Configuration Management Automation Using Ansible

By - Yathish Nagaraj
Infra Dev Specialist
Automation PS Team

Introduction to Ansible

- Why Ansible?
- What is Ansible?
- Ansible Vs other Config Management Tools
- Ansible Architecture
- Playbook
- Inventory
- Working of Ansible
- Ansible Tower

Why Ansible?

This is Mr. Nags, a system administrator.
He is responsible for his company’s infrastructure

Why Ansible?

Apache

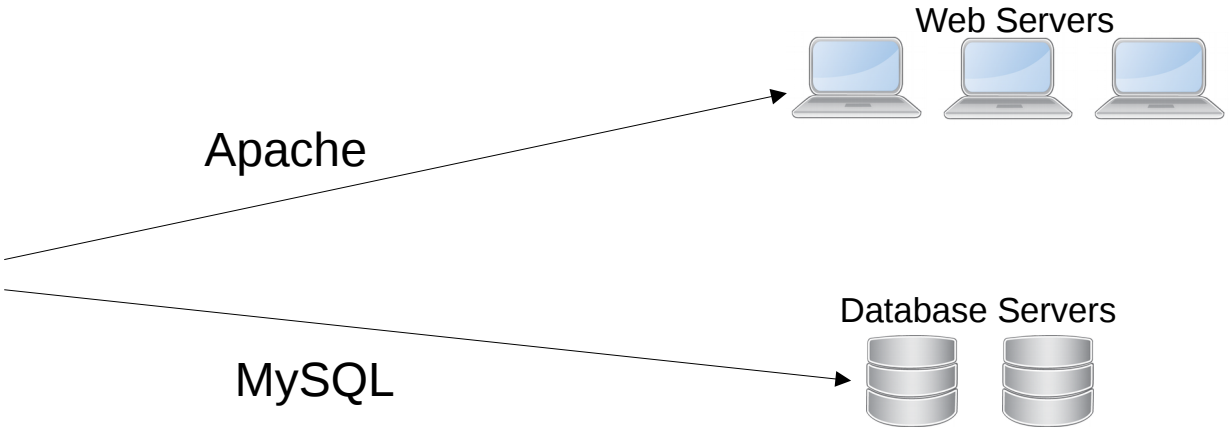
MySQL

Web Servers

Database Servers

He must install Apache on all 3 Web servers and MySQL on the 2 Database servers

Why Ansible?



That’s easy! Wouldn’t take much time either

Why Ansible?

Apache

MySQL

But what if the number of servers increase?

Why Ansible?

Apache

MySQL

The same task must be repeated multiple times.
Moreover, humans are prone to make mistakes.

Why Ansible?

Apache

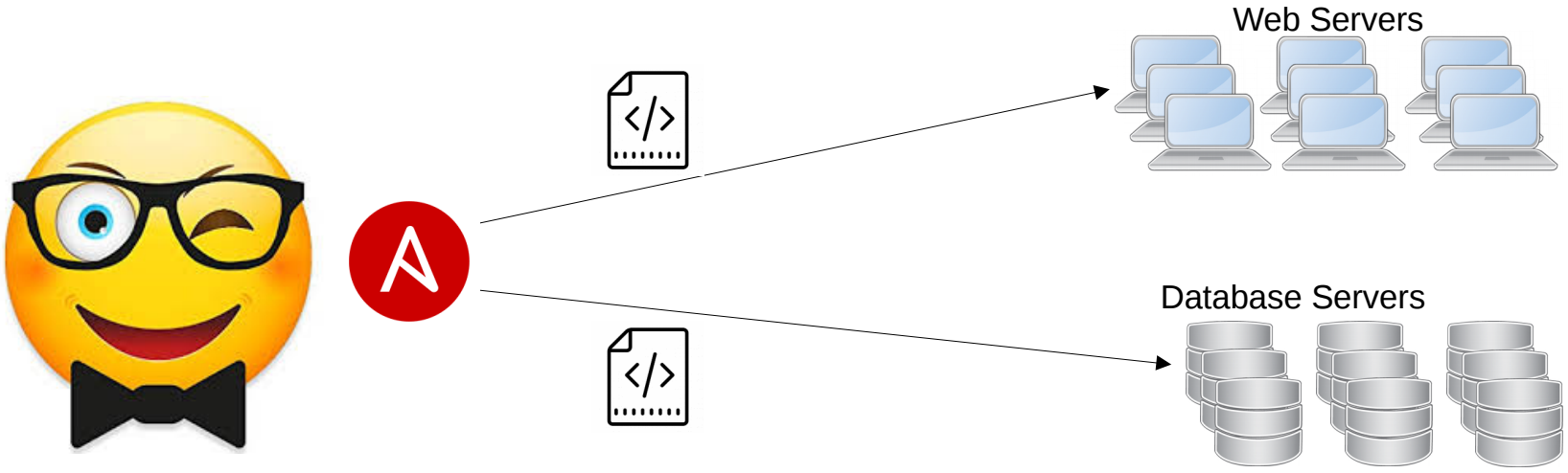
Web Servers

MySQL

Database Servers

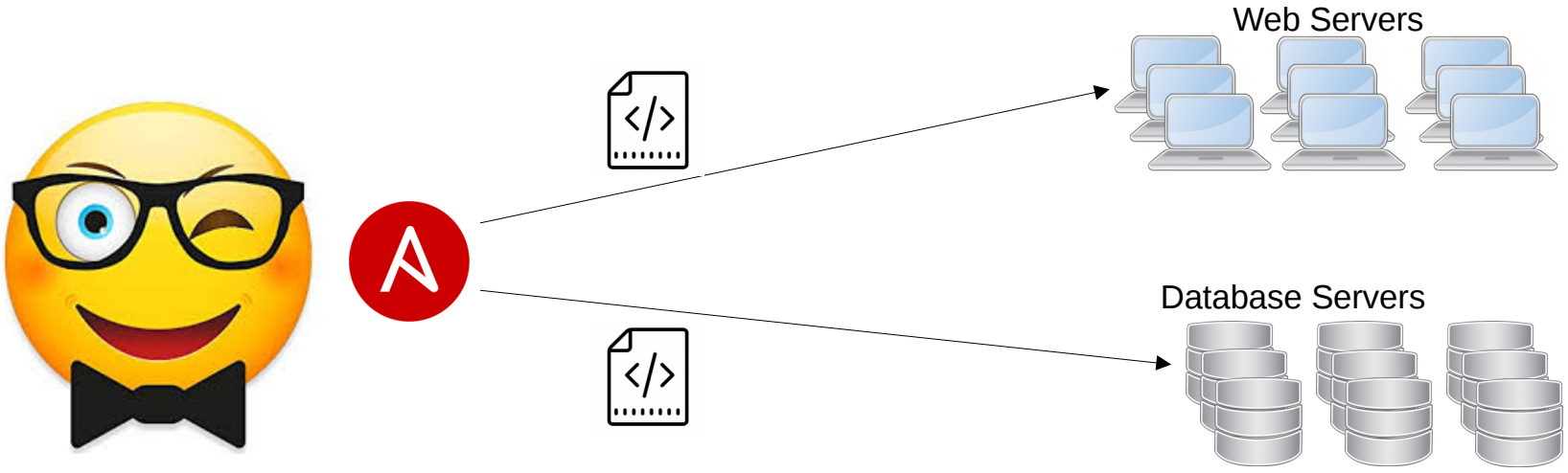
This is where Ansible comes to the rescue

Why Ansible?



With Ansible, a code is written once for the installation and deployed multiple times

Why Ansible?



Mr. Nags can now work on more productive tasks rather than repetitive once

What is Ansible?

Sounds great, right?
But what is Ansible?

What is Ansible?

Ansible is a tool that provides:

IT automation

Instructions are written to automate the IT professionals work

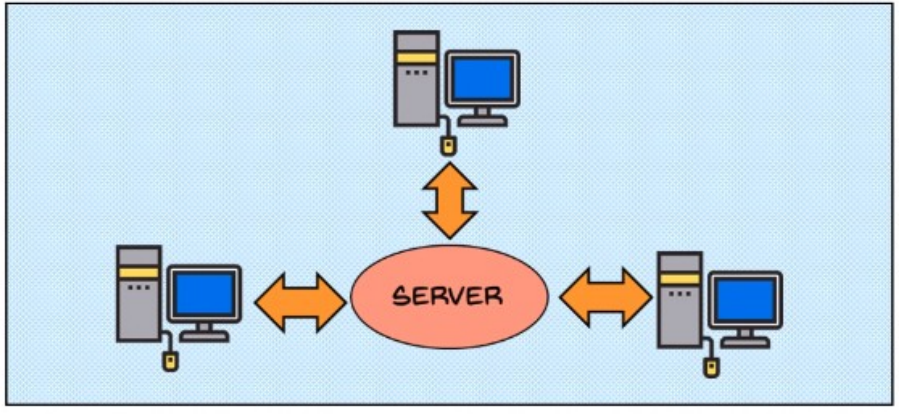
Configuration management

Consistence of all Systems in the Infrastructure is maintained

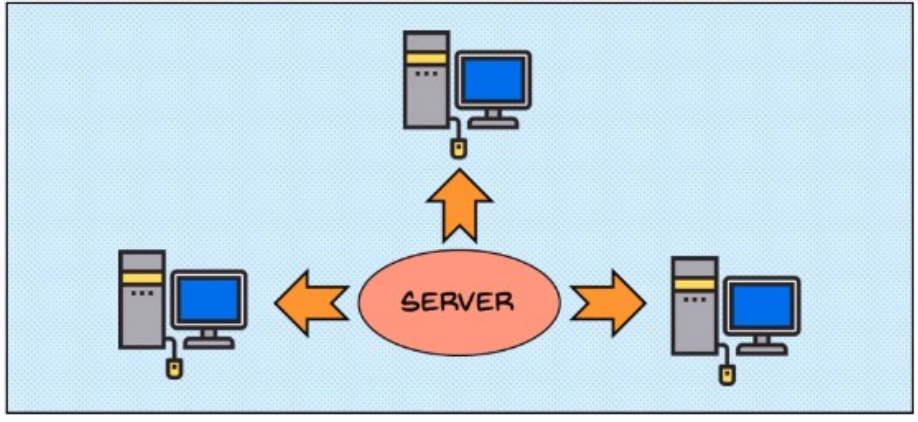
Automatic deployment

Applications are deployed automatically on a variety of environments

Ansible Vs other Config Management Tools

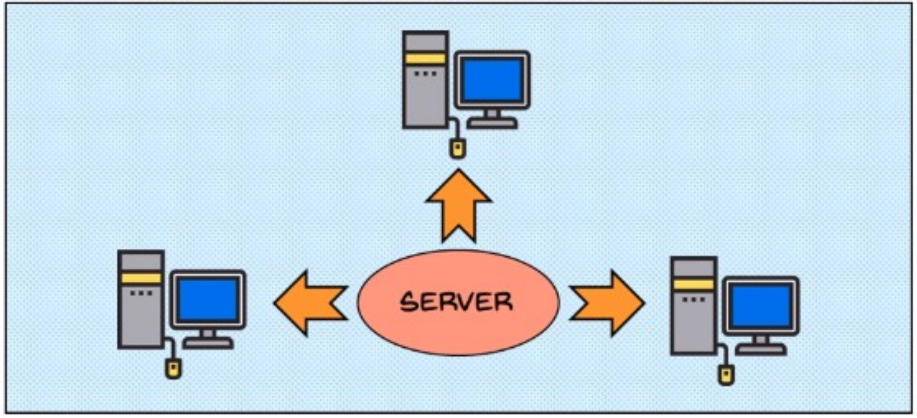


Pull Configuration: Nodes check with the server periodically and fetch the configurations from it



Push Configuration: Server pushes the configurations to the nodes

Ansible Vs other Config Management Tools



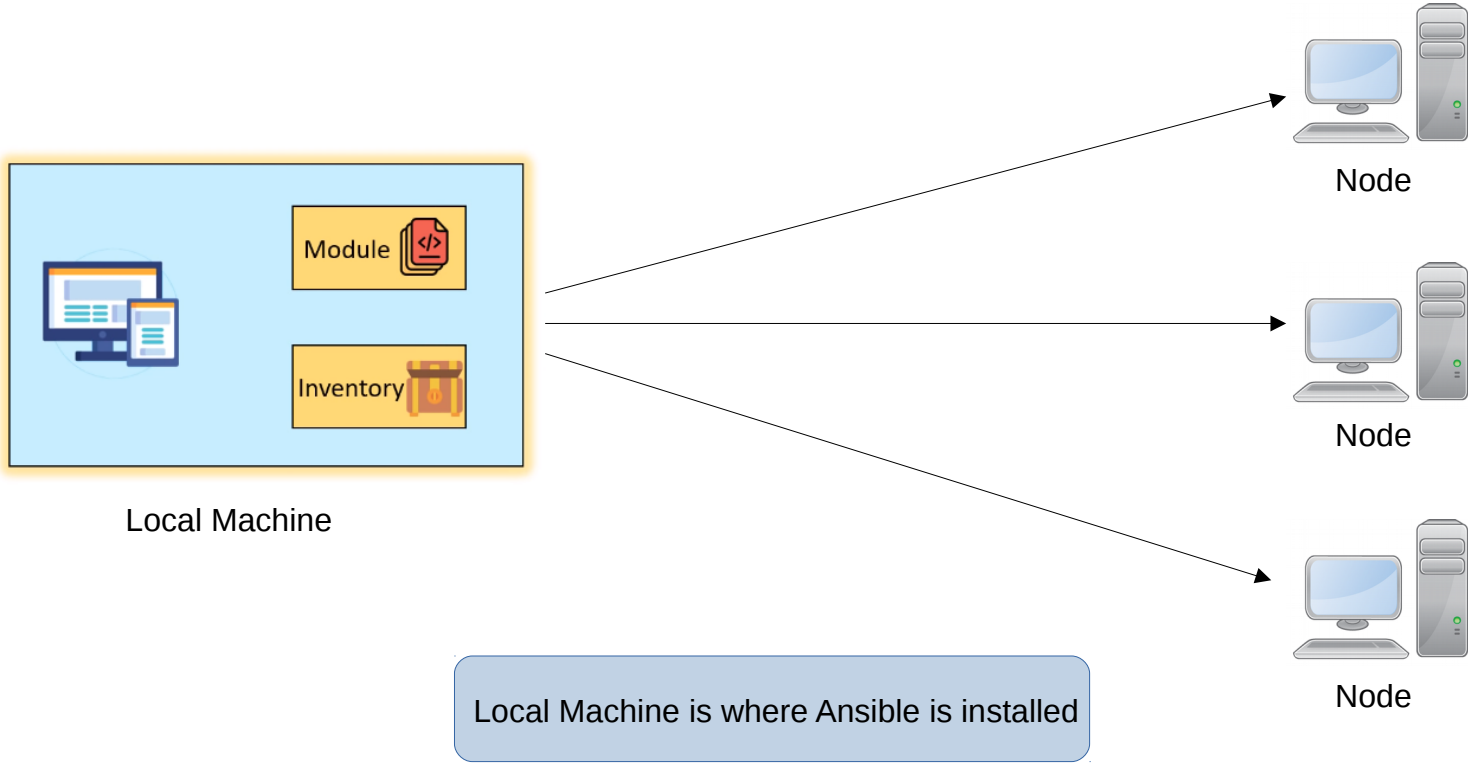
Unlike Chef and Puppet, Ansible is push type configuration management tool

Push Configuration: Server pushes the configurations to the nodes

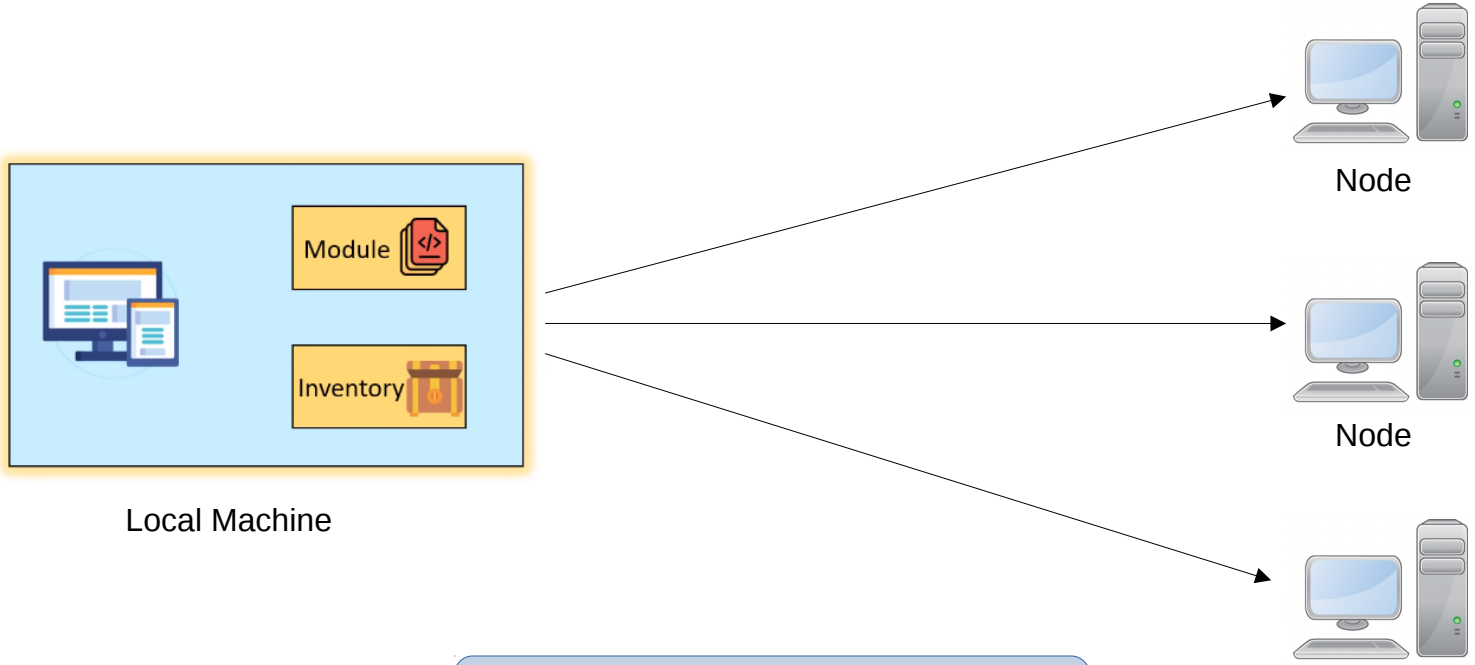
Ansible Vs other Config Management Tools

	CHEF	puppet	ANSIBLE	SALTSTACK
Architecture	Client-Server	Client-Server	Client Less	Client-Server
Ease of Setup	Complex	Moderate	Very easy	Moderate
Language	Ruby DSL	Puppet DSL	YAML	Python
Scalability	Scalable	Scalable	Scalable	Scalable
Management	Tough as it requires one to learn Ruby DSL	Tough as it requires one to learn Puppet DSL	Very easy	Very easy
Interoperability	High	High	High	High
Communication	Knife Tool	SSL	SSH	SSH

Ansible Architecture

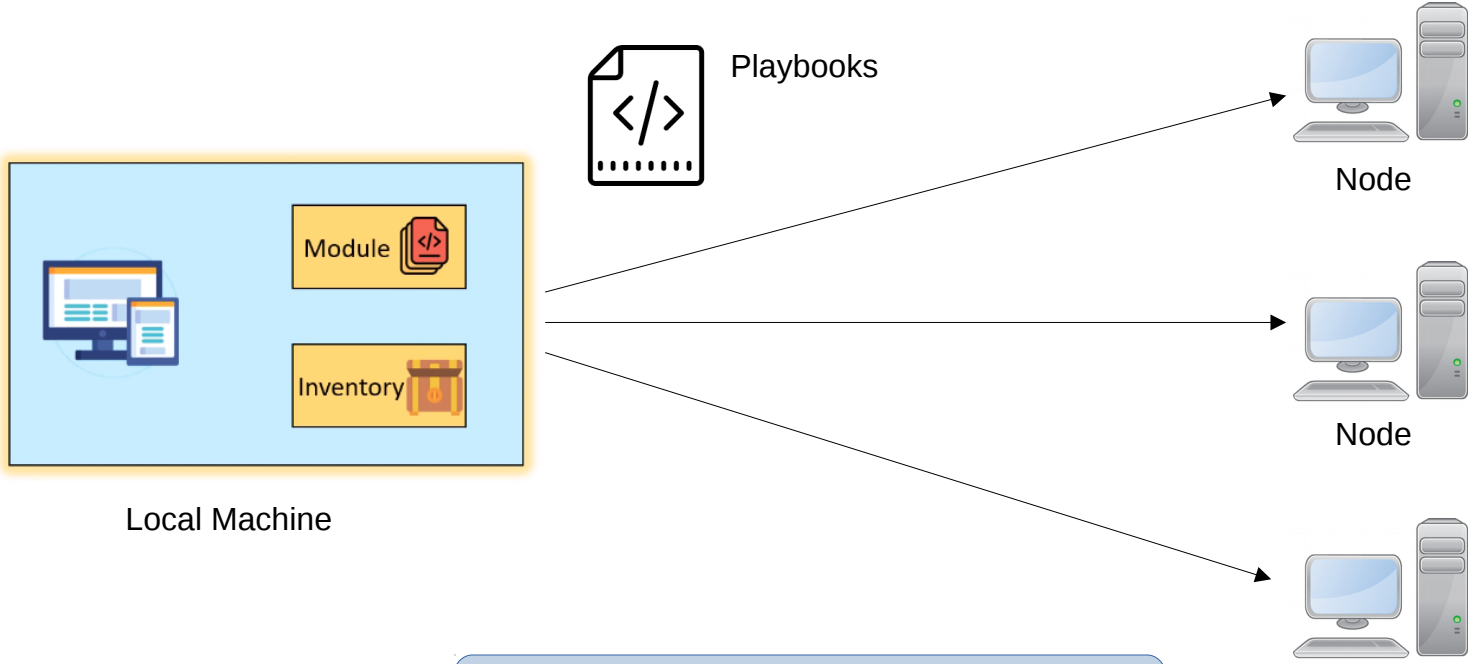


Ansible Architecture



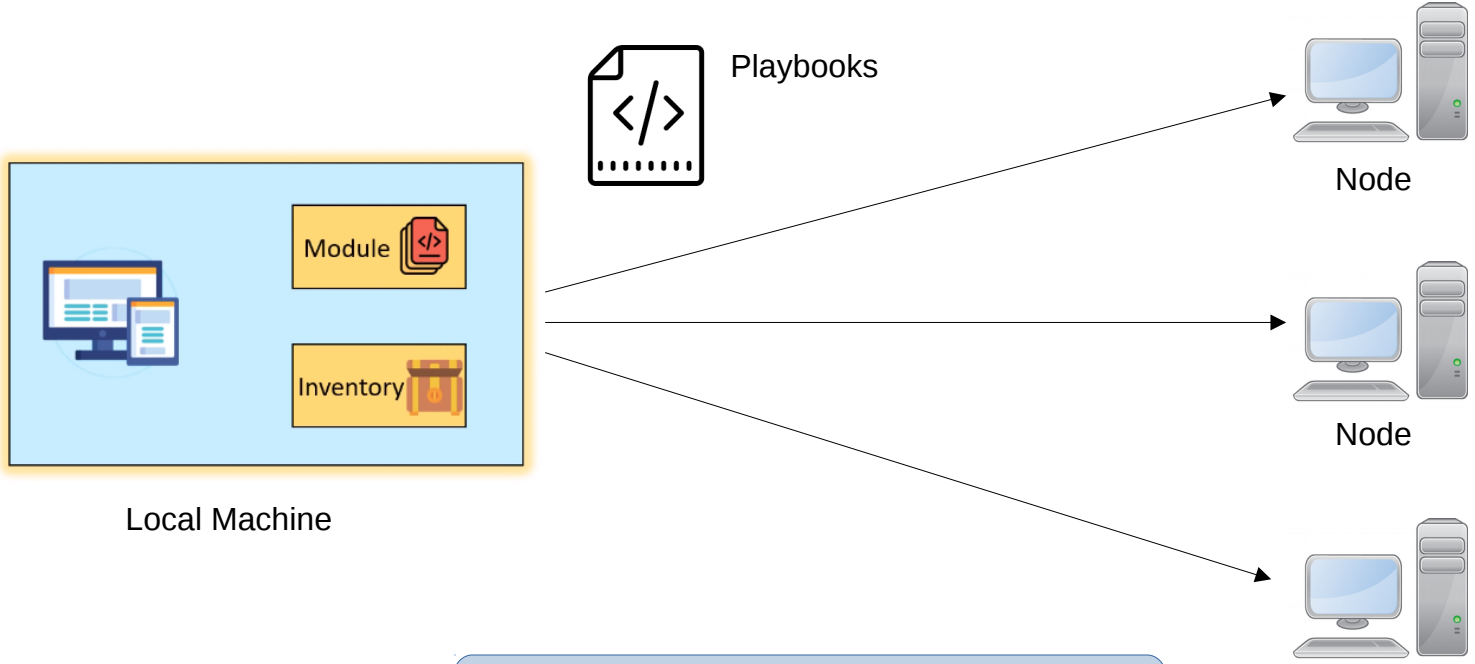
Nodes are the systems to be configured.
They are controlled by the local machine

Ansible Architecture



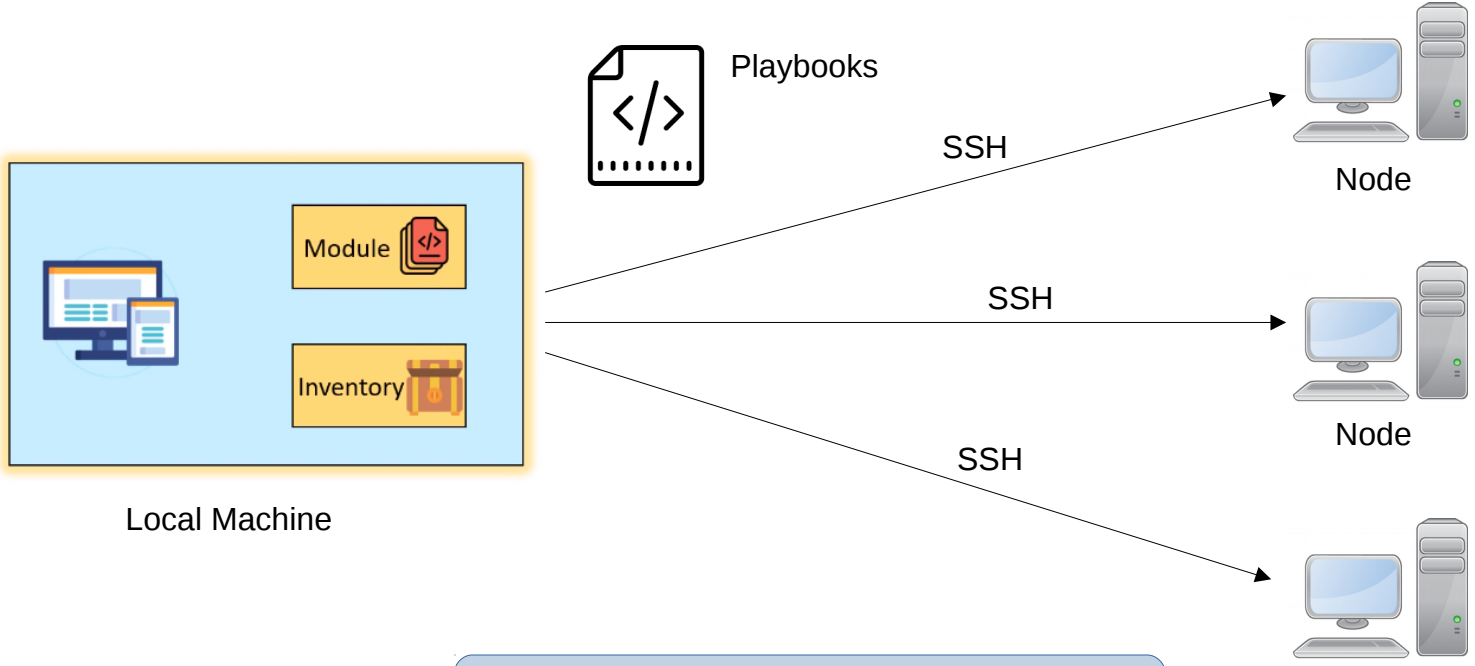
Module is a collection of configuration code files

Ansible Architecture



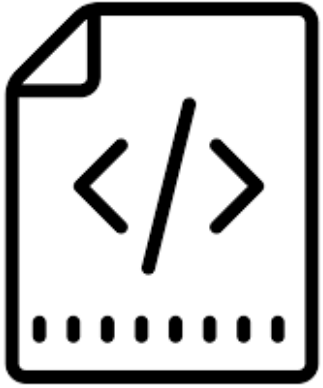
Inventory is a document that groups the nodes under specific labels

Ansible Architecture

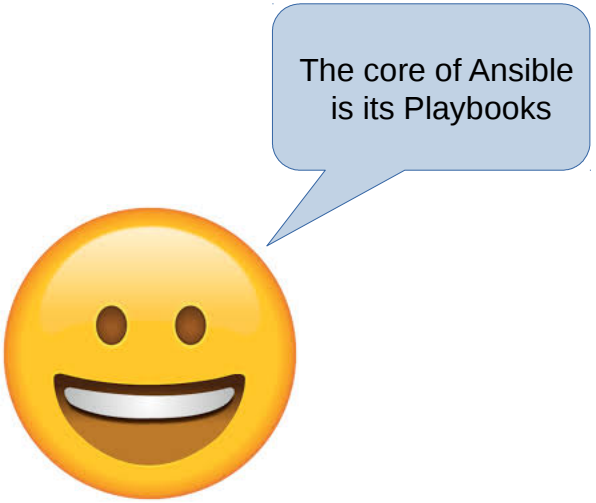


The local Machine connects to the nodes through a SSH Client

Playbook



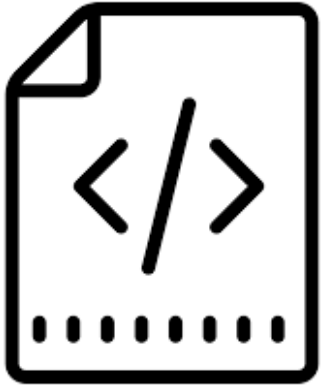
Playbook



Playbook

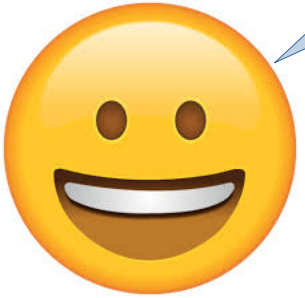
Playbooks are the instructions to configure the nodes

They are written in YAML, a Language used to describe data



Playbook

Did you know , YAML Stands for “YAML Ain’t Markup Language”



Playbook

```
---
-name: play 1
 hosts: webserver
 tasks:
   -name: install apache
     yum:
          name: apache
          state: present
   -name: start apache
     service:
           name: apache
           state: start
-name: play 2
 hosts: databaseserver
 tasks:
   -name: install MySQL
  yum:
      name: MySQL
      state: present
```

Let’s have a look at the structure of a Playbook

Playbook

```
---
-name: play 1
 hosts: webserver
 tasks:
  -name: install apache
   yum:
        name: apache
        state: present
  -name: start apache
   service:
         name: apache
         state: start
-name: play 2
 hosts: databaseserver
 tasks:
  -name: install MySQL
  yum:
      name: MySQL
      state: present
```

Playbook begins with "---"

Playbook

```
---
-name: play 1
 hosts: webserver
 tasks:
  -name: install apache
   yum:
        name: apache
        state: present
  -name: start apache
   service:
         name: apache
         state: start
-name: play 2
 hosts: databaseserver
 tasks:
  -name: install MySQL
  yum:
      name: MySQL
      state: present
```

A Playbook is a list of Plays

Playbook

```
---
-name: play 1
 hosts: webserver
 tasks:
  -name: install apache
   yum:
       name: apache
       state: present
  -name: start apache
   service:
        name: apache
        state: start
-name: play 2
 hosts: databaseserver
 tasks:
  -name: install MySQL
  yum:
     name: MySQL
     state: present
```

Host is the target for the Play

Playbook

```
---
-name: play 1
 hosts: webserver
 tasks:
  -name: install apache
   yum:
        name: apache
        state: present
  -name: start apache
   service:
         name: apache
         state: start
-name: play 2
 hosts: databaseserver
 tasks:
  -name: install MySQL
  yum:
      name: MySQL
      state: present
```

Each play has a list of tasks

Playbook

```
---
-name: play 1
 hosts: webserver
 tasks:
  -name: install apache
    yum:
        name: apache
        state: present
  -name: start apache
    service:
        name: apache
        state: start
-name: play 2
 hosts: databaseserver
 tasks:
  -name: install MySQL
   yum:
      name: MySQL
      state: present
```

Each element in the list of tasks is given a name

Playbook

```
---
-name: play 1
 hosts: webserver
 tasks:
  -name: install apache
    yum:
        name: apache
        state: present
  -name: start apache
    service:
        name: apache
        state: start
-name: play 2
 hosts: databaseserver
 tasks:
  -name: install MySQL
   yum:
      name: MySQL
      state: present
```

The name is followed by instructions to execute the task

Playbook

```
---
-name: play 1
 hosts: webserver
 tasks:
   -name: install apache
     yum:
          name: apache
          state: present
   -name: start apache
     service:
           name: apache
           state: start
-name: play 2
 hosts: databaseserver
 tasks:
   -name: install MySQL
  yum:
      name: MySQL
      state: present
```

Simple isn't it?

Playbook

```
---
-name: play 1
 hosts: webserver
 tasks:
  -name: install apache
   yum:
       name: apache
       state: present
  -name: start apache
   service:
        name: apache
        state: start
-name: play 2
 hosts: databaseserver
 tasks:
  -name: install MySQL
  yum:
     name: MySQL
     state: present
```

Inventory

```
[webserver]
web1.machine
web2.machine
web3.machine

[databaseserver]
db1.machine
```

An Inventory file classifies nodes into groups

Inventory

```
[webserver]
web1.machine
web2.machine
web3.machine

[databaseserver]
db1.machine
```

We have two groups Here: “webserver” and “databaseserver”

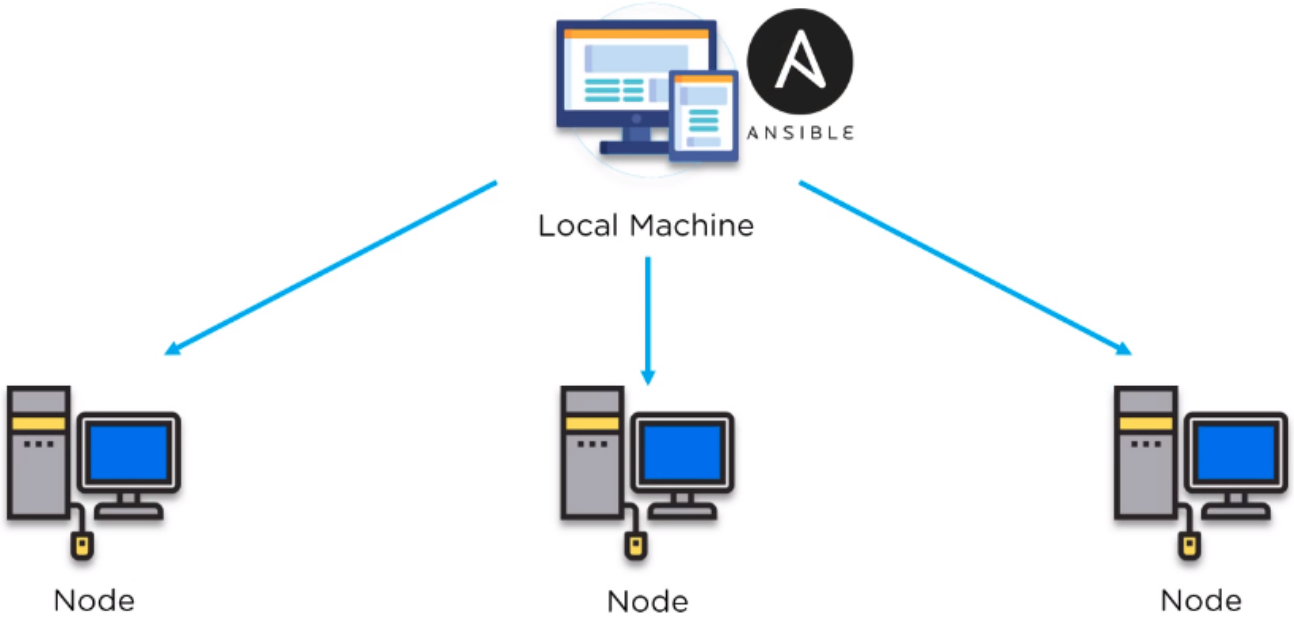
Inventory

```
[webserver]
web1.machine
web2.machine
web3.machine

[databaseserver]
db1.machine
```

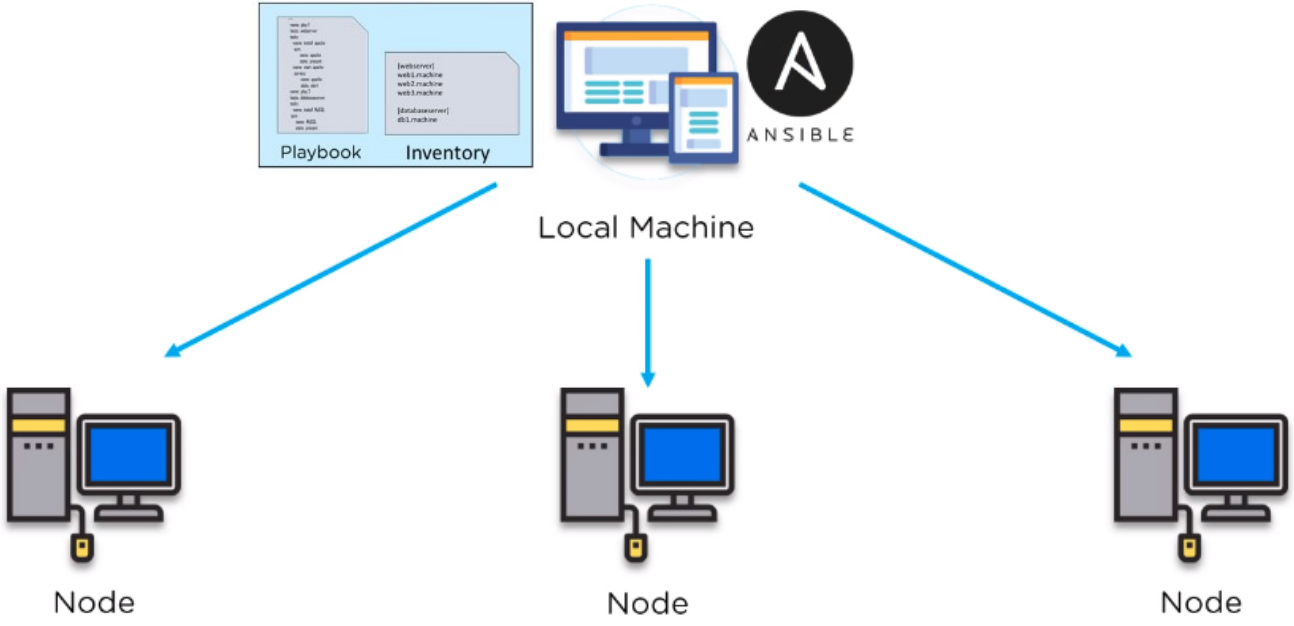
The hostnames of the Nodes are specified under the group name

Working of Ansible



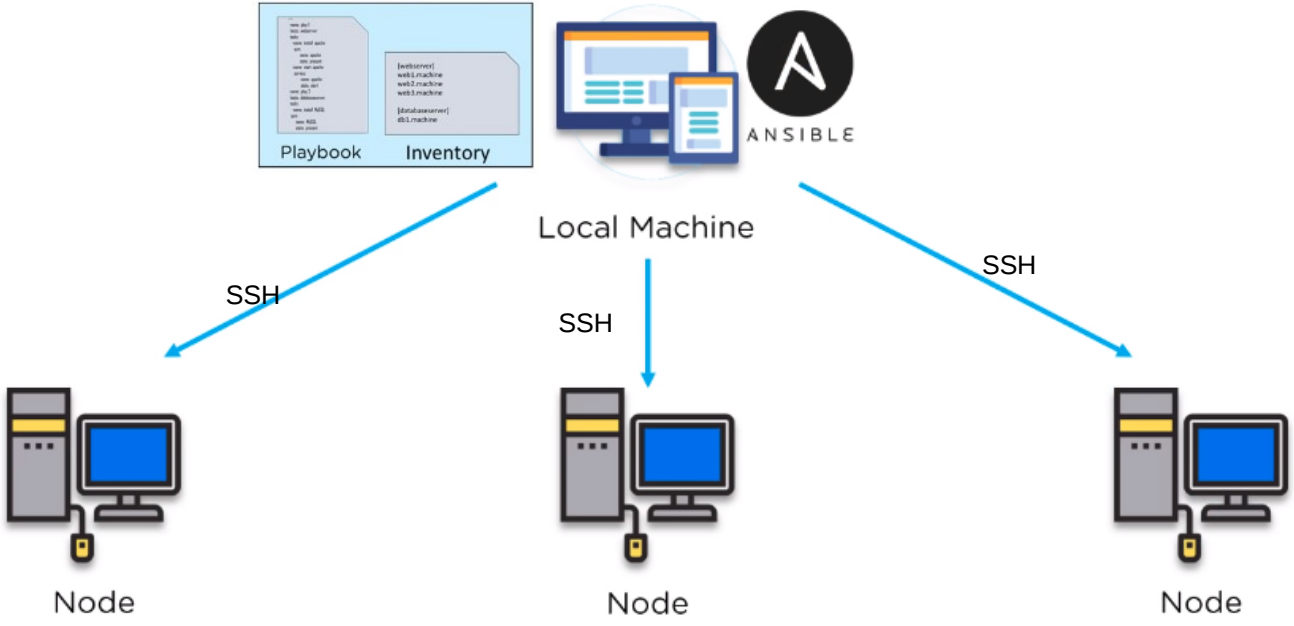
Ansible is installed only on the local machine.
This makes Ansible agent-less

Working of Ansible



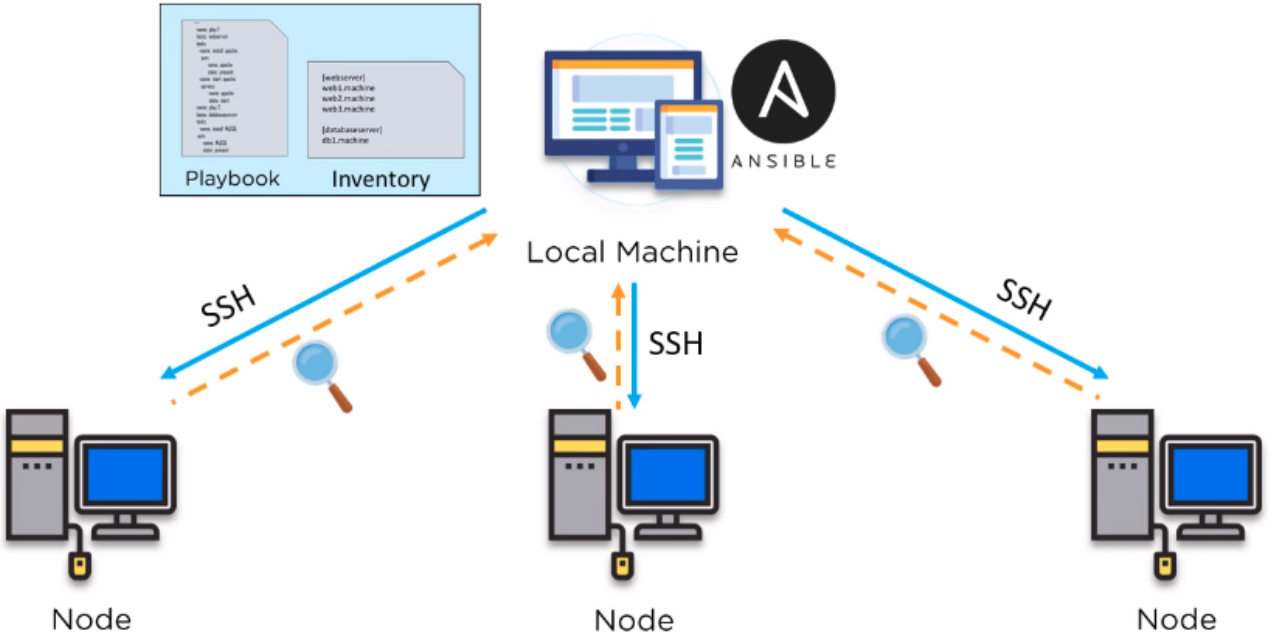
The playbook and inventory are written
at the local machine.

Working of Ansible



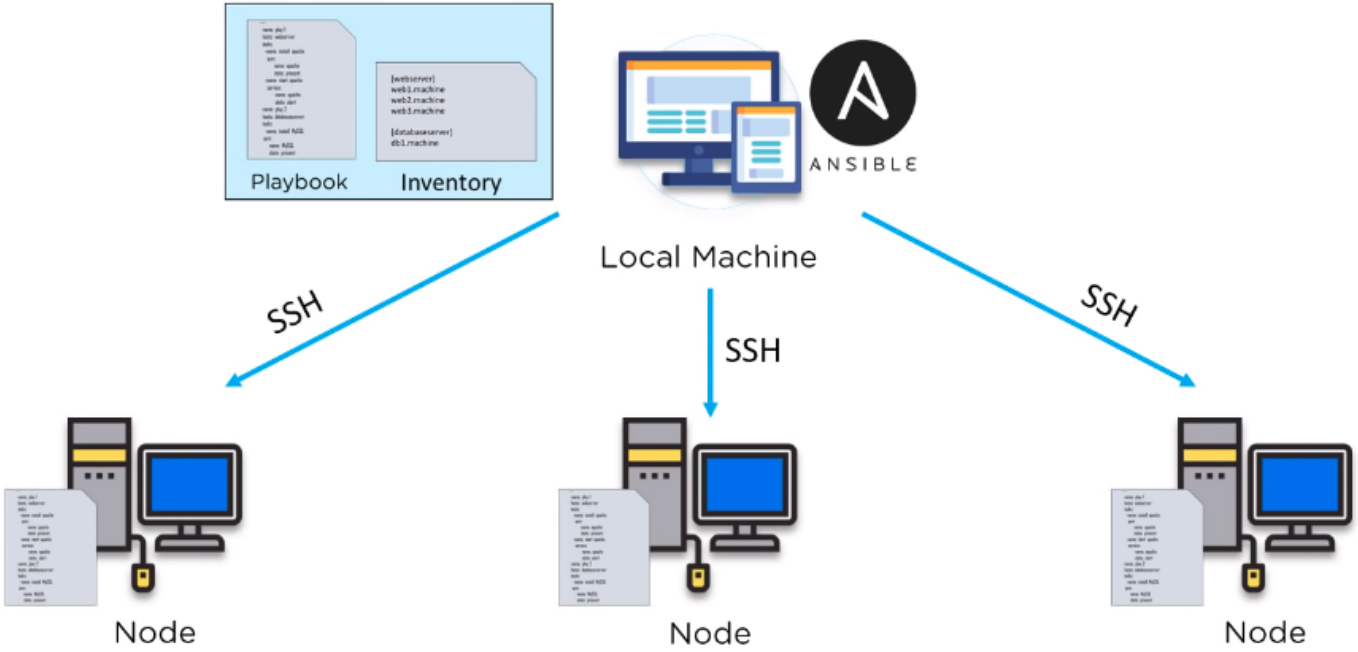
The local machine connects to the nodes through SSH

Working of Ansible



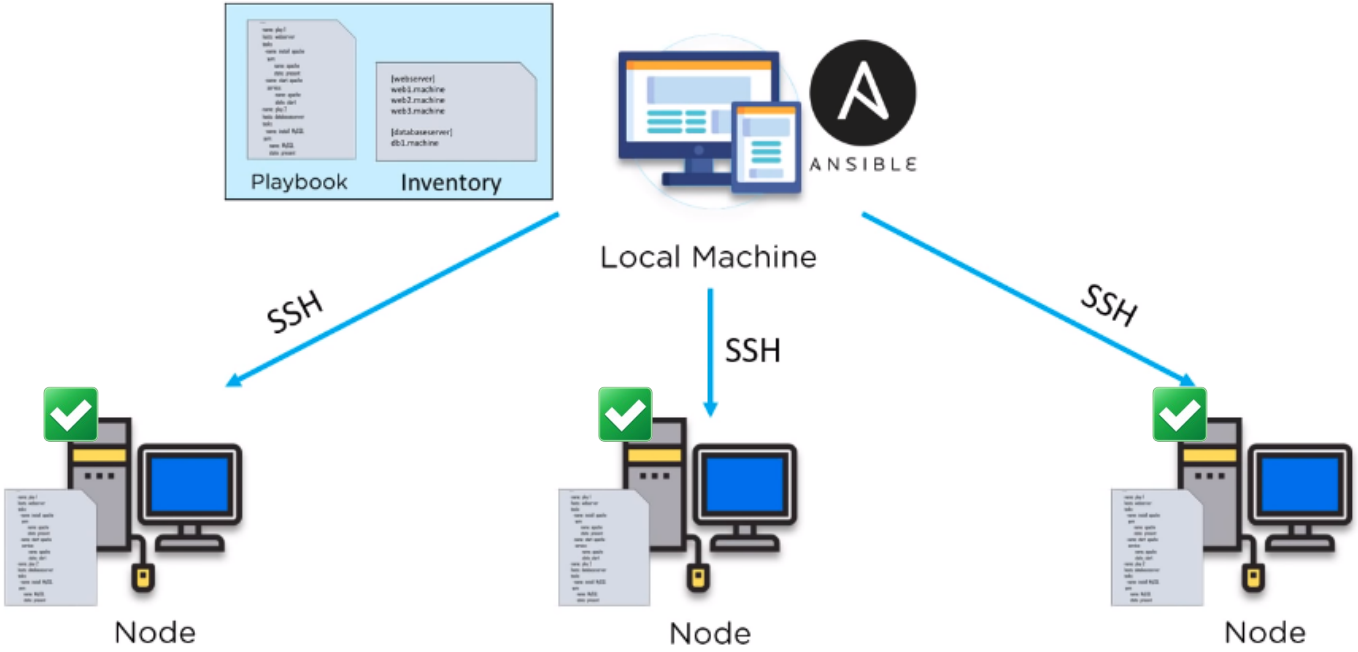
The local machine gathers the facts of each Node. Facts indicate the state of the nodes

Working of Ansible



The Playbooks are sent to nodes

Working of Ansible



The Playbooks are now executed. This configures the nodes to their desired states

Ansible Tower

Now, let's have a look at the icing on the cake – **Ansible Tower**

Red Hat
Ansible
Tower

Ansible Tower

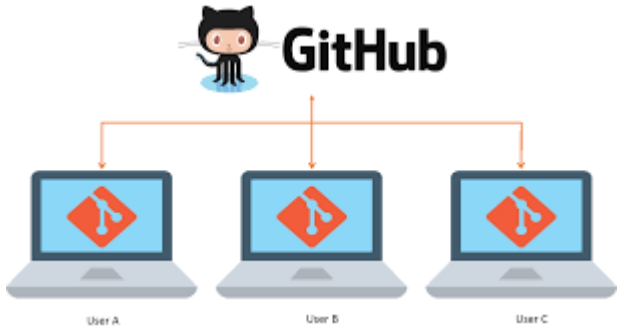
Red Hat Ansible Tower

Ansible Tower is a framework for Ansible

It provides a GUI. Thus, reducing the dependency on the command prompt window

Instead of typing long commands, tasks can now be performed in a single click

Other Tools used to facilitate the training



- Git is a Version Controlling system (VCS) that helps software developers to work together and maintain complete history of their work.
- GitHub is a Git repository hosting service. GitHub provides a Web-based graphical interface. Its like a social network for developers
- Together Git and Github provides a complete Source Code Management (SCM) Solution

VAGRANT

- Vagrant is an open-source software product for building and maintaining portable virtual software development environments for VirtualBox, KVM, Hyper-V, Docker containers, VMware, and AWS.
- It tries to simplify software configuration management of virtualizations in order to increase development productivity.

Git Commands

To create a new local repository:

- git init

To download /clone an exiting repository in github:

- git clone “<github repository URL>”

To link local repository with the remote repository on github:

- git remote add origin “<github repository URL>”

To pull from remote github repository:

- git pull origin <branch name>

To check the status of local git repository:

- git status

To add a file to the local git repo index:

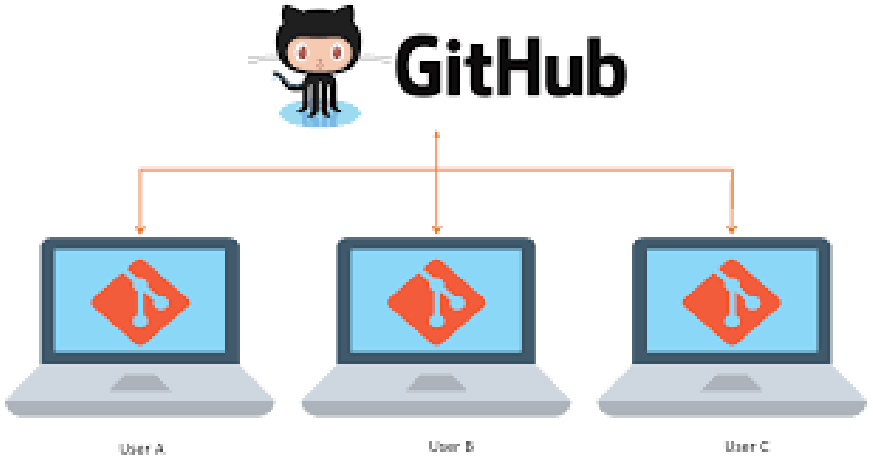
- git add -A or git add <file name>

To commit the files in local git repo:

- git commit -m “<commit message>”

To push the committed local repo into remote github repository:

- git push origin <branch name>



Vagrant Commands

To initializes a new Vagrant environment by creating a Vagrantfile:

- vagrant init

To start and provision the vagrant environment as defined in Vagrantfile:

- vagrant up

To output status of the vagrant machine/s:

- vagrant status or vagrant status <machine_name>

To stop a vagrant machine/s:

- vagrant halt or vagrant halt <machine_name>

To stop and delete all traces of the vagrant machine/s:

- vagrant destroy or vagrant destroy <machine_name>

To connect to machine via SSH:

- vagrant ssh <machine_name>

  note: vagrant by default takes Username as “vagrant” and Password as “vagrant”