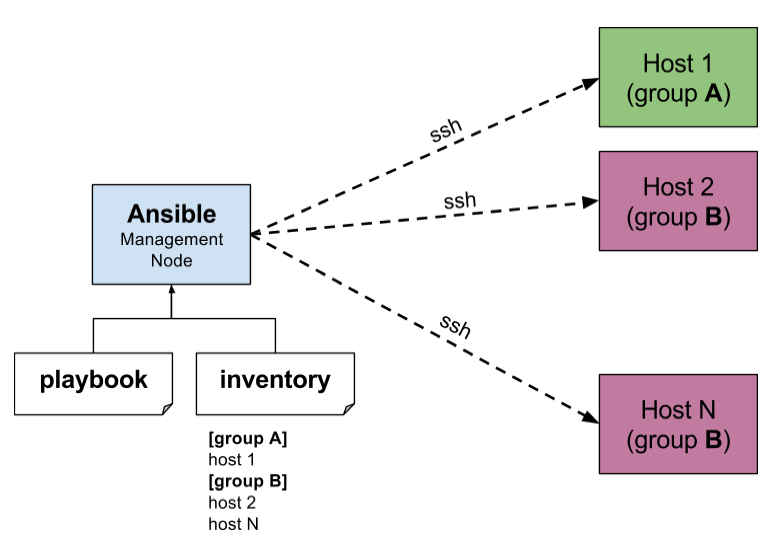
**Prerequisites to work ansible concepts**

Control node

Client or Managed node



**In this document I have launched 2 instance(ctrl & Client node) in AWS**

i) Redhat 7 -> consider this has control machine

ii) Redhat or Amazon linux with any version -> Consider this has client machine

iii) Ubuntu with any version -> Consider this has another client Machine

Ensure (22-ssh port) opened in both server’s security groups with (Anywhere IP)

**Step-1**

**In Control machine - Redhat 7**

**2 Methods to install ansible via yum installation and via source code.**

**1 - Method**

(i) Install epel using below command

yum install epel-release -y

(ii) Add latest epel repository to this machine for install ansible.

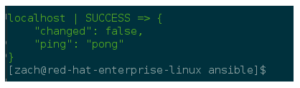
rpm -Uvh<https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm>

(iii) now install Ansible using below command

yum install ansible -y

iv) Check ansible working status, Run below command

ansible localhost -m ping



**2- Method**

## **Via Source**

If using third-party package repositories is impractical or impossible, your next best option is compiling Ansible via source. While this seems like a much more involved process, it is just as straightforward as the yum install. To begin, let’s clone the GitHub repository:

**i)**

**yum install git -y**

**cd /opt/**

**git clone git://github.com/ansible/ansible.git --recursive  
cd ./ansible  
git submodule update --init --recursive**

**ii)**

Once the Ansible repository is finished downloading, we need to install some dependencies. First, we need to install Python via yum:

**sudo yum install python**

**iii)**

Next, we need to install a few required Python modules:

**sudo easy\_install pip**

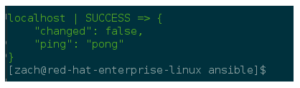
**iv)**

And finally, we simply need to install Ansible to our system:

**sudo make install**

Now, like before, simply run the ping command to test our newly installed Ansible!

**ansible localhost -m ping**



**STEP -2**

i) In client Machine -> Ensure installed Openssh, port 22 enabled,

If not installed openssh run this command -> **yum install openssh -y**

ii) My Client Machine is in AWS, So We should place that client machine Pem file in control machine with 600 permission

For example my client machine pem file is **kiruba.pem**

So I need to download that pem file & placed in this path **/etc/ansible/kiruba.pem**

**By default this pem file in execute permission, so u should change permission by using below command**

**chmod 600 /etc/ansible/kiruba.pem**

**STEP -3**

**cd /etc/ansible**

Place the client ssh(remote) file information to below inventory host file using below command.

I have created one devserver group and added my client private IP, username, port and pem file path, if your client machine in local, instead of pem file, u just give “ansible\_ssh\_password=password”

**vim /etc/ansible/hosts**

**[dashboard:vars]**

ansible\_ssh\_private\_key\_file=./kiruba.pem

**[dashboard]**

172.31.30.217 ansible\_ssh\_port=22 ansible\_ssh\_user=ec2-user

172.31.30.214 ansible\_ssh\_port=22 ansible\_ssh\_user=ec2-user

**(or)**

**[devservers]**

172.31.30.217 ansible\_ssh\_port=22 ansible\_ssh\_user=ec2-user ansible\_ssh\_private\_key\_file=./kiruba.pem

**or**

**[stagingservers]**

172.31.30.[100-110] ansible\_ssh\_port=22 ansible\_ssh\_user=ec2-user ansible\_ssh\_private\_key\_file=./kiruba.pem

or

**[webservers]**

www-01.example.com ansible\_ssh\_port=22 ansible\_ssh\_user=ec2-user ansible\_ssh\_pass=password

www-02.example.com ansible\_ssh\_port=22 ansible\_ssh\_user=ec2-user

or

**[webservers1]**

www\*.example.com ansible\_ssh\_port=22 ansible\_ssh\_user=ec2-user

**Step-3 - Command line**

i) We have to verify the client node connectivity

ansible devservers -s -m ping (**or)** ansible -m ping all

**Note :** devservers = group, -s= sudo, -m=module

**Explanation :** the above cmd will check connectivity to devservers group only. Ping all means all group

**ii) get full client machine information using below command**

ansible devservers -s -m setup

**iii) U can also filter particular information using below example**

ansible devservers -s -m setup -a ‘filter=ansible\_os\_family’

ansible devservers -s -m setup | grep distribution

**iv) Install apache package through command line using shell Module :**

ansible devservers -s -m shell -a 'yum install httpd -y'

ansible devservers -s -m shell -a 'service httpd start'

ansible devservers -s -m shell -a 'yum remove httpd -y'

**PLAYBOOK CREATION**

**YAML SYNTAX**

• The first line of a playbook should begin with "--- " (three hyphens) which indicates the beginning of the YAML document.

• Lists in YAML are represented with a hyphen followed by a white space. A playbook contains a list of plays; they are represented with "- ". Each play is an associative array, a dictionary, or a map in terms of key-value pairs.

• Indentations are important. All members of a list should be at the same indentation level.

• Each play can contain key-value pairs separated by ":" to denote hosts, variables, roles, tasks, and so on.

**Step-4**

**How to create the playbook**

**Below is sample playbook for installing apache package to redhat client machine.**

**Note: if client machine is Redhat/Amazon linux/centos -----> Use (yum) module**

**If Client machine is Ubuntu -----> use (apt) module.**

cd /etc/ansible

vim first-playbook.yml

---

- hosts: devservers

connection: ssh

gather\_facts: no

tasks:

- name: Install Apache package

**yum:** pkg=httpd state=present update\_cache=true

**Execute the ansible through playbook**

ansible-playbook -s first-playbook.yml

**Note: there are different state are there**

**latest :** Install with updated package

**absent:** Remove particular Package

**present:** Install present or available package

**installed:** Install available or present package

**removed:** remove particular package

**Other yum module Examples:**

- name: **install the latest version of Apache**  
 yum: name=httpd state=latest  
  
- name: **remove the Apache package**  
 yum: name=httpd state=absent  
  
- name: **install the latest version of Apache from the testing repo**  
 yum: name=httpd enablerepo=testing state=present  
  
- name: **install one specific version of Apache** yum: name= httpd-2.2.29-1.4.amzn1 state=present  
  
- name: **upgrade all packages** yum: name= '\*' state=latest  
  
- name: **upgrade all packages, excluding kernel & foo related packages** yum: name= '\*' state=latest exclude=kernel\*,foo\*  
  
- name: **install the nginx rpm from a remote repo**  
 yum: name= http://nginx.org/packages/centos/6/noarch/RPMS/nginx-release-centos-6-0.el6.ngx.noarch.rpm  
 state=present  
  
- name: **install nginx rpm from a local file**  
 yum: name= /usr/local/src/nginx-release-centos-6-0.el6.ngx.noarch.rpm state=present  
  
- name: **List ansible packages and register result to print with debug later.** yum: list=ansible register=result

**Variable Section:**

**In this example, I have used variable inside the playbook and outside the playbook.**

**vim vars.yml**

pkg2: telnet

pkg3: php

**vim first-playbook.yml**

---

- hosts: apache

connection: ssh

gather\_facts: no

vars:

pkg: httpd

vars\_files:

- vars.yml

tasks:

- name: Install apache package to host

yum: pkg={{ pkg }} state=installed update\_cache=true

- name: Install package through variable and variable file

yum: pkg={{ pkg2 }} state=installed update\_cache=true

- name: Install another package through variable file

yum: pkg={{ pkg3 }} state=installed update\_cache=true

**Using (with\_items) might be a good idea**

When you use the **with\_items** clause, Ansible will create a variable called **{{item}}** containing the value for the current iteration. Some modules handle collections of items really well and are actually faster than running the same task multiple times with different parameters.

**# Installing all packages with one task (faster) - Just use below example in task section**

- name: install required packages using the yum module

yum: pkg={{ item }} update\_cache=yes

with\_items:

- git

- memcached

- nginx

- httpd

**Example 9: Handler & Notify Section:**

vim first-playbook.yml

---

- hosts: apache

connection: ssh

gather\_facts: no

tasks:

- name: Install apache package to host

yum: pkg=httpd state=installed update\_cache=true

Notify: Restart HTTPD

handlers:

- name: Restart HTTPD

service: name=httpd state=started ( started, stopped, restarted)

**Example 10: Conditional sections**

**Note : some Package name will be differ in linux flavours redhat linux & ubuntu linux**

**For example : If we need to install webserver**

**In redhat → package name is (httpd)**

**In Ubuntu → package name is (apache2)**

**Below is sample playbook for install webserver to different linux flavour in single playbook using condition**

**Conditionals Section:**

vim first-playbook.yml

---

- hosts: apache

connection: ssh

gather\_facts: yes

vars:

centos-pkg: httpd

ubuntu-pkg: apache2

tasks:

- name: Install apache package to host

yum: pkg={{ centos-pkg }} state=installed update\_cache=true

when: ansible\_os\_family = “RedHat”

- name: Install apache package to host

apt: pkg={{ ubuntu-pkg }} state=installed update\_cache=true

when: ansible\_os\_family = “Debian”

**Example 11**

**You can run things in “Dry Run” Mode**

Ansible will not make any changes to your host, but simply report what changes would have been made if the playbook was run without this flag.

**ansible-playbook -s first-playbook.yml --check**

**Example 12**

**Until Section:**

tasks:

- name: Install apache webserver

yum: pkg=httpd state=latest

- name: Verify service status

shell: service httpd status

register: status\_result

until: status\_result.stdout.find ( “running” ) != -1

retries: 5

delay: 5

- debug: var=status\_result

**Example 13**

**Prompt - Interactive playbook**

---

- hosts: apache

connection: ssh

gather\_facts: yes

vars\_prompt:

- name: centospkg

prompt: Install which package ?

default: telnet

tasks:

- name: Install apache package to host

yum: pkg={{ centospkg }} state=installed update\_cache=true

**o/p > it will ask -> Install which Package-> it will install provided package, otherwise install telnet by default**

**Example 14**

**Pass variable in command line**

---

- hosts: {{ apache }}

connection: ssh

gather\_facts: yes

tasks:

- name: Install apache package to host

yum: pkg={{ centos-pkg }} state=installed update\_cache=true

**ansible-playbook -s first-playbook.yml --extra-vars “ hosts=apache centos-pkg=httpd ”**

**Explained Important Modules**

**setup module:** shows Client node informations

ansible apache -s -m setup

**copy module:** Copy required files to client machine, we need to mention source and destination path

- name : copy the files to remote server

copy: src=files/index.html dest=/var/www/html/ owner=apache group=apache

**wait\_for module:**

- name: waiting for port 80 to listen

wait\_for: port=80 state=started

**yum module:** If centos/redhat use this module for install packages

- name: install apache webserver in centos

yum: pkg=httpd state=installed

**apt module:** If Ubuntu or other debian Os use this module for install packages

- name: install apache webserver in centos

apt: pkg=apache2 state=installed

**service module:** If u need to start or stop or restart particular service in client machine, use this module

- name: Restart HTTPD

service: name=httpd state=started

**cron module:**

- name: add a cron job to the node

cron: name=”list files” minute=”0” hour=”1” job=”sh /opt/script.sh”

**debug module:**

tasks:

- name: Install apache webserver

yum: pkg=httpd state=latest

- name: Verify service status

shell: service httpd status

register: status\_result

- debug: var=status\_result

**user module:** For create user in client machine

tasks:

- name: create the user call tomcat

user: name=tomcat shell=/bin/bash

**script module:** For run any script, use this module

- script: files/command.sh

**raw module:** To findout date in client machines

- tasks:

- name: find the date in hosts

raw: /usr/bin/uptime

**ping module:**

- tasks:

- name: ping all the hosts

ping:

**unarchive module:**

- name: copy and unarchive a file

unarchive: src=files/test.tar.tz dest=/var/www/html/

**get\_url module:**

- name: download the files from urls

get\_url: url=https://url dest=/var/www/html

**group module:**

tasks:

- name: create the user call tomcat

group: name=tomcat shell=/bin/bash

**mail module:**

tasks:

- name: send a email

mail:

host: “localhost”

port: “25”

to: “test@example.com”

subject: “Host is finished”

body: “system called {{ ansible\_hostname }} has been successful”

**git module:**

tasks:

-name: check out the git repo to remote host

git: repo=ssh://giturl dest=/var/www/html

**template module: Same like copy module, but its for customized configuration**

tasks:

- name: update the conf file to host node

template: src=httpd.conf.j2 dest=/etc/httpd/http.conf owner=root group=root

# 

**File Module :**

Ensure a particular line is in a file, or replace an existing line using a back-referenced regular expression

tasks:

- name: update the conf file to host node

**lineinfile:** path=/etc/httpd/conf/httpd.conf regexp: '^Listen=' line: 'Listen 8080'

**Task execution order**

tasks:

- name: Task-1

yum: pkg=httpd state=installed update\_cache=true

- name: Task-2

yum: pkg=telnet state=installed update\_cache=true

- name: Task-3

yum: pkg=telnet state=installed update\_cache=true

**ansible-playbook first-playbook.yml -- start-at- task=’Task-1’**

**ansible-playbook first-playbook.yml -- step**

**ANSIBLE VAULT**

Use **ansible-vault** when you want to store sensitive information

If one of your tasks requires sensitive information (let’s say the database user and password), it’s a good practice to keep this information encrypted, instead of storing it in plain text.

Ansible ships with a command line tool called ansible-vault, that allows you to create and manage encrypted files. This way you can commit the encrypted file to your source control and only users with the decryption password will be able to read it.

**# Encrypt an existing file. You'll need to create an encryption password.**

ansible-vault encrypt secrets.yml

**# Creates a new, encrypted file. You'll need to create an encryption password.**

ansible-vault create secrets.yml

**# Decrypt a file. You'll have to enter password used for encryption.**

**# Use it with caution! Don't leave your files encrypted.**

ansible-vault decrypt secrets.yml

**# Edit an encrypted file (uses vim by default, can be overridden by the environment variable $EDITOR)**

ansible-vault edit secrets.yml

**# Print the contents of the encrypted file**

ansible-vault edit secrets.yml

If you import the vars\_file **secrets.yml** in your playbook, Ansible will fail, as it will not know how to read the encrypted file. You’ll have to specify the command line argument **--ask-vault-pass,** which will make Ansible prompt you the password of the encrypted file.

**ansible-playbook playbook.yml --ask-vault-password**

Another way is to store the password in a file (which should not be commited) and specify the path to the file using the **--vault-password-file** argument. If this file is marked as executable, Ansible will run it and use the output as the password.

# **Tags**

If you have a large playbook it may become useful to be able to run a specific part of the configuration without running the whole playbook.

Both plays and tasks support a “tags:” attribute for this reason. You can **ONLY** filter tasks based on tags from the command line with --tags or --skip-tags. Adding “tags:” in any part of a play (including roles) adds those tags to the contained tasks.

Example:

tasks:

- name: Install apache package to host

yum: pkg=httpd state=installed update\_cache=true

**tags:**

**- httpd**

- name: Install telnet package to host

yum: pkg=telnet state=installed update\_cache=true

**tags:**

**- telnet**

**ansible-playbook -s firstplaybook.yml --tags “httpd”** -> it executes only httpd package installation.

**ansible-playbook apache -s firstplaybook.yml --skip-tags “telnet” ->** its skip telnet section and install only http section.

**Role:**

Roles are a further level of abstraction that can be useful for organizing playbooks. As you add more and more functionality and flexibility to your playbooks, they can become unwieldy and difficult to maintain as a single file. Roles allow you to create very minimal playbooks that then look to a directory structure to determine the actual configuration steps they need to perform.

**1 - Directory Structure:**

cd /etc/ansible/role/ or

# mkdir role

# cd role

# mkdir webserver

# cd webserver

# mkdir files handlers meta templates tasks vars

These are the directories that will contain all of the code to implement our configuration. You may not use all of the directories, so in real practice, you may not need to create all of these directories.

**This is what they are all for:**

**files:** This directory contains regular files that need to be transferred to the hosts you are configuring for this role. This may also include script files to run.

**handlers:** All handlers that were in your playbook previously can now be added into this directory.

**meta:** This directory can contain files that establish role dependencies. You can list roles that must be applied before the current role can work correctly.

**templates:** You can place all files that use variables to substitute information during creation in this directory.

**tasks:** This directory contains all of the tasks that would normally be in a playbook. These can reference files and templates contained in their respective directories without using a path.

**vars:** Variables for the roles can be specified in this directory and used in your configuration files.

**2 - Role Based Tasks:**

role/

# ls

webserver

# vim webserver.yml

---

- hosts: apache

connection: ssh

gather\_facts: yes

roles:

- webserver

# cd webserver

# ls

files

handlers

meta

templates

tasks

vars

# cd tasks

vim main.yml

- name: Install centos apache web server

yum: pkg=httpd state=latest

ansible-playbook -s webserver.yml

**3 - Task Order - Pre and Post Task**

# vim webserver.yml

---

- hosts: apache

connection: ssh

gather\_facts: yes

pre\_tasks:

- name: When did the role start

raw: date

roles:

- webserverl

post\_tasks:

- name: When did the role end

raw: date

**4 - Conditional execution:**

**webservers → tasks → main.yml**

- name: Install centos apache web server

yum: pkg=httpd state=latest

when: “ ansible\_os\_family == ‘RedHat’ “

- name: Install centos apache web server

yum: pkg=apache2 state=latest

when: “ ansible\_os\_family == ‘Debian’ ”

**5 - Variable Substitution:**

**webservers → vars → main.yml**

**redhat\_apache: httpd**

**debian\_apache: apache2**

**webservers → tasks → main.yml**

- name: Install centos apache web server

yum: pkg={{ redhat\_apache }} state=latest

when: “ ansible\_os\_family == ‘RedHat’ “

- name: Install centos apache web server

yum: pkg={{ debian\_apache }} state=latest

when: “ ansible\_os\_family == ‘Debian’ ”

**6 - Handlers & Notify**

**webservers → handlers → main.yml**

- name: Restart HTTPD

service: pkg={{ redhat\_apache }} state=restarted

when: “ ansible\_os\_family == ‘RedHat’ “

- name: Restart Apache2

service: pkg={{ debian\_apache }} state=restarted

when: “ ansible\_os\_family == ‘Debian’ ”

**webservers → tasks → main.yml**

- name: Install centos apache web server

yum: pkg={{ redhat\_apache }} state=latest

when: “ ansible\_os\_family == ‘RedHat’ “

Notify: Restart HTTPD

- name: Install centos apache web server

yum: pkg={{ debian\_apache }} state=latest

when: “ ansible\_os\_family == ‘Debian’ ”

notify: Restart Apache2

**7 - Configure alternative role paths:**

We have to configure the roles path in ansible.cfg and place the webservice.yml file to anywhere in server, then run the playbook

It’ll automatically take the role path from ansible.cfg file

**8 - Conditional Include Statement:**

**We have three roles “ webserver, redhat\_webservers, debian\_webservers”. Include redhat and debian role to main playbook**

# vim webserver.yml

---

- hosts: apache

connection: ssh

gather\_facts: yes

pre\_tasks:

- name: When did the role start

raw: date

roles:

- { role: redhat\_webservers, when: “ ansible\_os\_family == ‘RedHat’ ”}

- { role: debian\_webservers, when: “ ansible\_os\_family == ‘Debian’ ”}

post\_tasks:

- name: When did the role end

raw: date

**9 - Waiting for events**

**webservers → tasks → main.yml**

- name: Install centos apache web server

yum: pkg={{ redhat\_apache }} state=latest

when: “ ansible\_os\_family == ‘RedHat’ “

- name: waiting for port 80

wait\_for:

port: 80

state: started

- name: Install centos apache web server

yum: pkg={{ debian\_apache }} state=latest

when: “ ansible\_os\_family == ‘Debian’ ”

**10 - Execution a task until**

**webservers → tasks → main.yml**

- name: Install centos apache web server

yum: pkg={{ redhat\_apache }} state=latest

when: “ ansible\_os\_family == ‘RedHat’ “

- shell: systemctl status httpd

register: redhat\_result

until: redhat\_result.stdout.find ( “ active (running)” ) != -1

retries: 5

delay: 5

- name: waiting for port 80

wait\_for:

port: 80

state: started

- name: Install centos apache web server

yum: pkg={{ debian\_apache }} state=latest

when: “ ansible\_os\_family == ‘Debian’ ”

**11 - Using Tags**

**webservers → tasks → main.yml**

- name: Install centos apache web server

yum: pkg={{ redhat\_apache }} state=latest

when: “ ansible\_os\_family == ‘RedHat’ “

tags: Install\_List

- shell: systemctl status httpd

register: redhat\_result

until: redhat\_result.stdout.find ( “ active (running)” ) != -1

retries: 5

delay: 5

- name: waiting for port 80

wait\_for:

port: 80

state: started

- name: Install centos apache web server

yum: pkg={{ debian\_apache }} state=latest

when: “ ansible\_os\_family == ‘Debian’ ”

**12- Passing Variable from command line**

# vim webserver.yml

---

- hosts: {{ host }}

connection: ssh

gather\_facts: yes

pre\_tasks:

- name: When did the role start

raw: date

**roles:**

**- webserver**

post\_tasks:

- name: When did the role end

raw: date

ansible-playbook -s webserver.yml --extra-vars “ hosts=apache ”

**13 - Using Jinja2 Template**

* **templates: You can place all files that use variables to substitute information during creation in this directory.**

**webservers → tasks → main.yml**

- name: Install centos apache web server

yum: pkg={{ redhat\_apache }} state=latest

when: “ ansible\_os\_family == ‘RedHat’ “

- shell: systemctl status httpd

register: redhat\_result

until: redhat\_result.stdout.find ( “ active (running)” ) != -1

retries: 5

delay: 5

- name: waiting for port 80

wait\_for:

port: 80

state: started

**- name: customized configuration file**

**template: src=httpd.conf.j2 dest=/etc/httpd/conf owner=root group=root**

Place the httpd.conf.j2 template to template directory.

**14 - Task Order - Pre and Post Task**

Call multiple role to single playbook

# vim webserver.yml

---

- hosts: apache

connection: ssh

gather\_facts: yes

pre\_tasks:

- name: When did the role start

raw: date

roles:

- java

- tomcat

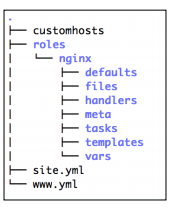
- webserver

post\_tasks:

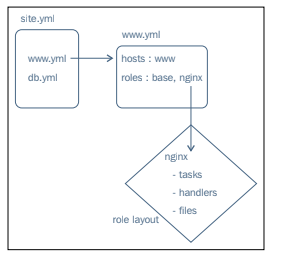
- name: When did the role end

raw: date

**Another Example diagram for better understanding to host full web application in single playbook (site.yml) -> this playbook will install and configure web server, application server, DB Server etc using roles.**

****

**nginx  
├── files  
├── handlers  
│ └── main.yml  
├── meta  
│ └── main.yml  
├── tasks  
│ └── main.yml  
├── templates  
└── vars  
└── main.yml**



Here is our **site.yml** file:

---

# site.yml : This is a sitewide playbook

- include: www.yml

---

#www.yml : playbook for web servers

- hosts: www

remote\_user: vagrant

sudo: yes

roles:

- nginx

Now u need to create folders like (files,handlers etc) under nginx role and create main.yml in every folder and run site.yml file.