Ansible

Introduction :

Ansible is an open source, a Configuration Management Tool and Deployment tool, maintained by Redhat.
The main components of Ansible are playbooks, configuration management, deployment.
Ansible uses playbooks to deploy, manage, build, test and configure anything from full server environments to custom compiled source code for applications.
Ansible was written in Python.

Ansible Features:

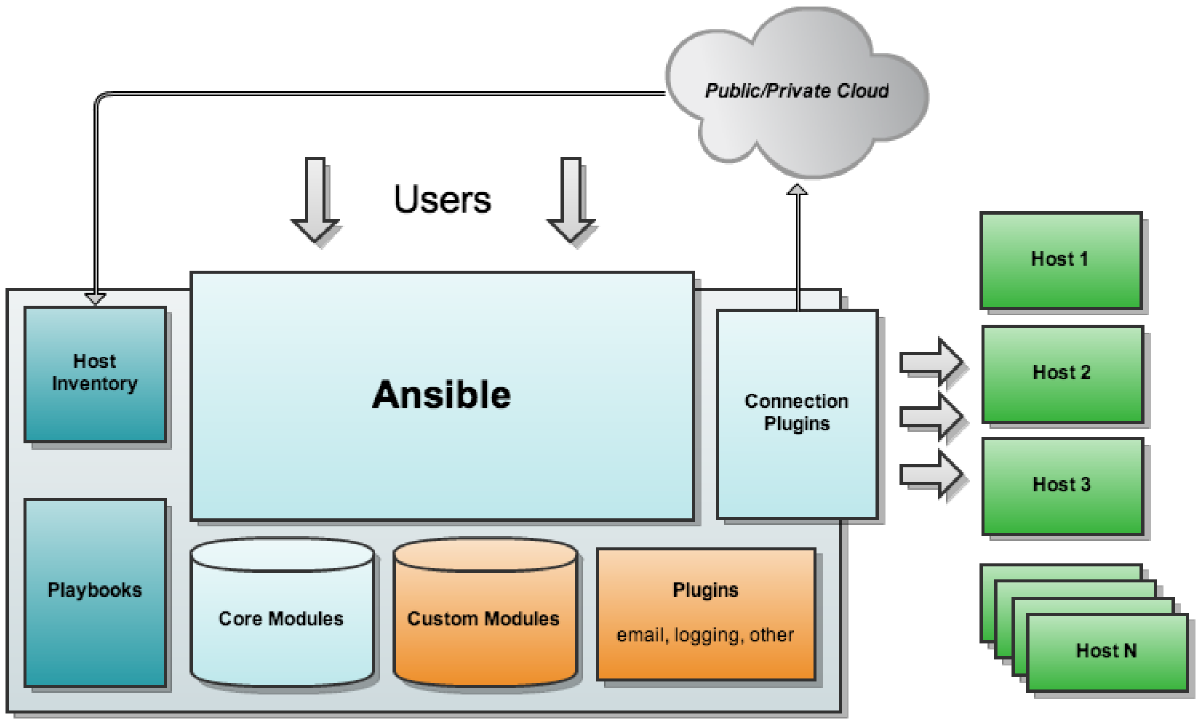
1)Ansible configure machines in an agent-less manner using SSH.
2)Built on top of Python and hence provides a lot of Python's functionality.
3)YAML-Based Playbooks
4)Uses SSH for secure connections.
5)Follows Push based architecture for sending configurations.

Push Based Vs Pull Based

Tools like Puppet and Chef are pull based.
Agents on the server periodically checks for the configuration information from central server (Master).

Ansible is push based.
Central server pushes the configuration information on target servers.
You control when the changes are made on the servers

Ansible Architecture:



Inventory file :

Ansible's inventory hosts file is used to list and group your servers. Its default location is /etc/ansible/hosts.

See the contents in hosts file as follows.

cat /etc/ansible/hosts (default inventory file path)
#192.168.122.1 ---> This is one of the nodes IP
192.168.122.2

In Inventory file you can mention IP address or Hostnames also.

Some important points in Inventory file.
- Comments begin with the '#' character
- Blank lines are ignored
- Groups of hosts are delimited by [header] elements
- You can enter hostnames or ip addresses
- A hostname/ip can be a member of multiple groups
- Ungrouped hosts are specifying before any group headers, like below.

Sample Inventory file:

```
# We can use '#' for comments in inventory file.

#Blank line are ignored.

#Ungrouped hosts are specifying before any group headers, like below
192.168.122.1
192.168.122.2

[webservers]
#192.168.122.1
192.168.122.2
192.168.122.3

[dbservers]
govardhan.db1.com

govardhan.db2.com
```

There are two types of Inventory:

1)Static Inventory: It's static file in which we will list the servers. Default location of static Inventory:

/etc/ansible/hosts

2) Dynamic Inventory: It's a Script(python, shell) which get the server details dynamically from external soures like AWS, AZURE,Database.

Ansible AD-HOC Commands :

To run any ansible command we will follow below syntax.

ansible [group Name|HostName] -m <<Module Name>> -a <<Command Name>>

Here -m is the module name and -a is the arguments to module.

Example:
ansible all -m shell -a date: It will display date from all host machines

There are two default groups, **all** and **ungrouped**. **all** contains every host. **ungrouped** contains all hosts that don't have another group

ansible-doc -l: It will display the all the modules available in Ansible.
ansible-doc yum: It will display more information about yum module along with examples.

Ping Module:

Use ping module to test Ansible and after successful run you can see the output.

ansible all -m ping: It will ping all the servers which you have mentioned in inventory file (/etc/ansible/hosts).
ansible all -m ping -o: It will display the output in single line.

Shell Module
----
ansible all -m shell -a 'uptime' : Uptime of all the machines.
Here m means module and -a means argument.
(OR)
ansible all -a 'uptime'
ansible all -m shell -a 'date' : Date of all machines

Yum Module
----
ansible all -b -m yum -a "name=vim" : It will install vim package in all node machine which you have menyioned in host inventory file.
ansible localhost -b -m yum -a "name=git" : To install git package in localhost.
ansible all -b -m yum -a "name=httpd state=present" : To install httpd package in all node machines.

Playbooks:

Playbook is a single YAML file, containing one or more ‘plays’ in a list.

Plays are ordered sets of tasks to execute against host servers from your inventory file.

Play defines a set of activities (tasks) to be run on hosts.

Task is an action to be perform on the host.

Examples are a) Execute a command
b) Run a shell script
c) Install a package
d) Shutdown/Restart the hosts.

Playbooks start with the YAML three dashes (---) and end with …

Each play has first hosts, variables, and tasks

FileName: pingServers.yml

```
---
- hosts: all
  gather_facts: no
  remote_user: ansible
  tasks:
        - name: Test connection
          ping:
          remote_user: ansible

…
```

Run the playbook Using below command:

ansible-playbook <<Playbbok file name>>:

ansible-playbook samplePlayBook.yml: It will run the samplePlayBook.yml playbook.

ansible-playbook --help: It will provide help on ansible_command command.

ansible-playbook playbook.yml --syntax-check: It will check the syntax of a playbook.

ansible-playbook playbook.yml --check: It will do in dry run.

ansible-playbook playbook.yml --list-hosts: It will display the which hosts would be affected by a playbook before you run it.

ansible-playbook playbook.yml --step: It execute one-step-at-a-time, confirm each task before running with (N)o/(y)es/(c)ontinue .

FileName: createFilePlaybook.yml

```
---
- hosts: all
  become: true
  tasks:
      - name: Creating a file
        file:
         path: /tmp/mithuntecnoroot.sh
         owner: root
         mode: 0777
         state: touch
...
```

FileName: installHTTPServerandStart.yml

```
---
- hosts: all
  become: true
  tasks:
      - name: Install Apache HTTP server
        yum: name=httpd update_cache=yes state=latest

      - name: Start HTTP Server
        service: name=httpd enabled=yes state=started

...
```

Nginx HTTP server installation:

```
---
- hosts: localhost
  tasks:
      - name: Install nginx server
        yum: name=nginx state=present
        become: true

      - name: Start nginx server
        service: name=nginx enabled=yes state=started
        become: true
...
```

Handlers:

Handlers are special task that run at the end of a play if notified by another task.
If a configuration file gets changed notify a service restart task it needs to run.

FileName: installHTTPServerWithHandlers.yml

```
---
- hosts: all
  become: true
  tasks:
      - name: Install Apache HTTP Server
        yum: name=httpd update_cache=yes state=latest
        notify:
             - Start HTTP Server
  handlers:
         - name: Start HTTP Server
           service:
                name=httpd
                state=restarted
```

Variables:

There are different ways in which you can define variables in Ansible. The simplest way is by using the **vars** section of a playbook. The below example defines a variable name called **package** and it is using in a task called **Install a Package**.

FileName: variables-playbook.yaml:

```
---
- hosts: localhost
  become: true
  vars:
     package: vim
  tasks:
     - name: Install a Package
       yum:
       name: "{{package}}"
       state: latest
```

Group Variables and Host Variables :

You can define custom variables for each group and host that you define in host inventory.

These variables are known as group_vars for groups and host_vars for hosts. Any variables that you define for a host or a group can be used in both playbooks and templates.

Both group_vars and host_vars are defined in their own folders, ‘groups_vars’ and ‘host_vars’, respectively. For group_vars, the file must be named exactly the same as the group.

For host_vars, the file has to be named exactly the same as the host.

Say you have a group ‘appServers’ and you want to define a variable for all of them. Create an empty file first in the group_vars folder in the root of your ansible directory (where you put you playbooks or in ansible home(installation) directory):

```
group_vars/appServers
```

Then add a variable to the file:

```
package=httpd
```

This makes the variable ‘package’ available to all playbooks that run on this group. I’ll show you how to use these in your playbook in a bit.

Say you have one group appServer you want to execute a certain action in a playbook by using varible defined in group_vars.
The following is an action you could use, using a group_var definition:

```
---
- hosts: appServers
  become: true
  tasks:
    - name: Install a Package
      yum:
        name: "{{package}}"
        state: latest
```

You can also do this with host_vars:

host_vars are similar to this. Create a file first:

host_vars/localhost

And add a variable:

package=git

this makes the variable ‘package’ available to all playbooks that run on this host.

```
---
- hosts: localhost
  become: true
  tasks:
     - name: Install a Package
       yum:
         name: "{{package}}"
         state: latest
```

If group_vars,host_vars has same variable with different values.While play is executed in that host variable value from host_vars will take precedence.

Conditional Statements :

The When Statement :

Sometimes you will want to skip a particular step on a particular host. This could be something as simple as not installing a certain package if the operating system is a particular version, or it could be something like performing some cleanup steps if a file system is getting full.

This is easy to do in Ansible with the *when* clause, which contains a raw Jinja2 expression without double curly braces

```
tasks:
- name: "shut down Debian flavored systems"
  command: /sbin/shutdown -t now
  when: ansible_facts['os_family'] == "Debian"
   # note that all variables can be directly in conditionals without double curly braces
```

You can also use parentheses to group conditions:

```
  tasks:
     - name: "shut down CentOS 6 and Debian 7 systems"
       command: /sbin/shutdown -t now
       when: (ansible_facts['distribution'] == "CentOS" and
ansible_facts['distribution_major_version'] == "6") or
(ansible_facts['distribution'] == "Debian" and ansible_facts['distribution_major_version'] == "7")
```

Multiple conditions that all need to be true (a logical ‘and’) can also be specified as a list:

```
tasks:
- name: "shut down CentOS 6 systems"
  command: /sbin/shutdown -t now
  when:
     - ansible_facts['distribution'] == "CentOS"
     - ansible_facts['distribution_major_version'] == "6"
```

Tags :

Tags are useful to be able to run a specific without running the whole playbook.

We can use tags on play and task level.

```
---
- hosts: localhost
  become: yes

  tasks:
    - name: Install Apache HTTP server on RedHat Server
      tags:
        - install
      yum:
        name: httpd
        state: present
       when: ansible_os_family == "RedHat"
   - name: Install Apache HTTP server on Ubuntu server
     tags:
       - install
       - start
    apt:
       name: apache2
       state: present
       when: ansible_os_family == "Debian"
  - name: Install Apache HTTP server on CentOS server
       yum:
          name: httpd
      state: present
      when:
        - ansible_facts['distribution'] == "CentOS"
         - ansible_facts['distribution_major_version'] == "7"
```

ansible-playbook sampleplaybook.yaml --list-tags: It will display all available tags in specified playbook.
ansible-playbook sampleplaybook.yml --tags "install,start" : This command will run the tags install and start.
ansible-playbook sampleplaybook.yml --skip-tags "install": This command will skip the tags specified tags, install.

Ansible Roles:

With more complexity in functionality, it becomes difficult to manage everything in one ansible playbook file. Sharing code among teams become difficult. Ansible Role helps solve these problems. Ansible role is an independent component which allows reuse of common configuration steps. Ansible role has to be used within playbook. Ansible roleis a set of tasks to configure a host to serve a certain purpose like configuring a service.
 Roles are defined using YAML files with a predefined directory structure.

What is Ansible roles?

1. Ansible roles are consists of many playbooks, which is similar to modules in puppet and cook books in chef. We term the same in ansible as roles.

2. Roles are a way to group multiple tasks together into one container to do the automation in very effective manner with clean directory structures.

3. Roles are set of tasks and additional files for a certain role which allow you to break up the configurations.

4. It can be easily reuse the codes by anyone if the role is suitable to someone.

5. It can be easily modify and will reduce the syntax errors.

Below is a sample playbook codes to deploy Apache web server. Let’s convert this playbook codes into Ansible roles.

```
---
- hosts: all
  become: true
  tasks:
     - name: Install httpd Package
       yum: name=httpd update_cache=yes state=latest
     - name: Copy httpd configuration file
       copy: src=httpd.conf dest=/etc/httpd/conf/httpd.conf
     - name: Copy index.html file
       copy: src=index.html dest=/var/www/html
       notify:
          - restart apache
      - name: Start and Enable httpd service
        service: name=httpd state=restarted enabled=yes
 handlers:
    - name: restart apache
      service: name=httpd state=restarted
```

How do we create Ansible Roles?

To create a Ansible roles, use ansible-galaxy command which has the templates to create it. This will default directories and do the modifications else we need to create each directories and files manually.

Let’s take an example to create a role for Apache Web server.

```
# mkdir /etc/ansible/rolesDemo

# ansible-galaxy init /etc/ansible/rolesDemo/apache

- apache was created successfully
```

where, ansible-glaxy is the command to create the roles using the templates.

init is to initiliaze the role.

apache is the name of role.

List out the directory created under /etc/ansible/rolesDemo.

To view the roles we use tree command.

```
# tree  /etc/ansible/rolesDemo/apache/
```

It will display like this:

```
|-- README.md
|-- defaults
    | `-- main.yml
|-- files
|-- handlers
     | `-- main.yml
|-- meta
   | `-- main.yml
|-- tasks
   | `-- main.yml
|-- templates
|-- tests
| |-- inventory
| `-- test.yml
   `-- vars
 `-- main.yml
```

We have got the clean directory structure with the ansible-galaxy command. Each directory must contain a main.yml file, which contains the relevant content.

Directory Structure:

A role directory structure contains directories: defaults, vars, tasks, files, templates, meta, handlers. Each directory must contain a main.yml file which contains relevant content. Let’s look little closer to each directory.

1. defaults: contains default variables for the role. Variables in default have the lowest priority so they are easy to override.
2. vars: contains variables for the role. Variables in vars have higher priority than variables in defaults directory.
3. tasks: contains the main list of steps to be executed by the role.
4. files: contains files which we want to be copied to the remote host. We don’t need to specify a path of resources stored in this directory.
5. templates: contains file template which supports modifications from the role. We use the Jinja2 templating language for creating templates.
6. meta: contains metadata of role like an author, support platforms.
7. handlers: contains handlers which can be invoked by “notify” directives and are associated with service.

We have got all the required files for Apache roles. Let’s apply this role into the ansible playbook "site.yml" as below to deploy it on the client nodes.

```
# cat  /etc/ansible/rolesDemo/site.yml

 ---
- hosts: appServers
  roles:
    - apache
```

We have defined this changes should be run only on appServers, you can also use "all" if need. Specify the role name as "apache", also if you have created multiple roles, you can use the below format to add it.

```
- apache
- common
```

Let’s verify for syntax errors:

```
# ansible-playbook /etc/ansible/rolesDemo/site.yml --syntax-check

playbook: /etc/ansible/rolesDemo/site.yml
```

If No errors found. Let move on to deploy the roles.

```
# ansible-playbook /etc/ansible/rolesDemo/site.yml
```

Ansible Vault :

A typical Ansible setup will involve needing some sort of secret to fully setup a server or application.Common types of "secret" include passwords, SSH keys, SSL certificates, API tokens and anything else you don't want the public to see.

Since it's common to store Ansible configurations in version control, we need a way to store secrets securely.

Ansible Vault is the answer to this. Ansible Vault can encrypt anything inside of a YAML file, using a password of your choice.