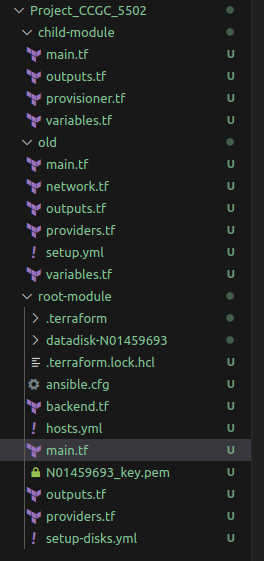
**CCGC 5502 Automation**

**Project**

**Student: Kate Karakow**

**Tree:**

****

**Terafform code for the main.tf in root that created 2 linux vms and 1 windows vm:**

**variable "location" {**

**description = "The Azure location where resources will be created"**

**default = "eastus"**

**}**

**variable "resource\_group\_name" {**

**description = "The name of the resource group"**

**default = "N01459693-group"**

**}**

**# Resource Group**

**resource "azurerm\_resource\_group" "main" {**

**name = var.resource\_group\_name**

**location = var.location**

**}**

**# Virtual Network and Subnet**

**resource "azurerm\_virtual\_network" "main\_vnet" {**

**name = "vmVNet"**

**address\_space = ["10.0.0.0/16"]**

**location = azurerm\_resource\_group.main.location**

**resource\_group\_name = azurerm\_resource\_group.main.name**

**}**

**resource "azurerm\_subnet" "main\_subnet" {**

**name = "vmSubnet"**

**resource\_group\_name = azurerm\_resource\_group.main.name**

**virtual\_network\_name = azurerm\_virtual\_network.main\_vnet.name**

**address\_prefixes = ["10.0.1.0/24"]**

**}**

**# SSH Public Key Data Source**

**data "azurerm\_ssh\_public\_key" "main\_key" {**

**name = "N01459693\_key"**

**resource\_group\_name = var.resource\_group\_name**

**}**

**# Public IPs for all VMs**

**resource "azurerm\_public\_ip" "vm\_public\_ip" {**

**count = 3 # Total VMs = 2 Linux VMs + 1 Windows VM**

**name = "vmPublicIP-${count.index + 1}"**

**location = azurerm\_resource\_group.main.location**

**resource\_group\_name = azurerm\_resource\_group.main.name**

**allocation\_method = "Dynamic"**

**}**

**# Network Interface for all VMs**

**resource "azurerm\_network\_interface" "vm\_nic" {**

**count = 3 # Total VMs = 2 Linux VMs + 1 Windows VM**

**name = "vmNIC-${count.index + 1}"**

**location = azurerm\_resource\_group.main.location**

**resource\_group\_name = azurerm\_resource\_group.main.name**

**ip\_configuration {**

**name = "internal"**

**subnet\_id = azurerm\_subnet.main\_subnet.id**

**private\_ip\_address\_allocation = "Dynamic"**

**public\_ip\_address\_id = azurerm\_public\_ip.vm\_public\_ip[count.index].id**

**}**

**}**

**# Linux Virtual Machines**

**resource "azurerm\_linux\_virtual\_machine" "linux\_vm" {**

**count = 2**

**name = "N01459693-c-vm${count.index + 1}"**

**resource\_group\_name = azurerm\_resource\_group.main.name**

**location = azurerm\_resource\_group.main.location**

**size = "Standard\_B1s"**

**admin\_username = "adminuser"**

**network\_interface\_ids = [element(azurerm\_network\_interface.vm\_nic.\*.id, count.index)]**

**os\_disk {**

**caching = "ReadWrite"**

**storage\_account\_type = "Standard\_LRS"**

**disk\_size\_gb = 10**

**}**

**source\_image\_reference {**

**publisher = "OpenLogic"**

**offer = "CentOS"**

**sku = "8\_2"**

**version = "latest"**

**}**

**admin\_ssh\_key {**

**username = "adminuser"**

**public\_key = data.azurerm\_ssh\_public\_key.main\_key.public\_key**

**}**

**}**

**# Windows Virtual Machine**

**resource "azurerm\_windows\_virtual\_machine" "windows\_vm" {**

**name = "N01459693-w-vm1"**

**resource\_group\_name = azurerm\_resource\_group.main.name**

**location = azurerm\_resource\_group.main.location**

**size = "Standard\_B1s"**

**admin\_username = "adminuser"**

**admin\_password = "AComplexP@ssw0rd!"**

**network\_interface\_ids = [azurerm\_network\_interface.vm\_nic[2].id]**

**os\_disk {**

**caching = "ReadWrite"**

**storage\_account\_type = "Standard\_LRS"**

**disk\_size\_gb = 10**

**}**

**source\_image\_reference {**

**publisher = "MicrosoftWindowsServer"**

**offer = "WindowsServer"**

**sku = "2019-Datacenter"**

**version = "latest"**

**}**

**}**

**# Network Security Group to allow ICMP (ping)**

**resource "azurerm\_network\_security\_group" "allow\_icmp" {**

**name = "allow-icmp-nsg"**

**location = azurerm\_resource\_group.main.location**

**resource\_group\_name = azurerm\_resource\_group.main.name**

**security\_rule {**

**name = "allow-inbound-icmp"**

**priority = 100**

**direction = "Inbound"**

**access = "Allow"**

**protocol = "Icmp"**

**source\_port\_range = "\*"**

**destination\_port\_range = "\*"**

**source\_address\_prefix = "\*"**

**destination\_address\_prefix = "\*"**

**}**

**# Add SSH rule**

**security\_rule {**

**name = "allow-inbound-ssh"**

**priority = 110 # Ensure this is unique and not conflicting with other priorities**

**direction = "Inbound"**

**access = "Allow"**

**protocol = "Tcp" # SSH uses TCP**

**source\_port\_range = "\*"**

**destination\_port\_range = "22" # Default SSH port**

**source\_address\_prefix = "\*"**

**destination\_address\_prefix = "\*"**

**}**

**security\_rule {**

**name = "allow-inbound-rdp"**

**priority = 120 # Ensure this is unique and not conflicting with other priorities**

**direction = "Inbound"**

**access = "Allow"**

**protocol = "Tcp" # RDP uses TCP**

**source\_port\_range = "\*"**

**destination\_port\_range = "3389" # Default RDP port**

**source\_address\_prefix = "\*" # Allow any IP to access**

**destination\_address\_prefix = "\*"**

**}**

**}**

**# Associate NSG with Network Interface of VMs**

**resource "azurerm\_network\_interface\_security\_group\_association" "vm\_nic\_nsg\_association" {**

**count = 3 # Assuming 3 VMs as per your setup**

**network\_interface\_id = azurerm\_network\_interface.vm\_nic[count.index].id**

**network\_security\_group\_id = azurerm\_network\_security\_group.allow\_icmp.id**

**}**

Project Script:

*Present terraform code and tell this:*

This Terraform configuration script is designed to deploy a network infrastructure along with multiple virtual machines (VMs) on Microsoft Azure. The script sets up the necessary Azure resources to support both Linux and Windows VMs, making it versatile for various application needs.

### Resource Group

The script starts by defining a resource group named N01459693-group in the eastus Azure region. This resource group serves as a container that holds related Azure resources for the project, providing a way to manage and organize them collectively.

### Virtual Network and Subnet

Within the resource group, it creates a virtual network (main\_vnet) with an address space of 10.0.0.0/16, which is a private IP address range commonly used within internal networks. A subnet (main\_subnet) is also defined within this virtual network, with a smaller address range of 10.0.1.0/24. This setup provides a segmented network environment where resources can communicate securely.

### SSH Public Key Data Source

The script retrieves a public SSH key named N01459693\_key from Azure, intended to be used for authenticating SSH sessions to the Linux VMs. This enhances security by using cryptographic keys instead of passwords for SSH access.

### Public IPs and Network Interfaces

To enable external access to the VMs, the script creates dynamic public IP addresses for three VMs. Associated with these IPs, network interfaces (vm\_nic) are defined, providing the VMs with network connectivity both internally (within the virtual network) and externally (through the public IPs).

### Virtual Machines

Two Linux VMs and one Windows VM are provisioned. The Linux VMs (azurerm\_linux\_virtual\_machine) are configured with the CentOS 8.2 operating system, using previously defined network interfaces and the SSH key for access. The Windows VM (azurerm\_windows\_virtual\_machine) is set up with Windows Server 2019, using a password for administrative access. All VMs are configured with standard B1s size (1 vcpu, 1 GiB memory), which is cost-effective and suitable for light workloads typical in educational projects.

### Network Security Group

A network security group (allow\_icmp) is created to define and control network traffic rules. It includes rules to allow ICMP (ping) traffic, SSH (port 22) for Linux VMs, and RDP (port 3389) for the Windows VM. These rules ensure that the VMs can be accessed for management and monitoring while maintaining control over network access to enhance security.

### Association of NSG with VMs

Finally, the network security group is associated with the network interfaces of all VMs, applying the defined security rules to each VM to regulate inbound and outbound traffic according to the policy.

**This Terraform script effectively sets up a complete virtual network environment with multiple VMs, demonstrating a practical implementation of cloud resources for educational or developmental purposes in Azure. It showcases the automation of cloud infrastructure deployment, which is crucial for scalable and reproducible environments in modern IT landscapes.**

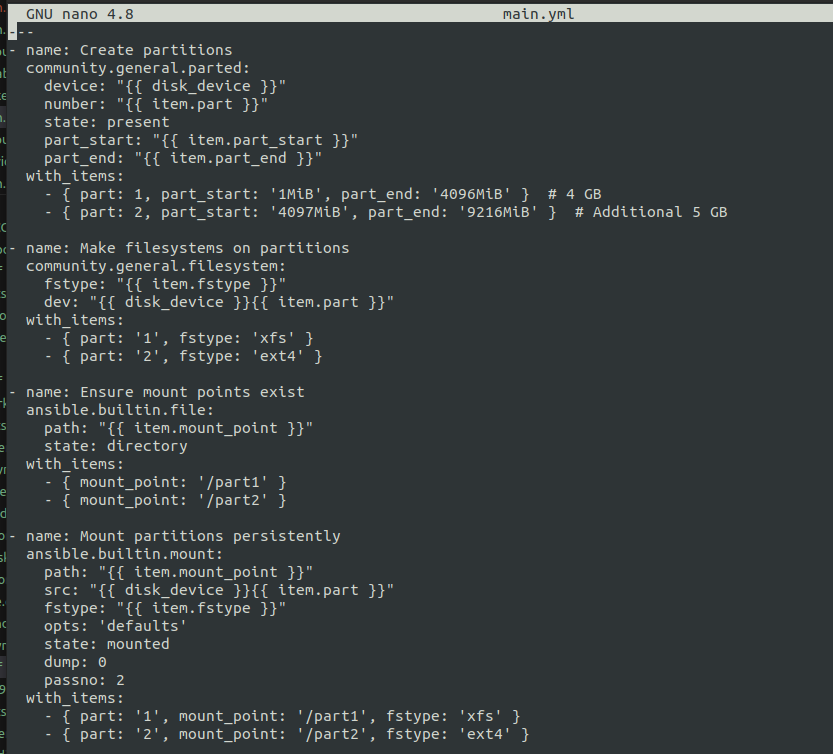
**Phase I – Development**

**Develop a Role for Disk Configuration:**

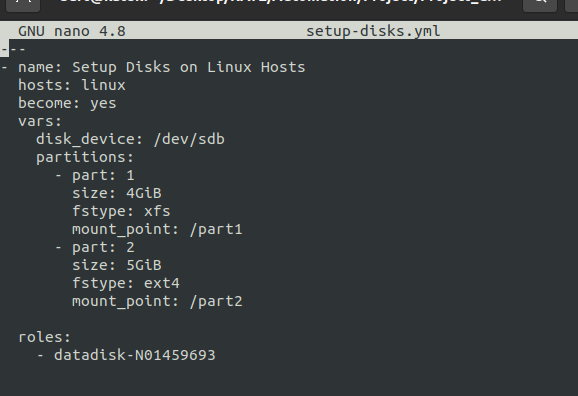
**This playbook performs a duet of tasks: it begins by partitioning a storage device into two distinct sections, each with its own specific purpose and filesystem.**

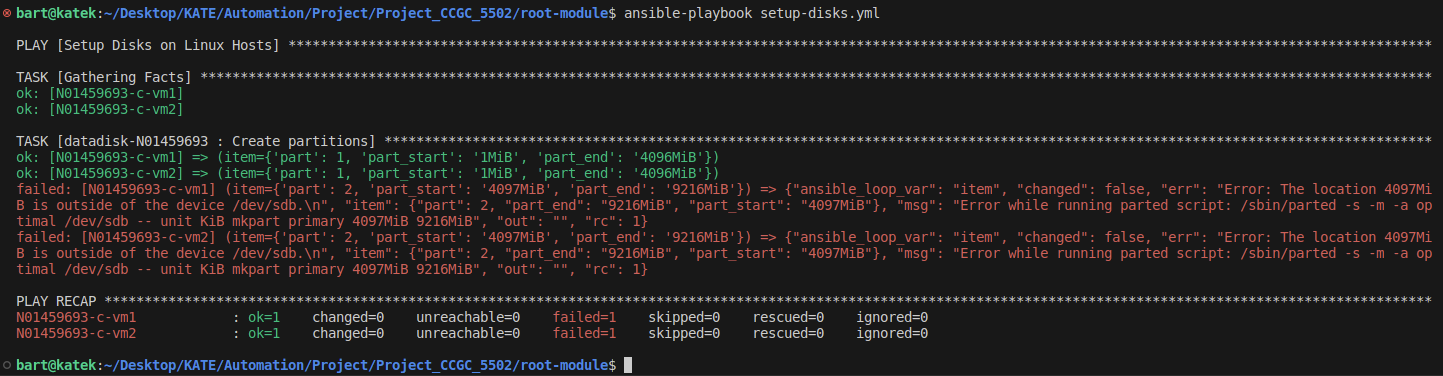
The first partition is like the lead violinist, with a 4 GB space finely tuned with an 'XFS' filesystem, known for its performance and reliability. It's the soloist that carries the melody of high-performance tasks with grace. The second partition is the cello, a robust 5 GB section that provides the depth and stability of an 'EXT4' filesystem, underpinning the soloist with its steadfastness.

But our playbook goes beyond just crafting these sections; it ensures that they have a stage to perform on—by creating mount points. These are like music stands, one for '/part1' and another for '/part2', each ready to display the scores of data they'll be tasked to play.

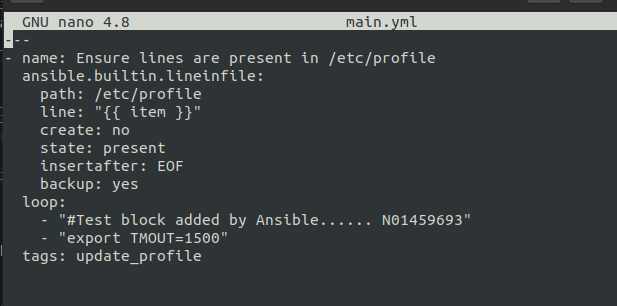
****

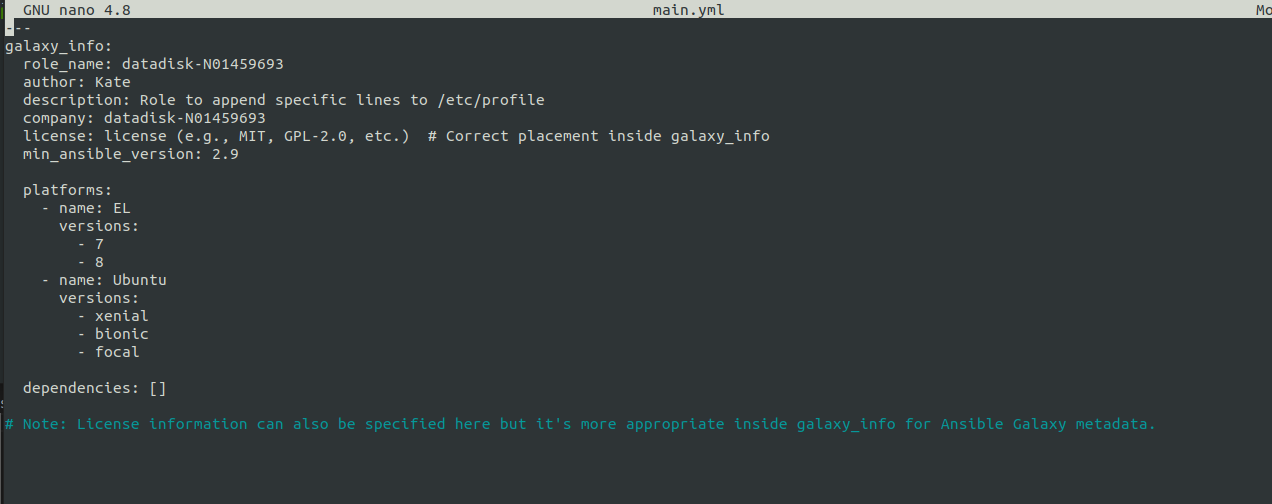
**I was not able to create the second partition.**

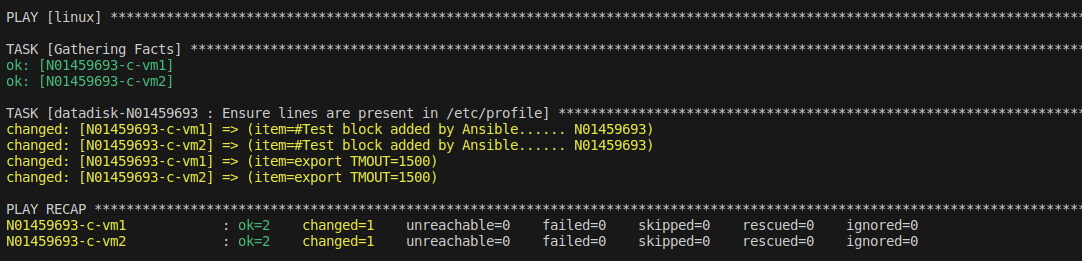
****

****

**Develop a Role for File Update:**

****



****

**This Ansible playbook is designed to manage system environment settings for Linux servers. It specifically targets the /etc/profile file, which is used to set environment variables globally for all system users. The playbook consists of tasks executed under the role named profile-N01459693.**

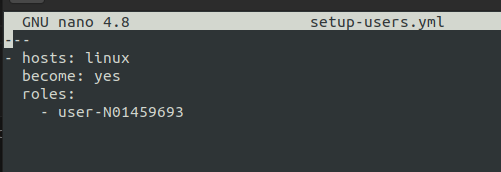
The main operation of this playbook is to append two lines at the end of the /etc/profile file. These lines are:

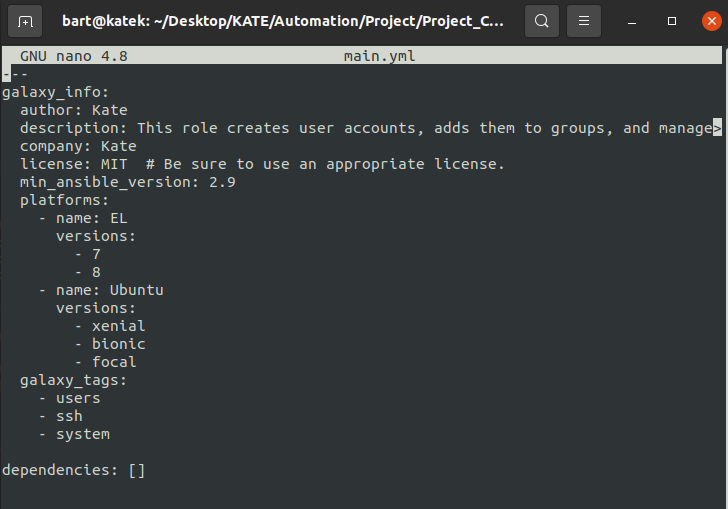
1. A comment line that reads #Test block added by Ansible...... N01459693. This serves as an identifier for the changes made by this playbook, allowing system administrators to quickly recognize that this block of configuration was automatically added by an Ansible playbook.
2. An environment variable export statement export TMOUT=1500. This sets a timeout for the shell, which means if the user's session remains inactive for the specified number of seconds (1500 in this case), it will automatically log out for security reasons.

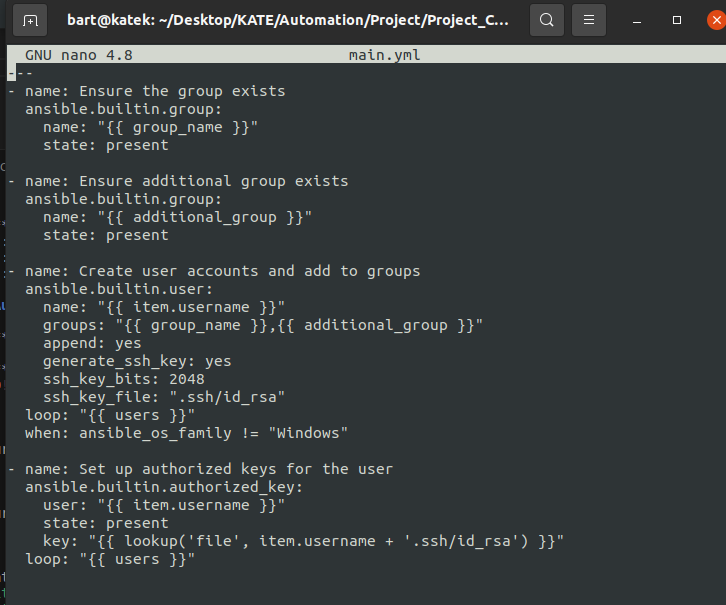
The tasks include checking if the lines are already present to avoid duplicates and ensuring they are correctly added if not present. The playbook uses the ansible.builtin.lineinfile module to manage the file's content, which is efficient for such line-based manipulations.

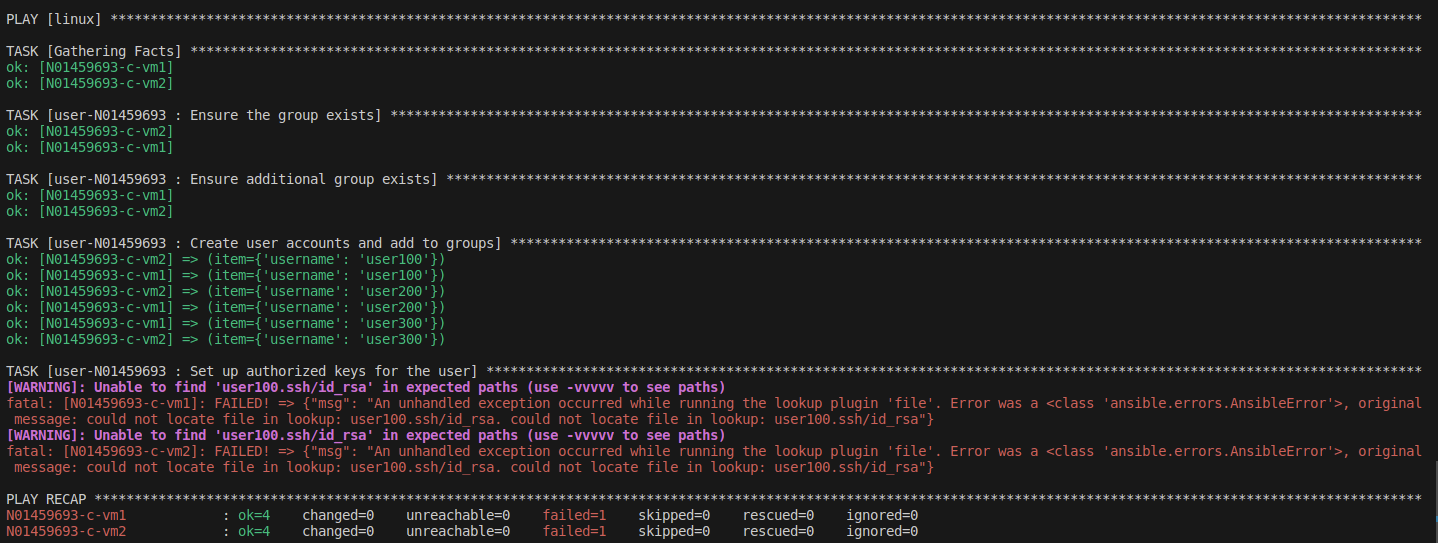
Upon successful execution, the playbook will report back indicating that the tasks have been completed, which ensures that the environment variable is set and will persist across system reboots and new user logins. This makes sure that the server environment is configured consistently as per the requirements set out in the playbook.

**Develop a Role for User/Group Creation:**

****

****

****



**This Ansible playbook is part of a project to create user accounts on Linux systems. It includes a role named user-N01459693, which has a specific set of tasks to ensure that user accounts and groups are set up as specified.**

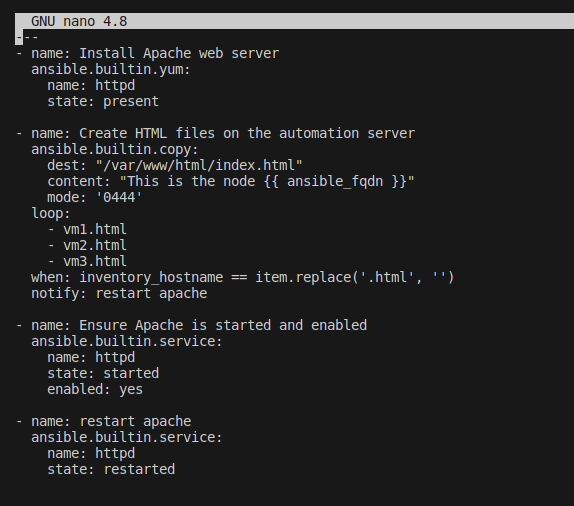
Here's what each part of the playbook is intended to do:

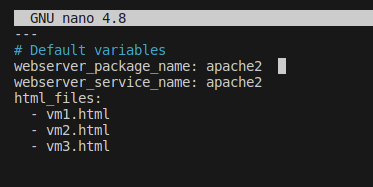
* Ensure Group Exists: This task creates a group if it doesn't already exist, ensuring that any user accounts created can be assigned to this group. It uses Ansible's built-in group module, which is a straightforward way to manage groups on a system.
* Ensure Additional Group Exists: Similar to the first task, this ensures another group is present, perhaps for finer access control or categorizing users differently.
* Create User Accounts and Add to Groups: This task creates user accounts and adds them to the specified groups. The user module in Ansible is very flexible, allowing not only to create users but also to add them to groups, manage home directories, and even remove users when necessary.
* Generate SSH Keys: For secure remote access, SSH keys are generated for each user without a passphrase, which can be used for passwordless authentication. It's a common practice in automation to use SSH keys instead of passwords for security and convenience.
* Set Up Authorized Keys for the User: This part of the playbook makes sure that the SSH public key generated in the previous step is added to the authorized\_keys file of the user's account, allowing them to securely log in using SSH.

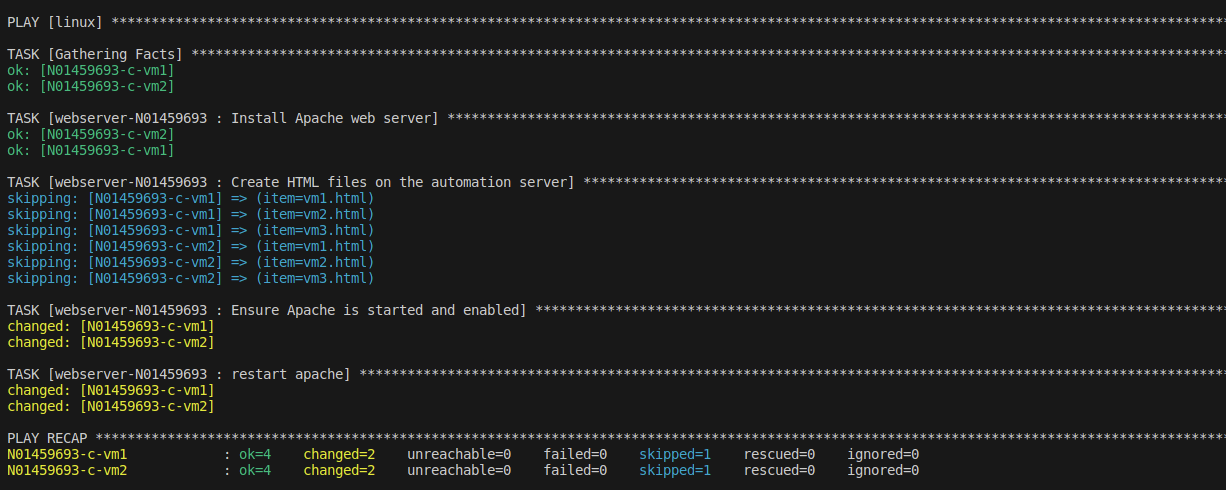
The output indicates whether each task has been executed successfully (ok), changed something on the system (changed), or failed (failed). In a typical playbook run, you'd want all tasks to succeed without any failures.

If this playbook is executed correctly, it will ensure that all specified Linux systems have the necessary user accounts and groups set up with the right SSH access, providing a consistent and secure user management across your infrastructure.

**Develop a Role to Configure a Load-Balanced Website:**

****





This Ansible playbook is designed for configuring a load-balanced website by automating the deployment of an Apache web server across multiple nodes. Here’s a breakdown of the playbook and its output:

1. Install Apache Web Server: The playbook begins with a task that installs the Apache web server using the Yum package manager. This is a standard step to ensure that the HTTP server is present on the system and ready to serve web content.
2. Create HTML Files: Next, the playbook creates HTML files named vm1.html, vm2.html, and vm3.html. These files contain the fully qualified domain name (FQDN) of the respective node where they're created, providing a simple way to verify load balancing. The files are set with read-only permissions (0444), meaning they can be read by any user but cannot be modified or executed.
3. Ensure Apache is Started and Enabled: After installing Apache and setting up the web pages, the playbook ensures that the Apache service is running and enabled to start on boot. This step is crucial for maintaining the availability of the web service even after the system reboots.
4. Restart Apache: If any changes are made that require the Apache service to pick up new configurations, the playbook includes a handler to restart Apache. This means any new settings, such as the addition of HTML files, will be recognized by the web server once it restarts.
5. Playbook Output: The output displayed on the terminal provides feedback on the execution of the playbook. Each task is marked with ok if it was successful and didn't need to make any changes, changed if it successfully made changes to the system, and failed if any errors occurred. The summary at the end (PLAY RECAP) shows the overall status of the playbook run for each node, allowing the user to quickly identify any issues.

In the presentation, you might say:

"By automating the setup with this Ansible playbook, we streamline the deployment of our Apache web servers across multiple nodes, ensuring consistency in configuration and simplifying our workload. As the output indicates, each step is checked and verified, allowing us to deploy or update our load-balanced website efficiently and with confidence."