**Assignment:**

You are tasked with building an infrastructure for a web application using AWS with Terraform. The infrastructure needs to be highly available across multiple Availability Zones and consist of the following components:  
1. A VPC with subnets across 2 availability zones.  
2. Two EC2 instances (web servers) in an Auto Scaling Group with a load balancer distributing traffic between them.  
3. Security groups to allow HTTP traffic from the internet to the load balancer and SSH traffic to the web servers only from a specific CIDR block.  
4. Use Terraform modules to encapsulate reusable components for VPC, Auto Scaling Group, and Security Groups.  
5. The EC2 instances should have a provisioner to install NGINX on each instance after it's launched.  
6. Use a remote backend (such as S3) to store the Terraform state securely.  
7. Fetch an existing AMI for the EC2 instances dynamically using a data source.  
8. Ensure the instance type of the EC2 instances can be dynamically selected based on an environment variable (like dev, prod).  
9. Set up an output that will return the DNS name of the load balancer.

Requirements:  
1. Create Terraform Modules:  
VPC Module: Should create a VPC with subnets in 2 availability zones.  
Auto Scaling Module: Should create the auto-scaling group for EC2 instances.  
Security Group Module: Should create security groups for the load balancer and EC2 instances.  
2. EC2 Instances Provisioning:  
Use a Terraform provisioner to install NGINX on the EC2 instances.  
Use the AMI ID fetched dynamically using a data source.  
3. Remote State:  
Store the state in an S3 bucket and enable state locking with DynamoDB.  
4. Conditionally Set the EC2 Instance Type:  
Use a variable for environment (dev or prod). In the dev environment, use t2.micro instances; in the prod environment, use t3.medium instances.  
5. Outputs:  
Output the DNS of the Load Balancer.

---------------------------------------------------------------------------------------------------------------------------------------

**Folder structure:**

terraform/

**|\_** modules/

**|\_**vpc/

**|\_** ec2/

**|\_** security\_groups/

**|\_**envs/

|\_ dev/

**|\_** main.tf

**|\_** variables.tf

|\_ backend.tf

|\_ outputs.tf

|\_ prod

|\_ variables.tf

|\_ outputs.tf

**Configure AWS CLI in local**

* Install AWS CLI & verify

$ aws --version

* Create user

IAM > Users > Create user

Username – admin > Check – Provide access to AWS Console > Choose – I want to create IAM user

* Generate access key

IAM > User > username (admin) > Create access key > Select – Command Line Interface > Next

* Create keypair

EC2 > Key pairs > Create key pair

Name – terraform-key > Key pair type – RSA > Private key file format – .pem

* Configure AWS

$ aws configure

AWS Access Key ID [\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*AGRT]: <access-key>

AWS Secret Access Key [\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*2s5C]: <secret-access-key>

Default region name [d]:

Default output format [d]:

Verify

$ aws configure list

$ aws sts get-caller-identity

---------------------------------------------------------------------------------------------------------------------------------------

**Git Setup**

* Install Git Bash
* Create project folder & navigate into it

$ mkdir <project-name>

$ cd <project-name>

* Create .gitignore file

\*\*/.terraform/

\*\*/\*.tfstate

\*\*/\*.tfstate.backup

\*\*/.terraform.lock.hcl

\*\*/\*.pem

\*\*/\*.tfvars

* Create GitHub repository

GitHub > Repositories > New > Repository name – <repo-name> > Public/Private > Create Repository

* Push local project to GitHub repostory

$ git init # initialize git repository

$ git add . # staging all files

$ git commint -m "Initial commit" # commit changes

$ git branch -M main # rename current branch master to main

$ git remote add origin <repo-link> # add remote repository

$ git push -u origin main # push code to github

**Create Terraform EC2 instance through Terraform**

* Create key-pair

EC2 > Network & Security > Key Pairs > Create key pairs

Name – terraform-key

Key pair type – RSA  
Private key file format – .pem

* setup/main.tf

resource "aws\_security\_group" "terraform\_sg" {

name = "terraform-ssh-access"

description = "Allow SSH access"

ingress {

description = "SSH from local"

from\_port = 22

to\_port = 22

protocol = "tcp"

cidr\_blocks = ["103.244.155.136/32"] # Access from MyIP

}

egress {

from\_port = 0

to\_port = 0

protocol = "-1" # Allowed all protocol

cidr\_blocks = ["0.0.0.0/0"]

}

tags = {

Name = "Terraform-SG"

}

}

resource "aws\_instance" "terraform\_ec2" {

ami = var.ami\_id

instance\_type = var.instance\_type

key\_name = var.terraform\_key

user\_data = file("${path.module}/userdata.sh")

vpc\_security\_group\_ids = [aws\_security\_group.terraform\_sg.id]

tags = {

Name = "Terraform-Server"

}

}

* setup/variable.tf

variable "instance\_type" {}

variable "ami\_id" {}

variable "terraform\_key" {}

* setup/terraform.tfvars

ami\_id = "ami-0ec18f6103c5e0491" # Red Hat Enterprise Linux 10

instance\_type = "t2.micro"

terraform\_key = "terraform-key"

* setup/userdata.sh

#!/bin/bash

sudo yum update -y

sudo yum install -y yum-utils

sudo yum-config-manager --add-repo https://rpm.releases.hashicorp.com/AmazonLinux/hashicorp.repo

sudo yum -y install terraform

* Run below commands

$ terraform init # initialize terraform working directory

$ terraform validate # check syntax

$ terraform fmt # format configuration files

$ terraform plan # show execution plan

$ terraform apply # apply the changes

* Verify

$ ssh -i "terraform-key.pem" [ec2-user@<ec2-public-ip>](mailto:ec2-user@3.85.171.166)

---------------------------------------------------------------------------------------------------------------------------------------

**Configure Terraform instance for AWS CLI/Terraform**

* Install AWS CLI

$ curl "https://s3.amazonaws.com/aws-cli/awscli-bundle.zip" -o "awscli-bundle.zip"

$ sudo yum install zip

$ unzip awscli-bundle.zip

$ sudo ./awscli-bundle/install -i /usr/local/aws -b /usr/local/bin/aws

Verify

$ aws --version

* Configure AWS

$ aws configure

AWS Access Key ID [\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*AGRT]: <access-key>

AWS Secret Access Key [\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*2s5C]: <secret-access-key>

Default region name [d]:

Default output format [d]:

Verify

$ aws sts get-caller-identity

* Pull code from github repo into terraform server

$ git clone <https://github.com/shivam-th/infrastructure_as_code_using_terraform.git>

$ cd infrastructure\_as\_code\_using\_terraform/terraform/envs/dev

* Initialize or Verify Terraform

$ terraform init

$ terraform plan

**Remote Backend Setup**

* Create S3 bucket

$ aws s3api create-bucket \

  --bucket my-terraform-state-bucket \

  --region us-east-1 \

  --create-bucket-configuration LocationConstraint=us-east-1

$ aws s3api put-bucket

-versioning \

  --bucket my-terraform-state-bucket \

  --versioning-configuration Status=Enabled

* Create Dynamo Table

$ aws dynamodb create-table \

--table-name terraform-lock \

--attribute-definitions AttributeName=LockID,AttributeType=S \

--key-schema AttributeName=LockID,KeyType=HASH \

--provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5

---------------------------------------------------------------------------------------------------------------------------------------

* Apply changes

$ terraform plan

$ terraform apply --auto-approve

**If Changes did in local repo and wanted to pull in remote**

* Local

$ git add <file-name>

$ git commit -m "<commit-message>"

$ git push

Or

$ git add .

$ git commit -m "<commit-message>"

$ git push --force

* Remote

$ git pull

---------------------------------------------------------------------------------------------------------------------------------------