Sardar Noor Ul Hassan
Cloud Intern

Contents

1. Introduction 2
2. Objectives 2
3. Architecture Overview 3
4. Application & Dockerization 4
    Application Structure 4
    Dockerfile 4
5. IAM Role & OIDC Configuration 5
    Why OIDC? 5
Steps Performed 5
    1. Created OIDC Identity Provider: 5
    2.Created IAM Role: 5
    3.Attached policies: 6
6. Amazon ECR Repository 7
    Repository Configuration 7
7. ECS Fargate Setup 7
    ECS Cluster 7
    Task Definition 7
    Service Configuration 8
    GitHub Actions CI Workflow 9
9. CD Pipeline (Deploy to ECS) 11
    Deployment Workflow 11
10. Verification & Rollback 15
    Verification 15
    Rollback Strategies 15

Option 1 – Task Definition Rollback 15
Option 2 – Image SHA Rollback 15
11. Issues Faced & Resolutions 15
Issue 1: GitHub Actions authentication failure 15
Issue 2: Port mapping conflict 16
12. Conclusion 16

CI/CD Pipeline for Dockerized Application on AWS ECS Fargate using GitHub Actions (OIDC)

1. Introduction

This project implements a complete CI/CD pipeline using GitHub Actions and AWS ECS Fargate, following practices such as:

1. Containerized application delivery
2. Infrastructure managed via AWS Console
3. Secure authentication using OIDC (OpenID Connect)
4. No long-term AWS credentials
5. Automated deployment with rollback capability

The pipeline ensures that every code change pushed to GitHub is automatically built, tested, containerized, and deployed to AWS.

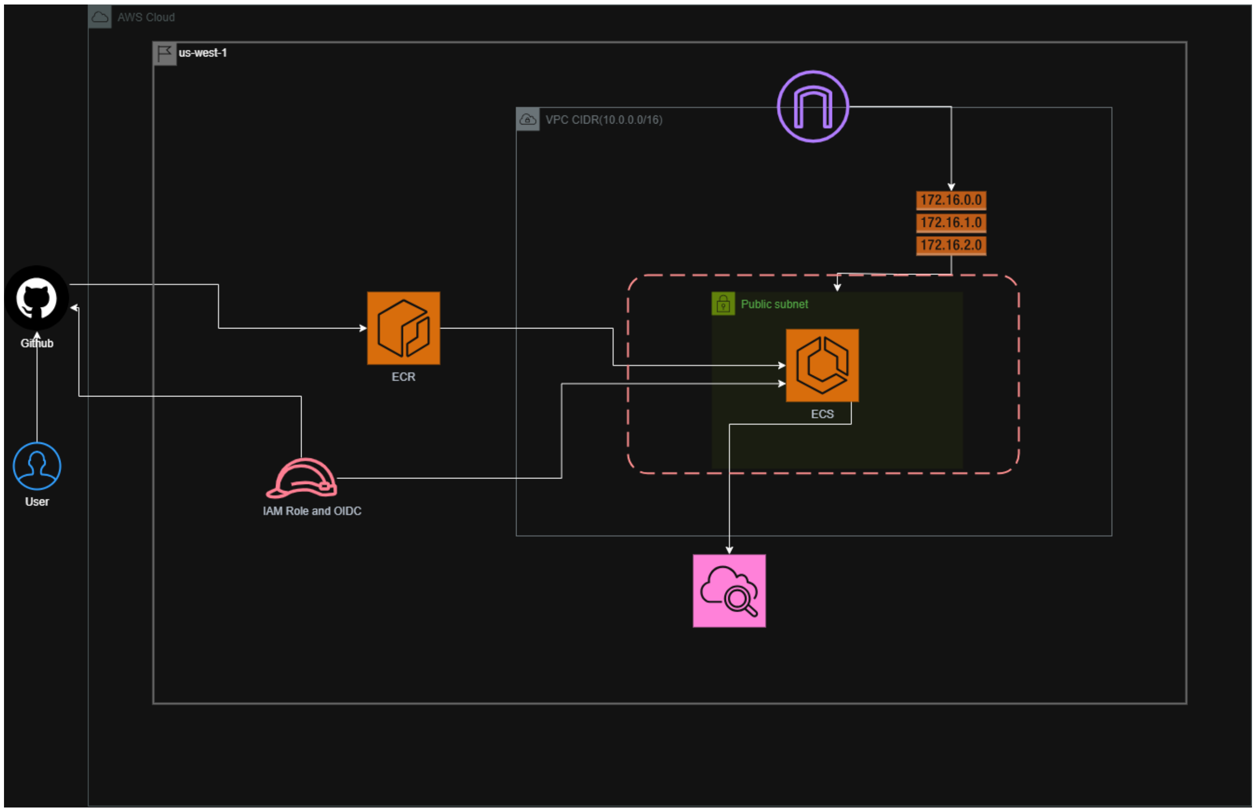
2. Objectives

The primary objectives of this project are:

1. To containerize a Node.js web application using Docker
2. To store container images securely in Amazon ECR
3. To deploy containers using Amazon ECS Fargate (serverless)
4. To implement CI/CD pipelines using GitHub Actions
5. To use IAM Role + OIDC for secure authentication

6. To verify deployments and enable rollback mechanisms

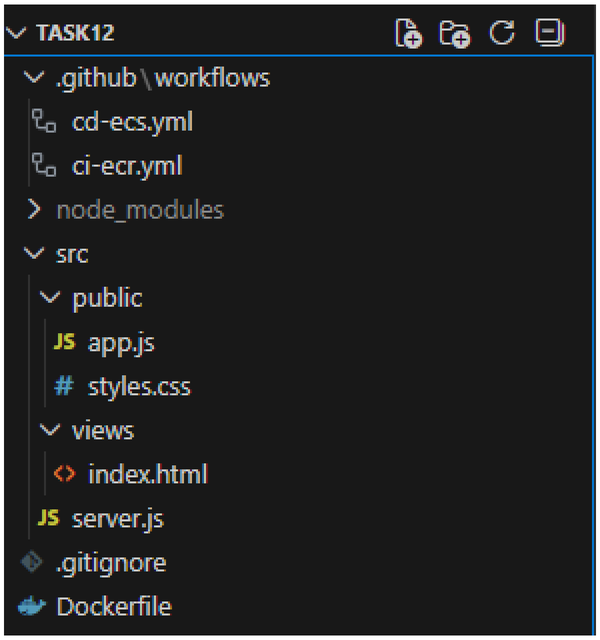
3. Architecture Overview



1. Developer pushes code to GitHub repository
2. GitHub Actions CI pipeline builds Docker image
3. Image is pushed to Amazon ECR
4. GitHub Actions CD pipeline updates ECS task definition
5. ECS service deploys new task revision
6. Application runs on ECS Fargate and is accessible via public IP

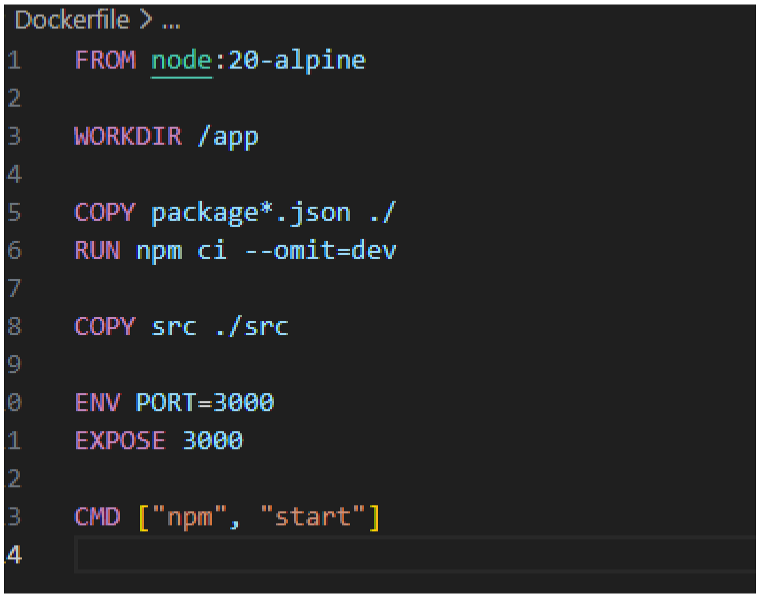
4. Application & Dockerization

Application Structure



```
TASK12
  .github\workflows
    cd-ecs.yml
    ci-ecr.yml
  node_modules
  src
    public
      app.js
      styles.css
    views
      index.html
    server.js
  .gitignore
  Dockerfile
```

Dockerfile



Dockerfile > ...

```dockerfile
FROM node:20-alpine

WORKDIR /app

COPY package*.json ./
RUN npm ci --omit=dev

COPY src ./src

ENV PORT=3000
EXPOSE 3000

CMD ["npm", "start"]

```

The Dockerfile:

1. Uses a lightweight Node.js base image
2. Copies application source code

3. Installs dependencies
4. Exposes application port
5. Starts the application

5. IAM Role & OIDC Configuration

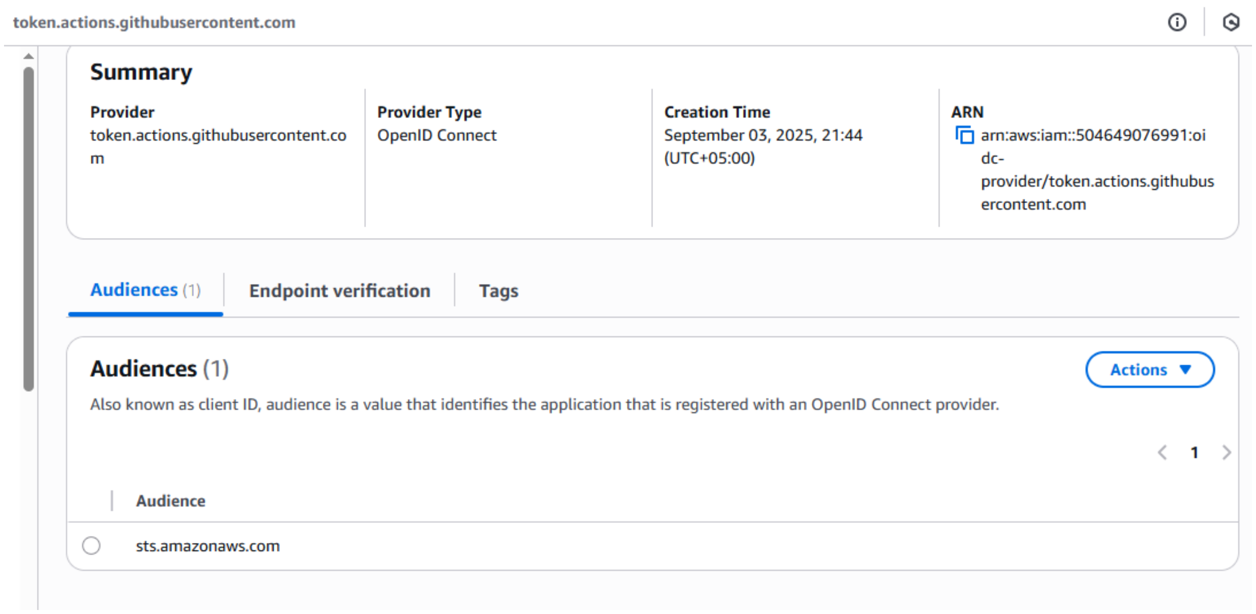
Why OIDC?

1. Eliminates long-term AWS access keys
2. Uses short-lived credentials
3. GitHub Actions assumes IAM role securely

Steps Performed

1. Created OIDC Identity Provider:

- token.actions.githubusercontent.com



token.actions.githubusercontent.com

Summary

Provider	Provider Type	Creation Time	ARN
token.actions.githubusercontent.com	OpenID Connect	September 03, 2025, 21:44 (UTC+05:00)	arn:aws:iam::504649076991:oidc-provider/token.actions.githubusercontent.com

Audiences (1)

Also known as client ID, audience is a value that identifies the application that is registered with an OpenID Connect provider.

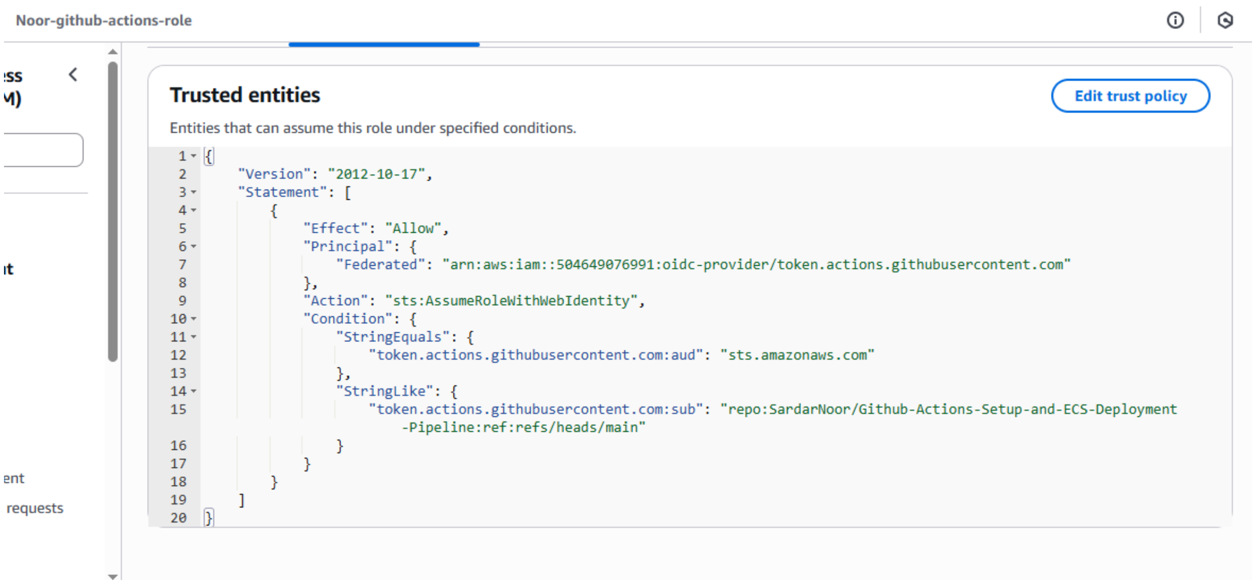
Audience
sts.amazonaws.com

2.Created IAM Role:

Trust relationship restricted to:

1. GitHub user

2. Repository
3. Branch (main)

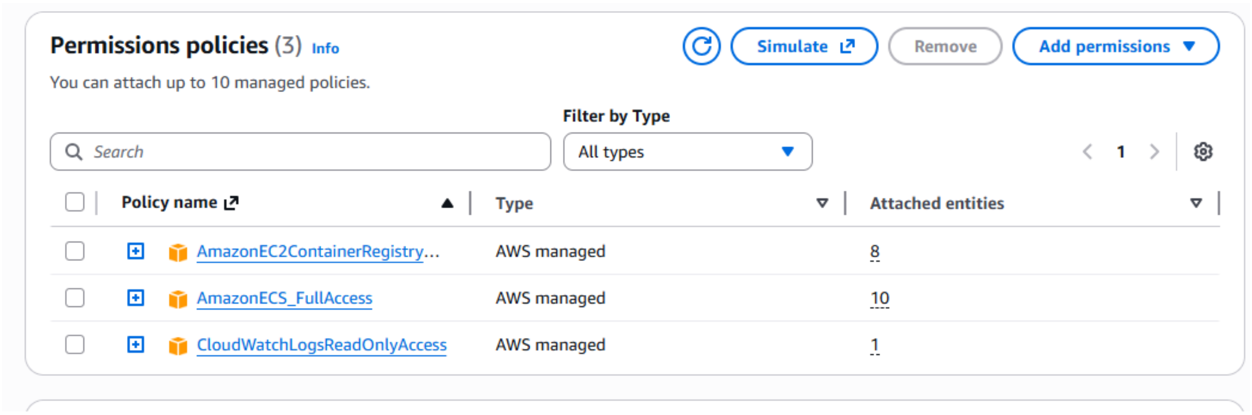


Entities that can assume this role under specified conditions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::504649076991:oidc-provider/token.actions.githubusercontent.com"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "token.actions.githubusercontent.com:aud": "sts.amazonaws.com"
                },
                "StringLike": {
                    "token.actions.githubusercontent.com:sub": "repo:SardarNoor/Github-Actions-Setup-and-ECS-Deployment
                        -Pipeline:ref:refs/heads/main"
                }
            }
        }
    ]
}
```

3.Attached policies:

1. AmazonEC2ContainerRegistryPowerUser
2. AmazonECS_FullAccess
3. CloudWatchLogsReadOnlyAccess



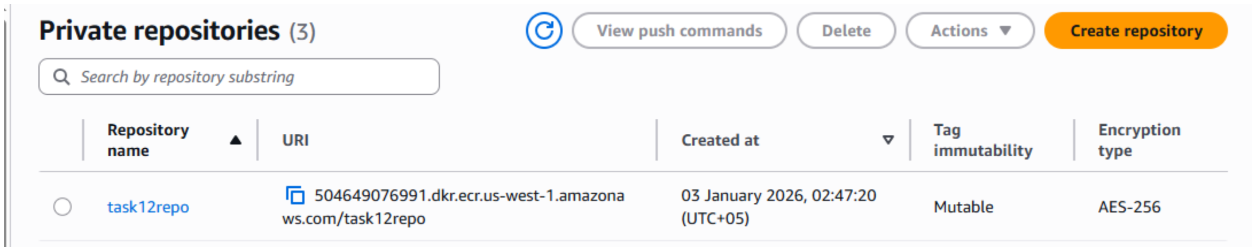
You can attach up to 10 managed policies.

Policy name	Type	Attached entities
AmazonEC2ContainerRegistry...	AWS managed	8
AmazonECS_FullAccess	AWS managed	10
CloudWatchLogsReadOnlyAccess	AWS managed	1

6. Amazon ECR Repository

Repository Configuration

1. Repository name: task12repo
2. Image tag mutability: **Mutable**
3. Encryption: AES-256 (default)



Private repositories (3)

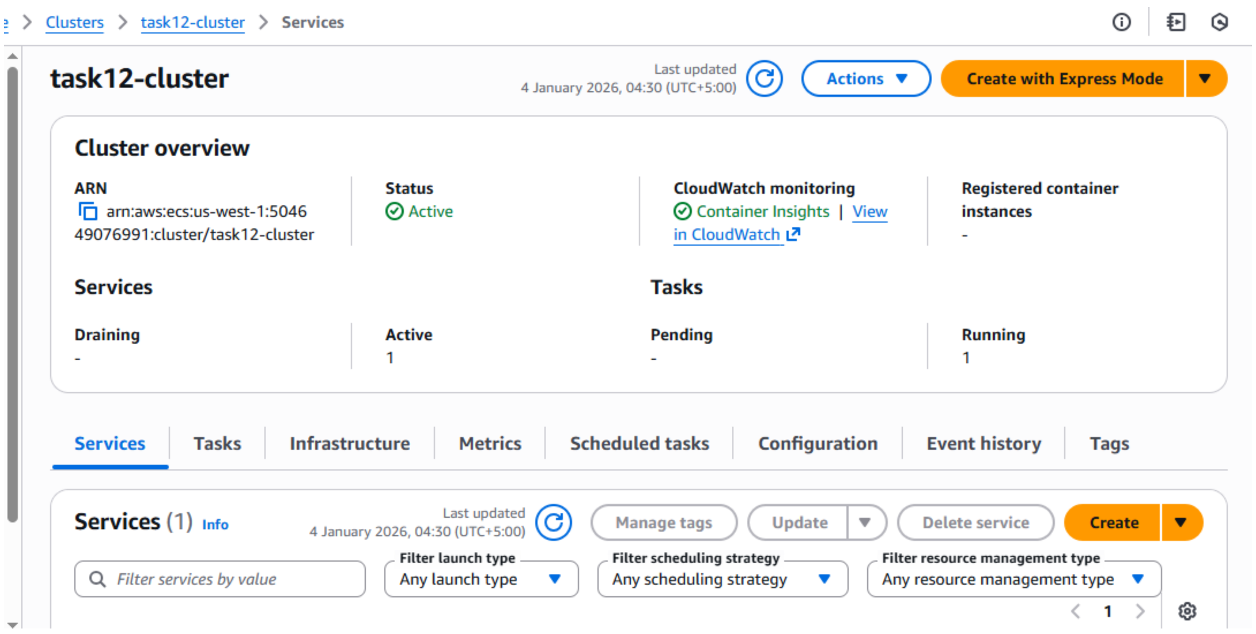
Search by repository substring

Repository name	URI	Created at	Tag immutability	Encryption type
task12repo	504649076991.dkr.ecr.us-west-1.amazonaws.com/task12repo	03 January 2026, 02:47:20 (UTC+05)	Mutable	AES-256

7. ECS Fargate Setup

ECS Cluster

1. Cluster name: task12-cluster
2. Launch type: **Fargate**



task12-cluster

Last updated 4 January 2026, 04:30 (UTC+5:00)

Cluster overview

ARN: arn:aws:ecs:us-west-1:504649076991:cluster/task12-cluster

Status: Active

CloudWatch monitoring: Container Insights | View in CloudWatch

Registered container instances: -

Services

Draining: -

Active: 1

Tasks

Pending: -

Running: 1

Services (1)

Last updated 4 January 2026, 04:30 (UTC+5:00)

Filter services by value

Filter launch type: Any launch type

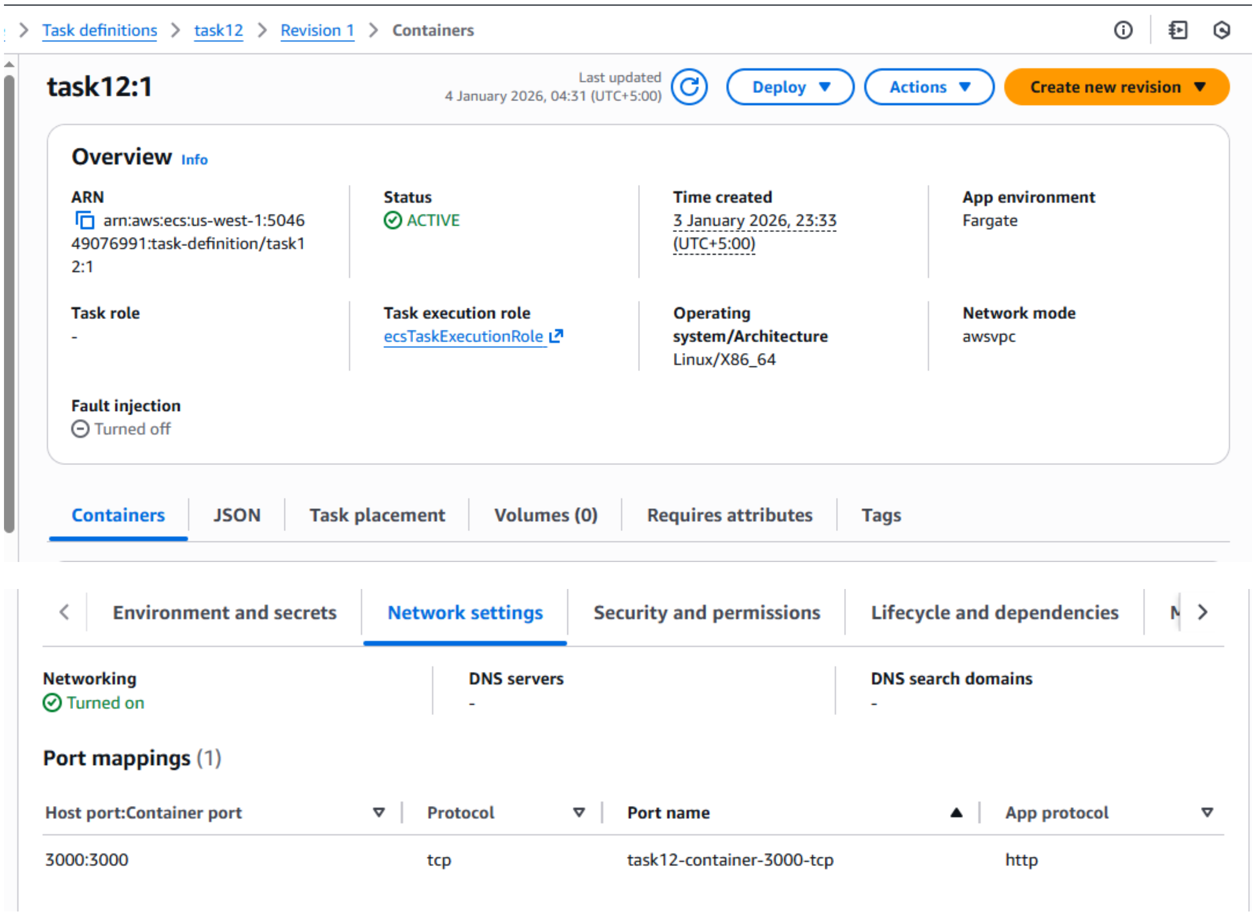
Filter scheduling strategy: Any scheduling strategy

Filter resource management type: Any resource management type

Task Definition

1. CPU: 0.25 vCPU

2. Memory: 0.5 GB
3. Network mode: awsvpc
4. Container port: 3000
5. Logging: CloudWatch Logs enabled



Task definitions > task12 > Revision 1 > Containers

task12:1

Last updated 4 January 2026, 04:31 (UTC+5:00)

Overview Info

ARN arn:aws:ecs:us-west-1:504649076991:task-definition/task12:1	**Status** ACTIVE	**Time created** 3 January 2026, 23:33 (UTC+5:00)	**App environment** Fargate
Task role -	**Task execution role** ecsTaskExecutionRole	**Operating system/Architecture** Linux/X86_64	**Network mode** awsvpc
Fault injection Turned off			

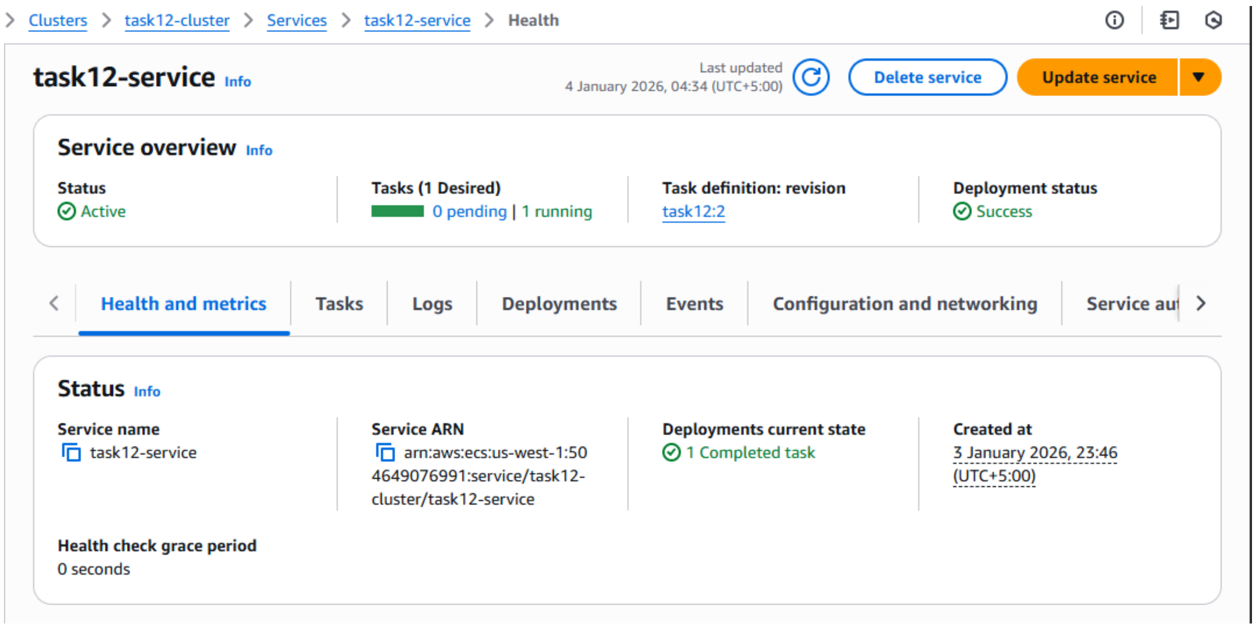
Networking	**DNS servers**	**DNS search domains**
Turned on	-	-

Port mappings (1)

Host port:Container port	Protocol	Port name	App protocol
3000:3000	tcp	task12-container-3000-tcp	http

Service Configuration

1. Desired tasks: 1
2. Capacity provider: FARGATE
3. Public subnets
4. Auto-assign public IP: **Enabled**


> Clusters > task12-cluster > Services > task12-service > Health

task12-service Info

Last updated 4 January 2026, 04:34 (UTC+5:00)

Delete service | Update service

Service overview Info

Status	Tasks (1 Desired)	Task definition: revision	Deployment status
Active	0 pending | 1 running	task12:2	Success

Health and metrics | Tasks | Logs | Deployments | Events | Configuration and networking | Service au

Status Info

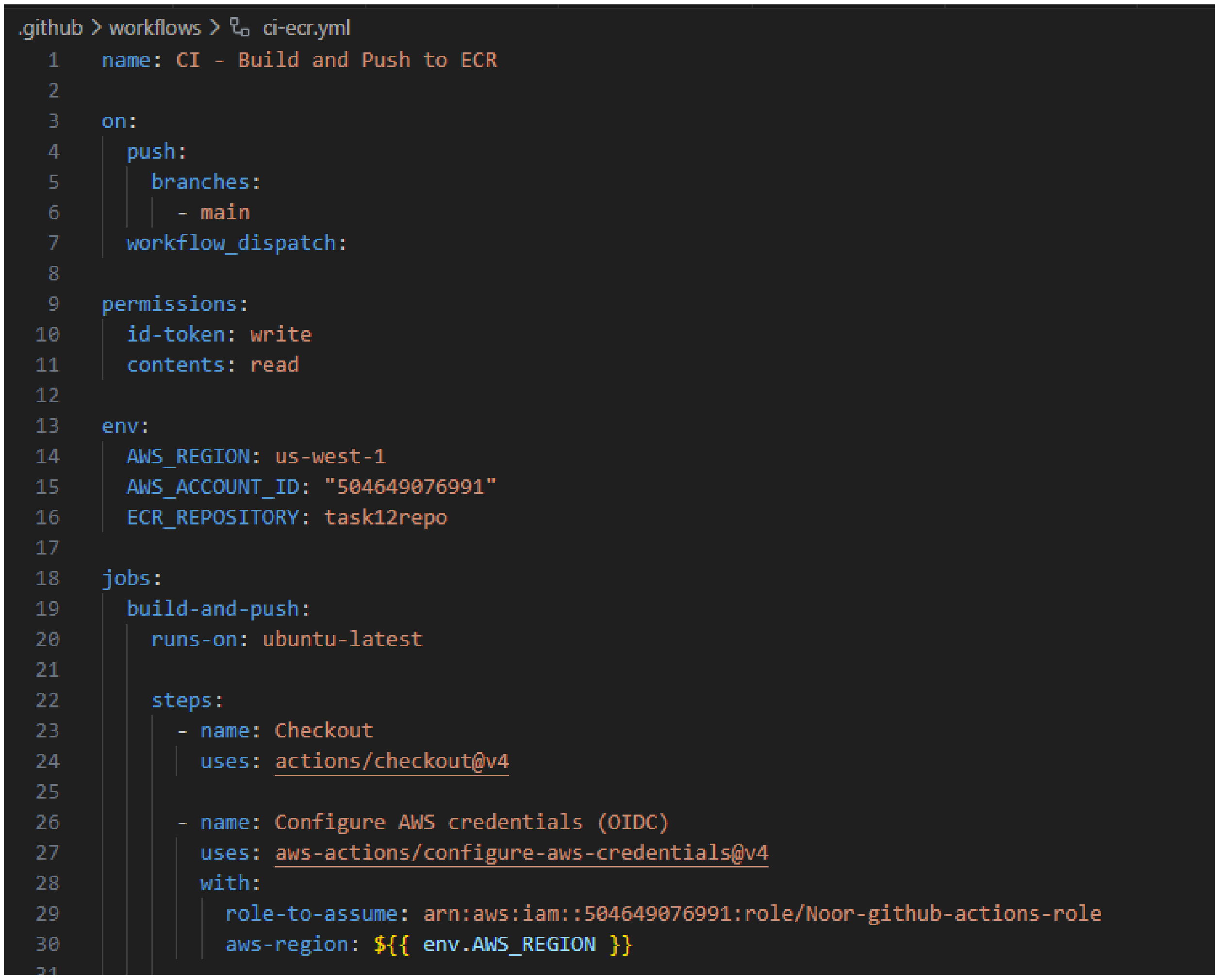
Service name	Service ARN	Deployments current state	Created at
task12-service	arn:aws:ecs:us-west-1:504649076991:service/task12-cluster/task12-service	1 Completed task	3 January 2026, 23:46 (UTC+5:00)

Health check grace period
0 seconds

8. CI Pipeline (Build & Push to ECR)

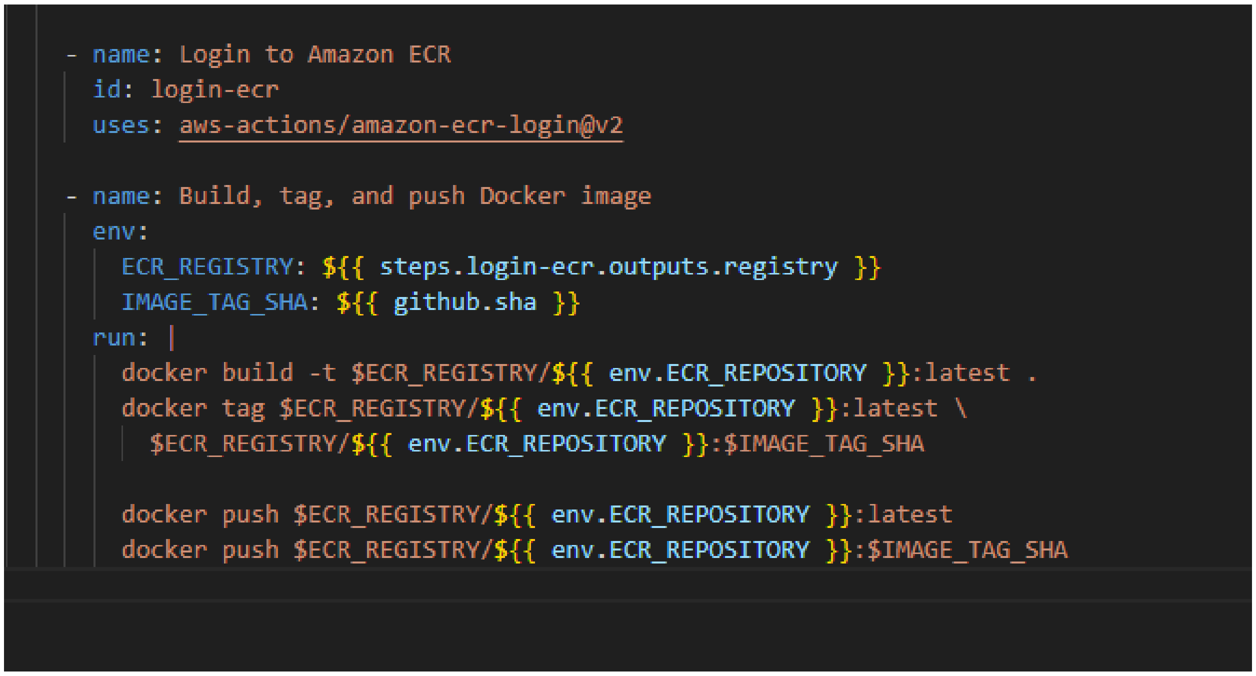
GitHub Actions CI Workflow

1. Trigger: push to main
2. Authenticate using OIDC
3. Build Docker image
4. Tag image
5. Push to ECR

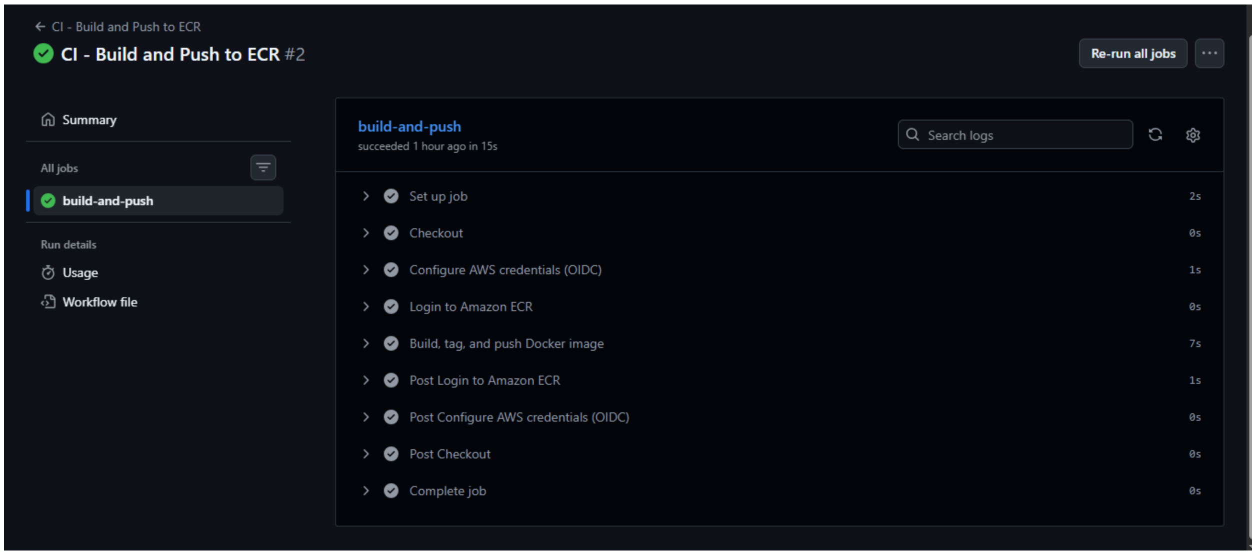


.github > workflows > ci-ecr.yml

```yaml
name: CI - Build and Push to ECR

on:
  push:
    branches:
      - main
  workflow_dispatch:

permissions:
  id-token: write
  contents: read

env:
  AWS_REGION: us-west-1
  AWS_ACCOUNT_ID: "504649076991"
  ECR_REPOSITORY: task12repo

jobs:
  build-and-push:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout
        uses: actions/checkout@v4

      - name: Configure AWS credentials (OIDC)
        uses: aws-actions/configure-aws-credentials@v4
        with:
          role-to-assume: arn:aws:iam::504649076991:role/Noor-github-actions-role
          aws-region: ${{ env.AWS_REGION }}
```



```yaml
- name: Login to Amazon ECR
  id: login-ecr
  uses: aws-actions/amazon-ecr-login@v2

- name: Build, tag, and push Docker image
  env:
    ECR_REGISTRY: ${{ steps.login-ecr.outputs.registry }}
    IMAGE_TAG_SHA: ${{ github.sha }}
  run: |
    docker build -t $ECR_REGISTRY/${{ env.ECR_REPOSITORY }}:latest .
    docker tag $ECR_REGISTRY/${{ env.ECR_REPOSITORY }}:latest \
      $ECR_REGISTRY/${{ env.ECR_REPOSITORY }}:$IMAGE_TAG_SHA

    docker push $ECR_REGISTRY/${{ env.ECR_REPOSITORY }}:latest
    docker push $ECR_REGISTRY/${{ env.ECR_REPOSITORY }}:$IMAGE_TAG_SHA
```



← CI - Build and Push to ECR

CI - Build and Push to ECR #2

Re-run all jobs

Summary

All jobs

build-and-push

Run details

Usage

Workflow file

build-and-push
succeeded 1 hour ago in 15s

Step	Duration
Set up job	2s
Checkout	0s
Configure AWS credentials (OIDC)	1s
Login to Amazon ECR	0s
Build, tag, and push Docker image	7s
Post Login to Amazon ECR	1s
Post Configure AWS credentials (OIDC)	0s
Post Checkout	0s
Complete job	0s

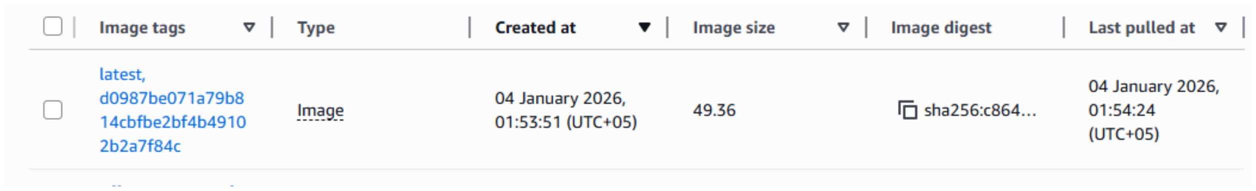
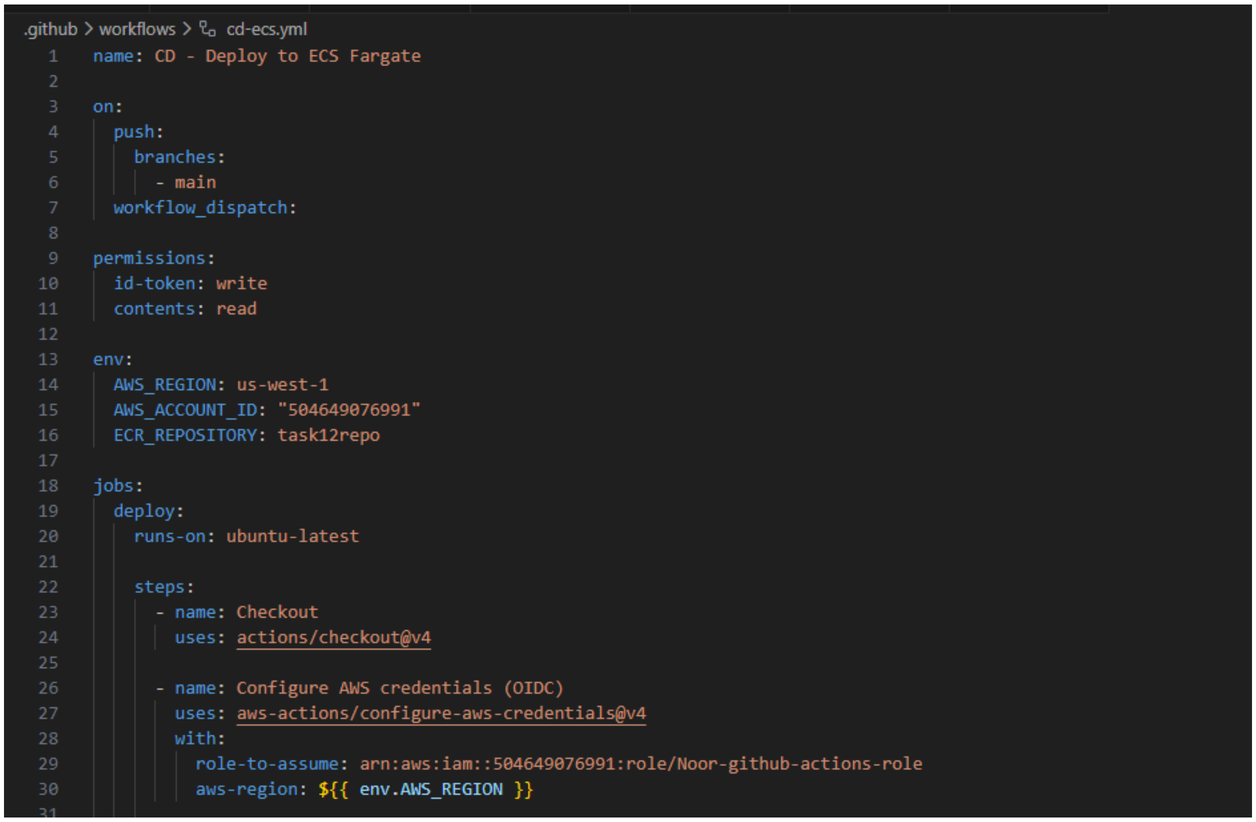


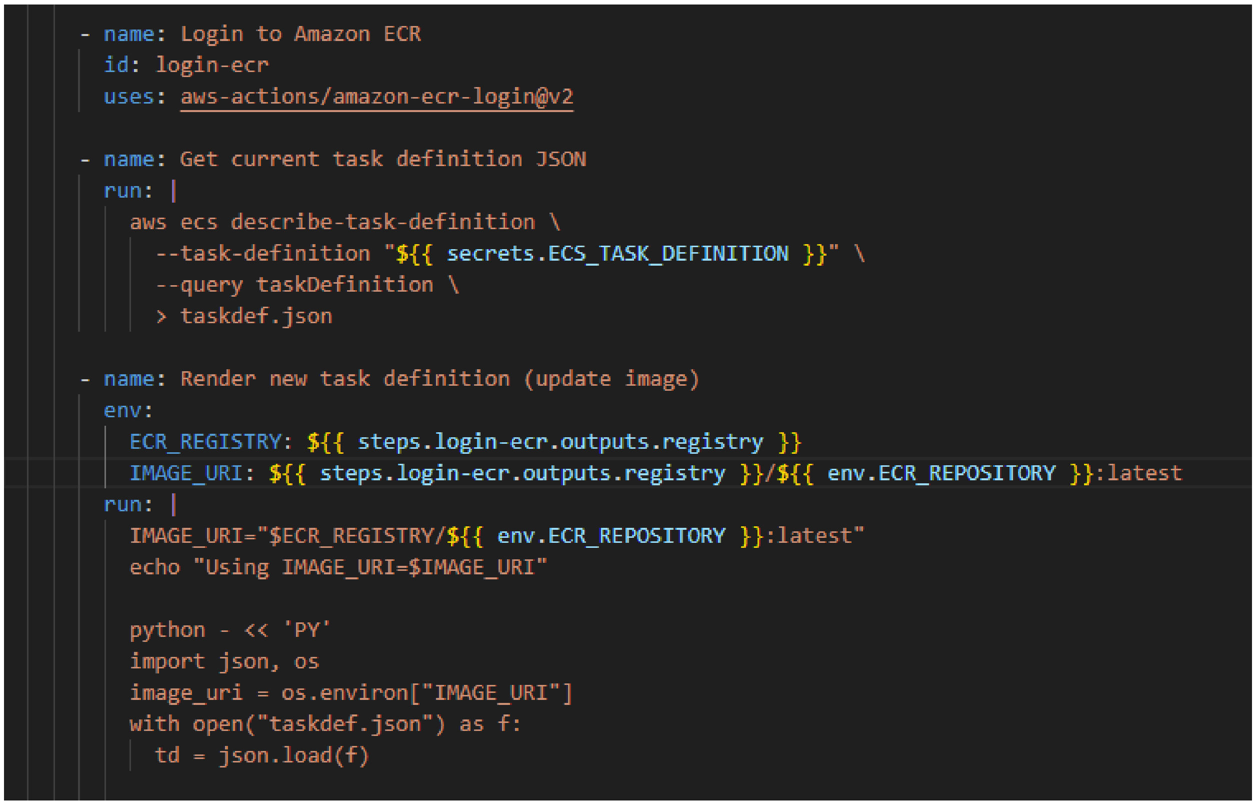
Image tags	Type	Created at	Image size	Image digest	Last pulled at
latest, d0987be071a79b814cbfbe2bf4b49102b2a7f84c	Image	04 January 2026, 01:53:51 (UTC+05)	49.36	sha256:c864…	04 January 2026, 01:54:24 (UTC+05)

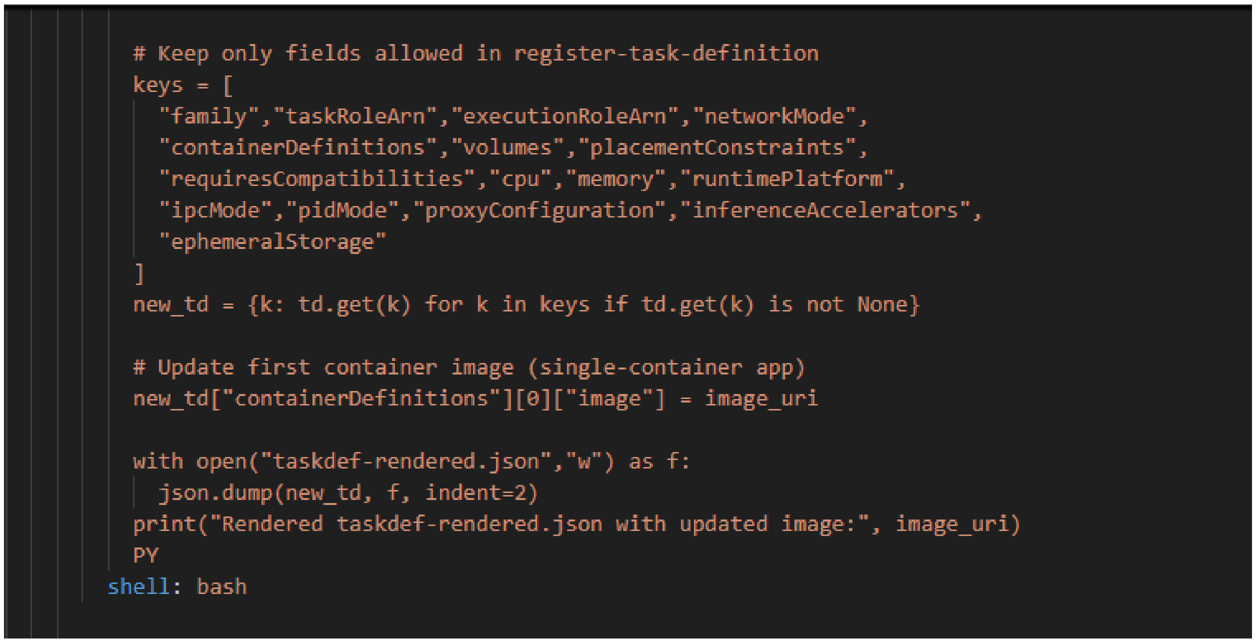
9. CD Pipeline (Deploy to ECS)

Deployment Workflow

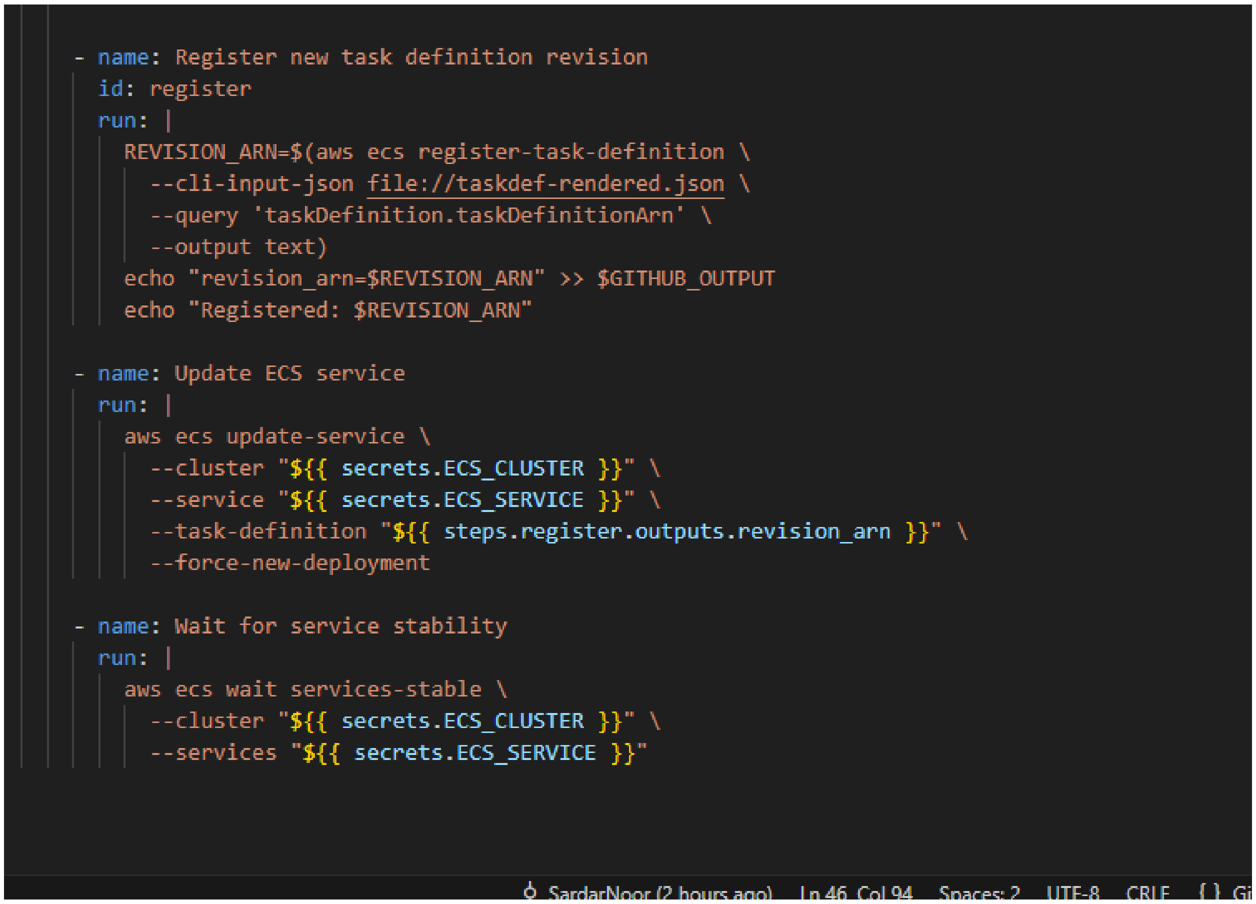
1. Fetch existing task definition
2. Update container image
3. Register new revision
4. Update ECS service
5. Wait for service stability


.github > workflows > cd-ecs.yml
```yaml
name: CD - Deploy to ECS Fargate

on:
  push:
    branches:
      - main
  workflow_dispatch:

permissions:
  id-token: write
  contents: read

env:
  AWS_REGION: us-west-1
  AWS_ACCOUNT_ID: "504649076991"
  ECR_REPOSITORY: task12repo

jobs:
  deploy:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout
        uses: actions/checkout@v4

      - name: Configure AWS credentials (OIDC)
        uses: aws-actions/configure-aws-credentials@v4
        with:
          role-to-assume: arn:aws:iam::504649076991:role/Noor-github-actions-role
          aws-region: ${{ env.AWS_REGION }}
```

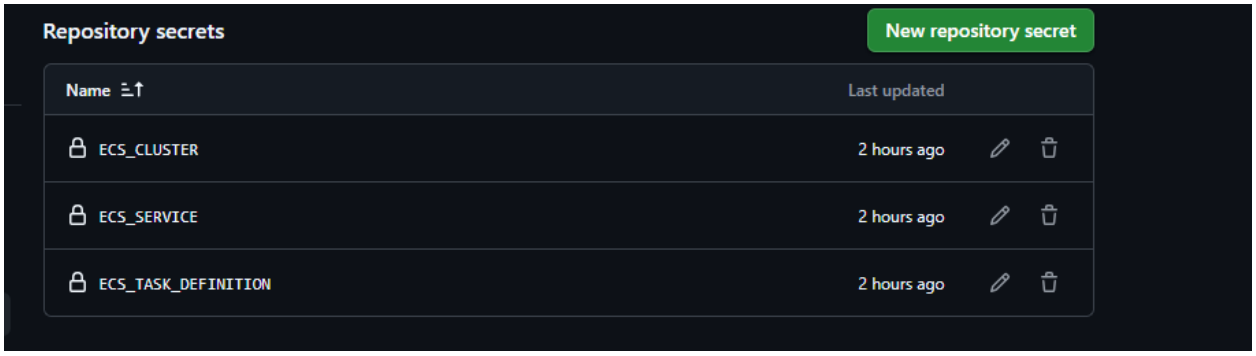

```yaml
      - name: Login to Amazon ECR
        id: login-ecr
        uses: aws-actions/amazon-ecr-login@v2

      - name: Get current task definition JSON
        run: |
          aws ecs describe-task-definition \
            --task-definition "${{ secrets.ECS_TASK_DEFINITION }}" \
            --query taskDefinition \
            > taskdef.json

      - name: Render new task definition (update image)
        env:
          ECR_REGISTRY: ${{ steps.login-ecr.outputs.registry }}
          IMAGE_URI: ${{ steps.login-ecr.outputs.registry }}/${{ env.ECR_REPOSITORY }}:latest
        run: |
          IMAGE_URI="$ECR_REGISTRY/${{ env.ECR_REPOSITORY }}:latest"
          echo "Using IMAGE_URI=$IMAGE_URI"

          python - << 'PY'
          import json, os
          image_uri = os.environ["IMAGE_URI"]
          with open("taskdef.json") as f:
            td = json.load(f)
```



```
  # Keep only fields allowed in register-task-definition
  keys = [
    "family","taskRoleArn","executionRoleArn","networkMode",
    "containerDefinitions","volumes","placementConstraints",
    "requiresCompatibilities","cpu","memory","runtimePlatform",
    "ipcMode","pidMode","proxyConfiguration","inferenceAccelerators",
    "ephemeralStorage"
  ]
  new_td = {k: td.get(k) for k in keys if td.get(k) is not None}

  # Update first container image (single-container app)
  new_td["containerDefinitions"][0]["image"] = image_uri

  with open("taskdef-rendered.json","w") as f:
    json.dump(new_td, f, indent=2)
  print("Rendered taskdef-rendered.json with updated image:", image_uri)
  PY
shell: bash
```



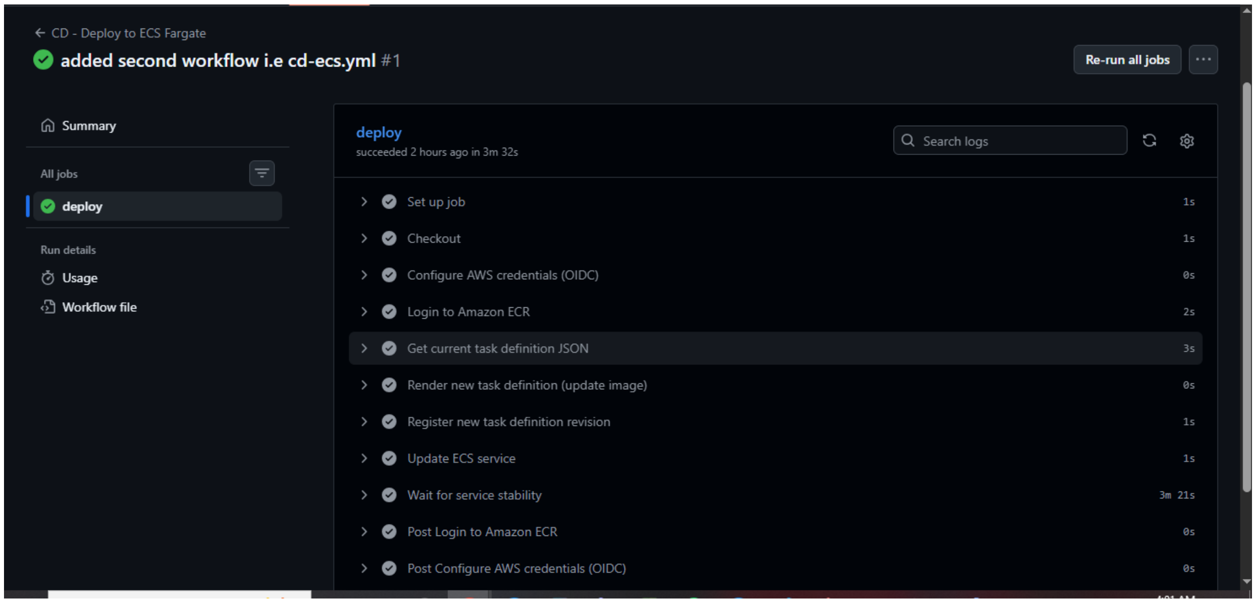
```
- name: Register new task definition revision
  id: register
  run: |
    REVISION_ARN=$(aws ecs register-task-definition \
      --cli-input-json file://taskdef-rendered.json \
      --query 'taskDefinition.taskDefinitionArn' \
      --output text)
    echo "revision_arn=$REVISION_ARN" >> $GITHUB_OUTPUT
    echo "Registered: $REVISION_ARN"

- name: Update ECS service
  run: |
    aws ecs update-service \
      --cluster "${{ secrets.ECS_CLUSTER }}" \
      --service "${{ secrets.ECS_SERVICE }}" \
      --task-definition "${{ steps.register.outputs.revision_arn }}" \
      --force-new-deployment

- name: Wait for service stability
  run: |
    aws ecs wait services-stable \
      --cluster "${{ secrets.ECS_CLUSTER }}" \
      --services "${{ secrets.ECS_SERVICE }}"
```



Repository secrets

New repository secret

Name	Last updated
ECS_CLUSTER	2 hours ago
ECS_SERVICE	2 hours ago
ECS_TASK_DEFINITION	2 hours ago



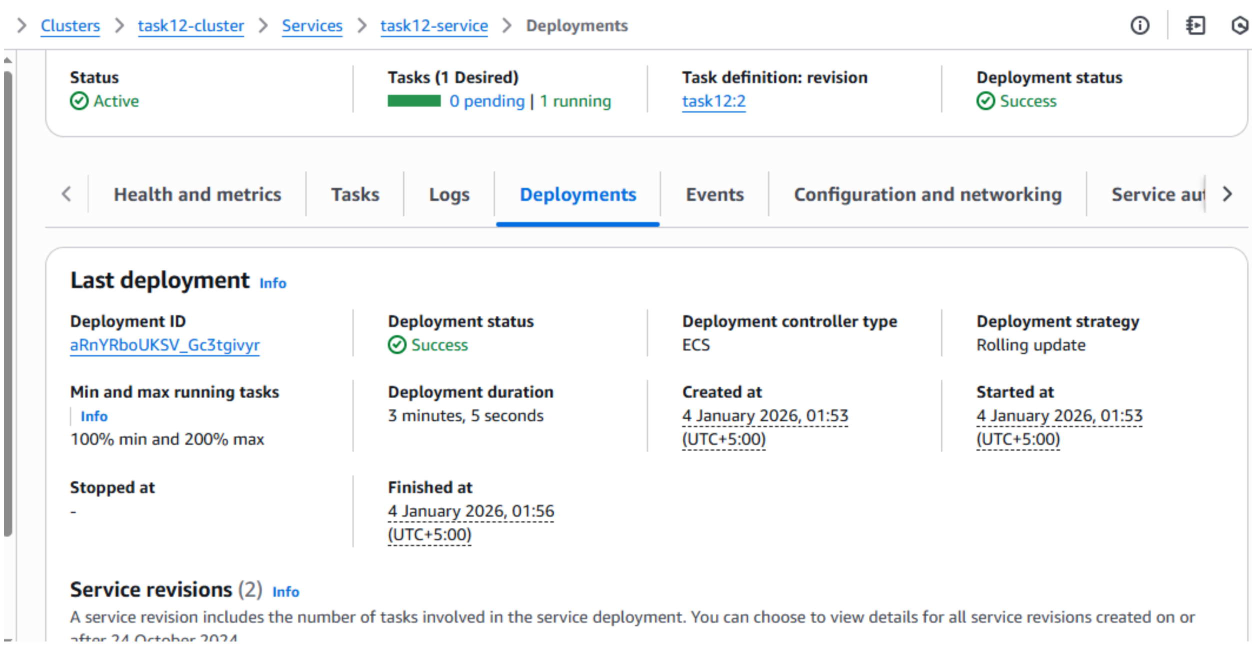
← CD - Deploy to ECS Fargate

added second workflow i.e cd-ecs.yml #1

Re-run all jobs

deploy
succeeded 2 hours ago in 3m 32s

Set up job	1s
Checkout	1s
Configure AWS credentials (OIDC)	0s
Login to Amazon ECR	2s
Get current task definition JSON	3s
Render new task definition (update image)	0s
Register new task definition revision	1s
Update ECS service	1s
Wait for service stability	3m 21s
Post Login to Amazon ECR	0s
Post Configure AWS credentials (OIDC)	0s



Clusters > task12-cluster > Services > task12-service > Deployments

Status	Tasks (1 Desired)	Task definition: revision	Deployment status
Active	0 pending | 1 running	task12:2	Success

Last deployment Info

Deployment ID aRnYRboUKSV_Gc3tgivyr	**Deployment status** Success	**Deployment controller type** ECS	**Deployment strategy** Rolling update
Min and max running tasks Info 100% min and 200% max	**Deployment duration** 3 minutes, 5 seconds	**Created at** 4 January 2026, 01:53 (UTC+5:00)	**Started at** 4 January 2026, 01:53 (UTC+5:00)
Stopped at -	**Finished at** 4 January 2026, 01:56 (UTC+5:00)		

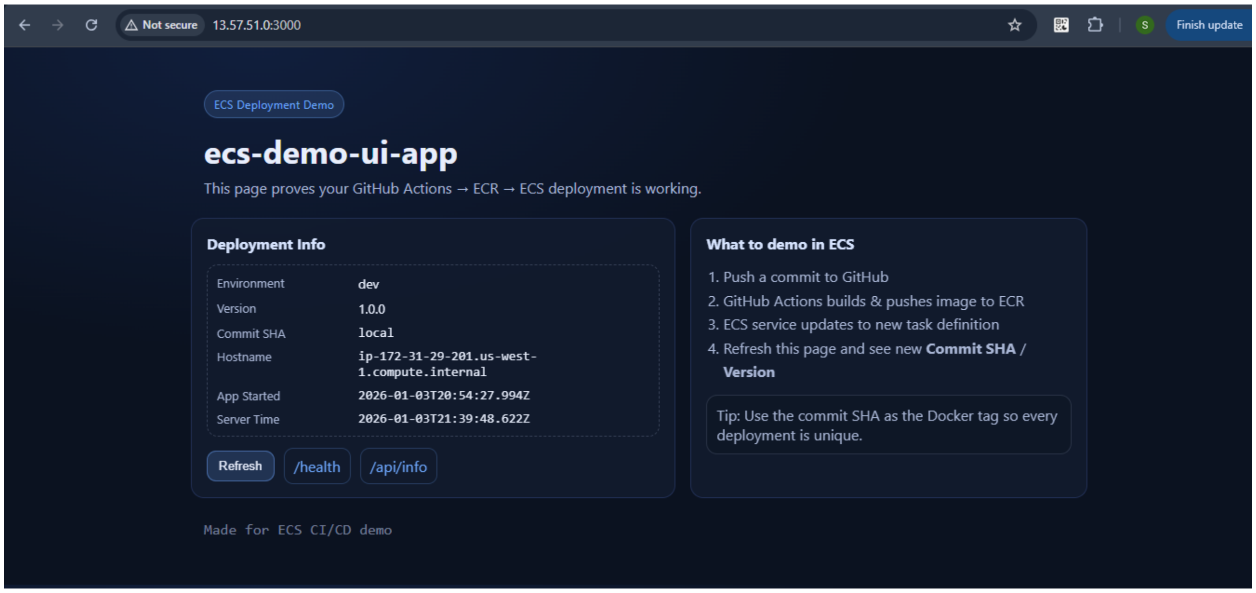
Service revisions (2) Info

A service revision includes the number of tasks involved in the service deployment. You can choose to view details for all service revisions created on or

10. Verification & Rollback

Verification

1. Confirm ECS service is stable
2. Task state: **RUNNING**
3. Retrieve task public IP
4. Access the application



ecs-demo-ui-app

This page proves your GitHub Actions → ECR → ECS deployment is working.

Deployment Info

Environment	dev
Version	1.0.0
Commit SHA	local
Hostname	ip-172-31-29-201.us-west-1.compute.internal
App Started	2026-01-03T20:54:27.994Z
Server Time	2026-01-03T21:39:48.622Z

What to demo in ECS

1. Push a commit to GitHub
2. GitHub Actions builds & pushes image to ECR
3. ECS service updates to new task definition
4. Refresh this page and see new **Commit SHA / Version**

Tip: Use the commit SHA as the Docker tag so every deployment is unique.

Made for ECS CI/CD demo

Rollback Strategies

Option 1 – Task Definition Rollback

Update ECS service to previous revision

Option 2 – Image SHA Rollback

Use specific ECR image tag

11. Issues Faced & Resolutions

Issue 1: GitHub Actions authentication failure

Cause: Missing OIDC trust conditions
Resolution: Corrected IAM trust policy

Issue 2: Port mapping conflict

Cause: Duplicate port mapping in Fargate
Resolution: Ensured unique port mapping name

12. Conclusion

This project demonstrates a modern deployment architecture using AWS native services and GitHub Actions. By leveraging OIDC authentication, container best practices, and ECS Fargate, the solution eliminates manual deployments, reduces security risks, and enables rapid, reliable releases.