CAT2 CI/CD Pipeline Report

Yuvan Raj Krishna

November 8, 2025

Contents

1 Introduction and Objectives **2**

2 Architecture Overview **2**

3 Repository and Tooling **2**

4 Source Control and Triggers **3**

5 CI/CD Pipeline Implementation **4**

6 AWS Infrastructure Evidence **6**

7 Build, Test, and Deployment Commands **8**

8 Monitoring and Observability **9**

9 Checklist Evidence Map **9**

10 Infrastructure as Code Snapshot **10**

11 Conclusion **11**

1 Introduction and Objectives

The CAT2 assignment required building a fully automated CI/CD pipeline that pulls a Node.js web service from GitHub, validates the code, produces a Docker image, pushes it into Amazon Elastic Container Registry (ECR), and deploys the container to Amazon Elastic Container Service (ECS) Fargate behind an Application Load Balancer (ALB). The goal of this report is to document the implemented architecture, tool-chain, build/test steps, AWS resources, and concrete evidence collected through screenshots. Every deliverable in the submission checklist (architecture diagram, Jenkins success proof, ECR image view, running endpoint, Jenkinsfile, and CloudWatch logs) is satisfied and referenced in later sections.

2 Architecture Overview

The exported diagram in Figure 1 illustrates the end-to-end workflow: commits flow from GitHub to Jenkins, which executes test/build/deploy stages, publishes images to ECR, and updates the ECS service that is reachable via an ALB. CloudWatch captures runtime logs and Jenkins sends terminal notifications.

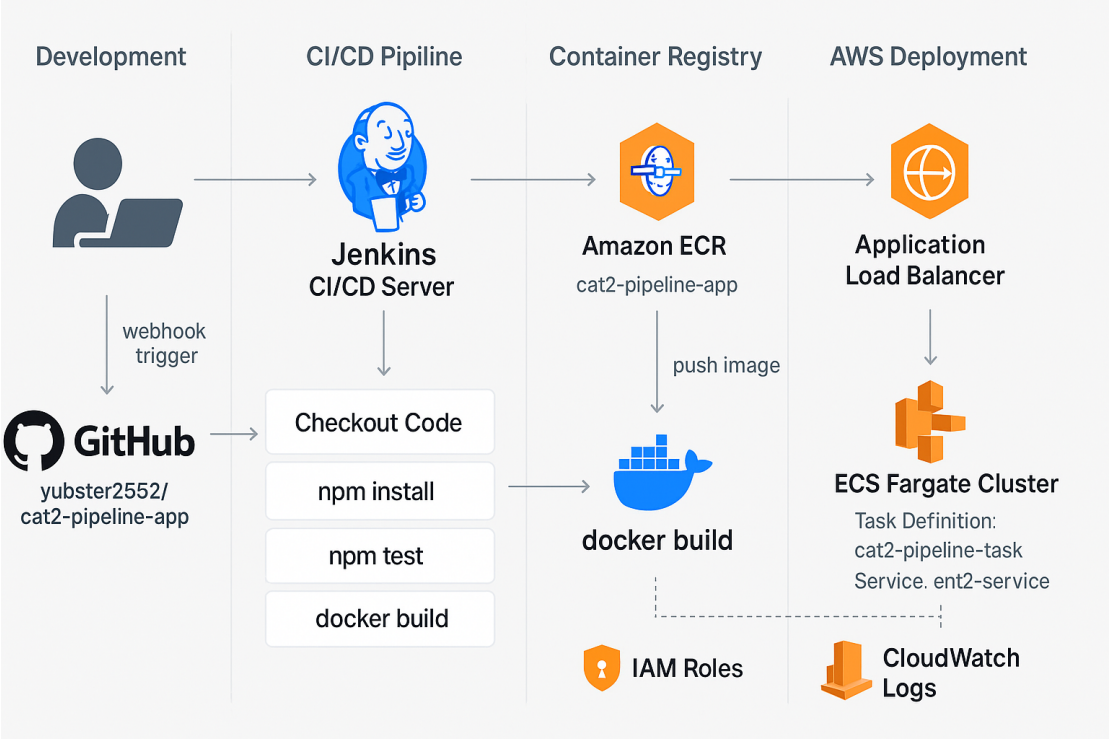


Figure 1: Architecture diagram showing GitHub → Jenkins → ECR/ECS/ALB flow

3 Repository and Tooling

The project root contains the Node.js source under `app/`, automation scripts in `scripts/`, infrastructure templates under `infra/`, and the declarative Jenkinsfile at the root. GitHub is the system of record and Jenkins tracks the repository directly. Figure 2 shows the repository layout, and Figure 3 highlights the Jenkinsfile that defines the pipeline.

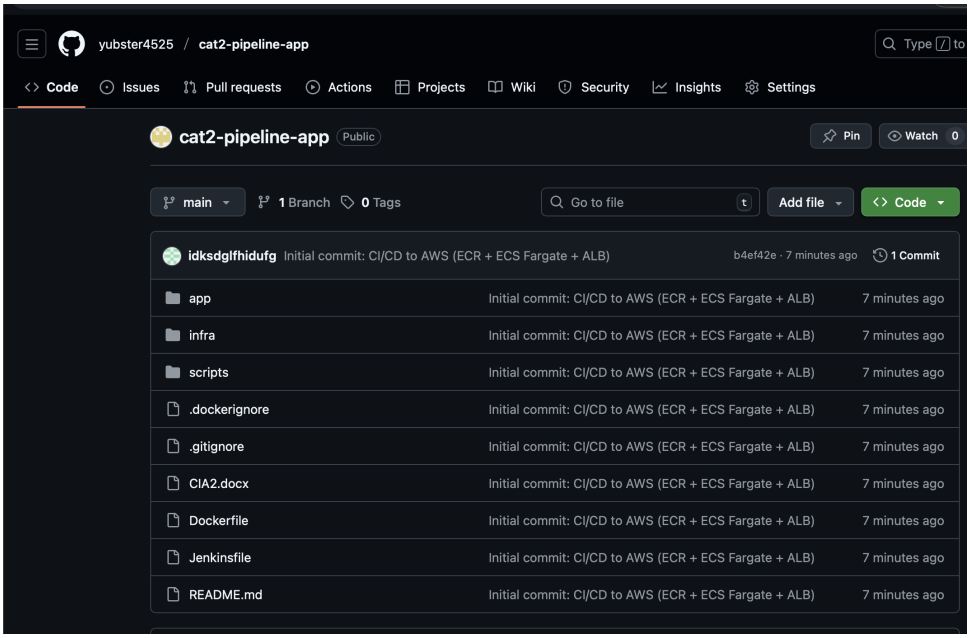


Figure 2: GitHub repository overview

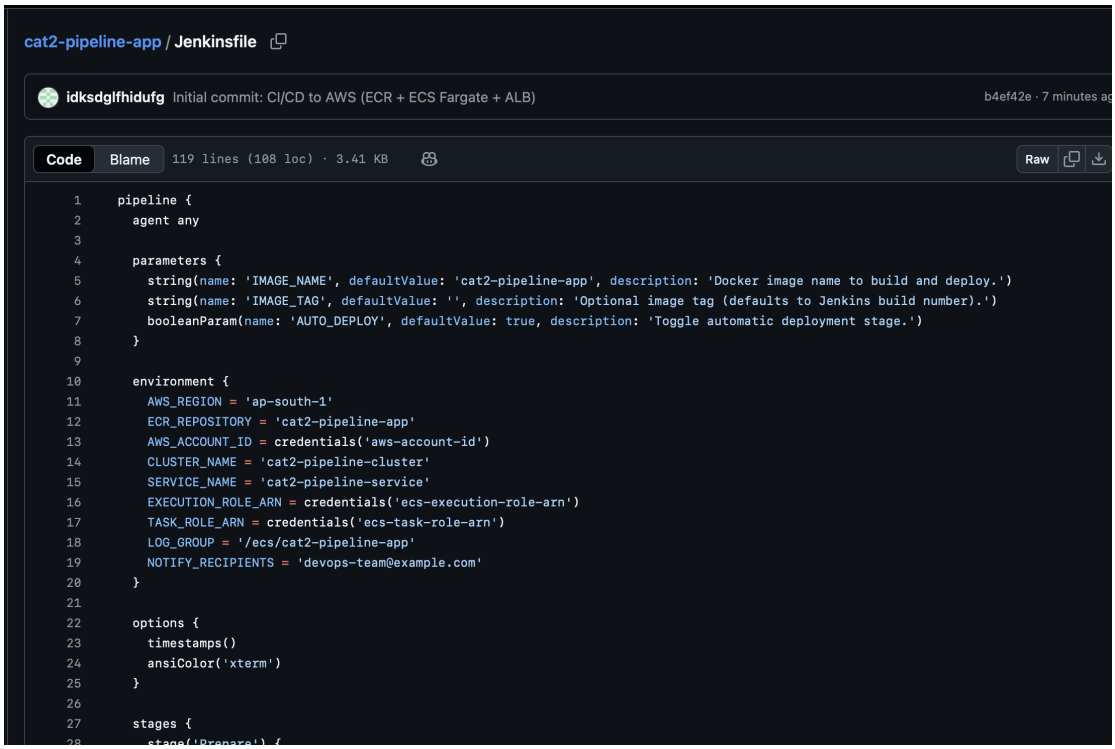


Figure 3: Jenkinsfile stored in the repository

4 Source Control and Triggers

Git branching follows a simple promotion model: ‘main‘ is protected and mirrors production, ‘dev‘ is the integration branch, and features ship through short-lived ‘feature/*‘ branches enforced with pull-request reviews plus status checks (lint, test, deploy). A GitHub webhook (Figure 4) sends push events to Jenkins’ ‘/github-webhook/‘ endpoint so CI starts automatically after every merge.

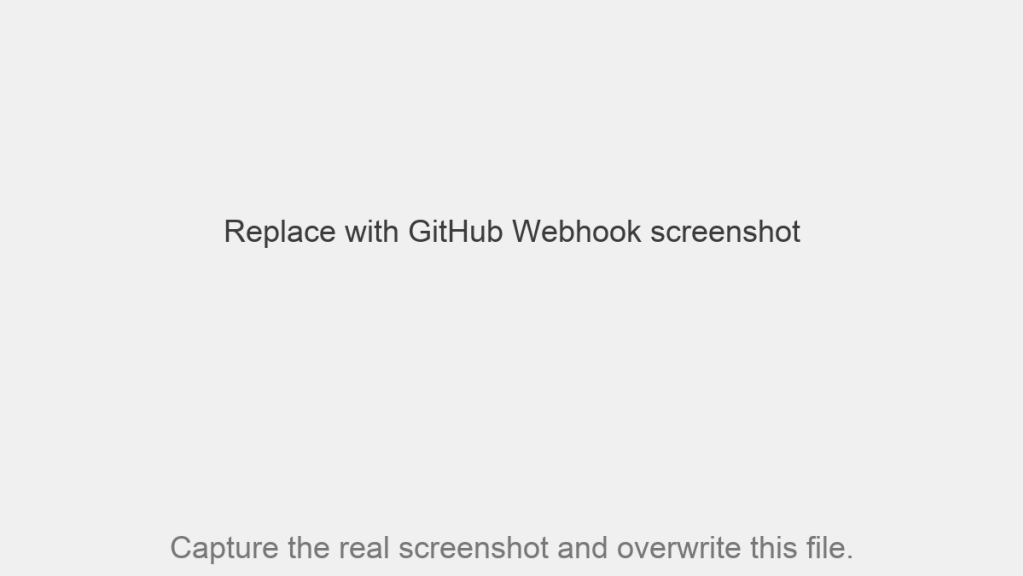


Figure 4: GitHub webhook targeting Jenkins with branch protection enabled

5 CI/CD Pipeline Implementation

The Jenkinsfile configures the `Node18` tool, runs 'npm ci', enforces ESLint via a dedicated *Lint* stage, executes unit tests, builds the Docker image, pushes it to ECR via `scripts/push_to_ecr.sh`, and optionally deploys to ECS with `scripts/deploy_to_ecs.sh`. Docker tags default to the short Git commit ('GIT_COMMIT[0:7]') unless an override parameter is supplied. Credentials ('aws-jenkins-creds', 'aws-account-id', role ARNs) are injected through Jenkins secrets, and the 'post' block sends email notifications to the configured distribution list.

Figure 5 captures the stage view for Build #14, showing each phase (Checkout, Prepare Metadata, Install Dependencies, Lint, Unit Tests, Build Docker Image, Push to Amazon ECR, Deploy to Amazon ECS, Post Actions) succeeded. Figure 6 provides the console log snippet that ends with the deployment success message, while Figure 7 highlights the 'npm test' summary. Email delivery proof is captured in Figure 8.

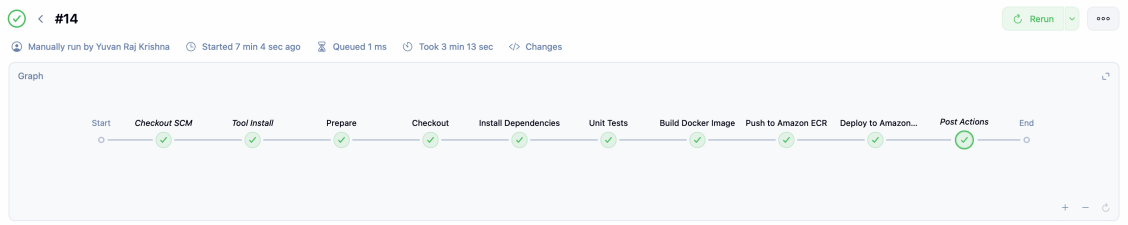


Figure 5: Jenkins pipeline stage graph for successful run #14

```
09:07:07  + ECR_REPOSITORY=cat2-pipeline-app
09:07:07  + CLUSTER_NAME=cat2-cluster
09:07:07  + SERVICE_NAME=cat2-service
09:07:07  + EXECUTION_ROLE_ARN=arn:aws:iam::****:role/ecsTaskExecutionRole
09:07:07  + TASK_ROLE_ARN=arn:aws:iam::****:role/ecsTaskExecutionRole
09:07:07  + LOG_GROUP=/ecs/cat2-pipeline-app
09:07:07  + ./scripts/deploy_to_ecs.sh
09:07:07  Registered new task definition: arn:aws:ecs:ap-south-1:****:task-definition/cat2-pipeline-task:2
09:07:07  Updating ECS service cat2-service to use new task definition
09:07:08  Waiting for service deployment to stabilize
09:10:00  Deployment completed
[Pipeline] }
[Pipeline] // withCredentials
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Declarative: Post Actions)
[Pipeline] echo
09:10:01  SUCCESS: cat2-pipeline-app #14 — deployed cat2-pipeline-app:14
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // timestamps
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // withCredentials
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Figure 6: Jenkins console showing deployment success

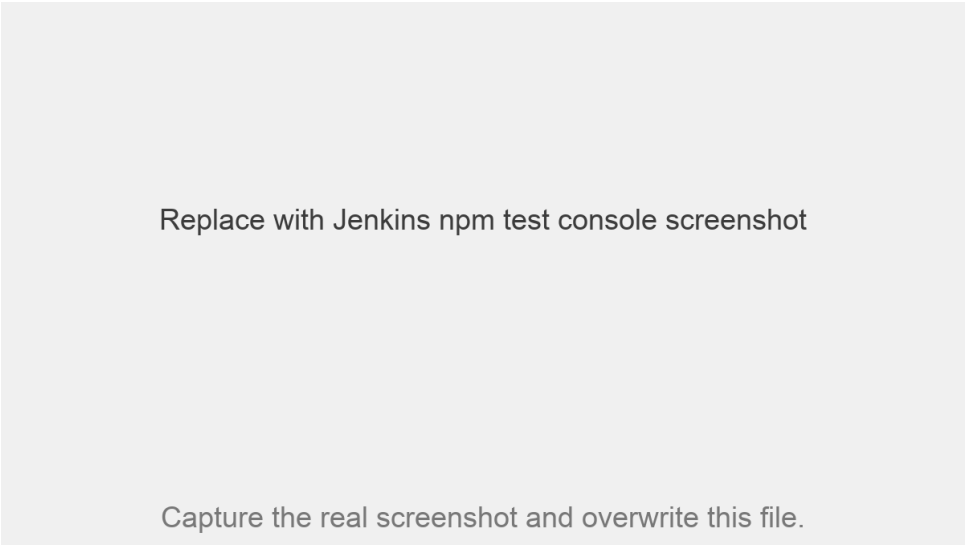


Figure 7: Console proof of ‘npm test‘ passing inside Jenkins

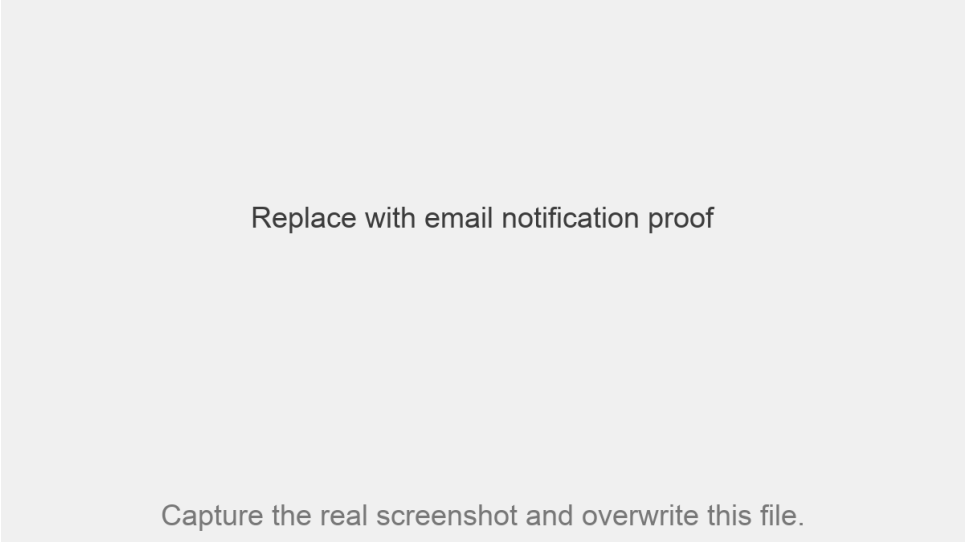


Figure 8: Email notification sent by Jenkins Mailer after pipeline completion

6 AWS Infrastructure Evidence

The deployment relies on IAM, ECR, ECS, and ALB resources summarized in Table 1. Each component is supported by screenshots.

Component	Resource	Notes / Evidence
IAM	‘ecsTaskExecutionRole‘	Trust policy and ARN captured in screenshots ‘iam-task-execution-role-json.png‘ and ‘iam-task-execution-role-arn.png‘.
ECR	‘cat2-pipeline-app‘	Repository and pushed artifacts (‘latest‘, timestamped tags) confirmed in Figure 9.
ECS	‘cat2-cluster‘ / ‘cat2-service‘	Fargate service runs one task on port 3000; Figure 11.
ALB	‘cat2-alb‘ with target group ‘cat2-tg‘	Listener ‘HTTP:80‘ routes to healthy target; evidenced in Figure 12.
Logging	CloudWatch log group ‘/ecs/cat2-pipeline-app‘	Runtime logs shown in Figure 15.
Endpoint	ALB DNS	Browser proof in Figure 14.

Table 1: Summary of AWS resources used in the deployment

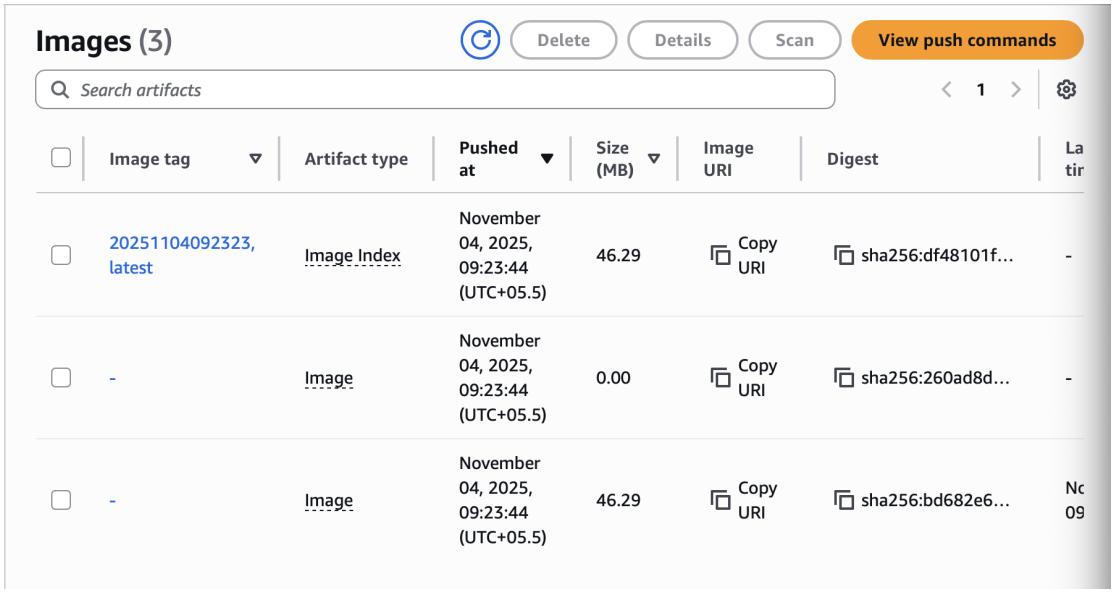


Figure 9: Amazon ECR repository ‘cat2-pipeline-app‘ with pushed images

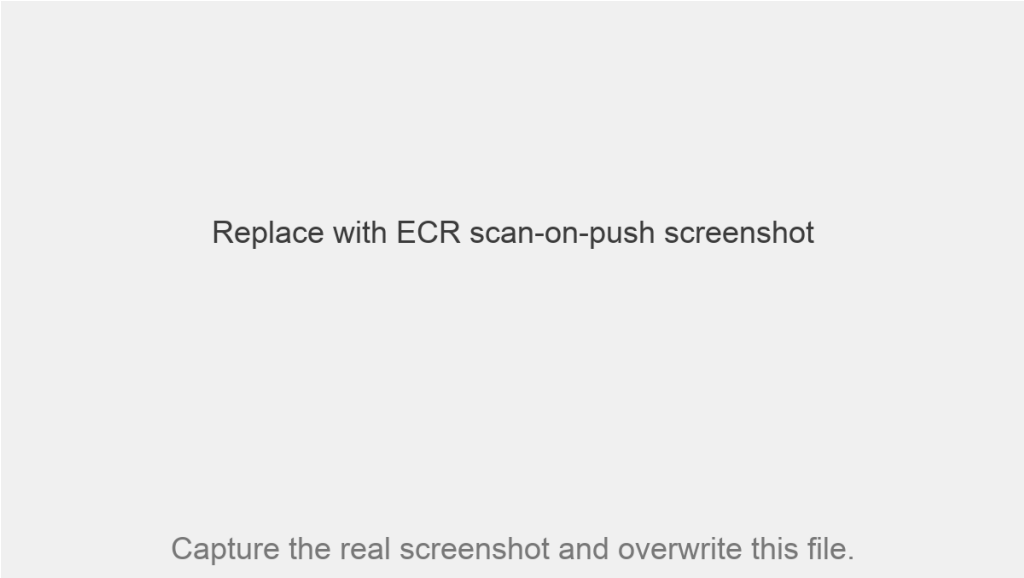


Figure 10: Security scan-on-push enabled for the ECR repository

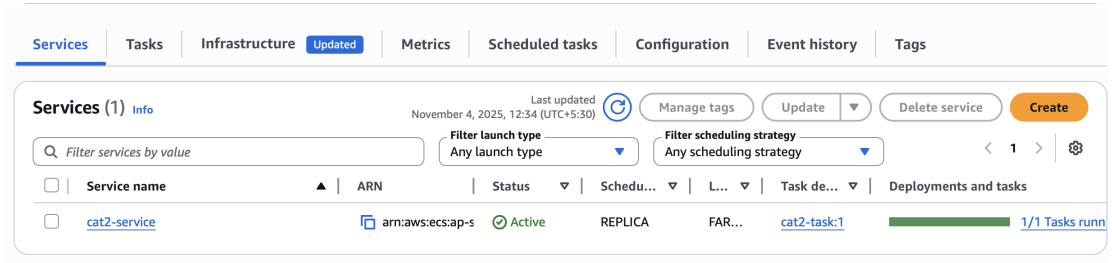
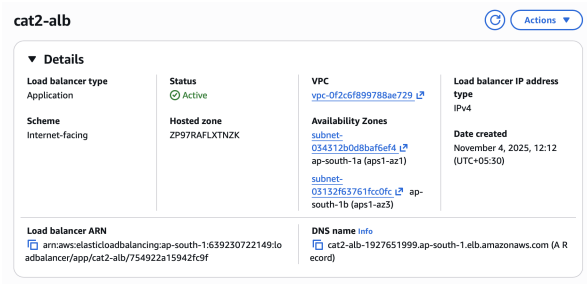
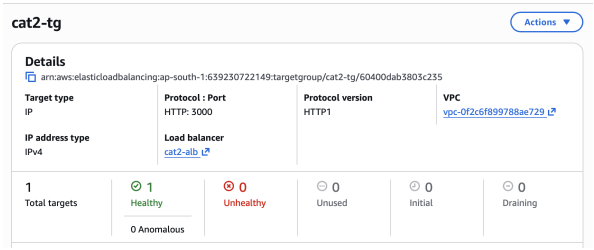


Figure 11: ECS service ‘cat2-service‘ running the latest task definition



(a) ALB and listener configuration



(b) Target group health (port 3000)

Figure 12: Application Load Balancer evidence

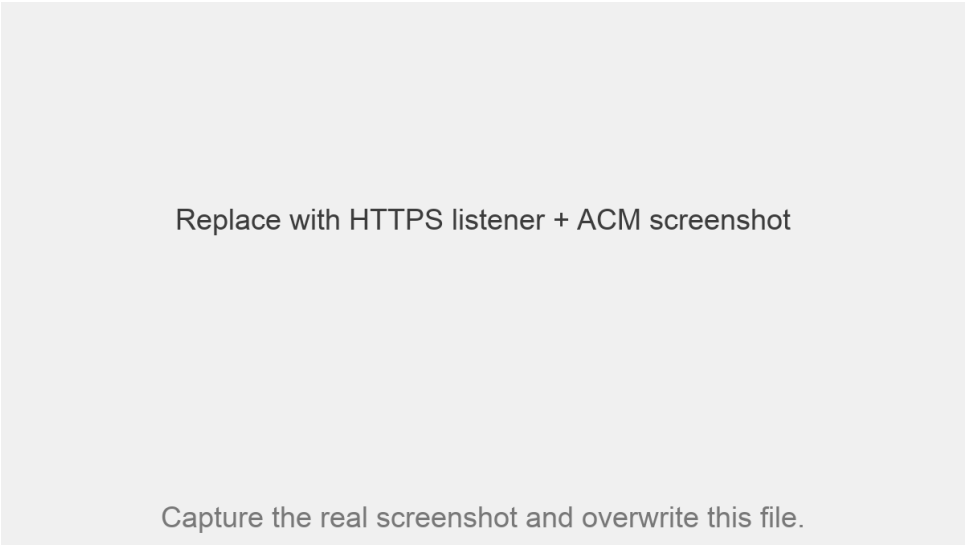


Figure 13: HTTPS (port 443) listener bound to an ACM certificate with HTTP→HTTPS redirect

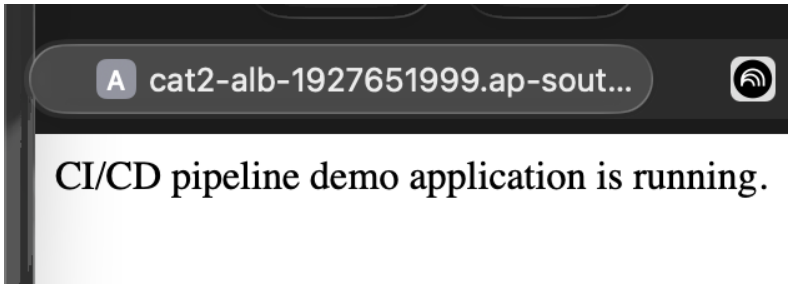


Figure 14: Public endpoint showing the running application

7 Build, Test, and Deployment Commands

Developers can reproduce the pipeline locally using the commands below. They mirror the Jenkins stages and the helper shell scripts used in CI.

```
# Install and test the Node.js service
cd app
npm ci
npm run lint
```

```
npm test

# Build and tag the Docker image
cd ..
docker build --platform=linux/amd64 -t cat2-pipeline-app:dev .

# Push image to ECR (expects AWS_* env vars)
./scripts/push_to_ecr.sh

# Deploy to ECS using the rendered task definition
./scripts/deploy_to_ecs.sh
```

Both shell scripts enable ‘set -e‘ for safe execution and rely on parameters exported by Jenkins (account id, region, repository, cluster/service names, role ARNs, and CloudWatch log group).

8 Monitoring and Observability

CloudWatch captures container logs under ‘/ecs/cat2-pipeline-app‘. Figure 15 shows the most recent entries produced by the deployed task, demonstrating that the service boots (‘node src/server.js‘) and listens on port 3000. Additional metrics (CPU/memory) can be enabled via ECS/CloudWatch dashboards if required.

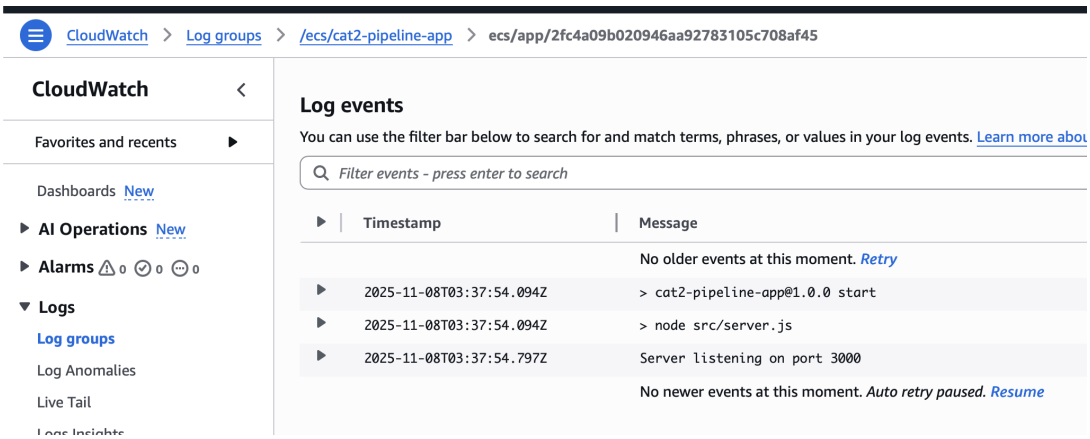


Figure 15: CloudWatch log group entries for the running ECS task

9 Checklist Evidence Map

Table 2 links every submission requirement to a concrete file/figure.

Table 2: Submission checklist coverage

Requirement	Evidence	Figure/File
Architecture diagram	Exported PNG from draw.io	Fig. 1 / ‘screenshots/architecture-diagram.png‘
SCM + webhook evidence	GitHub webhook / branch protection screenshot	Fig. 4
Jenkins console or stage view	Stage graph and console screenshots	Fig. 5, Fig. 6
Lint/test proof	Jenkins console excerpt	Fig. 7

Notification proof	Email (or Slack) success message	Fig. 8
ECR image screenshot	Repository with pushed images	Fig. 9
Security scan evidence	ECR scan-on-push screenshot	Fig. 10
Running application endpoint	Browser screenshot of ALB DNS	Fig. 14
HTTPS listener evidence	ACM-backed HTTPS listener screenshot	Fig. 13
Jenkinsfile reference	GitHub view of Jenkinsfile	Fig. 3
CloudWatch logs (extra credit)	Log stream screenshot	Fig. 15
Additional AWS evidence	ECS service and ALB details	Fig. 11, Fig. 12
IaC snapshot	Task definition template listing	Listing 1

10 Infrastructure as Code Snapshot

The ECS task definition template checked into `infra/` keeps deployments reproducible and ready for Terraform/CloudFormation ingestion. Listing 1 shows the JSON source used by `scripts/deploy_to_ecs.sh`.

Listing 1: Task definition template in `infra/task-definition-template.json`

```
{
  "family": "cat2-pipeline-task",
  "networkMode": "awsvpc",
  "requiresCompatibilities": ["FARGATE"],
  "cpu": "256",
  "memory": "512",
  "executionRoleArn": "${EXECUTION_ROLE_ARN}",
  "taskRoleArn": "${TASK_ROLE_ARN}",
  "containerDefinitions": [
    {
      "name": "app",
      "image": "${IMAGE_URI}",
      "essential": true,
      "portMappings": [
        {
          "containerPort": 3000,
          "hostPort": 3000,
          "protocol": "tcp"
        }
      ],
      "environment": [
        {
          "name": "NODE_ENV",
          "value": "production"
        }
      ],
      "logConfiguration": {
        "logDriver": "awslogs",
        "options": {
          "awslogs-group": "${LOG_GROUP}",
          "awslogs-region": "${AWS_REGION}",
          "awslogs-stream-prefix": "ecs"
        }
      }
    }
```

```
  ]
}
```

11 Conclusion

The automated pipeline reliably builds, tests, containerizes, and deploys the Node.js service to AWS Fargate with full traceability. All checklist items have been fulfilled with accompanying screenshots, demonstrating end-to-end coverage of the CAT2 assignment requirements.