Streamlined Serverless Container Deployment with ECS Fargate and CodePipeline

1. Introduction:

ECS Fargate: AWS ECS (Elastic Container Service) Fargate is a serverless compute engine for containers that allows running Docker containers without managing the underlying infrastructure. With Fargate, you can focus on designing and building your applications without worrying about provisioning, configuring, or scaling the servers that run them.

CodePipeline: AWS CodePipeline is a fully managed continuous integration and continuous delivery (CI/CD) service that automates the build, test, and deployment phases of your release process.

2. Prerequisites:

1. Project Code Repository: (Bitbucket/Github/Code Commit)
2. Docker Image deployed on ECR, ECS Cluster and ECS Task Definitions already up and running.

3. Table of Contents:

1. Setting up an ECS Cluster and creating task definitions
2. Amazon CodePipeline Building Blocks
3. Add Build Specification file
4. Create New Pipeline
5. Configure the Source Provider

6. Set Up the Build Stage
7.Choose the Deploy Provider
8. Test the Pipeline

---

1. Setting up an ECS Cluster and creating task definitions, service, task:

> Create an Amazon ECR Repository:
Go to ECR and create a repository for your docker image.

> Create and setup an ECS Cluster with Task Definitions:
> Go to ECS and Create an ECS Cluster
> In the ECS dashboard, click on "Task Definitions" and create.

- Select the launch type compatibility as "Fargate".
- Enter a name and optional description for your task definition.
- Configure your container:
- Click on "Add container" to add a container to your task definition.
- Enter a name for your container.
- Enter the image URL for your Docker container.
- Configure CPU and memory settings according to your application's requirements.
- Optionally, configure environment variables, port mappings, and other container settings.
- Click on "Add" to add the container to your task definition.
- Configure task execution role, network mode, and other settings as needed.
- Review your task definition and click on "Create" to create the task definition.
  > Create a Cluster Service
  - Choose the task definition you created earlier.
  - Configure the service:

- Enter a service name.
- Set the number of tasks you want to run.
- Configure the network and load balancer settings if needed.
- Click on "Next step" and review your service configuration.
- Click on "Create Service" to create the service.

> Monitor the Service:
Once the service is created, you can monitor its status and view running tasks in the ECS dashboard.

2. Amazon CodePipeline Building Blocks:

Below is the example AWS CodePipeline building blocks that we will be using in this article where GitHub, CodeBuild, and ECS are included.

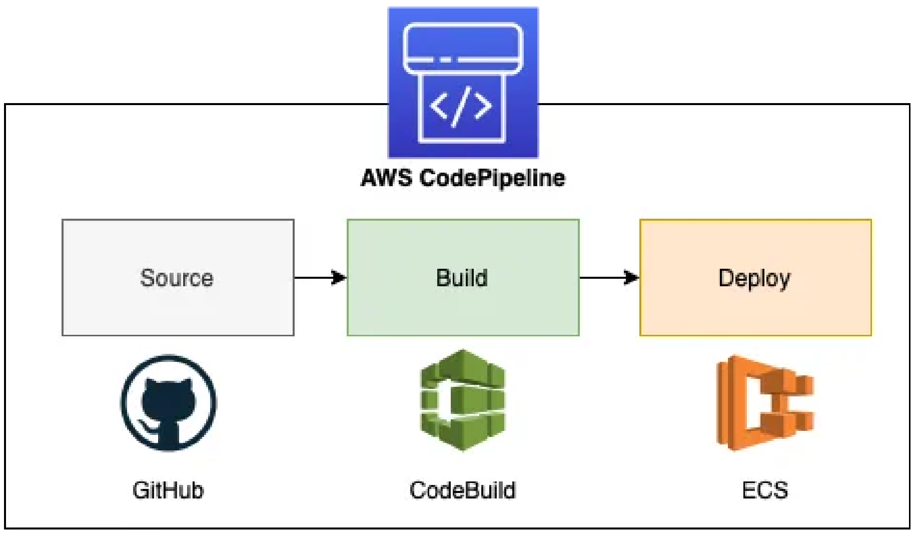


Figure 1: Building Blocks of AWS CodePipeline

There are three main building blocks in the above CodePipeline which are:

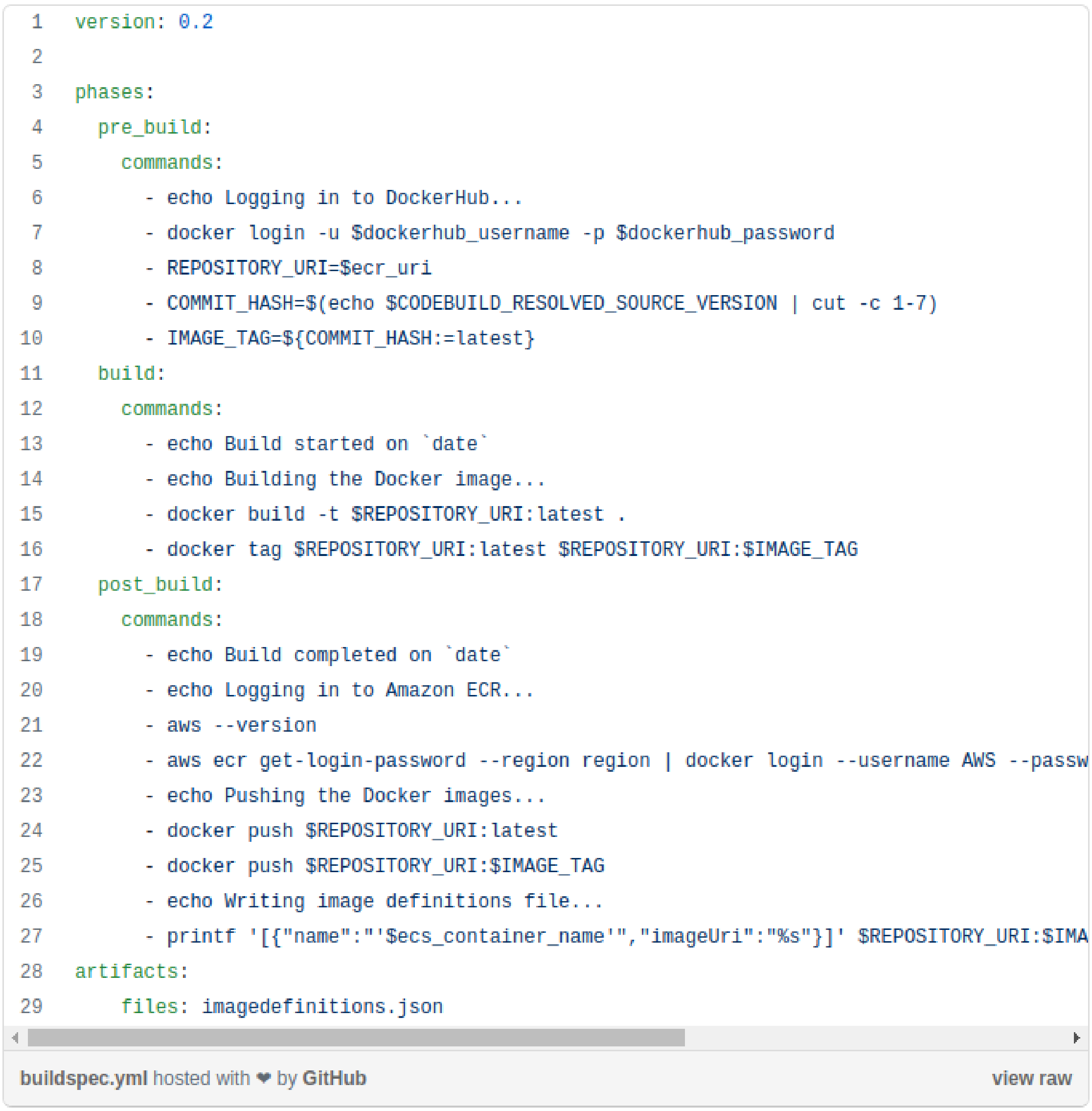
Source Provider: Holds the version of a source change that triggers a pipeline execution. Examples of source providers other than GitHub are Amazon S3, ECR, CodeCommit &, etc.

Build Provider: Compiles the source code, builds docker images, and produces software packages that are ready to deploy.
Deploy Provider: Deploy the updated image to Amazon ECS.

3. Add Build Specification file:

> A buildspec is a series of build commands together with related configurations in YAML format, that CodeBuild uses to run a build.

> We will be creating the buildspec file and locating it in our source code root folder. A sample configuration is as follows:



```yaml
version: 0.2

phases:
  pre_build:
    commands:
      - echo Logging in to DockerHub...
      - docker login -u $dockerhub_username -p $dockerhub_password
      - REPOSITORY_URI=$ecr_uri
      - COMMIT_HASH=$(echo $CODEBUILD_RESOLVED_SOURCE_VERSION | cut -c 1-7)
      - IMAGE_TAG=${COMMIT_HASH:=latest}
  build:
    commands:
      - echo Build started on `date`
      - echo Building the Docker image...
      - docker build -t $REPOSITORY_URI:latest .
      - docker tag $REPOSITORY_URI:latest $REPOSITORY_URI:$IMAGE_TAG
  post_build:
    commands:
      - echo Build completed on `date`
      - echo Logging in to Amazon ECR...
      - aws --version
      - aws ecr get-login-password --region region | docker login --username AWS --passw
      - echo Pushing the Docker images...
      - docker push $REPOSITORY_URI:latest
      - docker push $REPOSITORY_URI:$IMAGE_TAG
      - echo Writing image definitions file...
      - printf '[{"name":"'$ecs_container_name'","imageUri":"%s"}]' $REPOSITORY_URI:$IMA
artifacts:
    files: imagedefinitions.json
```

buildspec.yml hosted with ❤ by **GitHub** **view raw**

> There are few variables which are retrieving values from CodePipeline build stage environment variables. Here are the variables:

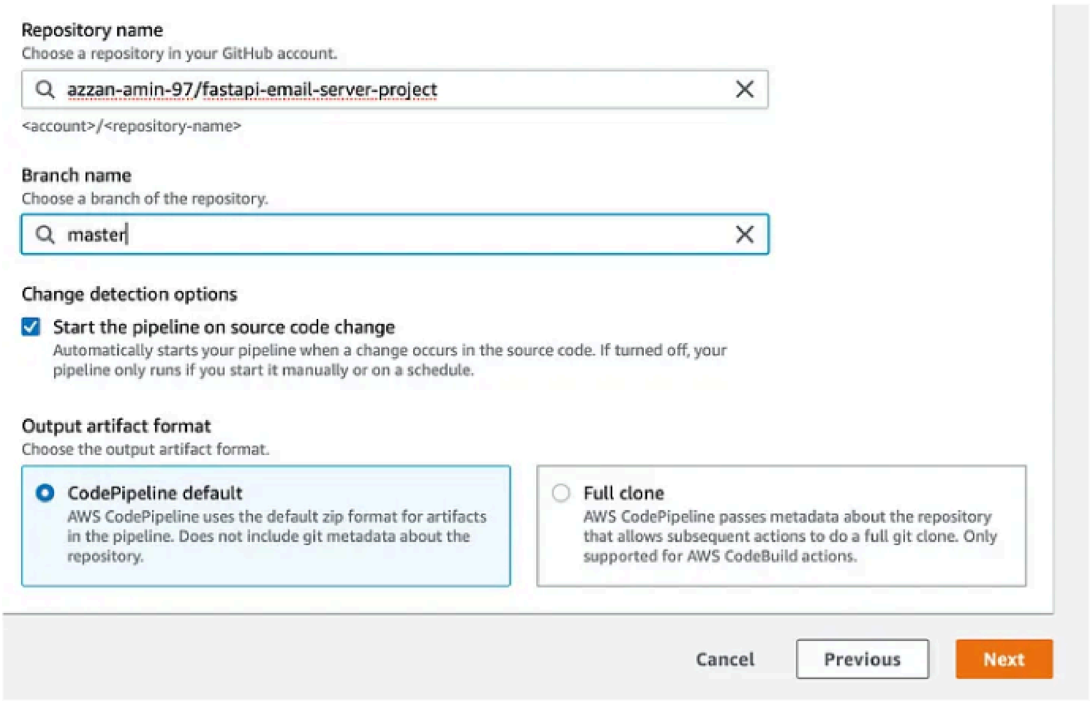
$docker_username
$docker_password
$ecs_container_name
$ecs_uri
Later in the build stage, we will define the values for these variables.

4. Create New CodePipeline:

> Proceed to CodePipeline in the console and click on Create Pipeline.
> Let’s give a name to our pipeline. For service roles, if you do not have any suitable roles for this particular service, just choose the New service role and then click on Next.

5. Configure the Source Provider: GitHub (Let’s say we choose Github for our code):

> In the source stage, choose GitHub Version 2 as the source provider. For connection, we will need to connect our GitHub.
> After successfully connecting to our GitHub Account, we need to select the code repository and the branch and click on Next.


Repository name

Choose a repository in your GitHub account.

azzan-amin-97/fastapi-email-server-project

<account>/<repository-name>

Branch name

Choose a branch of the repository.

master

Change detection options

- [x] Start the pipeline on source code change

  Automatically starts your pipeline when a change occurs in the source code. If turned off, your pipeline only runs if you start it manually or on a schedule.

Output artifact format

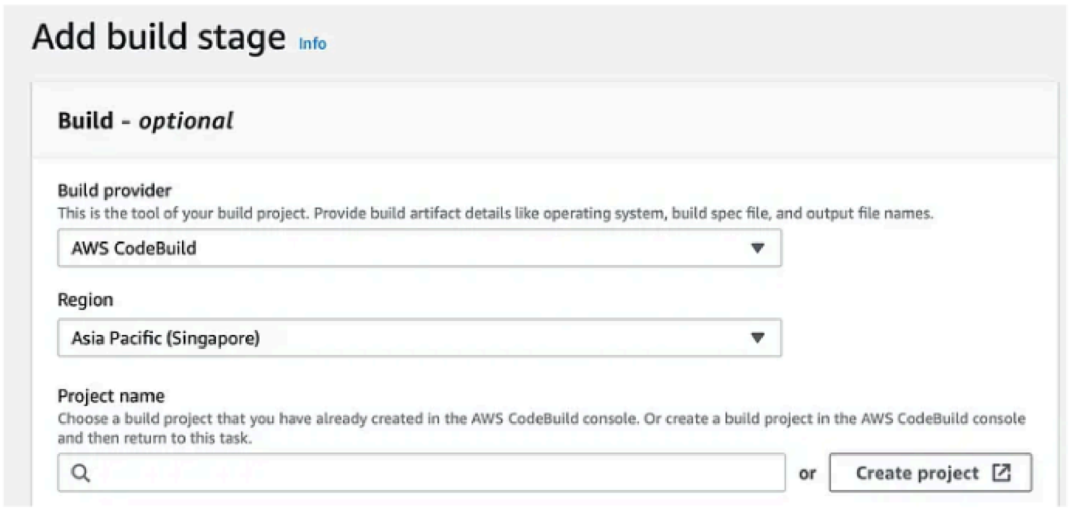
Choose the output artifact format.

- [✓ CodePipeline default] AWS CodePipeline uses the default zip format for artifacts in the pipeline. Does not include git metadata about the repository.
- Full clone: AWS CodePipeline passes metadata about the repository that allows subsequent actions to do a full git clone. Only supported for AWS CodeBuild actions.

Figure 5: Choose repository and branch for the source provider

6. Set Up the Build Stage:

> Add build step by choosing AWS CodeBuild as the build provider and our preferred region. Then, click on Create project.


Add build stage Info

Build - *optional*

Build provider

This is the tool of your build project. Provide build artifact details like operating system, build spec file, and output file names.

AWS CodeBuild

Region

Asia Pacific (Singapore)

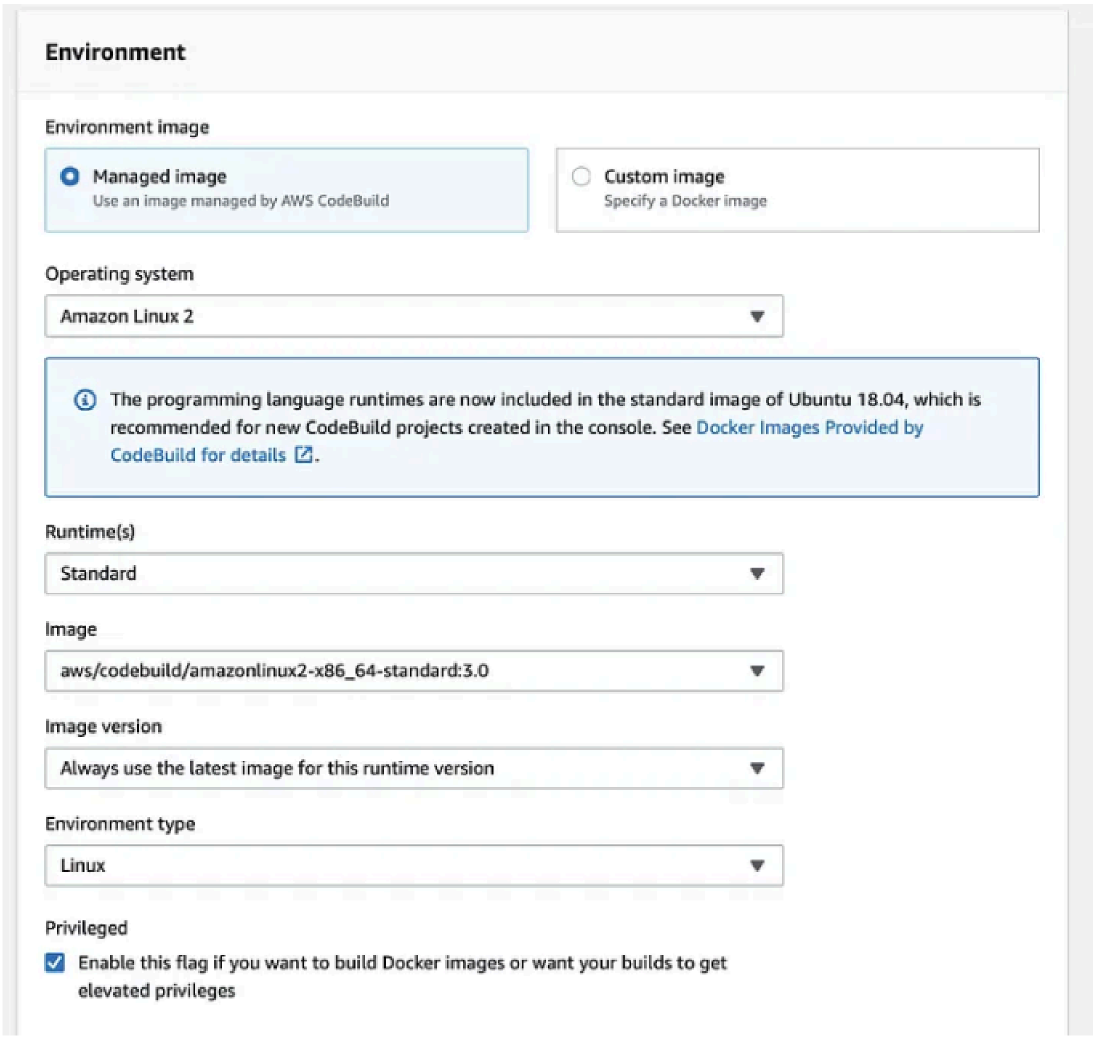
Project name

Choose a build project that you have already created in the AWS CodeBuild console. Or create a build project in the AWS CodeBuild console and then return to this task.

Figure 6: Add Build Stage

> Once we click on the Create project, it will redirect us to a new tab/window. We need to fill in some information for our CodeBuild Project.

> Configure environment for Build: In the Environment section, you can configure the settings as below. We will be using Amazon Linux 2 as our operating system. Please make sure to check the Privileged checkbox in order to allow CodeBuild to build your docker image with elevated privileges.


Environment

Environment image

● Managed image
Use an image managed by AWS CodeBuild

○ Custom image
Specify a Docker image

Operating system
Amazon Linux 2

The programming language runtimes are now included in the standard image of Ubuntu 18.04, which is recommended for new CodeBuild projects created in the console. See Docker Images Provided by CodeBuild for details.

Runtime(s)
Standard

Image
aws/codebuild/amazonlinux2-x86_64-standard:3.0

Image version
Always use the latest image for this runtime version

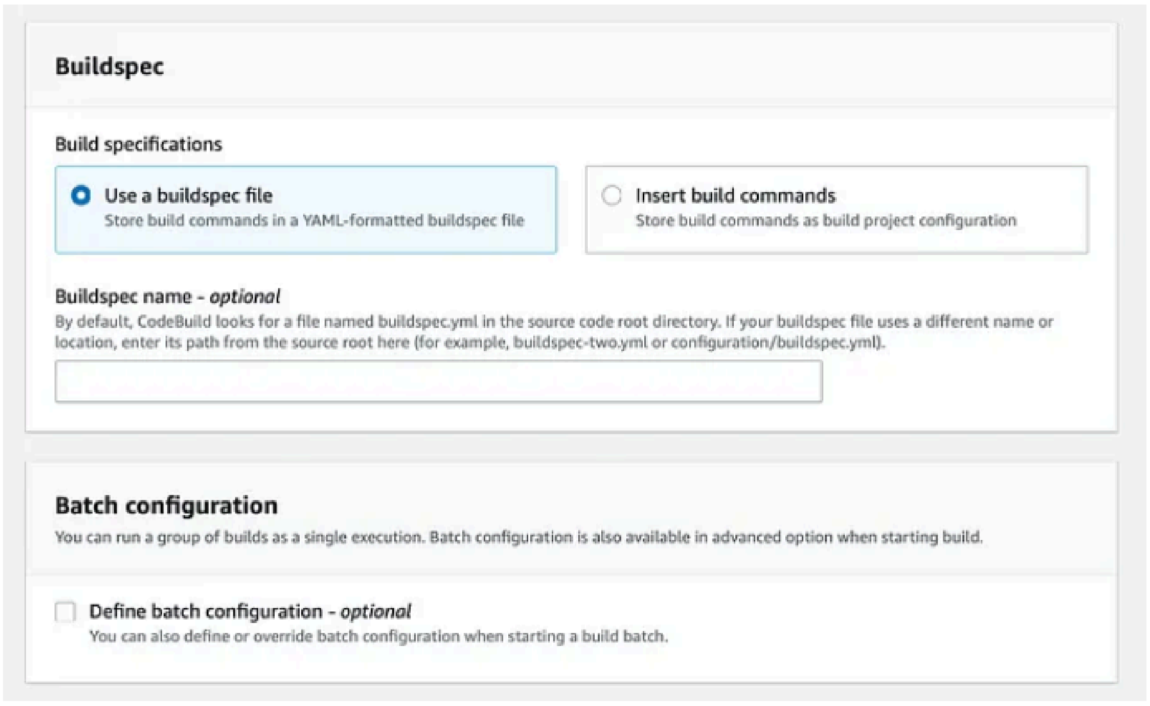
Environment type
Linux

Privileged
✓ Enable this flag if you want to build Docker images or want your builds to get elevated privileges

Figure 8: Build Project Environment

> For service roles, if you do not have any suitable roles for this particular service, just choose the New service role and then click on

Next and leave the other settings as default and proceed to click on **Continue to CodePipeline.**



Buildspec

Build specifications

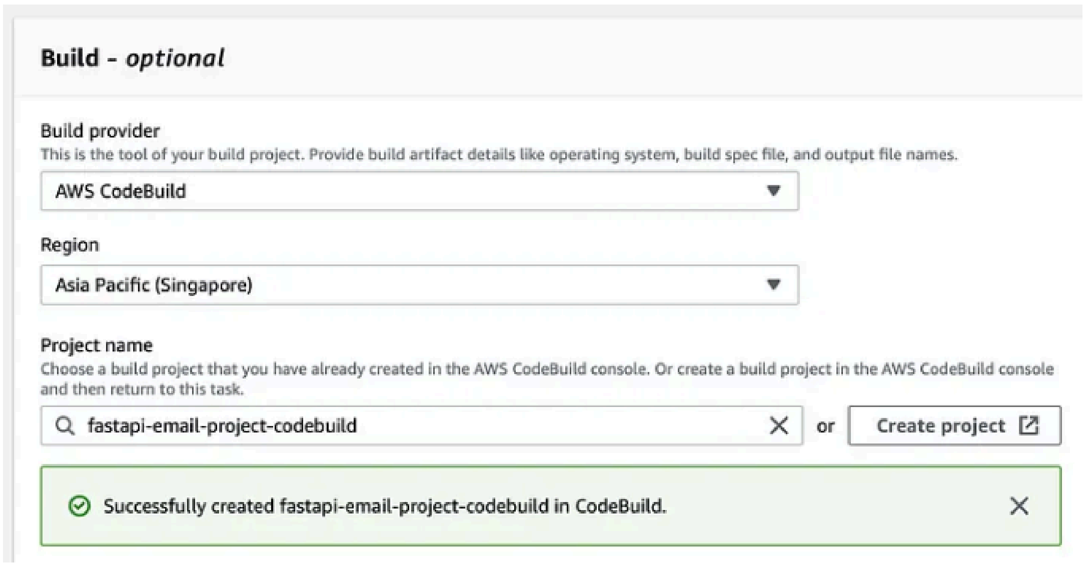
- [✓ Use a buildspec file] Store build commands in a YAML-formatted buildspec file
- Insert build commands Store build commands as build project configuration

Buildspec name - *optional*
By default, CodeBuild looks for a file named buildspec.yml in the source code root directory. If your buildspec file uses a different name or location, enter its path from the source root here (for example, buildspec-two.yml or configuration/buildspec.yml).

Batch configuration
You can run a group of builds as a single execution. Batch configuration is also available in advanced option when starting build.

- Define batch configuration - *optional* You can also define or override batch configuration when starting a build batch.

> Finishing the Build Stage



Build - *optional*

Build provider
This is the tool of your build project. Provide build artifact details like operating system, build spec file, and output file names.
AWS CodeBuild

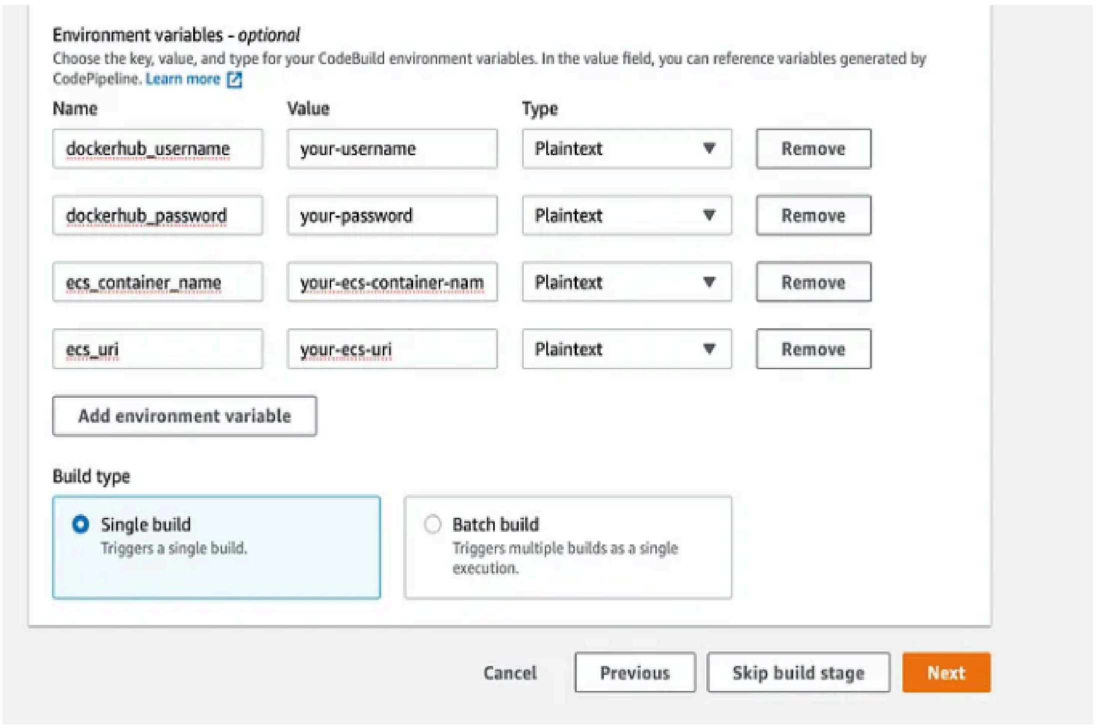
Region
Asia Pacific (Singapore)

Project name
Choose a build project that you have already created in the AWS CodeBuild console. Or create a build project in the AWS CodeBuild console and then return to this task.
fastapi-email-project-codebuild or Create project

Successfully created fastapi-email-project-codebuild in CodeBuild.

Figure 11: Successfully created CodeBuild Project

> Then, we will need to define some environment variables in our build stage as it is essential for our buildspec file as we have been discussed in the section earlier.



Environment variables - *optional*

Choose the key, value, and type for your CodeBuild environment variables. In the value field, you can reference variables generated by CodePipeline. Learn more

Name	Value	Type	
dockerhub_username	your-username	Plaintext	Remove
dockerhub_password	your-password	Plaintext	Remove
ecs_container_name	your-ecs-container-nam	Plaintext	Remove
ecs_uri	your-ecs-uri	Plaintext	Remove

Add environment variable

Build type

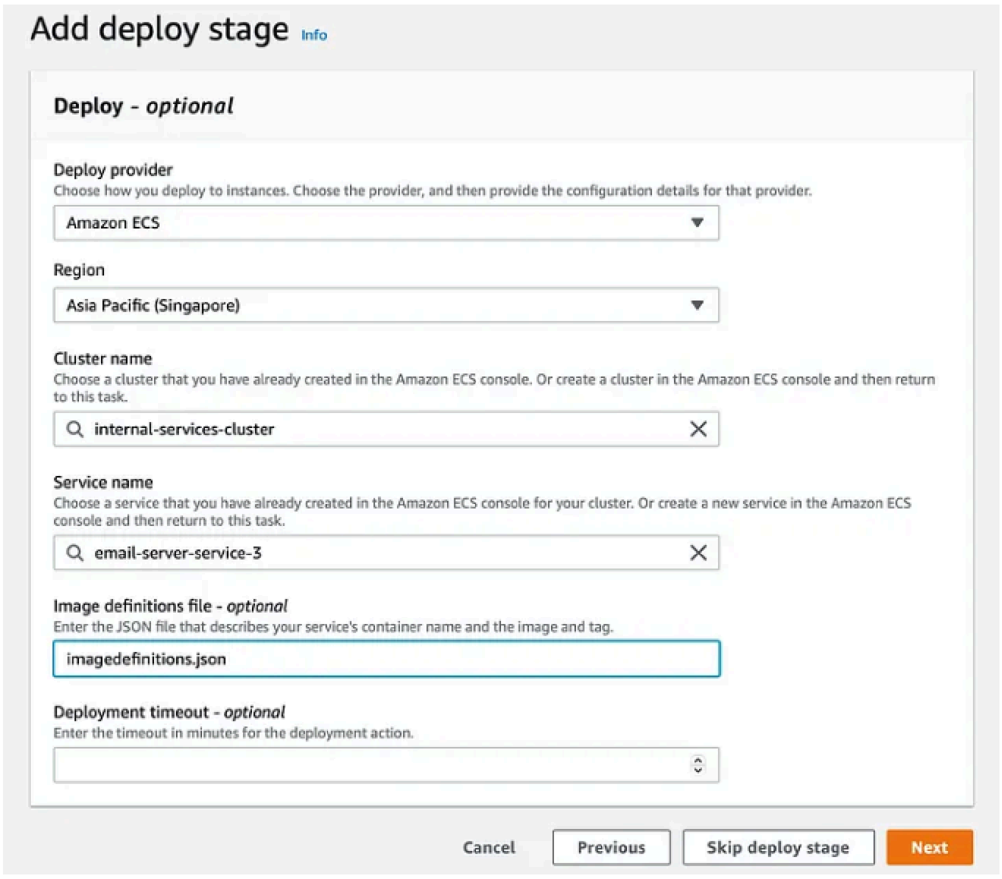
[✓ Single build] Triggers a single build.

Batch build Triggers multiple builds as a single execution.

Figure 12: Build Stage Environment Variables

7. Choose the Deploy Provider:

> Select the region where our cluster resides, and choose the cluster name and service name.
> The image definition file name must be the same as the artifacts file name defined in the buildspec.yaml.


Add deploy stage Info

Deploy - *optional*

Deploy provider
Choose how you deploy to instances. Choose the provider, and then provide the configuration details for that provider.
Amazon ECS

Region
Asia Pacific (Singapore)

Cluster name
Choose a cluster that you have already created in the Amazon ECS console. Or create a cluster in the Amazon ECS console and then return to this task.
internal-services-cluster

Service name
Choose a service that you have already created in the Amazon ECS console for your cluster. Or create a new service in the Amazon ECS console and then return to this task.
email-server-service-3

Image definitions file - *optional*
Enter the JSON file that describes your service's container name and the image and tag.
imagedefinitions.json

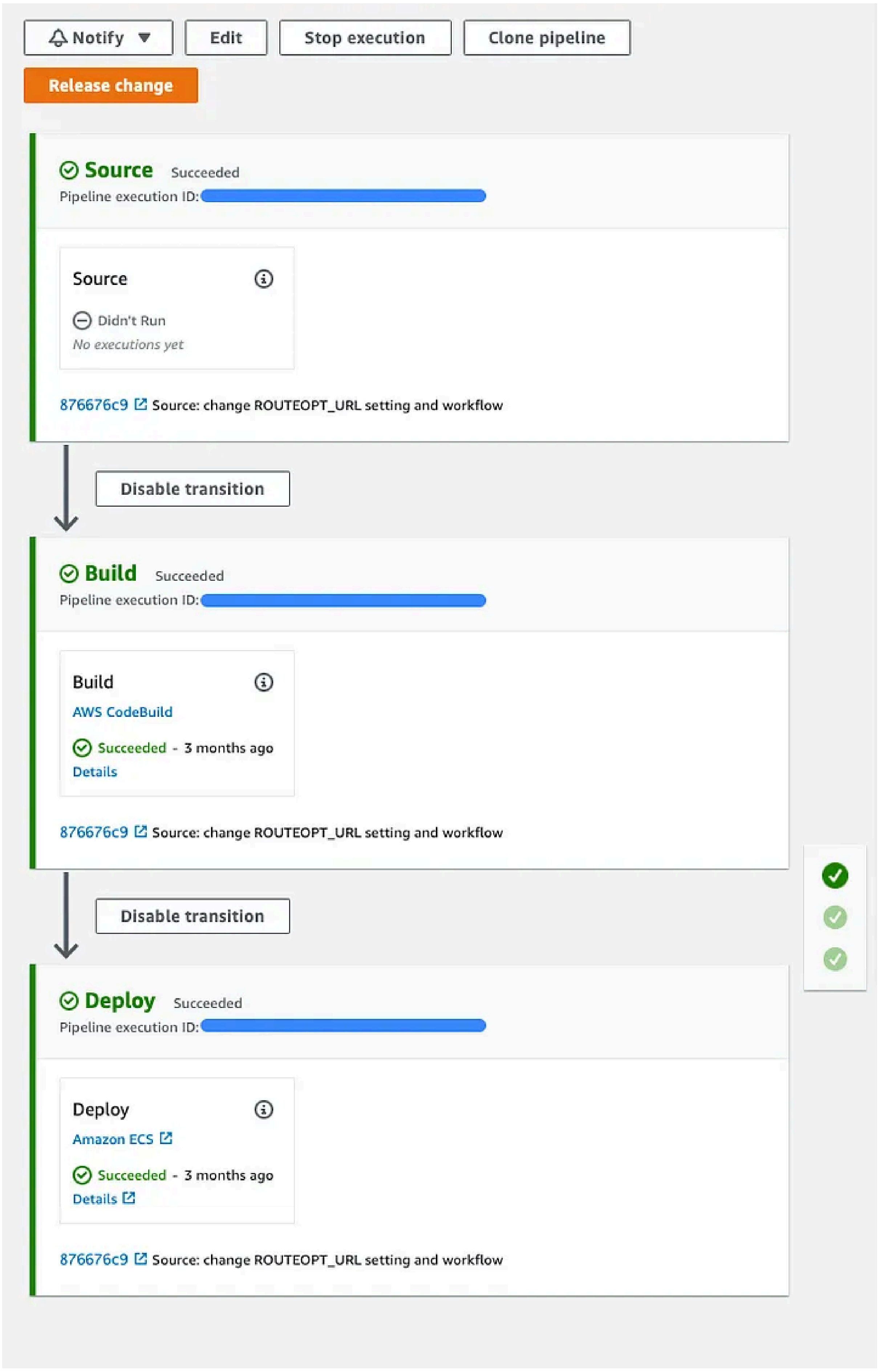
Deployment timeout - *optional*
Enter the timeout in minutes for the deployment action.

Figure 13: ECS as Deploy Provider

> After that, we need to review our pipeline configurations and click on Create Pipeline to complete the pipeline creation.

8. Test the Pipeline:

Our end-to-end native AWS continuous deployment is now set up and now, let's try to push some code changes in our repository.
> Once we push the commit code to GitHub, it will trigger the pipeline.
> Every update that triggers the ECS deployment will be a rolling update by default.
> It means that a new version update will be deployed alongside the previous version of the application. The older version will be terminated once the latest version is stable.
The pipeline will look like:



Source Succeeded
Pipeline execution ID: [redacted]

Source
Didn't Run
No executions yet

876676c9 Source: change ROUTEOPT_URL setting and workflow

Build Succeeded
Pipeline execution ID: [redacted]

Build
AWS CodeBuild
Succeeded - 3 months ago
Details

876676c9 Source: change ROUTEOPT_URL setting and workflow

Deploy Succeeded
Pipeline execution ID: [redacted]

Deploy
Amazon ECS
Succeeded - 3 months ago
Details

876676c9 Source: change ROUTEOPT_URL setting and workflow

We have successfully create your own continuous deployment pipeline for your very own applications.

9. Conclusion:

Setting up AWS ECS Fargate with CodePipeline provides a scalable and automated solution for serverless deployment of containerized applications. By following this guide and incorporating best practices, we can streamline your development and deployment processes on AWS.