Abstract

Ansible is a configuration management tool that is designed to automate controlling servers for administrators and operations teams. With Ansible we can use a single central server to control and configure many different remote systems using SSH and Python as only requirements. Designed for multi-tier deployments since day one, Ansible models our IT infrastructure by describing how all of our systems inter-relate, rather than just managing one system at a time.

Ansible carries out tasks on servers that it manages based on task definitions. These tasks invoke built-in and community maintained Ansible modules using small snippets of YAML for each task. As the number and variety of systems that we manage with a single Ansible control node become more complex, it makes sense to group tasks together into Ansible Playbooks. Using playbooks eliminates the need to run many individual tasks on remote systems, instead letting us configure entire environments at once with a single file.

It uses no agents and no additional custom security infrastructure, so it's easy to deploy - and most importantly, it uses a very simple language (YAML, in the form of Ansible Playbooks) that allow you to describe your automation jobs in a way that approaches plain English.

However, playbooks can become complex when they are responsible for configuring many different systems with multiple tasks for each system, so Ansible also lets us organize tasks in a directory structure called a **Role**. In this configuration, playbooks invoke roles instead of tasks, so you can still group tasks together and then reuse roles in other playbooks. Roles also allow you to collect templates, static files, and variables along with your tasks in one structured format.

Ansible releases a new major release of Ansible approximately three to four times per year. The core application evolves somewhat conservatively, valuing simplicity in language design and setup. However, the community around new modules and plugins being developed and contributed moves very quickly, adding many new modules in each release.

Table of Contents

|  |  |
| --- | --- |
| Contents | Page No. |
| Abstract | i |
| Acknowledgement | ii |
| Table of Contents | iii |

|  |  |  |
| --- | --- | --- |
| Chapter-1 | Introduction | 4 |
| Chapter-2 | Requirement Analysis & System Specification | 8 |
| Chapter-3 | System Design | 11 |
| Chapter-4 | Implementation, Testing & Maintenance | 15 |
| Chapter-5 | Conclusion & Future Scope | 16 |
| References | - | 16 |

Chapter-1

Introduction

**1.1 Introduction to Ansible**

Ansible is an IT automation tool. It can configure systems, deploy softwares, and orchestrate more advanced IT tasks such as CICD or zero downtime rolling updates. Ansible is easy to deploy because it does not use any agent or custom security infrastructure.

Ansible’s main goals are simplicity and ease-of-use. It also has strong focus on security and reliability, featuring a minimum of moving parts, usage of OpenSSH for transport, and a language that is designed around auditability by humans, even those not familiar with program.

* 1. **Project Category**

Ansible is an Industry-Automation tool which ease the efforts of system or network administrator. It is used to carry the tasks which are either used very often or need to execute in large number of systems. It also includes version control system which can be used to roll back the changes made in any number of systems. Ansible proved a very prominent tool in deployment phase because the deployment phase includes certain number tasks that need to be executed at every time. It can be done manually but is very time consuming. Therefore, DevOps Engineers prefer to adopt an approach which can ease their tasks. Luckily, Ansible can do those tasks and can save DevOps Engineers from headache of repeating same boring tasks.

* 1. **Objectives of Ansible**
* IT Industry Automation: Automate the tasks which are either repeated very often or need to execute in large number of systems
* Configuration Management: Manages the software and packages in any number of computers.
* Orchestration: Helps to define how multiple configurations interact and ensure the disparate pieces can be managed as a whole.
* Reduce Downtime: It aims to reduce downtime by executing the tasks at one go.
* Increase Productivity: Since the downtime is reduced, it increases productivity.

**1.4 Problem Formulation**

Ansible is an Automation Tool used for decreasing downtime and increasing productivity by allowing one to perform tasks such as orchestration, configuration management, software deployment, CICD or zero downtime rolling updates.

**1.5 Identification/Reorganization of Need**

There were already existing systems available as a substitute of ansible and still available but they aren’t as efficient as ansible. What makes ansible different from them is that ansible is open source, agentless and its security relies on ssh protocol which is one of the most secured existing protocols. Ansible is easy to install and configure, and the programming language it supports for execution of tasks is YAML which is human readable i.e. even a non-tech guy can read it and get an idea of what is happening.

**1.6 Existing System**

* Puppet Labs:

A long-standing tool in the CM space, Puppet has been tested and proven in some of the most demanding environments. It uses a customized DSL, master-client setup and a model-driven approach. Puppet code design works as a list of dependencies.

Pros:

* Well-established support community through Puppet Labs
* Mature interface and runs on nearly every OS
* Simple installation and initial setup
* Most complete Web UI in this space
* Strong reporting capabilities

Cons:

* Ruby-based CLI
* Ruby support is declining
* Code base can become complex
* Model-driven approach means less control compared to code-driven approaches
* SaltStack:

SaltStack is CLI-based and can be configured as a master-client or non-centralized model. Based in Python, SaltStack offers a push method and an SSH method of communication. Salt allows for grouping of clients and configuration templates to simplify environmental control.

Pros:

* Straightforward organization and usage
* Feature-rich DSL
* Consistent input, output and configs – all YAML
* Introspection is transparent
* High scalability and resiliency master model with minions and hierarchical tiers

Cons:

* Challenging setup for new users
* Documentation somewhat complex at introductory level
* Web UI is newer and less complete than other tool’s Web UIs in the space
* Not great support for non-Linux OS

* Chef:

Its command-line interface, testing mode, and large database make it ideal for companies that need large storage capacity. Chef is highly customizable, accommodating the installation and creation of different modules. That makes it one of the most adaptable CM solutions on the market.

Pros:

* Rich collection of modules and configuration recipes
* Code-driven approach provides control and flexibility configurations
* Git foundations provides strong version control capabilities
* “Knife” tool eases installation burdens

Cons:

* Steep learning curve for non-Ruby users
* Large code bases and complicated environments
* Does not support push functionality
* Ansible:

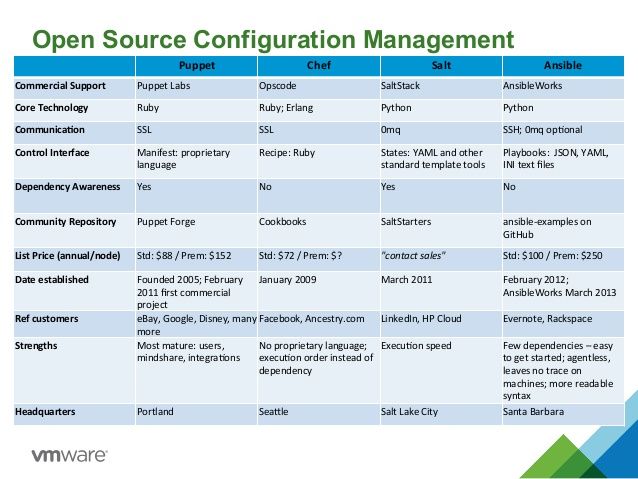
A simple solution that uses SSH. It offers several other services in addition to configuration management, such as workflow monitoring, automated app deployment for updates and more. Ansible, written in Python, has strong security features and focuses on five foundational principles: A small learning curve; ease of use; automation for most elements; efficiency and strong security.

Pros:

* SSH-based. No need for agents or remote nodes
* Easy learning curve thanks to the use of YAMLPlaybook structure is simple and clearly structured
* Variable registration feature enables tasks to register variables for later tasks
* Much more streamlined code base than some other tools

Cons:

* Less powerful than tools based in other programming languages
* Logic comes through its DSL, which requires frequent checking on documentation
* Difficult to see the values of variables within the playbooks
* No consistency between formats of input, output, and config files
* Struggles with performance speed at times



**1.7 Unique Features of Ansible**

* **Free**: Ansible is an open-source tool.
* **Set up & Use**: Ansible is very simple to set up and use.
* **Powerful**: Ansible lets you model even highly complex IT workflows.
* **Flexible**: You can orchestrate entire application environment no matter where it is deployed.
* **Agentless**: You don’t need to install any other software and firewall ports on the client systems.
* **Efficient**: Since you don’t need to install any extra software therefore, there’s more room for application resources on your server.

Chapter-2

Requirement Analysis & System Specification

**2.1 Feasibility Study**

**Technical:**

Ansible is a powerful and become a prominent tool in IT industry for tasks automation, orchestration, application deployment, CICD, rolling back changes and much more. Due to its flexibility and ease of usage it is widely used by many great organizations like NASA, Microsoft, BMW, etc. In an organization with large number of computers there is a need for a solution which can reduce the downtime and can increase productivity. Ansible offers very large number of modules for accomplishing any task. These modules are written in Python and Ruby. Ansible also supports plugins to leverage its strength and gives DevOps Engineers more time to focus on new things rather than wasting time on repeated tasks.

With great offerings, it has also some restrictions like when it comes to portability, it is not efficient in cross-platform. It works well in Linux/Unix environment but have some issues in windows environment. It has one more restriction i.e. control machine must be Linux/Unix machine, while remote machine could be anything. The error prompt method is not user-friendly, a noob in ansible might not be able to resolve the errors cast during execution of ansible-playbooks.

**Economical:**

Being an open-source tool, ansible is freely available but it also offers paid service i.e. Ansible Tower (used for monitoring purpose). Fortunately, we can do a lot without even ansible-tower. Since ansible is agentless and its installation does not require any extra set up therefore, it is an economical solution for tasks automation. All its modules are available for free version as well as for paid one. This is an easy to learn and zero-cost automation tool. This is an open-source project by RedHat Enterprises. Anyone can learn it, use it and even can contribute in project.

**Operational:**

Ansible in action requires a linux machine as control node and windows/linux machine as remote node. The installation of ansible requires Python (2.7 or 3.x) and open-ssh need to be installed on both control and remote machine. For executing ansible-playbooks, ansible needs to be installed only in control machine whereas installation of ansible in remote machine is optional. Before executing any playbook, the public ssh key of control machine should be pasted in authorized\_keys of remote machine(s). Once it is done, verify whether all hosts in host\_inventory file is accessible. If all hosts in host\_inventory file are accessible, then we can now execute playbooks.

**2.2 System Requirement & Environment Set Up**

Before heading to system requirements, we will first go through basic terminologies used in ansible.

**Basic Terminologies:**

* **Controller Machine**: The machine on which the ansible is installed, responsible for running provisioning on the servers you are managing.
* **Remote Machine**: The machine(s) on which the tasks need to executed.
* **Inventory**: A simple text file which contains the information about the remote machines. This file contains remote machine’s IP address.
* **Playbook**: The entry point for ansible provisioning, where the automation is defined through tasks using YAML format.
* **Task**: A block that defines a single procedure to be executed e.g. installing a package.
* **Module**: A module typically abstracts a system task, like dealing with packages or creating and changing files.
* **Role**: A pre-defined way for organizing playbooks and other files in order to facilitate sharing and reusing portions of a provisioning.
* **Play**: A provisioning executed from start to finish is called a play. In simple words, execution of playbooks is called a play.
* **Facts**: Global variables that containing information about the system, like network interfaces or operating system.
* **Handlers**: Used to trigger service status changes, like restarting or stopping a service.

**System requirements & Environment Set Up:**

For ansible working environment, we need a linux machine which act as a control machine and several windows/linux machine which acts as remote machines. All the machines involved in the process should have Python (2.7 or 3.x) and open- ssh installed.

**Control Machine Configurations:**

1. Install ansible in Ubuntu using following commands:

* sudo apt update
* sudo apt install software-properties-common
* sudo apt-add-repository -y --update ppa:ansible/ansible
* sudo apt install ansible

1. Generate ssh key using following command:

* ssh-keygen

1. Copy your public ssh-key(id\_rsa.pub) to remote machine’s .ssh/authorized\_keys

**Remote Machine Configurations:**

1. Remote machines only require open-ssh and python (2.7 or 3.x) to be installed. Fortunately, that comes by default in linux distributions.

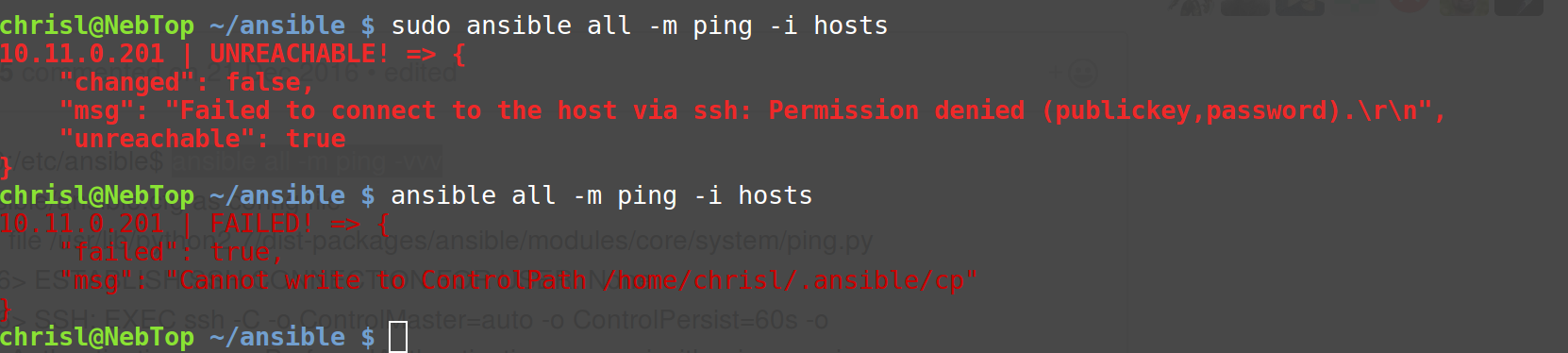
**2.3 Validation**

After setting up control and remote machines, we need an inventory file called host\_inventory file which contains the information about the remote hosts. Before executing the playbooks on the target machine, first we need to validate whether remote hosts are alive or not. We can validate by using following command:

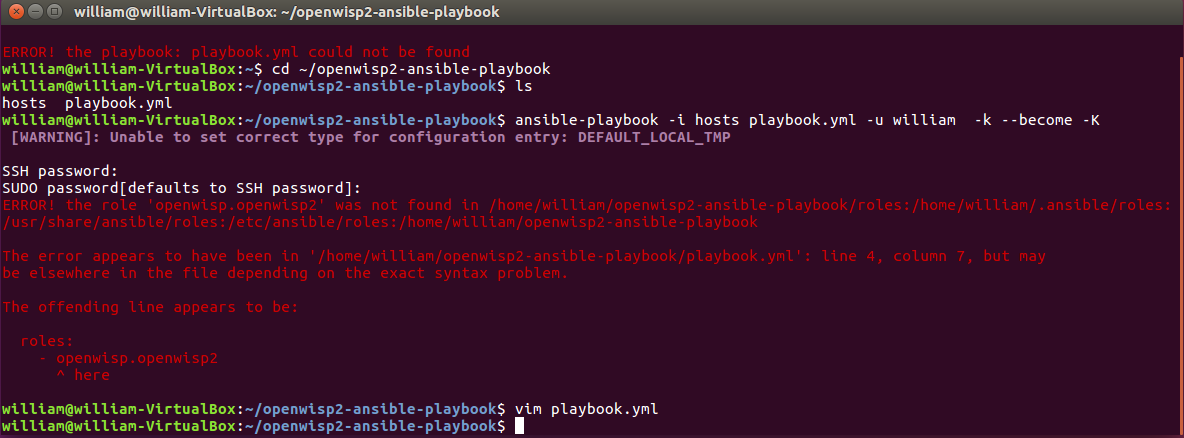
* ansible -i host\_inventory -m ping all

**2.4 Expected hurdles:**

2.4.1 “host is unreachable”: Either remote host is not alive or control’s public ssh- key is not pasted into .ssh/authorized\_keys



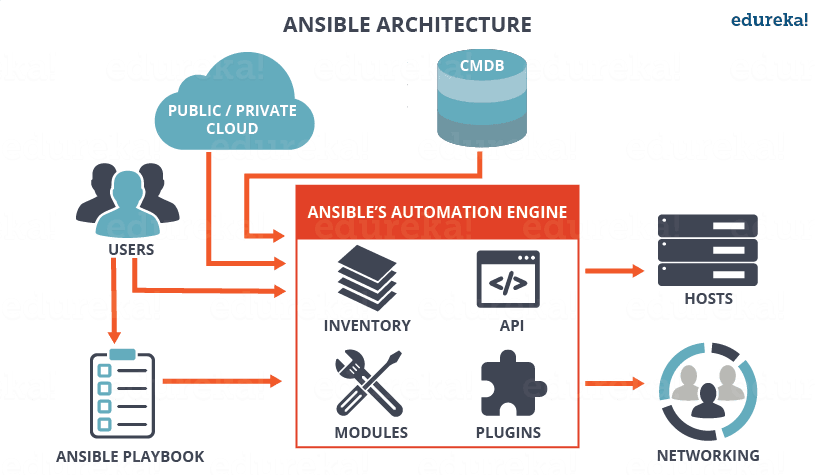
2.4.2 “syntax error”: This is due to syntax mistake in YAML file.



Chapter-3

System Design

**3.1 Ansible Architecture:**



As you can see, in the diagram above, the Ansible automation engine has a direct interaction with the users who write playbooks to execute the Ansible Automation engine. It also interacts with cloud services and Configuration Management Database (CMDB).

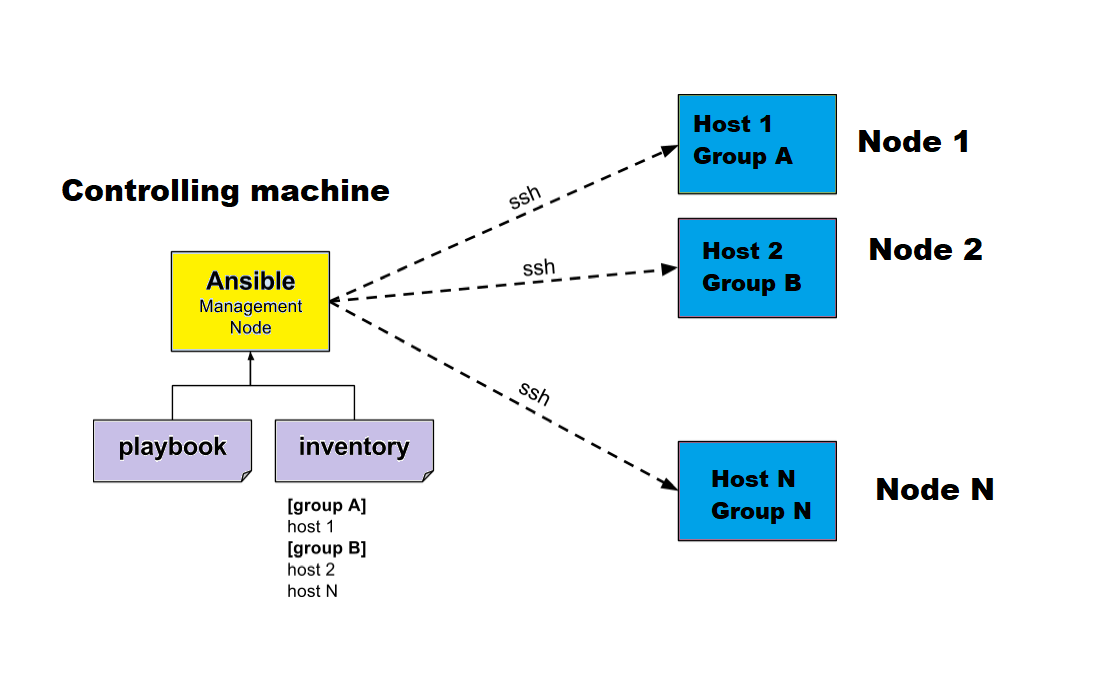
The Ansible Automation engine consists of:

* **Inventories**: Ansible inventories are lists of hosts (nodes) along with their IP addresses, servers, databases etc. which needs to be managed. Ansible then takes action via a transport – SSH for UNIX, Linux or Networking devices and WinRM for Windows system.
* **APIs**: APIs in Ansible are used as transport for Cloud services, public or private.
* **Modules**: Modules are executed directly on remote hosts through playbooks. The modules can control system resources, like services, packages, or files (anything really), or execute system commands.
* **Plugins**: Plugins allows to execute Ansible tasks as a job build step. Plugins are pieces of code that augment Ansible’s core functionality. Ansible ships with a number of handy plugins, and you can easily write your own.

There are a few more components in Ansible Architecture which are explained below:

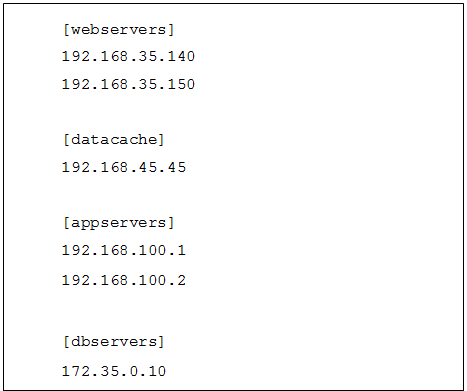
* **Networking**: Ansible can also be used to automate different networks. Ansible uses the same simple, powerful, and the agentless automation framework IT operations and development are already using. It uses a data model (a playbook or role) that is separate from the Ansible automation engine that easily spans different network hardware.
* **CMDB**: It is a repository that acts as a data warehouse for IT installations. It holds data relating to a collection of IT assets (commonly referred to as configuration items (CI)), as well as to describe relationships between such assets.
* **Cloud**: It is a network of remote servers hosted on the Internet to store, manage, and process data, rather than a local server. You can launch your resources and instances on cloud and connect to your servers.
  1. **How Ansible works?**

Ansible works by connecting the server of the user through the SSH keys and pushing the small programs out. With the use of modules, the playbooks help the ansible clients in performing all the specific tasks. The particular functions can include the service of restarting, installing packages, rebooting servers, and much more.



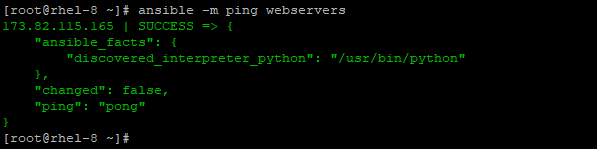
**3.2 Host Inventory file:**

It is a simple text file which contains the information about the hosts. Sample of a host\_inventory file is shown below:



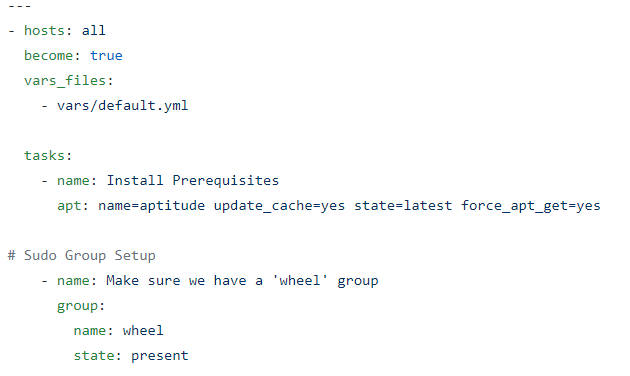
**3.3 Ad-hoc Commands:**

Ad hoc commands are commands which can be run individually to perform quick functions. These commands need not be performed later. For example, you have to reboot all your company servers.



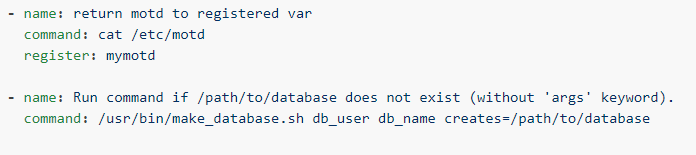
**3.4 Ansible Playbooks:**

An Ansible playbook is an **organized unit of scripts that defines work for a server configuration managed by the automation tool Ansible**. Ansible is a configuration management tool that automates the configuration of multiple servers by the use of Ansible playbooks. The playbook is the core component of any Ansible configuration.

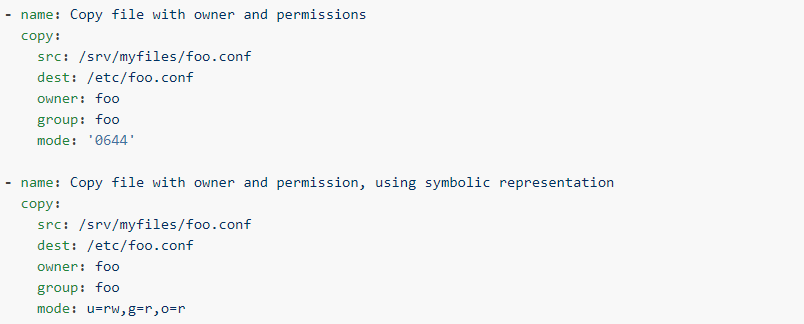


**3.4 Some Basic Modules:**

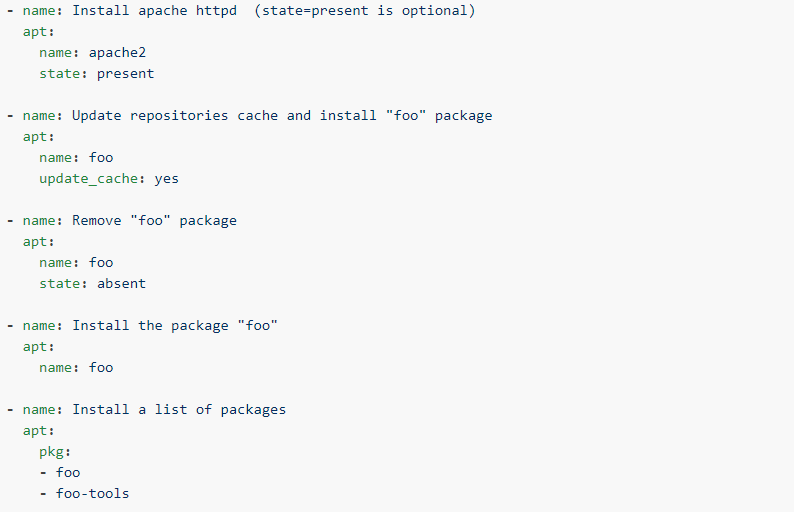
3.4.1 Command Module: Execute shell commands on targets



3.4.2 Copy Module: Copies files to remote locations



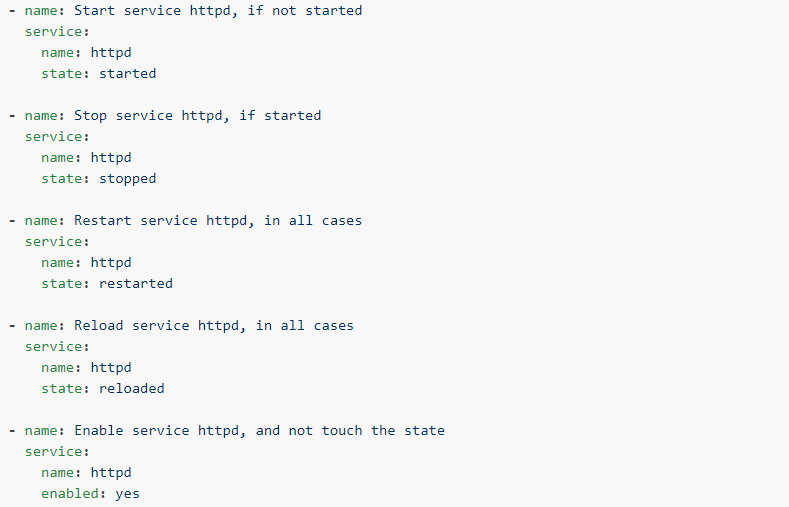
3.4.3 Apt Module: Manages apt-packages



3.4.4 Yum Module: Manages packages with yum package-manager



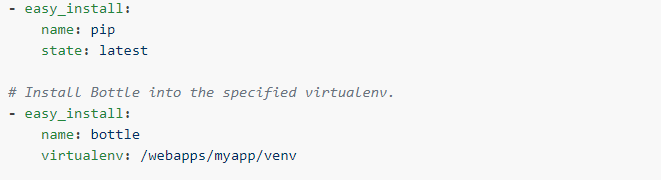
3.4.5 Service Module: Manage services



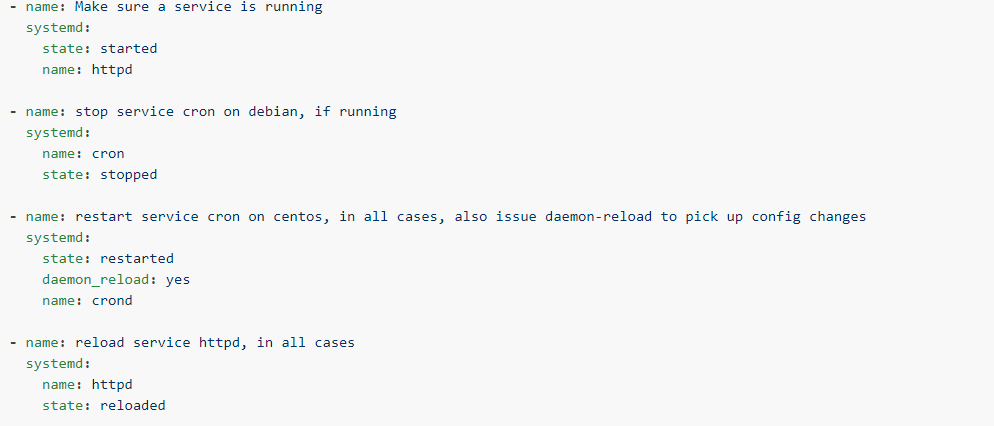
3.4.6 Pip Module: Manages python library dependencies



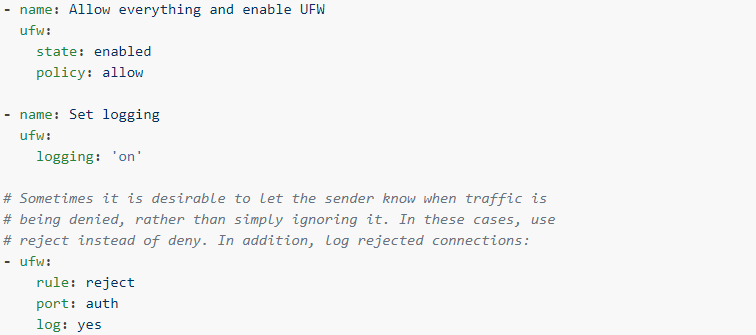
3.4.7 Easy\_Install: Install python libraries



3.4.8 Systemd Module: Manage services



3.4.9 UFW: Manages firewall



3.4.10 Yarn: Manage node.js packages



Chapter-4

Implementation, Testing & Maintenance

**4.1 Introduction of Languages:**

Ansible has large number of built-in modules for playbook execution. These modules can be written in Python or Ruby. Since ansible offers custom modules, you can write your own modules. For playbooks, ansible supports YAML language which is an easy language and even a non-tech guy can get an idea of what is happening.

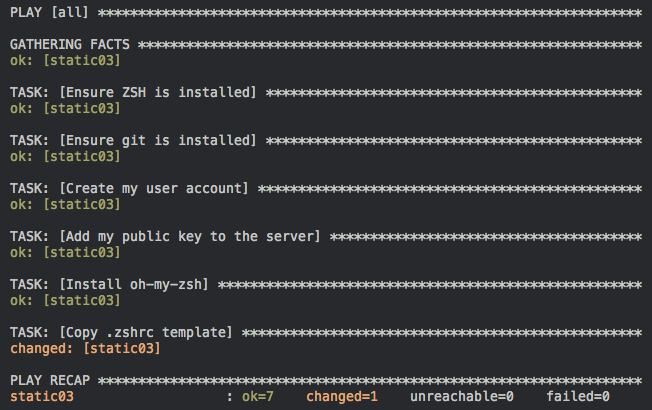
**4.2 Tools Involved:**

Ansible is an agentless and does not involve installation of any extra software or tool. It is complete in itself. It relies upon ssh service and requires open-ssh to be installed, which is installed by default in linux/Unix distributions.

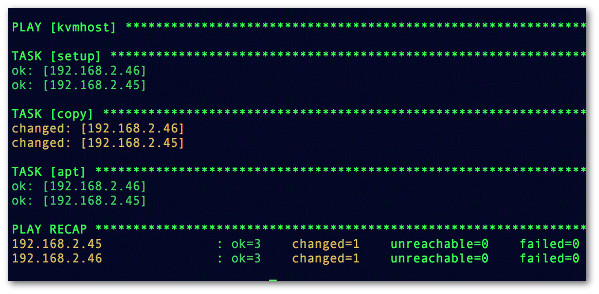
**4.3 Implementation:**

The implementation of ansible requires at least three linux machines out of which two are remote machines and one is control machine. This is a functional project. Screenshots of YAML commands, host\_inventory are already attached.

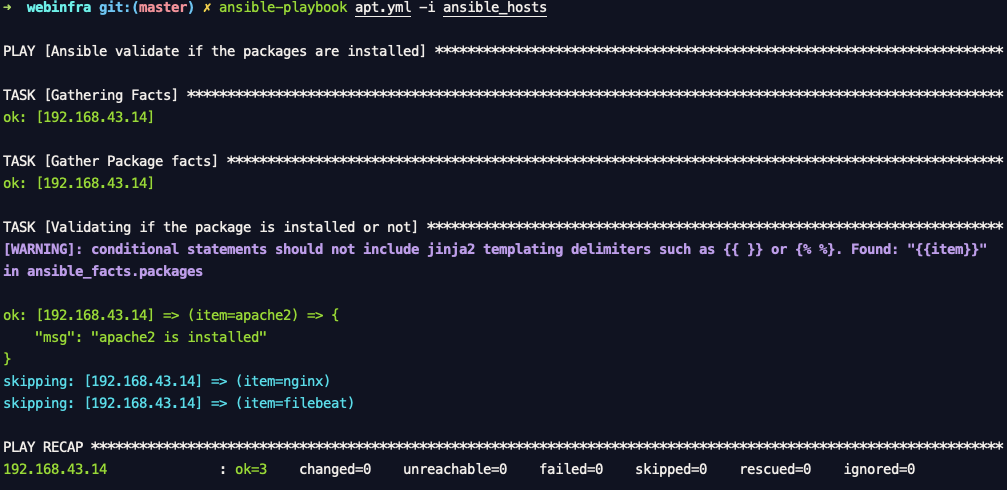
4.3.1 Playbook in Action:



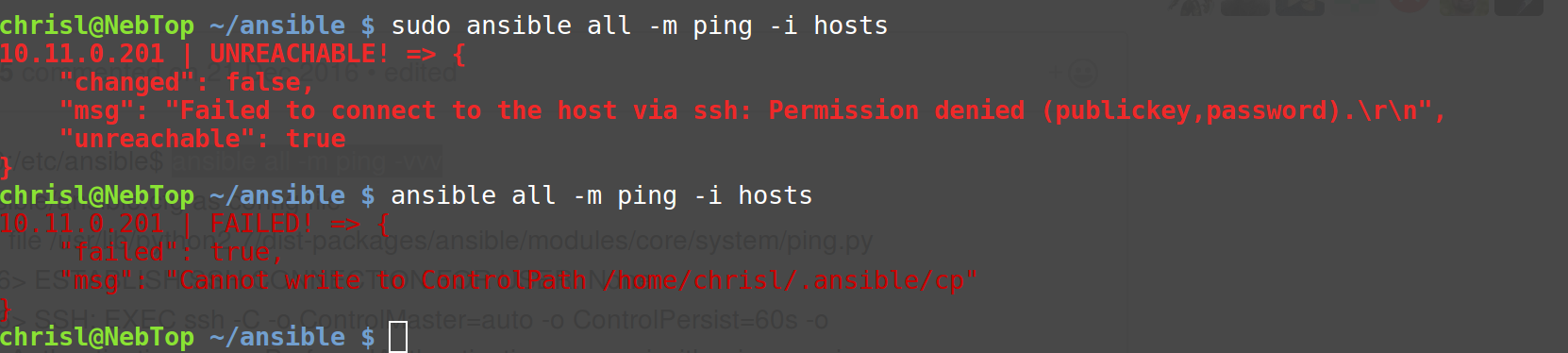
4.3.2 Copy Module in Action:

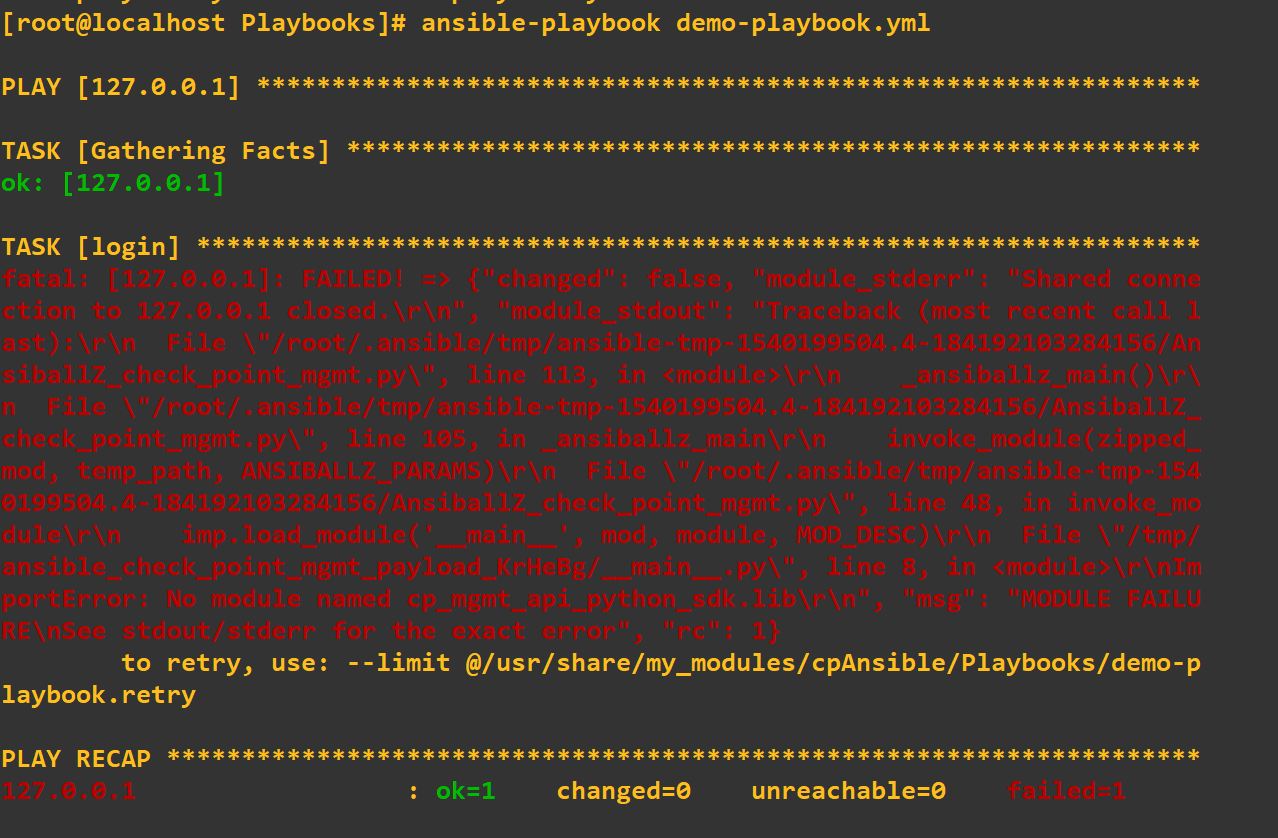


4.3.3 Apt Module in Action:



4.3.3 Errors Encounter during Execution:





Chapter-5

Conclusion & Future Scope

The future of ansible is very promising, and many more companies are set to accept this automation tools. As mentioned earlier, it is already in use by giant organizations like NASA, Microsoft, BMW and companies with large networks. Ansible has tendency to combine with emerging technologies like docker.

Advantages of using ansible with docker:

* Portability/Flexibility
* Auditability
* Management of Entire Environment
* Similar Syntax

Being an open-source project, it is a developing project. There is a community of DevOps Engineers working to make ansible more efficient and reliable. Undoubtedly, we can say it is going to be need for DevOps and has bright future.

References:

* <https://docs.ansible.com/ansible/latest/user_guide/index.html>
* <https://www.edureka.co/blog/what-is-ansible>
* <https://www.ansible.com/blog/the-future-of-ansible-content-delivery>
* <https://www.ansible.com/blog/nasa-automation>