

**PROFESSIONAL TRAINING REPORT**  
**at**  
**Sathyabama Institute of Science and Technology**  
**(Deemed to be University)**

Submitted in partial fulfillment of the requirements for the award of  
Bachelor of Technology in Information Technology

By

**NAME Pravin I**  
**(Reg. No.38120062)**



**DEPARTMENT OF INFORMATION TECHNOLOGY**

**SCHOOL OF COMPUTING**

**SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY**  
**JEPPIAAR NAGAR, RAJIV GANDHI SALAI,**  
**CHENNAI – 600119, TAMILNADU**

**NOVEMBER 2021**



**SATHYABAMA**  
INSTITUTE OF SCIENCE AND TECHNOLOGY  
(DEEMED TO BE UNIVERSITY)



**Accredited with Grade “A” by NAAC**  
(Established under Section 3 of UGC Act, 1956)  
JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI– 600119  
[www.sathyabamauniversity.ac.in](http://www.sathyabamauniversity.ac.in)

---

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**BONAFIDE CERTIFICATE**

This is to certify that this Project Report is the bonafide work of **Pravin I (38120062)** who carried out the project entitled “**Hardware Report using Red HAT**” under my supervision from April 2020 to June 2020.

**Internal Guide**

**Ms.Nirmal Rani**

**Head of the Department**

---

**Submitted for Viva voce Examination held on** \_\_\_\_\_

**Internal Examiner**

**External Examiner**

## DECLARATION

I, **Pravin** I hereby declare that the Project Report entitled **HARDWARE REPORT using Red Hat** done by me under the guidance of **Ms.Nirmal Rani** is submitted in partial fulfillment of the requirements for the award of Bachelor of Technology in Information Technology

**DATE:**

**PLACE:**

**SIGNATURE OF THE CANDIDATE**

## ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management** of **SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T.Sasikala M.E., Ph.D., Dean**, School of Computing , **Dr. R. Subhashini, M.E., Ph.D. Head of the Department of Information Technology** for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. Nirmal Rani M.E., Ph.D.**,for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my projectwork.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Information Technology**who were helpful in many ways for the completion of the project

## Training Certificate



# Red Hat

# Certificate of Participation

Pravin

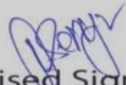
has attended and successfully completed

Red Hat Certified System Administrator, ON-LINE course

at Sathyabama Institute of Science and Technology

held from 27th Sep 2021 to 16th Oct 2021



  
Authorised Signatory

S.No : 2021/ADV/12002

Date : 27th Oct 2021

## Abstract

As data centers grew, and hosted applications became more complex, administrators realized they couldn't scale their manual systems management as fast as the applications they were enabling. It also hampered the velocity of the work of the developers since the development team was agile and releasing software frequently, but IT operations were spending more time configuring the systems. That's where automation tools came to flourish. And one such a type of tool is **ANSIBLE**.

Ansible is a helpful tool that allows us to create groups of machines, describe how these machines should be configured or what actions should be taken on them. Ansible issues all commands from a central location to perform these tasks. No other client software is installed on the node machines. It uses SSH to connect to the nodes. Ansible only needs to be installed on the control machine. It is a simple solution to a complicated problem

## TABLE OF CONTENTS

| Chapter No | TITLE                                                                                                                                                                                                                                                                            | Page No |
|------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------|
|            | Abstract                                                                                                                                                                                                                                                                         | vi      |
| 1          | Introduction<br>1.1 Outline of the project<br>1.2 Objectives of automation<br>1.3 Purpose of the Project<br>1.4 Existing System<br>1.5 Proposed System                                                                                                                           | 1       |
| 2          | Ansible and its Applications<br>2.1 What is Ansible<br>2.2 Strengths of ansible<br>2.3 concepts and Architecture<br>2.4 Uniqueness of Ansible<br>2.5 usecases                                                                                                                    | 4       |
| 3          | Installing and Deploying Ansible<br>3.1 Control Nodes<br>3.2 Manage Nodes<br>3.3 Configuring the Inventory Files<br>3.4 Managing Ansible Configuration Files<br>3.5 Implementing Playbooks<br>3.6 Remote Users and Privilege Escalation in Plays<br>3.7 Finding Modules Fortasks | 9       |

|          |                                                                                            |           |
|----------|--------------------------------------------------------------------------------------------|-----------|
|          |                                                                                            |           |
| <b>4</b> | <b>Appendix</b><br><b>4.1 Screenshots</b><br><b>4.2 Sourcecode</b><br><b>4.3 Reference</b> | <b>24</b> |

# CHAPTER - 1

## INTRODUCTION

### 1.1 Outline of the project

Automation is the technology by which a process or procedure is performed with minimal human interference through the use of technological or mechanical devices. It is the technique of making a process or a system operate automatically. Today the development of technologies made it possible to introduce industry automation systems into almost all manufacturing fields. Industry automation of production processes enhances labour efficiency and allows cutting net cost and improving product quality. Moreover, it extends the equipment service life, saves consumables and raw materials, and improves production safety as a whole.

### 1.2 Objectives of Automation

- Run regression tests overnight and weekends
- Reduce testing staff
- Find more bugs
- Reduce elapsed time for testing
- Automate x% of the testing (Yes, DON'T automate 100% every time, because neither ever it is possible to automate everything)
- If useful tests, then making use of under-used resource is a good idea
- Run most important tests using spare resource (top 10% of usefulness rating, run out of hours)
- Find more regression bugs (measured by increase in DDP of regression tests by 10% over 6 months)
- Reduce elapsed time of tool-supported activities (measured for maintenance & failure analysis time)
- Improve automation support for testers (testers rate usefulness of automation support, how often utilities/automation features are used)
- Remember: Good objectives for automation are: measurable, realistic and achievable
- Improve the completeness and consistency

### 1.3 purpose of the project

Every time source code is modified software tests should be repeated. For each release of the software it may be tested on all supported operating systems and hardware configurations. Manually repeating these tests is costly and time consuming. Once

created, automated tests can be run over and over again at no additional cost and they are much faster than manual tests. Automated software testing can reduce the time to run repetitive tests from days to hours. A time savings that translates directly into cost savings. Automated software testing can increase the depth and scope of tests to help improve software quality. Lengthy tests that are often avoided during manual testing can be run unattended. They can even be run on multiple computers with different configurations. Automated software testing can look inside an application and see memory contents, data tables, file contents, and internal program states to determine if the product is behaving as expected. Test automation can easily execute thousands of different complex test cases during every test run providing coverage that is impossible with manual tests

## 1.4 Existing systems

Nowadays, the whole world depends upon automation because of its advantages like reduction of time, has more efficiency, has more success rate, needs less manpower and so on. At this situation we have a lot of automation tools, some of the most common tools are puppet, chef, CFEngine, rudder and so on. These are more complex and costs more and the output efficiency is less.

### Disadvantages

- Runs on Master Slave architecture.
- needs a certificate signing for client machine configuration.
- Client machine have to pull configuration from master server. For that the configuration execution facility will take more time.
- Initial setup is difficult and installation process is unable to point out the errors at the time of installation.
- Not so highly secured tool.
- Doesn't have comprehensive reporting features.
- Can be difficult for those new to programming.

## 1.5 Proposed system

A good automation system allows us to implement Infrastructure as Code practices. Infrastructure as Code means that we can use a machine-readable automation language to define and describe the state we want our IT infrastructure to be in. Ideally, this automation language should also be very easy for humans to read, because then we can easily understand what the state is and make changes to it. This code is then applied to our infrastructure to ensure that it is actually in that state. If the automation language is represented as simple text files, it can easily be managed in a version control system like software code. And such a type of tool is called **Ansible**. The advantage of this is that every change can be checked into the version control system, so we have a history of the changes we make over time. If we want to revert to an earlier known-good configuration, we simply can check out

that version of the code and apply it to our infrastructure

### **Advantages**

- Ansible platform is written on python but It supports YAML commandScripts.
- Its running system is based on single active node.
- For client system configuration Ansible do not need any proper machine VM and also it is faster to set up
- It uses ssh running system to configure nodes
- It uses YAML language which is close to English that's why it is easy to master and manage the configuration
- Its initial setup is easy and easy to install
- It basically focuses on computer automation system over configuration management
- It is a highly secured tool with SSH
- It uses both pull and push process to distribute configuration to the nodes. By this process it makes capable of all servers for question each other and makes a faster communication
- It has immediate remote execution facility because server is responsible for pushing configuration to all the nodes
- Can be easy for those who're new to programming

## CHAPTER - 2

### Ansible and its applications

#### 2.1 What is Ansible

Ansible is an open source automation platform. It is a simple automation language that can perfectly describe an IT application infrastructure in Ansible Playbooks. It is also an automation engine that runs Ansible Playbooks. Ansible can manage powerful automation tasks and can adapt to many different workflows and environments. At the same time, new users of Ansible can very quickly use it to become productive. Ansible Playbooks provide human-readable automation. This means that playbooks are automation tools that are also easy for humans to read, comprehend, and change. No special coding skills are required to write them. Playbooks execute tasks in order. The simplicity of playbook design makes them usable by every team, which allows people new to Ansible to get productive quickly. We can use Ansible to deploy applications, for configuration management, for workflow automation, and for network automation. Ansible can be used to orchestrate the entire application life cycle. Ansible is built around an agentless architecture. Typically, Ansible connects to the hosts it manages using OpenSSH or WinRM and runs tasks, often (but not always) by pushing out small programs called Ansible modules to those hosts. These programs are used to put the system in a specific desired state. Any modules that are pushed are removed when Ansible is finished with its tasks. We can start using Ansible almost immediately because no special agents need to be approved for use and then deployed to the managed hosts. Because there are no agents and no additional custom security infrastructure, Ansible is more efficient and more secure than other alternatives.

#### 2.2 Strengths of ansible

Ansible has a number of important strengths:

- Cross platform support: Ansible provides agentless support for Linux, Windows, UNIX, and network devices, in physical, virtual, cloud, and container environments.
- Human-readable automation: Ansible Playbooks, written as YAML text files, are easy to read and help ensure that everyone understands what they will do.
- Perfect description of applications: Every change can be made by Ansible Playbooks, and every aspect of our application environment can be described and documented.
- Easy to manage in version control: Ansible Playbooks and projects are plaintext. They can be

treated like source code and placed in our existing version control system.

- Support for dynamic inventories: The list of machines that Ansible manages can be dynamically

updated from external sources in order to capture the correct, current list of all managed servers

all the time, regardless of infrastructure or location.

- Orchestration that integrates easily with other systems: HP SA, Puppet, Jenkins, Red Hat

Satellite, and other systems that exist in our environment can be leveraged and integrated into

our Ansible workflow

## 2.3 Concepts and Architecture

There are two types of machines in the Ansible architecture: control nodes and managed hosts. Ansible is installed and run from a control node, and this machine also has copies of our Ansible project files. A control node could be an administrator's laptop, a system shared by a number of administrators, or a server running Red Hat Ansible Tower. Managed hosts are listed in an inventory, which also organizes those systems into groups for easier collective management. The inventory can be defined in a static text file, or dynamically determined by scripts that get information from external sources. Instead of writing complex scripts, Ansible users create high-level plays to ensure a host or group of hosts are in a particular state. A play performs a series of tasks on the hosts, in the order specified by the play. These plays are expressed in YAML format in a text file. A file that contains one or more plays is called a playbook. Each task runs a module, a small piece of code (written in Python, PowerShell, or some other language), with specific arguments. Each module is essentially a tool in our toolkit. Ansible ships with hundreds of useful modules that can perform a wide variety of automation tasks. They can act on system files, install software, or make API calls. When used in a task, a module generally ensures that some particular aspect of the machine is in a particular state. For example, a task using one module may ensure that a file exists and has particular permissions and contents, while a task using a different module may make certain that a particular file system is mounted. If the system is not in that state, the task should put it in that state. If the system is already in that state, it does nothing. If a task fails, Ansible's default behavior is to abort the rest of the playbook for the hosts that had a failure. Tasks, plays, and playbooks are designed to be idempotent. This means that we can safely run a playbook on the same hosts multiple times. When our systems are in the correct state, the playbook makes no changes when we run it. This means that we should be able to run a playbook on the same hosts multiple times safely. When our systems are in the correct state the playbook should make no changes when we run it. There are a handful of modules that we can use to run arbitrary commands. However, we must use those modules with care to ensure that they run in an idempotent way. Ansible also uses plug-ins. Plug-ins are code that we can add to Ansible to extend it and adapt it

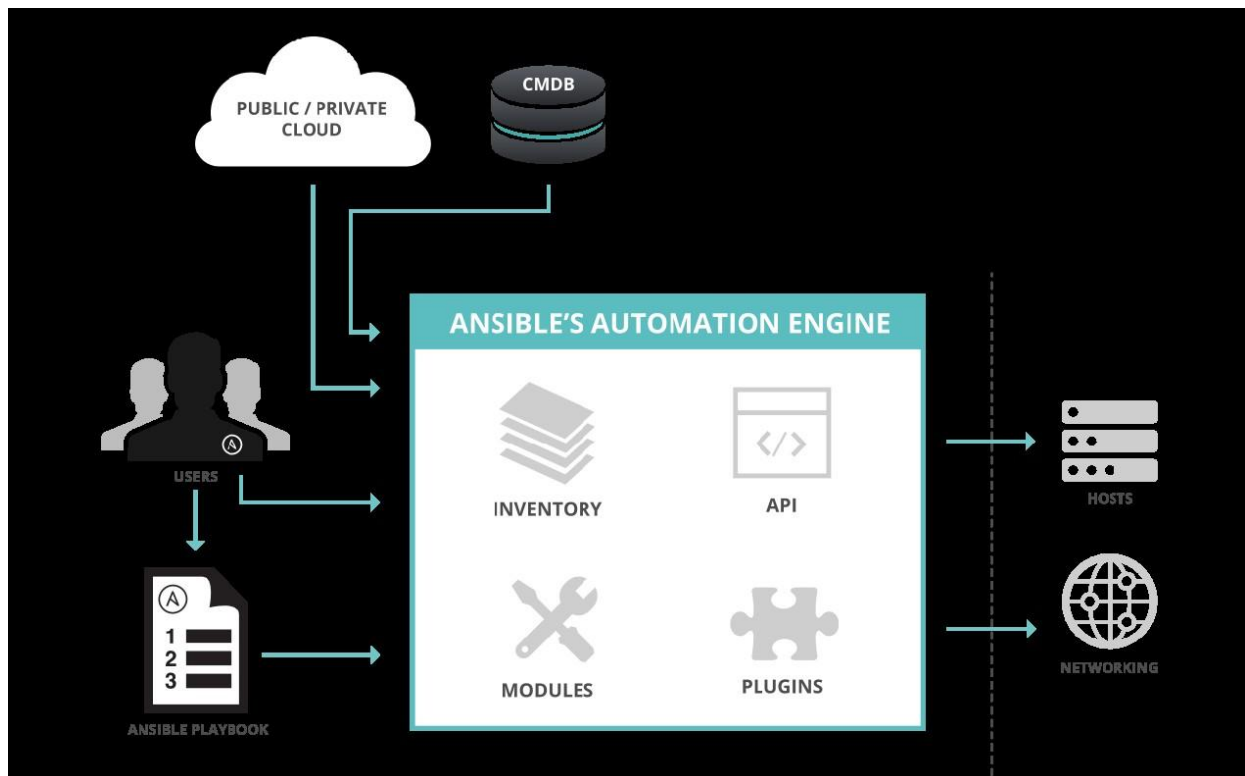
to new uses and platforms.

The Ansible architecture is agentless. Typically, when an administrator runs an Ansible Playbook or an ad hoc command, the control node connects to the managed host using SSH (by default)

or WinRM. This means that clients do not need to have an Ansible-specific agent installed on managed hosts, and do not need to permit special network traffic to some nonstandard port. Red Hat Ansible Tower is an enterprise framework to help us control, secure, and manage our

Ansible automation at scale. We can use it to control who has access to run playbooks on which hosts, share the use of SSH credentials without allowing users to transfer or see their contents, log all of our Ansible jobs, and manage inventory, among many other things. It provides a web-based

user interface (web UI) and a RESTful API. It is not a core part of Ansible, but a separate product that helps us use Ansible more effectively with a team or at a large scale.



## 2.4 Uniqueness of ansible

### Complexity Kills Productivity

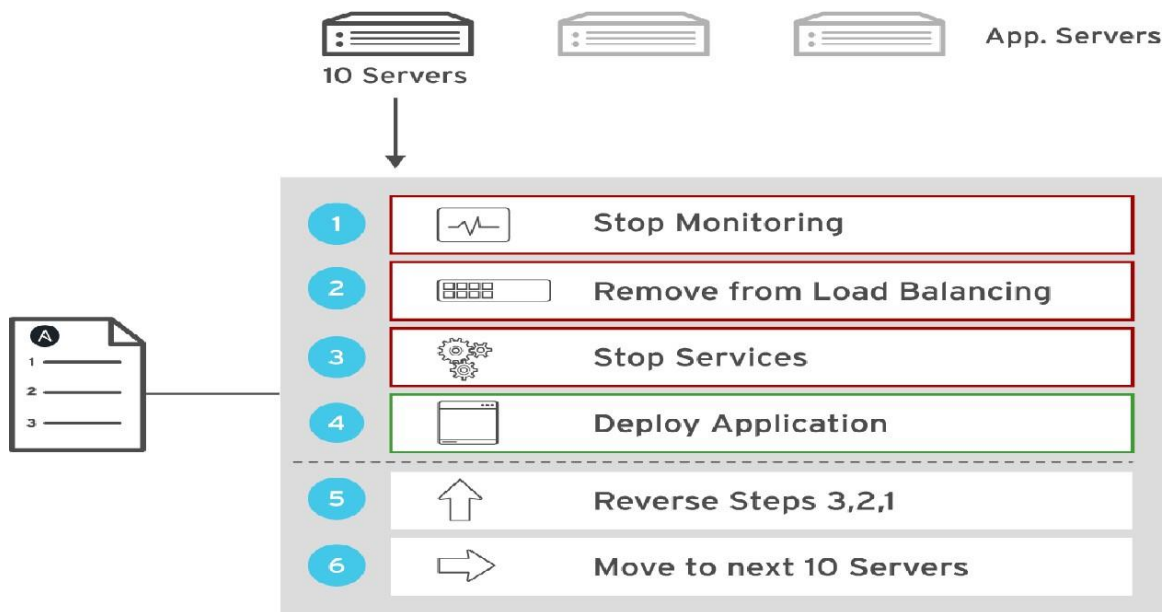
Simpler is better. Ansible is designed so that its tools are simple to use and automation is simple to write and read. we should take advantage of this to strive for simplification in how we create our automation.

### Optimize for Readability

The Ansible automation language is built around simple, declarative, text-based files that are easy for humans to read. Written properly, Ansible Playbooks can clearly document our workflow automation.

### Think Declaratively

Ansible is a desired-state engine. It approaches the problem of how to automate IT deployments by expressing them in terms of the state that we want our systems to be in. Ansible's goal is to put our systems into the desired state, only making changes that are necessary. Trying to treat Ansible like a scripting language is not the right approach



## 2.5 USECASES

Unlike some other tools, Ansible combines and unites orchestration with configuration management, provisioning, and application deployment in one easy-to-use platform.

**Some use cases for Ansible include:**

### **Configuration Management**

Centralizing configuration file management and deployment is a common use case for Ansible, and it is how many power users are first introduced to the Ansible automation platform.

### **Application Deployment**

When we define our application with Ansible, and manage the deployment with Red Hat Ansible Tower, teams can effectively manage the entire application life cycle from development to production.

### **Provisioning**

Applications have to be deployed or installed on systems. Ansible and Red Hat Ansible Tower can help streamline the process of provisioning systems, whether we are PXE booting and kickstarting bare-metal servers or virtual machines, or creating virtual machines or cloud instances from templates. Applications have to be deployed or installed on systems.

### **Continuous Delivery**

Creating a CI/CD pipeline requires coordination and buy-in from numerous teams. We cannot do it without a simple automation platform that everyone in our organization can use. Ansible Playbooks keep our applications properly deployed (and managed) throughout their entire life cycle.

### **Security and Compliance**

When our security policy is defined in Ansible Playbooks, scanning and remediation of sitewide security policies can be integrated into other automated processes. Instead of being an afterthought, it is an integral part of everything that is deployed.

### **Orchestration**

Configurations alone do not define our environment. We need to define how multiple configurations interact, and ensure the disparate pieces can be managed as a whole.

## Chapter - 3

### Installing and Deploying Ansible

#### 3.1 Installing Ansible

##### 3.1.1 Control nodes

Ansible is simple to install. The Ansible software only needs to be installed on the control node (or nodes) from which Ansible will be run. Hosts that are managed by Ansible do not need to have Ansible installed. This installation involves relatively few steps and has minimal requirements. The control node should be a Linux or UNIX system. Microsoft Windows is not supported as a control node, although Windows systems can be managed hosts. Python 3 (version 3.5 or later) or Python 2 (version 2.7 or later) needs to be installed on the control node. If we are running Red Hat Enterprise Linux 8, Ansible 2.8 can automatically use the platform-python package that supports system utilities that use Python. We do not need to install the python36 or python27 package from AppStream

#### 3.2 Manage Nodes

One of the benefits of Ansible is that managed hosts do not need to have a special agent installed. The Ansible control node connects to managed hosts using a standard network protocol to ensure that the systems are in the specified state. Managed hosts might have some requirements depending on how the control node connects to them and what modules it will run on them. Linux and UNIX managed hosts need to have Python 2 (version 2.6 or later) or Python 3 (version 3.5 or later) installed for most modules to work. For Red Hat Enterprise Linux 8, we may be able to depend on the platform-python package. We can also enable and install the python36 application stream (or the python27 application stream). If SELinux is enabled on the managed hosts, we also need to make sure the python3-libselenium package is installed before using modules that are related to any copy, file, or template functions. If the other Python components are installed, we can use Ansible modules such as yum or package to ensure that this package is also installed. Some modules do not need Python. For example, arguments passed to the Ansible raw modules

run directly through the configured remote shell instead of going through the module subsystem. This can be useful for managing devices that do not have

Python available or cannot have Python installed, or for bootstrapping Python onto a system that does not have it. However, the raw module is difficult to use in a safely idempotent way. If we can use a normal module instead, it is generally better to avoid using raw and similar command modules. This is discussed further later in the course

### **3.2.1 Microsoft Windows-based ManagedHosts**

Ansible includes a number of modules that are specifically designed for Microsoft Windows systems. These are listed in the Windows Modules [[https://docs.ansible.com/ansible/latest/modules/list\\_of\\_windows\\_modules.html](https://docs.ansible.com/ansible/latest/modules/list_of_windows_modules.html)] section of the Ansible module index. Most of the modules specifically designed for Microsoft Windows managed hosts require PowerShell 3.0 or later on the managed host rather than Python. In addition, the managed hosts need to have PowerShell remoting configured. Ansible also requires at least .NET Framework 4.0 or later to be installed on Windows managed hosts. This course uses Linux-based managed hosts in its examples, and does not go into great depth on the specific differences and adjustments needed when managing Microsoft Windows-based managed hosts. More information is available on the Ansible web site at [https://docs.ansible.com/ansible/latest/user\\_guide/windows.html](https://docs.ansible.com/ansible/latest/user_guide/windows.html)

### **3.2.2 Managed Network Hosts**

We can also use Ansible automation to configure managed network devices such as routers and switches. Ansible includes a large number of modules specifically designed for this purpose. This includes support for Cisco IOS, IOS XR, and NX-OS; Juniper Junos; Arista EOS; and VyOS-based networking devices, among others. We can write Ansible Playbooks for network devices using the same basic techniques that we use when writing playbooks for servers. Because most network devices cannot run Python, Ansible runs network modules on the control node, not on the managed hosts. Special connection methods are also used to communicate with network devices, typically using either CLI over SSH, XML over SSH, or API over HTTP(S). Ansible connects to managed hosts using standard network protocols such as SSH, and runs code or commands on the managed hosts to ensure that they are in the state specified by Ansible

## **3.3 Configuring the Inventory Files**

### **3.3.1 Defining the Inventory**

An inventory defines a collection of hosts that Ansible will manage. These hosts can also be assigned to groups, which can be managed collectively. Groups can contain child groups, and hosts can be members of multiple groups. The inventory can also set variables that apply to the hosts and groups that it defines. Host inventories can be defined in two different ways. A static host inventory can be defined by a text file. A dynamic host inventory can be generated by a script or other program as needed, using external information providers.

### **3.3.2 Specifying Managed Hosts with a Static Inventory**

A static inventory file is a text file that specifies the managed hosts that Ansible targets. We can write this file using a number of different formats, including INI-style or YAML. The INI-style format is very common and will be used for most examples in this course. Hosts can be in multiple groups. In fact, recommended practice is to organize our hosts into multiple groups, possibly organized in different ways depending on the role of the host, its physical location, whether it is in production or not, and so on. This allows us to easily apply Ansible plays to specific hosts

### **3.3.3 Defining Nested Groups**

Ansible host inventories can include groups of host groups. This is accomplished by creating a host group name with the: `childrensuffix`

### **3.3.4 Defining Variables in theInventory**

Values for variables used by playbooks can be specified in host inventory files. These variables only apply to specific hosts or host groups. Normally it is better to define these inventory variables in special directories and not directly in the inventory file. This topic is discussed in more depth elsewhere in the course

### **3.3.5 Verifying theInventory**

If the inventory contains a host and a host group with the same name, the ansible command prints a warning and targets the host. The host group is ignored. There are various ways to deal with this situation, the easiest being to ensure that host groups do not use the same names as hosts in the inventory

### **3.3.6 Describing a Dynamic Inventory**

Ansible inventory information can also be dynamically generated, using information provided by external databases. The open source community has written a number of dynamic inventory scripts that are available from the upstream Ansible project. If those scripts do not meet our needs, we can also write our own

## **3.4 Managing Ansible ConfigurationFiles**

### **3.4.1 ConfiguringAnsible**

The behavior of an Ansible installation can be customized by modifying settings in the Ansible configuration file. Ansible chooses its configuration file from one of several possible locations on the control node.

Using `/etc/ansible/ansible.cfg`

The ansible package provides a base configuration file located at

`/etc/ansible/ansible.cfg`. This file is used if no other configuration file is found.

Using `~/. ansible.cfg`

Ansible looks for a `.ansible.cfg` file in the user's home directory. This configuration is used instead of the `/etc/ansible/ansible.cfg` if it exists and if there is no `ansible.cfg` file in the current working directory.

Using `./ansible.cfg`

If an `ansible.cfg` file exists in the directory in which the `ansible` command is executed, it is used instead of the global file or the user's personal file. This allows administrators to create a directory structure where different environments or projects are stored in separate directories, with each directory containing a configuration file tailored with unique set of settings.

The recommended practice is to create an `ansible.cfg` file in a directory from which we run Ansible commands. This directory would also contain any files used by our Ansible project, such as an inventory and a playbook. This is the most common location used for the Ansible configuration file. It is unusual to use a `~/. ansible.cfg` or `/etc/ansible/ansible.cfg` file in practice.

### **3.4.2 Using the `ANSIBLE_CONFIG` environment variable**

We can use different configuration files by placing them in different directories and then executing Ansible commands from the appropriate directory, but this method can be restrictive and hard to manage as the number of configuration files grows. A more flexible option is to define the location of the configuration file with the `ANSIBLE_CONFIG` environment variable. When this variable is defined, Ansible uses the configuration file that the variable specifies instead of any of the previously mentioned configuration files.

### **3.4.3 Configuration File Precedence**

The search order for a configuration file is the reverse of the preceding list. The first file located in the search order is the one that Ansible selects. Ansible only uses configuration settings from the first file that it finds. Any file specified by the `ANSIBLE_CONFIG` environment variable overrides all other configuration files. If that variable is not set, the directory in which the `ansible` command was run is then checked for an `ansible.cfg` file. If that file is not present, the user's home directory is checked for an `.ansible.cfg` file. The global `/etc/ansible/ansible.cfg` file is only used if no other configuration file is found. If the `/etc/ansible/ansible.cfg` configuration file is not present, Ansible contains defaults which it uses. Because of the multitude of locations in which Ansible configuration files can be placed, it can be

confusing which configuration file is being used by Ansible. We can run the `ansible --version` command to clearly identify which version of Ansible is installed, and which configuration file is being used. Another way to display the active Ansible configuration file is to use the `-v` option when executing Ansible commands on the command line. Ansible only uses settings from the configuration file with the highest precedence. Even if other files with lower precedence exist, their settings are ignored and not combined with those in the selected configuration file. Therefore, if we choose to create our own configuration file in favor of the global `/etc4/ansible/ansible.cfg` configuration file, we need to duplicate all desired settings from that file to our own user-level configuration file. Settings not defined in the user level configuration file remain set to the built-in defaults, even if they are set to some other value by the global configuration file

### 3.4.4 Managing Settings in Configuration Files

The Ansible configuration file consists of several sections, with each section containing settings defined as key-value pairs. Section titles are enclosed in square brackets. For basic operation use the following two sections:

- `[defaults]` sets defaults for Ansible operation
- `[privilege_escalation]` configures how Ansible performs privilege escalation on managed hosts

#### Ansible Configuration

| DIRECTIVE   | DESCRIPTION                                                                                                                         |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------|
| Inventory   | Specifies the path to the inventory file.                                                                                           |
| remote_user | The name of the user to log in as on the managed hosts. If not specified, the current user's name is used                           |
| ask_pass    | Whether or not to prompt for an SSH password. Can be false if using SSH public key authentication                                   |
| become      | Whether to automatically switch user on the managed host (typically to root) after connecting. This can also be specified by a play |

|                 |                                                                                   |
|-----------------|-----------------------------------------------------------------------------------|
| become_method   | How to switch user (typically sudo, which is the default, but su is an option).   |
| become_user     | The user to switch to on the managed host (typically root, which is the default). |
| become_ask_pass | Whether to prompt for a password for our become_method. Defaults to false         |

### 3.4.5 Configuring Connections

Ansible needs to know how to communicate with its managed hosts. One of the most common reasons to change the configuration file is to control which methods and users Ansible uses to administer managed hosts.

**Some of the information needed includes:**

- The location of the inventory that lists the managed hosts and host groups
- Which connection protocol to use to communicate with the managed hosts (by default, SSH), and whether or not a nonstandard network port is needed to connect to the server
- Which remote user to use on the managed hosts; this could be root or it could be an unprivileged user
- If the remote user is unprivileged, Ansible needs to know if it should try to escalate privileges to root and how to do it (for example, by using sudo)
- Whether or not to prompt for an SSH password or sudo password to log in or gain privileges

### 3.4.6 connection Settings

By default, Ansible connects to managed hosts using the SSH protocol. The most important parameters that control how Ansible connects to the managed hosts are set in the [defaults] section. By default, Ansible attempts to connect to the managed host using the same user name as the local user running the Ansible commands. To specify a different remote user, set the remote\_user parameter to that user name. If the local user running Ansible has private SSH keys configured that allow them to authenticate as the remote user on the managed hosts, Ansible automatically logs in. If that is not the case, we can configure Ansible to prompt the local user for the password used by the remote user by setting the directive ask\_pass = true

### 3.4.7 Inventory Location

In the [defaults] section, the inventory directive can point directly to a static inventory file, or to a directory containing multiple static inventory files and dynamic inventory scripts

```
[defaults]
inventory = ./inventory
```

### 3.4.8 Connection privileges

By default, Ansible connects to managed hosts using the SSH protocol. The most important parameters that control how Ansible connects to the managed hosts are set in the [defaults] section. By default, Ansible attempts to connect to the managed host using the same user name

as the local user running the Ansible commands. To specify a different remote user, set the remote\_user parameter to that user name. If the local user running Ansible has private SSH keys configured that allow them to authenticate as the remote user on the managed hosts, Ansible automatically logs in. If that is not the case, we can configure Ansible to prompt the local user for the password used by the remote user by setting the directive ask\_pass = true.

```
[defaults]
inventory = ./inventory
remote_user = root
ask_pass = true
```

Assuming that we are using a Linux control node and OpenSSH on our managed hosts, if we can log in as the remote user with a password then we can probably set up SSH key-based authentication, which would allow us to set ask\_pass = false. The first step is to make sure that the user on the control node has an SSH key pair configured in ~/.ssh. We can run the ssh-keygen command to accomplish this. For a single existing managed host, we can install our public key on the managed host and use the ssh-copy-id command to populate our local ~/.ssh/known\_hosts file with its host key, as follows:

```
[root@workstation ~] # ssh-copy-id root@node1
The authenticity of host 'node1' can't be established.
ECDSA key fingerprint is 70:9c:03:cd:de:ba:2f:11:98:fa:a0:b3:7c:40:86:4b.
Are you sure you want to continue connecting (yes/no)? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter
out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted
now it is to install the new keys
root@node1 password:
Number of key(s) added: 1
```

We can also use an Ansible Playbook to deploy our public key to the

remote\_user account on all managed hosts using the authorized\_key module. A play that ensures that our public key is deployed to managed hosts' root accounts might read. Because the managed host would not have SSH key-based authentication configured yet, we would have to run the playbook using the ansible-playbook command with the --ask-pass option in order for the command to authenticate as the remote user.

### **3.4.9 Escalating Privileges**

For security and auditing reasons, Ansible might need to connect to remote hosts as an unprivileged user before escalating privileges to get administrative access as root. This can be set up in the [privilege\_escalation] section of the Ansible configuration file. To enable privilege escalation by default, set the directive become = true in the configuration file. Even if this is set by default, there are various ways to override it when running ad hoc commands or Ansible Playbooks. The become\_method directive specifies how to escalate privileges. Several options are available, but the default is to use sudo. Likewise, the become\_user directive specifies which user to escalate to, but the default is root. If the become\_method mechanism chosen requires the user to enter a password to escalate privileges, we can set the become\_ask\_pass = true directive in the configuration file

### **3.4.10 Non SSH Connections**

The protocol used by Ansible to connect to managed hosts is set by default to smart, which determines the most efficient way to use SSH. This can be set to other values in a number of ways. For example, there is one exception to the rule that SSH is used by default. If we do not have localhost in our inventory, Ansible sets up an implicit localhost entry to allow us to run adhoc commands and playbooks that target localhost. This special inventory entry is not included in the all or ungrouped host groups. In addition, instead of using the smart SSH connection type, ansible connects to it using the special local connection type by default. The local connection type ignores the remote\_user setting and runs commands directly on the local system. If privilege escalation is being used, it runs sudo from the user account that ran the Ansible command, not remote\_user. This can lead to confusion if the two users have different sudoprivileges

## **3.5 Implementing Playbooks**

### **3.5.1 Ansible Playbook Commands**

Ad hoc commands can run a single, simple task against a set of targeted hosts as a one-time command. The real power of Ansible, however, is in learning how to use playbooks to run multiple, complex tasks against a set of targeted hosts in an easily

repeatable manner. A play is an ordered set of tasks run against hosts selected from our inventory. A playbook is a text file containing a list of one or more plays to run in a specific order. Plays allow us to change a lengthy, complex set of manual administrative tasks into an easily repeatable routine with predictable and successful outcomes. In a playbook, we can save the sequence of tasks in a play into a human-readable and immediately runnable form. The tasks themselves, because of the way in which they are written, document the steps needed to deploy our application or infrastructure. A playbook is a text file written in YAML format, and is normally saved with the extension yml. The playbook uses indentation with space characters to indicate the structure of its data. YAML does not place strict requirements on how many spaces are used for the indentation, but there are two basic rules

- Data elements at the same level in the hierarchy (such as items in the same list) must have the same indentation.
- Items that are children of another item must be indented more than their parents

### 3.5.2 Writing Playbook

A playbook begins with a line consisting of three dashes (---) as a start of document marker. It may end with three dots (...) as an end of document marker, although in practice this is often omitted. In between those markers, the playbook is defined as a list of plays. An item in a YAML list starts with a single dash followed by a space. For example, a YAML list might appear as follows:

```
- apple  
- orange  
- grape
```

The play itself is a collection of key-value pairs. Keys in the same play should have the same indentation. The following example shows a YAML snippet with three keys. The first two keys have simple values. The third has a list of three items as a value

```
name: just an example  
hosts: localhost  
tasks:  
  - first  
  - second  
  - third
```

The original example play has three keys, name, hosts, and tasks, because these keys all have the same indentation. The first line of the example play starts with a dash and a space (indicating the play is the first item of a list), and then the first key, the name attribute. The name key associates an arbitrary string with the play as a label. This identifies what the play is for. The name key is optional, but is recommended because it helps to document our playbook. This is especially useful when a playbook contains multiple plays. The second key in the play is a host's attribute, which specifies the hosts

against which the play's tasks are run. Like the argument for the ansible command, the hosts attribute takes a host pattern as a value, such as the names of managed hosts or groups in the inventory. Finally, the last key in the play is the tasks attribute, whose value specifies a list of tasks to run for this play. This example has a single task, which runs the user module with specific arguments

tasks:

- name: newbie exists with UID 4000
  - user:
    - name: newbie
    - uid: 4000
    - state: present

The tasks attribute is the part of the play that actually lists, in order, the tasks to be run on the managed hosts. Each task in the list is itself a collection of key-value pairs.

In this example, the only task in the play has two keys:

- name is an optional label documenting the purpose of the task. It is a good idea to name all our tasks to help document the purpose of each step of the automation process.
- user is the module to run for this task. Its arguments are passed as a collection of key-value pairs, which are children of the module (name, uid, and state)

### 3.5.3 Writing Multiple Playbooks

A playbook is a YAML file containing a list of one or more plays. Remember that a single play is an ordered list of tasks to execute against hosts selected from the inventory. Therefore, if a playbook contains multiple plays, each play may apply its tasks to a separate set of hosts. This can be very useful when orchestrating a complex deployment which may involve different tasks on different hosts. We can write a playbook that runs one play against one set of hosts, and when that finishes runs another play against another set of hosts. Writing a playbook that contains multiple plays is very straightforward. Each play in the playbook is written as a top-level list item in the playbook. Each play is a list item containing the usual play keywords

### 3.5.4 Running Playbooks

The ansible-playbook command is used to run playbooks. The command is executed on the control node and the name of the playbook to be run is passed as an argument

```
[root@workstation ~]# ansible-playbook site.yml
```

When we run the playbook, output is generated to show the play and tasks being executed. The output also reports the results of each task executed.

### 3.5.5 Syntax Verification

Prior to executing a playbook, it is good practice to perform a verification to ensure that the syntax of its contents is correct. The `ansible-playbook` command offers a `--syntax-check` option that we can use to verify the syntax of a playbook. The following example shows the successful syntax verification of a playbook

```
root@workstation # ansible-playbook --syntax-check (playbook name). yml
```

### 3.5.6 Increasing Output Verbosity.

The default output provided by the `ansible-playbook` command does not provide detailed task execution information. The `ansible-playbook -v` command provides additional information, with up to four total levels. Configuring the Output Verbosity of Playbook Execution

| OPTION | DESCRIPTION                                                                                                                                                      |
|--------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| -v     | The task results are displayed                                                                                                                                   |
| -vv    | Both task results and task configuration are displayed                                                                                                           |
| -vvv   | Includes information about connections to managed hosts                                                                                                          |
| -vvvv  | Adds extra verbosity options to the connection plug-ins, including users being used in the managed hosts to execute scripts, and what scripts have been executed |

The following example shows the contents of a simple playbook, and then the result of running it.

```
[student@workstation playdemo] $ cat webserver.yml
```

```
---
```

```
- name: play to setup web server
  hosts: servera.lab.example.com
```

```
tasks:
- name: latest httpd version installed
yum:
name: httpd
state: latest
...output omitted...
```

```
[student@workstation playdemo] $ ansible-playbook webserver.yml
```

```
PLAY [play to setup web server] *****
```

```
TASK [Gathering Facts] *****
ok: [servera.lab.example.com]
```

```
TASK [latest httpd version installed] *****
changed: [servera.lab.example.com]
```

```
PLAY RECAP *****
servera.lab.example.com: ok=2 changed=1 unreachable=0 failed=0
```

Note that the value of the name key for each play and task is displayed when the playbook is run. (The Gathering Facts task is a special task that the setup module usually runs automatically at the start of a play. This is covered later in the course.) For playbooks with multiple plays and tasks, setting name attributes makes it easier to monitor the progress of a playbook's execution. We should also see that the latest httpd version installed task is changed for servera.lab.example.com. This means that the task changed something on that host to ensure its specification was met. In this case, it means that the httpd package probably was not installed or was not the latest version. In general, tasks in Ansible Playbooks are idempotent, and it is safe to run a playbook multiple time. If the targeted managed hosts are already in the correct state, no changes should be made

## 3.6 Remote Users and Privilege Escalation in Plays

Plays can use different remote users or privilege escalation settings for a play than what is specified by the defaults in the configuration file. These are set in the play itself at the same level as the hosts or tasks keywords

### 3.6.1 UserAttributes

Tasks in playbooks are normally executed through a network connection to the managed hosts. As with ad hoc commands, the user account used for task execution depends on various keywords in the Ansible configuration file, /etc/ansible/ansible.cfg. The user that runs the tasks can be defined by the remote\_user keyword. However, if

privilege escalation is enabled, other keywords such as `become_user` can also have an impact. If the remote user defined in the Ansible configuration for task execution is not suitable, it can be overridden by using the `remote_user` keyword within a play.

```
remote_user: ansible
```

### 3.6.2 Privilege escalation

Additional keywords are also available to define privilege escalation parameters from within a playbook. The `become` Boolean keyword can be used to enable or disable privilege escalation regardless of how it is defined in the Ansible configuration file. It can take `yes` or `true` to enable

privilege escalation, or `no` or `false` to disable it.

```
become: true
```

If privilege escalation is enabled, the `become_method` keyword can be used to define the privilege escalation method to use during a specific play. The example below specifies that `sudo` be used for privilege escalation. `become_method: sudo`

Additionally, with privilege escalation enabled, the `become_user` keyword can define the user account to use for privilege escalation within the context of a specific play.

```
become_user: privileged_user
```

## 3.7 Finding Modules For tasks

### 3.7.1 Module Documentation

The large number of modules packaged with Ansible provides administrators with many tools for common administrative tasks. The Module Index on the website is an easy way to browse the list of modules shipped with Ansible. For example, modules for user and service management can be found under Systems Modules and modules for database administration can be found under Database Modules. For each module, the Ansible documentation website provides a summary of its functions and instructions on how each specific function can be invoked with options to the module. The documentation also provides useful examples that shows us how to use each module and how to set their keywords in a task. We have already worked with the `ansible-doc` command to look up information about modules installed on the local system. As a review, to see a list of the modules available on a control node, run the `ansible-doc -l` command. This displays a list of module names and a synopsis of their functions

```
[root@workstation modules] # ansible-doc -l
```

```
a10_server          Manage A10 Networks ... devices' server object.
a10_server_axapi3    Manage A10 Networks ... devices
a10_service_group    Manage A10 Networks ... devices' service groups.
a10_virtual_server   Manage A10 Networks ... devices' virtualservers.
...output omitted...
```

|                   |                                               |
|-------------------|-----------------------------------------------|
| zfs_facts         | Gather facts about ZFS datasets.              |
| znode             | Create, ... and update znodes using ZooKeeper |
| zpool_facts       | Gather facts about ZFS pools.                 |
| zipper            | Manage packages on SUSE and openSUSE          |
| zypper_repository | Add and remove Zypper repositories            |

Use the `ansible-doc [module name]` command to display detailed documentation for a module. Like the Ansible documentation website, the command provides a synopsis of the module's function, details of its various options, and examples. The following example shows the documentation displayed for the `yum` module

### 3.7.2 Module Maintenance

Ansible ships with a large number of modules that can be used for many tasks. The upstream community is very active, and these modules may be in different stages of development. The `ansible-doc` documentation for the module is expected to specify who maintains that module in the upstream Ansible community, and what its development status is. This is indicated in the METADATA section at the end of the output of `ansible-doc` for that module.

The status field records the development status of the module:

- **stable interface:** The module's keywords are stable, and every effort will be made not to remove keywords or change their meaning.
- **preview:** The module is in technology preview, and might be unstable, its keywords might change, or it might require libraries or web services that are themselves subject to incompatible changes.
- **deprecated:** The module is deprecated, and will no longer be available in some future release.
- **removed:** The module has been removed from the release, but a stub exists for documentation purposes to help former users migrate to new modules.
- **core:** Maintained by the "core" Ansible developers upstream, and always included with Ansible.
- **curated:** Modules submitted and maintained by partners or companies in the community. Maintainers of these modules must watch for any issues reported or pull requests raised against the module. Upstream "core" developers review proposed changes to curated modules after the community maintainers have approved the changes. Core committers also ensure that any issues with these modules due to changes in the Ansible engine are remediated. These modules are currently included with Ansible, but might be packaged separately at some point in the future.
- **community:** Modules not supported by the core upstream developers, partners, or companies, but maintained entirely by the general open source community. Modules in this category are still fully usable, but the

response rate to issues is purely up to the community. These modules are also currently included with Ansible, but will probably be packaged separately at some point in the future.

## CHAPTER 4

### APPENDIX

#### 4.1 Screenshots

```
File Edit View Search Terminal Help
[root@localhost ~]# dnf install epel-release
Last metadata expiration check: 0:00:58 ago on Thursday 20 August 2020 03:06:23 AM EDT.
Modular dependency problems:
  Problem 1: conflicting requests
    - nothing provides module(perl:5.30) needed by module perl-FCGI:0.78:803002020
0618214557:2fbcb20-0.x86_64
  Problem 2: conflicting requests
    - nothing provides module(perl:5.30) needed by module perl-YAML:1.24:803002020
0723225335:a6d43775-0.x86_64
Dependencies resolved.
=====
Package                Architecture Version      Repository    Size
=====
Installing:
epel-release            noarch      8-8.el8       extras        23 k
=====
Transaction Summary
=====
Install 1 Package

Total download size: 23 k
Installed size: 32 k
Is this ok [y/N]: y
Downloading packages:
epel-release-8-8.el8.noarch.rpm                68 kB/s | 23 kB    00:00
-----
Total                                          21 kB/s | 23 kB    00:01
warning: /var/cache/dnf/extras-2770d521ba03e231/packages/epel-release-8-8.el8.no
arch.rpm: Header V3 RSA/SHA256 Signature, key ID 8483c65d: NOKEY
CentOS-8 - Extras                               1.6 MB/s | 1.6 kB    00:00
Importing GPG key 0x8483c65d:
  Userid : "CentOS (CentOS Official Signing Key) <security@centos.org>"
  Fingerprint: 99DB 70FA E1D7 CE22 7FB6 4882 0585 55B3 8483 C65D
  From    : /etc/pki/rpm-gpg/RPM-GPG-KEY-centosofficial
Is this ok [y/N]: y
Key imported successfully
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing                : 1/1
  Installing               : epel-release-8-8.el8.noarch 1/1
  Running scriptlet        : epel-release-8-8.el8.noarch 1/1
  Verifying                : epel-release-8-8.el8.noarch 1/1

Installed:
epel-release-8-8.el8.noarch
```

Installed:

```
ansible-2.9.11-1.el8.noarch      python3-paramiko-2.4.3-1.el8.noarch
python3-babel-2.5.1-5.el8.noarch python3-jinja2-2.10.1-2.el8_0.noarch
python3-jmespath-0.9.0-11.el8.noarch python3-markupsafe-0.23-19.el8.x86_64
python3-pyasnl-0.3.7-6.el8.noarch  libsodium-1.0.18-2.el8.x86_64
python3-bcrypt-3.1.6-2.el8.1.x86_64 python3-pynacl-1.3.0-5.el8.x86_64
sshpas-1.06-9.el8.x86_64
```

Complete!

```
[root@localhost ~]# ansible --version
```

```
ansible 2.9.11
```

```
  config file = /etc/ansible/ansible.cfg
```

```
  configured module search path = ['/root/.ansible/plugins/modules', '/usr/share
/ansible/plugins/modules']
```

```
  ansible python module location = /usr/lib/python3.6/site-packages/ansible
```

```
  executable location = /usr/bin/ansible
```

```
  python version = 3.6.8 (default, Nov 21 2019, 19:31:34) [GCC 8.3.1 20190507 (R
ed Hat 8.3.1-4)]
```

```
student@workstation:~/ansible
File Edit View Search Terminal Help
1 [server1]
2 node1
3 [server2]
4 node2
5 [server3]
6 node3
7 [server4]
8 node4
9 [server5]
10 node5
11 [server6]
12 node6
13 [server7]
14 node7
15 [server8]
16 node8
17
18
19
10,5 All
```

```
student@workstation:~/ansible
File Edit View Search Terminal Help
1 [defaults]
2 remote_user=student
3 inventory=/home/student/ansible/inventory
4 roles_path=/home/student/ansible/roles
5
6 [privilege_escalation]
7 become=true
4,1 All
```

```
student@workstation:~/ansible
File Edit View Search Terminal Help
1 ---
2 - name: hwreport
3   hosts: all
4   ignore_errors: yes
5   tasks:
6     - name: Download the file
7       get_url:
8         url: "https://github.com/Balajil102/hwrpt/blob/main/hwrpt.txt"
9         dest: /root/hwreport.txt
10    - name: Collect report 1
11      set_fact:
12        HOSTNAME: "{{ ansible_hostname }}"
13        MEMORY: "{{ ansible_memtotal_mb }}"
14        BIOS: "{{ ansible_bios_version }}"
15        CPU: "{{ ansible_processor }}"
16        DISK_SIZE_VDA: "{{ ansible_devices['vda']['size'] }}"
17    - name: Collect report 2
18      set_fact:
19        DISK_SIZE_VDB: "{{ ansible_devices['vdb']['size'] }}"
20    - name: Copy the content to the managed node
21      copy:
22        content: |
23          #hwreport
24          HOSTNAME={{ HOSTNAME | default('NONE') }}
25          MEMORY={{ MEMORY | default('NONE') }}
26          BIOS={{ BIOS | default('NONE') }}
27          CPU={{ CPU | default('NONE') }}
28          DISK_SIZE_VDA={{ DISK_SIZE_VDA | default('NONE') }}
29          DISK_SIZE_VDB={{ DISK_SIZE_VDB | default('NONE') }}
30        dest: /root/hwreport.txt
```

## Final Output

```
student@workstation:~/ansible
File Edit View Search Terminal Help
node2 | CHANGED | rc=0 >>
#hwreport
HOSTNAME=node2
MEMORY=809
BIOS=ArchLinux 1.14.0-1
CPU=['0', 'AuthenticAMD', 'AMD EPYC-Rome Processor', '1', 'AuthenticAMD', 'AMD EPYC-Rome Processor']
DISK_SIZE_VDA=5.00 GB
DISK_SIZE_VDB=NONE
node4 | CHANGED | rc=0 >>
#hwreport
HOSTNAME=node4
MEMORY=809
BIOS=ArchLinux 1.14.0-1
CPU=['0', 'AuthenticAMD', 'AMD EPYC-Rome Processor', '1', 'AuthenticAMD', 'AMD EPYC-Rome Processor']
DISK_SIZE_VDA=5.00 GB
DISK_SIZE_VDB=NONE
node5 | CHANGED | rc=0 >>
#hwreport
HOSTNAME=node5
MEMORY=809
BIOS=ArchLinux 1.14.0-1
CPU=['0', 'AuthenticAMD', 'AMD EPYC-Rome Processor', '1', 'AuthenticAMD', 'AMD EPYC-Rome Processor']
DISK_SIZE_VDA=5.00 GB
DISK_SIZE_VDB=NONE
```

```
student@workstation:~/ansible
File Edit View Search Terminal Help
node1 | CHANGED | rc=0 >>
#hwreport
  HOSTNAME=node1
  MEMORY=809
  BIOS=ArchLinux 1.14.0-1
  CPU=['0', 'AuthenticAMD', 'AMD EPYC-Rome Processor', '1', 'AuthenticAMD', 'AMD EPYC-Rome Processor']
  DISK_SIZE_VDA=5.00 GB
  DISK_SIZE_VDB=NONE
node3 | CHANGED | rc=0 >>
#hwreport
  HOSTNAME=node3
  MEMORY=809
  BIOS=ArchLinux 1.14.0-1
  CPU=['0', 'AuthenticAMD', 'AMD EPYC-Rome Processor', '1', 'AuthenticAMD', 'AMD EPYC-Rome Processor']
  DISK_SIZE_VDA=5.00 GB
  DISK_SIZE_VDB=NONE
node6 | CHANGED | rc=0 >>
#hwreport
  HOSTNAME=node6
  MEMORY=809
  BIOS=ArchLinux 1.14.0-1
  CPU=['0', 'AuthenticAMD', 'AMD EPYC-Rome Processor', '1', 'AuthenticAMD', 'AMD EPYC-Rome Processor']
  DISK_SIZE_VDA=5.00 GB
  DISK_SIZE_VDB=NONE
```

```
#hwreport
  HOSTNAME=node8
  MEMORY=809
  BIOS=ArchLinux 1.14.0-1
  CPU=['0', 'AuthenticAMD', 'AMD EPYC-Rome Processor', '1', 'AuthenticAMD', 'AMD EPYC-Rome Processor']
  DISK_SIZE_VDA=5.00 GB
  DISK_SIZE_VDB=NONE
node7 | CHANGED | rc=0 >>
#hwreport
  HOSTNAME=node7
  MEMORY=809
  BIOS=ArchLinux 1.14.0-1
  CPU=['0', 'AuthenticAMD', 'AMD EPYC-Rome Processor', '1', 'AuthenticAMD', 'AMD EPYC-Rome Processor']
  DISK_SIZE_VDA=5.00 GB
  DISK_SIZE_VDB=NONE
```

## 4.2 Source code

- 
- name: hwreport
  - hosts: all
  - ignore\_errors: yes
  - tasks:
    - name: Download the file
      - get\_url:
        - url: "http://content.example.com/Rhce/hwreport.txt"
        - dest: /root/hwreport.txt
    - name: Collect report 1
      - set\_fact:

```

    HOSTNAME: "{{ ansible_hostname }}"
    MEMORY: "{{ ansible_memtotal_mb }}"
    BIOS: "{{ ansible_bios_version }}"
    CPU: "{{ ansible_processor }}"
    DISK_SIZE_VDA: "{{ ansible_devices[vda]['size'] }}"
- name: Collect report 2
  set_fact:
    DISK_SIZE_VDB: "{{ ansible_devices[vdb]['size'] }}"
- name: Copy the content to the managed node
- copy:
  content: |
    #hwreport
    HOSTNAME={{ HOSTNAME | default('NONE') }}
    MEMORY={{ MEMORY | default('NONE') }}
    BIOS={{ BIOS | default('NONE') }}
    CPU={{ CPU | default('NONE') }}
    DISK_SIZE_VDA={{ DISK_SIZE_VDA | default('NONE') }}
    DISK_SIZE_VDB={{ DISK_SIZE_VDB | default('NONE') }}
  dest: /root/hwreport.txt

```

# ansible-playbook hwreport.yml

### 4.3 Reference

- <http://www.redhat.com/>
- <http://www.centos.org/>
- <https://www.ansible.com/>
- <https://www.apache.org/>
- <https://httpd.apache.org/>