**Ansible Documentation**

**Ansible Setup:**

**Installation Way1:**

* First we need to install python in linux machine because ansible written in python language.
* Then we need to go ‘**/opt’** and needs to create directory called ansible it would be path like **‘/opt/ansible’** then we need to go path called **‘/opt/ansible’.**
* Then we need to go <http://releases.ansible.com/ansible/> URL and download tar.gz file versions depends on our choice. In windows you can download directly then you need to copy it into linux machine with ‘winscp’ or ‘Filezilla’.
* You can also download in linux directly by following link:

**wget** [**http://releases.ansible.com/ansible/ansible-2.3.1.0.tar.gz**](http://releases.ansible.com/ansible/ansible-2.3.1.0.tar.gz)

* Then you need to untar by using command called **‘ tar -xvzf ansible-2.3.1.0.tar.gz’** then we will get **ansible** directory from tar file.
* After extracting tar file you will see **‘ansible’** it will be available in where we are extracted. In our practice it should be **‘/opt/ansible’** directory then total path **‘/opt/ansible/ansible’.**
* Then go to directory **‘/opt/ansible/ansile’** then you need to set owner for **‘/opt/ansible’** like following.

**# chown -R root:root /opt/ansible**

**# source ./hacking/env-setup**

* If you give owner as normal user like # **chown -R testuser:testuser /opt/ansible**  then you need to switch into that user ‘**# su testuser’** then execute following command

**# source ./hacking/env-setup**

Then it should be installed successfully without rpm packages. Now you can check version of ansible use following command

**# ansible --version**

* Then you need to create directory called **‘ansible’** in **‘/etc’** the path should be

**‘/etc/ansible’.** Then go to this directory and create file called **‘hosts’** then the path should be **‘/etc/ansible/hosts’**

* Then you need to add client systems in **‘/etc/ansible/hosts file** like following

# **export** ANSIBLE\_HOSTS=**/**etc**/**ansible**/**hosts

**# vi /etc/ansible/hosts**

[Web\_Servers]

192.168.0.1

192.168.0.2

[DB\_Servers]

172.168.0.1

172.168.0.2

The above process we have installed ansible with tar file we have downloaded it from ansible official document. There are another two ways to install ansible like following.

**Installation way2:**

* We can install ansible by using GitHub repository for that we can use following process
* Install **python** in your linux machine which you are going to put it as ansible-control server
* Then you need to create directory called **‘ansible’** in **‘/opt’ then** path would called as **‘/opt/ansible’**
* Then go to directory called **‘/opt/ansible’**
* Then we need to clone ansible from **‘GitHub’** repository using command called as below

**# git clone git://github.com/ansible/ansible.git**

* Then you will see **‘ansible’** in **‘/opt/ansible’**
* Then go to **‘ansible’** directory in **‘/opt/ansible’**
* Then give owner permission to **‘/opt/ansible’ directory** for **root** or any **normal user** like following

**# chown -R root:root /opt/ansible**

* Then execute following command to install ansible

**# source ./hacking/env-setup**

* Then you can check wether the ansible is installed successfully or not by using command called **# ansible --version**
* Then you need to setup the hosts file for ansible using following commands

**# mkdir /etc/ansible**

**# export** ANSIBLE\_HOSTS=**/**etc**/**ansible**/**hosts

**# vi /etc/ansible/hosts**

In this file you need to add client systems like following to distribute the servers

[Web\_Servers]

192.168.0.1

192.168.0.2

[DB\_Servers]

172.168.0.1

172.168.0.2

Then you can start you playbooks here.

**Installation way3 using yum or rpm:**

* You can follow below steps to to install ansible using **‘yum or rpm’**

**# sudo yum install epel-release -y**

**# yum update –y**

**# yum install ansible -y**

**# ansible –version**

**# export** ANSIBLE\_HOSTS=**/**etc**/**ansible**/**hosts

**# vi /etc/ansible/hosts**

Adding client systems inot ansible hosts file like following

[Web\_Servers]

192.168.0.1

192.168.0.2

[DB\_Servers]

172.168.0.1

172.168.0.2

🡪 After giving entries in hosts file you can wether the hosts are added in hosts file successfully or not using following

**# ansible -m ping all (**To check all the hosts are a live in hots file or not**)**

# **ansible -m ping web\_servers (**To check webservers are added properly or not**)**

As of now have settedup the ansible then we need to write playbooks, roles.

**Adhoc-Commands:**

To get modules list using command line use below is the command

# ansible-doc -l | more

To find list of available modules

# ansible-doc -l | wc -l

To find perticular module

# ansible-doc -s yum(module name)

To get output in single line

# ansible -m ping all -o

To get group servers pinging

# ansible -m ping dev

To find mount points in remote systems as well as to find kernel version

# ansible -m all shell -a “uname -a;df -h”

To install apache package using command line to remote hosts

# ansible -m all yum -a “name=httpd state=present”

To install apache package using command line to remote hosts for sudo users

# ansible -m all yum -a “name=httpd state=present” -s

To install apache package using command line to remote hosts

# ansible -m all service -a “name=httpd state=started”

To install apache package using command line to remote hosts for sudo users

# ansible -m all service -a “name=httpd state=started” -s

To copy files from server to remote systems

# ansible -m dev copy -a “src=/root dest=/tmp”

**Playbooks:**

Components of playboks:

1. **Hosts:**

Define target system hostname or group name which was defined in host file

Ex:

Hosts: webservers

1. **Tasks:**

Each playbook is composed of one or more 'plays' in a list. The goal of a play is to map a group of hosts to some well defined roles, represented by things **ansible** calls **tasks**. At a basic level, a **task** is nothing more than a call to an **ansible** module

Ex:

* Name: install apache

1. **Notify:**

The ‘notify’ actions are triggered at the end of each block of tasks in a play, and will only be triggered once even if notified by multiple different tasks.

For instance, multiple resources may indicate that apache needs to be restarted because they have changed a config file, but apache will only be bounced once to avoid unnecessary restarts.

**Ex:**

notify:

- restart memcached

- restart apache

1. **Handlers:**

Handlers are lists of tasks, not really any different from regular tasks, that are referenced by a globally unique name, and are notified by notifiers. If nothing notifies a handler, it will not run. Regardless of how many tasks notify a handler, it will run only once, after all of the tasks complete in a particular play.

Ex:

handlers:

- name: restart memcached

service: name=memcached state=restarted

- name: restart apache

service: name=apache state=restarted

**Modules:**

Modules are the ones that do the actual work in ansible, they are what gets executed in each playbook task. But you can also run a single one using the ‘ansible’ command.

**Types:**

**Command** – Command module allows us to execute commands on the remote machine. A simple play book for the Command module is,

# vi command.yml

---

- hosts: test

  tasks:

   - name: Check for Command ls

     command: /bin/echo Hello

**Raw** – This module is used when there is more need than Command module or if the command module does not support the operation. This module makes a SSH to the remote machine and run the command. For the Ansible to work we need to have Python available but for this module we don’t need a Python to be available. A simple playbook looks as.

# vi raw.yml

---

- hosts: test

  tasks:

   - name: Install VIM

     raw: yum -y install vim-common

**Shell –** The Shell module also allows you to execute commands on the remote machine. The major difference between the Command and the Shell modules are that the Shell module uses a shell (/bin/bash) to run commands. We can get all the shell variables and use other shell features. A simple play book includes,

# vim shell.yml

---

- hosts: test

  tasks:

   - name: create File

     shell: /bin/touch /tmp/hai

     register: out

   - debug: var="{{ out.stdout }}"

**File –** There are certain modules available in Ansible that helps in managing files, directories etc. One such module is File. The file module allows you to change the attributes of a file. It can touch a file, create or delete recursive directories, and create or delete symlinks.

A simple playbook includes setting a file /tmp/hai with 644 permissions and creating a directory in tmr in /tmp

# vi test.yml

---

- hosts: test

  tasks:

   - name: File Permissions

     file: dest=/tmp/hai mode=0644

   - name: Create Directory

     file: dest=/tmp/tmr state=directory

**Copy –** Another module in managing file is Copy module. A Copy module allows in copying file to the remote location. A Simple example yml file is

# vi copy.yml

---

- hosts: test

  tasks:

   - name: Copy File

     copy: src=./shell.yml dest=/tmp

Copies a file shell.yml to the /tmp location on remote machine.

**Template –** Template module is a very important module provided by Ansible. Templating is nothing but creating file on the machine. Ansible uses Jinja2 templating language for creating templates based on the Django Framework.

Create a test file as

# vi test.yml

This is a test file in {{ ansible\_hostname }}

Now let’s copy the test file to the remote system using the below playbook as,

# vi template.yml

---

- hosts: test

  tasks:

   - name: Create a Test Template

     template: src=test dest=/tmp/testFile mode=0644

Once we execute the playbook we can see that the file is not just moved but the variable “Ansible\_hostname” is replaced with the Hostname variable.

Another option provided by template is the validate parameter that allows you to run a command to validate the file before copying it. This is like a hook that Ansible provides to make sure files that might break the service are not written.

A simple playbook for copying a conf file to the HTTPD location and validating it before copying that look as,

# vi template1.yml

---

- hosts: test

  tasks:

   - name: Create a Virtaul Host and Validate

     template: src=test.conf dest=/etc/httpd/conf.d/test.conf validate="httpd -t %s"

Upon executing the Playbiook it throws an error as Validation failed.

**Creating files using ansible playbooks into nodes:**

Before creating playbook we need to open file filename.yml format then you can write your YAML script to automate the servers.

# vi test.yml

---

- hosts: web\_servers

tasks:

- name: Creating The File

file: path=/root/demo state=touch

- name: Creating The Directory

file: path=/tmp/test state=directory

**Creating the files with permissions:**

# vi sample.yml

- file:

path: /etc/httpd.conf

owner: root

group: root

mode: 0644

- file:

src: /file/to/link/to

dest: /path/to/symlink

owner: root

group: root

state: link

- file:

src: '/tmp/**{{** item.src **}}**'

dest: '**{{** item.dest **}}**'

state: link

with\_items:

- { src: 'x', dest: 'y' }

- { src: 'z', dest: 'k' }

*# touch a file, using symbolic modes to set the permissions (equivalent to 0644)*

- file:

path: /etc/test.conf

state: touch

mode: "u=rw,g=r,o=r"

*# touch the same file, but add/remove some permissions*

- file:

path: /etc/test.conf

state: touch

mode: "u+rw,g-wx,o-rwx"

*# create a directory if it doesn't exist*

- file:

path: /etc/test

state: directory

mode: 0755

# vi httpd.yml

---

- hosts: dev

  user: root

  sudo: no

  tasks:

   - name: Install Apache

     yum: pkg=httpd state=latest

     notify:

     - restart apache

   - name: ensure apache is running

     service: name=httpd state=started

  handlers:

   - name: restart apache

     service: name=httpd state=restarted

# vi apache.yml

---

* hosts: test

tasks:

   - name: Install Apache

     yum: pkg=httpd state=latest

     notify:

     - restart apache

   - name: ensure apache is running

     service: name=httpd state=started

  handlers:

   - name: restart apache

     service: name=httpd state=restarted

**Installing httpd package in client systems:**

---

- hosts: web\_servers

vars:

http\_port: 80

max\_clients: 200

remote\_user: root

tasks:

- name: ensure apache is at the latest version

yum: name=httpd state=latest

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

**Conditional Statements:**

Ansible provides conditional statements that help run a task only when a specified condition is met.Ansible provide a **WHEN** condition to execute tasks based on certain conditions. Lets see a basic example of usin When condition

**Playbook using when condition:**

# vi with.yml

---

- hosts: test

  tasks:

    - name: install httpd Package

      yum: name=httpd state=latest

      sudo: yes

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

    - name: install apache2 package

      apt: name=apache2 state=latest

      sudo: yes

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

**Playbook is with\_item:**

# vi with\_item.yml

---

- hosts: webserver

tasks:

- name: install httpd

yum: name=httpd state=installed

- name: install php and libs

yum: name={{ item }} state=installed

with\_items:

- php-xml

- php

- php-zip

- php-mbstring

- php-mysqlnd

- php-gd

# vi loop.yml

---

- hosts: all

tasks:

# emit a debug message containing the content of each file.

- debug:

msg: "**{{** item **}}**"

with\_file:

- first\_example\_file

- second\_example\_file

**Roles:**

**Components of Roles:**

1. defaults = Data about the role / application. Default variables
2. files = Put the static files here. Files will be then copied on remote machines
3. handlers = Tasks which are based some actions. Triggers.

Example: in case my httpd.conf changes, it should trigger service restart

1. meta = Information about role. Author, supported platforms etc Dependencies , if any.
2. Tasks = Core logic or code. Installing package, copying files etc
3. Templates = Similar to files except that templates support dynamic files. Jinja2 template language.
4. Vars = Both vars and defaults stores variable. Variables stored under “vars” has got higher priority and difficult to override.

* First we need to go to the /etc/ansible directory and create roles directory using ‘mkdir roles’.
* Then go to roles directory and create roles using following command

# ansible-galaxy init rolename(ex:apache)

* Then check wether role is created successfully or not using following command

# tree rolesname

* Then you will see lot of directories in that. Here every directory having main.yml file.
* Then go to tasks directory and write code like following here we are going to deploy apache server.

# cd tasks

# vi main.yml

# tasks file for apache

* Include: install.yml
* Include: configure.yml
* Include: service.yml

Then save and exit:wq!

Then create install.yml

# vi install.yml

# installing httpd package

* name: install apache

yum:

name: httpd

state: latest

Then save and exit

:wq!

tasks # ls

You will see two files here main.yml , install.yml

Then we need to create configure.yml as we declared in maiol.yml in tasks

# vi configure.yml

---

* name: httpd.conf file

copy: src=httpd.conf dest=/etc/httpd/conf/httpd.conf

notify:

* restart apache service
* name: deploying index.html file

copy: src=index.html dest=/var/www/html/index.html

Then save and exit check the files are having 3 in tasks directory.

Then create another file as per tasks in main file as we defined.

# vi service.yml

---

* name: starting apache service

service: name=httpd state=started

Then save and exit. Then check wether these files are created or not using ls command in tasks directory.

Then go back into files directory in apache role and copy the file from ‘/etc/httpd/conf/httpd.conf to **/etc/ansible/roles/files**

**# cp -R /etc/httpd/conf/httpd.conf /etc/ansible/roles/files**

Then go to httpd.conf file using vim httpd.conf

Then give entry like ‘# COMING FROM ANSIBLE APACHE ROLE’

Then save and exit.

Then we need to write index.html file in files directory

# vi index.html

<html><body><h1>Welcome to Ansible Roles</h1></body></html>

:wq!

Then go back to handlers directory in apache role then we need to configure the handlers

Open main.yml file write like following

# vi main.yml

---

* name: restart apache service

service: name=httpd state=restarted

:wq!

In tasks directory there is a file configure.yml in this notify is there in notify you have mentioned restart apache service it should be same with files directory of main file name.

Then go to meta directory and do some modifications in that main.yml file like:

Author, platforms, company name like that.

Then go back to ‘/etc/ansible’ directory and write ‘site.yml’ file give entries like following

---

* hosts: all

roles:

- apache

If you have more roles you can write like

- httpd

- ntpd like this you can write the above site.yml file is main file like site.pp in puppet.

Then save and exit.

Then check sysntax of site.yml using below command

# ansible-playbook site.yml –syntax-check

Then push playbook site.yml file

# ansible-playbook site.yml

Then check client machine wether the apache server is properly setted up or not using roles

Then you have successfully configured apache server in client machines.

**To find nodes information using System Modules:**

# Display only facts regarding memory found by ansible on all hosts and output them.

# ansible all -m setup -a 'filter=ansible\_\*\_mb'

To find the particular system information using following command

# ansible -m setup -a hostname/ip of node 'filter=ansible\_\*\_mb'

# Display only facts returned by facter.

# ansible all -m setup -a 'filter=facter\_\*'

# Display only facts about certain interfaces.

# ansible all -m setup -a 'filter=ansible\_eth[0-2]'

# Restrict additional gathered facts to network and virtual.

# ansible all -m setup -a 'gather\_subset=network,virtual'

# Do not call puppet facter or ohai even if present.

# ansible all -m setup -a 'gather\_subset=!facter,!ohai'

# Only collect the minimum amount of facts:

# ansible all -m setup -a 'gather\_subset=!all'

# Display facts from Windows hosts with custom facts stored in C(C:\custom\_facts).

# ansible windows -m setup -a "fact\_path='c:\custom\_facts'"

# ansible devservers -m setup (to get full system information)

**To get System information using playbooks:**

---

- hosts: all

remote\_user: root

tasks:

- debug: var=ansible\_memory\_mb

- debug: msg="total RAM is {{ ansible\_memory\_mb.real.total }}"

- fail: msg="not enough RAM"

when: ansible\_memory\_mb.real.total < 4096

**Ansible Vault:**

Ansible vault is tool provided by ansible for encrypting or decrypting the secret information stored in the playbooks and files controlled and ansible. For example critical information such as passwords or ssh keys. With version 2.3 ansible is capable to encrypt single values which was not possible earlier.

Earlier it was only possible to encrypt the whole file. Ansible vault is also capable to encrypt/decrypt the binary files too, for example if we got file in the files directory and we use copy module to deploy the files on the remote server, it will be deployed in unencrypted form. Security check or vault password authentication will be performed when you run the playbook responsible for copying the file over. Ansible use AES256 cipher for the encryption.

**Ansible-Vault Practical:**

While encrypting user you will be asked to supply a new password, when editing/viewing/decrypting a existing file need to enter the existing password.

To create a new encrypted file

# ansible-vault create playbook.yml

To encrypted existing files

# ansible-vault encrypt existing1.yml existing2.yml

Edit the encrypted file

# ansible-valut edit playbook.yml

To update/re-key the files(new password)

# ansible-vault rekey playbook.yml existing1.yml existing2.yml

Viewing the encrypted files

# ansible-vault view playbook.yml

Decrypting the files

# ansible-vault decrypt playbook.yml

To encrypt the file

# ansible-vault encrypt playbook.yml

While execute/running playbook you will get error message like ‘decryption failed on ‘/etc/ansible/playbook.yml’ that time you need to execute like following

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

**Playbooks Using ansible-vault:**

Before this we need to create encrypted password using below command

# openssl passwd -1 -salt testuser(this one any name you can give like testuser,test,user)

then write playbook

# vi user.yml

---

* hosts: all

vars:

password: $1$test$C5gTd1RYpgKDj4WuX5yG8/

tasks:

* name: user creation using encrypted password

user: name=testuser password={{password}}