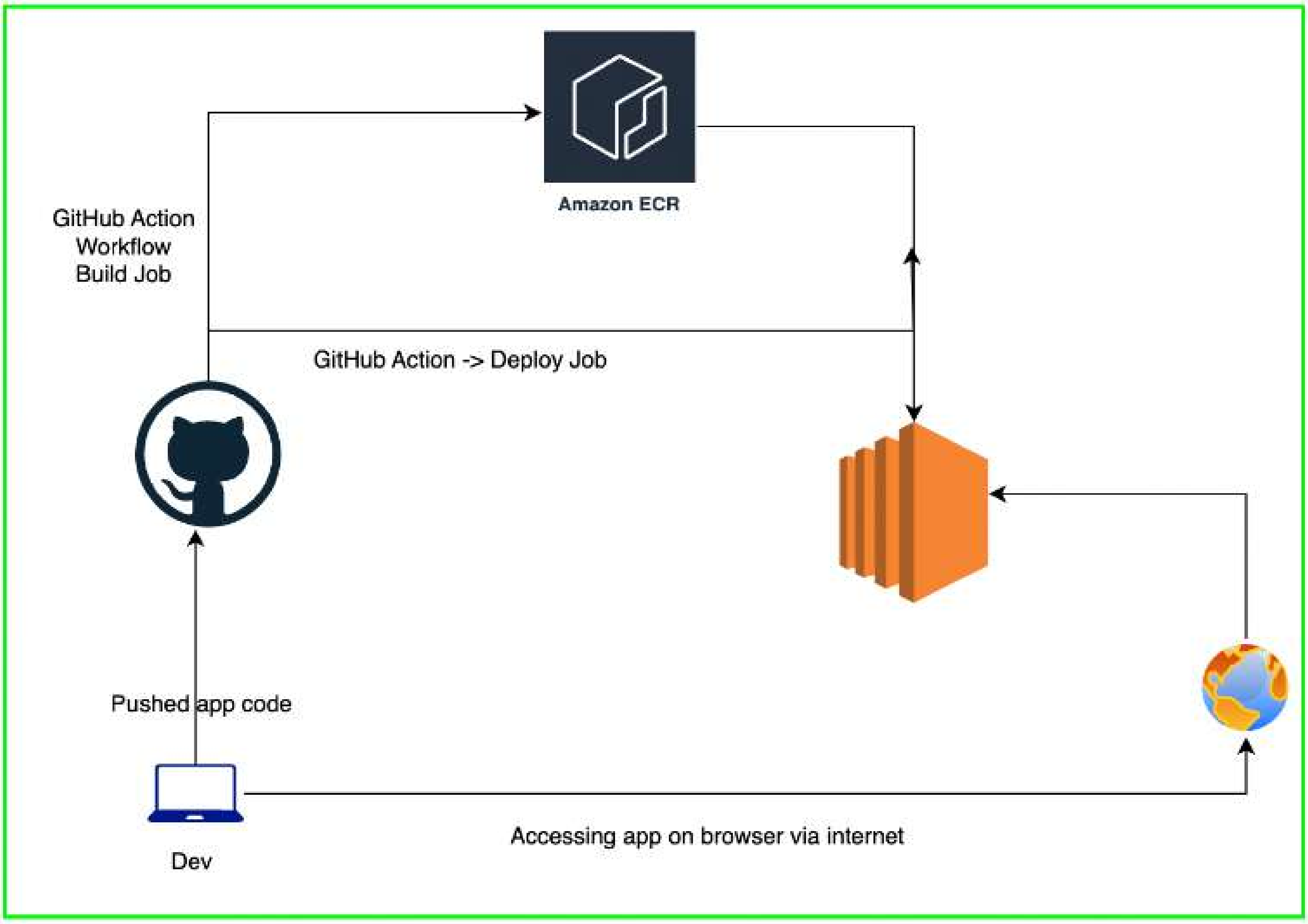
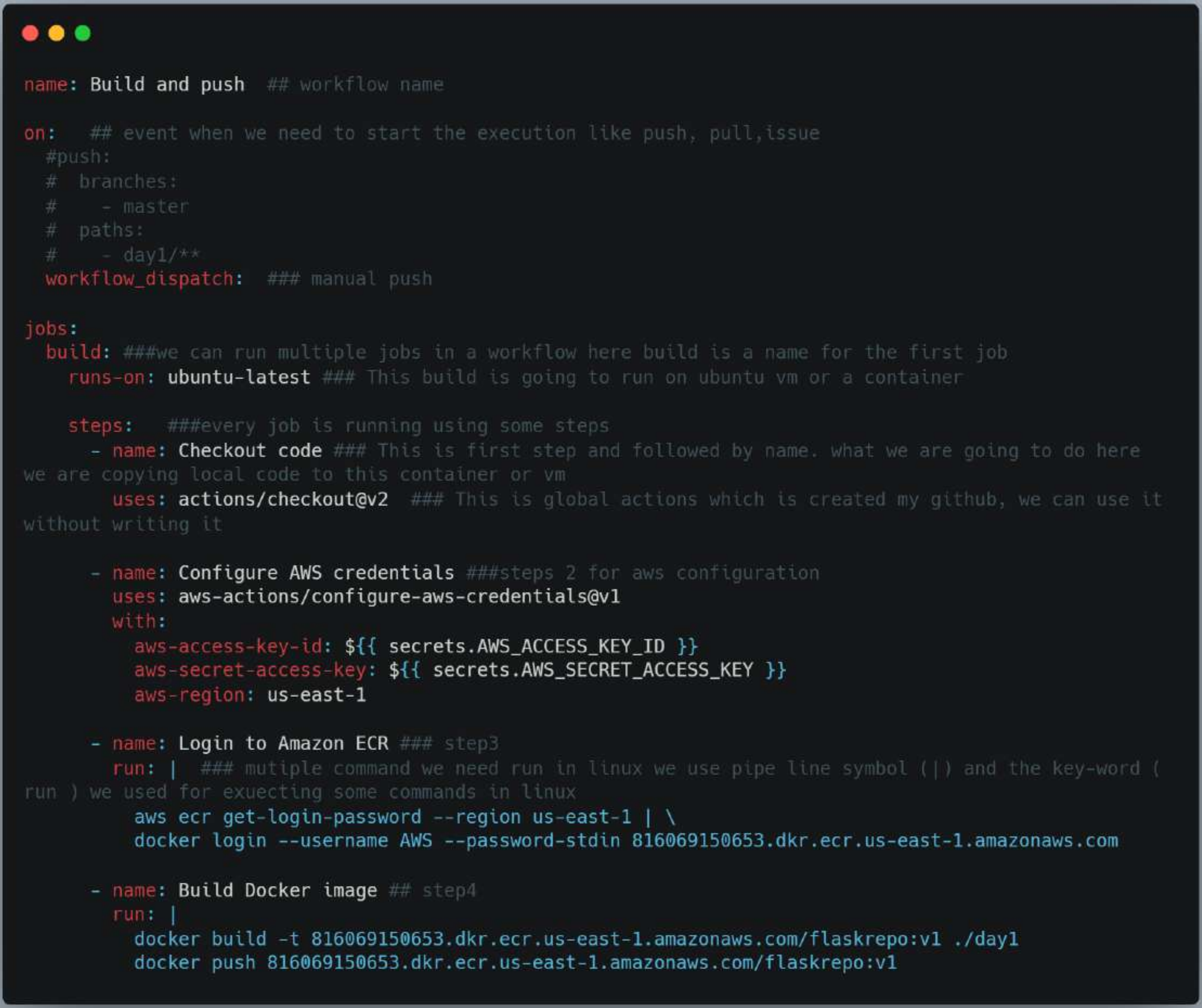
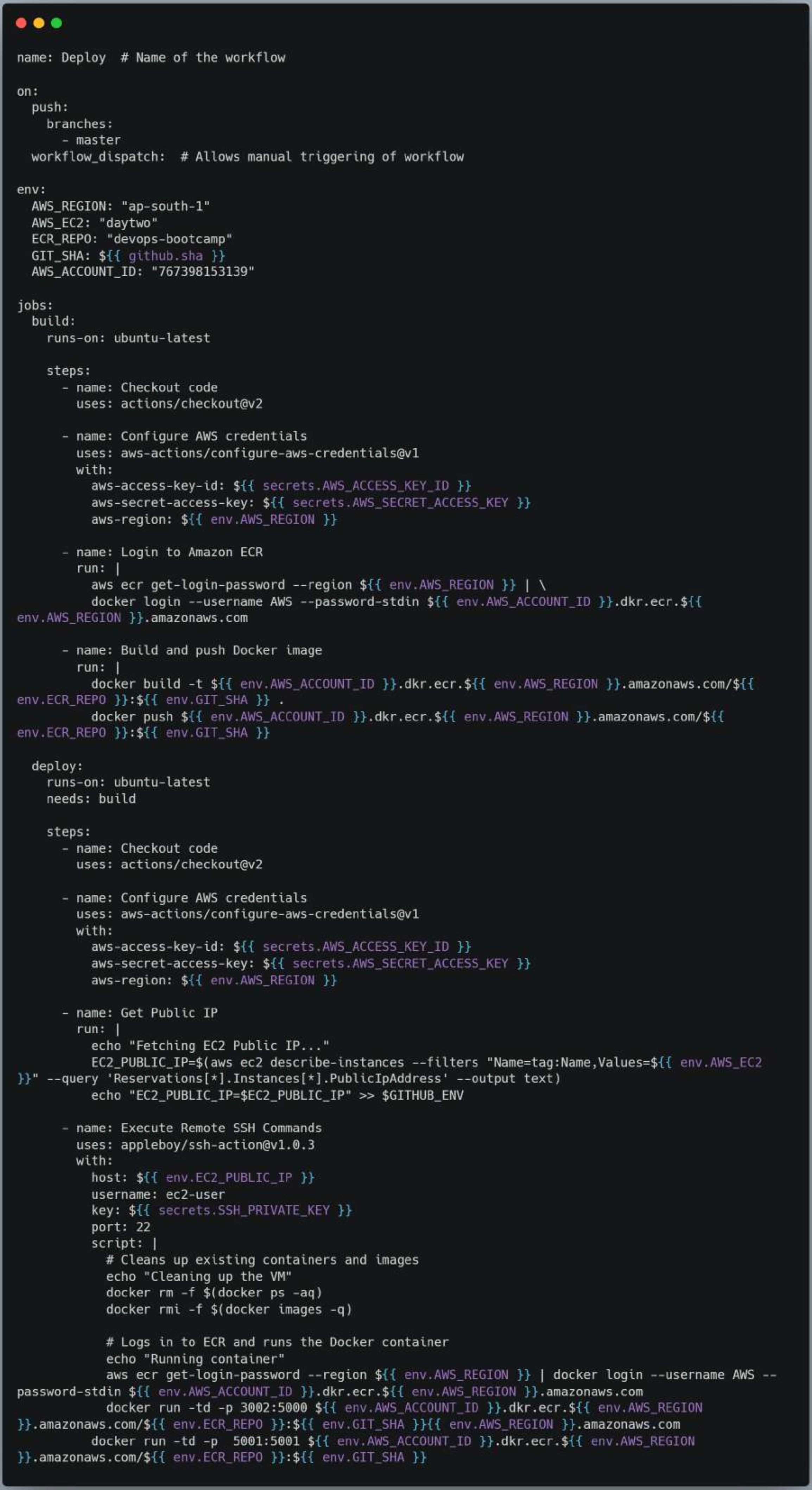

Amazon ECR
GitHub Action
Workflow
Build Job
GitHub Action -> Deploy Job
Pushed app code
Dev
Accessing app on browser via internet



```yaml
name: Build and push  ## workflow name

on:   ## event when we need to start the execution like push, pull,issue
  #push:
  #  branches:
  #    - master
  #  paths:
  #    - day1/**
  workflow_dispatch:  ### manual push

jobs:
  build: ###we can run multiple jobs in a workflow here build is a name for the first job
    runs-on: ubuntu-latest ### This build is going to run on ubuntu vm or a container

    steps:   ###every job is running using some steps
      - name: Checkout code ### This is first step and followed by name. what we are going to do here
we are copying local code to this container or vm
        uses: actions/checkout@v2  ### This is global actions which is created my github, we can use it
without writing it

      - name: Configure AWS credentials ###steps 2 for aws configuration
        uses: aws-actions/configure-aws-credentials@v1
        with:
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          aws-region: us-east-1

      - name: Login to Amazon ECR ### step3
        run: |  ### mutiple command we need run in linux we use pipe line symbol (|) and the key-word (
run ) we used for exuecting some commands in linux
          aws ecr get-login-password --region us-east-1 | \
          docker login --username AWS --password-stdin 816069150653.dkr.ecr.us-east-1.amazonaws.com

      - name: Build Docker image ## step4
        run: |
          docker build -t 816069150653.dkr.ecr.us-east-1.amazonaws.com/flaskrepo:v1 ./day1
          docker push 816069150653.dkr.ecr.us-east-1.amazonaws.com/flaskrepo:v1
```



```yaml
name: Deploy  # Name of the workflow

on:
  push:
    branches:
      - master
  workflow_dispatch:  # Allows manual triggering of workflow

env:
  AWS_REGION: "ap-south-1"
  AWS_EC2: "daytwo"
  ECR_REPO: "devops-bootcamp"
  GIT_SHA: ${{ github.sha }}
  AWS_ACCOUNT_ID: "767398153139"

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v2

      - name: Configure AWS credentials
        uses: aws-actions/configure-aws-credentials@v1
        with:
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          aws-region: ${{ env.AWS_REGION }}

      - name: Login to Amazon ECR
        run: |
          aws ecr get-login-password --region ${{ env.AWS_REGION }} | \
          docker login --username AWS --password-stdin ${{ env.AWS_ACCOUNT_ID }}.dkr.ecr.${{
env.AWS_REGION }}.amazonaws.com

      - name: Build and push Docker image
        run: |
          docker build -t ${{ env.AWS_ACCOUNT_ID }}.dkr.ecr.${{ env.AWS_REGION }}.amazonaws.com/${{
env.ECR_REPO }}:${{ env.GIT_SHA }} .
          docker push ${{ env.AWS_ACCOUNT_ID }}.dkr.ecr.${{ env.AWS_REGION }}.amazonaws.com/${{
env.ECR_REPO }}:${{ env.GIT_SHA }}

  deploy:
    runs-on: ubuntu-latest
    needs: build

    steps:
      - name: Checkout code
        uses: actions/checkout@v2

      - name: Configure AWS credentials
        uses: aws-actions/configure-aws-credentials@v1
        with:
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          aws-region: ${{ env.AWS_REGION }}

      - name: Get Public IP
        run: |
          echo "Fetching EC2 Public IP..."
          EC2_PUBLIC_IP=$(aws ec2 describe-instances --filters "Name=tag:Name,Values=${{ env.AWS_EC2
}}" --query 'Reservations[*].Instances[*].PublicIpAddress' --output text)
          echo "EC2_PUBLIC_IP=$EC2_PUBLIC_IP" >> $GITHUB_ENV

      - name: Execute Remote SSH Commands
        uses: appleboy/ssh-action@v1.0.3
        with:
          host: ${{ env.EC2_PUBLIC_IP }}
          username: ec2-user
          key: ${{ secrets.SSH_PRIVATE_KEY }}
          port: 22
          script: |
            # Cleans up existing containers and images
            echo "Cleaning up the VM"
            docker rm -f $(docker ps -aq)
            docker rmi -f $(docker images -q)

            # Logs in to ECR and runs the Docker container
            echo "Running container"
            aws ecr get-login-password --region ${{ env.AWS_REGION }} | docker login --username AWS --
password-stdin ${{ env.AWS_ACCOUNT_ID }}.dkr.ecr.${{ env.AWS_REGION }}.amazonaws.com
            docker run -td -p 3002:5000 ${{ env.AWS_ACCOUNT_ID }}.dkr.ecr.${{ env.AWS_REGION
}}.amazonaws.com/${{ env.ECR_REPO }}:${{ env.GIT_SHA }}{{ env.AWS_REGION }}.amazonaws.com
          docker run -td -p  5001:5001 ${{ env.AWS_ACCOUNT_ID }}.dkr.ecr.${{ env.AWS_REGION
}}.amazonaws.com/${{ env.ECR_REPO }}:${{ env.GIT_SHA }}
```

Introduction

CI/CD are parts of the DevOps process for delivering new software as soon as possible with help of automated test and automation build tools like Jenkins, GitHub-Actions.
Few benefits of implementing CI/CD in your organization:

- Faster Delivery
- Observability
- Smaller Code Change
- Easier Rollbacks
- Reduce Costs

AWS Elastic Container Service it gives you a managed set of tools to run Docker containers over AWS maintained compute resources.
In this blog post, I will explain "how to Dockerize a flask hello-world application that takes a message from an env variable and pushes it to AWS ECR"

Prerequisites

- AWS Components

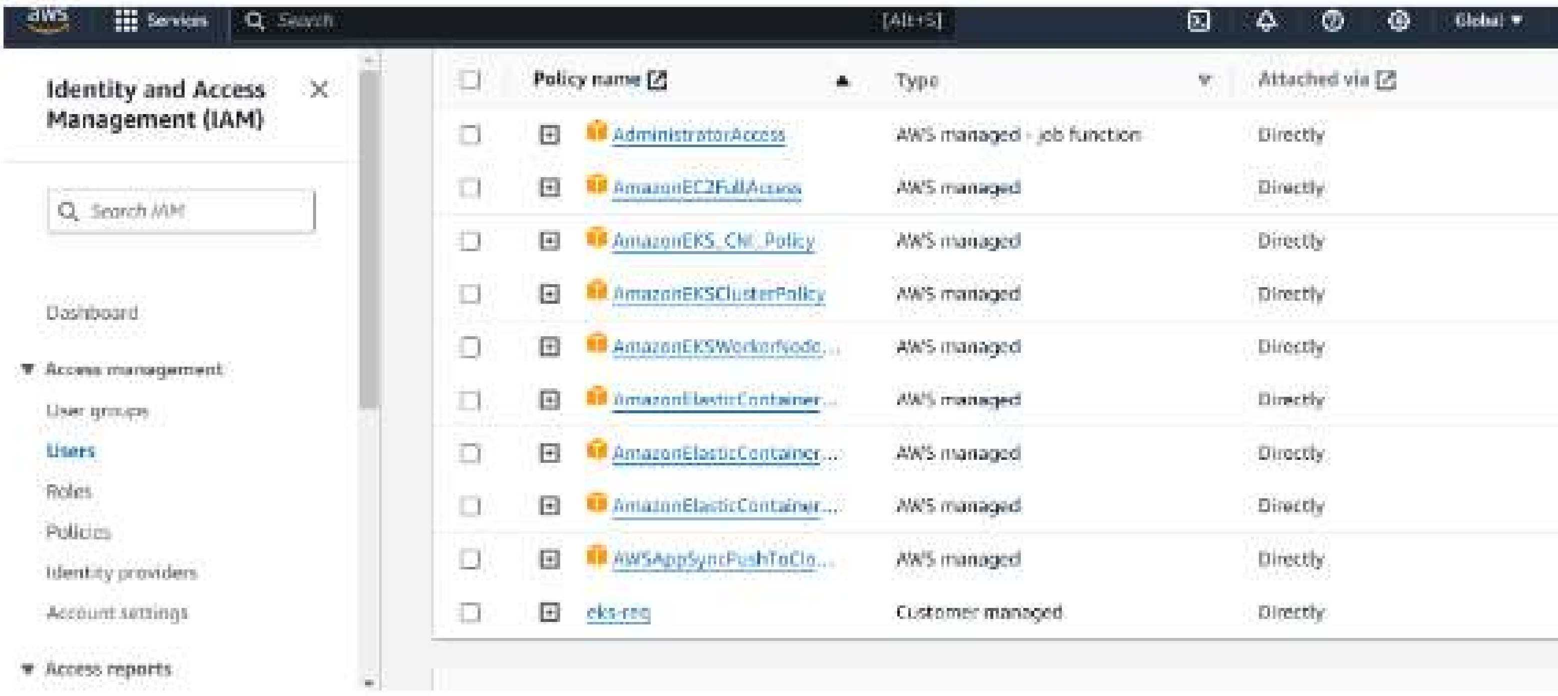
> *Identity and Access Management (IAM)*
> *Elastic Container Registry (ECR)*
> *Elastic Container Service (ECS)*
> *Elastic Compute Cloud (EC2)*

Creating IAM users (console)

You can use the AWS Management Console to create IAM users.

1. Sign in to the AWS Management Console and open the **IAM console**
2. In the navigation pane, choose Users and then choose Add users
3. Type the user name for the new user. This is the sign-in name for AWS
4. Select the type of access the user will have. Programmatic access is enough.
5. Choose Next: **Permissions**
6. **Tags** is Optional, you can skip this
7. Now, **Review** to see all of the choices you made up to this point. When you are ready to proceed, choose **Create user**
8. To save the access keys, choose Download .csv and then save the file to a safe location

```
AmazonEC2FullAccess
AmazonEC2ContainerRegistryFullAccess
AmazonECS_FullAccess
EC2InstanceProfileForImageBuilderECRContainerBuilds
```

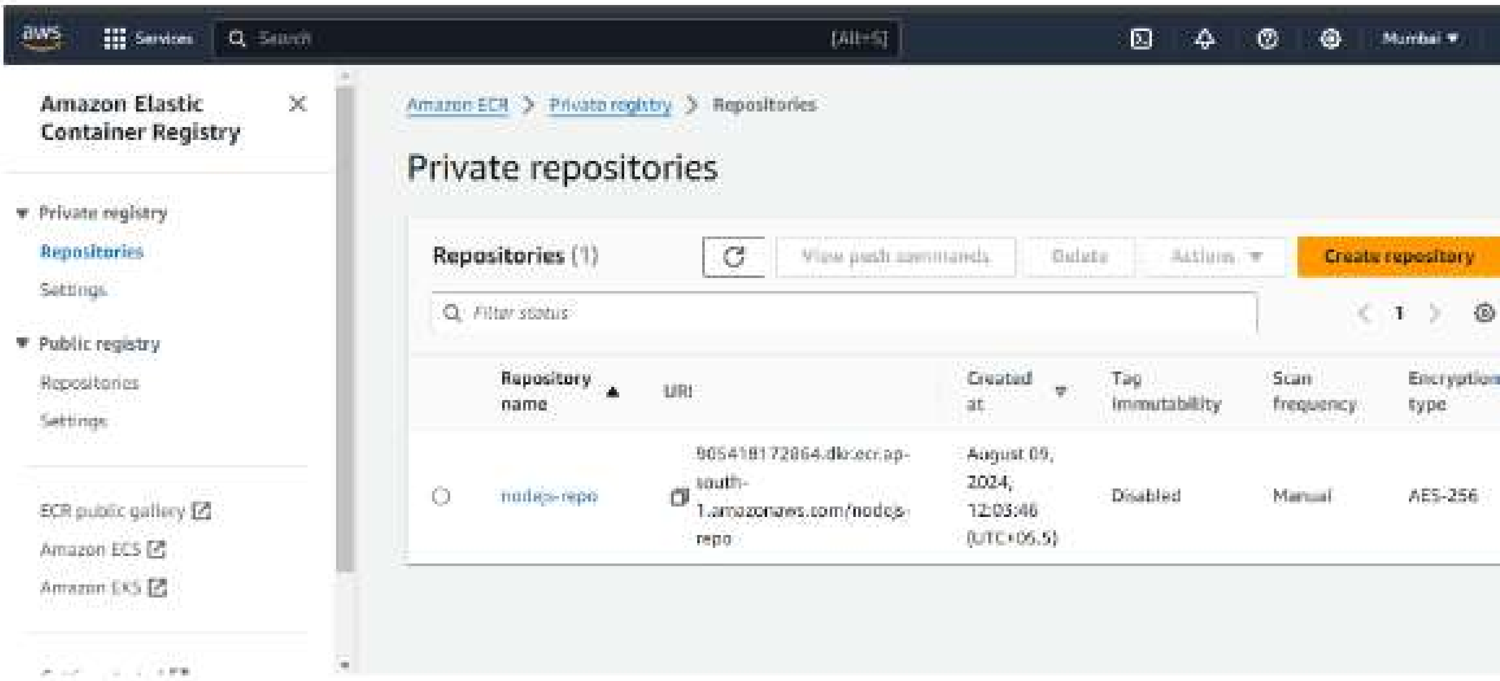


Policy name	Type	Attached via
AdministratorAccess	AWS managed - job function	Directly
AmazonEC2FullAccess	AWS managed	Directly
AmazonEKS_CNI_Policy	AWS managed	Directly
AmazonEKSClusterPolicy	AWS managed	Directly
AmazonEKSWorkerNode...	AWS managed	Directly
AmazonElasticContainer...	AWS managed	Directly
AmazonElasticContainer...	AWS managed	Directly
AmazonElasticContainer...	AWS managed	Directly
AWSAppSyncPushToClo...	AWS managed	Directly
eks-req	Customer managed	Directly

Elastic Container Registry (ECR)

Now we are going to create an image repository

- Open the Amazon ECR console
- Choose to Get Started
- For Visibility settings, choose Private
- For Repository name, specify a name for the repository
- Choose to Create a repository



Private repositories

Repositories (1)

	Repository name	URI	Created at	Tag immutability	Scan frequency	Encryption type
	nodejs-repo	905418172864.dkr.ecr.ap-south-1.amazonaws.com/nodejs-repo	August 09, 2024, 12:03:46 (UTC+05.5)	Disabled	Manual	AES-256

GitHub secrets

Now we are going to put our AWS credentials in GitHub secrets in the working repository.

- Under your repository name, click Settings
- In the left sidebar, click Secrets
- Under Secrets, click on Actions
- Now set New repository secret

```
AWS_ACCESS_KEY_ID= xxxxxxxxxxxxx
AWS_REGION= xxxxxxxxxxxx
AWS_SECRET_ACCESS_KEY=xxxxxxxx
```

For running our CI/CD we need task-definition, it is the requirement for the CI/CD pipeline with GitHub-actions.
Go to the Cluster, click on the "Tasks Definitions " tab, and then open the running "Task". Click on the "JSON" and copy all the JSON text and put into a .json file and push it on GitHub

GitHub Actions

What is GitHub action?

> *GitHub Actions is a continuous integration and continuous delivery platform that allows you to automate your development workflow. GitHub Actions allows you to create, test, and deploy your code all from within GitHub in a fast, safe, and scalable manner. Every time you push, a build is immediately generated and executed, allowing you to quickly test each and every commit.*

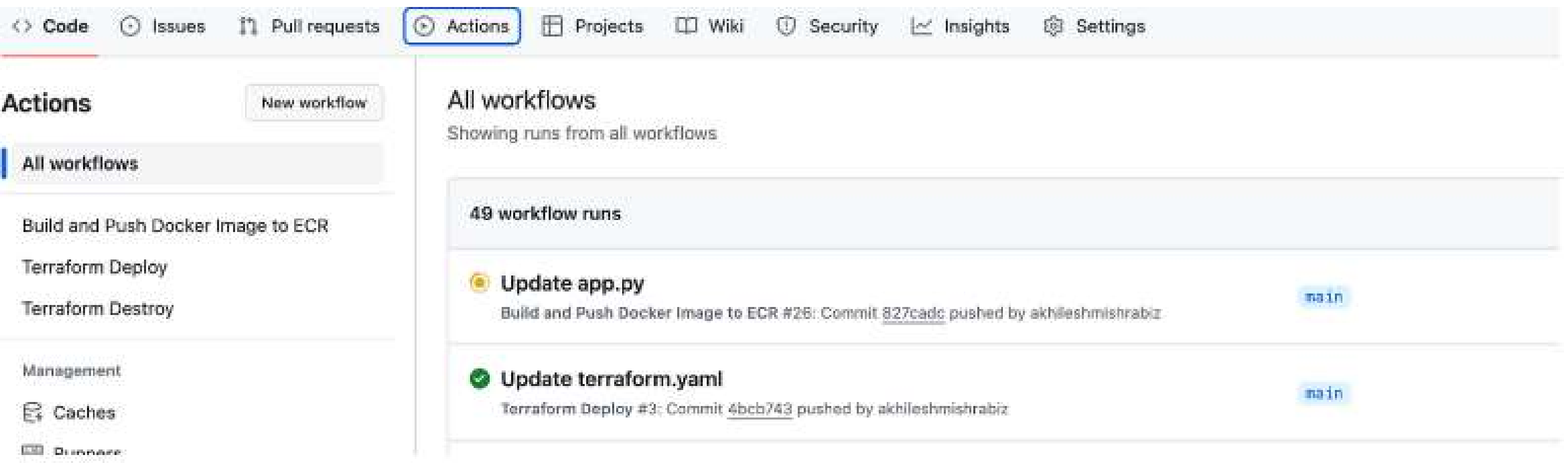
GitHub-Actions **Workflow**:

> *Workflow is a configurable, automated process that we can use in our repository to build, test, package, release, or deploy your project. Workflows are made up of one or more "jobs" and can be triggered by GitHub events*

Create your pipeline with Github Actions
On your GitHub repository select the Actions tab.
In search bar search for **Deploy to Amazon ECS** and configure it.

Environment Variables:

1. AWS_REGION — Operating region of AWS services.
2. ECR_REPOSITORY — Name of the ECR repository that you have created.
3. ECS_SERVICE — Service name of the ECS Cluster.
4. ECS_CLUSTER — Name of the ECS Cluster.
5. ECS_TASK_DEFINITION — Path of the ECS task definition in JSON format which is stored in GitHub repository.
6. CONTAINER_NAME — Docker container name under the ECS task definition.
7. After setting all of this env's start committing the .yaml


Code Issues Pull requests Actions Projects Wiki Security Insights Settings

Actions New workflow

All workflows

Build and Push Docker Image to ECR
Terraform Deploy
Terraform Destroy

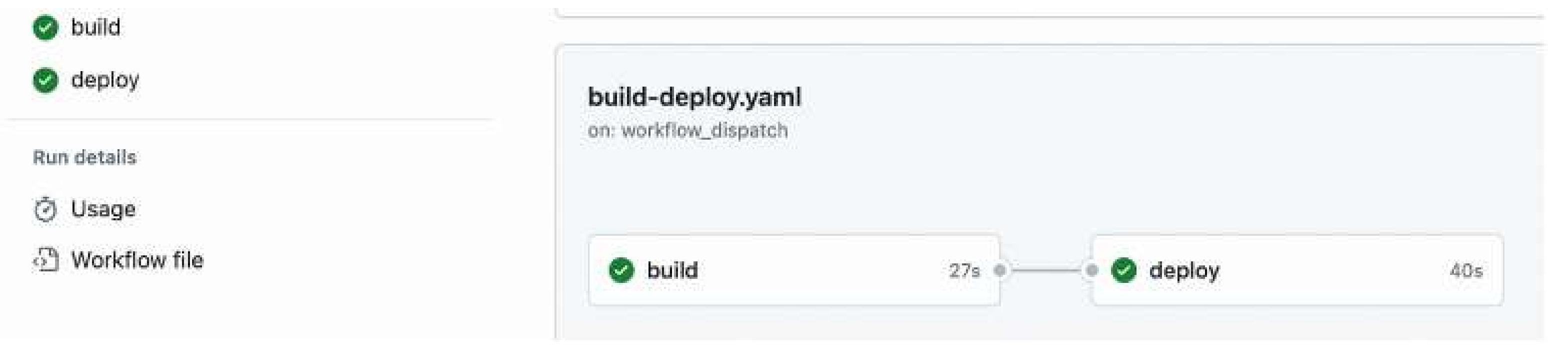
Management
Caches

All workflows
Showing runs from all workflows

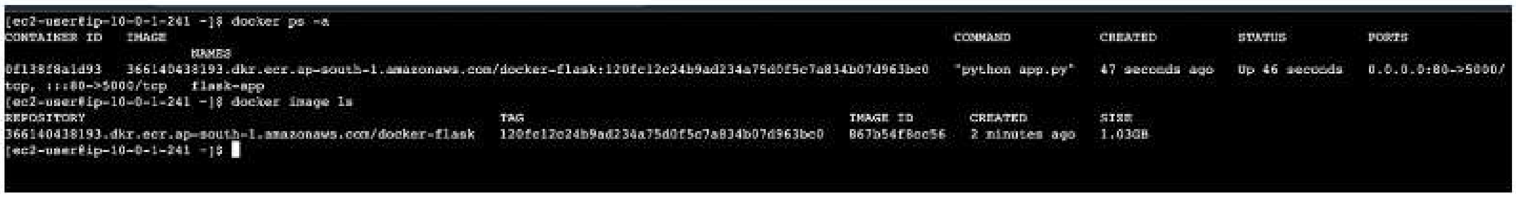
49 workflow runs

Update app.py
Build and Push Docker Image to ECR #26: Commit 827cadc pushed by akhileshmishrabiz main

Update terraform.yaml
Terraform Deploy #3: Commit 4bcb743 pushed by akhileshmishrabiz main



Let’s log in to the EC2 machine and see



```
[ec2-user@ip-10-0-1-241 ~]$ docker ps -a
CONTAINER ID   IMAGE                                                                                                COMMAND           CREATED          STATUS          PORTS
                 NAMES
0f138f8a1d93   366140438193.dkr.ecr.ap-south-1.amazonaws.com/docker-flask:120fc12c24b9ad234a75d0f5c7a834b07d963bc0   "python app.py"   47 seconds ago   Up 46 seconds   0.0.0.0:80->5000/
tcp, :::80->5000/tcp   flask-app
[ec2-user@ip-10-0-1-241 ~]$ docker image ls
REPOSITORY                                                  TAG                                        IMAGE ID       CREATED         SIZE
366140438193.dkr.ecr.ap-south-1.amazonaws.com/docker-flask   120fc12c24b9ad234a75d0f5c7a834b07d963bc0   867b54f8cc56   2 minutes ago   1.03GB
[ec2-user@ip-10-0-1-241 ~]$ 
```