PRJ-DOP-021: Canary Deployment Strategy for Microservices

Certification: AWS Certified DevOps Engineer – Professional
Domain: Deployment Strategies

---

1. Project Overview

This project demonstrates how to implement a **canary deployment strategy** for a containerized application running on Amazon ECS. While blue/green deployments are safe, they shift 100% of traffic to the new version at once. A canary release is a more cautious approach where the new version (the “canary”) is gradually exposed to a small subset of production traffic. This allows you to test the new version with real users and monitor for errors or performance degradation before rolling it out to the entire user base.

We will use **AWS CodeDeploy** in conjunction with an **Application Load Balancer (ALB)** to perform a weighted routing of traffic between the old and new versions of our application. The pipeline will deploy the canary, pause to monitor its performance using **CloudWatch alarms**, and then either proceed with a full rollout or automatically roll back if an issue is detected.

Key Objectives

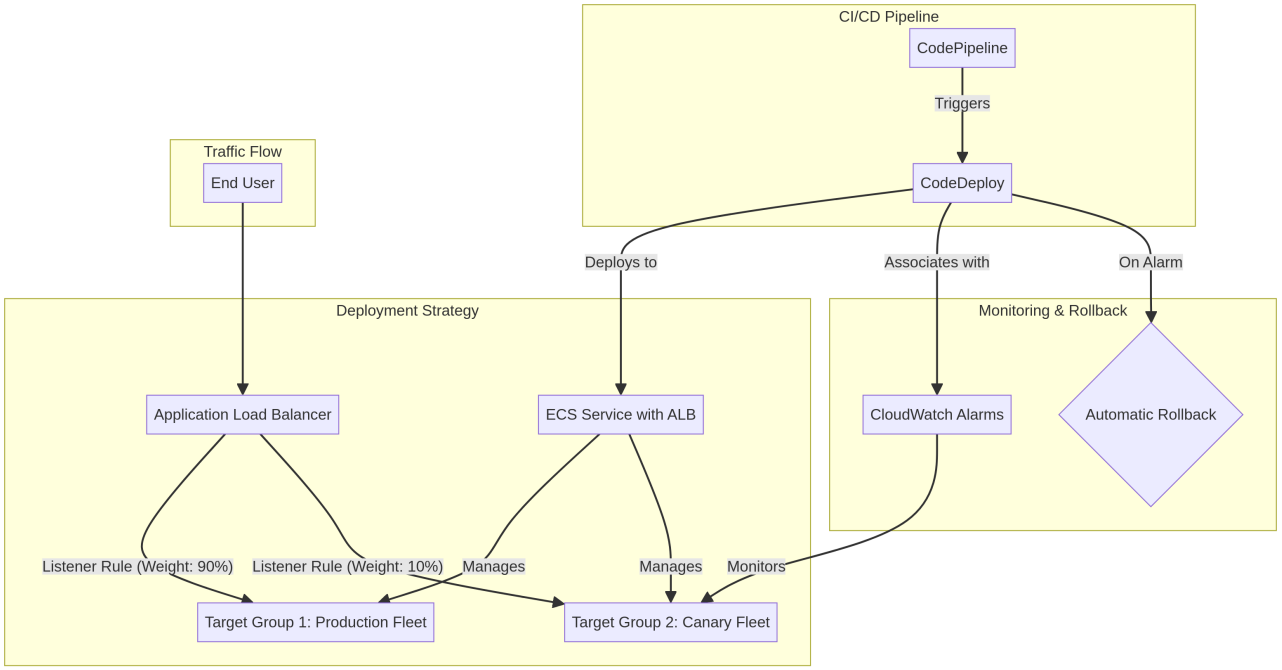
- Configure an Amazon ECS service and Application Load Balancer to support canary deployments.
- Use AWS CodeDeploy to manage the deployment process.
- Implement a traffic-shifting strategy that sends a small percentage of traffic (e.g., 10%) to the new version.
- Create CloudWatch alarms to monitor key metrics of the canary deployment (e.g., HTTP 5xx errors, latency).

- Configure CodeDeploy to automatically roll back the deployment if the CloudWatch alarms are triggered.
- Demonstrate a safe, gradual, and automated release process.

---

2. Architecture

The architecture relies on the ALB's ability to route traffic to multiple target groups with specific weights, orchestrated by CodeDeploy.



Deployment Flow:

1. **Initial State:** The ECS service is running the current production version (v1). The ALB has a listener rule that forwards 100% of the traffic to a primary target group (`Target Group 1`), which contains the v1 tasks.

2. **Deployment Trigger:** A new version (v2) of the application is pushed through the CI/CD pipeline, triggering a **CodeDeploy** deployment.

3. **Canary Launch:** CodeDeploy launches the new v2 tasks and registers them with a second, separate target group (`Target Group 2`). This is the canary fleet.

4. **Traffic Shift:** CodeDeploy modifies the ALB listener rule. It now forwards a small percentage of traffic (e.g., 10%) to the canary target group (`Target Group 2`)

and the remaining 90% to the original production target group (`Target Group 1`).

5. **Baking and Monitoring:** The deployment is paused for a pre-configured "baking" period (e.g., 15 minutes). During this time:
    - **CloudWatch Alarms**, which are associated with the CodeDeploy deployment group, monitor the metrics of the canary target group.
    - If the canary tasks produce a high rate of 5xx errors or increased latency, the CloudWatch alarm will enter the `ALARM` state.
6. **Decision - Rollback or Promote:**
    - **Rollback (Alarm Triggered):** If a CloudWatch alarm is triggered during the baking period, CodeDeploy immediately initiates an automatic rollback. It shifts 100% of the traffic back to the original target group and terminates the canary tasks.
    - **Promote (Success):** If the baking period completes without any alarms, CodeDeploy proceeds with the full rollout. It shifts the remaining traffic to the canary target group and then drains and terminates the old v1 tasks.

---

3. Prerequisites

- An AWS account with administrative permissions.
- A CI/CD pipeline that builds a container image and pushes it to ECR (you can adapt the one from PRJ-SAP-018).
- An ECS Cluster running on Fargate.

---

4. Step-by-Step Implementation Guide

Step 4.1: Create the ECS Service with Two Target Groups

This is the most critical setup step. The ECS service must be created with two target groups from the start.

1. **Create two Target Groups:**
    - Go to the **EC2 Console -> Target Groups -> Create target group**.
    - Create a target group named `app-tg-blue`.
    - Create a second target group named `app-tg-green`.
2. **Create an Application Load Balancer:**
    - Create an ALB and a listener on port 80.
    - Configure the listener's default rule to forward 100% of the traffic to the `app-tg-blue` target group.
3. **Create the ECS Service:**
    - Go to your ECS cluster and **Create service**.
    - **Deployment configuration:**
        - **Application type:** Service
        - **Task Definition:** Your application's task definition.
        - **Service name:** `canary-app-service`
        - **Deployment type:** Blue/green deployment (CodeDeploy)
    - **Load balancing:**
        - Select your ALB.
        - **Target group 1:** `app-tg-blue` (Production listener)
        - **Target group 2:** `app-tg-green` (Test listener)
    - This will create the CodeDeploy Application and Deployment Group for you.

Step 4.2: Configure the CodeDeploy Deployment Group

1. Go to the **CodeDeploy console** and find the deployment group created by ECS.
2. **Edit** the deployment group configuration.
3. **Deployment settings:** Choose **CodeDeployDefault.ECSCanary10Percent15Minutes**.

- This is a built-in configuration that shifts 10% of traffic, waits 15 minutes, and then shifts the remaining 90%.

4. **Create CloudWatch Alarms:**
   - Go to the **CloudWatch Console -> Alarms -> Create alarm**.
   - **Select metric:** Browse to **ApplicationELB -> Per-Target-Group Metrics**.
   - Find the `HTTPCode_Target_5XX_Count` metric for your **canary target group** (`app-tg-green`).
   - **Statistic:** Sum
   - **Period:** 1 minute
   - **Conditions:** `Greater/Equal` to a threshold of `5` for 2 consecutive periods.
   - Create the alarm. Name it `Canary-5XX-Error-Alarm`.
5. **Associate Alarms with CodeDeploy:**
   - Go back to editing your CodeDeploy deployment group.
   - Under **Alarms**, enable alarms and select the `Canary-5XX-Error-Alarm` you just created.
   - Check the box for **Roll back deployment when alarms are in ALARM state**.
   - Save the changes.

Step 4.3: Update the CI/CD Pipeline

Modify your `CodePipeline` to use the new CodeDeploy configuration.

1. Go to your pipeline in the **CodePipeline console** and **Edit** it.
2. In the **Deploy** stage, ensure the provider is **Amazon ECS (Blue/Green)**.
3. Select the correct CodeDeploy application and the deployment group you just configured.
4. The rest of the pipeline (source, build) remains the same as in the previous CI/CD project.

---

5. How to Test the Canary Deployment

Test 1: Successful Deployment

1. **Make a code change:** Update your application to return “Version 2”.
2. **Push the change** to trigger the pipeline.
3. **Monitor CodeDeploy:**
    - The deployment will start. CodeDeploy will launch the new v2 tasks.
    - **Step 1:** Traffic shifting. The ALB listener rule will be updated to send 10% of traffic to the green target group.
    - **Step 2:** Wait for bake time. The deployment will pause for 15 minutes.
    - During this time, if you hit your application’s URL repeatedly, approximately 1 in 10 requests will be served by “Version 2”.
    - After 15 minutes (with no alarms), CodeDeploy will proceed to **Step 3**, shifting the remaining 90% of traffic.
    - Finally, the old tasks will be terminated.

Test 2: Automatic Rollback

1. **Introduce a bug:** Modify your application code to sometimes return a 500 error. For example:

```
app.get('/', (req, res) => {
  if (Math.random() > 0.5) {
    res.status(500).send('Internal Server Error!');
  } else {
    res.send('Hello from buggy Version 3');
  }
});
```

2. **Push the change** to trigger the pipeline.
3. **Generate traffic:** While the deployment is in the “baking” phase (Step 2), use a script or a tool like `curl` in a loop to send traffic to your application’s endpoint. This will generate 5xx errors.

4. **Monitor the rollback:**
    - The 5xx errors will cause the `Canary-5XX-Error-Alarm` to go into the `ALARM` state.
    - As soon as the alarm is triggered, CodeDeploy will immediately stop the deployment and initiate a rollback.
    - It will shift 100% of the traffic back to the original blue target group.
    - The deployment will be marked as **Failed** in the CodeDeploy console.
    - Your application will be safely running the last known good version (Version 2).

---

6. Best Practices

- **Meaningful Alarms:** The effectiveness of a canary release depends entirely on the quality of your monitoring. Set up alarms for key business and operational metrics, not just basic error rates. This could include latency, CPU utilization, conversion rates, or shopping cart abandonment rates.
- **Linear vs. All-at-Once:** CodeDeploy supports different traffic shifting profiles. `CodeDeployDefault.ECSCanary10Percent15Minutes` is a good starting point, but you can create custom profiles that shift traffic more gradually (e.g., 10% for 10 mins, then 25% for 10 mins, then 50%, etc.).
- **Session Stickiness:** For stateful applications, enable stickiness on your ALB target groups to ensure that a user, once routed to the canary, stays on the canary for the duration of their session.

---

7. Cleanup

1. **Delete the CodePipeline.**
2. **Delete the CodeDeploy application and deployment group.**
3. **Delete the ECS service and cluster.**
4. **Delete the Application Load Balancer and its target groups.**
5. **Delete the CloudWatch alarms.**

6. Clean up any other associated resources (ECR repo, CodeCommit repo, etc.).

Business Context

The Problem

Development teams face slow deployment cycles, manual testing bottlenecks, and security vulnerabilities introduced during the CI/CD process. Traditional deployment methods lack security scanning, leading to vulnerabilities reaching production. Manual processes create inconsistency and human error.

The Solution

Secure CI/CD pipeline with integrated security scanning, automated testing, and infrastructure as code. Implements shift-left security with SAST, DAST, and dependency scanning. Automates deployments while enforcing security gates and compliance checks at every stage.

Business Value

- **Faster Time to Market:** Automated pipelines reduce deployment time from weeks to hours
- **Improved Security Posture:** Catches vulnerabilities before production deployment
- **Reduced Manual Effort:** Eliminates 80%+ of manual deployment tasks
- **Audit Trail:** Complete deployment history and security scan results for compliance

Risk Mitigation

Prevents deployment of vulnerable code, hardcoded secrets, misconfigured infrastructure, and non-compliant resources to production environments.

GRC Mapping

Compliance Frameworks

- **NIST CSF:** PR.IP-1 (Baseline configuration), PR.DS-6 (Integrity checking), DE.CM-4 (Code analysis)
- **ISO 27001:** A.12.1 (Operational procedures), A.14.2 (Security in development), A.12.4 (Logging)
- **CIS Controls:** Control 16 (Application Software Security), Control 11 (Secure Configuration)
- **OWASP DevSecOps:** Security testing integration, Secret management, Dependency checking

Security Controls Implemented

- Static Application Security Testing (SAST)
- Dynamic Application Security Testing (DAST)
- Software Composition Analysis (SCA)
- Secrets scanning and prevention
- Infrastructure as Code security validation

Audit Evidence

- Pipeline execution logs with security scan results
- Code commit and approval records
- Security test reports (SAST/DAST/SCA)
- Deployment approval and rollback records

Regulatory Alignment

- **SOX:** Section 404 (Change management controls)
- **PCI DSS:** Requirement 6.3 (Secure development), Requirement 6.5 (Common vulnerabilities)

- **GDPR:** Article 25 (Data protection by design), Article 32 (Security of processing)
- **SOC 2:** CC8.1 (Change management), CC7.2 (System monitoring)