Ansible

DevOps Training

What is Ansible?

Ansible is an open-source configuration management and provisioning tool, similar to Chef, Puppet or Salt.

It uses SSH to connect to servers and run the configured Tasks. Ansible lets you control and configure nodes from a single machine.

What makes it different from other management software is that Ansible uses SSH infrastructure. The project was founded in 2013 and bought by Red Hat in 2015.

Why Ansible?

No Agent- As long as the box can be ssh'd into and it has python, it can be configured with Ansible.

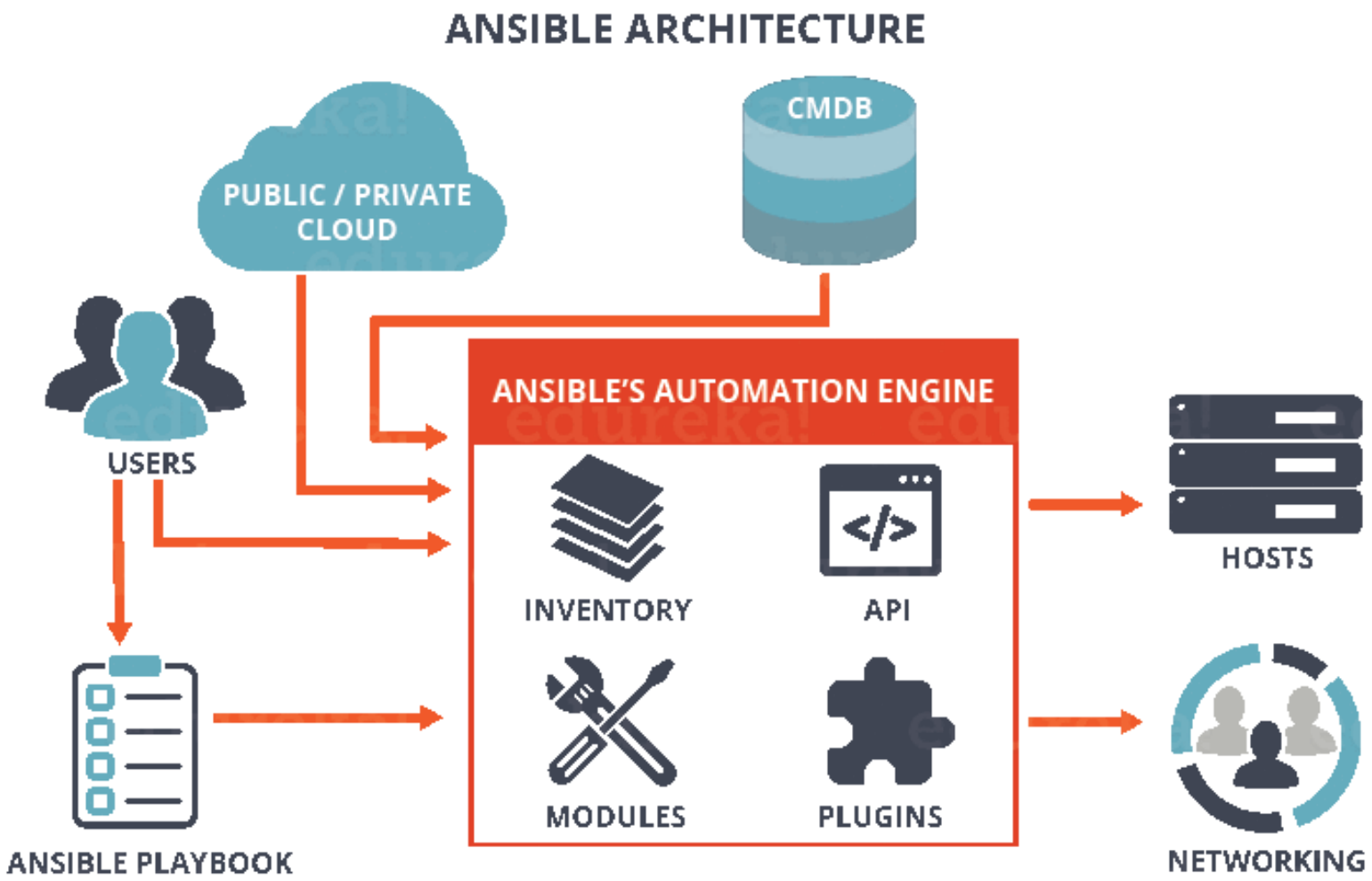
Idempotent- Ansible's whole architecture is structured around the concept of idempotency. The core idea here is that you only do things if they are needed and that things are repeatable without side effects.

Tiny Learning Curve- Ansible is quite easy to learn. It doesn't require any extra knowledge.

Ansible Use Cases

- Provisioning
- Configuration Management
- App Deployment
- Continuous Delivery
- Security & Compliance
- Orchestration

Architecture of Ansible



Inventory

Inventory is a collection of hosts (nodes) with associated data and groupings that Ansible can connect and manage.

Hosts (nodes)

Groups

Inventory-specific data (variables)

Static or dynamic sources

By default, the Inventory is described by a configuration file, whose default location is in./etc/ansible/hosts

The configuration file lists either the IP address or hostname of each node that is accessible by Ansible.

Example of an Inventory File

```
mail.example.com

[webservers]
foo.example.com
bar.example.com

[dbservers]
one.example.com
two.example.com
three.example.com
```

Ad-hoc Commands

An ad-hoc command is a single Ansible task to perform quickly, but don't want to save for later.

```
# check all my inventory hosts are ready to be
# managed by Ansible

$ ansible all -m ping

# collect and display the discovered facts
# for the localhost

$ ansible localhost -m setup
```

Tasks

Tasks are the application of a module to perform a specific unit of work.

file: A directory should exist

yum: A package should be installed

service: A service should be running

template: Render a configuration file from a template

get_url: Fetch an archive file from a URL

Example Tasks in a Play

```
tasks:

- name: httpd package is present
  yum:
    name: httpd
    state: latest

- name: latest index.html file is present
  copy:
    src: files/index.html
```

Playbook

Plays are ordered sets of tasks to execute against host selections from your inventory.
A playbook is a file containing one or more play

Playbooks are simple YAML files.

These files are descriptions of the desired state of your systems.
Ansible then does the hard work of getting your systems to that state no matter what state they are currently in.

Playbooks make your installations, upgrades and day-to-day management repeatable and reliable.

Playbooks are simple to write and maintain.
Playbooks are written in a natural language so they are very easy to evolve and edit.

Playbook contains Plays.

Plays contain tasks.

tasks call modules.

Example of an ansible playbook

—

```
- hosts: webservers
remote_user: root

tasks:

- name: ensure apache is at the latest version
    yum: name=httpd state=latest

- name: ensure apache is running
    service: name=httpd state=started enabled=yes
```

Doing More with Playbooks

Here are some more essential playbook features that you can apply:

Templates

Loops

Conditionals

Tags

Blocks

Loops

Loops can do one task on multiple things,
such as create a lot of users,
install a lot of packages,
or repeat a polling step until a certain result is reached.

```
-yum:

 name: "{{ item }}"

 state: latest

with_items:
```

Conditionals

Ansible supports the conditional execution of a task based on the run-time evaluation of variable, fact, or previous task result.

```
-yum:

name: httpd

state: latest

 when: ansible_os_family == "RedHat"
```

Tags

Tags are useful to be able to run a subset of a playbook on-demand.

```
-yum:
 name: "{{ item }}"
 state: latest
with_items:
- httpd
- mod_wsgi
tags:
 - packages

- template:
src: templates/httpd.conf.j2
dest: /etc/httpd/conf/httpd.conf
tags:
 - configuration
```

Blocks

Blocks cut down on repetitive task directives, allow for logical grouping of tasks and even in play error handling

```
- block:
- yum:
name: "{{ item }}"
state: latest
with_items:
- httpd
- mod_wsgi
- template:
src: templates/httpd.conf.j2
dest: /etc/httpd/conf/httpd.conf
when: ansible_os_family == "RedHat"
```

Blocks

```
tasks:
  - name: Install, configure, and start Apache
    block:
      - name: install httpd and memcached
        yum:
          name:
          - httpd
          - memcached
          state: present

      - name: apply the foo config template
        template:
          src: templates/src.j2
          dest: /etc/foo.conf
      - name: start service bar and enable it
        service:
          name: bar
          state: started
          enabled: True
    when: ansible_facts['distribution'] == 'CentOS'
    become: true
    become_user: root
    ignore_errors: yes
```

Modules

There are over 1000 modules provided by Ansible to automate every part of the environment. Modules are like plugins that do the actual work in Ansible, they are what gets executed in each playbook task.

Each module is mostly standalone and can be written in a standard scripting language (such as Python, Perl, Ruby, Bash, etc.).

One of the guiding properties of modules is idempotency, which means that even if an operation is repeated multiple times, it will always place the system into the same state.

Example of Modules

There are lots of modules such as :

Service, file, copy, iptables etc.

Any Module can be used as :

```
ansible 127.0.0.1 -m service -a "name=httpd
state=started"
ansible localhost -m ping
```

Roles

Roles are a way to group tasks together into one container.

We could have a role for setting up MySQL, another one for configuring iptables etc.

Roles makes it easy to configure hosts. Any role can be performed on any host or group of hosts such as:

```
- hosts: all
roles:
- role_1
- role_2
```

Roles

Improves readability and maintainability of complex plays

Eases sharing, reuse and standardization of automation processes

Enables Ansible content to exist independently of playbooks, projects & even organizations

Provides functional conveniences such as file path resolution and default values

Project with Roles Example

```
site.yml
roles/
common/
files/
templates/
tasks/
handlers/
vars/
defaults/
meta/
apache/
files/
templates/
tasks/
handlers/
vars/
defaults/
```

THANK YOU!

training@laksans.com