DevOps Ansible Up and Running

Ansible is a configuration management and provisioning tool similar to Chef, Puppet and Salt stack.

Ansible Tasks are idempotent. Without a lot of extra coding, bash scripts are usually not safety run again and again. Ansible uses "Facts", which is system and environment information it gathers ("context") before running Tasks.

Ansible uses these facts to check state and see if it needs to change anything in order to get the desired outcome. This makes it safe to run Ansible Tasks against a server over and over again.

An introduction to Ansible Configuration Management:

A Brief History about configuration management system:

* CFEngine - Released 1993. Written in C

* Puppet - Released 2005 - Written in Ruby. Domain Specific Language (DSL. SSL Nightmare.)

* Chef - Released 2009 - Written in Ruby, also a DSL, more like pure Ruby

* Juju - Released 2010, Python, Very ubuntu.

* Salt - Released 2011, Python, Never got it working right

* Ansible - Released 2012, Python. Awesome

Note: You can install Ansible upon any linux flavour but here i am using REDhat 7.2

1) Installing Ansible

i) Method first: - Using RPM package or with YUM command

ii) Method Second: - Using python based PIP installer

Note: Here we are using PIP installer

Step 1: First install Pip installer you don't have in your redhat 6.4/7.2

```
[root@desktop83 ~]# yum  install python-pip
```

then Install ansible :

=================

```
[root@desktop83 ~]# pip  install  ansible
```

After Installation process this operation you can check there will be /etc/ansible directory

OR:

You can use Yum installer if you have repopath setup already.
==

For Redhat 6.4 and later

```
[root@desktop57 ~]# rpm -ivh  http://download.fedoraproject.org/pub/epel/6/x86_64/epel-release-6-8.noarch.rpm
```

```
[root@desktop57 ~]# yum   install  ansible
```

For Redhat 7.1 and Later

```
[root@desktop57 ~]# rpm -iUvh http://dl.fedoraproject.org/pub/epel/7/x86_64/e/epel-release-7-5.noarch.rpm
```

```
[root@desktop57 ~]#  cd /etc/yum.repos.d
[root@desktop57 yum.repos.d]# cat  live.repo
[aa]
baseurl=http://mirror.centos.org/centos-7/7.2.1511/os/x86_64/
gpgcheck=0

[bb]
baseurl=http://mirror.centos.org/centos-7/7.2.1511/extras/x86_64/
gpgcheck=0
```

```
[root@desktop57 ~]#  yum   install  ansible
root@ashulinux:/etc/ansible# cd  /etc/ansible/
root@ashulinux:/etc/ansible# ls
hosts
```

Step 2: Managing Servers

Ansible was designed to managed multiple servers from a single system by using SSH

Important :
Here We have three machine one is Ansible installed and other two are the targets where we want to perform operation

----

Ansible Installed machine is: **192.168.100.104**

Target1 --   **192.168.100.9**

target2 --   **192.168.100.10**

Note: Setup and and share ssh-keys from Ansible machine to target

i) Generating ssh-keys

```
 [root@hmaster ~]# ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
fe:4c:85:36:03:b8:3a:35:35:ec:70:f0:28:bb:ee:a1 root@hmaster.example.com
The key's randomart image is:
+--[ RSA 2048]----+
|      .          |
|       *         |
|    . + B        |
|     o * o .     |
|    . + S = .    |
|     + o . +     |
|    =   . .      |
|   o o   +       |
|  E.o     o      |
+-----------------+
```

ii) share keys to both the targets machine

```
[root@hmaster ~]# ssh-copy-id   192.168.100.9
[root@hmaster ~]# ssh-copy-id   192.168.100.10
```

Now go to Ansible machine and configure the hosts file

==

iii) make a backup of host file

```
root@ashulinux:~# cp  /etc/ansible/hosts    /etc/ansible/hosts.backup
```

iv) Now edit this file and specify the targets IPS

Important: File /etc/ansible/hosts also known as inventory file

```
root@ashulinux:~# vim  /etc/ansible/hosts
```

This will look like this

```
root@ashulinux:~# cat  /etc/ansible/hosts

[testing]

192.168.100.9

192.168.100.10
```

v) Running Some basic commands

sending icmp packets

a) Sending Icmp echo-request

```
root@ashulinux:~# ansible    testing   -m  ping
192.168.100.9 | success >> {
    "changed": false,
    "ping": "pong"
}

192.168.100.10 | success >> {
    "changed": false,
    "ping": "pong"
}
```

Here:

ansible :- is the command

testing :- name defined in inventory file for calling all the list server

-m :- use for specify the module name

ping : This is the name of module which simply send icmp packets to all the define servers

b) In case you have many entries in inventory file then want to send icmp packets

```
root@ashulinux:~# vim  /etc/ansible/hosts

root@ashulinux:~#

root@ashulinux:~#

root@ashulinux:~#

root@ashulinux:~# cat    /etc/ansible/hosts

[testing]

192.168.100.9

192.168.100.10




[apache]

192.168.100.11


root@ashulinux:~# ansible   all -m ping

192.168.100.10 | success >> {

   "changed": false,

   "ping": "pong"

}


192.168.100.9 | success >> {
   "changed": false,

   "ping": "pong"

}
```

```
192.168.100.11 | success >> {
    "changed": false,
    "ping": "pong"
}
```

Note:
All: for all inventory file entries

Modules:

Modules are predefined functions in ansible which are used to perform some specific task :
I am listing some names with examples of modules.

Module LIst:

a) ping

b) shell

c) command

=========

Example: testing date command

```
root@ashulinux:~# ansible  testing  -m   shell  -a  date
192.168.100.9 | success | rc=0 >>
Thu Feb 18 06:54:36 EST 2016


root@ashulinux:~# ansible  all  -m   shell  -a  date
192.168.100.10 | success | rc=0 >>
Thu Feb 18 06:54:45 EST 2016
192.168.100.9 | success | rc=0 >>
Thu Feb 18 06:54:53 EST 2016
```

Service Restart for apache web services

```
root@ashulinux:~# ansible  all  -m   shell  -a  "service  httpd  restart"

192.168.100.10 | success | rc=0 >>

192.168.100.9 | success | rc=0 >>
```

Using command modules:

```
root@ashulinux:/etc/ansible# ansible all -m command  -a  "date"

192.168.100.9 | success | rc=0 >>

Fri Feb 19 01:06:51 EST 2016


192.168.100.10 | success | rc=0 >>

Fri Feb 19 01:07:04 EST 2016
```

Note: with shell module you need to pass -a option for passing arguments

Note: you can find list of module index by clicking below given link

http://docs.ansible.com/ansible/modules_by_category.html

Step 3:-

Managing Basic PlayBook

Playbook can run multiple Tasks and provide some more advanced functionality that we would miss out on using ad-hoc commands. Let's move the above Task into a playbook.

Playbooks and Roles in Ansible all use YaML.

Creating YAML file for installing Nginx webserver

```
Note: Simple ansible Playbook
hosts – carrying hosts information
roles/ - defining what each type of server has to perform
    webservers/
        tasks/ - tasks performed on webservers
            main.yml
        handlers/ - running tasks under particular events

            main.yml

        templates/ - configuration files which can reference variables

            index.html.j2

        files/ - files to be copied to webservers

            cloud.png
```

Important:

Here hosts ==>> pointing to all hosts inside   /etc/ansible/hosts

Method for error free playbook methods according to YAML syntax:

==

i) Creating apache web server yml file

```
[root@be04c6686478 ansible]# cat  apache_final.yml
---
- hosts: webserver
  remote_user: root
  vars:
    http_port: 80
    max_client: 300
  remote_user: root
  tasks:
  - name: installing  httpd and check
    yum:
```

```
    name: httpd
    state: latest
- name: start the apache service
  service:
    name: httpd
    state: started
    enabled: yes
handlers:
- name: checking service status
  service:
    name: httpd
    state: restarted
[root@be04c6686478 ansible]#
```

<u>Important:-</u>

Here:

a) --- yaml file start with

YAML is very sensitive to white-space, and uses that to group different pieces of information together. You should use only spaces and not tabs and you must use consistent spacing for your file to be read correctly. Items at the same level of indentation are considered sibling elements.

Items that begin with a - are considered list items. Items that have the format of key: value operate as hashes or dictionaries. That's pretty much all there is to basic YAML.

ii) Run yml file

```
[root@be04c6686478 ansible]# ansible-playbook  apache_final.yml

PLAY [webserver] ***************************************************************

GATHERING FACTS ****************************************************************
ok: [192.168.100.11]

TASK: [installing  httpd and check] ********************************************
changed: [192.168.100.11]

TASK: [start the apache service] ***********************************************
changed: [192.168.100.11]


PLAY RECAP *********************************************************************

192.168.100.11             : ok=3    changed=2    unreachable=0    failed=0
```

Note: Discussion about component of YML file

==

Handlers

A Handler is exactly the same as a Task (it can do anything a Task can), but it will run when called by another Task. You can think of it as part of an Event system; A Handler will take an action when called by an event it listens for.

This is useful for "secondary" actions that might be required after running a Task, such as starting a new service after installation or reloading a service after a configuration change

Example:

```
[root@be04c6686478 ansible]# cat  apache_final.yml
---
- hosts: webserver
  remote_user: root
  vars:

    http_port: 80

    max_client: 300

  remote_user: root

  tasks:

  - name: installing  httpd and check

    yum:

      name: httpd

      state: latest

  - name: start the apache service

    service:

      name: httpd

      state: started

      enabled: yes

  handlers:

  - name: checking service status

    service:

      name: httpd

      state: restarted

[root@be04c6686478 ansible]#
```

Handler called "checking service status". This Handler is the Task called when "checking service status" is notified.
This particular Handler uses the Service module, which can start, stop, restart, reload (and so on) system services. Here we simply tell Ansible that we want Nginx to be started.

Variables:

The vars directory contains a main.yml file which simply lists variables we'll use. This provides a convenient place for us to change configuration-wide settings.

```
[root@be04c6686478 ansible]# cat  apache_final.yml
---
- hosts: webserver
  remote_user: root
  vars:
    http_port: 80
    max_client: 300
  remote_user: root
```