What is ansible?

Ansible is simple, open-source, configuration management tool used for IT automation engine for cloud infrastructure, in-house servers.

Ansible is **push based configuration** management system/tool

1. Ansible does not require any dedicated agent running on the target host machines.
2. Minimum ansible requirement is host machines with python installed on it.
3. We only require a proper ssh connection between the controller and the host machines.

Advantages:

1. Free/open source
2. Very simple to setup up
3. Agent less CM tool
4. Execute task from our own machine
5. Configuration/installation/deployment steps in a single yaml file
6. Reuse same file multiple times for different environments

Ansible Architecture

https://docs.ansible.com/ansible/latest/dev_guide/overview_architecture.html

Ansible Inventory

Inventory is a simple file which contains complete information about remote target machines on which ansible has to run the configuration

Static Inventory

Static inventory file is a list of managed hosts declared under a host group using either hostnames or IP addresses in a plain text file.

- default location of host file is /etc/ansible/hosts
- each line except group name is considered as a single host connection configuration.

- To pass custom inventory file "ansible -m ping -i <inventoryfile>"

Dynamic Inventory

Dynamic Inventory is generated by a script written in Python or any other programming language or by using plugins

- Is basically a script which will gives the inventory output.
- we can use python, perl, bash

To install ansible (Ubuntu)

1. Establish a passwordless connection
ssh-keygen -t rsa (optional)

use ssh-copy-id username@remote_host (optional) or manually paste

sudo apt update

sudo apt install ansible

To setup inventory file

$ sudo nano /etc/ansible/hosts

To check ansible inventory

ansible-inventory --list -y

To test connection to host

ansible all -m ping

Ansible playbook

- can contain n number of play

- each play is designated to run n number task

- each task is designated to execute a module (only one module per task)

Playbook to install nginx on RHEL distribution

```
tasks:
 -name:create a directory nginx
  file:
    path: /ubuntu/home/nginx
    state: directory

 -name: Installing nginx
  yum:
    name:nginx
    state:latest

 -name: start nginx
  service
    name:nginx
    state:start
```

Playbook to install tomcat on ubuntu

```
tasks:
 -name: installing tomcat
  apt:
   name: tomcat
   state: latest
 -name: start tomcat
  systemctl:
    name:tomcat
    state:start

 -name: status of tomcat running or not
  systemctl:
    name:tomcat
    state:status
```

Playbook to install tomcat on server using groupname from inventory

```
 -name:play_1
  hosts: frontend
  tasks:
   -name: install tomcat
    apt:
     name: tomcat
     state: latest
   -name: start tomcat
    systemctl:
     name:tomcat
     state:start

 - name: Play_1
   hosts: backend
   tasks:
     - name: Install git (backend)
       become: true
       apt:
         name: git
         state: present
         update_cache: yes
     - name: Install jq (backend)
       become: true
       apt:
         name: jq
```

```
      state: latest
      update_cache: yes
```

Installing multiple items with single task

```
- name: Demo
  hosts: frontend
  tasks:
    - name: Install git (frontend)
      become: true
      apt:
        name: "{{ item }}"
        state: present
        update_cache: yes
      with_items:
          - git
          - nginx
          - memcached
          - jq
          - curl
          - wget
```

Ansible Playbook explanation

`name` Name of the playbook

`hosts` A set of hosts usually grouped together as a host group and defined in inventory file

`become` To tell ansible this play has to be executed with elevated privileges

`become_user` the user's name that we want to switch to like compare it with `sudo su - user`

`tasks` set of tasks to execute, All tasks would be defined below this

Ansible modules

Ansible modules are reusable, standalone scripts that can be used by the Ansible API, or by the ansible or ansible-playbook programs

System- User, Group, Hostname, service, Mount, Ping
Command- Command, Expect, Shell, script
Files- Archive, File, Find, replace, Copy
Database- Mongo, MySQL, postgress

Ansible facts

- Ansible facts are the variable which contains the details of target hosts
- all the facts are prefixed with ansible_ word
- setup module is used to gather all facts (ansible all -m setup)

Install package on both ubuntu and redhat by referring the facts

```
- name: Play_1
  hosts: all
  tasks:
    - name: Install git (frontend)
      become: true
      apt:
        name: "{{ item }}"
        state: present
        update_cache: yes
      when:
            - ansible_distribution == "Ubuntu"
                - ansible_pkg_mgr == "apt"
      with_items:
         - git
         - nginx
         - memcached
         - jq
         - curl
         - wget
    - name: Install git (frontend)
      become: true
      yum:
        name: "{{ item }}"
        state: present
        update_cache: yes
      when: ansible_distribution == "RedHat"
      with_items:
         - git
         - nginx
         - memcached
         - jq
         - curl
         - wget
```

Ansible run a task only if previous is success

- using register, we can able to achieve this condition
- register will save the output in a variable in JSON format only if the task executes successfully

- we can register the output of task which we want check the status and, on another task

```
- name: Play_1
  hosts: all
  tasks:
    - name: Install multiple packages on Ubuntu
      become: true
      apt:
        name: "{{ item }}"
        state: present
        update_cache: yes
      when:
       - ansible_pkg_mgr == "apt"
       - ansible_distribution == "Ubuntu"
      with_items:
         - git
         - nginx
         - memcached
         - jq
         - curl
         - wget
      register: result

    - name: Print debug message
      debug:
        var: result
        verbosity: 0
           when: result is changed
```

Ansible debug

- print output of a task with register varibale

```
---
- name: Play_1
  hosts: all
  tasks:
    - name: Install multiple packages on Ubuntu
      become: true
      apt:
        name: "{{ item }}"
        state: present
        update_cache: yes
      when:
       - ansible_pkg_mgr == "apt"
       - ansible_distribution == "Ubuntu"
      with_items:
         - git
```

```
        - nginx
        - memcached
        - jq
        - curl
        - wget
    register: result

  - name: Print debug message
    debug:
      var: result
      verbosity: 0
```

Ansible Run specific tasks / plays

- Using tags, we can run specific task or play in a playbook

```
- name: Play_1
  hosts: all
  tasks:
    - name: Install git on Ubuntu
      become: yes
      apt:
        name: git
        state: present
        update_cache: yes
      tags:
        - ubuntu_pkg
    - name: Install curl
      become: yes
      apt:
        name: curl
        state: present
        update_cache: yes
      tags:
        - curl_pkg
```

ansible-playbook playbook.yaml --tags="ubuntu_pkg"
ansible-playbook playbook.yaml --skip-tags="ubuntu_p

To run only task with the tag

ansible-playbook playbook.yaml --tags="tag_name1,tag_name2"

ansible-playbook playbook.yaml --tags="ubuntu_pkg"

To skip a tagged play / task

ansible-playbook playbook.yaml --skip-tags="tag_name1,tag_name2"

ansible-playbook playbook.yaml --skip-tags="ubuntu_pkg"

Ansible Roles

- Instead of defining everything play, handlers, templates etc. in a single file we can define it in a proper folder structure using roles
- default location of roles is /etc/ansible/role (if directory is not present create it)
- ansible-galaxy is the tool which ansible gives us to create a role

  ansible-galaxy init <role_name>
- Download the roles from ansible galaxy

  ansible-galaxy install <role_name>
- Role directory structure

```
roles
└── install_pkg
    ├── README.md
    ├── defaults
    │   └── main.yml
    ├── files
    ├── handlers
    │   └── main.yml
    ├── meta
    │   └── main.yml
    ├── tasks
    │   └── main.yml
    ├── templates
    ├── tests
    │   ├── inventory
    │   └── test.yml
    └── vars
        └── main.yml
```

tasks - Contains the main list of tasks that we want to execute by the role.
main.yml is where we put our tasks

handlers - Contains tasks which are executed only if it is notified by another task after successful execution (if there is change from the notifying task).

defaults - It contains default variables for the role and it is the least precedence variable in ansible.

vars - The variables which will be used by role. These variables can be defined in us playbook also. (task/play variables)

files - Contains static file that can be copied to hosts by role
In task I don't need use the complete path just filename is used to copy.
templates - used to define dynamic value in files and copy it to host with values defined.

meta - To define role dependencies.

Ansible Vault

Ansible Vault is a feature of ansible that allows you to keep sensitive data such as passwords or keys in encrypted files, rather than as plaintext in playbooks or roles

ansible-vault create vault.yml

ansible-vault encrypt inventory.txt ansible-vault decrypt vault.yml

ansible-vault view vault.yml ansible-vault edit vault.yml

Ansible variables

Ansible uses variables to manage differences between systems

```
- hosts: all
  vars:
    hello: world
  tasks:
  - name: Ansible Basic Variable Example
    debug:
      msg: "{{ hello }}"
```

command - ansible-playbook <file> --extra-var hello=123

Diff b/w Ansible and terraform

.

Ansible	Terraform
Ansible is a Configuration Management Tool.	Terraform is Provisioning Tool
Ansible supports the provisioning of Bare Metal servers.	Terraform by default does not support provisioning of Bare Metal servers.
Ansible follows a procedural approach.	Terraform follows a declarative infrastructure as a code approach.
Ansible provides full support for packaging and templating.	Terraform does not provide better support when it comes to packaging and templating.
Ansible uses LaaC for procedural execution.	Terraform uses declarative LaaC.
Ansible can be used to deploy apps on top of the cloud.	Terraforms can be used to deploy load balances, computing, VPCs, and storage.
Ansible is well prepared for configuring servers with the right software and updates on an already configured cloud.	Terraform is well prepared for orchestrating cloud services and setting up cloud infrastructure from scratch.
Ansible does not have life-cycle management.	Terraform highly depends on the state or life-cycle management.