CI/CD Pipeline Documentation

Overview

This document outlines the continuous integration and continuous deployment (CI/CD) setup for the Django application deployed on AWS Elastic Kubernetes Service (EKS) using GitHub Actions. It covers the complete automation of Docker image builds, ECR uploads, EKS deployments, and VPN-based secure access.

Pipeline Stages

1. Build & Push (Docker to Amazon ECR)

- **Checkout Code**: Uses `actions/checkout@v3` to pull the latest code.
- **AWS Credentials**: Configures AWS CLI using GitHub Secrets.
- **ECR Login**: Logs into Amazon ECR using `aws-actions/amazon-ecr-login@v2`.
- **Docker Build**: Builds Docker image from the Django source directory with tag format `<ECR_REGISTRY>/<REPO>:<RUN_NUMBER>`.
- **Docker Push**: Pushes the built image to the Amazon ECR repository.

Test Stage Details:

- **Python Setup:**
  - Uses actions/setup-python@v4 with Python 3.11
- **Dependency Installation:**
  - Installs from requirements.txt
  - Adds dev tools: flake8, pytest, pytest-django
- **Linting: (a**utomatically analyzing your code to find potential errors, bugs, or style issues**)**
  - Lints the Django app using flake8
- **Test Execution:**
  - Looks for test_*.py under tests/ directory
  - Runs pytest if test files exist
  - Logs and skips test run if no tests found

2. Deploy to EKS (with VPN)

- **VPN Setup**:
  - Installs `openvpn` on the GitHub Actions runner.
  - Decodes the Base64 `.ovpn` configuration into a file.
  - Authenticates using secrets `VPN_USER` and `VPN_PASS`.
  - Connects to VPN and confirms the process is running.
- **kubectl Setup**:

    - Installs `kubectl` manually using a pinned version (`v1.30.1`).
    - Updates kubeconfig to point to the private EKS cluster.
- **Image Injection in YAML**:
    - Replaces the placeholder `REPLACE_IMAGE` in `k8s/deployment.yaml` with the correct ECR image URL.
- **Kubernetes Apply**:
    - Deploys the updated manifest to EKS using `kubectl apply`.
    - Validates rollout using `kubectl rollout status`.
- **VPN Disconnect**:
    - Ensures cleanup by killing the OpenVPN process post-deployment.

GitHub Actions Workflow File

Location: `.github/workflows/main.yaml`

Stages Implemented:

1. Build:

- Code checkout
- Configure AWS credentials
- Docker build using multistage build
- Push Docker image to ECR

2. Test:

- Python setup
- Install dependencies
- Run `flake8` for linting
- Check for test files and run `pytest` (if present)

3. Deploy:

- Connect to VPN using OpenVPN
- Install `kubectl`, update kubeconfig
- Replace image in `k8s/deployment.yaml`
- Deploy to EKS via `kubectl apply`
- Verify rollout with `kubectl rollout status`

Required GitHub Secrets

Name	Purpose
`AWS_ACCESS_KEY_ID_EKS`	IAM user for CI/CD with ECR/EKS access
`AWS_SECRET_ACCESS_KEY_EKS`	IAM secret key
`AWS_REGION_EKS`	e.g., `ap-south-1`
`EKS_CLUSTER_NAME_EKS`	Your EKS cluster name
`VPN_PROFILE_B64`	Base64-encoded VPN config (.ovpn)
`VPN_USER`	VPN login username
`VPN_PASS`	VPN login password

Best Practices Followed

- Docker multi-stage build for lean image.
- Secrets stored securely using GitHub Secrets.
- Separate build, test and deploy jobs.
- VPN guardrail for private EKS access.
- Kubernetes manifest updated dynamically per deployment.

Conclusion

This CI/CD setup automates the full lifecycle of application delivery to a secure AWS EKS environment using GitHub Actions, Docker, and OpenVPN. It ensures consistency, traceability, and secure connectivity for each deployment run.
