**AWS END TO END CLI PROJECT – PART 1 Continuous Integration**



DevOps Pipeline Using AWS Services – CI/CD Flowchart

This image is a well-structured flowchart that visually represents a Continuous Integration (CI) and Continuous Delivery (CD) pipeline using AWS services, set against a dark green background for contrast and clarity.

🔹 Starting Point

• User initiates the process by performing a Code Commit.

• This commit is sent to AWS CodeCommit, represented by a green cube logo.

🔹 Pipeline Orchestration

• The commit triggers AWS CodePipeline, which orchestrates the entire CI/CD workflow.

• CodePipeline is shown with a green icon of stacked horizontal bars.

🔧 Continuous Integration Section

Labeled in bold yellow text, this section includes:

• Check out

• Build & UT (Unit Testing)

• Code Scan

These steps are executed by AWS CodeBuild, which is centrally placed and visually connected to the CI steps.

🚀 Continuous Delivery Section

Also labeled in bold yellow, this section includes:

• Image Build

• Image Scan

• Image Push

These steps represent the containerization and deployment flow, likely involving Docker and image repositories.

Visual Highlights

• Clear separation between CI and CD phases

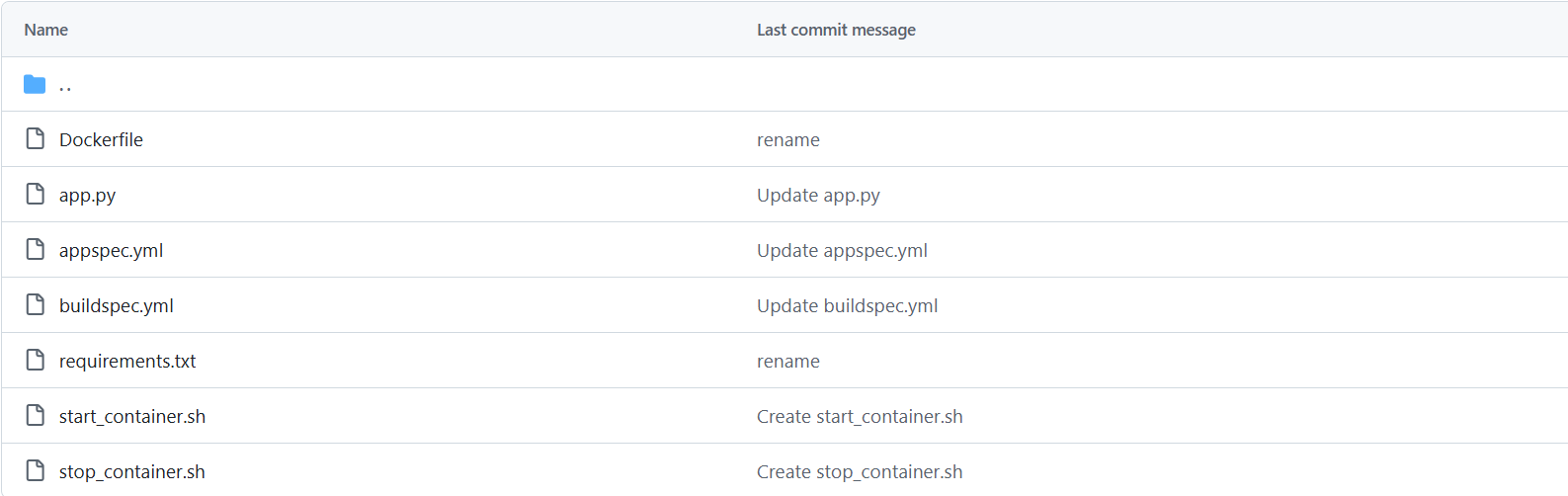
• Use of AWS service logos for intuitive recognition

• Yellow arrows and labels guide the viewer through the process

• Clean layout with white rectangles and black text for each step

🚀 Step 1: Set Up Your GitHub Repository

Before we dive into Continuous Integration (CI), we need a version-controlled home for our Python application. If you already have a repository, feel free to skip ahead. Otherwise, follow these steps to create one:



🔧 Create a New GitHub Repository

1. Visit github.com and sign in to your account.

2. Click the "+" icon in the top-right corner and select "New repository."

3. Enter a repository name (e.g., ) and an optional description.

4. Choose the visibility:

• Public: Open-source and visible to everyone.

• Private: Restricted to collaborators only.

5. Check the box to initialize with a README (recommended).

6. Click "Create repository" to finalize.

📁 Recommended Initial Files

Once your repo is created, consider adding:

• – Your main Python application

• – Python dependencies

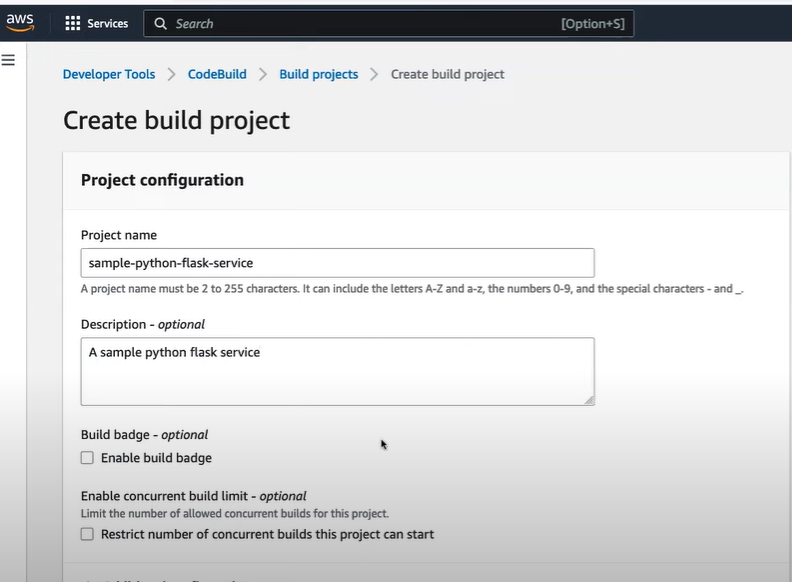
• – For containerization

• – For CI configuration (e.g., AWS CodeBuild)

• – For deployment (e.g., AWS CodeDeploy)

• and – Shell scripts for container lifecycle

🚀 Step 2: AWS Code Build Setup



AWS CodeBuild interface for creating a build project

• Section: Project configuration

• Project name:

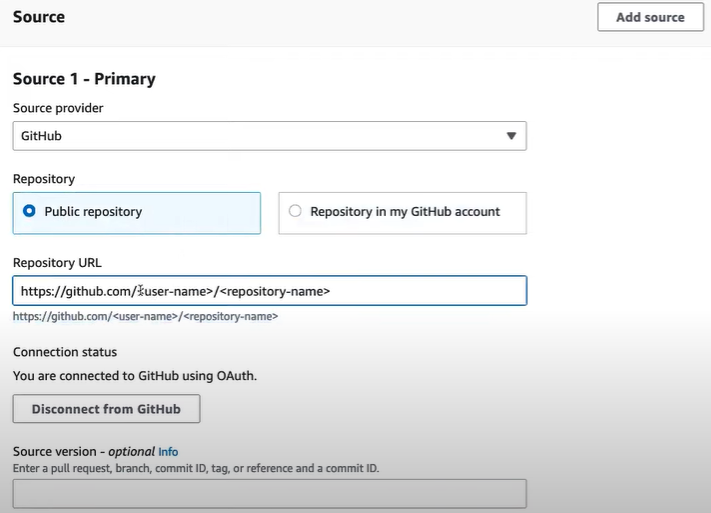
• Description: A sample python flask service

• Options available:

• Enable build badge (unchecked)

• Restrict concurrent builds (unchecked)

• Navigation path: Developer Tools > CodeBuild > Build projects > Create build project



Section: Source 1 – Primary

• Source provider selected: GitHub

• Repository type: Public repository

• Repository URL placeholder:

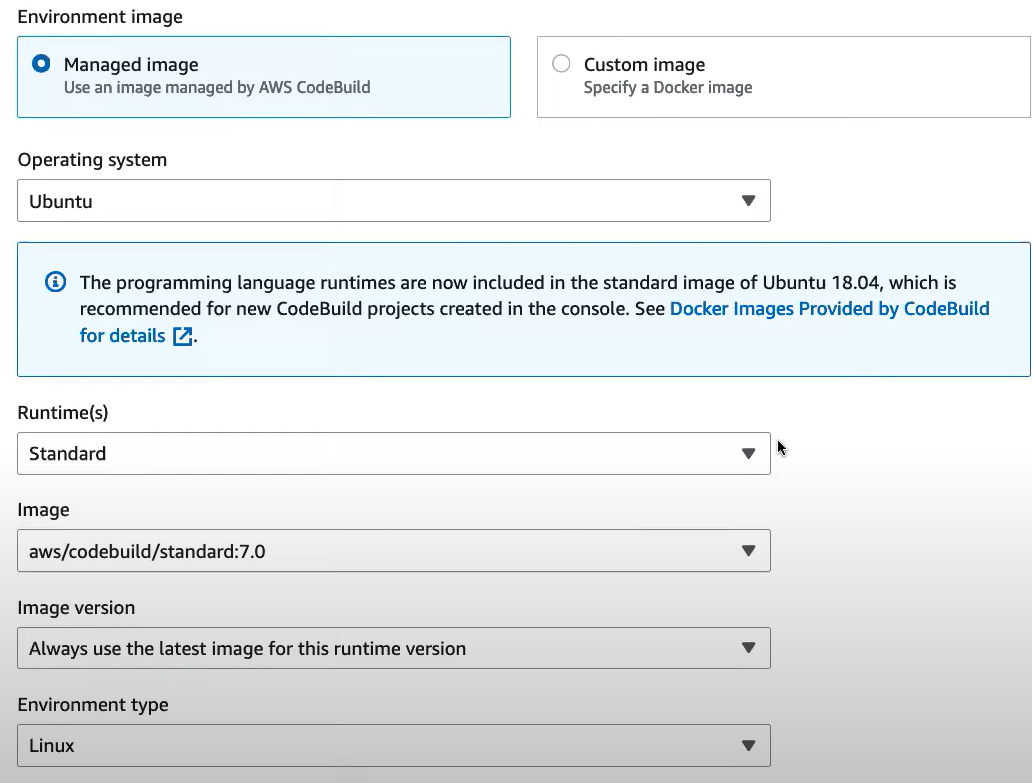
• Connection status: Connected to GitHub via OAuth

• Option to disconnect from GitHub

• Section: Version source – optional info

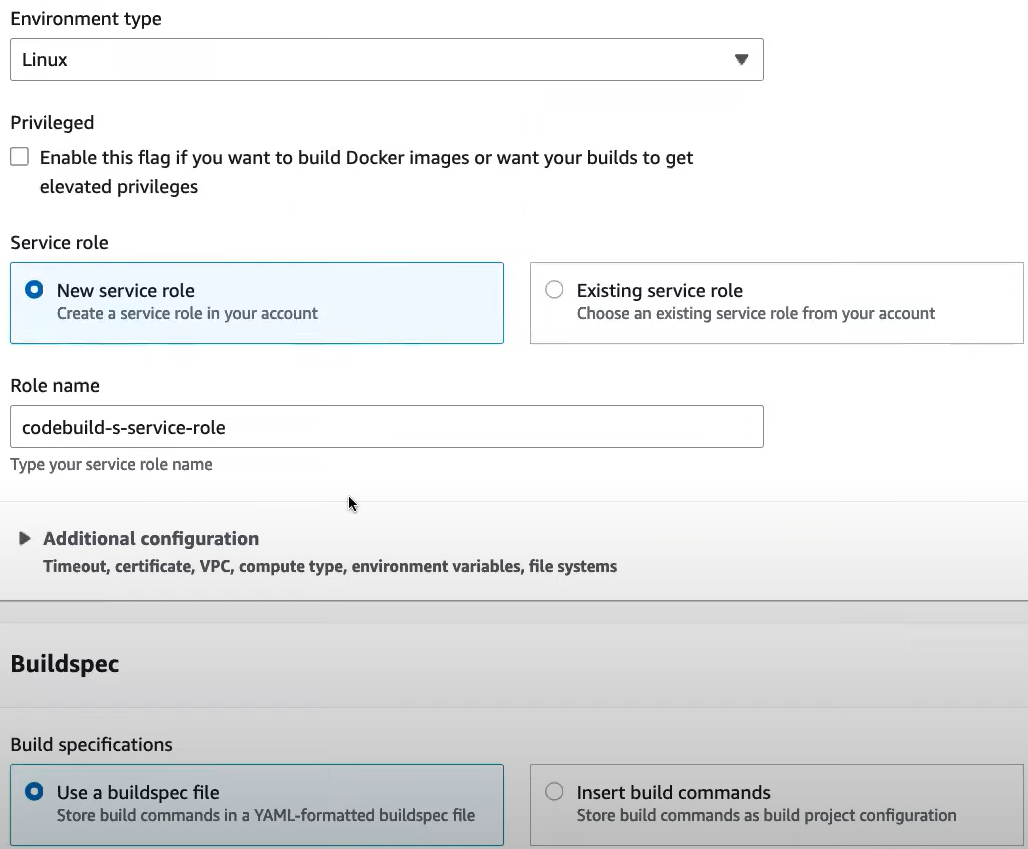
• Field to enter branch, PR, commit ID, tag, or reference

• Button in top-right: Add source



 Section: **Environment image configuration** in AWS CodeBuild

* Selected image type: **Managed image**
* Operating system: **Ubuntu**
* Runtime: aws/codebuild/standard:7.0
* Image version: **Always use the latest**
* Environment type: **Linux**
* Info message: Ubuntu 18.04 includes standard runtimes, recommended for new projects



Section: Build environment configuration in AWS CodeBuild

• Environment type: Linux

• Privileged mode: Option available to enable Docker builds or elevated privileges

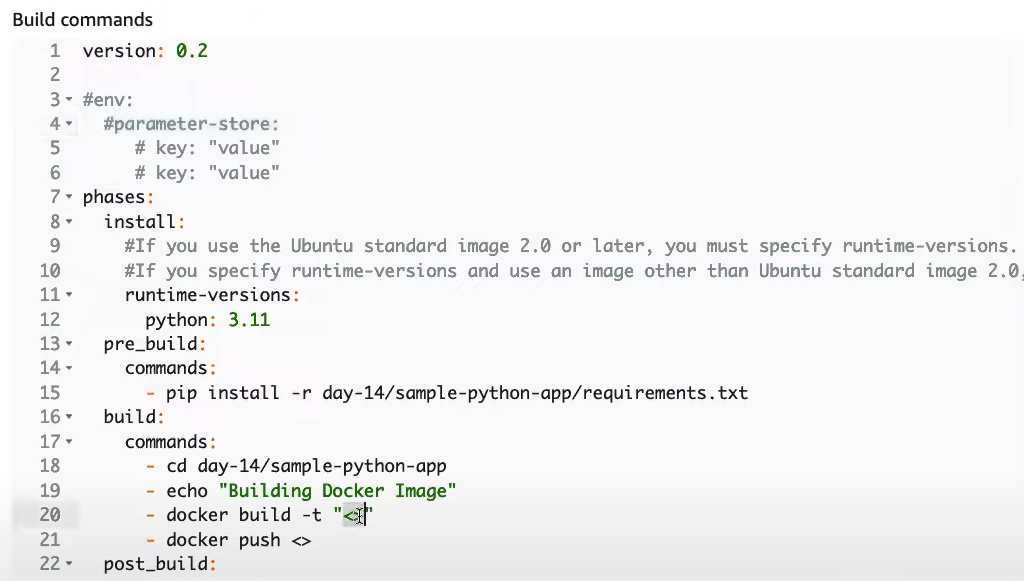
• Service role:

• Selected: New service role

• Role name entered:

• Option to choose an existing role also available

• Additional configuration: Expandable section for VPC, compute type, environment variables.





✅ Buildspec file used: YAML format ()

• 🐍 Runtime version: Python 3.11 specified

• 📦 Dependency installation:

• 🐳 Docker build phase:

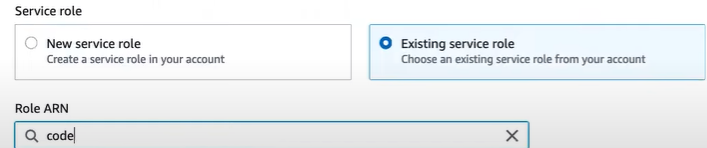
• Build Docker image

• Push image to registry

• 📣 Post-build message:

• 🔄 CI flow includes: install → build → post\_build phases

• 📁 Artifacts section: Specifies files to be packaged after build



Section: Service role setup

• Selected option: Existing service role

• Role input field: Partially typed value

• Purpose: Assign IAM role permissions for CodeBuild project

• Alternate option available: Create new service role

• Role ARN field used to search or specify existing role name

After that click on create Build Project :



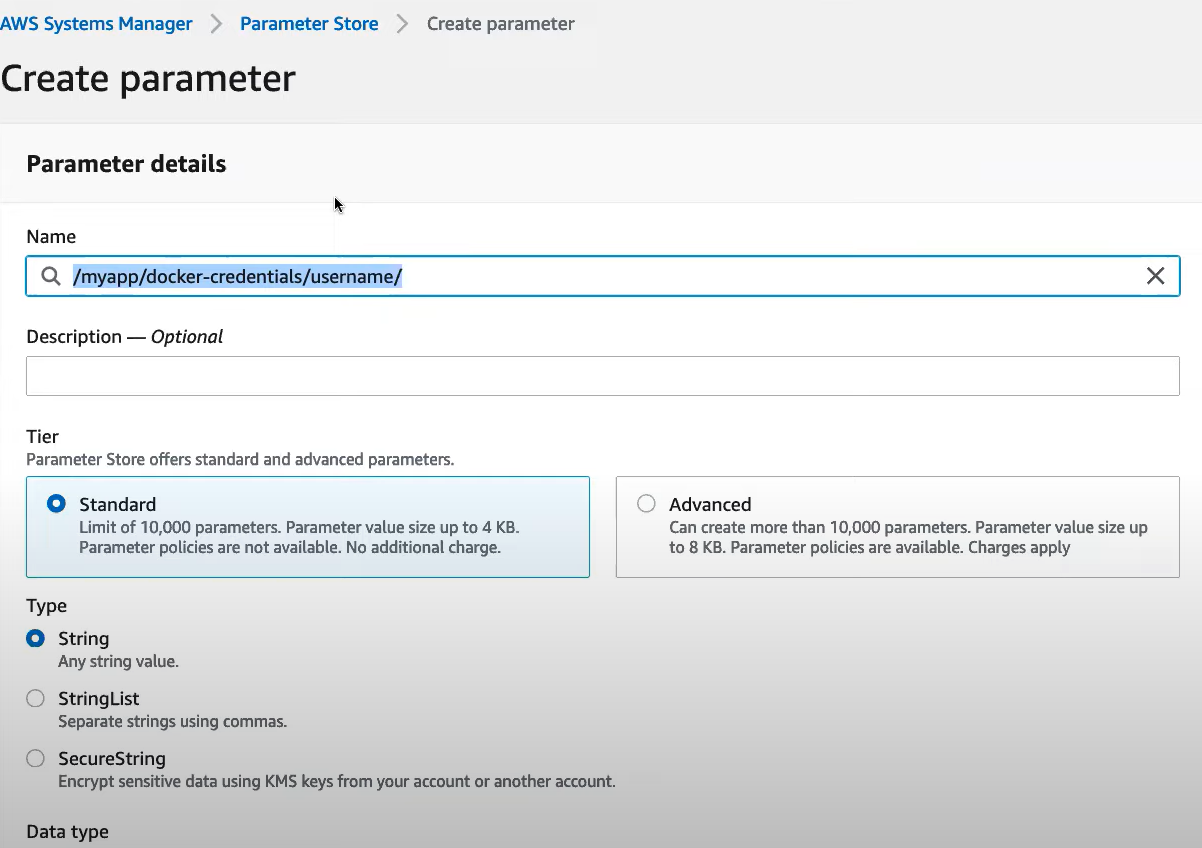
Secure Parameter Creation

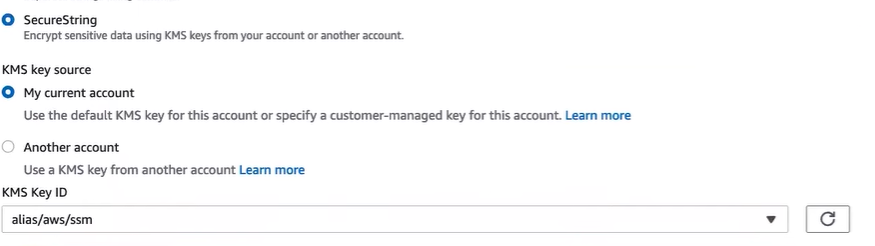
• Parameter name:

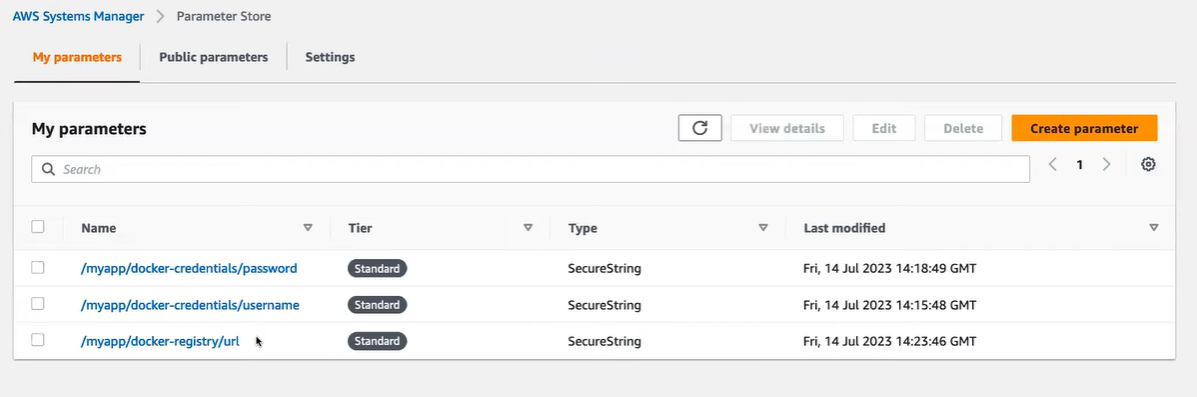
• Tier selected: Standard

• Type selected: String (plain text)

• Optional fields: Description, concurrent build limits







🔐 SecureString with KMS Encryption

• Type: SecureString for sensitive data

• KMS key source: My current account

• KMS Key ID: alias/aws/ssm (default key used)

• Option to use a key from another account available

📋 Stored Parameters Overview

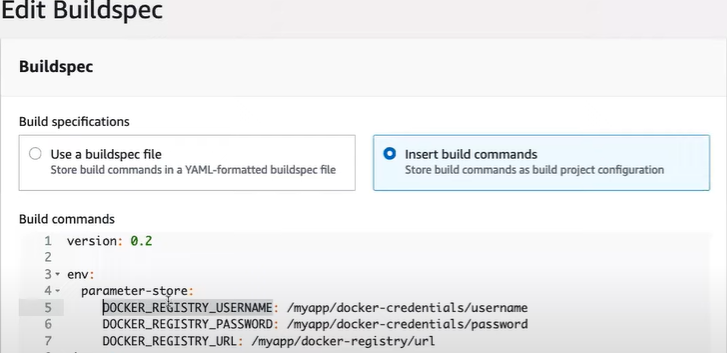
• Parameters listed:

* /myapp/docker-orderdetails/password
* /myapp/docker-orderdetails/username
* /myapp/docker-registry/url

All are:

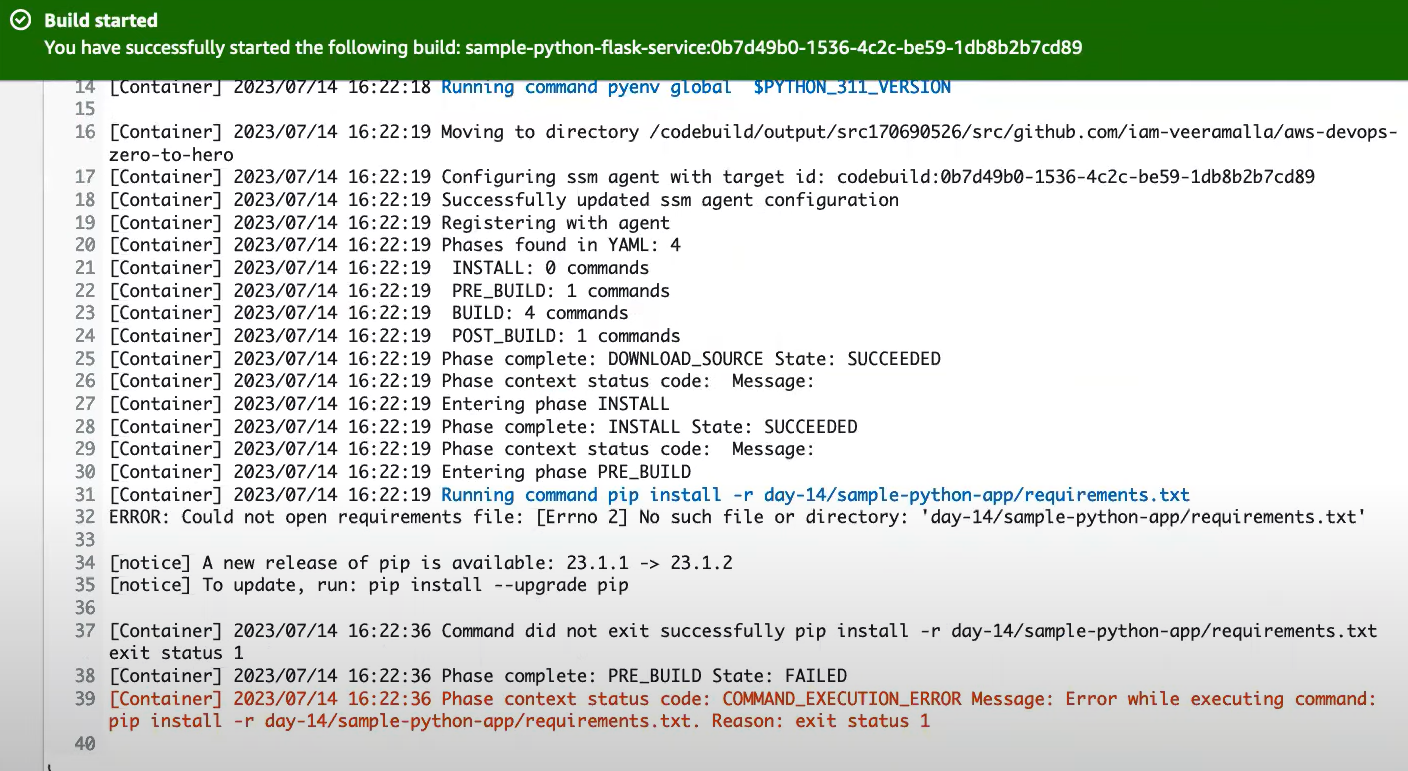
• Tier: Standard

• Type: SecureString





Start the Build after updating the details. Pipeline will definitely fail. To overcome the failure we need to fix the below errors

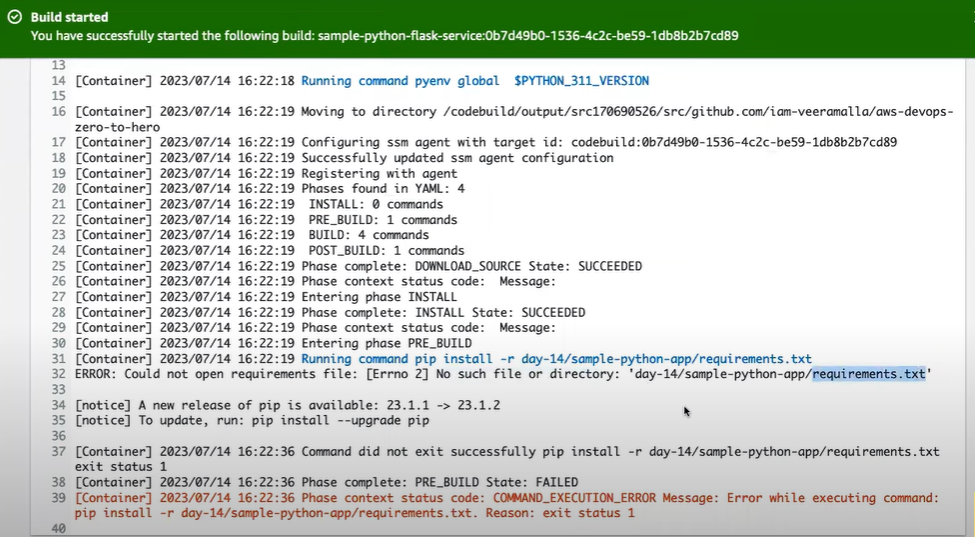


To fix the above error

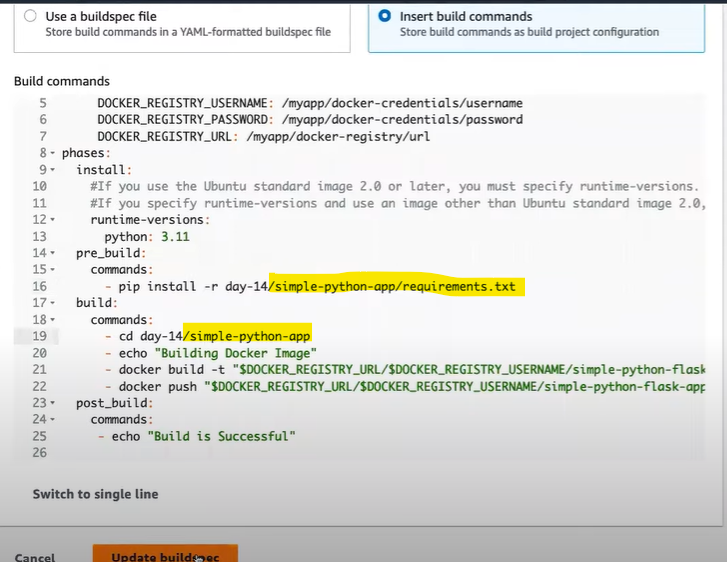
Add permission policies to the IAM user ( AmazonSSMFullAccess )



Error 2 : No such file or directory line 32



To fix the error go to edit the build details and make it to simple as shown in the below image

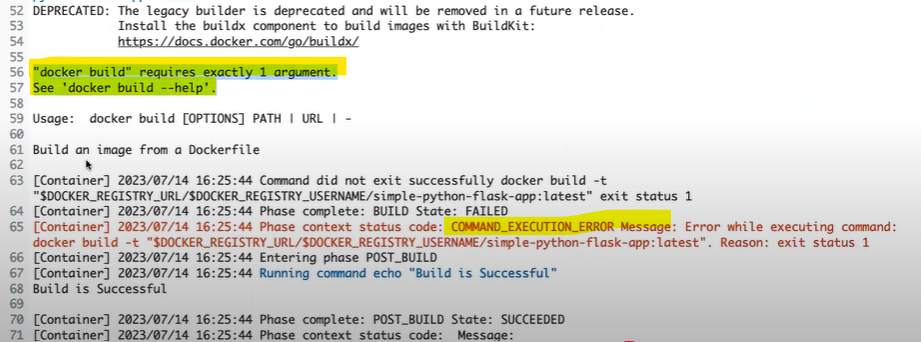


Update the buildsec and re run the AWS build .

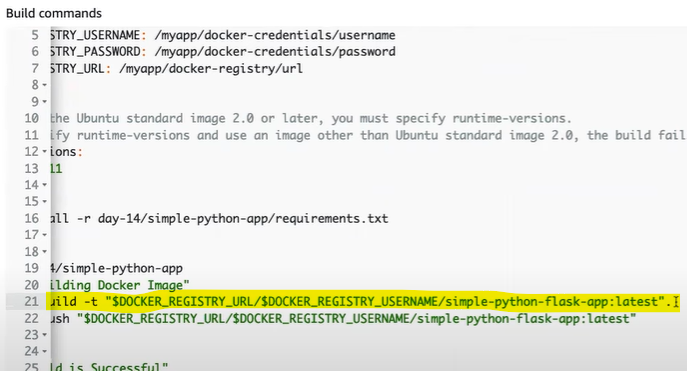
Now we need run manually the AWS CodeBuild . After the project we will integrate it with AWS codepipeline , it will then act as an orchestrator and then invoke the AWS code build . Whenever someone make changes to the AWS code commit or Github it will call AWS code pipeline and code pipeline will automatically call the AWS code build.

After executing the AWS code build the build will fail since we will receive one more error for the docker image build as it cannot execute the docker command

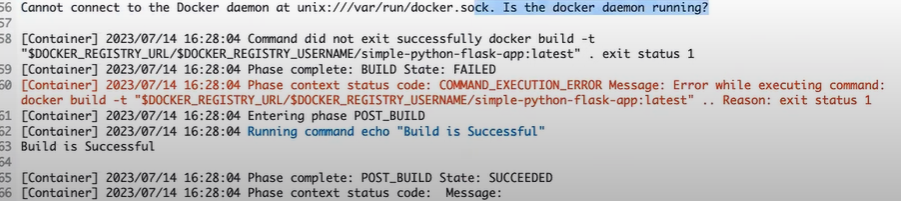
Error 4 : Line no 65

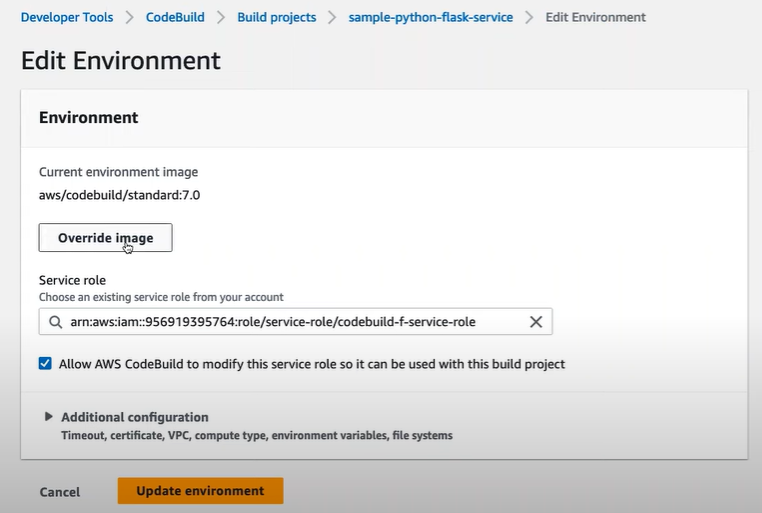


Solution : In build commands go the AWS code script in line 21 dot is not added which is why it did not fetch the docker image

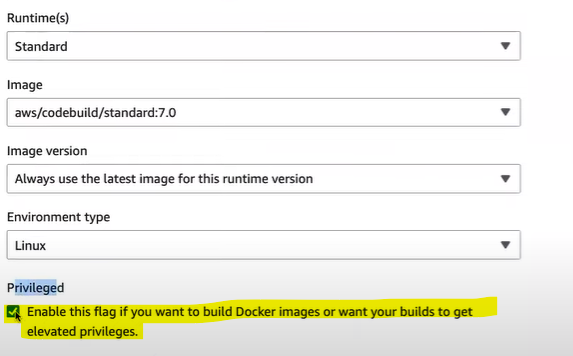


After re running the code build we will see an expected error as shown in the below image . To fix this we need to grant additional permissions in the AWS code build

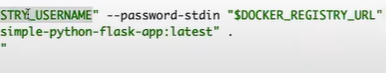
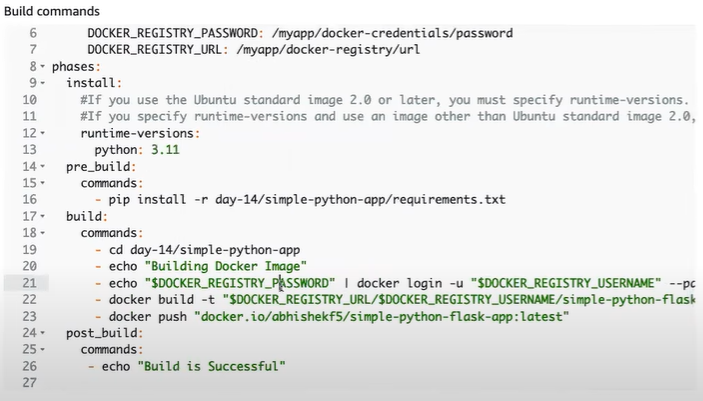




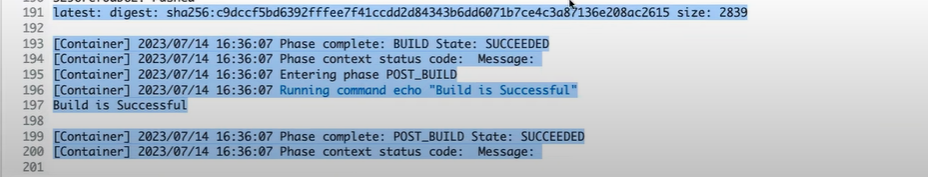
Normally just enable the privileged buttion so that it enables the flag that wants to build the docker images



Check the build commands once before running the AWS code build .The final AWS code build is the proper one

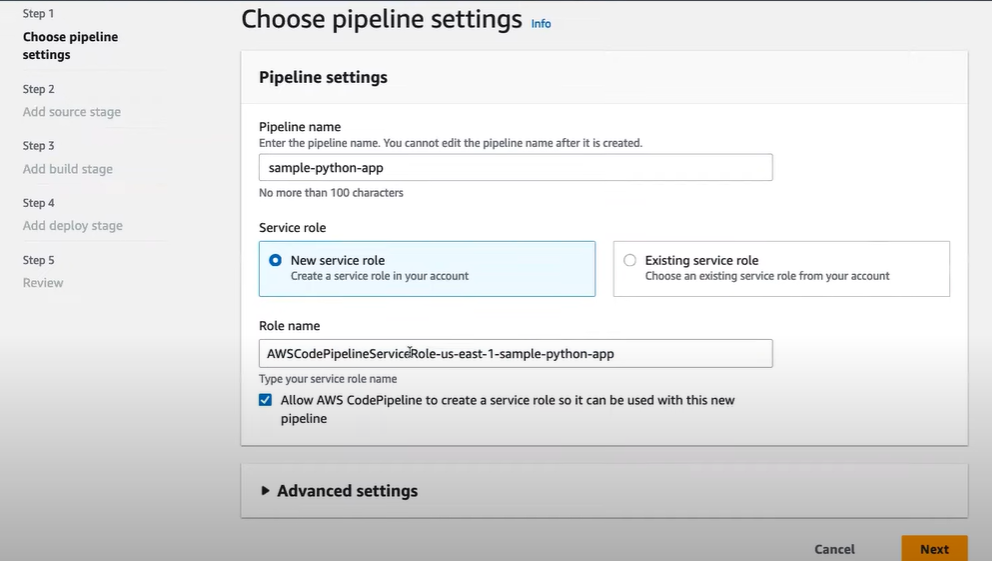


AWS Code build is complete and successful :

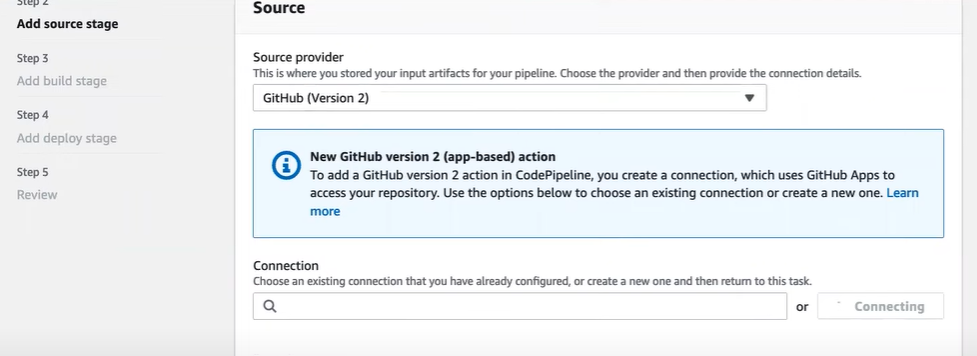


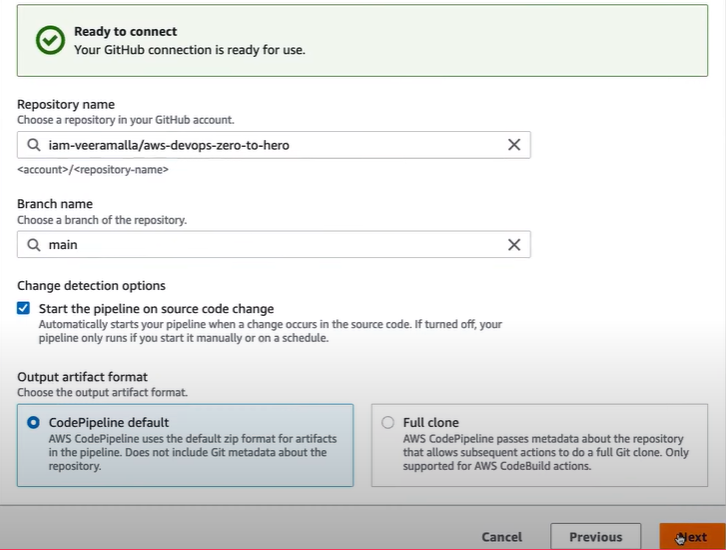
Integrating with AWS Code Pipeline :

1. Go to the AWS Code Pipeline

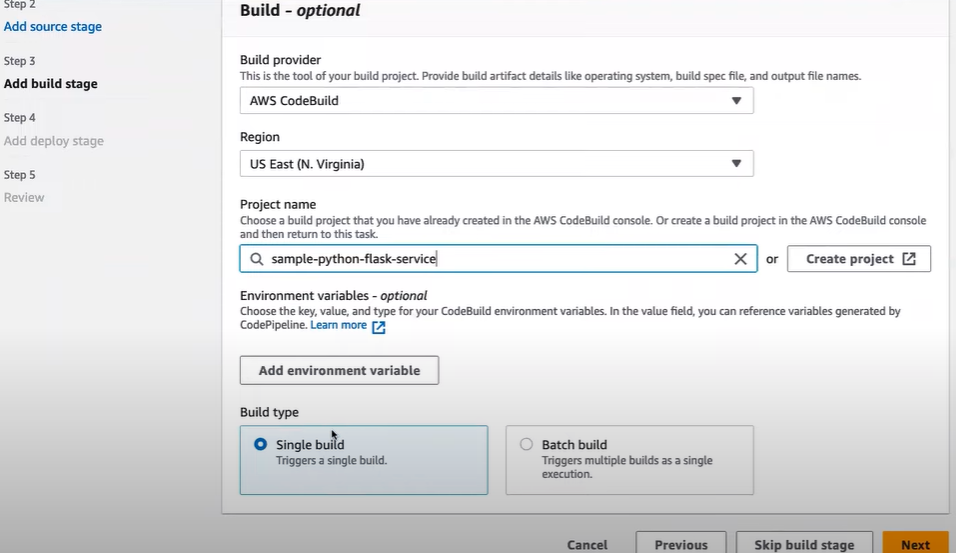


2. Connect to Github





3. Add Build Stage



4. Skip the deployment stage since we are working first on CI pipeline then create AWS code pipeline .

