**Web Application Deployment on Amazon EKS with Prometheus Monitoring**

**1. Objective**

This documentation explains the process of deploying a web application on Amazon Elastic Kubernetes Service (EKS) using Kubernetes. The application is containerized with Docker, stored in Amazon Elastic Container Registry (ECR), and monitored using Prometheus.

**2. Prerequisites**

**Tools and Utilities:**

* AWS CLI: Installed and configured with IAM permissions for EKS, ECR, and EC2.
* Docker: Installed for building container images.
* Terraform: Installed for infrastructure management.
* kubectl: Installed for interacting with the Kubernetes cluster.
* Helm: Installed for deploying Prometheus.

**Setup:**

* An Amazon EKS cluster and Node Group created using Terraform.
* Access to AWS Elastic Container Registry (ECR).

**3. Steps Followed**

**Step 1: Create the EKS Cluster and Node Groups with Terraform**

**Terraform Script for EKS Cluster:**

The Terraform script provisions:

* A VPC with public and private subnets.
* An EKS cluster with a managed Node Group.
* resource "aws\_eks\_cluster" "eks\_cluster" {
* name = "webapp"  # Your desired cluster name
* role\_arn = aws\_iam\_role.eks\_role.arn
* vpc\_config {
* subnet\_ids = ["subnet-045c88b1bac827cfd", "subnet-09d156411fc809c1d"]
* }
* depends\_on = [aws\_iam\_role.eks\_role, aws\_iam\_role\_policy\_attachment.eks\_policy\_attachment]
* }
* resource "aws\_iam\_role" "eks\_role" {
* name = "eks-role"
* assume\_role\_policy = jsonencode({
* Version = "2012-10-17"
* Statement = [{
* Effect = "Allow"
* Principal = {
* "Service": [
* "eks.amazonaws.com",
* "ec2.amazonaws.com"
* ]
* }
* Action = "sts:AssumeRole"
* }]
* })
* }
* resource "aws\_iam\_role\_policy" "eks\_role\_policy" {
* name = "eks-role-policy"
* role = aws\_iam\_role.eks\_role.name
* policy = jsonencode({
* Version = "2012-10-17"
* Statement = [{
* Action = "\*"
* Effect = "Allow"
* Resource = "\*"
* }]
* })
* }
* resource "aws\_iam\_role\_policy\_attachment" "eks\_policy\_attachment" {
* policy\_arn = "arn:aws:iam::aws:policy/AmazonEKSClusterPolicy"
* role       = aws\_iam\_role.eks\_role.name
* }
* resource "aws\_eks\_node\_group" "eks\_node\_group" {
* cluster\_name    = aws\_eks\_cluster.eks\_cluster.name
* node\_group\_name = "webapp-node-group"  # node group name
* node\_role\_arn   = aws\_iam\_role.eks\_role.arn
* subnet\_ids      = ["subnet-045c88b1bac827cfd", "subnet-09d156411fc809c1d"]  # subnets
* scaling\_config {
* desired\_size = 2
* max\_size     = 3
* min\_size     = 1
* }
* instance\_types = ["t3.medium"]
* }

**Deploy the Infrastructure:**

* terraform init
* terraform plan
* terraform apply

**Configure the Kubernetes Context:**

After successful deployment, configure the Kubernetes context:

* aws eks update-kubeconfig --region us-east-1 --name webapp-cluster

**Step 2: Build and Push the Docker Image to ECR**

**Dockerfile for the Web Application:**

The web application serves a static HTML page (index.html).

# Use a lightweight web server like Nginx

FROM nginx:alpine

# Copy static content into the web server

COPY ./index.html /usr/share/nginx/html/

# Expose port 80

EXPOSE 80

# Run Nginx in the foreground

CMD ["nginx", "-g", "daemon off;"]

**Build the Docker Image:**

* docker build -t webapp:latest .

**Push the Image to ECR:**

* Create an ECR repository:
* aws ecr create-repository --repository-name webapp
* Authenticate Docker with ECR:
* aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin <account-id>.dkr.ecr.us-east-1.amazonaws.com
* Tag and push the image:
* docker tag webapp:latest <account-id>.dkr.ecr.us-east-1.amazonaws.com/webapp:latest
* docker push <account-id>.dkr.ecr.us-east-1.amazonaws.com/webapp:latest

**Step 3: Kubernetes Configuration**

**Deployment YAML:**

Defines the Deployment with two replicas, pulling the image from ECR.

apiVersion: apps/v1

kind: Deployment

metadata:

name: webapp-deployment

spec:

replicas: 2

selector:

matchLabels:

app: webapp

template:

metadata:

labels:

app: webapp

spec:

containers:

- name: webapp-container

image: 713881811574.dkr.ecr.us-east-1.amazonaws.com/webapp-repo:devops

ports:

- containerPort: 80

**Service YAML:**

Exposes the application through an AWS LoadBalancer.

apiVersion: v1

kind: Service

metadata:

name: webapp-service

spec:

selector:

app: webapp

ports:

- protocol: TCP

port: 80

targetPort: 80

type: LoadBalancer

**Apply the Configuration:**

* kubectl apply -f deployment.yaml
* kubectl apply -f service.yaml

**Get the Web App URL:**

* kubectl get svc

Use the EXTERNAL-IP of the webapp-service to access the application.

**Step 4: Push the Files to a GitHub Repository**

**Initialize Git Repository:**

* git init

**Add Files to the Repository:**

* git add Dockerfile index.html deployment.yaml service.yaml terraform/

**Commit the Files:**

* git commit -m "Initial commit: Added infrastructure, Kubernetes configs, and Docker setup"

**Create a New Repository on GitHub:**

Create a repository on GitHub (e.g., webapp-deployment) and copy the repository URL.

**Push to GitHub:**

* git remote add origin <repository-url>
* git branch -M main
* git push -u origin main

**Step 5: Add Prometheus Monitoring**

**Deploy Prometheus with Helm:**

helm install prometheus prometheus-community/kube-prometheus-stack -n monitoring --create-namespace

**Configure the Web App for Prometheus:**

Add annotations to the webapp-deployment to enable Prometheus scraping.

metadata:

annotations:

prometheus.io/scrape: "true"

prometheus.io/port: "80"

**Port-Forward Prometheus for Access:**

* kubectl port-forward -n monitoring svc/prometheus-kube-prometheus-prometheus 9090:9090
* Access Prometheus at http://localhost:9090.

**Verify Metrics:**

Open Prometheus and check metrics related to the web application using the http\_requests\_total metric.

**4. Access URL**

* **Web Application URL:**
* Example: http://ab3d6941ae66d45aebed0d4a6ecb5a53-371752692.us-east-1.elb.amazonaws.com
* **Prometheus Monitoring:**
  + Accessed via http://localhost:9090 after port-forwarding.

**5. Conclusion**

* Successfully deployed the web application on Amazon EKS with Kubernetes.
* Dockerized the web application and stored it in Amazon ECR.
* Kubernetes managed deployment and exposed the app through a LoadBalancer.
* Prometheus was configured for monitoring the application’s performance and metrics.
* Project files and configurations were pushed to GitHub for version control.