CAPSTONE Project:

**Task 1:** Infrastructure Provisioning

In this task, you will **provision infrastructure using Terraform**, automating the setup of cloud resources on AWS and Azure for a seamless, scalable, and efficient deployment. This task is divided into two subtasks: AWS and Azure.

Aws sample code:

<https://github.com/orgs/openwriteup/repositories>

system:

networking

instances

AWS Setup

In the AWS part of this task, you will:

Configure VPC with two public and two private subnets and Internet Gateway (IGW), NAT Gateway, and Route Tables. You may use a modular or resource-based Terraform approach.

Create a security group with ports 22 (SSH) and 80 (HTTP) open to all and generate an EC2 key pair.

Deploy:

Two EC2 instances in the public subnet using the pre-created security group and key pair.

Designate one EC2 instance as the App Machine and the other as the Tools Machine.

Use Ubuntu OS for both instances.

Azure Setup

In the Azure part of this task, you will:

<https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs>

<https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/guides/service_principal_client_secret>

Configure Virtual Network (VNet), subnets, and Network Security Groups (NSGs).

Deploy:

<https://learn.microsoft.com/en-us/azure/virtual-network/quickstart-create-virtual-network?tabs=terraform>

Virtual Machines (VMs) for application hosting, keeping 22 and 80 ports open.

Ensure SSH connectivity to both AWS and Azure instances from your local machine to verify if you have correctly set up the services.

<https://github.com/orgs/openwriteup/repositories>

**Task 2: Configuration Management**

In this task, you will:

* Install Ansible on the Tools Machine (AWS). (EC2 vm)
* Create an Ansible inventory file listing the details of AWS (App Machine) and Azure (VM).
* Develop an Ansible playbook to:
  + Install and configure Nginx web server on both AWS and Azure machines.
  + Ensure the Nginx process starts automatically and verify its running status in the playbook file.

To verify if Nginx is actively running, access the application via HTTP (IP:80) and confirm if both instances show the text, Welcome to Nginx.

ansible controller and manage nodes (2 aws and 2 azure)

control node-->ec2 vm

inventory

aws1 (public ip addr)

aws2

az1

az2

ec2-->control node

ssh-keygen -t rsa (public/private key pair)

ssh-copy-id az1@ip

ssh-copy-id aws@ip

This will enable passwordless login

<https://github.com/amitopenwriteup/ansiblelab/blob/main/lab1.txt>

<https://github.com/amitopenwriteup/ansiblelab/blob/main/lab3.txt>

**Task 3: Application Deployment**

In this task, you will:

* Create two customised index.html files:
  + Create two index html files, index-aws.html and index-azure.html as a copy of the default nginx welcome html page.
  + index-aws.html: Change the content from "Welcome to Nginx" to "Welcome to AWS".
  + index-azure.html: Change the content from "Welcome to Nginx" to "Welcome to Azure".
* Modify the Ansible playbook to:
  + Copy the respective index-${cloud}.html file to /var/www/html/ on the corresponding machine.
  + Update Nginx configuration to serve the new HTML page.

<https://github.com/amitopenwriteup/ansiblelab/blob/main/lab3.txt>

To verify the changes are applied, access the application via HTTP (IP:80) and confirm if:

* AWS instance displays: "Welcome to AWS".
* Azure instance displays: "Welcome to Azure".

**Task 4: Jenkins Setup for Continuous Deployment**

In this task, you will:

* Install Jenkins on the Tools Machine (AWS).

<https://www.openwriteup.com/?page_id=1096>

Inbound rules, open port 8080

* Set up a Jenkins declarative pipeline to:
  + Pull index-aws.html and index-azure.html from GitHub
  + Deploy the files to their respective locations (/var/www/html/)
  + Restart the Nginx service

<https://www.openwriteup.com/?page_id=1152>

To verify if the Jenkins pipeline runs correctly, modify web content in GitHub, trigger the Jenkins job, and check if the updates reflect correctly on AWS and Azure instances.

**Task 5: Traffic Management Using AWS Route 53**

In this task, you will:

* Create a hosted zone upgrad.com in AWS Route 53.
* Add a DNS entry for app.upgrad.com pointing to the AWS App Machine's public IP.
* Implement a failover policy:
  + Route 53 directs traffic to the AWS App Machine if the AWS instance is healthy.
  + If the AWS instance fails, Route 53 redirects traffic to the Azure VM's public IP.

To verify if you have done this task correctly, simulate a failure by stopping the AWS instance and confirm that traffic is redirected to Azure. Restore the AWS instance and confirm traffic returns to it.