DevOps / Site Reliability Engineer - Technical Assessment Documentation

Introduction

This document outlines the solution for the CI/CD pipeline that deploys a NodeJS web application to a load-balanced environment on AWS using EKS. The infrastructure is managed using Terraform, with CI handled by GitHub Actions and CD handled by ArgoCD.

Infrastructure Setup

Overview

The target environment consists of an AWS ALB load balancer accessible via HTTP on port 80, which uses a round-robin strategy to distribute traffic.

Tools and Services

- **Infrastructure as Code**: Terraform managed by tfscaffold.
- **CI/CD System**: GitHub Actions (CI) and ArgoCD (CD).
- **AWS Services**: VPC, Subnets, Security Groups, IAM Roles, Application Load Balancer (ALB), EKS.

Repository Details

- **Application Repository**: exa-devops-assessment
- **Kubernetes Manifest Repository**: sample-app

Setup Details

Infrastructure Setup Details

The infrastructure setup is automated and secured using the `tfscaffold` Terraform framework, which manages the backend and state files efficiently. To match the prerequisites of `tfscaffold`, a custom Docker image `georgeulahannan/terraform:07` is used, reducing build time significantly.

In `tfscaffold`, each infrastructure component is defined as a separate module (e.g., VPC, EKS, EKS ALB, EKS ArgoCD, ArgoCD App, Autoscaler). Each component can be deployed to different environments using the following command:

```
./bin/terraform.sh -p emis -c vpc -e dev -r us-east-1 -a apply/plan
```

Application Setup Details

Application files are contained in the `app` folder. A Dockerfile is included to build the Docker image, and a health check URL is added to verify that the application is running correctly when deployed in Kubernetes.

CI/CD Pipeline Configuration

Overview

The CI/CD pipeline is implemented using GitHub Actions and is triggered on push events to feature branches. Two workflows are created to separate application and infrastructure deployment: `application.yml` and `terraform.yml`.

Terraform Workflow

The `terraform.yml` workflow runs jobs in sequence, ensuring each job must succeed before proceeding to the next. This setup guarantees the integrity and availability of the infrastructure.

Terraform Workflow Jobs

1. **Checkout Code**: Checks out the repository code.
2. **Configure AWS Credentials**: Sets up AWS credentials using GitHub Secrets.
3. **Get Branch Name**: Retrieves the current branch name.
4. **Get Build ID**: Retrieves the current build ID.
5. **Terraform VPC Plan**: Runs the Terraform plan command for the VPC module.
6. **Terraform VPC Apply**: Applies the Terraform changes for the VPC module if the plan is successful.
7. **Terraform EKS Plan**: Runs the Terraform plan command for the EKS module if the VPC apply is successful.
8. **Terraform EKS Apply**: Applies the Terraform changes for the EKS module if the plan is successful.
9. **Terraform Cluster Autoscaler Plan**: Runs the Terraform plan command for the Cluster Autoscaler module if the EKS apply is successful.
10. **Terraform Cluster Autoscaler Apply**: Applies the Terraform changes for the Cluster Autoscaler module if the plan is successful.
11. **Terraform EKS ALB Plan**: Runs the Terraform plan command for the EKS ALB module if the Cluster Autoscaler apply is successful.
12. **Terraform EKS ALB Apply**: Applies the Terraform changes for the EKS ALB module if the plan is successful.
13. **Terraform EKS ArgoCD Plan**: Runs the Terraform plan command for the EKS ArgoCD module if the EKS ALB apply is successful.
14. **Terraform EKS ArgoCD Apply**: Applies the Terraform changes for the EKS ArgoCD module if the plan is successful.

15. **Terraform ArgoCD App Plan**: Runs the Terraform plan command for the ArgoCD App module if the EKS ArgoCD apply is successful.
16. **Terraform ArgoCD App Apply**: Applies the Terraform changes for the ArgoCD App module if the plan is successful.

Application Workflow

The `application.yml` workflow handles the Docker build process and updates the Kubernetes manifests.

Application Workflow Jobs

1. **Checkout Code**: Checks out the repository code.
2. **Login to Docker Hub**: Logs into Docker Hub using credentials stored in GitHub Secrets.
3. **Build and Push Docker Image**: Builds the Docker image and pushes it to DockerHub.
4. **Update Kubernetes Manifests**: Updates the image tag in the Kubernetes manifests stored in the `sample-app` repository, so ArgoCD can consume and deploy the updated Kubernetes services.

Kubernetes Configuration

- **Cluster Autoscaling**: Configured to automatically adjust the number of nodes in the EKS cluster based on the workload.
- **Horizontal Pod Autoscaling**: Configured to automatically scale the number of pod replicas based on CPU and memory usage.

Bastion Host Setup (Extra)

A bastion host is deployed manually to securely access EC2 resources, using Packer and Ansible for AWS AMI creation. This setup is particularly useful for deploying resources in private subnets.

Conclusion

This solution demonstrates the implementation of a modern CI/CD pipeline that automates the deployment of a NodeJS web application to AWS EKS. By following DevOps best practices, the infrastructure is provisioned using Terraform, and the deployment process is streamlined using GitHub Actions and ArgoCD. The setup ensures scalability, maintainability, and efficient resource utilization.