**Task Completion Summary**

**1. Basic Infrastructure Setup**

**Steps:**

1. Define configurations for:
   * **Web VM**: Angular 18.5 + Node.js 22
   * **App VM**: .NET Framework 4.8 + .NET Core 8 + JDK 23
   * **DB VM**: SQL Server 2022 + Oracle 19c (Community editions) on **Windows 2022** and **RHEL 9**.

**Terraform Code:**

provider "aws" {

region = "us-east-1"

}

variable "web\_instance\_type" {

default = "t2.medium"

}

variable "app\_instance\_type" {

default = "t3.large"

}

variable "db\_instance\_type" {

default = "m5.large"

}

resource "aws\_instance" "web\_vm" {

ami = "ami-xxxxxxxx" # Replace with RHEL 9 or Windows 2022 AMI

instance\_type = var.web\_instance\_type

tags = {

Name = "WebServer"

Role = "Web"

}

user\_data = <<-EOF

#!/bin/bash

yum install -y nodejs

npm install -g @angular/cli@18.5

EOF

}

resource "aws\_instance" "app\_vm" {

ami = "ami-xxxxxxxx"

instance\_type = var.app\_instance\_type

tags = {

Name = "AppServer"

Role = "App"

}

user\_data = <<-EOF

#!/bin/bash

yum install -y dotnet-sdk-8

yum install -y java-23

EOF

}

resource "aws\_instance" "db\_vm" {

ami = "ami-xxxxxxxx"

instance\_type = var.db\_instance\_type

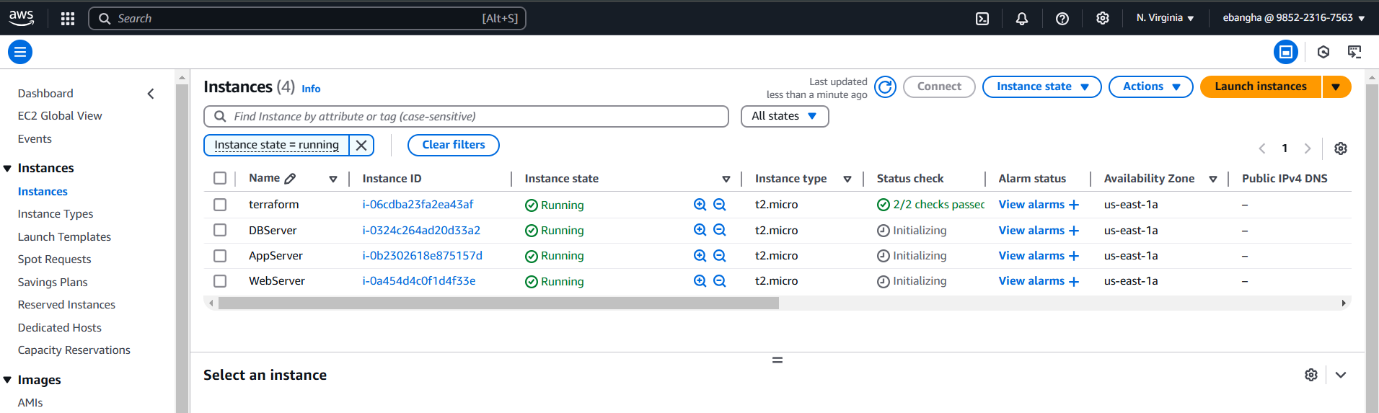
tags = {

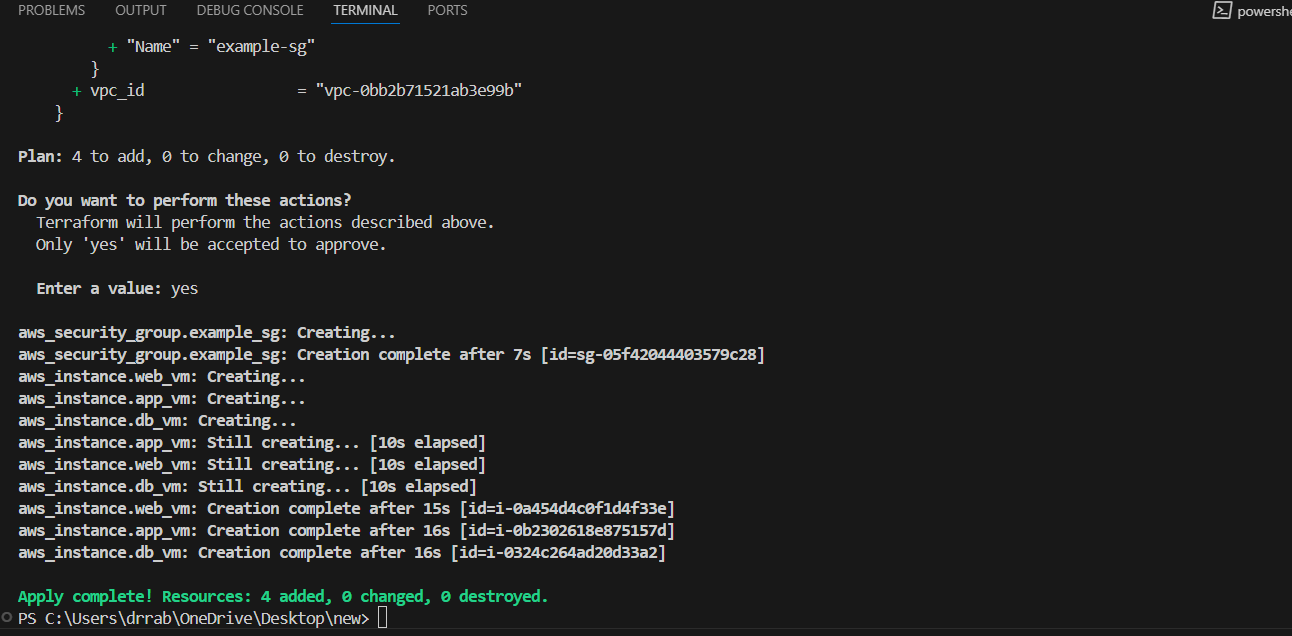
Name = "DBServer"

Role = "Database"

}

}



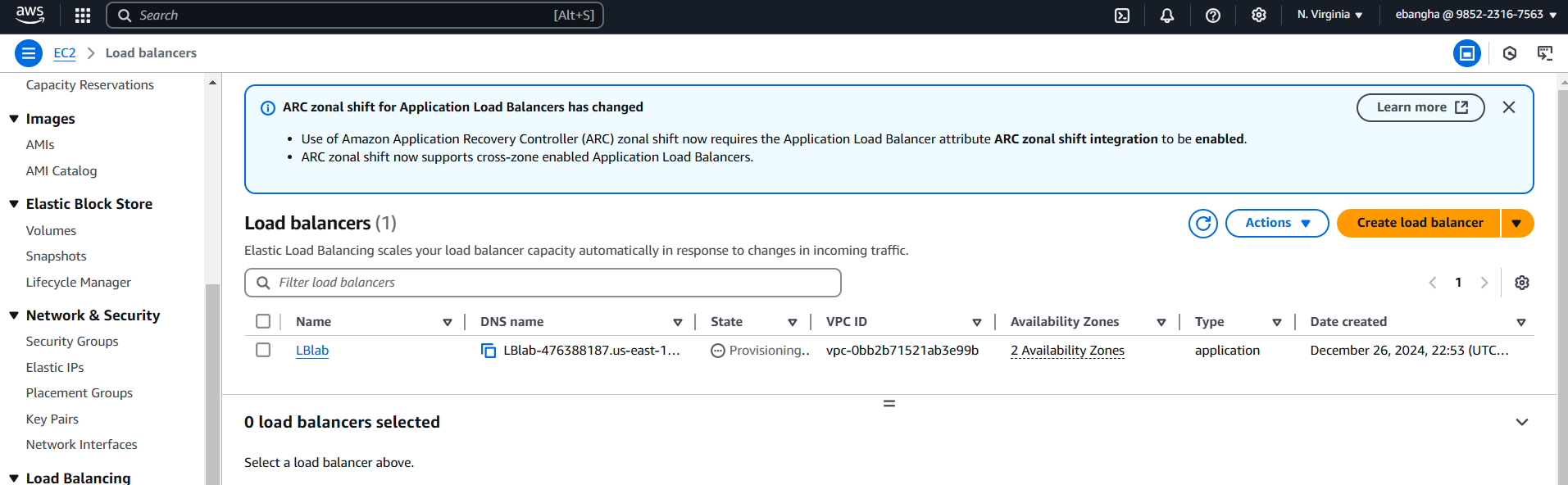


**2. Scaling**

**Vertical Scaling**

* Monitor **CPU**, **Memory**, and **IOPS** using **Amazon CloudWatch** metrics.
* Use **Auto Scaling Groups (ASGs)** with minimum and maximum instance types for vertical scaling.

**Horizontal Scaling**

* **Application Load Balancers (ALB)** with dynamic instance counts based on scaling policies.
* 

**Horizontal Scaling Terraform Example:**

variable "instance\_count" {

default = 2

}

resource "aws\_instance" "web\_vm" {

count = var.instance\_count

ami = "ami-xxxxxxxx"

instance\_type = "t2.medium"

tags = {

Name = "WebServer-${count.index}"

}

}

resource "aws\_lb" "web\_lb" {

name = "web-load-balancer"

internal = false

load\_balancer\_type = "application"

security\_groups = [aws\_security\_group.web\_lb\_sg.id]

dynamic "listener" {

for\_each = [80, 443]

content {

port = listener.value

protocol = "HTTP"

}

}

target\_group {

name = "web-target-group"

port = 80

protocol = "HTTP"

target\_type = "instance"

}

}

**3. Serverless and Containerized Solutions**

**a. Serverless with Lambda**

* Deploy functions using AWS Lambda for serverless operations.
* **API Gateway** to expose these functions as REST APIs.

**b. Dockerized Services**

* Build Docker images for the web, app, and DB layers.
* Deploy them using **AWS Elastic Container Service (ECS)** or **AWS Fargate**.

**c. Kubernetes with EKS**

* Deploy a Kubernetes cluster using **AWS EKS**.
* **Ingress Controller** for API Gateway integration.

**4. CI/CD Pipeline**

**Steps:**

1. Jenkins or GitHub Actions to implement the pipeline.
2. Integrate tools like:
   * **SonarQube** for code quality.
   * **NUnit** and **JUnit** for testing.
3. Use Terraform to automate CI/CD pipelines:

resource "aws\_codepipeline" "my\_pipeline" {

name = "ci-cd-pipeline"

role\_arn = aws\_iam\_role.codepipeline\_role.arn

stage {

name = "Source"

action {

name = "Source"

action\_type\_id = {...}

configuration = {...}

output\_artifacts = ["source\_code"]

}

}

}

**5. Security Configurations**

**IAM Roles**

Assign least-privilege IAM roles to all services.

**NSGs and Routing**

**Network Security Groups (NSGs)** and routing tables for secure traffic flow.

**Example Terraform:**

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

name\_prefix = "db-sg"

ingress {

from\_port = 1433

to\_port = 1433

protocol = "tcp"

cidr\_blocks = ["10.0.0.0/16"]

}

egress {

from\_port = 0

to\_port = 0

protocol = "-1"

cidr\_blocks = ["0.0.0.0/0"]

}

}

**6. Cost Management**

**AWS Budgets Example:**

resource "aws\_budgets\_budget" "monthly\_budget" {

name = "MonthlyBudget"

budget\_type = "COST"

limit\_amount = "50"

limit\_unit = "USD"

time\_unit = "MONTHLY"

}

**7. Validation and Monitoring**

* Used **CloudWatch Logs** for detailed monitoring.
* Python scripts to validate infrastructure:

import boto3

ec2 = boto3.client('ec2')

instances = ec2.describe\_instances()

for reservation in instances['Reservations']:

for instance in reservation['Instances']:

print(f"Instance {instance['InstanceId']} is in state {instance['State']['Name']}.")

**9. Patching and Runtime Management**

**Using Ansible:**

yaml

Copy code

- name: Patch servers

hosts: all

tasks:

- name: Apply updates

yum:

name: '\*'

state: latest