Introduction To Ansible

- Ansible is one of the open source IT-Automation tool.
- It is use for Server configuration management, application deployment, task automation.
- Ansible written in Python which needs to be installed on the remote host.
- Unlike Puppet, Ansible doesn't have a client server module where a server is a master and Ansible support python 2.6 to 3 and currently ansible not support to 3.x version.
- a client has the agent role. Ansible is more of a standalone solution where a computer manages other devices without those other devices having to be "Ansible aware".

Ansible Installation prerequisite:

On control machine ansible requires:

Python 2.6 or 2.7.

On nodes

SSHPython 2.6 or later

Note: Windows can't be used as control machine

Ansible Installation :

Ubuntu:

```
# manage the repositories that you install software from (common)
$ sudo apt-get install software-properties-common
$ sudo apt-add-repository ppa:ansible/ansible
$ sudo apt-get update
$ sudo apt-get install ansible
```

Centos:

```
# install the epel-release RPM if needed on CentOS, RHEL, or Scientific Linux
$ sudo yum install ansible
```

Above commands will install latest stable version of ansible. Current stable version is 2.2.1.

Ansible home directory will be created after installation, below is the directory structure. On ubuntu home directory path is /etc/ansible

```
├──── ansible.cfg ( file contains ansible config ]
└──── hosts [ inventory file , use to configure the hosts ]
└──── roles [ empty dir to create roles ]
```

ansible.cfg : This file contains ansible configuration settings. Certain settings in ansible can be adjustable through this file.
hosts : This file contains hosts(remote node machines) list. This file is called as Inventory. This
roles: Empty directory to create roles.

Inventory:

Ansible works for all systems listed under hosts file located at **/etc/ansible/hosts**, this file is considered as default Inventory file. If anyone want to use other file this can be achieved by using -i <file_path> in command line.

Format for hosts/inventory file is INI-like format.

Example:

```
www.example.com

[dev_servers]
www.dev1.com
www.dev2.com

[qa_servers]
www.qa1.com
www.qa2.com
```

In the above example:
The headings in brackets are group names, which are used in classifying systems and deciding what systems you are controlling at what times and for what purpose.

www.example.com: It is in all group.
www.qa1.com, www.qa2.com: These are in qa_servers growww.dev1.com, www.dev2.com: These are in dev_servers group.
up.

Ansible Playbook :

A playbook is a set of instructions written in YAML. It tells Ansible which devices to connect and what to do with them. Playbooks are the entry point of an Ansible provisioning. They contain information about the systems where the provisioning should be executed, as well as the directives or steps that should be executed.

Example: vim.yml

```
---
- hosts: all
 tasks:
 - name: Update apt-cache
   apt: update_cache=yes
 - name: Install Vim
   apt: name=vim state=latest
```

Command to execute playbook

```
$ ansible-playbook all -i hosts vim.yml
```

This playbook will update cache and install vim on all hosts from inventory file.

Below are the few points that are useful while writing playbook:

1. Variables
   Variables can be declared and used with vars section in playbook

   Eg:

```
- hosts: all

 vars:

        package: vim # declaration of variable package=vim

 tasks:

 - name: Install Vim

   apt: name={{ package }} state=latest // package variable is used
```

1. Register variables
   Output of command execution can be stored and used as variable.

   Eg:

```
---
- hosts: all

 tasks:

 - name: Get php version

   shell: php -v

   register: php_veriosn # store output of php-v
```

Conditionals:

The When Statement :

Ex:

```
 tasks:

  - name: "shut down Debian flavored systems"

        command: /sbin/shutdown -t now

        when: ansible_os_family == "Debian"
```

parentheses to group conditions:

```
tasks:
 - name: "shut down CentOS 6 and Debian 7 systems"
        command: /sbin/shutdown -t now
        when: (ansible_distribution == "CentOS" and ansible_distribution_major_version == "6") or
        (ansible_distribution == "Debian" and ansible_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_distribution == "CentOS"
        - ansible_distribution_major_version == "6"
```

Loops and Conditionals :

```
tasks:
        - command: echo {{ item }}
        with_items: [ 0, 2, 4, 6, 8, 10 ]
        when: item > 5
```

Loops

Loops are used to repeat task using different input values. To create a loop within task use with_items with multiple values, each value is accessed through loop variable item.

Eg:

```
---
- hosts: all
  vars:
    packages: [ 'wget', 'gedit' ]
  tasks:

   - name: Install Package
     apt: name={{ item }} state=latest
    # Install wget and gedit in one task
     with_items: packages

  - name: Install Packages
     apt: name={{ item }} state=latest
    # Install vim, git and curl in one task
     with_items:
       - vim
       - git
       - curl
```

Templates

Templating is used to enable dynamic expressions and access to variables. Ansible uses jinja2 template engine. It is used to set up configurations which are reusable.

Eg: virtual host setup with template

vhost.j2 template file

```
<VirtualHost *:80>
    ServerName {{ server_name }}
   ServerAdmin {{ server_admin }}
    DocumentRoot {{ doc_root }}

   <Directory {{ doc_root }}>
     AllowOverride None
          Order allow,deny
          Allow from all
          DirectoryIndex app_dev.php
```

```
  </Directory>
 </VirtualHost>
```

In playbook

```
---
hosts: all
vars:
  server_name: www.test.com #defined server_name
  server_admin: webmaster@localhost #defined server_admin
  doc_root: /var/www/html/test_project #defined doc_root

tasks:
- name: Change default Apache virtual host
  template: src=vhost.j2 dest=/etc/apache2/sites-available/000-default.conf
```

In above example server_name, server_admin and doc_root is defined which are required in vhost.j2 file, so variables in vhost.j2 will replace with value. In this way we can use vhost.j2 to configure number of hosts.

Triggering Handlers

Handlers in playbook are just like regular tasks, but handlers will execute from notify directive in task. handlers are only fired when certain tasks report changes, and are run at the end of each play.

```
---
hosts: all
vars:
  server_name: www.test.com #defined server_name
  server_admin: webmaster@localhost #defined server_admin
  doc_root: /var/www/html/test_project #defined doc_root

tasks:
- name: Change default Apache virtual host
  template: src=vhost.j2 dest=/etc/apache2/sites-available/000-default.conf
  notify: restart apache #invoke restart apache

handlers:
  - name: restart apache
```

```
    service: name=apache2 state=restarted

  - name: other handler
    service: name=other state=restarted
```

Ansible Role:

Role is a set of tasks and additional files to configure host to serve for a certain role. Role is the best way to organize your playbooks. Roles are ways of automatically loading certain vars_files, tasks, and handlers based on a known file structure. Grouping content by roles also allows easy sharing of roles with other users.

The first step in creating a role is creating its directory structure. In order to create the base directory structure, use ansible-galaxy or create following directory structure manually :

To create basic folder structure of role use following command:
$ ansible-galaxy init test
Note : In above command test is name of role

Role Directory Structure:
A role directory structure consists of defaults, vars, files, handlers, meta, tasks, and templates

1. Defaults : Within defaults, there is a main.yml file with the default variables used by a role.
2. Vars : vars and defaults contain variables, but variables in vars have a higher priority, which means that they are more difficult to override. Variables in defaults have the lowest priority of any variables available, which means they're easy to override.
3. 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.
4. Handlers: All handlers that were in your playbook previously can now be added into this directory.
5. 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.
6. 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.

7. Templates : This directory contains files that use variables to substitute information during creation in this directory. Modifications are achieved through the Jinja2 templating language. Most software configuration files become templates.

Roles directory path is configurable, default roles directory path is ansible installation directory path. This path can be changed using ansible.cfg file which is located in ansible installation directory. To use different directory path change value of roles_path variable.
User can have separate roles directory for each playbook. To use separate roles directory, ansible.cfg needs to be created in each playbook file. Add below code in each file and change path.

```
[defaults]
roles_path=path/to/roles
```

Example:

Below is the example to install apache on ubuntu with roles. Directory structure for playbook will be as shown below.

```
apache_installation
|-- hosts
|-- ansible.cfg
|-- main.yml
|-- roles
|   |-- apache
|       |-- tasks
|           |-- main.yml
|       |-- handlers
|           |-- main.yml
```

hosts: add remote node entries
localhost ansible_user=test

ansible.cfg: update roles path

```
[defaults]
roles_path=path/to/roles
```

main.yml: play to install apache on ubuntu

```
---
- hosts: all
 gather_facts: True
 become: true

 # add role apache
 roles:
  - { role: apache }
```

roles:

roles/apache/tasks/main.yml : this file contains tasks, as shown in below code snippet

```
---
- name: Install apache ubuntu
 apt: pkg=apache2 state=present update_cache=yes cache_valid_time=86400
 when: ansible_distribution == 'Debian' or ansible_distribution == 'Ubuntu'
 notify:
  - start apache # invoke start apache handlers
```

roles/apache/handlers/main.yml : this file contains handlers which are used in notify.

```
---
- name: start apache
 service: name=apache2 state=started
```

To execute above playbook use below command:

```
$ ansible-playbook -i hosts main.yml --ask-sudo-pass
```

Useful Commands:

1. List playbook task:
   ansible-playbook -i hosts site.yml --list-tasks
2. Dry Run play book
   ansible-playbook --check playbook.yml

3. Run Step By Step:
   ansible-playbook playbook.yml -i hosts --step
4. Run based on task tag
   ansible-playbook --tags=dependencies playbook.yml
5. Clear the cache
   ansible-playbook playbook.yml --flush-cache
6. Execute playbook task with sudo user
   ansible-playbook -i hosts apache2.yml --ask-sudo-pass
7. For finding syntax of ansible module
   ansible-doc module_name
1. Syntax check
   Ansible-playbook --syntax-check path/to/Playbook

Important topics :

ansible-doc

ansible-doc is very useful for finding the syntax of a module without having to look it up online

e.g. ansible-doc mysql_user

Ansible Playbook Execution Steps :

1. Command to execute playbook :

   Command :ansible-playbook -i hosts images.yml

2. Sample output after executing above command :

```
$ ansible-playbook -i hosts images.yml

PLAY [local] ********************************************************************

GATHERING FACTS *****************************************************************
ok: [localhost]

TASK: [build Apache image] ******************************************************
changed: [localhost]

PLAY RECAP **********************************************************************
localhost                  : ok=2    changed=1    unreachable=0    failed=0
```