Continuous Delivery & Deployment — Quick-Start
GitHub Actions to AWS (Amazon ECR + Amazon EKS)

Practical DevOps Handout

November 4, 2025

Contents

1 Goals and Scope

This quick-start shows how to build a container with GitHub Actions, push it to **Amazon ECR**, and deploy it to **Amazon EKS**. We use **GitHub OpenID Connect (OIDC)** to assume an AWS IAM role (no static keys).

You will:

- Create an ECR repository and an EKS cluster (Managed Node Group or Fargate).
- Create an IAM role trusted by GitHub OIDC with least-privilege permissions.
- Map that role into EKS RBAC via the `aws-auth` ConfigMap.
- Deploy a `Deployment` & `Service`, then roll out new images on pushes to `main`.

Outcomes:

- Commits to `main` build, tag, and push an image to ECR and roll the EKS `Deployment`.
- Review gates via GitHub Environments (`production`).
- Optional status badge in `README.md`.

2 One-Time AWS Setup

2.1 Create or Choose an ECR Repository

Pick a region (e.g., `us-east-1`) and repository name (e.g., `myapp`).

```
aws ecr create-repository \
  --repository-name myapp \
  --image-scanning-configuration scanOnPush=true \
  --region us-east-1
```

2.2 Create an EKS Cluster

You can use the console or `eksctl`. Example (Managed Node Group):

```
eksctl create cluster \
  --name my-eks \
  --region us-east-1 \
  --nodes 2 --node-type t3.small
```

Or with Fargate (serverless pods):

```
eksctl create cluster --name my-eks --region us-east-1 --fargate
```

2.3 IAM Role for GitHub OIDC

Enable the GitHub OIDC provider once per account. Then create an IAM role (e.g., `GitHubActionsDeployRole`) with this trust policy. Replace `ACCOUNT_ID`, `OWNER`, and `REPO`; the `sub` pins to the `main` branch.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated":
        ↪ "arn:aws:iam::ACCOUNT_ID:oidc-provider/token.actions.githubusercontent.com"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "token.actions.githubusercontent.com:aud": "sts.amazonaws.com",
          "token.actions.githubusercontent.com:sub":
          ↪ "repo:OWNER/REPO:ref:refs/heads/main"
        }
      }
    }
  ]
}
```

Attach a minimal policy for ECR push and EKS describe (needed to build kubeconfig). Scope ARNs where possible.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "EcrPushPull",
      "Effect": "Allow",
      "Action": [
        "ecr:GetAuthorizationToken",
        "ecr:BatchCheckLayerAvailability",
        "ecr:CompleteLayerUpload",
        "ecr:UploadLayerPart",
        "ecr:InitiateLayerUpload",
        "ecr:PutImage",
        "ecr:DescribeRepositories",
        "ecr:CreateRepository"
      ],
      "Resource": "*"
    },
    {
      "Sid": "EksDescribe",
      "Effect": "Allow",
      "Action": [ "eks:DescribeCluster" ],
      "Resource": "*"
    }
  ]
}
```

Map the Role into EKS RBAC Add the role ARN to the cluster's `aws-auth` ConfigMap so kubectl can act. For quick-start you can map to `system:masters`; in production, map to a specific group and bind a scoped `ClusterRole`.

```
kubectl edit configmap aws-auth -n kube-system
```

Then add an entry like:

```
mapRoles:
- rolearn: arn:aws:iam::ACCOUNT_ID:role/GitHubActionsDeployRole
  username: gha-deployer
  groups:
    - system:masters
```

Node IAM for ECR pulls For Managed Node Groups, ensure the node instance role has `AmazonEC2ContainerRegistryReadOnly`. Fargate profiles have ECR access handled by the service.

3 GitHub Repository Setup

3.1 Environment and Protection Rules

Create `production` environment. Optionally add required reviewers and a wait timer.

3.2 Status Badge in README

```
1 ![deploy](https://github.com/OWNER/REPO/actions/workflows/deploy.yml/badge.svg)
```

4 Kubernetes Manifests (Apply Once)

A minimal `Deployment` and `Service` exposing port 3000. Replace `ACCOUNT_ID`, `REGION`, and names.

Listing 1: deployment.yaml + service.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
  namespace: prod
spec:
  replicas: 2
  selector:
    matchLabels: { app: myapp }
  template:
    metadata:
      labels: { app: myapp }
    spec:
      containers:
      - name: app
        image: ACCOUNT_ID.dkr.ecr.REGION.amazonaws.com/myapp:latest
        ports: [{containerPort: 3000}]
---
apiVersion: v1
kind: Service
metadata:
  name: myapp
  namespace: prod
spec:
  type: LoadBalancer
  selector: { app: myapp }
  ports:
  - name: http
    port: 80
    targetPort: 3000
```

Apply once:

```
kubectl create namespace prod
kubectl apply -n prod -f k8s/deployment.yaml
kubectl apply -n prod -f k8s/service.yaml
```

5 GitHub Actions Workflow (ECR + EKS Rollout)

Save as `.github/workflows/deploy.yml`. Adjust variables under `env:`.

Listing 2: Build, push to ECR, and roll out on EKS

```yaml
name: build-and-deploy

on:
  workflow_dispatch:
  push:
    branches: [ "main" ]

permissions:
  contents: read
  id-token: write   # OIDC for AWS

env:
  AWS_REGION: us-east-1
  ECR_REPO: myapp
  CLUSTER_NAME: my-eks
  K8S_NAMESPACE: prod
  K8S_DEPLOYMENT: myapp
  K8S_CONTAINER: app
  K8S_SERVICE: myapp

jobs:
  build:
    runs-on: ubuntu-latest
    outputs:
      image: ${{ steps.meta.outputs.image }}
    steps:
      - name: Checkout
        uses: actions/checkout@v4

      - name: Configure AWS credentials (OIDC)
        uses: aws-actions/configure-aws-credentials@v4
        with:
          role-to-assume: arn:aws:iam::ACCOUNT_ID:role/GitHubActionsDeployRole
          aws-region: ${{ env.AWS_REGION }}

      - name: Ensure ECR repository exists
        run: |
          aws ecr describe-repositories --repository-names "$ECR_REPO" >/dev/null 2>&1 || \
          aws ecr create-repository --repository-name "$ECR_REPO" \
            --image-scanning-configuration scanOnPush=true

      - name: Log in to Amazon ECR
        id: ecr
        uses: aws-actions/amazon-ecr-login@v2

      - name: Build and push image
        uses: docker/build-push-action@v6
        with:
```

```yaml
          context: .
          push: true
          tags: |
            ${{ steps.ecr.outputs.registry }}/${{ env.ECR_REPO }}:latest
            ${{ steps.ecr.outputs.registry }}/${{ env.ECR_REPO }}:${{ github.sha }}

      - name: Set image URI output
        id: meta
        run: echo "image=${{ steps.ecr.outputs.registry }}/${{ env.ECR_REPO }}:${{ github.sha }}"
        ↪  >> $GITHUB_OUTPUT

  deploy:
    runs-on: ubuntu-latest
    needs: build
    environment:
      name: production
      url: ${{ steps.svc.outputs.url }}
    steps:
      - name: Configure AWS credentials (OIDC)
        uses: aws-actions/configure-aws-credentials@v4
        with:
          role-to-assume: arn:aws:iam::ACCOUNT_ID:role/GitHubActionsDeployRole
          aws-region: ${{ env.AWS_REGION }}

      - name: Update kubeconfig
        run: aws eks update-kubeconfig --name "$CLUSTER_NAME" --region "$AWS_REGION"

      - name: Set new image on Deployment
        run: |
          kubectl -n "$K8S_NAMESPACE" set image deployment/"$K8S_DEPLOYMENT" \
            "$K8S_CONTAINER"="${{ needs.build.outputs.image }}" --record

      - name: Wait for rollout
        run: kubectl -n "$K8S_NAMESPACE" rollout status deployment/"$K8S_DEPLOYMENT" --timeout=5m

      - name: Discover Service URL
        id: svc
        run: |
          URL=$(kubectl -n "$K8S_NAMESPACE" get svc "$K8S_SERVICE" \
            -o jsonpath='{.status.loadBalancer.ingress[0].hostname}')
          if [ -n "$URL" ]; then echo "url=https://$URL" >> $GITHUB_OUTPUT; fi
```

5.1 Why this works

- `id-token: write` + `configure-aws-credentials` enables short-lived AWS creds via OIDC.
- `aws eks update-kubeconfig` fetches the cluster endpoint/cert and uses your IAM role identity; EKS RBAC is granted by `aws-auth`.
- A simple `kubectl set image` + `rollout status` implements a clean, observable deploy.

6 Operational Tips

6.1 Zero-downtime and health checks

Use `readinessProbe` and `livenessProbe` on the container. Keep `maxUnavailable=0` if you require strict availability.

6.2 Security hardening

- Restrict the IAM trust policy to specific branches, tags, or environments.
- Scope policy `Resource` to your ECR repo ARN and (optionally) the specific cluster ARN for `eks:DescribeCluster`.
- In-cluster AWS access should use IRSA (IAM Roles for Service Accounts), not node credentials.

6.3 Cost awareness

EKS control plane has a fixed cost; nodes bill per instance (or Fargate per vCPU/memory). ECR charges for storage and data transfer. Start small.

7 Troubleshooting

Access denied to cluster. Ensure your IAM role appears in `aws-auth` and has a ClusterRoleBinding matching the verbs you need. Re-run `aws eks update-kubeconfig`.

Image cannot pull. Confirm node role has `AmazonEC2ContainerRegistryReadOnly` and the image URI/tag is correct; check `kubectl describe pod`.

Service has no external hostname. If using `type: LoadBalancer` on EKS in private subnets, provision an Ingress (ALB) or expose via `NodePort` and an external LB.

8 Appendix A: Minimal Dockerfile

```dockerfile
FROM node:20-alpine
WORKDIR /app
COPY package*.json ./
RUN npm ci --omit=dev
COPY . .
EXPOSE 3000
CMD ["node", "server.js"]
```

9 Appendix B: Variables and Secrets Checklist

- **GitHub Environment:** `production`
- **Workflow env:** `AWS_REGION`, `ECR_REPO`, `CLUSTER_NAME`, `K8S_NAMESPACE`, `K8S_DEPLOYMENT`, `K8S_CONTAINER`, `K8S_SERVICE`
- **IAM:** GitHub OIDC provider, role trust policy, least-privilege policy, node role ECR read access.