Configuration Management

This is process of configuring remote servers from one point of control.

Advantages

+++++++++++++++++++

1) Provisioning of servers

The applications that should be installed on server can be done very quickly from a single centralized location.

2) Idempotent

Configuration management tools are used to bring the server to a particular state, called as desired state. If a server already in the desired state, configuration management tools will not reconfigure that server.

Note: Cofiguration management tools cannot be used for installing OS from the scratch.

They can be used only for managing the applications on top of the OS.

COnfigutaion management tools - Ansible, chef, puppet, salt etc

++++++++++++++++++++++++++++++++++++++++++

Ansible -- It is a open source configuration management tool, created using Python.

Main machine in which anisble is installed, is called as controller.

Remote severs that Ansible configures, are called as managed nodes.

Ansible uses agent less policy for configures remote servers ie Ansible is installed only on 1 machine, and we do not require any client side software to be installed on the remote serers.

Ansible performs configuration management through password less ssh.

++++++++++++++++++++++++++++++++++

Create 4 Servers ( Ubuntu 18 )

1 is controller

3 are managed nodes

Name the instances as

Controller

Server1

Server2

Server3

Ubuntu machines default come with Python3

Ansible supports Python2

We need to downgrade the machines from python3 to Python2

Connect server1

Check the version

$ python3 --version

To Install Python2

$ sudo apt-get update

$ sudo apt-get dist-upgrade ( It will point to older apt repository where python2 is available)

$ sudo apt-get install -y python2.7 python-pip

$ sudo apt-get install python3-pip

Now check the version of python

$ python --version

+++++++++++++++++++++++++

Establish password less ssh connection

$ sudo passwd ubuntu

( lets give the password as ubuntu only )

$ sudo vim /etc/ssh/sshd\_config

change

PasswordAuthentication yes

Save and QUIT

$ sudo service ssh restart

$ exit

++++++++++++++++++

Repeat the same steps in server2 and server3

++++++++++++++++

Now, Connect to controller

Even in controller also python2 version should be available

(So, run the same commands)

$ sudo apt-get update

$ sudo apt-get dist-upgrade

$ sudo apt-get install -y python2.7 python-pip

Now check the version of python

$ python --version

+++++++++++++++++++

Now , We need to generate ssh connections

$ ssh-keygen

Now copy the key to managed nodes

$ ssh-copy-id ubuntu@172.31.0.98 ( private Ip of server1 )

$ ssh-copy-id ubuntu@172.31.1.183 ( private Ip of server2 )

$ ssh-copy-id ubuntu@172.31.14.179 ( private Ip of server3 )

+++++++++++++++++++++

Installing ansible now

Connect to controller.

$ sudo apt-get install software-properties-common

( software-properties-common , is a base package which is required to install ansible )

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

$ sudo apt-get update

$ sudo apt-get install -y ansible

+++++++++++++++++

To check ther version of ansible

$ ansible --version

++++++++++++

Write the ip address of nodes in the inventory file

$ cd /etc/ansible

$ ls

$ sudo vim hosts

insert the private ip addresss of 3 servers

save and quit

$ ls -la ( to see the list in the current machine )

$ ansible all -a 'ls -la' ( you will get the list of the files in all managed nodes )

2 Ways ansible can

1) adhoc commands

2) playbooks

adhoc commands

----------------

Important modules in ansible

1) command - This module is used for executing basic linux commands on managed nodes.

2) shell - This module is used to execute commands which involved redirection and piping and to execute shell scripts on managed nodes.

3) ping -- This module is used to check if the remote server is pingable or not.

4) user -- This module is used for user management like create user, setting password, assign home directory etc

5) copy -- This module is used to copy the files and folders from controller to managed nodes

6) fetch -- This module is used to copy files and folder from managed nodes to controller

7) file -- This module is used for creating or deleting files and folders on managed nodes.

8) stat -- Used to capture detailed information about files and folders present in managed nodes.

9) debug -- Used to display output of any module

10) apt -- Used for performing package management on managed nodes ie installing softwares / upgrading repositories etc . It works on ubuntu, debain flavours of linux.

11) yum -- similar to apt module. It works on Red hat linux, centos etc

12) git -- used to perform git version controlling on managed nodes

13) replace -- This is used to replace specific text in configuration file with some other text.

14) service -- used for starting / stoping / restarting services on managed nodes.

15) include -- Used for calling child play books from parent play book

16) uri -- useful in checking if remote url is reachable or not.

17) docker\_container -- used to execute docker commands related to container management on managed nodes

18) docker\_image -- used to execute commands related to docker images on managed nodes.

19) docker\_login -- used to login to docker hub from managed nodes.

20) setup -- used to capturing system information related to the managed nodes.

++++++++++++++++++++++++++++++++++++++++++++++

$ ansible all -i /etc/ansible/hosts -m command -a 'free'

$ ansible all -i /etc/ansible/hosts -m command -a 'touch file1'

To check the file which is created

$ ssh 172.31.2.173 ( this command will go that machine )

$ ls

$ exit ( to come back to controller )

++++++++++++++++

To install docker in all managed nodes

$ ansible all -i /etc/ansible/hosts -m shell -a 'curl -fsSL https://get.docker.com -o get-docker.sh'

$ ansible all -i /etc/ansible/hosts -m shell -a 'sh get-docker.sh'

+++++++++++++

To check docker is installed or not

$ ssh 172.31.2.173

$ docker --version

$ exit ( to come back to controller )

+++++++++++++++++

Notes:

Ansible performs remote configurations in 2 ways

1) using adhoc commands

2) using play books

Syntx of adhoc commands

$ ansible all/group\_name/ipaddress -i path\_of\_inventory\_file -m modulename -a 'arguments'

+++++++++++++++++

Ansible command module to check the memory info on all managed nodes

$ ansible all -i /etc/ansible/hosts -m command -a 'free'

++++++++++++++++++

To open the default inventory file

$ sudo vim /etc/ansible/hosts

( Observation: 3 ip address are available )

+++++++++++++++++

Now, I copy the first two IP address ( in a new notepad file )

quit the inventory file

+++++++++++++

Create my own inventory file

$ vim myinventory

go to insert mode

paste two ip address

save and quit

+++++++++++

$ ansible all -i myinventory -m command -a 'free'

Observation: free command works on only two machines

+++++++++++++

If you do not mention the inventory file, it takes default inventory file

Play books

---------------

Notes:

Adhoc commands are capable of working only on one module and one set of arguments.

When we want to perform complex configuration management activities,

adhoc commands will be difficult to manage.

In such scenarios, we use play books.

Play book is combination of plays.

Each play is designed to do some activity on the managed nodes.

These plays are created to work on single host or a group of hosts or all the hosts.

The main advantage of play books is reusability.

Play books are created using yaml files.

$ mkdir playbooks

$ cd playbooks

$ vim playbook1.yml

INSERT mode

---

- name: Install git and clone a remote repository

hosts: all

tasks:

- name: Install git

apt:

name: git

state: present

update\_cache: yes

- name: clone remote git repository

git:

repo: https://github.com/sunilkumark11/git-9am-batch.git

dest: /home/ubuntu/newgit

...

To check the syntax:

$ ansible-playbook playbook1.yml --syntax-check

( Do not use tab when creating yml file )

To run the playbook

$ ansible-playbook playbook1.yml -b

2nd example on playbook

---------------------------

Create user on all managed nodes and I want to copy passwd file.

$ vim playbook2.yml

---

- name: Create user and copy passwd file

hosts: all

tasks:

- name: User creation

user:

name: kiran

password: sunilsunil

uid: 6779

home: /home/kiran

- name: Copy password into users home dir

copy:

src: /etc/passwd

dest: /home/kiran

...

Save and quit

$

Check the syntax:

$ ansible-playbook playbook2.yml --syntax-check

To run

$ ansible-playbook playbook2.yml -b

TO check user is created in managed nodes:

$ ssh 172.31.2.173

$ vim /etc/passwd

To check if passwd file is copied to /home/kiran

$ cd /home/kiran

$ ls

$ exit

Playbook to configure tomcat8 ( earlier example )

1st uninstall tomcat

$ ansible all -m apt -a 'name=tomcat8 state=absent purge=yes' -b

$ vim playbook3.yml

---

- name: Configure tomcat8

hosts: all

tasks:

- name: Install tomcat8

apt:

name: tomcat8

state: present

- name: copy tomcat-users.xml file

copy:

src: /home/ubuntu/tomcat-users.xml

dest: /etc/tomcat8

- name: change port of tomcat from 8080 to 9090

replace:

regexp: 8080

replace: 9090

path: /etc/tomcat8/server.xml

- name: restart tomcat8

service:

name: tomcat8

state: restarted

- name: check url response of server 1

uri:

url: http://172.31.7.134:9090

- name: check url response of server 2

uri:

url: http://172.31.3.46:9090

...

$ ansible-playbook playbook3.yml --syntax-check

$ ansible-playbook playbook3.yml -b

So, we will have 1 controller 4 nodes.

In step 6 -- Add rule -- All Traffic -- Anywhere

Check the version in the new node

$ python3 --version

We need to downgrade the machines from python3 to Python2

To downgrade

$ sudo apt-get update

$ sudo apt-get dist-upgrade ( It will point to older apt repository where python2 is available)

$ sudo apt-get install -y python2.7 python-pip

Now check the version of python

$ python --version

Establish password less ssh connection

$ sudo passwd ubuntu

( lets give the password as ubuntu only )

$ sudo vim /etc/ssh/sshd\_config

change

PasswordAuthentication yes

Save and QUIT

$ sudo service ssh restart

$ exit

++++++++++++++++

Now, Connect to controller

Now , We need to generate ssh connections

$ ssh-keygen

Now copy the key to managed nodes

$ ssh-copy-id ubuntu@172.31.6.241 ( private Ip of server4 )

+++++++++++

Now, we need to add the information of managed nodes in the inventory file.

Location of inventory file /etc/ansible

$ cd /etc/ansible

$ ls

$ sudo vim hosts

insert the private ip addresss of 4th server

save and quit

$ ansible all -a 'ls -la' ( you will get the list of the files in all managed nodes )

++++++++++++++++++

We can do grouping using [groupname]

Ex:

To do grouping

$ sudo vim hosts

[webserver]

172.31.11.96

172.31.6.207

[appserver]

172.31.12.138

[dbserver]

172.31.31.161

+++++++++++++++++++

$ ansible appserver -a 'free' ( It runs on one machine 172.31.12.138)

$ ansible webserver -a 'free' ( It runs on two machines )

$ ansible all -a 'free'

+++++++++++++++++++++++

We can perform grouping on groups

$ sudo vim hosts

[webserver]

172.31.11.96

172.31.6.207

[appserver]

172.31.12.138

[dbserver]

172.31.31.161

[india:children]

webserver

dbserver

$ ansible india -a 'free'

Grouping in inventory file

----------------------------

$ sudo vim /etc/ansible/hosts

[webserver]

172.31.11.96

172.31.6.207

[appserver]

172.31.12.138

[dbserver]

172.31.31.161

[india:children]

webserver

dbserver

Host scope variables

--------------------

These variables are classified into 2 types

1) Variables to work on group of hosts

2) Variables to work on single hosts

Variables to work on group of hosts

---------------------------------------

These variables are designed to work on group of hosts.

They are definined in a folder called group\_vars

This group\_vars folder should be presnent in the same folder where all the playbooks are present.

In this group\_vars folder, we should create a file who's name is same as group\_name in Inventory file.

In this file we create variables.

Varible which works on group of hosts

$ cd ( enter)

$ cd playbooks

$ ls

Varibles which work in group of hosts are divided into two types

1) Variables which work in group of machines

2) Variables which work on one machine

Variables which work in group of machines

------------------------------------------------

playbooks$ mkdir group\_vars

Note: group\_vars folder should be present in the same location of playbook files.

$ cd group\_vars

$ vim webserver

a: Prakash

b: logiclabs

c: /home/Prakash

d: 67809

e: /bin/bash

$ cd ..

playbooks$ vim playbook8.yml

---

- name: Using host scope variables

hosts: webserver

tasks:

- name: User creation

user:

name: "{{a}}"

password: "{{b}}"

home: "{{c}}"

uid: "{{d}}"

shell: "{{e}}"

...

save and quit

TO run the playbook

$ ansible-playbook playbook8.yml -b ( It runs on two machines)

++++++++++++++++++++++++++++

Lets add few more variables

$ cd group\_vars

$ vim webserver

a: Prakash

b: durgasoft

c: /home/Prakash

d: 67809

e: /bin/bash

f: tree

g: present

h: no

save and quit

$ cd ..

Implementing loops

Notes: Modules in ansible can be executed multiple times using loops.

$ vim playbook11.yml

- name: Install software packages

hosts: webserver

tasks:

- name: Install software

apt:

name: "{{item}}"

state: present

update\_cache: no

with\_items:

- tree

- git

- default-jdk

- apache2

...

Handlers

------------

Handler is a piece of code which is executed, if some other module is executed successfully and it has made some changes.

Handlers are always executed only after all the tasks are executed.

Handlers are executed in the order that are mentioned in the handler section, and not in the order they are called in the tasks section.

Even if handler is called multiple times in the tasks section, it will be executed only once.

Requirement:

$ vim playbook14.yml

---

- name: Confugure apache2 using handlers

hosts: all

tasks:

- name: Install apache2

apt:

name: apache2

state: present

- name: Edit index.html file

copy:

content: "Logiclabs\n"

dest: /var/www/html/index.html

notify: Restart apache2

handlers:

- name: Restart apache2

service:

name: apache2

state: restarted

...

$ ansible-playbook playbook14.yml -b

Note:

As editing the index.html file is successfull, handler is executed.

If you re run the playbook, handler is not executed.

++++++++++++++++++++++

Error Handling

-------------

If any module fails in ansible,the execution of the playbook terminates over there.

When we know that certain module might fail, and still we want to continue playbook execution, we can use error handling.

The section of code which might generate an error should be given in block section.

If it generates an error, the control comes to rescue section.

Always section is executed every time, irespective of whether the block is successfull or failure.

$ vim playbook15.yml

---

- name: Error handling

hosts: all

tasks:

- block:

- name: Install apache1

apt:

name: apache1

state: present

rescue:

- name: Install apache2

apt:

name: apache2

state: present

always:

- name: Check url response

uri:

url: "{{item}}"

with\_items:

- http://172.31.7.134

- http://172.31.3.46

- http://172.31.2.140

- http://172.31.6.241

...

$ ansible-playbook playbook15.yml -b