

# Ansible

Ansible is a deployment automation tool which traditionally uses push approach to achieve its objectives, by managing all the servers through one single machine running the Ansible Configuration Management Tool

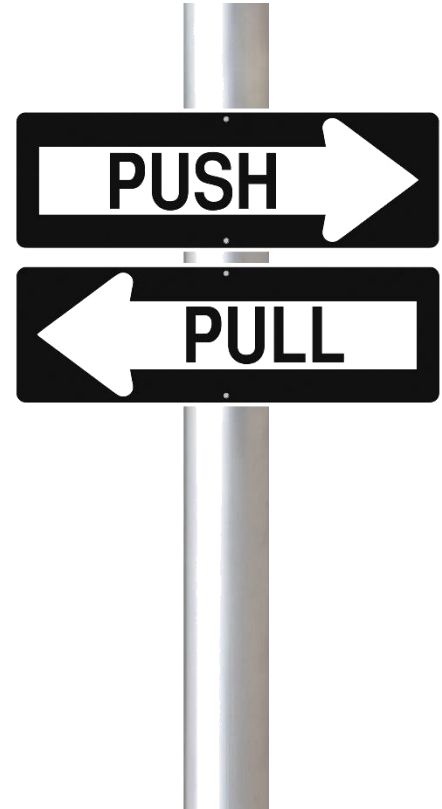


ANSIBLE

Ansible clients do not have any agents installed on them, therefore there is no concept of polling with central server. Ansible uses the **Push** approach instead. Still, ansible is flexible enough to let the user implement Pull architecture as well.

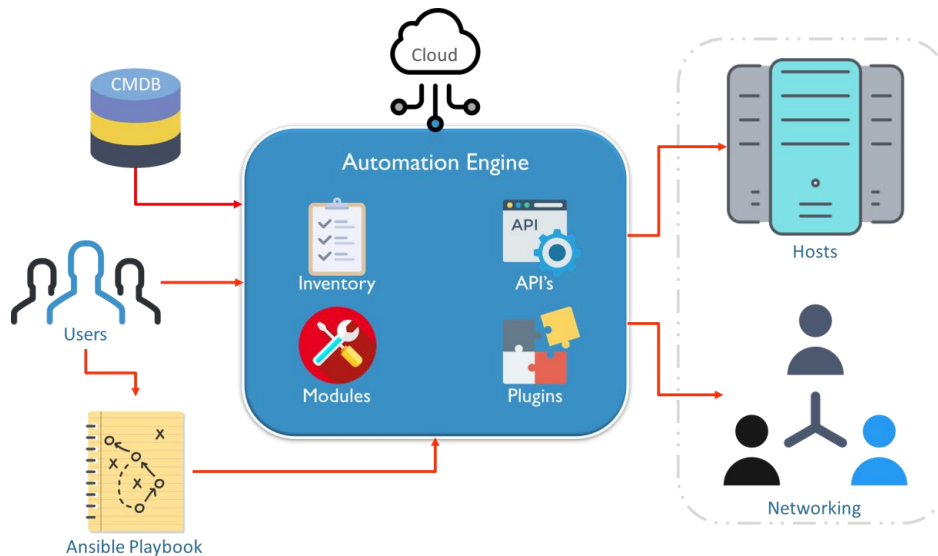
## Configuration Management Tools

- Push approach does not require agents set up on individual nodes like pull does
- Push based systems are completely Synchronous as you can see the changes made instantaneously and can fix the system if changes cause problems
- Systems using pull architecture can scale quite easily which is not the case with push model



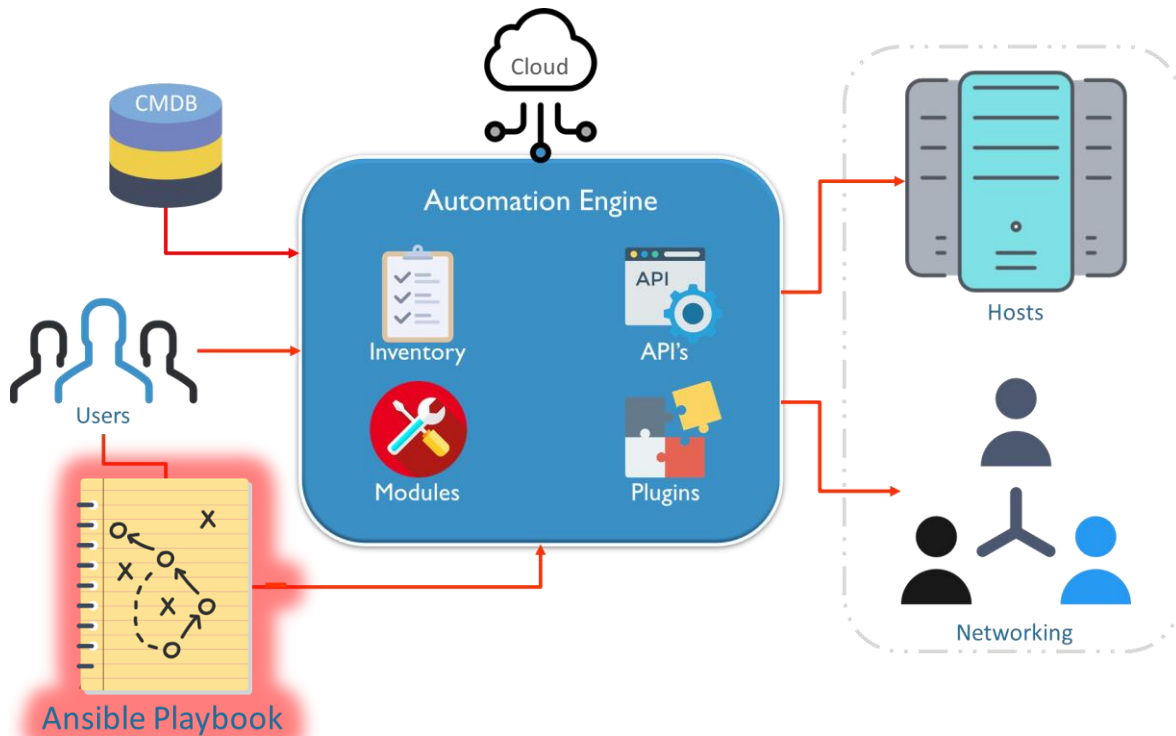
# Ansible Architecture

Ansible uses playbooks to implement the desired configuration changes. The user directly interacts with the ansible automation engine and playbooks.



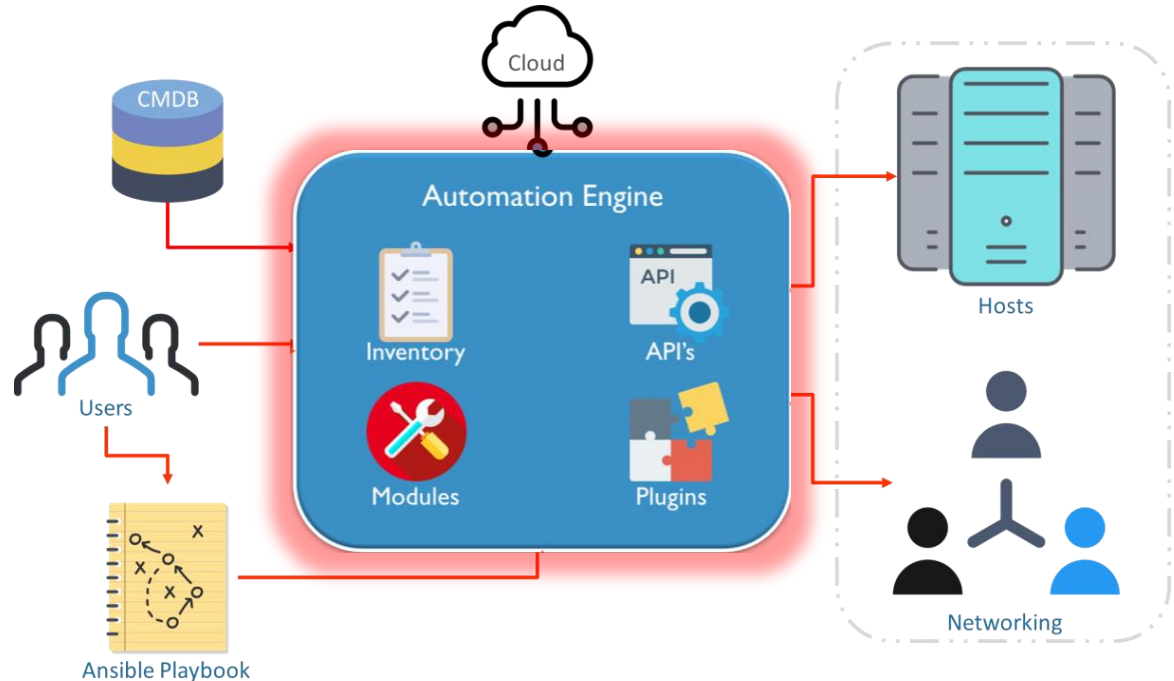
# Ansible Architecture - Playbooks

- Ansible uses playbooks to implement the changes desired by the users
- Playbooks contain plays, plays have tasks and tasks call modules
- Modules are the units which actually execute on the servers



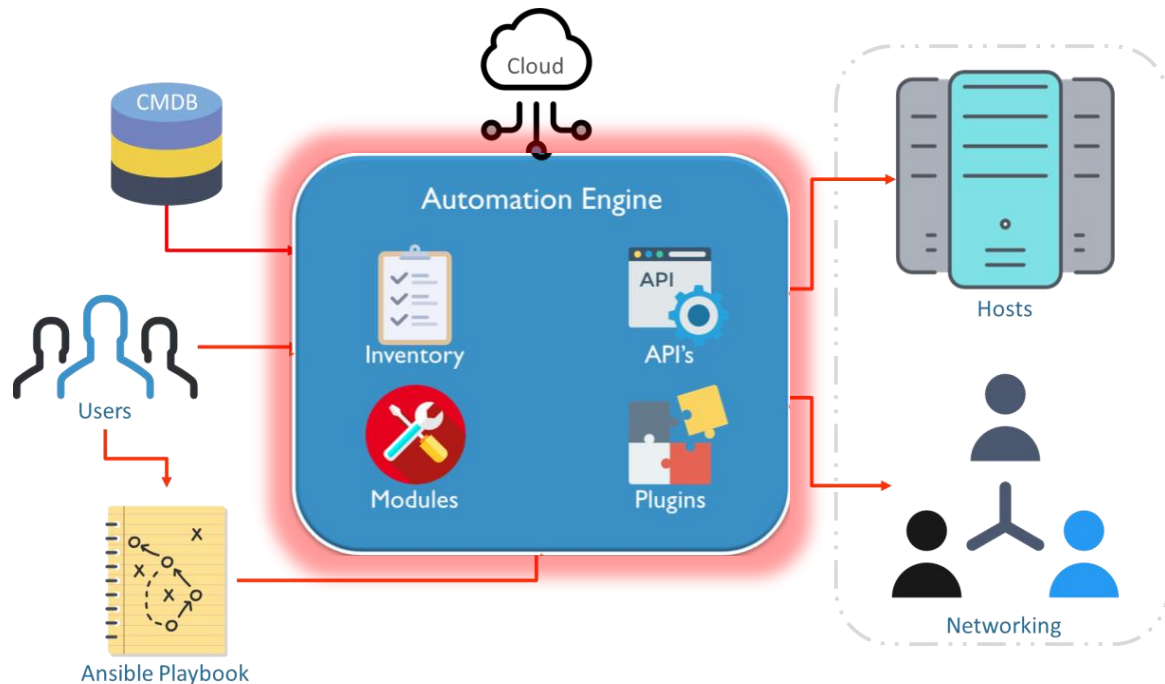
# Ansible Architecture – Automation Engine

- Ansible uses playbooks to implement the changes desired by the users
- Playbooks contain plays, plays have tasks and tasks call modules
- Modules are the units which actually execute on the servers



# Ansible Architecture – Automation Engine

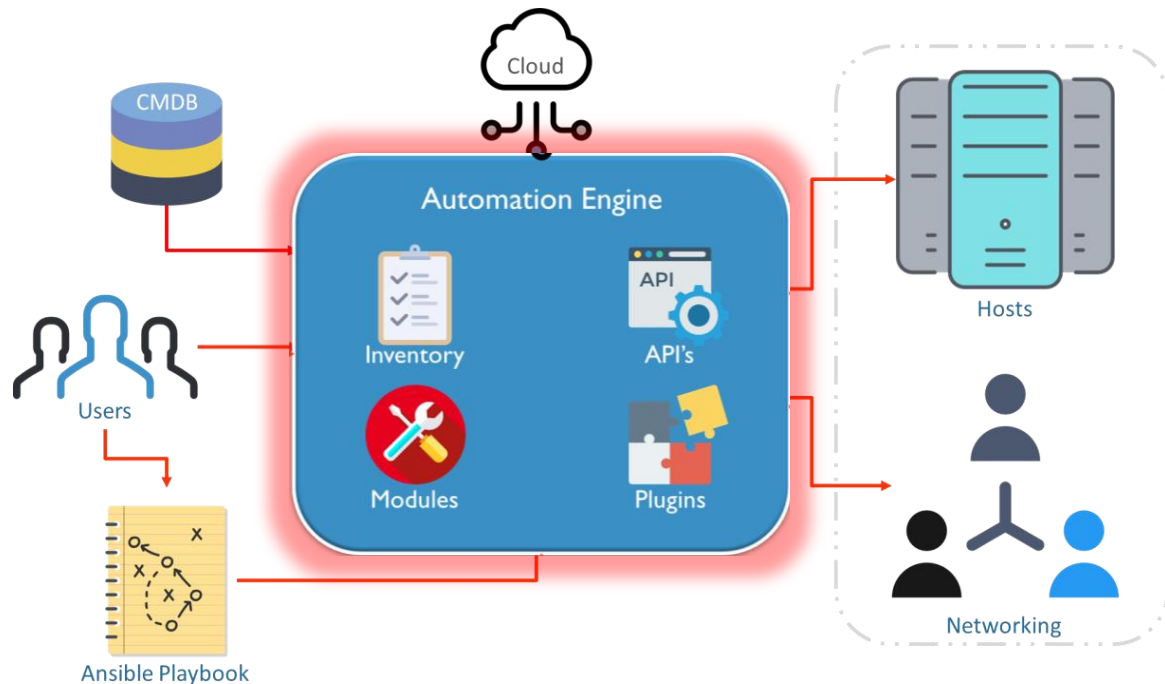
- The API's in ansible are there to support services like cloud and CLI
- Eg. Python API is used for Command Line Interface





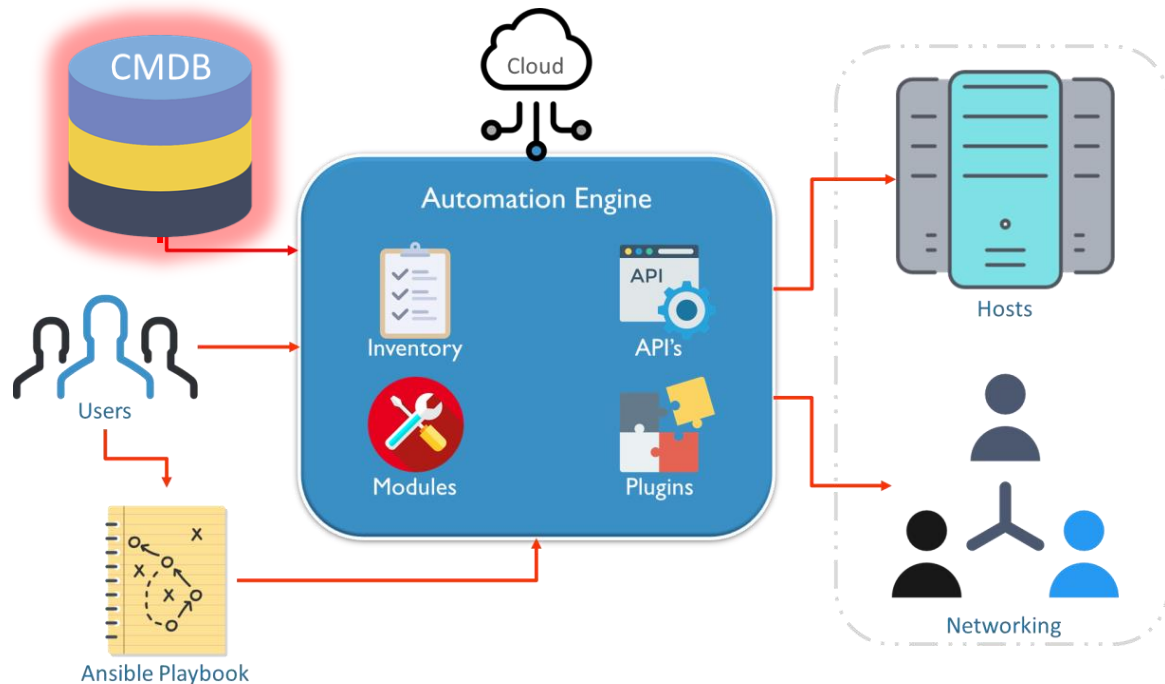
# Ansible Architecture – Automation Engine

- Plugins provide extra functionality to ansible
- Eg. Action plugin lets you perform tasks on your ansible machine before you execute a playbook



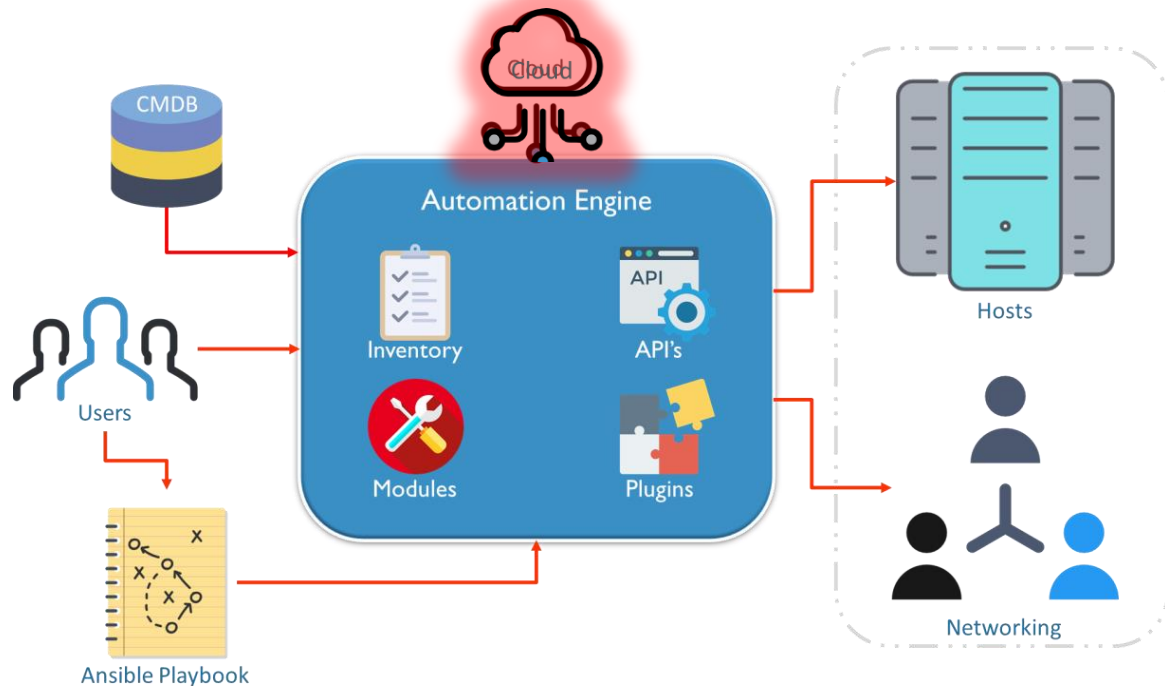
# Ansible Architecture – CMDB

- The configuration management keeps a record of all the configuration changes made in the system
- This database could be kept on the cloud platform or on the physical server as well



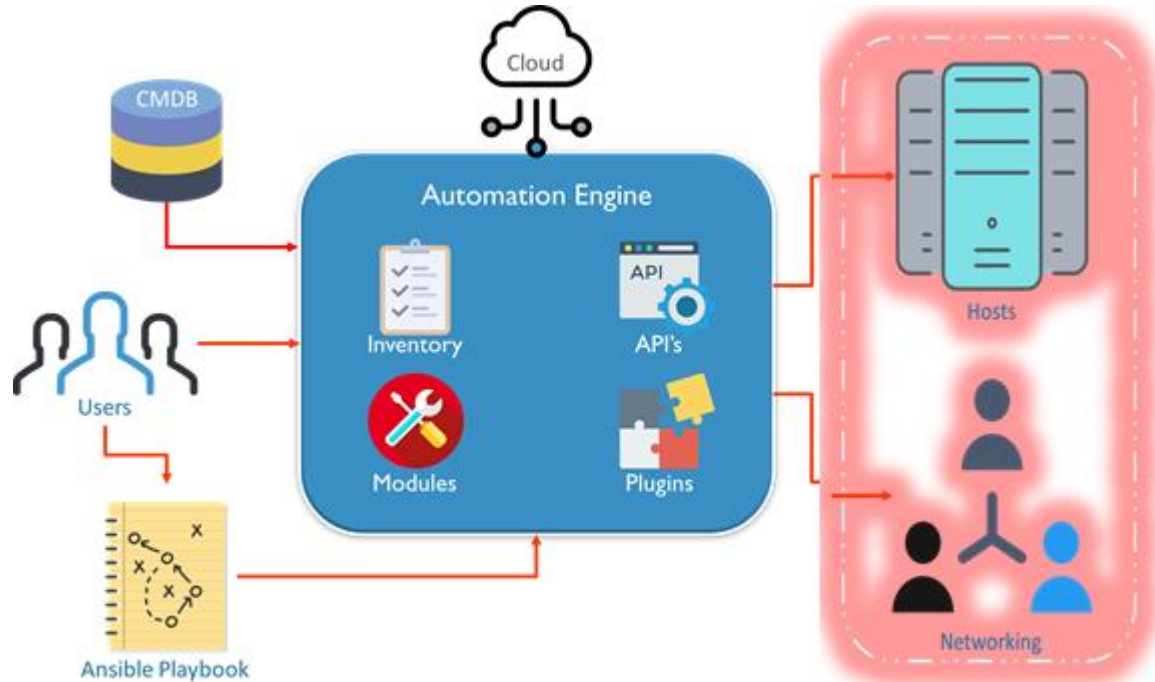
# Ansible Architecture – Cloud

- Ansible has a huge support for numerous cloud applications
- These cloud applications can seamlessly be integrated with ansible
- Ansible makes provisioning cloud infrastructures easy



# Ansible Architecture – Hosts & Networks

- The hosts are connected to the ansible system via secure SSH connection (though other encryption technologies can also be used).
- Different networks can be managed together giving each network separate access rights.



## Exercise 1: Ansible Installation

- Ansible requires a secure transport to nodes, hence openssh(linux/unix) or winrm(windows) must be setup on each machine
- Latest python libraries must be installed on the ansible machine

Ansible's inventory lists all the platforms you want to automate across. Ansible can at a single instance work on multiple hosts in the infrastructure. It is also possible to have multiple inventory files at the same time.

- The host inventory file can contain host names either individually or in groups
- Host Groups can be created by giving a group name within square brackets
- Group members can then be listed under, till there is a line brake

- Modules are units which actually get the work done in ansible. Ansible's modules are used to control system resources, like services, packages, execute system commands, or files
- There are over 450 in-built modules in ansible



- Modules abstract system tasks, like dealing with packages or handling files from ansible servers
- Users can choose to either use a module from ansible's in-built library or create their own custom modules
- Almost all of the modules support taking arguments in key-value format

## Exercise 2: Ansible Modules

- Run ad-hoc commands to use Ansible modules

- Playbooks are the access point to ansible provisioning
- It's the Ansible's way of deploying and configuring different remote servers and environments
- It is written in **YAML(Yet Another Mark-up Language)**
- On an advanced level playbooks can be used to
  - Handle multi-tier rollouts
  - Load balancing tasks for the servers

# Ansible Playbook Structure

- Besides the basic structure of a YAML file there are a few things to be kept in mind before writing a playbook

```
---
- hosts: all
  become: true
  vars:
    ansible_become_pass: service
  tasks:
    - name: install ntp
      apt:
        pkg: ntp
        state: present
      notify:
        - run update
    - name: install vim
      apt:
        pkg: vim
        state: present
    - name: start ntp service
      service:
        name: ntp
        state: started
        enabled: true
  handlers:
    - name: run update
      apt:
        update_cache: yes
```

# Ansible Playbook Structure

- All playbooks(YAML files) start with three hyphens in the top left corner

```
---  
- hosts: all  
  become: true  
  vars:  
    ansible_become_pass: service  
  tasks:  
    - name: install ntp  
      apt:  
        pkg: ntp  
        state: present  
      notify:  
        - run update  
    - name: install vim  
      apt:  
        pkg: vim  
        state: present  
    - name: start ntp service  
      service:  
        name: ntp  
        state: started  
        enabled: true  
  handlers:  
    - name: run update  
      apt:  
        update_cache: yes
```

## Ansible Playbook Structure : Hosts

- Each play in a playbook starts with the host(s) the play is supposed to be executed on.
- Hosts also defines the start of the play
- There may be multiple plays in a single playbook

```
---
hosts: all
become: true
vars:
  ansible_become_pass: service
tasks:
  - name: install ntp
    apt:
      pkg: ntp
      state: present
    notify:
      - run update
  - name: install vim
    apt:
      pkg: vim
      state: present
  - name: start ntp service
    service:
      name: ntp
      state: started
      enabled: true
handlers:
  - name: run update
    apt:
      update_cache: yes
```


## Ansible Playbook Structure : Become

- Become here is used to become a different user than the one logged into the remote machine
- This is mainly done for Privilege Escalation
- For example gaining su access

```
---  
- hosts: all  
  become: true  
  vars:  
    ansible_become_pass: service  
  tasks:  
    - name: install ntp  
      apt:  
        pkg: ntp  
        state: present  
      notify:  
        - run update  
    - name: install vim  
      apt:  
        pkg: vim  
        state: present  
    - name: start ntp service  
      service:  
        name: ntp  
        state: started  
        enabled: true  
  handlers:  
    - name: run update  
      apt:  
        update_cache: yes
```

## Ansible Playbook Structure : Vars

- Variables are defined by using the vars keyword
- Playbooks are executed line by line hence variables need to be defined before their use in playbooks

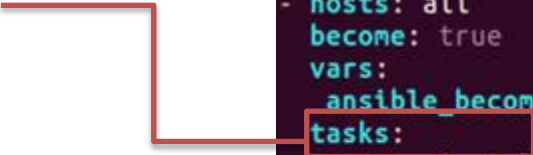


```
---  
- hosts: all  
  become: true  
  vars:  
    ansible_become_pass: service  
  tasks:  
    - name: install ntp  
      apt:  
        pkg: ntp  
        state: present  
      notify:  
        - run update  
    - name: install vim  
      apt:  
        pkg: vim  
        state: present  
    - name: start ntp service  
      service:  
        name: ntp  
        state: started  
        enabled: true  
  handlers:  
    - name: run update  
      apt:  
        update_cache: yes
```



## Ansible Playbook Structure : Tasks

- All the tasks that are to be executed on remote systems are defined in under the tasks section
- A single play can consist multiple tasks
- Tasks are executed in order



```
---  
- hosts: all  
  become: true  
  vars:  
    ansible_become_pass: service  
  tasks:  
    - name: install ntp  
      apt:  
        pkg: ntp  
        state: present  
      notify:  
        - run update  
    - name: install vim  
      apt:  
        pkg: vim  
        state: present  
    - name: start ntp service  
      service:  
        name: ntp  
        state: started  
        enabled: true  
  handlers:  
    - name: run update  
      apt:  
        update_cache: yes
```

## Ansible Playbook Structure : Handlers

- Tasks may contain notify actions which are triggered at the end of a block
- Notify triggers are called upon tasks in the handler section
- Handlers are tasks which are executed only once for

```
---  
- hosts: all  
  become: true  
  vars:  
    ansible_become_pass: service  
  tasks:  
    - name: install ntp  
      apt:  
        pkg: ntp  
        state: present  
        notify:  
          - run update  
    - name: install vim  
      apt:  
        pkg: vim  
        state: present  
    - name: start ntp service  
      service:  
        name: ntp  
        state: started  
        enabled: true  
  handlers:  
    - name: run update  
      apt:  
        update_cache: yes
```

## Exercise 3: Write a Playbook

- Write Ansible Playbook to install Apache

## Exercise 4: Write a Playbook Using Variables

- Write Ansible Playbook using variables

Overtime working with ansible a user may create hundreds of playbooks, variables, templates, defaults etc. Roles allow users to group this logic into an organized manner making reusability and sharing of ansible structure easier

- Overtime working with ansible a user may create hundreds of playbooks, variables, templates, defaults etc.
- Roles allow users to group this logic into an organized manner making reusability and sharing of ansible structure easier.

- Roles uses directories to structure and group all the playbooks, variables, templates, tasks, handlers, files, and defaults.
- This collected logic can be grouped in any way the user wants, for example you can group server specific roles together
- These roles can then be used inside playbooks and even as in-line commands

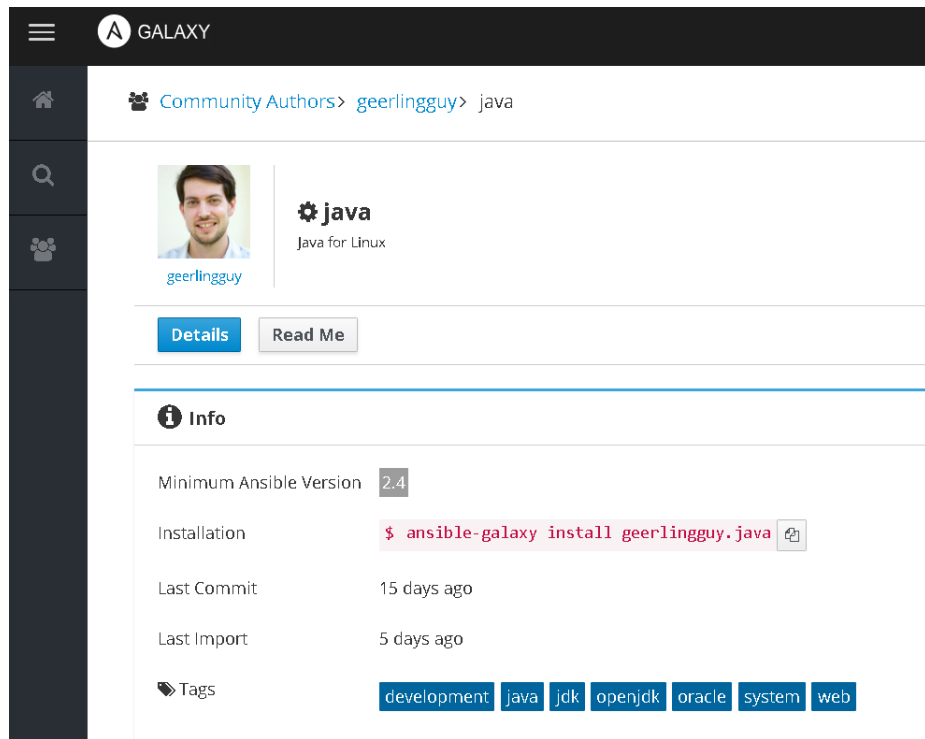
- Galaxy provides pre-packaged units of work known to Ansible as roles
- Roles can be dropped into Ansible PlayBooks and immediately put to work
- To create a Ansible roles, use ansible-galaxy command which has the templates to create it
- This will create it under the default directory /etc/ansible/roles and do the modifications else we need to create each directories and files manually

```
# ansible-galaxy init /etc/ansible/roles/apache --offline
```

- where, ansible-glaxy is the command to create the roles using the templates
- init is to initilize the role
- apache is the name of role
- offline - create offline mode rather than getting from online repository




- You can also download Ansible roles from <https://galaxy.ansible.com/>



The screenshot shows the Ansible Galaxy interface. At the top, there's a navigation bar with a hamburger menu, the 'A GALAXY' logo, and a home icon. Below the navigation bar, the breadcrumb path 'Community Authors > geerlingguy > java' is displayed. The main content area features a profile picture of geerlingguy, a gear icon, and the role name 'java' with the description 'Java for Linux'. Below this, there are two buttons: 'Details' (highlighted in blue) and 'Read Me'. A section titled 'Info' with an information icon contains the following details: 'Minimum Ansible Version' is 2.4; 'Installation' shows the command '\$ ansible-galaxy install geerlingguy.java' with a copy icon; 'Last Commit' is 15 days ago; 'Last Import' is 5 days ago; and 'Tags' includes development, java, jdk, openjdk, oracle, system, and web.

Community Authors > geerlingguy > java


 **java**  
Java for Linux

geerlingguy

**Details** Read Me

**i Info**

Minimum Ansible Version 2.4

Installation `$ ansible-galaxy install geerlingguy.java` 

Last Commit 15 days ago

Last Import 5 days ago

Tags development java jdk openjdk oracle system web

## Exercise 5 : Ansible Roles

- Write Ansible role to install Apache

- If you have a large playbook it may become useful to be able to run a specific part of the configuration without running the whole playbook
- Both plays and tasks support a “tags:” attribute for this reason
- You can ONLY filter tasks based on tags from the command line with --tags or -skip-tags
- Adding “tags:” in any part of a play (including roles) adds those tags to the contained tasks

- If you wanted to just run the “configuration” and “packages” part of a very long playbook, you could do this:

```
ansible-playbook example.yml --  
tags "configuration,packages"
```

- On the other hand, if you want to run a playbook without certain tasks, you could do this:

```
ansible-playbook example.yml --  
skip-tags "notification"
```

tasks:

- yum:  
    name: "{{ item }}"  
    state: installed  
loop:
  - httpd
  - memcached
- tags:
  - packages
- template:  
    src: templates/src.j2  
    dest: /etc/foo.conf  
tags:
  - configuration

- PyYAML is a YAML parser and emitter for Python
- PyYAML features
  - a *complete* [YAML 1.1](#) parser
  - The parsing algorithm is simple enough to be a reference for YAML parser implementors
  - Unicode support including UTF-8/UTF-16 input/output and \*
  - low-level event-based parser and emitter API (like SAX)
  - high-level API for serializing and deserializing native Python objects
  - support for all types from the [YAML types repository](#).
  - A simple extension API is provided
  - both pure-Python and fast [LibYAML](#)-based parsers and emitters
  - relatively sensible error messages

## Py YAML Requirement

- PyYAML requires Python 2.7 or Python 3.4+
- The current stable release of PyYAML: 3.13
- It can be downloaded from

<https://pyyaml.org/wiki/PyYAML>



**THANK YOU**