Step-by-Step Guide: Deploy Flask App via GitHub Actions → ECR → ECS (Fargate)

1. AWS Pre-requisites

1.1 Create ECR Repository

```
aws ecr create-repository --repository-name crypto-app
```

1.2 Create ECS Cluster

```
aws ecs create-cluster --cluster-name crypto-cluster
```

1.3 ECS Task Definition & Service - Create a Task Definition (Fargate recommended) - Network mode: awsvpc - CPU/Memory: 256/512 - Execution role: ecsTaskExecutionRole - Container: Flask app container, port mapping 5000:5000 - Create an ECS Service pointing to your task definition - Launch type: FARGATE - Subnets: Public subnets for auto-assigned public IP - Security group: Allow inbound port 5000 - Enable Auto-assign Public IP (optional if not using ALB)

2. GitHub Secrets

Secret Name	Example Value
AWS_ACCESS_KEY_ID	AKIAxxxxxx
AWS_SECRET_ACCESS_KEY	xxxxxxxxxxx
AWS_REGION	us-east-1
ECR_REGISTRY	123456789012.dkr.ecr.us-east-1.amazonaws.com
ECR_REPOSITORY	crypto-app
ECS_CLUSTER	crypto-cluster
ECS_SERVICE	crypto-service

3. GitHub Actions Workflow

Create `.github/workflows/deploy.yml`:

```
name: Deploy to AWS ECS

on:
  push:
```

```
    branches:
      - main

jobs:
  deploy:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout Code
      uses: actions/checkout@v3

    - name: Configure AWS Credentials
      uses: aws-actions/configure-aws-credentials@v2
      with:
        aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
        aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
        aws-region: ${{ secrets.AWS_REGION }}

    - name: Log in to Amazon ECR
      uses: aws-actions/amazon-ecr-login@v1

    - name: Build, Tag & Push Docker Image
      run: |
        IMAGE_TAG=latest
        docker build -t ${{ secrets.ECR_REGISTRY }}/$
{{ secrets.ECR_REPOSITORY }}:$IMAGE_TAG .
        docker push ${{ secrets.ECR_REGISTRY }}/${{ secrets.ECR_REPOSITORY }}:
$IMAGE_TAG

    - name: Deploy to ECS
      uses: aws-actions/amazon-ecs-deploy-task-definition@v1
      with:
        task-definition: ecs-task-def.json
        service: ${{ secrets.ECS_SERVICE }}
        cluster: ${{ secrets.ECS_CLUSTER }}
        wait-for-service-stability: true
```

4. ECS Task Definition Example (`ecs-task-def.json`)

```
{
  "family": "crypto-app-task",
  "networkMode": "awsvpc",
  "requiresCompatibilities": ["FARGATE"],
  "cpu": "256",
  "memory": "512",
```

```
  "executionRoleArn": "arn:aws:iam::123456789012:role/ecsTaskExecutionRole",
  "containerDefinitions": [
    {
      "name": "crypto-app",
      "image": "123456789012.dkr.ecr.us-east-1.amazonaws.com/crypto-app:latest",
      "essential": true,
      "portMappings": [
        {
          "containerPort": 5000,
          "hostPort": 5000
        }
      ]
    }
  ]
}
```

Replace account ID, region, and IAM role ARN with your values.

5. Configure ECS Fargate Task to Get a Public IP

- When creating/updating ECS service:
- Subnets: Public subnets
- Auto-assign Public IP: Enable ✅
- Security group: Allow inbound port 5000
- New tasks will automatically get a public IP.

6. Deployment Flow

1. Push code to main branch → GitHub Actions workflow triggers
2. Docker image builds and pushes to ECR
3. ECS Service updates with new task definition
4. Flask app is live
5. Option 1: Via public IP (changes on redeploy)
6. Option 2 (Recommended): Via ALB DNS → stable URL

7. Access Your App

7.1 Using Public IP - Get task ARN:

```
aws ecs list-tasks --cluster crypto-cluster --service-name crypto-service --
region us-east-1 --query "taskArns[0]" --output text
```

- Get ENI ID:

```
aws ecs describe-tasks --cluster crypto-cluster --tasks <TASK_ARN> --region us-
east-1 --query "tasks[0].attachments[0].details[?
name=='networkInterfaceId'].value" --output text
```

- Get Public IP:

```
aws ec2 describe-network-interfaces --network-interface-ids <ENI_ID> --region
us-east-1 --query "NetworkInterfaces[0].Association.PublicIp" --output text
```

- Access app: `http://<public-ip>:5000`

7.2 Using Application Load Balancer (Recommended) - Create ALB in public subnets → attach to ECS Service - Access app via ALB DNS name (stable URL)

8. Deployment Benefits

- Fully automated CI/CD with GitHub Actions
- Docker image versioning via ECR
- ECS Fargate tasks auto-deployed on updates
- Public IP access for testing OR ALB for stable production URL