End-to-End ECS Deployment using Terraform and GitHub Actions

Overview

This project consists of two GitHub Actions workflows that work together to deploy a two-tier application architectureon AWS

Workflow 1 is used to deploys the infrastructure (ECS cluster, RDS database, networking components) using Terraform

Workflow 2 is used to build a new Docker image of the application and deploy/update it to the existing ECS Service/Task definition

Workflow 1:

Infrastructure Deployment with Terraform {#wf_class12_1_tf_dev_deploy_ecs}

https://github.com/rajeshchandranaws3/bootcamp-august-rajesh/blob/main/.github/workflows/wf_class12_1_tf_dev_deploy_ecs.yaml

Purpose

This workflow provisions the complete AWS infrastructure required for running the two-tier application, including VPC,subnets, security groups, ECS cluster, RDS database, load balancers, and other necessary resources.

Trigger

```
on:
  push:
    branches:
      - main
    paths:
      - 'class10-terraform-dynamic/**'
  workflow_dispatch:
```

The workflow uses workflow_dispatch trigger as well as push event to main branch, and the path mentions it can trigger if there is any change in that path.

Manual triggering provides better control over infrastructure deployments

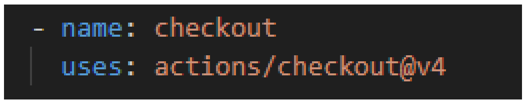
Environment Variables

```yaml
env:
  terraform_version: 1.5.7
  infra_path: class10-terraform-dynamic
```

Jobs

```yaml
jobs:
  terraform-dev:
    runs-on: ubuntu-latest
```

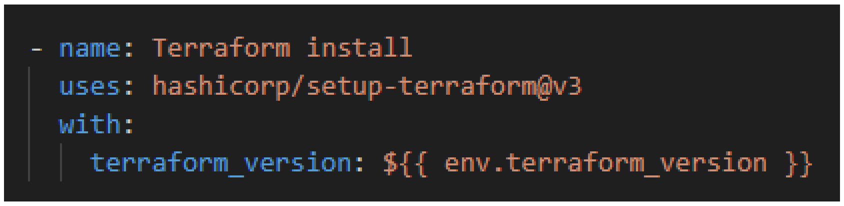
Step 1: Checkout Code



```yaml
- name: checkout
  uses: actions/checkout@v4
```

- Clones the repository code to the runner
- Uses version 4 of the checkout action
- Provides access to Terraform configuration files

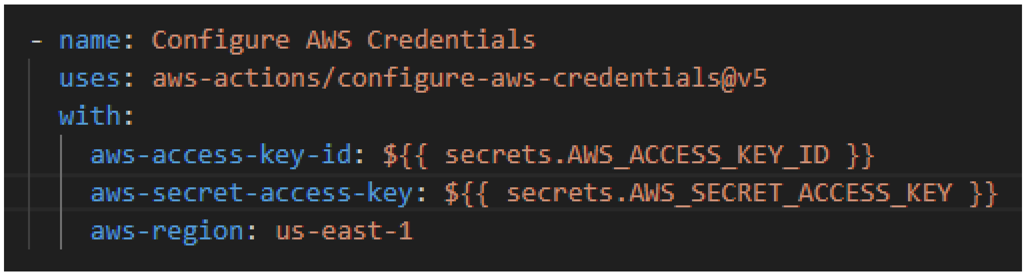
Step 2: Install Terraform



```yaml
- name: Terraform install
  uses: hashicorp/setup-terraform@v3
  with:
    terraform_version: ${{ env.terraform_version }}
```

- Installs Terraform CLI on the runner
- Uses HashiCorp's official action
- Installs the specific version defined in environment variables (1.5.7)

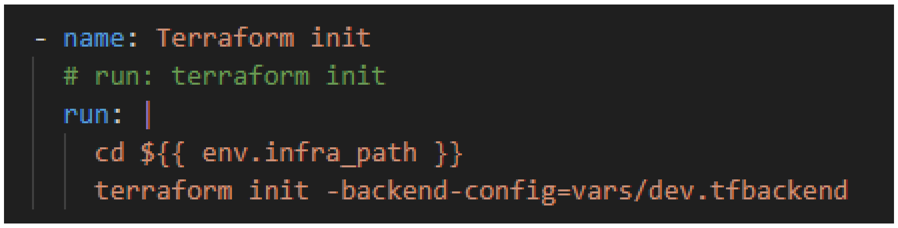
Step 3: Configure AWS Credentials



```yaml
- name: Configure AWS Credentials
  uses: aws-actions/configure-aws-credentials@v5
  with:
    aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
    aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
    aws-region: us-east-1
```

- Authenticates with AWS using stored credentials
- Credentials are stored securely in GitHub Secrets
- Sets AWS region to us-east-1
- Enables Terraform to create resources in AWS

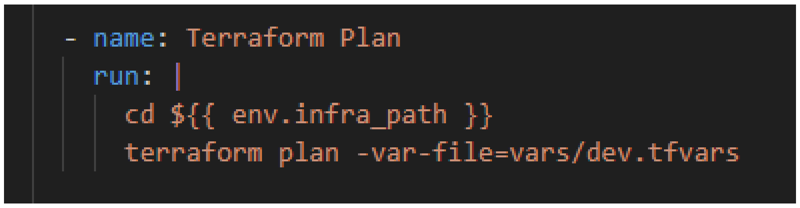
Step 4: Terraform Initialization



```yaml
- name: Terraform init
  # run: terraform init
  run: |
    cd ${{ env.infra_path }}
    terraform init -backend-config=vars/dev.tfbackend
```

- Navigates to the Terraform configuration directory
- Runs terraform init to initialize the working directory
- Configures backend using dev.tfbackend file
- Downloads required provider plugins (AWS provider)
- Sets up the backend for state management (likely S3 bucket)

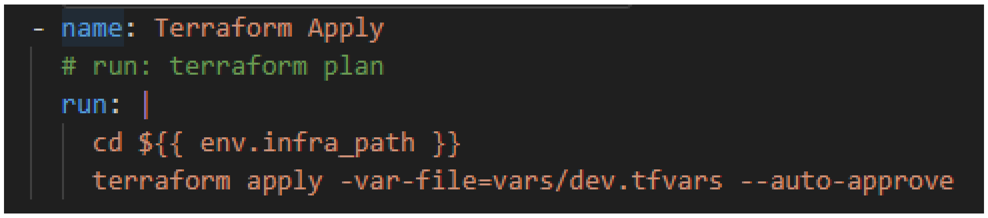
Step 5: Terraform Plan



```yaml
  - name: Terraform Plan
    run: |
      cd ${{ env.infra_path }}
      terraform plan -var-file=vars/dev.tfvars
```

- Creates an execution plan showing what changes Terraform will make
- Uses variables from dev.tfvars file for environment-specific configuration
- Displays resources to be created, modified, or destroyed
- Does not make any actual changes to infrastructure
- Helps review changes before applying them

Step 6: Terraform Apply



```
- name: Terraform Apply
  # run: terraform plan
  run: |
    cd ${{ env.infra_path }}
    terraform apply -var-file=vars/dev.tfvars --auto-approve
```

- Executes the Terraform plan to create/update infrastructure
- Uses --auto-approve flag to skip manual confirmation
- Creates resources such as:
  - VPC and networking components
  - ECS cluster and task definitions
  - RDS database instance
  - Security groups and IAM roles
  - Load balancers
  - ECR repository
- Infrastructure is now ready for application deployment

Workflow 2:

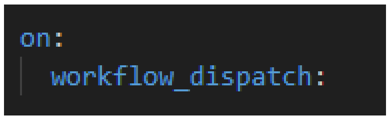
Build New Image and Update the Deployment {#wf_class12_2_build_image_update_ecs}

https://github.com/rajeshchandranaws3/bootcamp-august-rajesh/blob/main/.github/workflows/wf_class12_2_build_and_update_ecs.yaml

Purpose

This workflow builds a Docker image of the application, pushes it to Amazon ECR (Elastic Container Registry), and updates the ECS service to run the new image.

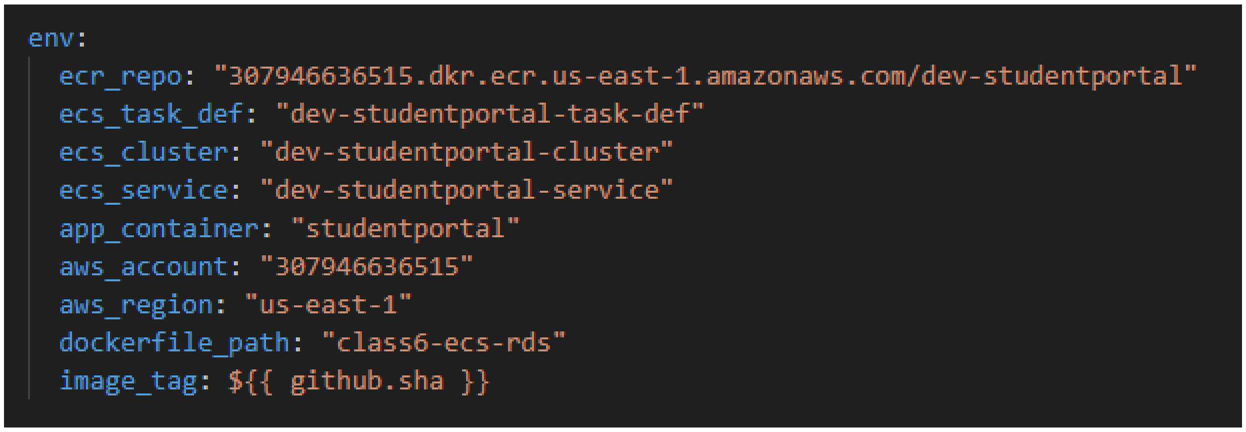
Trigger



```
on:
  workflow_dispatch:
```

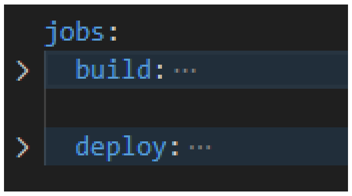
- Manual trigger only
- Provides control over when new versions are deployed

Environment Variables


```yaml
env:
  ecr_repo: "307946636515.dkr.ecr.us-east-1.amazonaws.com/dev-studentportal"
  ecs_task_def: "dev-studentportal-task-def"
  ecs_cluster: "dev-studentportal-cluster"
  ecs_service: "dev-studentportal-service"
  app_container: "studentportal"
  aws_account: "307946636515"
  aws_region: "us-east-1"
  dockerfile_path: "class6-ecs-rds"
  image_tag: ${{ github.sha }}
```

- ecr_repo: Full ECR repository URL for storing Docker images
- ecs_task_def: Name of the ECS task definition to update
- ecs_cluster: Name of the ECS cluster running the application
- ecs_service: Name of the ECS service to update
- app_container: Container name within the task definition
- dockerfile_path: Location of the Dockerfile
- image_tag: Uses Git commit SHA for unique image versioning

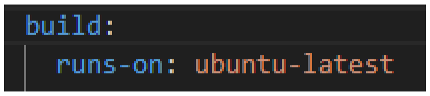
Jobs


```yaml
jobs:
  build: ...

  deploy: ...
```

There are 2 jobs, namely “build” and “deploy”. It runs sequentially since we have provided “needs” keyword in the “deploy” job

Job1: Build


```yaml
build:
  runs-on: ubuntu-latest
```

- It runs on the latest ubuntu machine

Step 1: Checkout Code

```yaml
- name: checkout the code
  uses: actions/checkout@v4
```

- Clones repository to access application code and Dockerfile

Step 2: Configure AWS Credentials

```yaml
- name: Configure AWS Credentials
  uses: aws-actions/configure-aws-credentials@v5
  with:
    aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
    aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
    aws-region: ${{ env.aws_region }}
```

- Authenticates with AWS for ECR access

Step 3: ECR Login

```yaml
- name: ecr login
  run: aws ecr get-login-password --region ${{ env.aws_region }} | docker login --username AWS --password-stdin ${{ env.aws_account }}.dkr.ecr.${{ env.aws_region
```

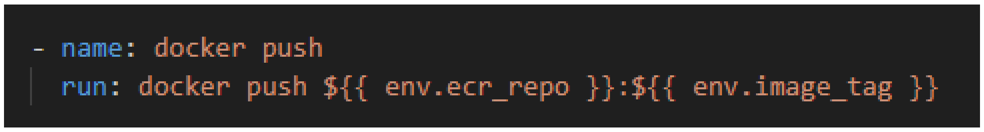
- Retrieves authentication token from ECR
- Pipes the password to Docker login command
- Authenticates Docker client with ECR
- Enables pushing images to ECR repository

Step 4: Docker Build

```yaml
- name: docker build
  run: |
    cd ${{ env.dockerfile_path }}
    docker build -t ecsapp .
    docker tag ecsapp ${{ env.ecr_repo }}:${{ env.image_tag }}
```

- Navigates to directory containing Dockerfile
- Builds Docker image with tag “ecsapp”
- Tags the image with ECR repository URL and Git commit SHA
- Creates a versioned image ready for deployment

Step 5: Docker Push

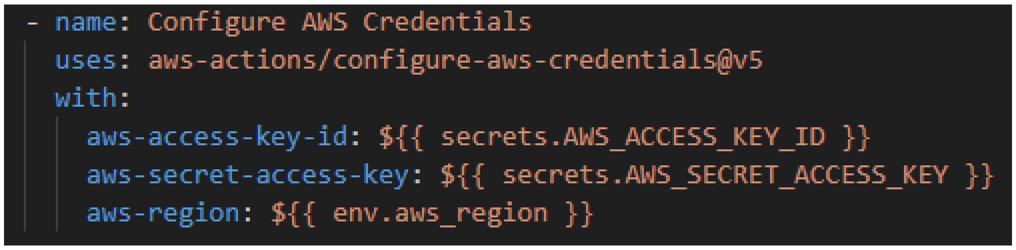


```yaml
- name: docker push
  run: docker push ${{ env.ecr_repo }}:${{ env.image_tag }}
```

- Pushes the built image to ECR repository
- Image is now available for ECS to pull and deploy
- Uses the commit SHA as the version tag

Job 2: Deploy

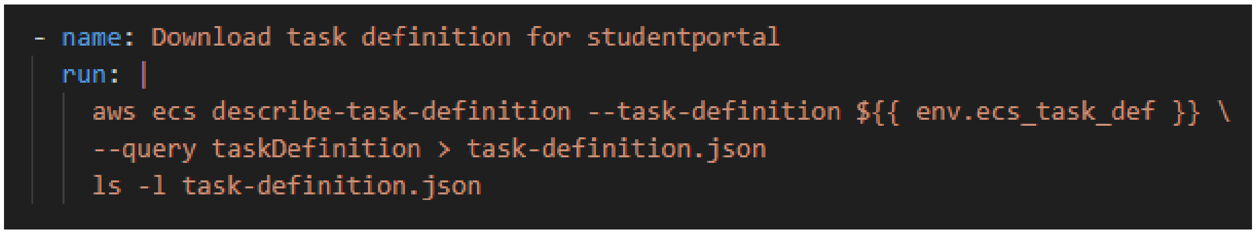
Step 1: Configure AWS Credentials



```yaml
- name: Configure AWS Credentials
  uses: aws-actions/configure-aws-credentials@v5
  with:
    aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
    aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
    aws-region: ${{ env.aws_region }}
```

- Re-authenticates with AWS (new job, new runner environment)

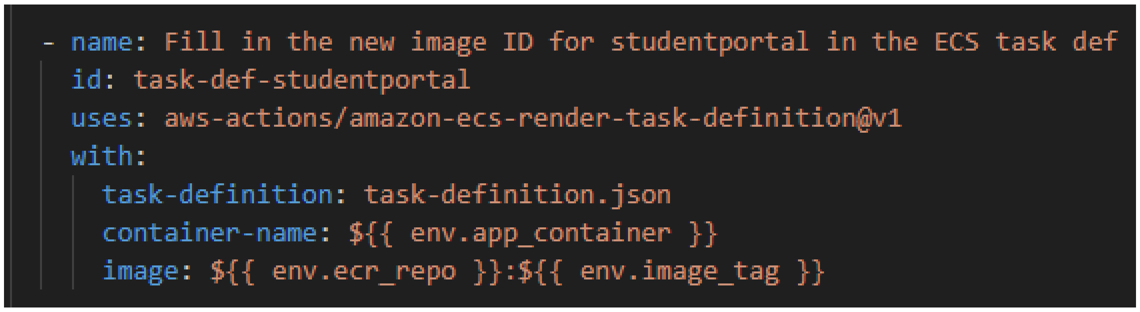
Step 2: Download Task Definition



```yaml
- name: Download task definition for studentportal
  run: |
    aws ecs describe-task-definition --task-definition ${{ env.ecs_task_def }} \
    --query taskDefinition > task-definition.json
    ls -l task-definition.json
```

- Retrieves the current ECS task definition from AWS
- Saves it as JSON file
- Task definition contains container configurations, resource requirements, etc.
- Lists the file to confirm it was created

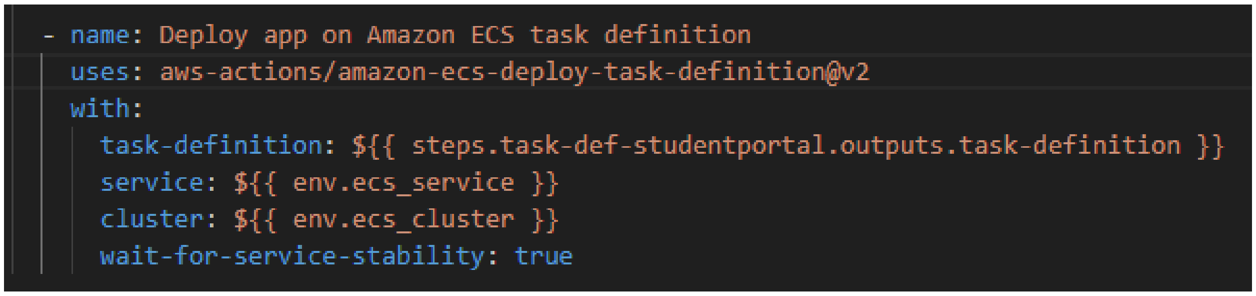
Step 3: Update Task Definition with New Image



```yaml
- name: Fill in the new image ID for studentportal in the ECS task def
  id: task-def-studentportal
  uses: aws-actions/amazon-ecs-render-task-definition@v1
  with:
    task-definition: task-definition.json
    container-name: ${{ env.app_container }}
    image: ${{ env.ecr_repo }}:${{ env.image_tag }}
```

- Uses AWS GitHub Action to modify the task definition
- Updates the image URI for the specified container
- Points to the newly built image using commit SHA tag
- Outputs modified task definition for next step
- Step ID allows referencing output in subsequent steps

Step 4: Deploy to ECS



```yaml
- name: Deploy app on Amazon ECS task definition
  uses: aws-actions/amazon-ecs-deploy-task-definition@v2
  with:
    task-definition: ${{ steps.task-def-studentportal.outputs.task-definition }}
    service: ${{ env.ecs_service }}
    cluster: ${{ env.ecs_cluster }}
    wait-for-service-stability: true
```

- Registers the updated task definition with ECS
- Updates the ECS service to use the new task definition
- Forces a new deployment of the service
- ECS performs rolling update:
  - Starts new tasks with updated image
  - Drains connections from old tasks
  - Stops old tasks once new ones are healthy
- wait-for-service-stability: true means workflow waits until:
  - New tasks are running
  - Old tasks are stopped
  - Service reaches stable state
- Ensures deployment completes successfully before marking workflow as done

Prerequisites

GitHub Secrets Required

Both workflows require the following secrets to be configured in GitHub repository settings:

1. AWS_ACCESS_KEY_ID: AWS IAM access key for authentication
2. AWS_SECRET_ACCESS_KEY: AWS IAM secret key for authentication

AWS Permissions Required

The IAM user/role must have permissions for:

- Terraform Workflow
  - EC2 (VPC, Security Groups, Subnets)
  - ECS (Cluster, Service, Task Definition)
  - RDS (Database instances)
  - IAM (Roles and policies)
  - S3 (Terraform state backend)
  - ECR (Repository creation)
- Build & Deploy Workflow
  - ECR (Push images, authentication)
  - ECS (Describe/update task definitions and services)

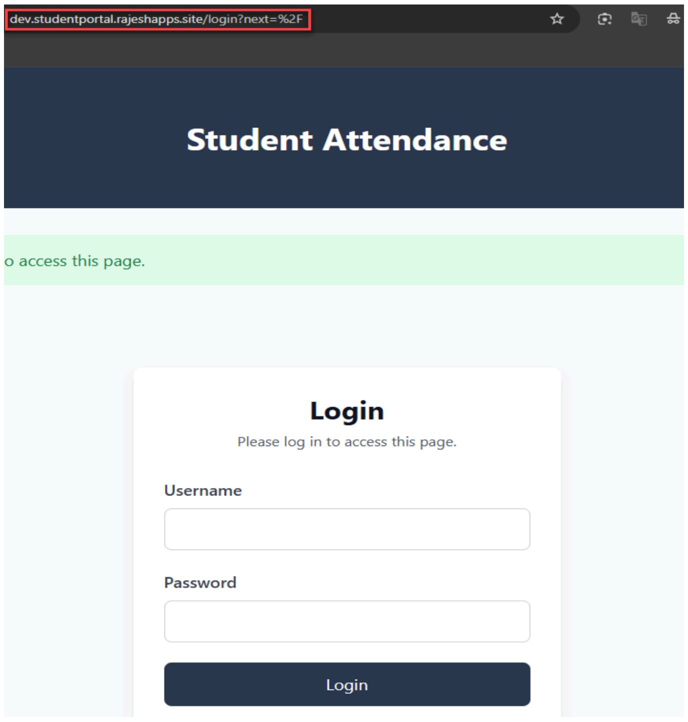
Terraform State Backend

S3 bucket for storing Terraform state (configured in dev.tfbackend)

Repository Structure

- Terraform configurations in class10-terraform-dynamic/
- Application code and Dockerfile in class6-ecs-rds/

Output



dev.studentportal.rajeshapps.site/login?next=%2F

Student Attendance

o access this page.

Login

Please log in to access this page.

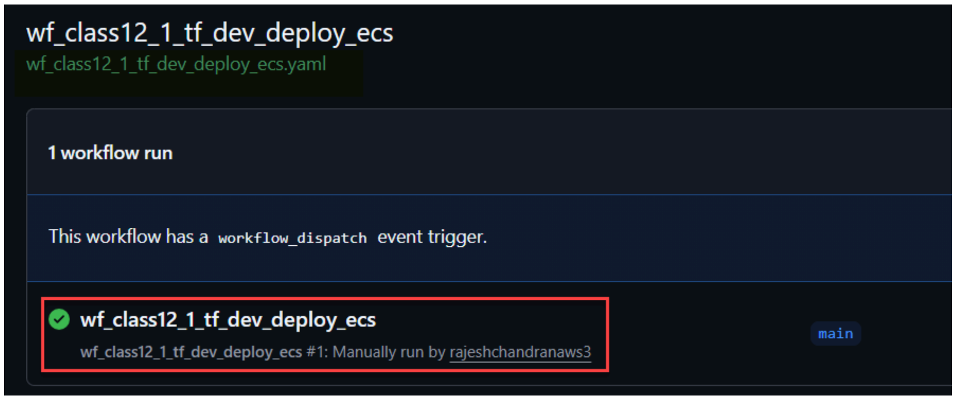
Username

Password

Login

Verification

Initial Workflow



wf_class12_1_tf_dev_deploy_ecs

wf_class12_1_tf_dev_deploy_ecs.yaml

1 workflow run

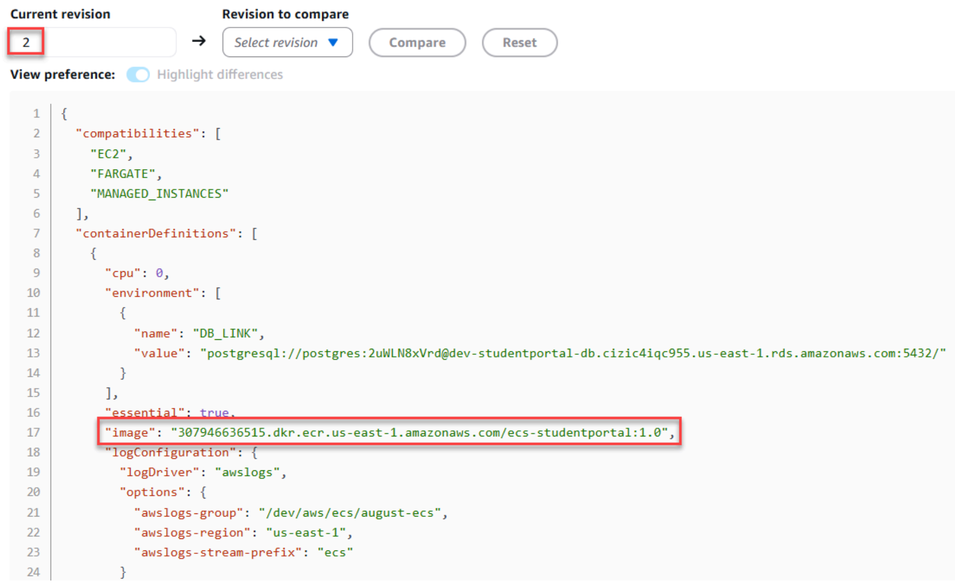
This workflow has a `workflow_dispatch` event trigger.

wf_class12_1_tf_dev_deploy_ecs

wf_class12_1_tf_dev_deploy_ecs #1: Manually run by rajeshchandranaws3

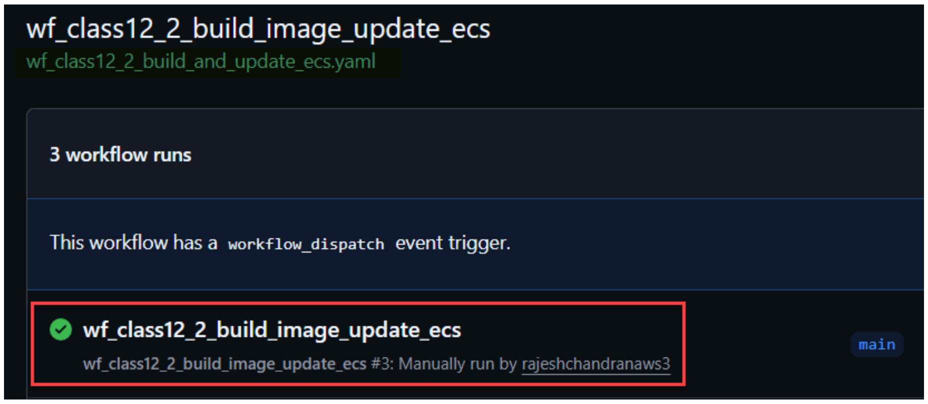
main

Initial Task Definition: Json


Current revision: 2 → Revision to compare: Select revision | Compare | Reset
View preference: Highlight differences

```
{
  "compatibilities": [
    "EC2",
    "FARGATE",
    "MANAGED_INSTANCES"
  ],
  "containerDefinitions": [
    {
      "cpu": 0,
      "environment": [
        {
          "name": "DB_LINK",
          "value": "postgresql://postgres:2uWLN8xVrd@dev-studentportal-db.cizic4iqc955.us-east-1.rds.amazonaws.com:5432/"
        }
      ],
      "essential": true,
      "image": "307946636515.dkr.ecr.us-east-1.amazonaws.com/ecs-studentportal:1.0",
      "logConfiguration": {
        "logDriver": "awslogs",
        "options": {
          "awslogs-group": "/dev/aws/ecs/august-ecs",
          "awslogs-region": "us-east-1",
          "awslogs-stream-prefix": "ecs"
        }
```

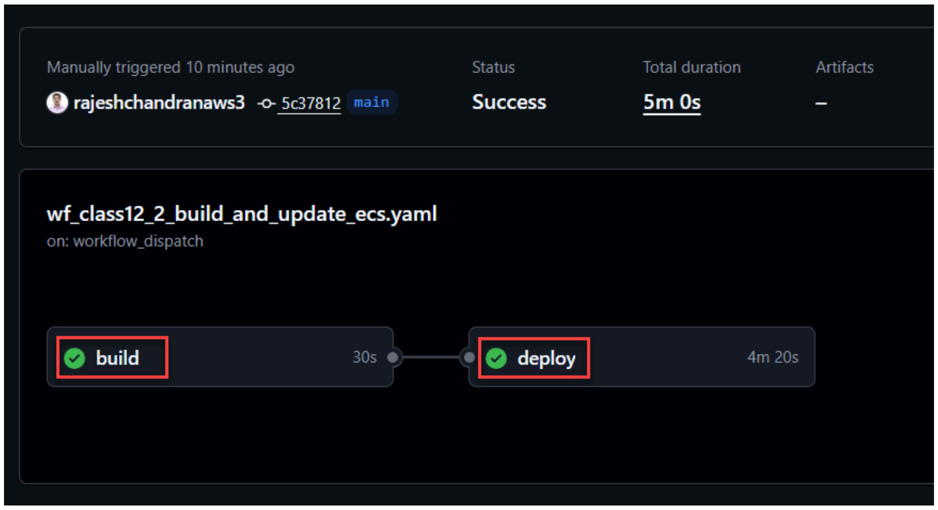
Final Workflow


wf_class12_2_build_image_update_ecs
wf_class12_2_build_and_update_ecs.yaml

3 workflow runs

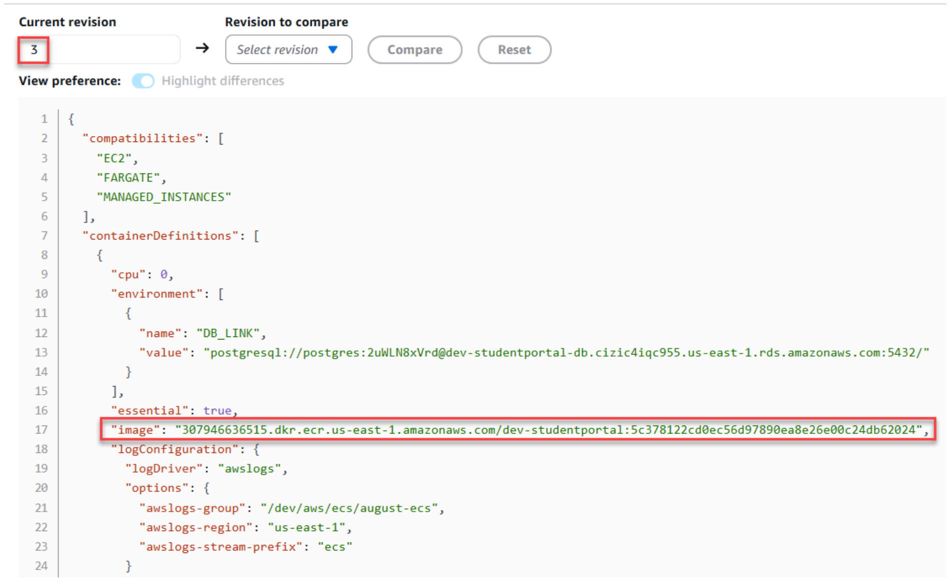
This workflow has a `workflow_dispatch` event trigger.

wf_class12_2_build_image_update_ecs
wf_class12_2_build_image_update_ecs #3: Manually run by rajeshchandranaws3 — main


Manually triggered 10 minutes ago — rajeshchandranaws3 — 5c37812 — main
Status: Success
Total duration: 5m 0s
Artifacts: –

wf_class12_2_build_and_update_ecs.yaml
on: workflow_dispatch

Final Task Definition: Json



```
{
  "compatibilities": [
    "EC2",
    "FARGATE",
    "MANAGED_INSTANCES"
  ],
  "containerDefinitions": [
    {
      "cpu": 0,
      "environment": [
        {
          "name": "DB_LINK",
          "value": "postgresql://postgres:2uWLN8xVrd@dev-studentportal-db.cizic4iqc955.us-east-1.rds.amazonaws.com:5432/"
        }
      ],
      "essential": true,
      "image": "307946636515.dkr.ecr.us-east-1.amazonaws.com/dev-studentportal:5c378122cd0ec56d97890ea8e26e00c24db62024",
      "logConfiguration": {
        "logDriver": "awslogs",
        "options": {
          "awslogs-group": "/dev/aws/ecs/august-ecs",
          "awslogs-region": "us-east-1",
          "awslogs-stream-prefix": "ecs"
        }
```

Conclusion

These two GitHub Actions workflows provide a complete CI/CD pipeline for deploying and managing a two-tierapplication on AWS:

- Workflow 1 handles infrastructure as code, ensuring consistent and repeatable infrastructure provisioning
- Workflow 2 handles continuous deployment, enabling rapid iteration and updates to the application

Together, they demonstrate:

- Infrastructure as Code (Terraform)
- Containerization (Docker)
- Automated deployments (GitHub Actions)
- Cloud-native architecture (AWS ECS/RDS)
- Version control and traceability (Git commit SHA tagging)