Zocket Assignment

Name: Antony Jebinraj Y Role: DevOps Engineer Email: jebin1848@gmail.com

Steps:

- 1. Write tf code (terraform) for EC2 creation.
- 2. Write Shell Script for Jenkins and Kubectl installation.
- 3. Write tf code for EKS cluster and node creation.
- 4. Create the Application using Mern Stack.
- 5. Design Kubernetes deployment and service.
- Configure application Horizontal auto-scaling.
- 7. Setup Prometheus and Grafana.
- 8. Push the code on GitHub
- 9. Create a Jenkins pipeline auto-trigger EKS cluster tf code and auto-deployment.

Step: 1.Write tf code (terraform) for EC2 creation.

The code block sets the AWS provider for Terraform to use the "us-west-2" region.

```
#provider.tf
provider "aws" {
   region = "us-west-2"
}
```

This Terraform configuration sets up remote state storage in an S3 bucket named "terraform-jenkins-eks-01" within the "us-west-2" region, using the key "jenkins-server/terraform.tfstate".

```
backend.tf
terraform {
  backend "s3" {
    bucket = "terraform-jenkins-eks-01"
    key = "jenkins-server/terraform.tfstate"
    region = "us-west-2"
  }
}
```

Defines variables for a Terraform configuration including VPC and subnet CIDR blocks, availability zone, environment prefix, and EC2 instance type.

```
#terraform.tfvars

vpc_cidr_block = "10.0.0.0/16"

subnet_cidr_block = "10.0.10.0/24"

availability_zone = "us-west-2a"

env_prefix = "dev"

instance_type = "t2.small"
```

Your provided files define variables for a Terraform configuration:

```
#variable "vpc_cidr_block" {
   type = string
   description = "To set cidr for vpc"
}
variable "subnet_cidr_block" {
   type = string
   description = "To set cidr for subnet"
}
variable "availability_zone" {
   type = string
   description = "To set AWS availability region"
}
variable "env_prefix" {
   type = string
   description = "Set as dev or prod or qa etc. based on desired
   environment"
}
variable "instance_type" {
   type = string
   description = "To desired instance type for AWS EC2 instance"
}
```

This Terraform script sets up a VPC with a subnet, an internet gateway, a default route table, and a default security group, all tagged and configured for a Jenkins server environment.

```
resource "aws vpc" "myjenkins-server-vpc" {
 cidr block = var.vpc cidr block
 tags = {
   Name = "${var.env prefix}-vpc"
resource "aws subnet" "myjenkins-server-subnet-1" {
 vpc id
                   = aws vpc.myjenkins-server-vpc.id
              = var.subnet cidr block
 availability zone = var.availability zone
 tags = {
   Name = "${var.env prefix}-subnet-1"
resource "aws_internet gateway" "myjenkins-server-igw" {
 vpc id = aws vpc.myjenkins-server-vpc.id
 tags = {
   Name = "${var.env prefix}-igw"
resource "aws default route table" "main-rtbl" {
aws vpc.myjenkins-server-vpc.default route table id
 route {
   gateway id = aws internet gateway.myjenkins-server-igw.id
 tags = {
   Name = "${var.env prefix}-main-rtbl"
resource "aws default security group" "default-sg" {
 vpc id = aws vpc.myjenkins-server-vpc.id
 ingress {
```

```
from_port = 0
  to_port = 0
  protocol = "-1"  # -1 means all protocols
  cidr_blocks = ["0.0.0.0/0"]
}
egress {
  from_port = 0
   to_port = 0
   protocol = "-1"
   cidr_blocks = ["0.0.0.0/0"]
}
tags = {
  Name = "${var.env_prefix}-default-sg"
}
```

This Terraform script retrieves the latest Amazon Linux 2 AMI and creates an EC2 instance with specified parameters, setting up Jenkins, and outputs the instance's public IP.

Step 2: Write Shell Script for Jenkins and Kubectl installation.

This script updates the system, installs Jenkins, Git, Terraform, and kubectl, and sets up Jenkins to run on system startup on an Amazon Linux system.

```
#jenkins-server-setup.sh
#!/bin/bash
sudo yum update
sudo wget -0 /etc/yum.repos.d/jenkins.repo \
sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key
sudo yum upgrade -y
sudo amazon-linux-extras install java-openjdk11 -y
sudo yum install jenkins -y
sudo systemctl enable jenkins
sudo systemctl start jenkins
sudo yum install git -y
#then install terraform
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
```

```
#finally install kubectl
sudo curl -LO
https://storage.googleapis.com/kubernetes-release/release/v1.23.6/bin/linu
x/amd64/kubectl
sudo chmod +x ./kubectl
sudo mkdir -p $HOME/bin && sudo cp ./kubectl $HOME/bin/kubectl && export
PATH=$PATH:$HOME/bin
```

Executing these terraform tf files for EC2 creation and Jenkins and Kubectl installation.

(terraform init) this command is for initializing our project.

```
C:\ANTONY_JEBINRAJ\Info\eks-terraform-jenkins\terraform-for-jenkins>terraform init

Initializing the backend...

Successfully configured the backend "s3"! Terraform will automatically use this backend unless the backend configuration changes.

Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.53.0...
```

(terraform apply) this command is for applying our infra for provisioning.

```
C:\ANTONY_JEBINRAJ\Info\eks-terraform-jenkins\terraform-for-jenkins>terraform apply
data.aws_ami.latest-amazon-linux-image: Reading...
aws_vpc.myjenkins-server-vpc: Refreshing state... [id=vpc-0ebb7cd98d6f117fc]
data.aws_ami.latest-amazon-linux-image: Read complete after 2s [id=ami-02e8e2a390064c712]
aws_internet_gateway.myjenkins-server-igw: Refreshing state... [id=igw-042afaab0b85c8597]
aws_subnet.myjenkins-server-subnet-1: Refreshing state... [id=subnet-05a2baef070dc077e]
aws_subiliterinty_group.default-sg: Refreshing state... [id=subiliterinty_group.default-sg: Refreshing state... [id=sg-69e56a884254dbe65] aws_default_route_table.main-rtbl: Refreshing state... [id=rtb-0b15500f367ef43bd] aws_instance.myjenkins-server: Refreshing state... [id=i-01dc8cf873283ef8d]
Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
   ~ update in-place
Terraform will perform the following actions:
   # aws_default_security_group.default-sg will be updated in-place
~ resource "aws_default_security_group" "default-sg" {
          ~ egress
                       - cidr_blocks
                                  "0.0.0.0/0",
                           from_port
                        - ipv6_cidr_blocks = []
- prefix_list_ids = []
- protocol = "-!
                          security_groups = []
self = false
                          self
                          to port
                                                       = 0
                           # (1 unchanged attribute hidden)
```

After, Successfully created an EC2 instance and installed Jenkins and Kubectl.

Step 3: Write tf code(terraform) for the EKS cluster and node creation.

The code block sets the AWS provider for Terraform to use the "us-west-2" region.

```
#provider.tf
provider "aws" {
  region = "us-west-2"
}
```

This Terraform configuration sets up remote state storage in an S3 bucket named "terraform-jenkins-eks-01" within the "us-west-2" region, using the key "jenkins-server/terraform.tfstate".

```
backend.tf
terraform {
  backend "s3" {
   bucket = "terraform-jenkins-eks-01"
   key = "jenkins-server/terraform.tfstate"
   region = "us-west-2"
  }
}
```

This Terraform variable definition file specifies the CIDR blocks for a VPC, along with its associated private and public subnets.

```
#terraform.tfvars

vpc_cidr_block = "10.0.0.0/16"

private_subnet_cidr_blocks=["10.0.1.0/24","10.0.2.0/24","10.0.3.0/24"]

public_subnet_cidr_blocks=["10.0.4.0/24","10.0.5.0/24","10.0.6.0/24"]
```

This Terraform configuration declares variables for VPC CIDR block, lists of private subnet CIDR blocks, and lists of public subnet CIDR blocks.

```
#variable.tf
variable "vpc_cidr_block" {
    type = string
}
variable "private_subnet_cidr_blocks" {
```

```
type = list(string)
}
variable "public_subnet_cidr_blocks" {
    type = list(string)
}
```

This Terraform module creates a VPC with public and private subnets, enables NAT gateway, DNS hostnames, and assigns Kubernetes tags for Jenkins server deployment on AWS.

```
#vpc.tf
data "aws availability zones" "azs" {}
module "myjenkins-server-vpc" {
                 = "terraform-aws-modules/vpc/aws"
  source
 name
                 = "myjenkins-server-vpc"
                  = var.vpc cidr block
 cidr
 private_subnets = var.private_subnet_cidr_blocks
 public subnets = var.public subnet cidr blocks
                  = data.aws availability zones.azs.names
  azs
 enable nat gateway
                      = true
  single_nat_gateway = true
  enable dns hostnames = true
  tags = {
    "kubernetes.io/cluster/myjenkins-server-eks-cluster" = "shared"
 public subnet tags = {
    "kubernetes.io/cluster/myjenkins-server-eks-cluster" = "shared"
    "kubernetes.io/role/elb"
                                              = 1
 private_subnet_tags = {
    "kubernetes.io/cluster/myjenkins-server-eks-cluster" = "shared"
    "kubernetes.io/role/internal-elb"
                                              = 1
  }
```

This Terraform module configures an Amazon EKS cluster named "myjenkins-server-eks-cluster" with version "1.24", accessible publicly, linked to a VPC and subnets, and managed node group with specified instance types for a development environment.

```
#eks-cluster.tf
module "eks" {
   source = "terraform-aws-modules/eks/aws"
   version = "~> 19.0"
   cluster name = "myjenkins-server-eks-cluster"
   cluster version = "1.24"
   cluster endpoint public access = true
   vpc id = module.myjenkins-server-vpc.vpc id
   subnet ids = module.myjenkins-server-vpc.private subnets
    tags = {
       environment = "development"
       application = "myjenkins-server"
    }
   eks managed node groups = {
       dev = {
           min size = 1
           max size = 3
           desired size = 2
           instance types = ["t2.small"]
       }
```

Step 4: Create the Application using Mern Stack.

Create the project and the working directory called node-react-k8s You can choose any name you want for your project. In the node-react-k8s folder, create a new folder named, and change the directory into the folder using the following command

mkdir backend and cd backend

The backend folder will contain all the code needed to create the Express server. Within the newly created folder, run the following command to initialize the Node.js project.

npm init --y

This command will initialize the Node.js project and create the 'package.json' to install the project dependencies. Next, we install the dependencies necessary to build and run the application. We will install the dependencies Express, and nodemon to help us create the backend server, monitor the application script, and detect changes, and errors.

npm i express and npm i -g nodemon

Thirdly, open the 'package.json' file, and add the following script under the "script" section.

"dev": "nodemon -L app.js"

4. Within the backend folder, create a file and name it app.js. Add the following code.

```
const express = require('express')

const cors = require('cors')
```

```
const app = express()
app.use(cors())
app.get('/', (req, res) => {
  res.json([
```

```
app.listen(4000, () => {
 console.log('listening for requests on port 4000')
```

With the code above, you will create an Express application using the 'get' route that will set

APIs to the React front end. This will be listening for all the requests coming from port 4000 on
the server. To run the application, use the following command on your server.

npm run dev

This will create the development server to run your application. Creating React.js Application.

After building your backend server, change the directory out of the backend folder, and create a React.js project using the following command

npx create-react-app frontend

We'd use this command to create a new folder named "frontend" in the working directory. Within the 'frontend' folder, navigate to the "src" folder, and open "App.js".

```
import { useEffect, useState } from 'react'
import './App.css';

function App() {
  const [blogs, setBlogs] = useState([])

  useEffect(() => {
    fetch('http://localhost:4000/')

    .then(res => res.json())
```

```
.then(data => setBlogs(data))
<header className="App-header">
 <h1>all blogs</h1>
 {blogs && blogs.map(blog => (
   <div key={blog.id}>{blog.title}</div>
```

```
export default App;
```

Taking a look at the 'fetch' section, you can see that it is fetching the API from the backend server to display on the front end. Don't worry if you do not understand. run your application using the following code.

npm start

Step 5: Design Kubernetes deployment and service.

Dockerfile. This Dockerfile will have all the commands that you will need to build the backend of the application. The code will look like the one below.

```
#Dockerfile

FROM node:18-alpine

RUN npm install -g nodemon

WORKDIR /app
```

```
COPY package.json .

RUN npm install

COPY . .

EXPOSE 4000

CMD ["npm", "run", "dev"]
```

This file defines a Kubernetes Deployment and Service configuration for a Node.js application, specifying resource limits, ports, and replicas.

```
#deployment-service.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

   name: node-js-deployment

spec:

   replicas: 2
```

```
selector:
```

```
cpu: "500m"
apiVersion: v1
kind: Service
metadata:
spec:
```

```
ports:
- port: 4000
targetPort: 4000
protocol: TCP
```

Just as we have dockerized the Node.js application, we are required to Dockerize the React.js application too.Navigate to the 'front end' and create a Dockerfile in the folder. Open the "Dockerfile" and add the following code.

```
FROM node:18-alpine

WORKDIR /app

COPY package.json .

RUN npm install

COPY . .

EXPOSE 3000
```

```
CMD ["npm", "start"]
```

This Kubernetes manifest defines a Deployment and a Service for a React.js application, with two replicas, resource limits, exposing port 3000.

```
#deployment-service.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
spec:
```

```
metadata:
```

```
apiVersion: v1
kind: Service
metadata:
spec:
```

Step 6: Configure application Horizontal auto-scaling.

Install metric server

```
[root@ip-10-0-10-168 ~] # kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/download/v0.3.6/components.yaml clusterrole.rbac.authorization.k8s.io/system:aggregated-metrics-reader created clusterrolebinding.rbac.authorization.k8s.io/metrics-server:system:auth-delegator created rolebinding.rbac.authorization.k8s.io/metrics-server-auth-reader created serviceaccount/metrics-server created deployment.apps/metrics-server created service/metrics-server created clusterrole.rbac.authorization.k8s.io/system:metrics-server created clusterrolebinding.rbac.authorization.k8s.io/system:metrics-server created clusterrolebinding.rbac.authorization.k8s.io/system:metrics-server created error: unable to recognize "https://github.com/kubernetes-sigs/metrics-server/releases/download/v0.3.6/components.yaml": no matches for kind "APIService" in version "apiregistration.k8s.io/vlbetal"
```

Create an Horizontal Pod Auto scaler yaml file

```
#hpa.yaml
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
   name: myapp-backend-hpa
spec:
   scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: node-js-deployment
minReplicas: 2
maxReplicas: 10
metrics:
   - type: Resource
   resource:
    name: cpu
   target:
    type: Utilization
    averageUtilization: 50
```

Step 7: Setup Prometheus and Grafana.

Install Prometheus:

Create a Prometheus YAML file for deployment.

```
apiVersion: apps/v1
kind: Deployment
```

```
metadata:
spec:
```

```
- name: prometheus-config
apiVersion: v1
kind: Service
metadata:
spec:
```

Create a Prometheus Configuration:

Create a ConfigMap for Prometheus configuration. prometheus-config.yaml:

```
apiVersion: v1
kind: ConfigMap
```

```
data:
```

```
bearer_token_file:
/var/run/secrets/kubernetes.io/serviceaccount/token

kubernetes_sd_configs:
    - role: node

- job_name: 'kubernetes-pods'

kubernetes_sd_configs:
    - role: pod
```

Install Grafana:

Create a Grafana YAML file for deployment.

```
apiVersion: apps/v1
kind: Deployment
metadata:
spec:
```

```
apiVersion: v1
kind: Service
metadata:
spec:
```

```
app: grafana

ports:
   - port: 3000
   targetPort: 3000
   nodePort: 32000
```

Step 8: Push the code on GitHub

• Create Repo (eks-terraform-jenkins) and using these command for push.

```
git add .
git commit -m ("update")
git push origin main
```

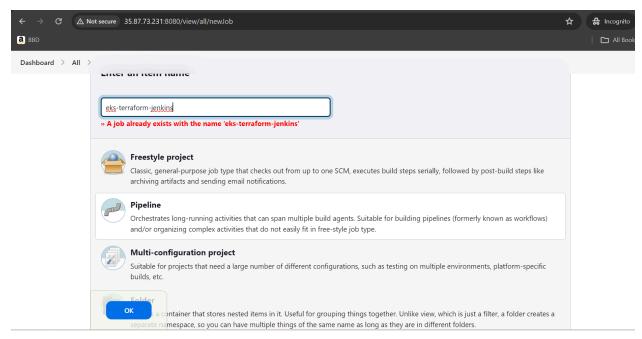
Step 9: Create a Jenkins pipeline auto-trigger EKS cluster tf code and auto-deployment.

This Jenkins pipeline script automates the creation of an EKS cluster, builds Docker images for both backend and frontend applications, pushes them to ECR, and deploys them to Kubernetes.

```
stage("Deploy to Backend") {
      steps {
        script {
          dir('my-app/backend') {
            sh "aws eks update-kubeconfig --name myjenkins-server-eks-cluster --region us-west-2"
            sh "docker build -t myapp-backend ."
            sh "aws ecr get-login-password --region us-west-2 | docker login --username AWS
 password-stdin 038675313786.dkr.ecr.us-west-2.amazonaws.com"
            sh "docker tag myapp-backend:latest
038675313786.dkr.ecr.us-west-2.amazonaws.com/myapp-backend:latest"
            sh "docker push
038675313786.dkr.ecr.us-west-2.amazonaws.com/myapp-backend:latest"
            sh "kubectl apply -f deployment-service.yaml"
          }
    stage("Deploy to Frontend") {
      steps {
        script {
          dir('my-app/frontend') {
            sh "aws eks update-kubeconfig --name myjenkins-server-eks-cluster --region us-west-2"
            sh "docker build -t myapp-frontend ."
            sh "aws ecr get-login-password --region us-west-2 | docker login --username AWS
 password-stdin 038675313786.dkr.ecr.us-west-2.amazonaws.com"
            sh "docker tag myapp-frontend:latest
038675313786.dkr.ecr.us-west-2.amazonaws.com/myapp-frontend:latest"
            sh "docker push
038675313786.dkr.ecr.us-west-2.amazonaws.com/myapp-frontend:latest"
            sh "kubectl apply -f deployment-service.yaml"
```

Go to our Jenkins console using this URL: http://35.87.73.231:3000/ (username: admin: password:admin)

Next, create a pipeline using a console like:



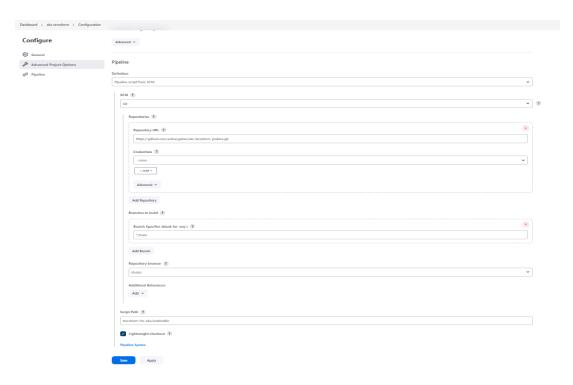
Choose pipeline option.

Select Git.

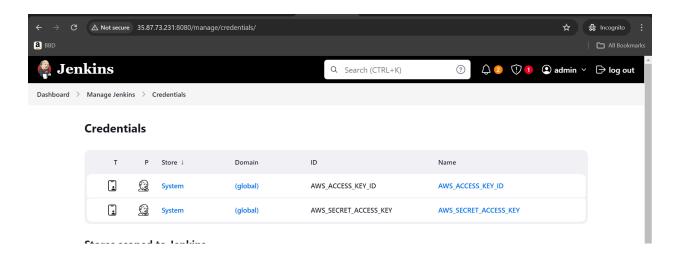
Configure our git URL.

Configure our Jenkinfile with path location.

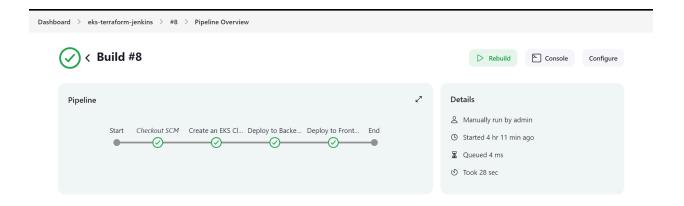
Save and Apply.



Next, Go Dashboard>Manage Jenkins>Credentials.
Create AWS Access key and secret key credentials to access our AWS account.



Next, Click Build button its automatically trigger our jenkins pipeline and create of an EKS cluster, builds Docker images for both backend and frontend applications, pushes them to ECR, and deploys them to Kubernetes.



Jenkins 2.452.1

[root@ip-10-0-10	-168 ~]# k	ubectl get a	11	READY	STATI	JS	
RESTARTS AGE							
pod/grafana-6968b75956-5w588					Runn	ing	0
106s							
pod/node-js-deployment-5c874fc856-			7	1/1 Runn		ing	0
29m							
pod/node-js-deployment-5c874fc856-zfmh8					Running 0		
29m							
pod/prometheus-6c8fc9bcdd-qzjfg					Running		0
29 m							
pod/react-js-deployment-67764899cd-ph6bx				Runn	ing	0	
29m							
pod/react-js-deployment-67764899cd-thbkk					Runn	ing	0
29m							
NAME		TYPE	CLU	STER-IP		EXTERNAL-IP	
PORT(S)	AGE						
service/grafana		NodePort	Port 172.20.63.1		42	<none></none>	
3000:32000/TCP	106s						
service/kubernetes		ClusterIP	172.20.0.1			<none></none>	
443/TCP	6h43m						
service/node-js-	service	NodePort	172	.20.126.	209	<none></none>	
4000:31140/TCP	29m						
service/prometheus		NodePort	172.20.141.		83 <none></none>		
9090:31000/TCP	108s						
service/react-js-service		NodePort	172.20.44.1		47	<none></none>	
3000:32752/TCP	29m						

NAME	READY	UP-T	O-DATE	AVAILABLE	AGE					
deployment.apps/grafana										
106s										
deployment.apps/node-js-deployment	2/2	2		2	29m					
deployment.apps/prometheus										
109s										
deployment.apps/react-js-deploymer	nt 1/2	2			29m					
NAME		D:	ESIRED	CURRENT	READY					
AGE										
replicaset.apps/grafana-6968b75956				1						
106s										
replicaset.apps/node-js-deployment	2		2	2						
29m										
replicaset.apps/prometheus-6c8fc9k				1						
109s										
replicaset.apps/react-js-deploymer	d 2		2	1						
29m										
NAME			REF	FERENCE						
TARGETS MINPODS MAXPODS	REPLICAS	AGE								
horizontalpodautoscaler.autoscaling/myapp-backend-hpa										
Deployment/node-js-deployment <	ınknown>/50%	2		.0	2					

Successfully set up all these things!