CI-CD_Pipeline_github_actions

Repository: jadalaramani/CI-CD_Pipeline_gha

Link: https://github.com/jadalaramani/CI-CD_Pipeline_gha.git

Table of contents

1. Project overview
2. Repo structure (high level)
3. Architecture & flow
4. Block diagram (Mermaid + diagram prompt)
5. Prerequisites
6. Step-by-step setup (local dev -> CI -> CD)
7. GitHub Actions workflows explained
8. Terraform (infra) overview
9. App (code) overview and local testing
10. Environment variables & secrets
11. Deploy target (EKS / ECR) — detailed steps
12. Observability & notifications (Slack webhook)
13. Troubleshooting checklist
14. Useful commands

1. Project overview

This repository implements a full CI/CD pipeline using **GitHub Actions** to build, test, containerize, push images to **AWS ECR**, and deploy to a Kubernetes cluster (example targets in repo). It also includes Terraform code for provisioning infrastructure.

Goal: provide an automated pipeline that builds the app (Node.js), runs tests, produces a Docker image, pushes it to ECR, then deploys/upgrades the application on Kubernetes using manifests or Helm (as present in this repo).

2. Repo structure (high level)

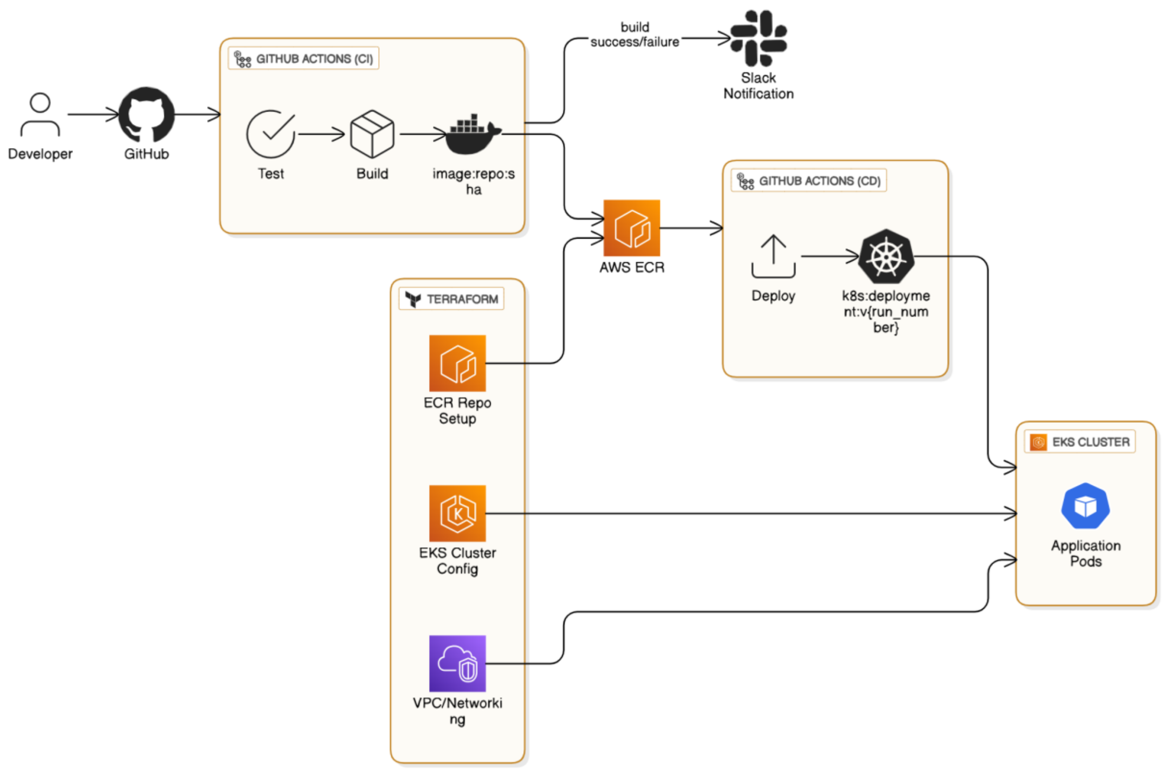
- .github/workflows/ — GitHub Actions workflows (CI/CD). Review these to see exact jobs & triggers.
- app/ — application source code (Node.js). Contains package.json, server, Dockerfile, health check endpoints.
- terraform/ — Terraform configuration for provisioning cloud resources (VPC, ECR, EKS, IAM, etc.).
- README.md — repo README with quick start notes.

Note: open .github/workflows in the repo to see the precise workflow filenames and job details.

3. Architecture & flow (high-level)

1. Developer pushes code to GitHub (branch main or feature branch).
2. GitHub Actions triggers pipeline:
   - Lint & unit tests
   - Build Docker image
   - Authenticate to AWS ECR
   - Push image to ECR with tag (e.g., commit SHA or github.run_number)
   - Deploy to Kubernetes (apply manifests / kubectl / Helm upgrade)
3. Kubernetes cluster receives new image; rolling update occurs.
4. Notifications sent to Slack (optional) and logs/metrics captured by cluster monitoring.

4. Block diagram



CI/CD pipeline for deploying containerized applications on Amazon Elastic Kubernetes Service (EKS) using GitHub Actions, Terraform, and Amazon Elastic Container Registry (ECR).

Developer & Source Control (GitHub)

- The **Developer** writes code and pushes it to the GitHub repository.
- A push or pull request triggers the **CI pipeline** in GitHub Actions.

CI Stage – Build & Test

- **GitHub Actions (CI)** runs automatically on each commit.
- Steps performed:
- **Test**: Run unit/integration tests to validate the code.
- **Build**: Create a Docker image for the application.
- **Push to ECR**: Tag the image with repo:sha and push it to ECR.
- **Slack Notification**: If integrated, sends build status updates (success/failure).

Infrastructure Setup – Terraform

- **Terraform** provisions cloud infrastructure components:
- **ECR Repo Setup**: Creates ECR repository to store images.
- **EKS Cluster Config**: Provisions EKS cluster.
- **VPC/Networking**: Sets up networking and subnets required for the cluster.
- This infrastructure supports both CI and CD pipelines.

AWS ECR (Artifact Repository)

- The built Docker image is stored in AWS ECR.
- The CD pipeline later pulls this image for deployment.

CD Stage – Deployment

- **GitHub Actions (CD)** is triggered after a successful build.
- Steps performed:
- **Deploy**: Pull the image from ECR.
- **Kubernetes Deployment**: Apply deployment manifests with image tags like k8s:deployment:{run_number} to EKS.

EKS Cluster

- The updated application image is deployed into the **EKS cluster**.
- Pods are created/updated in the relevant namespaces.
- Traffic is routed via services and ingress if configured.

Notifications

- Build results and deployment status can be notified through Slack.

End-to-End Flow

- Developer pushes code → GitHub triggers CI.
- CI builds and pushes Docker image → stored in ECR.
- CD pulls image from ECR → deploys on EKS.
- Terraform ensures infra (VPC, ECR, EKS) is provisioned.
- Slack notifies the team of build status.
- Application runs on EKS Pods.

Key Tools Used

- Source Control: GitHub
- CI/CD: GitHub Actions
- Container Registry: AWS ECR
- Infrastructure: Terraform
- Container Orchestration: EKS
- Notification: Slack

5. Prerequisites

- GitHub account with access to repository
- AWS account with permissions for ECR, EKS, IAM, S3 (as required by Terraform)
- AWS CLI configured locally (for manual testing): aws configure
- kubectl configured to talk to the target cluster (for manual deploys)
- Terraform (matching repo version) installed locally
- Node.js & npm for local app testing

6. Step-by-step setup (local dev -> CI -> CD)

A. Local development

1. Clone repo: git clone https://github.com/jadalaramani/CI-CD_Pipeline_gha.git
2. cd app
3. npm install

4. npm start — app will be available on http://localhost:3000; health at /health.

B. Prepare AWS (one-time)

1. Create an ECR repository (can be manual via console or via Terraform). Note repository name used in workflow (look in .github/workflows or terraform/ecr.tf).
2. Ensure IAM user/role for GitHub Actions has permission to push images to ECR and update Kubernetes (if using OIDC or a machine user + kubeconfig secret).
3. If using EKS via Terraform in this repo, run Terraform to provision infra (see Terraform section).

C. GitHub secrets (required for workflows)

Set these secrets in the GitHub repository (Settings → Secrets & variables → Actions):

- AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY (or use OIDC to avoid static creds)
- AWS_REGION (e.g., us-east-1)
- ECR_REPOSITORY (repo URI or name)
- KUBECONFIG or configure kubectl step to assume role and fetch kubeconfig (alternatively store KUBE_CONFIG_DATA as base64)
- SLACK_WEBHOOK_URL (optional)

7. GitHub Actions workflows explained

Look at files under .github/workflows/ — typical jobs you'll find:

1. **CI: build-and-test**
   - Triggers: push, pull_request
   - Runs: actions/checkout, node setup, npm ci, npm test, npm run lint
2. **Build & Push Docker**
   - Uses docker/build-push-action or aws-actions/amazon-ecr-login
   - Builds image, tags with github.sha or github.run_number
   - Pushes to ECR
3. **Deploy**
   - Runs kubectl or helm commands to apply manifests and rollout updates.
   - Might fetch kubeconfig using aws eks update-kubeconfig or use stored base64 KUBECONFIG secret.
4. **Notify**
   - Post a summary to Slack using the webhook after success/failure.

8. Terraform (infra) overview

- Terraform folder contains HCL code to create cloud resources: ECR repo, EKS cluster, IAM roles, VPCs.
- Typical flow:

1. terraform init
2. terraform plan -out=plan.tf (review)
3. terraform apply plan.tf

Ensure your aws credentials are available where you run Terraform, and check terraform/outputs.tf for values that need to be copied into GitHub secrets.

9. App (code) overview and local testing

- Node.js app in app/ directory
- Key files: package.json, server.js (or index.js), Dockerfile
- Healthcheck endpoint: /health
- Local test commands:
  - npm install
  - npm test
  - npm start

10. Environment variables & secrets

- App-specific env vars should be declared in the workflow or provided via Kubernetes Secret objects.
- Common variables:
  - NODE_ENV
  - DATABASE_URL (if used)
  - PORT

Kubernetes secrets can be created from GitHub Action outputs or external secret managers (recommended: AWS Secrets Manager or SSM Parameter Store).

11. Deploy target (EKS / ECR) — detailed steps

Push image to ECR (typical job snippet):

1. aws ecr get-login-password --region $AWS_REGION | docker login --username AWS --password-stdin $ECR_URI
2. docker build -t $IMAGE_NAME:$TAG .
3. docker tag $IMAGE_NAME:$TAG $ECR_URI/$IMAGE_NAME:$TAG
4. docker push $ECR_URI/$IMAGE_NAME:$TAG

Deploy to Kubernetes (typical job snippet):

1. Configure kubeconfig (via aws eks update-kubeconfig --region $AWS_REGION --name $CLUSTER_NAME or decode KUBECONFIG secret)
2. kubectl set image deployment/${DEPLOYMENT_NAME} ${CONTAINER_NAME}=$ECR_URI/$IMAGE_NAME:$TAG --record

3. kubectl rollout status deployment/${DEPLOYMENT_NAME}

12. Observability & notifications (Slack webhook)

- The README includes Slack webhook creation steps. Use the curl command in a workflow step to notify Slack on success/failure.
- For production-grade monitoring, integrate Prometheus + Grafana or CloudWatch Container Insights for EKS.

13. Troubleshooting checklist

- **Build fails**: check npm ci/npm test logs in Actions; run locally.
- **Docker push fails**: verify AWS credentials & ECR repo URI; confirm ecr:GetAuthorizationToken permission.
- **Deploy fails**: ensure kubeconfig is valid and GitHub Action runner has access to update kubeconfig or use OIDC.
- **Image not updating in cluster**: verify image tag is unique per run; use kubectl rollout status to inspect.
- **Secrets missing**: verify secrets in GitHub repo settings and their names in workflow.

14. Useful commands

- Local: cd app && npm install && npm test && npm start
- Terraform: cd terraform && terraform init && terraform apply -auto-approve
- Kubernetes: kubectl get pods -n <namespace>
- Inspect image: aws ecr describe-images --repository-name <name>