

Capstone Project: Production Deployment on AWS EKS - Step-by-Step Guide

Project Goal

The goal of this capstone project is to demonstrate your ability to structure, containerize, secure, and deploy a multi-component application stack into a production-like Kubernetes environment (AWS EKS) using a fully automated CI/CD pipeline.

Focus: DevOps, CI/CD, Kubernetes, ECR, and best practices. **Not** software development.

Duration: 7 Days (1 Week)

Part 1: Mandatory Requirements & Setup

1.1. Naming Convention (Mandatory)

All resources must be uniquely identified using your name. This is critical for resource isolation and management within the shared cluster.

Resource	Mandatory Naming Format	Example (for John Doe)
GitHub Repository	<u>firstname-lastname-capstone</u>	<u>john-doe-capstone</u>
Kubernetes Namespace	<u>firstname-lastname</u>	<u>john-doe</u>
ECR Repository	<u>.../firstname-lastname/frontend</u>	<u>.../john-doe/frontend</u>
Helm Maintainer	<u><firstname lastname> <email></u>	<u>John Doe <john.doe@company.com></u>

1.2. Application Stack (Keep it Simple)

You must deploy a minimum of two components:

- Frontend:** A simple static site served by Nginx, or a minimal React/Vue application.
 - Requirement:** Must have a homepage and a health endpoint (/health).

- **Requirement:** Must make one successful API call to your Backend service.
- 2 **Backend:** A simple microservice built with FastAPI or Flask.
 - **Requirement:** Must have a readiness endpoint (/health).
 - **Requirement:** Must expose a simple CRUD or single-route API.
 - **Requirement:** Must use environment variables for database connection details.

1.3. Environment Details

Detail	Value	Note
AWS Region	<u>provided-by-devops</u>	All resources must be deployed here.
Container Registry	AWS ECR Only	No Docker Hub or GHCR is permitted.
Kubernetes Cluster	Shared AWS EKS Cluster	Provided by the DevOps team.
Database	PostgreSQL	Connection details provided via Kubernetes Secret.

Part 2: Step-by-Step Execution Plan

Phase 1: Application & Repository Setup

Step 1.1: Create GitHub Repository

- 3 Create a new public GitHub repository named firstname-lastname-capstone.
- 4 Set up **Branch Protection Rules** for the main branch:
 - Require pull request reviews (minimum 1 reviewer).
 - Require status checks to pass before merging.
- 5 Clone the repository and create the following structure:

```
firstname-lastname-capstone/
```

```
|— frontend/  
|— backend/  
|— k8s/  
|— .github/workflows/
```

Step 1.2: Develop Application Components

- 6 **Frontend:** Create a minimal application (e.g., a simple index.html served by Nginx, or a basic React app). Ensure it has a /health endpoint and attempts to call the backend API.
- 7 **Backend:** Create a simple FastAPI/Flask application. Ensure it has a /health endpoint and is configured to read database credentials from environment variables.

Step 1.3: Create Production-Ready Dockerfiles

- 8 For both frontend and backend, create a Dockerfile using **multi-stage builds** to minimize image size and improve security.
- 9 Ensure your Dockerfiles use a non-root user and follow security best practices.

Phase 2: CI/CD Pipeline (GitHub Actions & ECR)

Step 2.1: Configure AWS Credentials

- 10 In your GitHub repository, go to **Settings** → **Secrets and variables** → **Actions**.
- 11 Add the following secrets (provided by the DevOps team):
 - AWS_ACCESS_KEY_ID
 - AWS_SECRET_ACCESS_KEY

Step 2.2: Create ECR Repositories

- 12 Manually create two ECR repositories in the provided-by-devops region:
 - firstname-lastname/frontend
 - firstname-lastname/backend

Step 2.3: Implement GitHub Actions Workflow

Create a single workflow file (.github/workflows/ci-cd.yml) that performs the following sequence on every push to main:

- 13 **Checkout Code**
- 14 **Configure AWS Credentials** (using the secrets from Step 2.1)
- 15 **Login to ECR** (using the AWS ECR login action)
- 16 **Build Frontend Image** (using multi-stage Dockerfile)
- 17 **Push Frontend Image** to firstname-lastname/frontend:latest and firstname-lastname/frontend:<git-sha>
- 18 **Build Backend Image** (using multi-stage Dockerfile)

- 19 **Push Backend Image** to firstname-lastname/backend:latest and firstname-lastname/backend:<git-sha>
- 20 **Deploy to EKS** (using kubectrl or helm to apply manifests to your namespace).

Phase 3: Kubernetes Deployment

Step 3.1: Create Kubernetes Manifests

In your k8s/ directory, create the following YAML files:

- 21 **namespace.yaml**: Defines your unique namespace (firstname-lastname).
- 22 **deployment-frontend.yaml**:
 - Uses your ECR image.
 - Sets resource requests and limits.
 - Includes liveness and readiness probes pointing to /health.
 - Uses imagePullSecrets for ECR authentication (details provided by DevOps team).
- 23 **service-frontend.yaml**: Defines a ClusterIP service for the frontend.
- 24 **deployment-backend.yaml**:
 - Uses your ECR image.
 - Sets resource requests and limits.
 - Includes liveness and readiness probes pointing to /health.
 - Mounts the database credentials from the provided Kubernetes Secret.
- 25 **service-backend.yaml**: Defines a ClusterIP service for the backend.
- 26 **ingress.yaml**: Defines an Ingress resource to route external traffic to your Frontend Service (e.g., https://<firstname-lastname>.capstone.company.com).

Step 3.2: Deploy and Verify

- 27 Commit and push all changes to your main branch.
 - 28 Verify that your GitHub Actions workflow successfully:
 - Builds and pushes both images to ECR.
 - Applies all Kubernetes manifests to the EKS cluster.
 - 29 Verify the deployment on the cluster:
 - Check that all pods are running in your namespace (kubectrl get pods -n firstname-lastname).
 - Verify the Ingress is working by accessing your application URL.
 - Verify the Frontend can successfully call the Backend API.
-

Part 3: Deliverables & Evaluation

3.1. Mandatory Deliverables (Pass/Fail)

- 30 **Working CI/CD Pipeline:** The GitHub Actions workflow must successfully run and deploy the application on every push to main.
- 31 **Deployed Application:** Frontend and Backend must be running in your unique namespace on the EKS cluster.
- 32 **Ingress Routing:** The application must be accessible via the cluster's Nginx Ingress Controller.
- 33 **Resource Management:** All deployments must have correct requests and limits defined.
- 34 **Documentation:** A comprehensive README.md and a RUNBOOK.md detailing troubleshooting steps for common issues (e.g., pod crash loop, high latency).

3.2. Bonus Points Criteria (Extra Credit)

You will receive extra credit for implementing the following advanced features:

Feature	Description
Terraform IaC	Using Terraform to provision your ECR repositories and possibly the EKS cluster components (if you chose to go deep).
EFK Stack	Configuring Fluentd in your namespace to send logs to the central Elasticsearch/CloudWatch logging solution.
Network Policies	Implementing a NetworkPolicy to restrict traffic between your Frontend and Backend services.
RBAC Hardening	Creating a dedicated ServiceAccount and Role/RoleBinding for your application deployments.
Secrets Encryption	Using a tool like Bitnami Sealed Secrets to manage your database credentials.
Automated Helm	Creating a reusable Helm chart for your application instead of raw YAML manifests.
Autoscaling (HPA)	Implementing a Horizontal Pod Autoscaler for your Backend service.

3.3. Final Demo

You will present a 10-minute live demonstration of your project:

- 35 **Code Change:** Make a small code change in the backend.

- 36 **CI/CD Demo:** Push the change and show the GitHub Actions workflow successfully building, pushing to ECR, and deploying to EKS.
 - 37 **Verification:** Show the application running, the Frontend calling the Backend, and the pods running in your isolated namespace.
 - 38 **Q&A:** Discuss your design choices, challenges, and how you implemented best practices.
-

Part 4: Resources & Support

Resource	Description
AWS ECR Login Action	https://github.com/aws-actions/amazon-ecr-login
AWS EKS Documentation	https://docs.aws.amazon.com/eks/latest/userguide/
Kubernetes Best Practices	https://kubernetes.io/docs/concepts/configuration/overview/
DevOps Team Contact	dan@tiberbu.com , njoroge@tiberbu.com , elvis@tiberbu.com
Mentor	Daniel, James, Elvis

Good luck! This project is your opportunity to showcase your skills.