End-to-End CI/CD Pipeline for Node.js App Deployment on EKS using GitHub Actions

# Project Description:

This project demonstrates the implementation of a complete CI/CD pipeline for a Node.js application, targeting deployment on Amazon EKS (Elastic Kubernetes Service) using Jenkins. It automates the entire software delivery process—from code commit to deployment—ensuring faster and more reliable delivery of application updates.

# Pre-requisite

Before you proceed, ensure you have the following installed:

* 🛠 Node.js (>=14.x)
* 🐳 Docker (latest version)
* 🏗️ Terraform (>=1.0)
* ☸ kubectl (latest version)
* 🎭 Kustomize
* ☁ AWS CLI & eksctl
* ⚙️ Jenkins
* 🔑 AWS IAM permissions to manage EKS

# Repository structure for Application Code

📂 app # Application source code

│ ├── app.py # Python application logic (if applicable)

│ ├── calculator.js # Business logic for calculations

│ ├── calculator.test.js # Unit tests for calculator functions

│ ├── Dockerfile # Dockerfile for building the Node.js app

│ ├── Dockerfile-python # Dockerfile for a Python-based version (if needed)

│ ├── index.js # Main entry point of the Node.js application

│ └── package.json # Project dependencies and scripts

The **CI/CD pipeline** automates the entire deployment process using **Jenkins**.

# 🔨 Build Job

* Set Up the Environment
* Install Node.js dependencies using npm install.
* Lint the code to ensure quality standards.
* Run Tests
* Execute unit tests with npm test.
* Generate test reports for visibility.
* Version Management
* Uses Semantic Versioning (major.minor.patch).
* Auto-increments the version based on commit messages.
* Build & Push Docker Image
* Builds a Docker image of the application.
* Pushes it to Amazon Elastic Container Registry (ECR).

# 🚀 Deployment Job

* Terraform Setup
* Initializes Terraform with terraform init.
* Ensures correct state management.
* Infrastructure Provisioning
* Executes terraform plan and terraform apply.
* Deploys EKS clusters, networking, and storage.
* Kubernetes Configuration
* Configures kubectl to interact with the cluster.
* Applies Kustomize overlays for environment-specific settings.
* ngress Controller Setup
  + Uses Helm to install NGINX Ingress.
* Application Deployment
  + Deploys the latest Docker image to Kubernetes.
  + Exposes the service via Ingress and Load Balancer.

# 🏗️ Infrastructure Details

|  |  |  |
| --- | --- | --- |
| **Environment** | **Instance Type** | **Replica Count** |
| **Dev** | t3.small | 1 |
| **Staging** | t3.medium | 3 |
| **Prod** | t3.large | 3 |

✅ **DNS Automation via Cloudflare**

* Environment-specific subdomains:
  + dev.example.com
  + staging.example.com
  + prod.example.com

# 📦 Application Deployment Strategy

This project supports multiple deployment strategies:

✅ Rolling Updates – Default strategy, ensuring zero downtime.   
✅ Blue-Green Deployment – Used in production environments.   
✅ Canary Deployments – Gradual rollout for safe updates.

# 🔄 GitOps Principles

✔ Git as the Source of Truth   
✔ Declarative Infrastructure (Terraform & Kubernetes)   
✔ Automated Deployments via GitHub Actions

Every infrastructure change must be made via a Git commit.

# 🔒 Security Best Practices

🔐 Secrets Management

Uses AWS Secrets Manager & GitHub Actions encrypted secrets.

🛡 Container Security

Uses Trivy and Docker Bench Security for vulnerability scanning.

🚧 IAM & Least Privilege

Uses AWS IAM roles with restricted access.

# 📢 Notifications & Alerts

🔔 Slack & Email Notifications

CI/CD Job Updates – Pipeline status alerts.

DNS Updates – Cloudflare integration for alerts.

📡 Monitoring & Logging

AWS CloudWatch for logs & metrics.

Prometheus & Grafana for observability.

# 📊 Monitoring & Logging

✅ Application Logs – Aggregated using Fluent Bit.   
✅ Infrastructure Logs – Stored in AWS CloudWatch Logs.

✅ Metrics Monitoring – Tracked using Prometheus & Grafana.  
  
  
  
