# AWS CodePipeline with ECR and Helm: Automated Kubernetes Deployment

## Table of Contents



---

## Overview

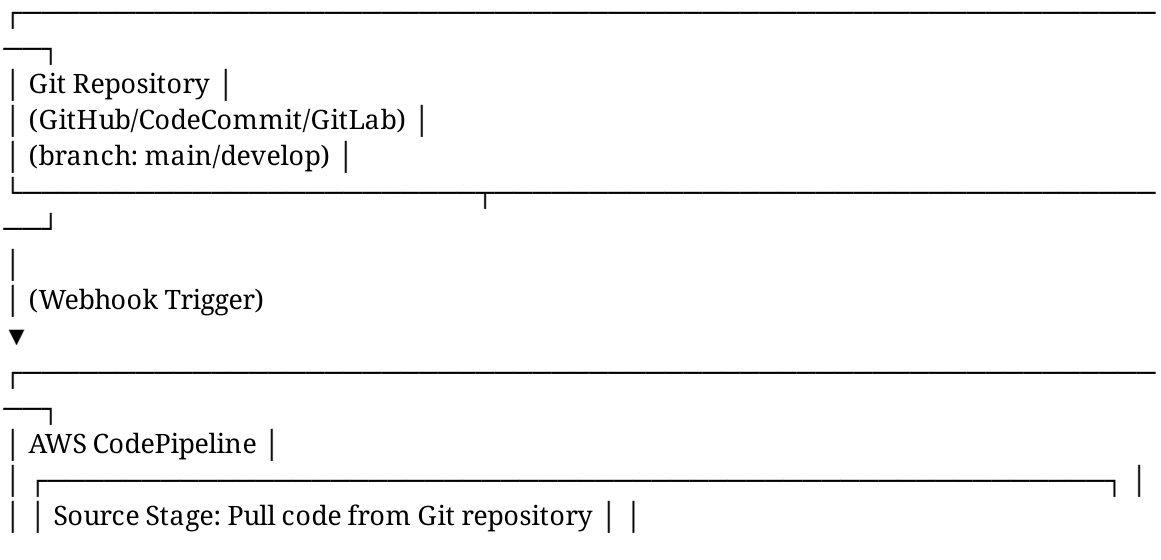
This project demonstrates an automated CI/CD pipeline built on AWS CodePipeline that:

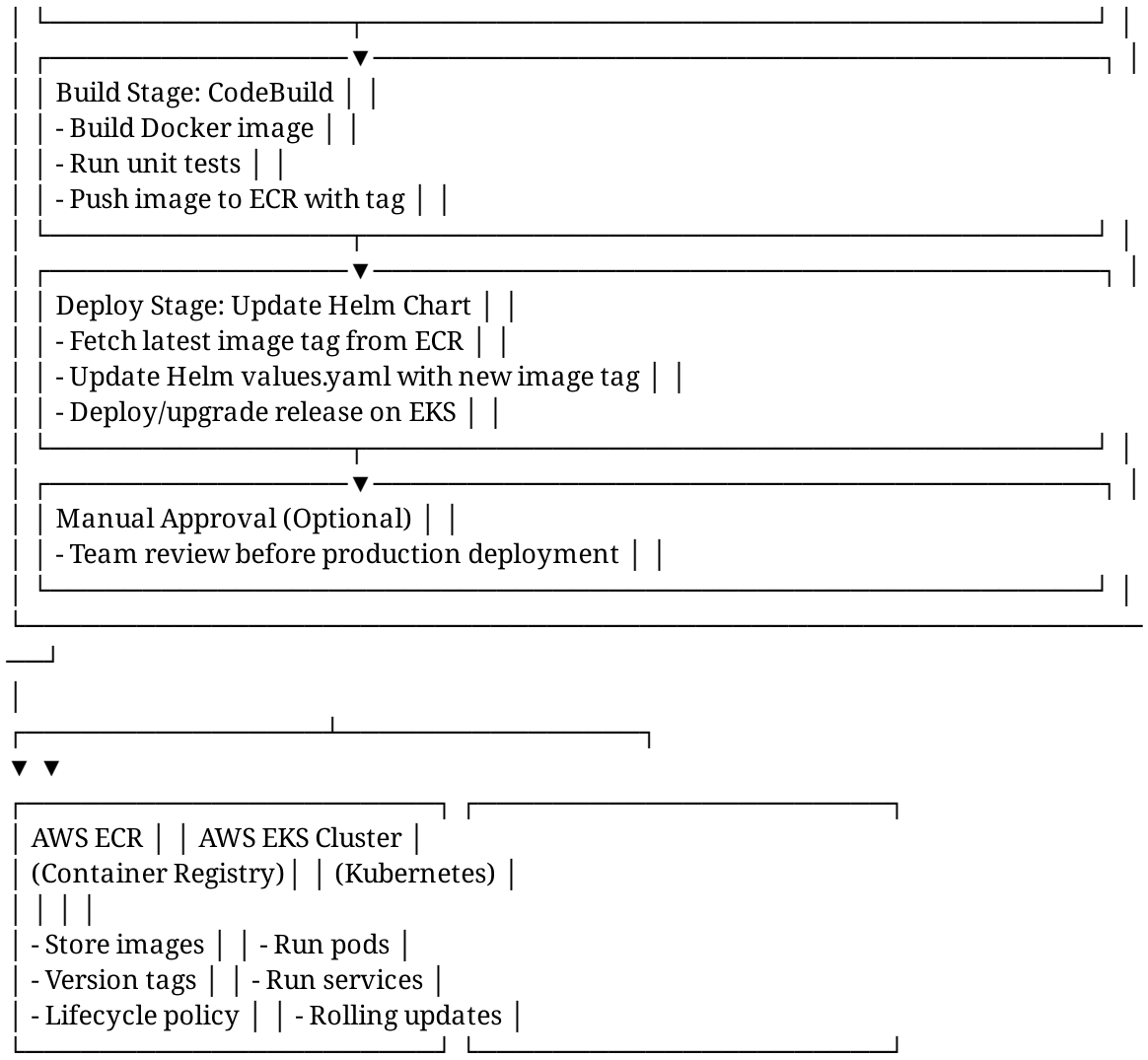
1. **Detects code changes** in a Git repository (GitHub, CodeCommit, or GitLab)
2. **Triggers automatic builds** when code is pushed to specific branches
3. **Builds Docker images** and pushes them to Amazon Elastic Container Registry (ECR)
4. **Updates Helm charts** with the new image version
5. **Deploys to Kubernetes** on AWS EKS with automatic image updates

This end-to-end automation eliminates manual deployments and ensures consistent, repeatable infrastructure management[1].

---

## Architecture






---

# Prerequisites

## AWS Resources Required

- **AWS Account** with appropriate permissions
- **IAM Roles and Policies** configured for CodePipeline, CodeBuild, EKS
- **Amazon EKS Cluster** (1.24 or later)
- **ECR Repository** to store Docker images
- **S3 Bucket** for pipeline artifacts
- **CodeCommit, GitHub, or GitLab** repository with source code

## Tools and Knowledge

- **kubectl** - Kubernetes command-line tool
- **Helm 3** - Package manager for Kubernetes
- **Docker** - Container runtime (local development)
- **AWS CLI** - AWS command-line interface
- **Git** - Version control

### IAM Permissions

Ensure the following IAM permissions are granted[2]:

- `codepipeline:*` - CodePipeline service
- `codebuild:*` - CodeBuild service
- `ecr:*` - ECR push/pull operations
- `eks:*` - EKS cluster access
- `iam:PassRole` - Role assumption for services
- `s3:*` - Artifact bucket operations

---

## Components

### 1. Git Repository with Webhooks

**Purpose:** Trigger pipeline on code changes

- Repository branch: `main` (production), `develop` (staging)
- Webhook configured to notify CodePipeline on push events
- Supported: GitHub, AWS CodeCommit, GitLab

**Configuration:**
GitHub: Settings → Webhooks → Add webhook
URL: AWS CodePipeline webhook endpoint
Events: Push events on specific branches

### 2. AWS CodePipeline

**Purpose:** Orchestrate the entire CI/CD workflow

**Stages:**

- **Source:** Retrieve code from Git
- **Build:** Compile, test, build Docker image
- **Deploy:** Update Helm chart and deploy to EKS

### 3. AWS CodeBuild

**Purpose:** Build Docker images and push to ECR

**Build specification file:** `buildspec.yml`

Example buildspec.yml:
version: 0.2

phases:
pre_build:
commands:
- echo "Logging in to Amazon ECR..."
- aws ecr get-login-password --region $AWS_REGION | docker login --username AWS --password-stdin $AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com
- REPOSITORY_URI=$AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/$ECR_REPO_NAME
- COMMIT_HASH=$(echo

$CODEBUILD_RESOLVED_SOURCE_VERSION|cut - c1 - 7) - IMAGE_TAG =$ {COMMIT_HASH:=latest}

build:
commands:
- echo "Building Docker image..."
- docker build -t $REPOSITORY_URI:IMAGE_TAG .
- docker tag $REPOSITORY_URI:IMAGE_TAG $REPOSITORY_URI:latest

post_build:
commands:
- echo "Pushing Docker image to ECR..."
- docker push $REPOSITORY_URI:IMAGE_TAG
- docker push $REPOSITORY_URI:latest
- echo "Writing image definitions file..."
- printf '[{"name":"app-container","imageUri":"%s"}]' $REPOSITORY_URI:IMAGE_TAG > imagedefinitions.json

artifacts:
files:
- imagedefinitions.json
- Dockerfile
- helm/

## 4. Amazon Elastic Container Registry (ECR)

**Purpose:** Store and manage Docker images

**Features:**

- Image versioning with tags (commit hash, latest, semver)
- Lifecycle policies to clean up old images
- Private repository for secure storage
- Integration with EKS for pull permissions

## 5. Helm Chart

**Purpose:** Templated Kubernetes manifests for deployments

**Chart structure:**
my-app-chart/
├── Chart.yaml
├── values.yaml
├── templates/
│ ├── deployment.yaml
│ ├── service.yaml
│ ├── ingress.yaml
│ └── configmap.yaml

**Example values.yaml:**
replicaCount: 3

```
image:
repository: 123456789.dkr.ecr.us-east-1.amazonaws.com/my-app
tag: "abc1234" # Updated automatically by pipeline
pullPolicy: IfNotPresent

service:
type: LoadBalancer
port: 80
targetPort: 8080

resources:
requests:
memory: "256Mi"
cpu: "100m"
limits:
memory: "512Mi"
cpu: "500m"
```

### 6. AWS EKS Cluster

**Purpose:** Run containerized microservices on managed Kubernetes

- **Node Groups:** Auto-scaling group of EC2 instances
- **Networking:** VPC, subnets, security groups
- **Add-ons:** CoreDNS, VPC CNI, kube-proxy

---

## Setup and Configuration

### Step 1: Create IAM Roles and Policies

Create IAM role for CodeBuild with ECR push permissions:

```
{
"Version": "2012-10-17",
"Statement": [
{
"Effect": "Allow",
"Action": [
"ecr:GetAuthorizationToken",
"ecr:BatchCheckLayerAvailability",
"ecr:GetDownloadUrlForLayer",
"ecr:PutImage",
"ecr:InitiateLayerUpload",
"ecr:UploadLayerPart",
"ecr:CompleteLayerUpload"
],
"Resource": ""
},
{
"Effect": "Allow",
"Action": [
"logs:CreateLogGroup",
```

*"logs:CreateLogStream",*
*"logs:PutLogEvents"*
*],*
*"Resource": "*"
}
]
}

### Step 2: Create ECR Repository

aws ecr create-repository
--repository-name my-app
--region us-east-1

### Step 3: Configure CodeBuild Project

aws codebuild create-project
--name my-app-build
--source type=GITHUB,location=https://github.com/your-org/my-app.git
--environment
computeType=BUILD_GENERAL1_SMALL,image=aws/codebuild/standard:5.0,type=LINUX_CONTAINER
--service-role arn:aws:iam::ACCOUNT_ID:role/CodeBuildRole

### Step 4: Create CodePipeline

aws codepipeline create-pipeline
--cli-input-json file://pipeline-config.json

Example `pipeline-config.json`:

```
{
"pipeline": {
"name": "my-app-pipeline",
"roleArn": "arn:aws:iam::ACCOUNT_ID:role/CodePipelineRole",
"stages": [
{
"name": "Source",
"actions": [
{
"name": "SourceAction",
"actionTypeId": {
"category": "Source",
"owner": "GitHub",
"provider": "GitHub",
"version": "1"
},
"configuration": {
"Owner": "your-org",
"Repo": "my-app",
"Branch": "main"
}
}
```

```
]
},
{
"name": "Build",
"actions": [
{
"name": "BuildAction",
"actionTypeId": {
"category": "Build",
"owner": "AWS",
"provider": "CodeBuild",
"version": "1"
},
"configuration": {
"ProjectName": "my-app-build"
}
}
]
},
{
"name": "Deploy",
"actions": [
{
"name": "DeployAction",
"actionTypeId": {
"category": "Deploy",
"owner": "AWS",
"provider": "AppConfig",
"version": "1"
}
}
]
}
]
}
}
```

### Step 5: Set Up Helm Deployment

Update Helm chart and commit to repository:

```
helm create my-app-chart
cd my-app-chart

# Update Chart.yaml

cat > Chart.yaml << EOF
apiVersion: v2
name: my-app
description: My application Helm chart
```

type: application
version: 1.0.0
appVersion: "1.0"
EOF

# Update values.yaml with image information

helm install my-app ./my-app-chart -n production --create-namespace

---

## Pipeline Workflow

### Trigger: Code Push to Repository

1. Developer pushes code to `main` branch
2. GitHub/CodeCommit webhook fires
3. CodePipeline receives notification

### Stage 1: Source

- CodePipeline retrieves source code from Git
- Stores in S3 artifact bucket

### Stage 2: Build

- CodeBuild project starts
- Executes buildspec.yml:
  - Logs into ECR
  - Builds Docker image with tag from commit hash
  - Runs unit tests
  - Pushes image to ECR
  - Outputs `imagedefinitions.json`

### Stage 3: Deploy

- Lambda function or manual step retrieves latest image tag from ECR
- Updates Helm `values.yaml` with new image tag:
  image:
  tag: "abc1234" # New commit hash
- Executes Helm upgrade:
  helm upgrade my-app ./my-app-chart
  --values values.yaml
  --namespace production
- Kubernetes rolling update starts
  - New pods spin up with new image
  - Old pods gradually terminate
  - Service remains available

### Stage 4: Verification

- Monitor deployment status:
  kubectl rollout status deployment/my-app -n production
  kubectl get pods -n production
  kubectl logs -f deployment/my-app -n production

---

## Key Features

### Automatic Triggering[3]

- **Webhook-based:** Triggered on Git push to configured branches
- **No manual intervention:** Fully automated from code commit to deployment
- **Branch-specific:** Different branches can trigger different deployments (dev, staging, production)

### Image Versioning

- **Commit hash tagging:** Each image tagged with short Git commit hash
- **Latest tag:** Always points to the most recent successful build
- **Semantic versioning:** Support for semantic version tags (v1.0.0, v1.1.0)

### Helm Chart Integration[4]

- **Templated deployments:** Reusable Kubernetes manifests
- **Environment-specific values:** Different configurations for dev, staging, production
- **Rollback capability:** Easy rollback to previous releases:
  helm rollback my-app 1 # Rollback to previous version

### Security[5]

- **IAM-based access control:** 100% role-based access control (RBAC) compliance
- **Private ECR repositories:** Images stored securely in AWS
- **Secret management:** Use AWS Secrets Manager for sensitive data
- **Encrypted artifacts:** S3 bucket encryption enabled

### Monitoring and Logging

- **CodePipeline dashboard:** Real-time pipeline execution status
- **CloudWatch logs:** Build and deployment logs
- **CloudWatch alarms:** Alert on pipeline failures
- **EKS events:** Monitor pod startup, crashing, resource exhaustion

---

## Best Practices

## 1. Separate Pipelines by Environment

Create distinct pipelines for development, staging, and production:

main branch → Development pipeline → Deploy to dev EKS cluster
staging branch → Staging pipeline → Deploy to staging EKS cluster
release/* branch → Production pipeline → Deploy to prod EKS cluster (with approval)

## 2. Implement Manual Approval for Production

Add approval gate before production deployment:

```
{
"name": "ProductionApproval",
"actionTypeId": {
"category": "Approval",
"owner": "AWS",
"provider": "Manual",
"version": "1"
}
}
```

## 3. Image Scanning for Vulnerabilities

Configure ECR to scan images on push:

```
aws ecr put-image-scan-config
--repository-name my-app
--scan-config scanOnPush=true
```

## 4. Lifecycle Policy for ECR Images

Clean up old images to reduce storage costs:

```
{
"rules": [
{
"rulePriority": 1,
"description": "Keep last 10 production images",
"selection": {
"tagStatus": "tagged",
"tagPrefixList": ["v"],
"countType": "imageCountMoreThan",
"countNumber": 10
},
"action": {
"type": "expire"
}
}
]
}
```

### 5. Helm Chart Versioning

Maintain chart versions aligned with application versions:

# Chart.yaml

version: 1.2.0 # Chart version
appVersion: "1.2.0" # Application version

### 6. Comprehensive Testing

Add test stages before production deployment:

# buildspec.yml

phases:
test:
commands:
- echo "Running unit tests..."
- npm test
- echo "Running integration tests..."
- npm run integration-test

### 7. GitOps Principles

Store all Helm charts and configurations in Git for auditability:

infrastructure-repo/
├── helm-charts/
│ └── my-app/
│ ├── values-dev.yaml
│ ├── values-staging.yaml
│ └── values-prod.yaml
└── pipelines/
├── dev-pipeline.yaml
├── staging-pipeline.yaml
└── prod-pipeline.yaml

---

## Troubleshooting

### Issue: CodeBuild Fails to Push to ECR

**Symptoms:** `AccessDenied: User is not authorized to perform: ecr:PutImage`

**Solution:**

1. Verify CodeBuild IAM role has ECR permissions
2. Ensure role is attached to CodeBuild project
3. Check ECR repository policies allow push from CodeBuild role

### Issue: Helm Deployment Fails

**Symptoms:** `error: release my-app failed, and has been uninstalled`

**Solution:**

# Check Helm release status

helm status my-app -n production

# View previous releases

helm history my-app -n production

# Rollback to previous version

helm rollback my-app 1 -n production

# Check pod logs for errors

kubectl logs -f deployment/my-app -n production

### Issue: Pod Crash Loop After Deployment

**Symptoms:** Pods stuck in `CrashLoopBackOff` state

**Solution:**

# Check pod events

kubectl describe pod POD_NAME -n production

# View container logs

kubectl logs POD_NAME -n production

# Check resource requests/limits

kubectl get pod POD_NAME -o yaml -n production

# Rollback to previous working version

helm rollback my-app 1 -n production

### Issue: Pipeline Trigger Not Working

**Symptoms:** Commits to Git don't trigger pipeline

**Solution:**

1. Verify webhook is configured in Git repository settings
2. Check webhook delivery history for failures
3. Confirm CodePipeline has permissions to read from repository
4. Test webhook manually:
   aws codepipeline put-job-success-result --job-id JOB_ID

### Issue: Image Tag Not Updating in Helm

**Symptoms:** Helm chart still uses old image after new build

**Solution:**

1. Verify buildspec.yml correctly outputs image tag
2. Check Helm values.yaml is being updated with new tag
3. Verify deploy script fetches latest ECR image tag
4. Manual update (if needed):
   helm upgrade my-app ./my-app-chart
   --set image.tag=NEW_TAG
   -n production

---

## Repository Structure

```
my-app-repository/
├── src/
│   ├── main.py
│   ├── app.py
│   └── requirements.txt
├── helm/
│   └── my-app/
│   ├── Chart.yaml
│   ├── values.yaml
│   └── templates/
│   ├── deployment.yaml
│   └── service.yaml
├── Dockerfile
├── buildspec.yml
├── .github/
│   └── workflows/
│   └── webhook.yml
├── README.md
└── .gitignore
```

## Conclusion

This automated CI/CD pipeline leverages AWS CodePipeline, CodeBuild, and ECR alongside Kubernetes (EKS) and Helm to create a production-grade deployment system. The workflow:

- ✅ Eliminates manual deployments
- ✅ Ensures consistency across environments
- ✅ Provides rapid feedback on code quality
- ✅ Enables secure, auditable deployments
- ✅ Supports rollback capabilities
- ✅ Scales with application growth

For more information, refer to the official AWS and Kubernetes documentation[6][7].

---

## References


[1] AWS CodePipeline Documentation. (2025). Continuous delivery with AWS CodePipeline. https://docs.aws.amazon.com/codepipeline/

[2] AWS Identity and Access Management. (2025). IAM roles and policies for AWS CodePipeline. https://docs.aws.amazon.com/IAM/latest/UserGuide/

[3] GitHub. (2025). Webhooks and integrations. https://docs.github.com/en/developers/webhooks-and-events/

[4] Helm Project. (2025). Helm 3 documentation. https://helm.sh/docs/

[5] AWS Security Best Practices. (2025). Container security in AWS. https://aws.amazon.com/containers/security/

[6] AWS EKS Documentation. (2025). Amazon Elastic Kubernetes Service. https://docs.aws.amazon.com/eks/

[7] Kubernetes Documentation. (2025). Kubernetes official documentation. https://kubernetes.io/docs/