port **8080** for incoming requests.

Here, the first task is to get the necessary package for installation of Nginx and then install it. Internally, Ansible will check if the directory exists and create it if it’s not, otherwise it will do nothing.

The next task is to configure Nginx. In Nginx, contexts contain configuration details.

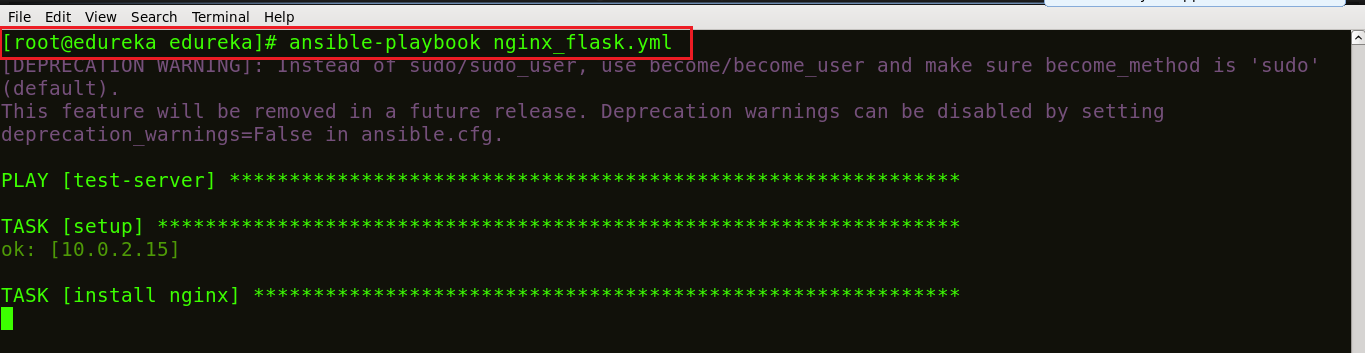
Here, the template is a file you can deploy on hosts. However, template files also include some reference variables which are pulled from variables defined as part of an Ansible playbook or facts gathered from the hosts. Facts containing the configuration details are being pulled from a source directory and being copied to a destination directory.

Handlers here define the action to be performed only upon notification of tasks or state changes. In this playbook, we defined, notify: restart Nginx handler which will restart Nginx once the files and templates are copied to hosts.

Now, save the file and exit.

**Step 6:** Now let’s run this playbook, using the command below:

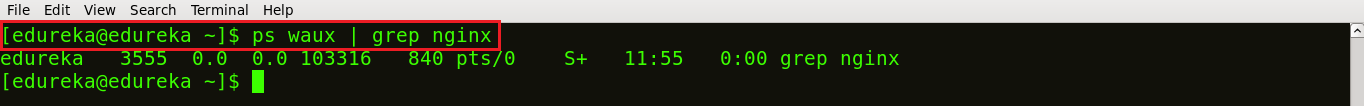
**ansible-playbook <name of your file>.yml**



We can see in the screenshot above that our task is getting executed; Nginx being installed.

**Step 7:** Let’s check if Nginx is installed on my host machine. Use the command below:

**ps waux | grep nginx**



You can see in the screenshot above, that different process ids 3555 and 103316 are running which ensures that Nginx is running on your host machines.