Deploying to Amazon Elastic Container Service

In this article

You can deploy to Amazon Elastic Container Service (ECS) as part of your continuous deployment (CD) workflows.

Note: GitHub-hosted runners are not currently supported on GitHub Enterprise Server. You can see more information about planned future support on the GitHub public roadmap.

Introduction

This guide explains how to use GitHub Actions to build a containerized application, push it to Amazon Elastic Container Registry (ECR), and deploy it to Amazon Elastic Container Service (ECS) when there is a push to the `main` branch.

On every new push to `main` in your GitHub repository, the GitHub Actions workflow builds and pushes a new container image to Amazon ECR, and then deploys a new task definition to Amazon ECS.

Note: If your GitHub Actions workflows need to access resources from a cloud provider that supports OpenID Connect (OIDC), you can configure your workflows to authenticate directly to the cloud provider. This will let you stop storing these credentials as long-lived secrets and provide other security benefits. For more information, see "About security hardening with OpenID Connect" and "Configuring OpenID Connect in Amazon Web Services."

Prerequisites

Before creating your GitHub Actions workflow, you will first need to complete the following setup steps for Amazon ECR and ECS:

1. Create an Amazon ECR repository to store your images.

   For example, using the AWS CLI:

   Bash

   ```
   aws ecr create-repository \
       --repository-name MY_ECR_REPOSITORY \
       --region MY_AWS_REGION
   ```

Ensure that you use the same Amazon ECR repository name (represented here by `MY_ECR_REPOSITORY`) for the `ECR_REPOSITORY` variable in the workflow below.

Ensure that you use the same AWS region value for the `AWS_REGION` (represented here by `MY_AWS_REGION`) variable in the workflow below.

2. Create an Amazon ECS task definition, cluster, and service.

   For details, follow the Getting started wizard on the Amazon ECS console, or the Getting started guide in the Amazon ECS documentation.

   Ensure that you note the names you set for the Amazon ECS service and cluster, and use them for the `ECS_SERVICE` and `ECS_CLUSTER` variables in the workflow below.

3. Store your Amazon ECS task definition as a JSON file in your GitHub repository.

   The format of the file should be the same as the output generated by:

   ```
   Bash

   aws ecs register-task-definition --generate-cli-skeleton
   ```

   Ensure that you set the `ECS_TASK_DEFINITION` variable in the workflow below as the path to the JSON file.

   Ensure that you set the `CONTAINER_NAME` variable in the workflow below as the container name in the `containerDefinitions` section of the task definition.

4. Create GitHub Actions secrets named `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` to store the values for your Amazon IAM access key.

   For more information on creating secrets for GitHub Actions, see "Using secrets in GitHub Actions."

   See the documentation for each action used below for the recommended IAM policies for the IAM user, and methods for handling the access key credentials.

5. Optionally, configure a deployment environment. Environments are used to describe a general deployment target like `production`, `staging`, or `development`. When a GitHub Actions workflow deploys to an environment, the environment is displayed on the main page of the repository. You can use environments to require approval for a job to proceed, restrict which branches can trigger a workflow, gate deployments with custom deployment protection rules, or limit access to secrets. For more information about creating environments, see "Using environments for deployment."

Creating the workflow

Once you've completed the prerequisites, you can proceed with creating the workflow.

The following example workflow demonstrates how to build a container image and push it to Amazon ECR. It then updates the task definition with the new image ID, and deploys the task definition to Amazon ECS.

Ensure that you provide your own values for all the variables in the `env` key of the workflow.

If you configured a deployment environment, change the value of `environment` to be the

name of your environment. If you did not configure an environment, delete the `environment` key.

YAML

```yaml
# This workflow uses actions that are not certified by GitHub.
# They are provided by a third-party and are governed by
# separate terms of service, privacy policy, and support
# documentation.

# GitHub recommends pinning actions to a commit SHA.
# To get a newer version, you will need to update the SHA.
# You can also reference a tag or branch, but the action may change without
warning.

name: Deploy to Amazon ECS

on:
  push:
    branches:
      - main

env:
  AWS_REGION: MY_AWS_REGION                   # set this to your preferred AWS
region, e.g. us-west-1
  ECR_REPOSITORY: MY_ECR_REPOSITORY           # set this to your Amazon ECR
repository name
  ECS_SERVICE: MY_ECS_SERVICE                 # set this to your Amazon ECS
service name
  ECS_CLUSTER: MY_ECS_CLUSTER                 # set this to your Amazon ECS
cluster name
  ECS_TASK_DEFINITION: MY_ECS_TASK_DEFINITION # set this to the path to your
Amazon ECS task definition
                                               # file, e.g. .aws/task-
definition.json
  CONTAINER_NAME: MY_CONTAINER_NAME           # set this to the name of the
container in the
                                               # containerDefinitions section of
your task definition

jobs:
  deploy:
    name: Deploy
    runs-on: ubuntu-latest
    environment: production

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

      - name: Configure AWS credentials
        uses: aws-actions/configure-aws-
credentials@0e613a0980cbf65ed5b322eb7a1e075d28913a83
        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
        id: login-ecr
        uses: aws-actions/amazon-ecr-
login@62f4f872db3836360b72999f4b87f1ff13310f3a

      - name: Build, tag, and push image to Amazon ECR
        id: build-image
        env:
          ECR_REGISTRY: ${{ steps.login-ecr.outputs.registry }}
          IMAGE_TAG: ${{ github.sha }}
        run: |
```

```
          # Build a docker container and
          # push it to ECR so that it can
          # be deployed to ECS.
          docker build -t $ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG .
          docker push $ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG
          echo "image=$ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG" >> $GITHUB_OUTPUT

      - name: Fill in the new image ID in the Amazon ECS task definition
        id: task-def
        uses: aws-actions/amazon-ecs-render-task-
definition@c804dfbdd57f713b6c079302a4c01db7017a36fc
        with:
          task-definition: ${{ env.ECS_TASK_DEFINITION }}
          container-name: ${{ env.CONTAINER_NAME }}
          image: ${{ steps.build-image.outputs.image }}

      - name: Deploy Amazon ECS task definition
        uses: aws-actions/amazon-ecs-deploy-task-
definition@df9643053eda01f169e64a0e60233aacca83799a
        with:
          task-definition: ${{ steps.task-def.outputs.task-definition }}
          service: ${{ env.ECS_SERVICE }}
          cluster: ${{ env.ECS_CLUSTER }}
          wait-for-service-stability: true
```

Additional resources

For the original starter workflow, see `aws.yml` in the GitHub Actions `starter-workflows` repository.

For more information on the services used in these examples, see the following documentation:

- "Security best practices in IAM" in the Amazon AWS documentation.
- Official AWS "Configure AWS Credentials" action.
- Official AWS Amazon ECR "Login" action.
- Official AWS Amazon ECS "Render Task Definition" action.
- Official AWS Amazon ECS "Deploy Task Definition" action.