Blue-Green Deployment Documentation

# Overview

# Overview

This document provides a complete overview of implementing Blue-Green Deployment for two microservices (`user-service` and `product-service`) using AWS. It covers all steps and services used including GitHub, EC2 (for Docker), Elastic Container Registry (ECR), Elastic Container Service (ECS), Application Load Balancer (ALB), CodePipeline, CodeBuild, and supporting configuration files.

# AWS Services Used

**1. GitHub**

* Serves as the source code repository for both microservices (user-service and product-service).
* Allows version control, code collaboration, and integration with AWS CodePipeline for automated builds.

**2. EC2 (Elastic Compute Cloud)**

* A virtual server where **Docker** is installed to build, test, and push Docker images manually to **Amazon ECR**.
* Useful for initial setup, testing, and debugging outside the automated pipeline.

**3. Amazon ECR (Elastic Container Registry)**

* A fully managed container image registry.
* Stores built Docker images securely, enabling **ECS** to pull the correct image for deployment.

**4. Amazon ECS (Elastic Container Service)**

* Orchestrates and manages Docker container deployments.
* In **Blue-Green deployment**, ECS runs two versions of the application (**blue** and **green**) and switches traffic using **ALB**.

**5. Application Load Balancer (ALB)**

* Distributes incoming requests between ECS tasks running in blue and green environments.
* Facilitates **zero-downtime deployments** by switching traffic only after the new version passes **health checks**.

**6. Target Groups**

* Logical groupings of ECS tasks that the **ALB** routes traffic to.
* Each environment (**blue** and **green**) has its own **target group** to isolate versions.

**7. AWS CodeBuild**

* Builds Docker images from source code using the **buildspec.yml** file.
* Pushes the built images to **Amazon ECR** and generates imagedefinitions.json for ECS deployments.

**8. AWS CodePipeline**

* Automates the CI/CD workflow.
* Integrates **GitHub (source)** → **CodeBuild (build)** → **ECS (deploy)** into one continuous process.
* Supports **Blue-Green deployments** by orchestrating traffic switching in ECS.

**9. IAM Role**

* Grants **CodeBuild**, **CodePipeline**, and **ECS tasks** the permissions they need.
* **Example permissions**:
  + ecr:GetAuthorizationToken, ecr:BatchCheckLayerAvailability, ecr:PutImage, ecr:InitiateLayerUpload
  + ecs:UpdateService, ecs:DescribeServices, ecs:RegisterTaskDefinition
  + elasticloadbalancing:DeregisterTargets, elasticloadbalancing:RegisterTargets
* Without correct IAM role permissions, builds fail with **AccessDeniedException** when trying to interact with ECR, ECS, or ALB.

# Folder Structure

BlueGreenDeployment/  
 |── user-service/  
│ ── app.py  
│ ── Dockerfile  
│ ── requirements.txt  
│ ── buildspec.yml  
│ ── ecs-task-def.json  
│ ── appspec.yaml  
├── product-service/  
├── .github/workflows/  
├── jenkins-pipeline.groovy  
├── .gitignore  
└── README.md

# Blue Green Deployment Steps

**1. Prepare the Application**

* Have two microservices (e.g., **user-service** and **product-service**) in separate folders in your GitHub repository.
* Each service should have:
  + Dockerfile
  + requirements.txt
  + Application code (app.py for Flask)
  + buildspec.yml (for CodeBuild)

The buildspec.yml should include Docker build, tag, push to ECR, and generation of imagedefinitions.json.

**2. Create an Amazon ECR Repository**

* For each microservice, create a private repository in Amazon ECR:
  + Example:  
    03xxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/user-service  
    03xxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/product-service
* Copy the repository URI for use in buildspec and Docker commands.

**3. Build & Push Docker Image (Manual Test)**

From your EC2 instance:

bash

CopyEdit

# Login to ECR

aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 03xxxxxxxx.dkr.ecr.us-east-1.amazonaws.com

# Build Docker image

docker build -t user-service .

# Tag image

docker tag user-service:latest 03xxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/user-service:latest

# Push image to ECR

docker push 03xxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/user-service:latest

**4. Create ECS Cluster**

* Create an ECS Cluster (e.g., blue-green-cluster).
* Choose **Fargate** for serverless container management.

**5. Create Task Definitions**

* Create a **Task Definition** for each microservice.
* Specify:
  + Container name
  + Image URI (from ECR)
  + Port mapping (e.g., 8081 for user-service, 8082 for product-service)
  + Environment variables (if needed)
  + Memory and CPU
* Enable **AWSVPC** networking.

**6. Create ALB & Target Groups**

* Create an **Application Load Balancer** (ALB).
* Create **two Target Groups** for each service:
  + Example for user-service: user-blue-tg and user-green-tg
  + Example for product-service: product-blue-tg and product-green-tg
* Set health check path (e.g., /health) to verify container health.
* Configure ALB listeners (HTTP/HTTPS) to forward traffic to the active target group.

**7. Create ECS Services (Blue Environment)**

* In ECS, create a service for the first version of your app:
  + Launch type: Fargate
  + Load balancer: ALB
  + Target group: Blue target group
  + Desired tasks: 1+
* Deploy and verify the application is running.

**8. Set Up CodeBuild**

* Create a **CodeBuild project** for each service:
  + Source: GitHub repository (specific folder for the microservice)
  + Environment: Managed image with Docker support
  + Buildspec: user-service/buildspec.yml or product-service/buildspec.yml
  + Service role with ECR & ECS permissions.

**9. Create CodePipeline**

* Stages:
  1. **Source**: GitHub (tracks changes in the microservice folder)
  2. **Build**: CodeBuild (builds Docker image, pushes to ECR, outputs imagedefinitions.json)
  3. **Deploy**: ECS Blue/Green deployment action using CodeDeploy.

**10. Blue-Green Deployment Process**

1. **Blue Environment** is live and serving traffic.
2. Deploy new version to **Green Environment**.
3. ALB performs health checks on Green tasks.
4. Once healthy, ALB switches traffic from Blue to Green.
5. Optionally, Blue environment remains running for quick rollback.
6. Stop/terminate Blue environment once confident.

**11. Common Errors & Fixes**

* **AccessDeniedException** → Add proper IAM role permissions (ecr:\*, ecs:\*, elasticloadbalancing:\*).
* **Docker Hub rate limit** → Authenticate with Docker Hub or use AWS public ECR base images.
* **no matching artifact paths found** → Ensure imagedefinitions.json is in root of build output.
* **Task stuck in provisioning** → Check VPC/Subnet/Security group settings.

# EC2 Instance & Docker Commands

Used to manually build and push Docker images to ECR.  
Example commands:

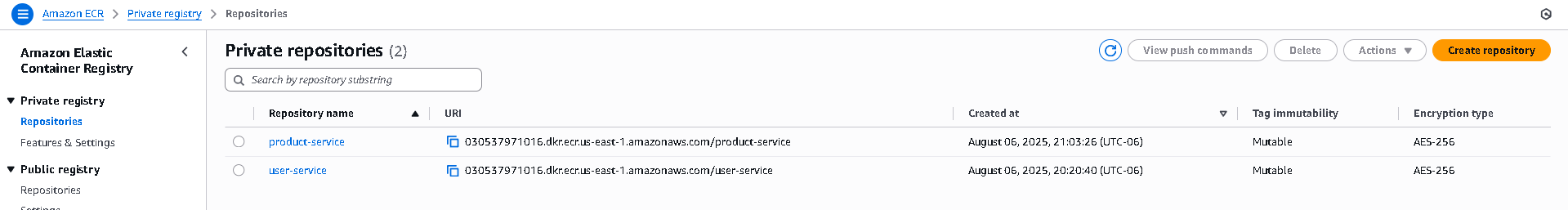
* docker build -t user-service .
* docker tag user-service:latest 03xxxxxxx.dkr.ecr.us-east-1.amazonaws.com/user-service:latest
* aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 03xxxxxxx.dkr.ecr.us-east-1.amazonaws.com
* docker push 03xxxxxxx.dkr.ecr.us-east-1.amazonaws.com/user-service:latest

# Git Commands and Notes

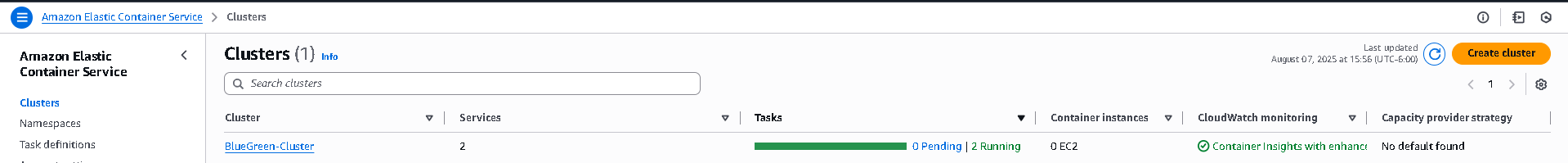
* Git clone
* git status
* git add .
* git commit -m "Message"
* git push origin main

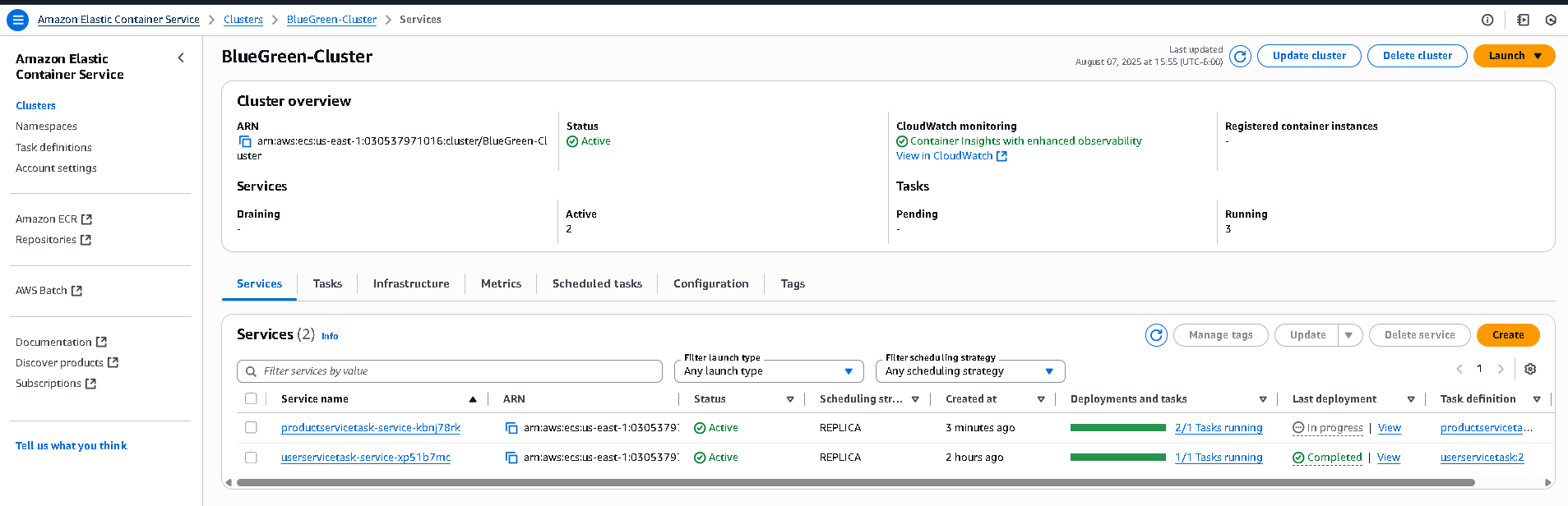
# Amazon ECR Repositories

Two repositories created: `user-service` and `product-service`.

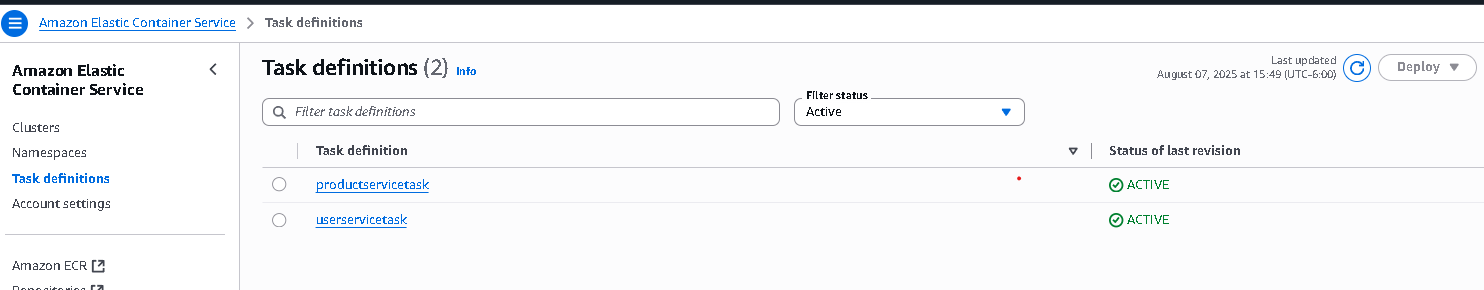


# ECS Cluster and Services

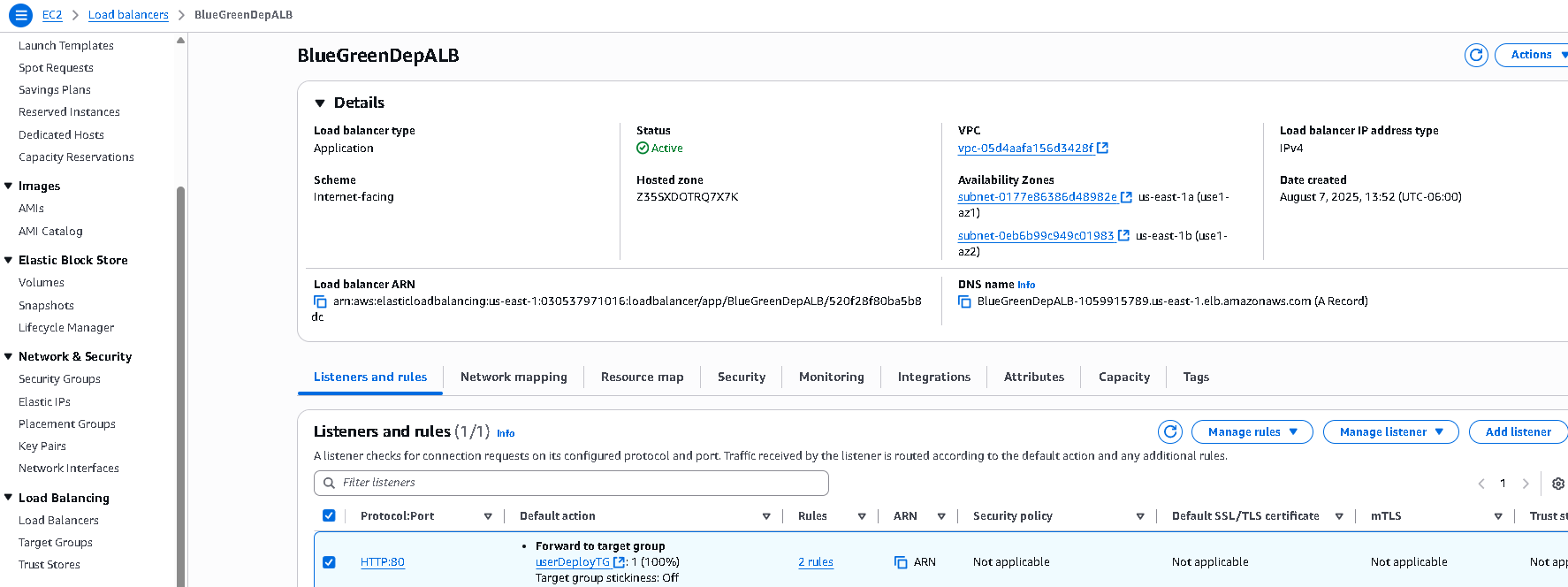




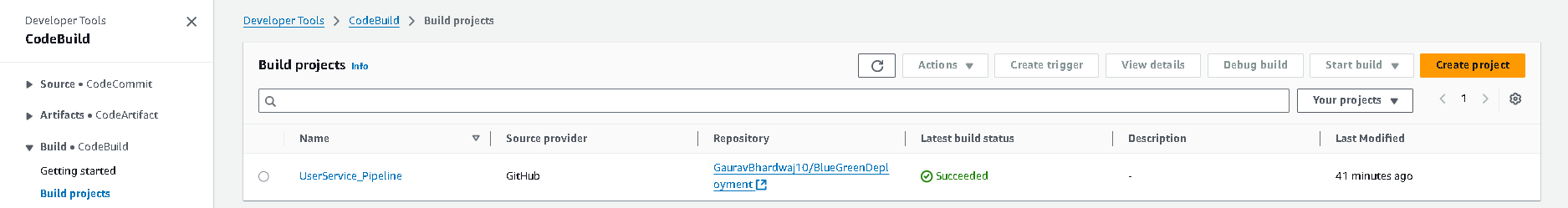
# ECS Task Definitions



# Application Load Balancer



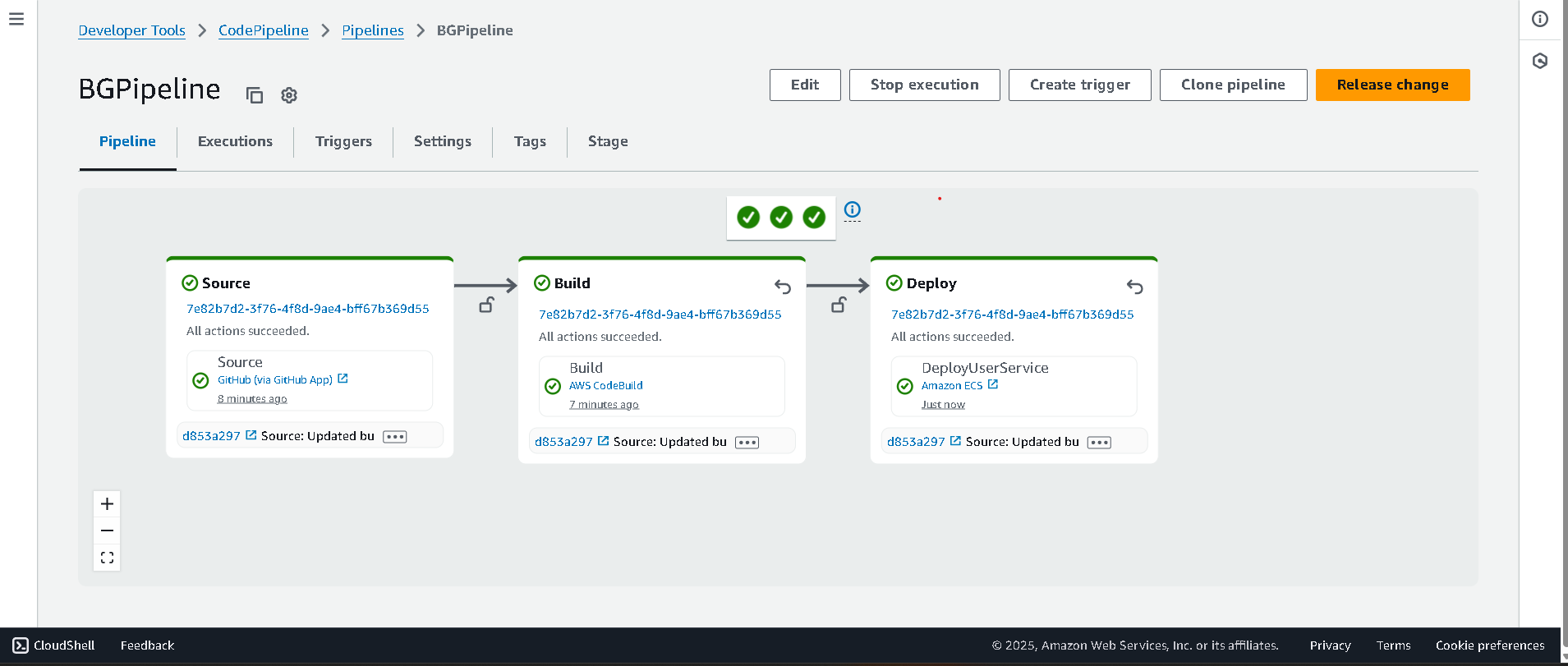
# CodeBuild Project



# CodePipeline

Pipeline stages:

1. **Source** – GitHub (triggered on push to main)
2. **Build** – CodeBuild project (for user or product service)
3. **Deploy** – Amazon ECS deployment using task definition and image



# User Service Output

