Soliz, Joseph
Simplilearn Post-Graduate Cloud Computing
AWS DevOps
12/22/2024

AWS DevOps Course-end Project 1 Automating CI/CD Pipeline for Spring Boot Application Deployment on AWS

Objectives

To implement a CI/CD pipeline using AWS services for automating the deployment of a Spring Boot application on Amazon ECS with Docker, integrating CodePipeline, CodeBuild and ECR for seamless updates.

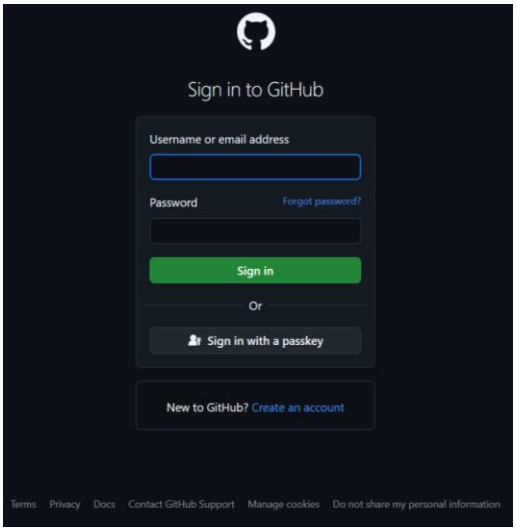
Tasks The following tasks outline the CI/CD pipeline creation process:

1. Fork the Spring Boot project GitHub repository and create an ECR to store the Docker images
2. Modify the build configuration file in the GitHub repository to include the ECR image URI
3. Create a CodeBuild project to pull code from the GitHub repository
4. Create an ECS cluster using the Fargate launch type and configure an ECS service to run tasks on the cluster
5. Deploy the application
6. Configure CodePipeline to trigger on GitHub changes, integrating CodeBuild and ECS for automated deployment
7. Test and monitor the pipeline execution in AWS CodePipeline

Step 1: GitHub Repository and ECR

1.1 Connect to GitHub Repository

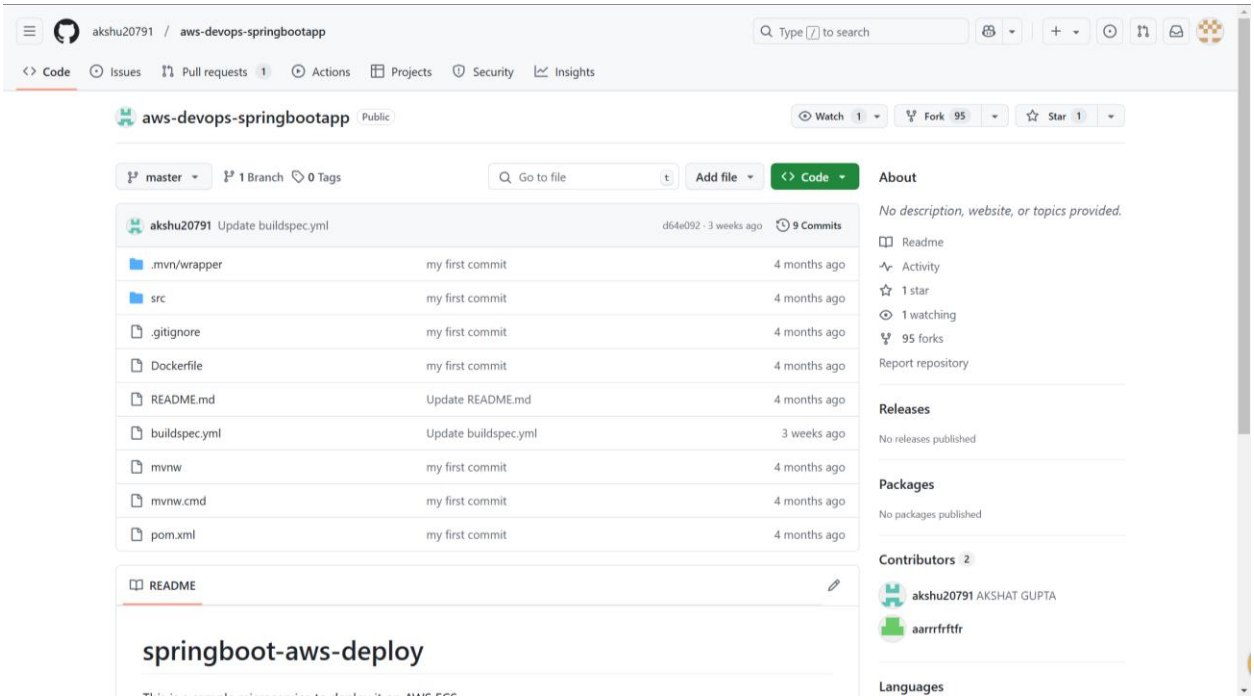
To start the project, log-in into your GitHub account.



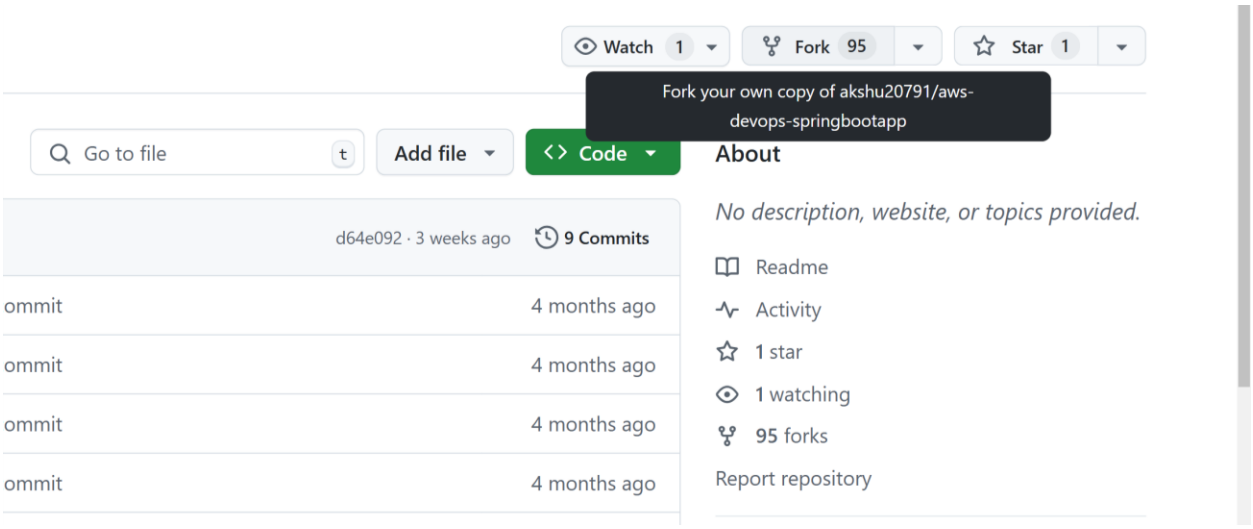
To host a Spring Boot application's source code in a GitHub repository, you will need to either create your own GitHub repository with a connection to Docker. Or use an established GitHub repository with Docker and Spring Boot Application.

The repository will be the source for the CI/CD pipeline. This means when a developer commits and pushes changes to the GitHub repository, it triggers the pipeline in AWS. We will also be using Docker through AWS.

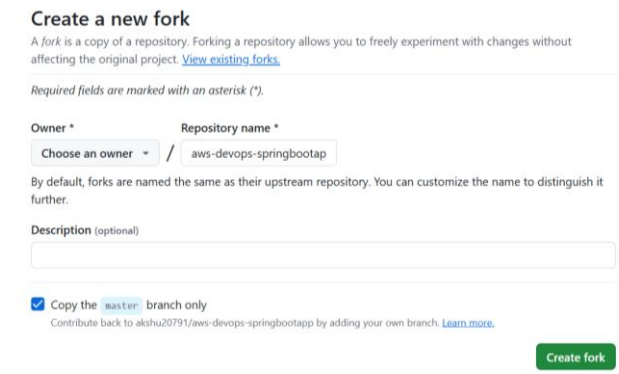
For this project, we will use an established GitHub repository: https://github.com/akshu20791/aws-devops-springbootapp



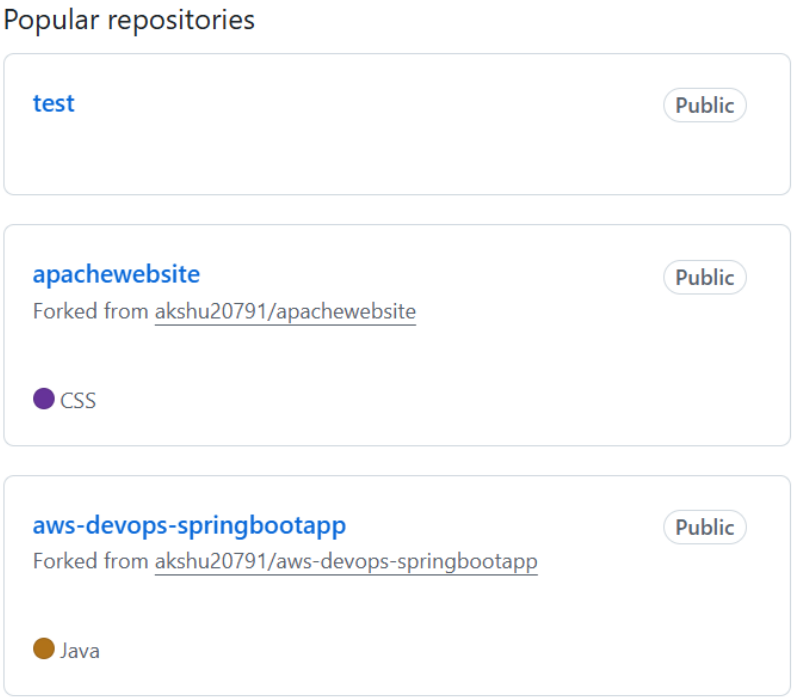
First establish a connection of the Spring Boot App by “forking” it to your account.



Click on “Fork”.



Select your GitHub account under Owner. Then select “Create fork.”

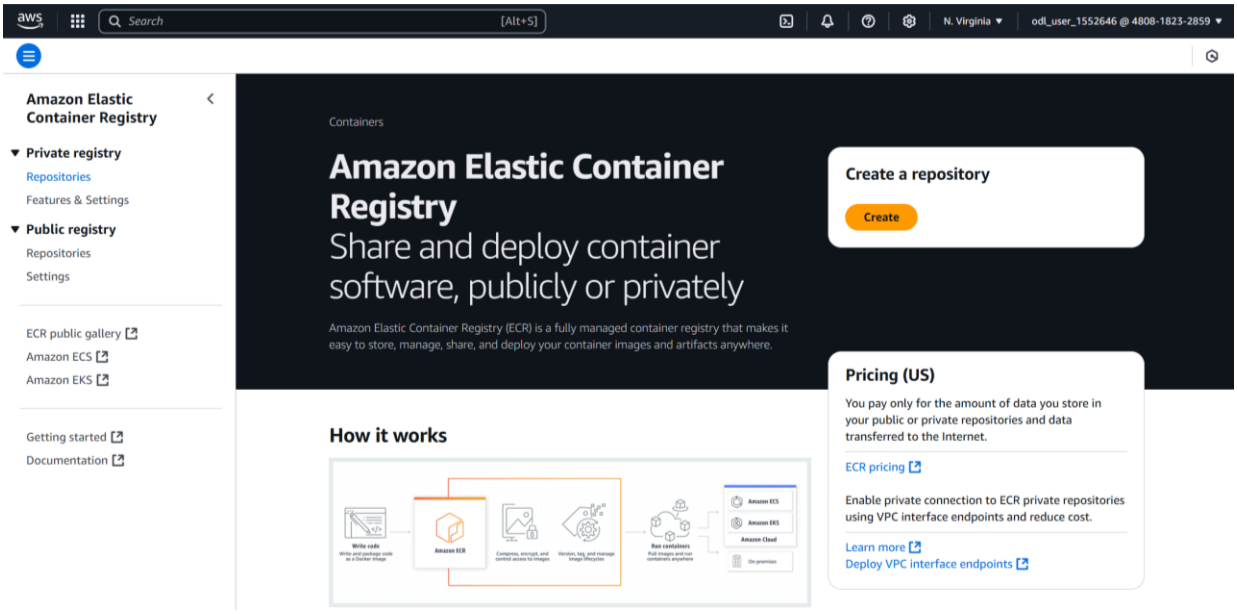


The GitHub repository “Spring Boot App” will appear as one of your repositories in your GitHub account. You can now begin the project in AWS.

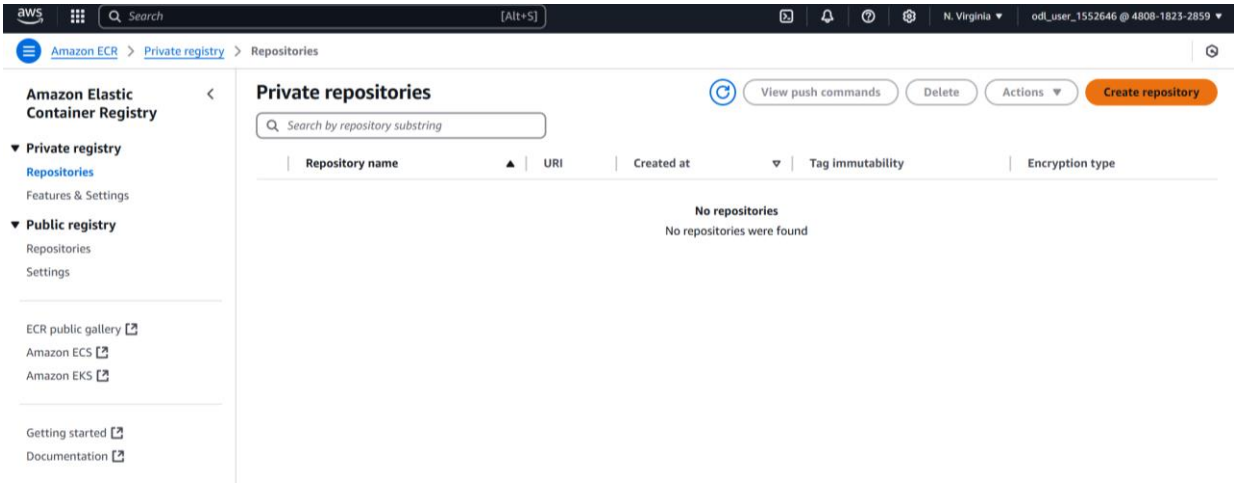
1.2 AWS ECR

Log-in to your AWS account. On the Homepage, search for Elastic Container Repository (ECR).

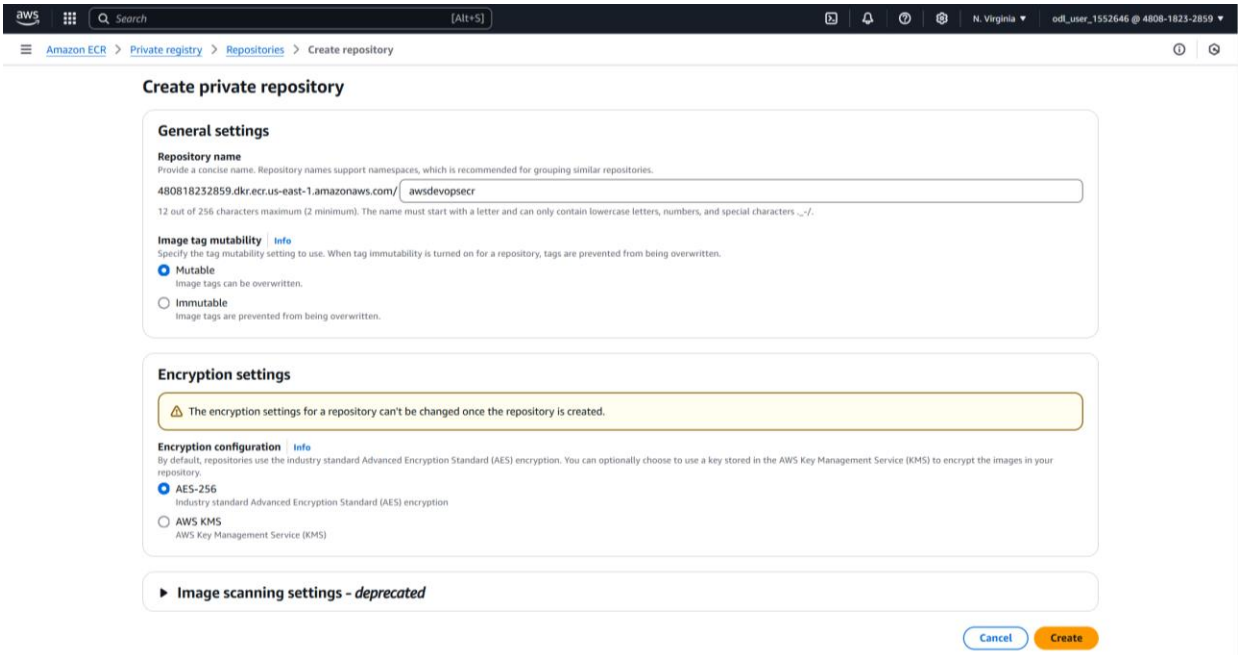
To create a private repository from Github, go to ECR. Click on Private Registry > Repository.



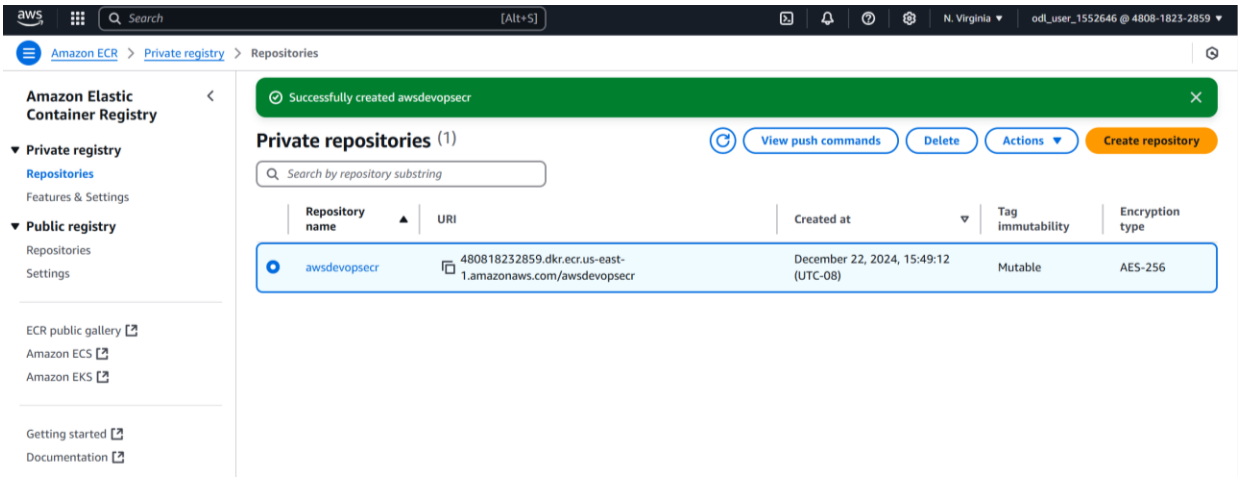
In private repository, click on Create Repository.



In Create Repository, provide a name for your ECR repository.

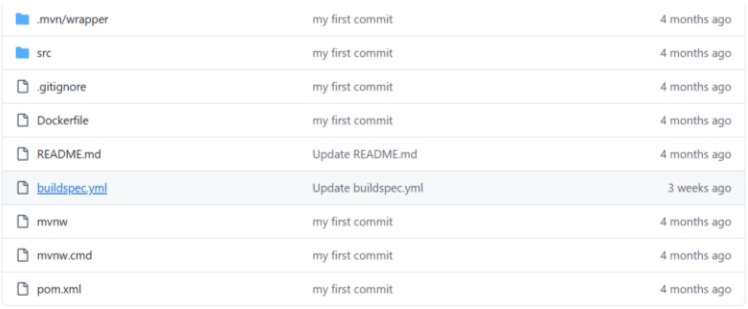


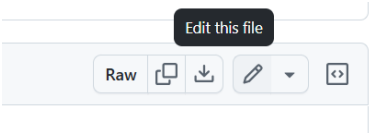
In this case, we will call our private repository “awsdevopsecr”. Leave all default settings and click on Create.



Now to establish first connection with GitHub to your ECR repository.

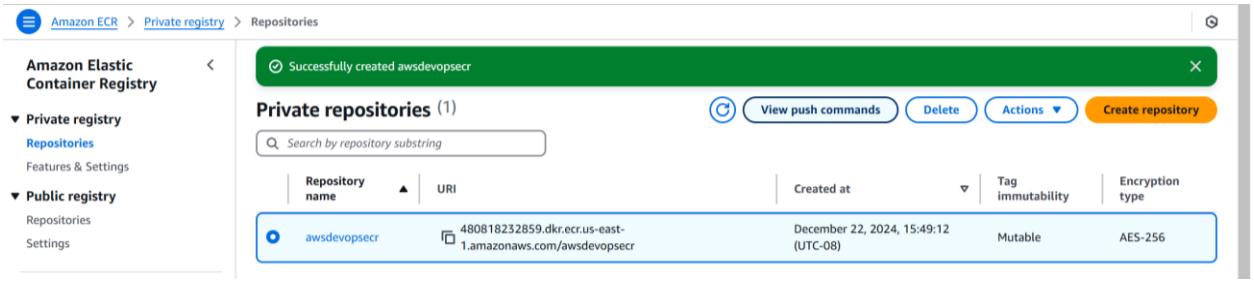
In the Spring Boot App repository you forked, click the file “buildspec.yml”.

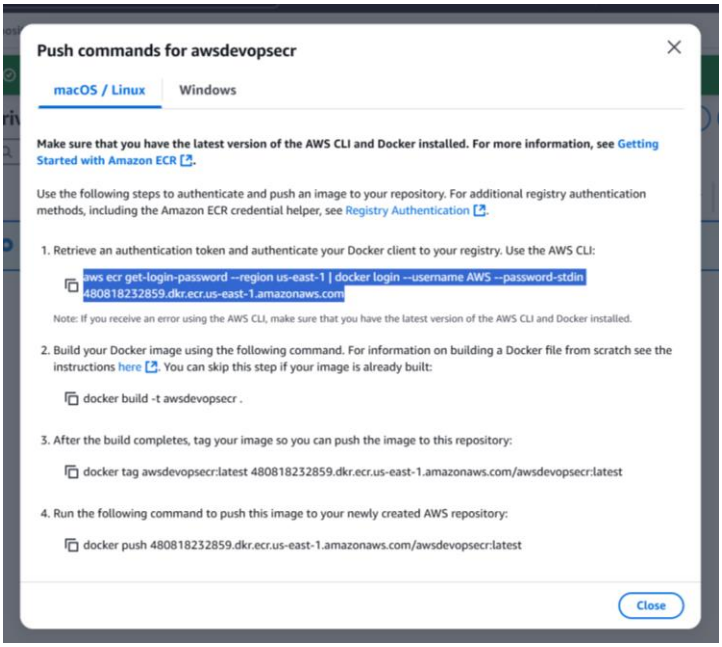




In the file, click on the pencil icon to edit file.

In ECR, select your repository and click on “View Push Commands.”

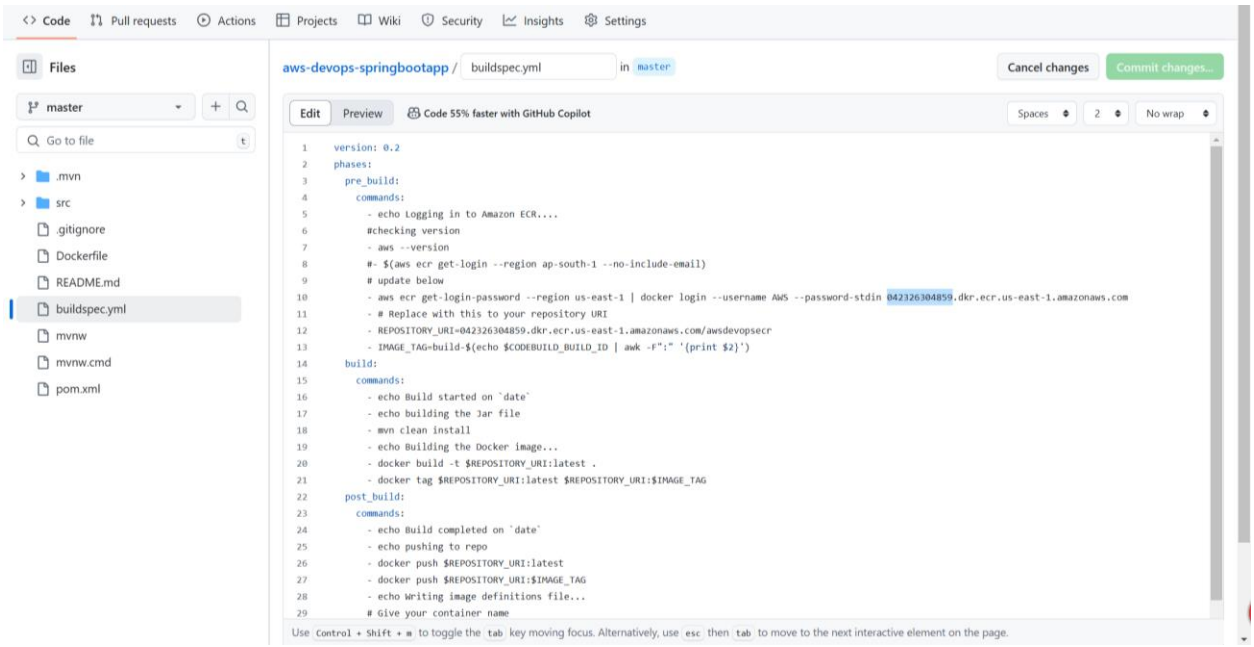




From the first push command. Copy the authentication token for macOS/Linux.

We will not be using Windows.

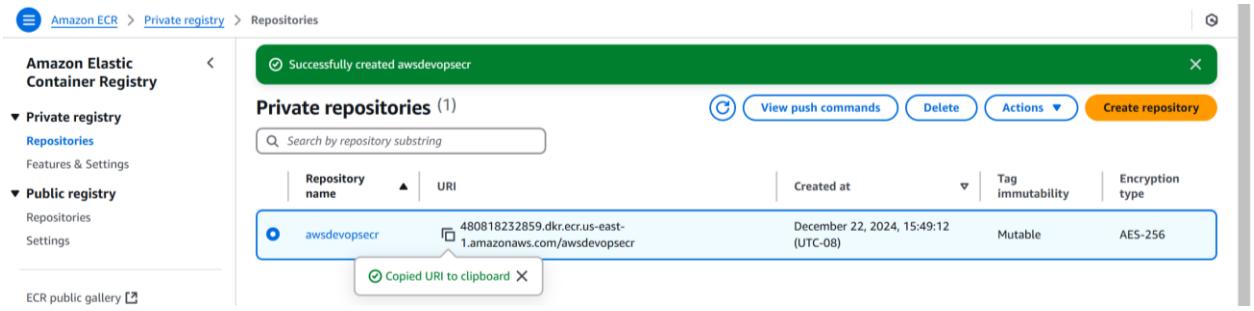
In editing the buildspec.yml file, go to line 10.



Edit it to match with the push command in ECR.

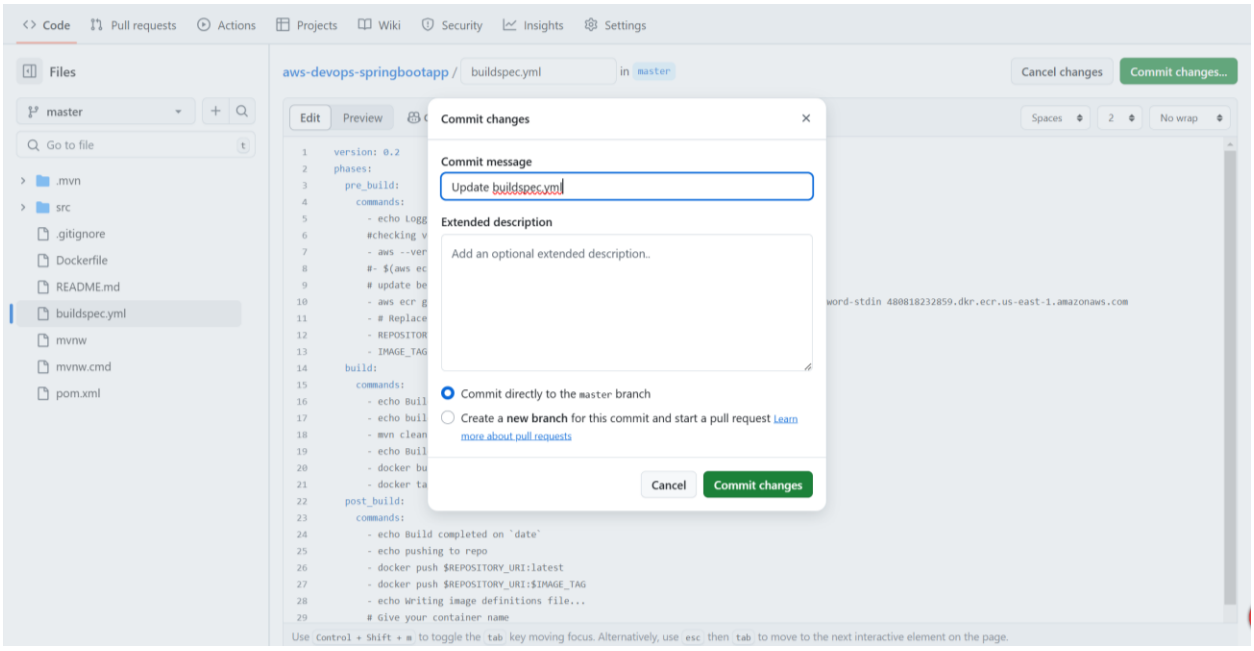
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 480818232859.dkr.ecr.us-east-1.amazonaws.com

Next copy the url of your ECR repository.

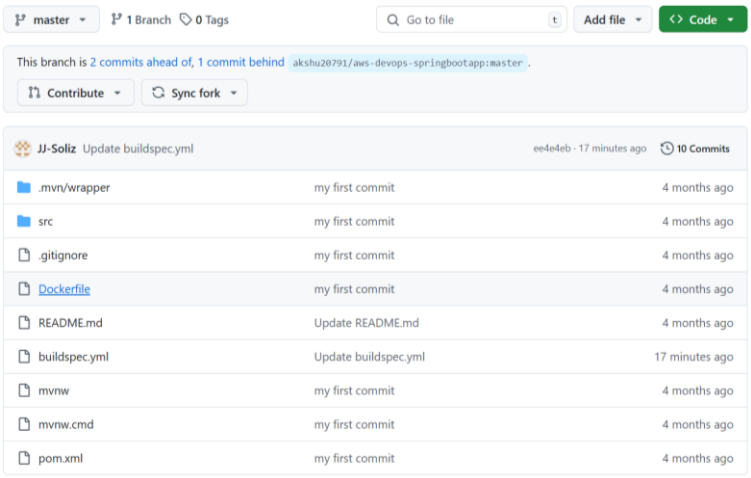


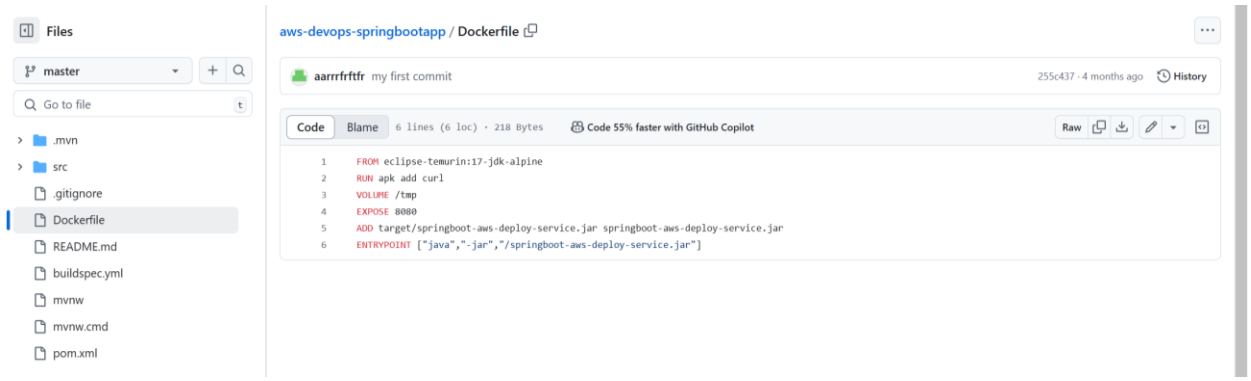
In repository URL in line 12, copy the url file and edit it in the yml file.

Now to save your edit, click on Commit Changes in GitHub.



As noted, the Spring Boot App comes with a Dockerfile to use.



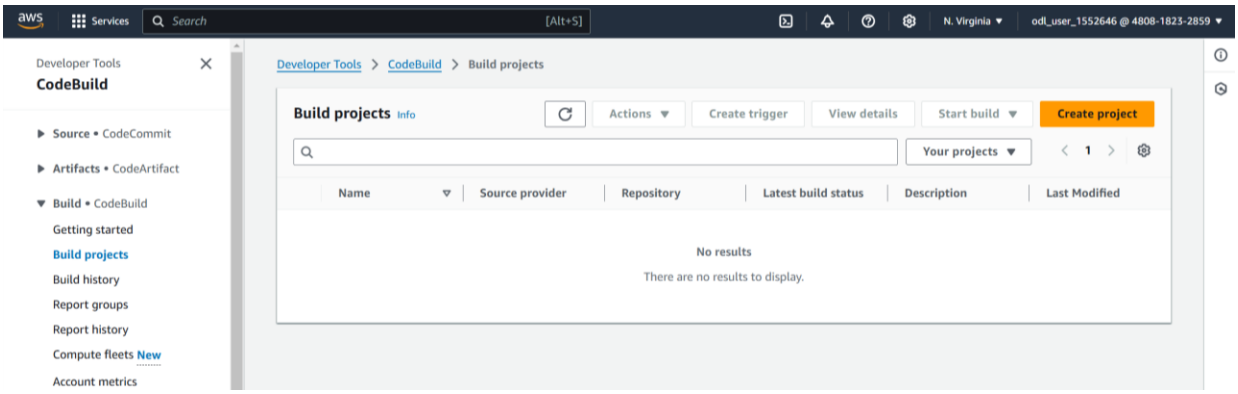


With our ECR repository connected to our GitHub repository, we can store the Docker images

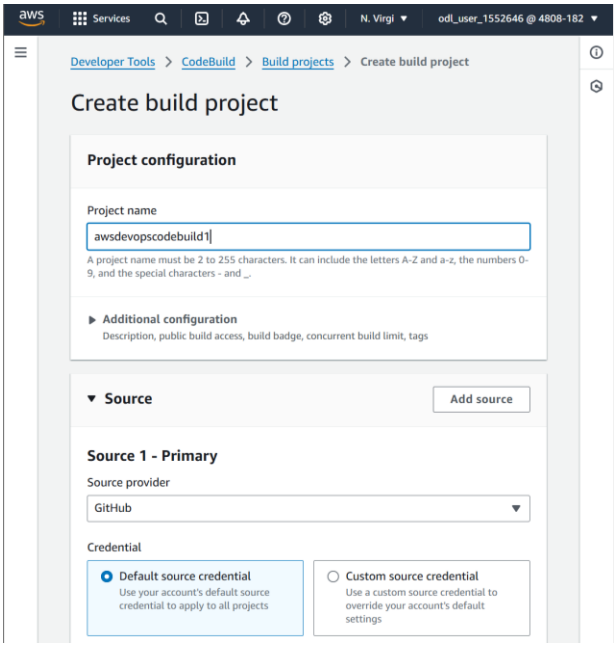
2. Create CodeBuild

2.1 AWS CodeBuild

In AWS, search for CodeBuild and click on it.

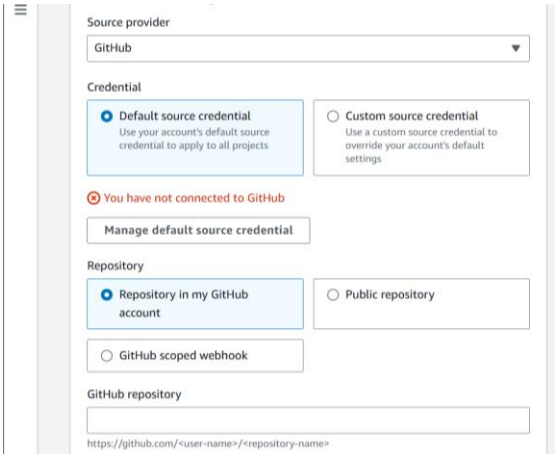


Click on Create project.

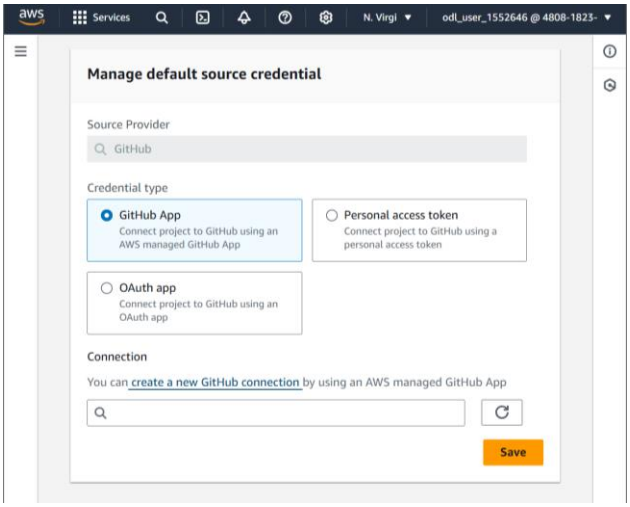


In Create build project, write in a project name for your build. In this case, we will call it “awsdevopscodebuild1”.

Under Source, select “GitHub” as your source provider.

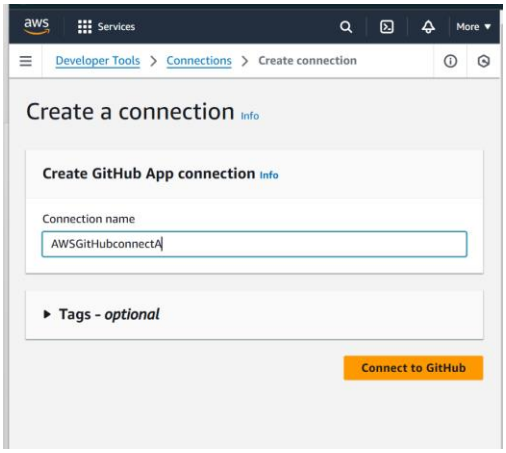


Note: your GitHub is not connected. Click on “Manage default source credential” and a new tab will appear.



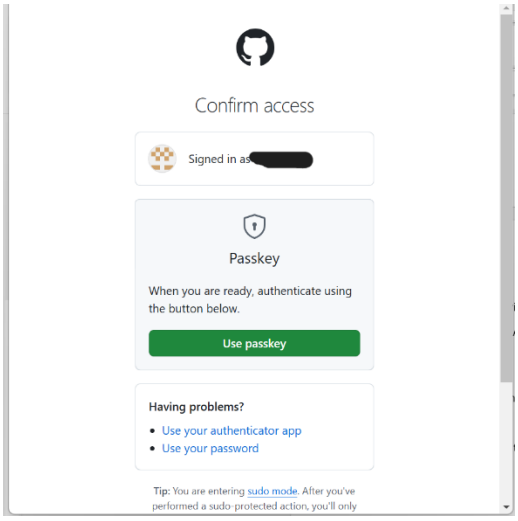
Select “GitHub App” and then click on “create a new GitHub connection”. Be sure you are logged into your GitHub account beforehand.

A pop up will appear.



For connection name, we will use “AWSGitHubconnectA”. Click on Connect to GitHub.

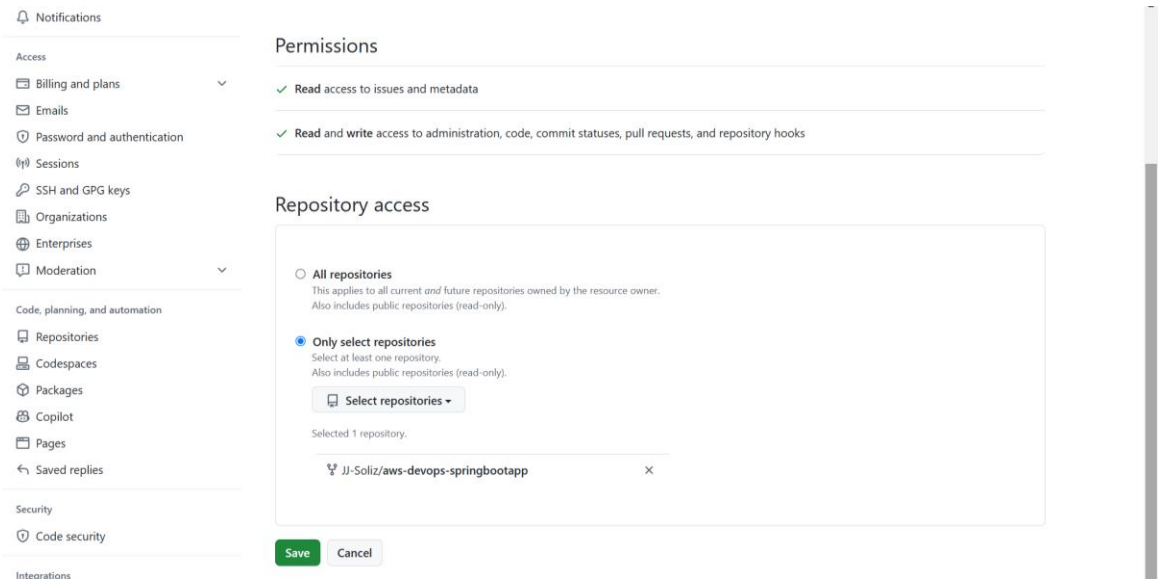
Next click on “Install a new App.” You will be brought to a page to select your GitHub account through AWS Connector.



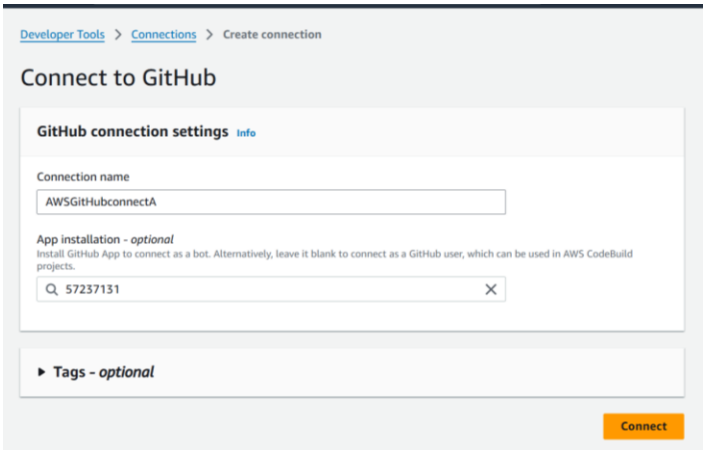
You have the option of Passkey, Authenticator app, and password. Select one and your Github account will be connected to AWS.

In the window, your Github Account settings will appear.

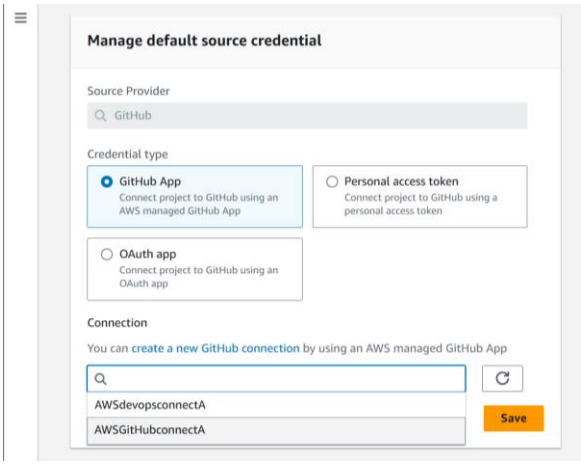
Under Permissions in Repository access, select repositories and select your Spring Boot App repository that you forked.



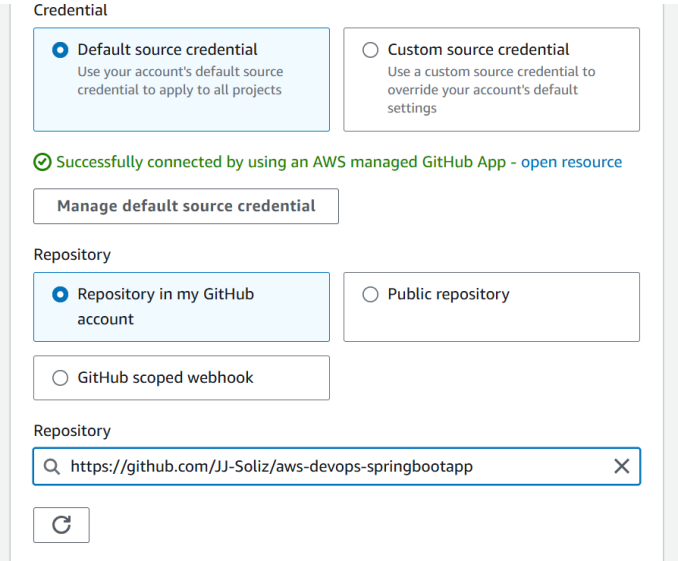
Click on Save.



Your connect is established, click on Connect again.



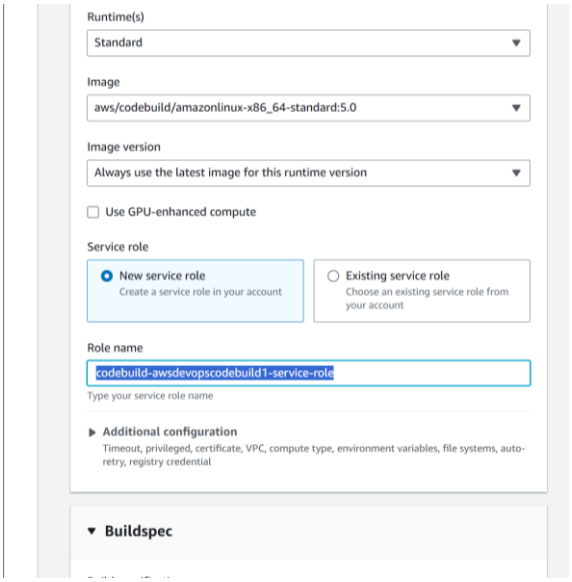
Click on your connection you created and click on Save.



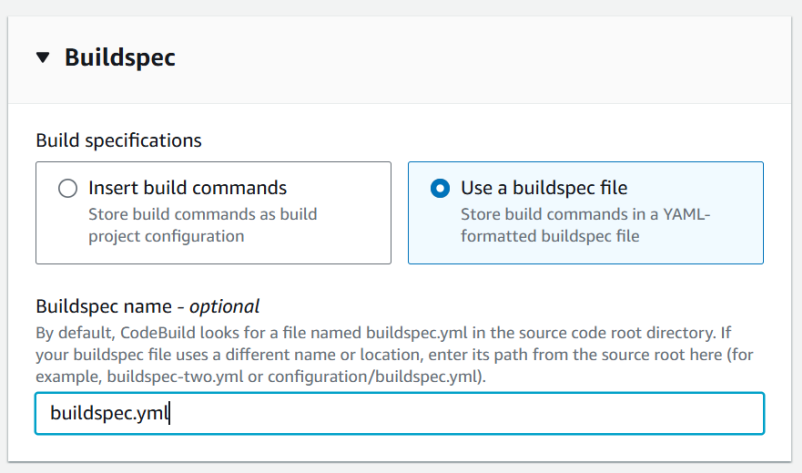
Your GitHub account is fully-connected. Select your GitHub repository and Scroll down.

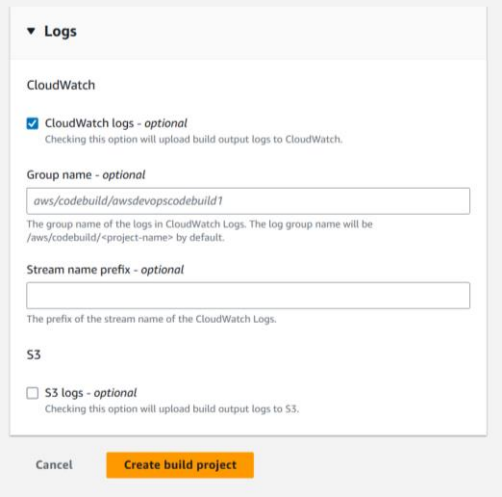
In Environment, be sure to select New Service Role and copy the Role name of the service role.

codebuild-awsdevopscodebuild1-service-role



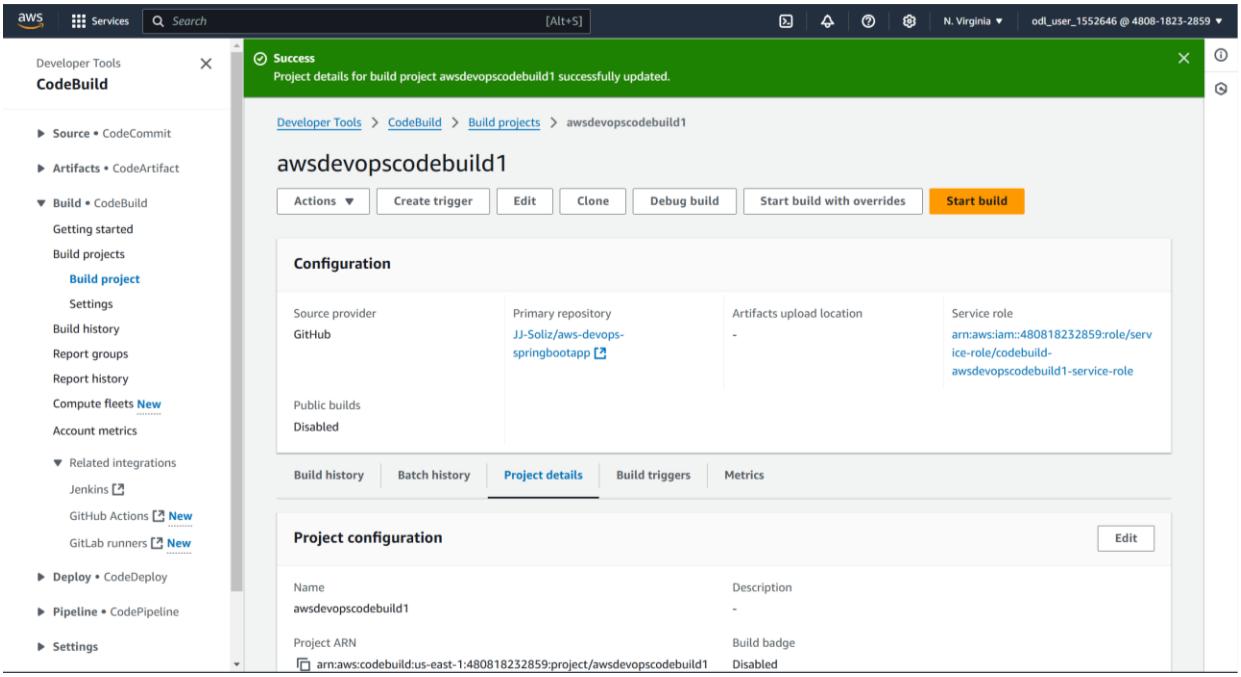
You will need the name when creating IAM roles.





Continue to Scroll Down until you reach the end and click on Create build project.

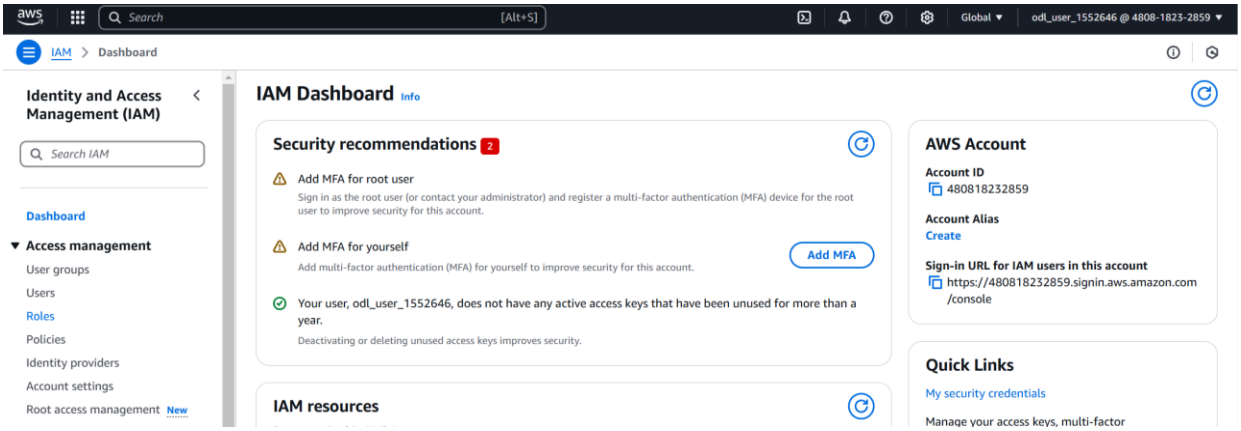
If you get an error, click on Edit and remove “Cloudwatch logs”. That should fix the issue.



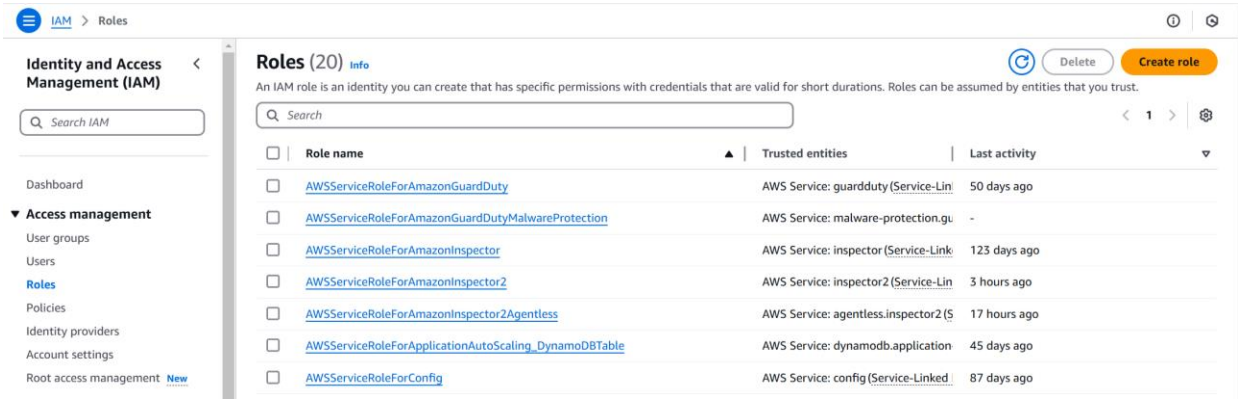
Your CodeBuild is created.

2.2 IAM Permission Roles

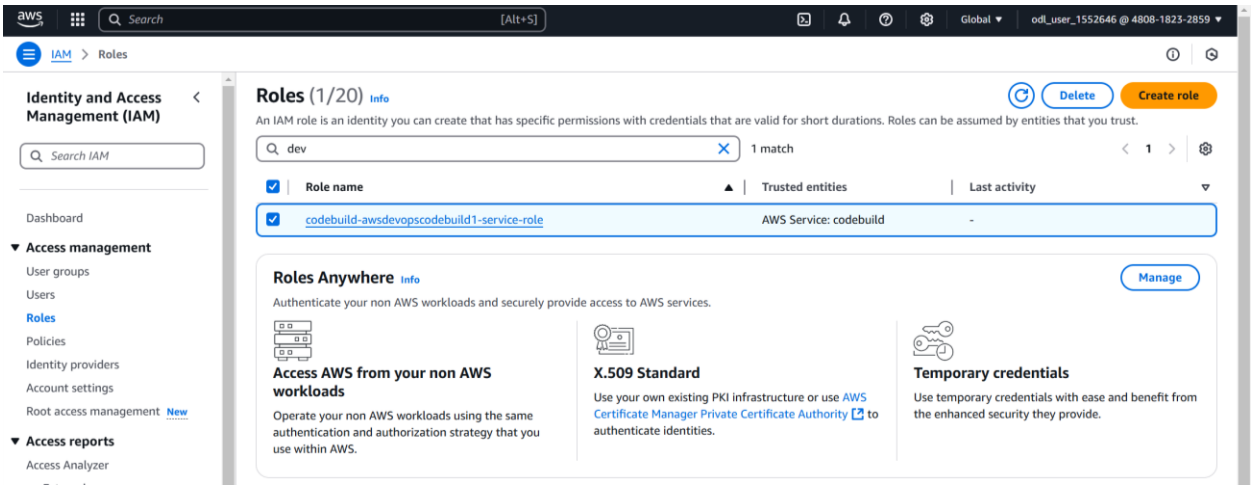
Go to IAM in AWS to establish permissions roles for your Code Build.



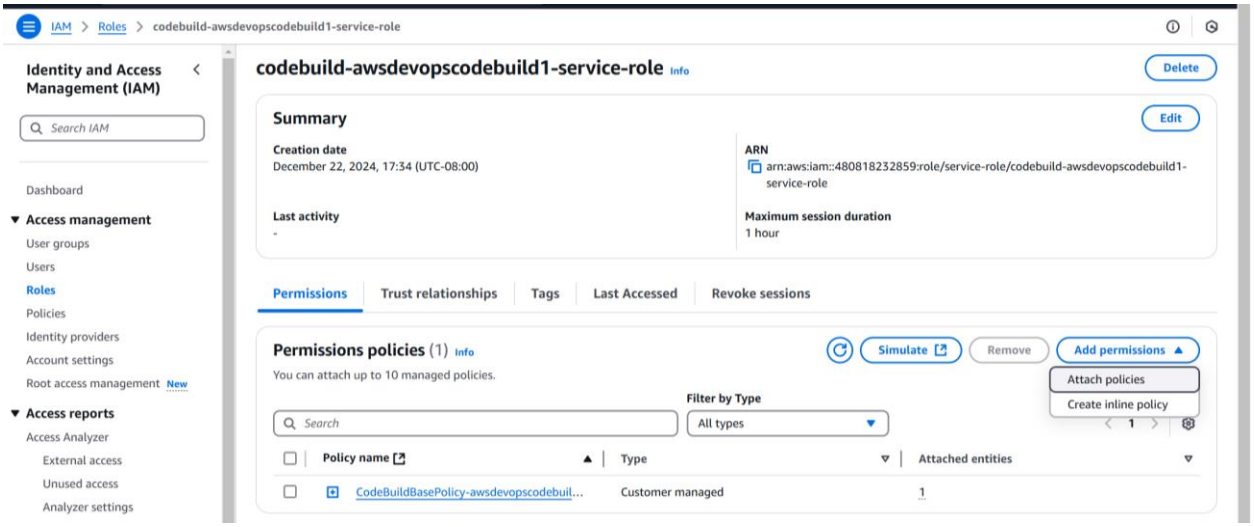
Select Roles in the left side of the menu.



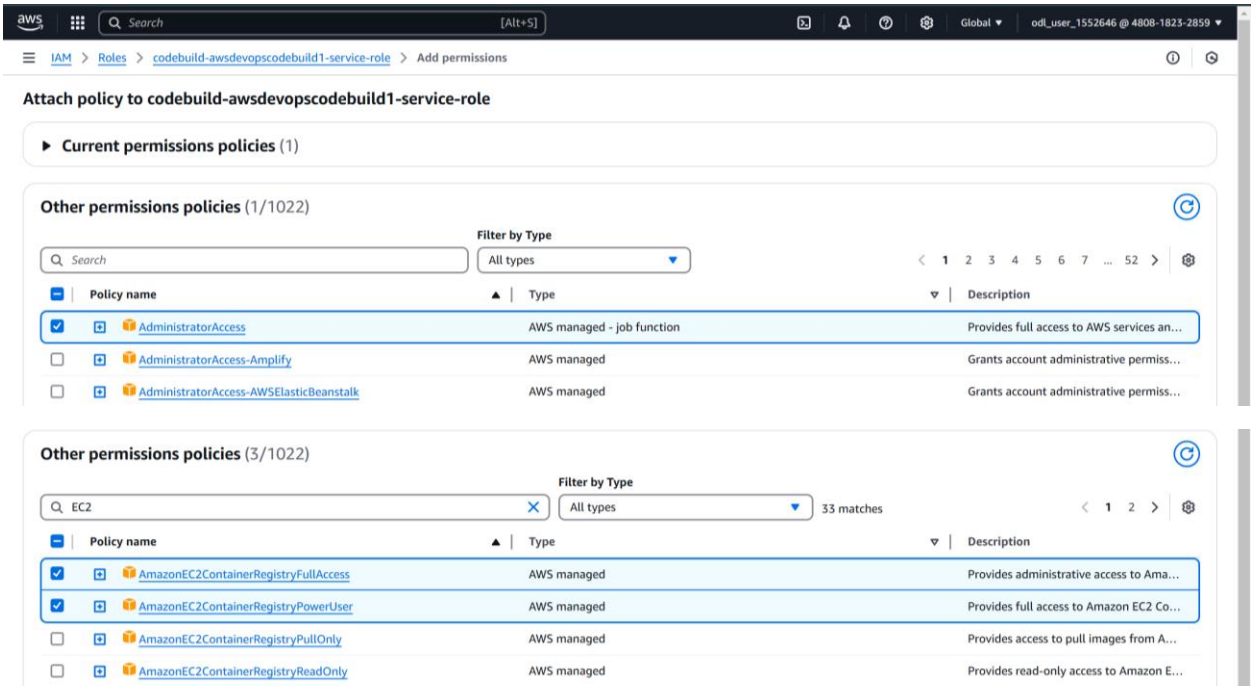
Select the role you created in the Code build.



Select and open.

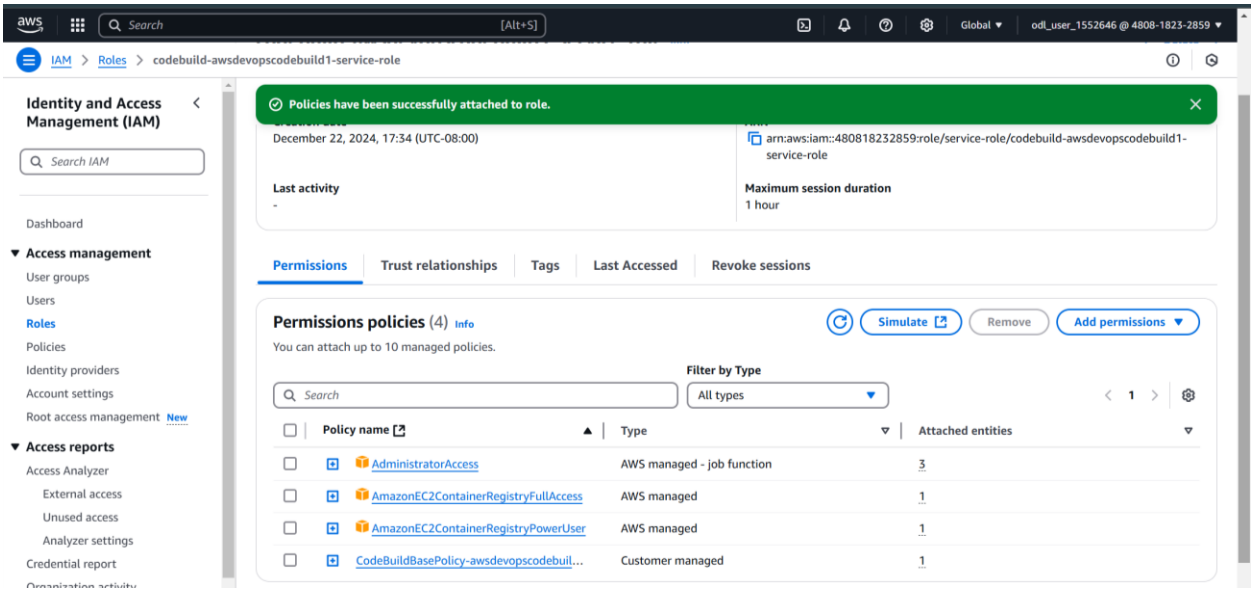


In Add permissions > Select Attach policies.



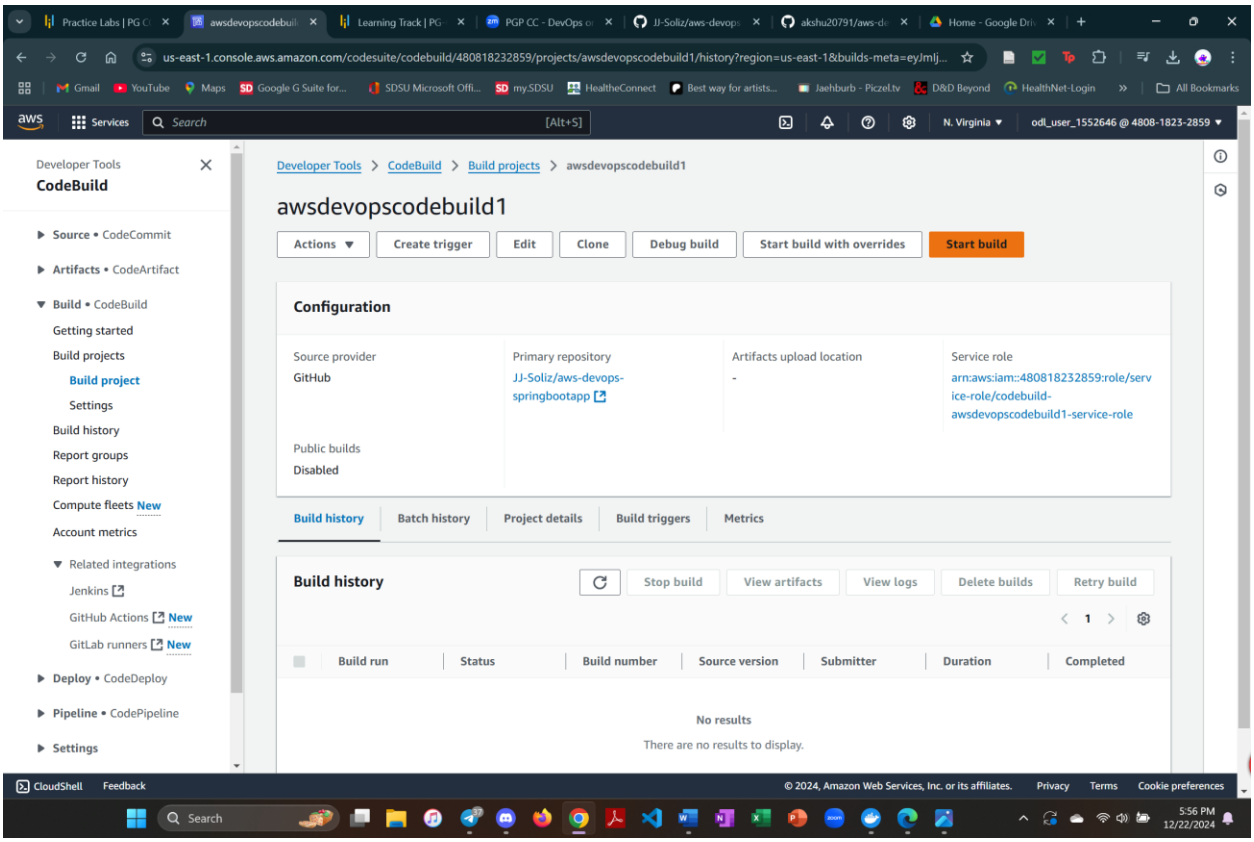
Select AdmistratorAccess, AmazonEC2ContainerRegistryFullAccess, and AmazonEC2ContainerRegistryPowerUser.

Select all three and then select Add Permissions.



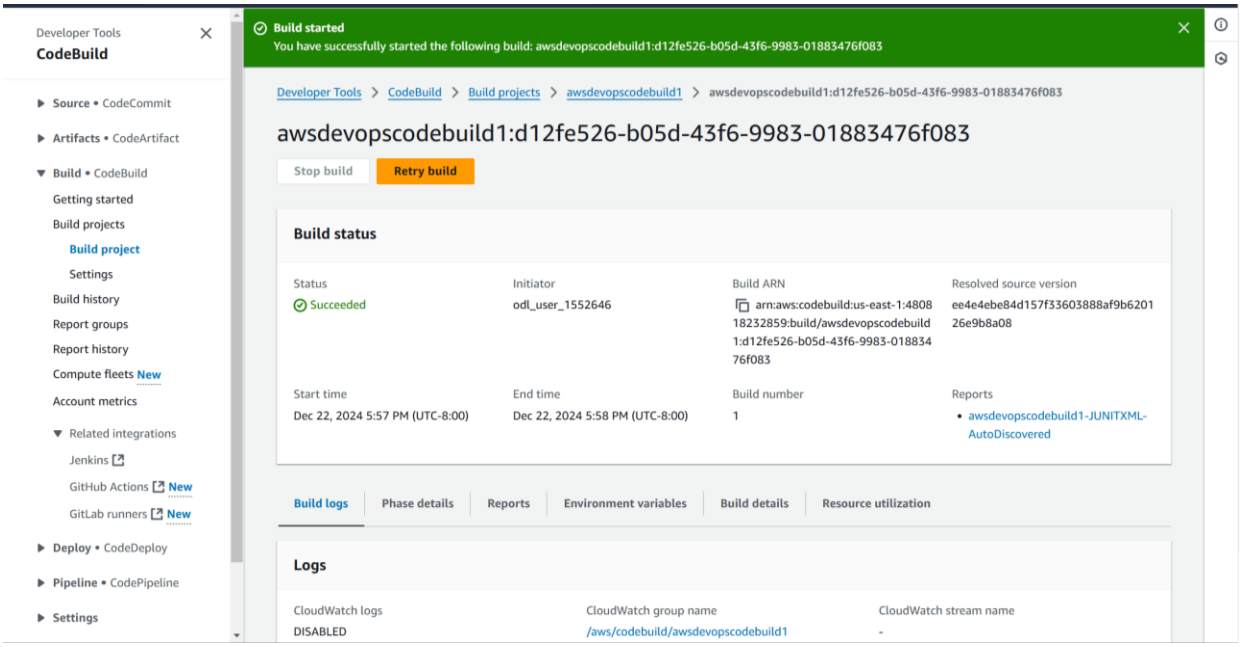
2.3 Create Build

Go back to your Code Build.



Select Start Build.

This will take a couple minutes.

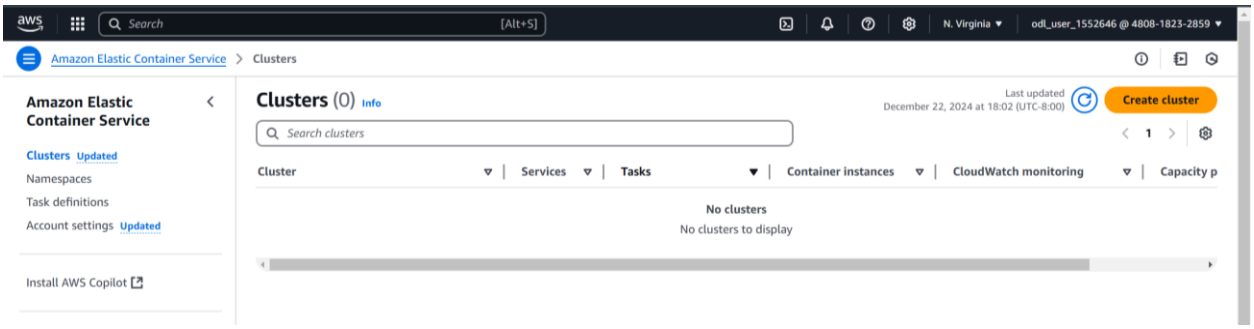


Build is successful.

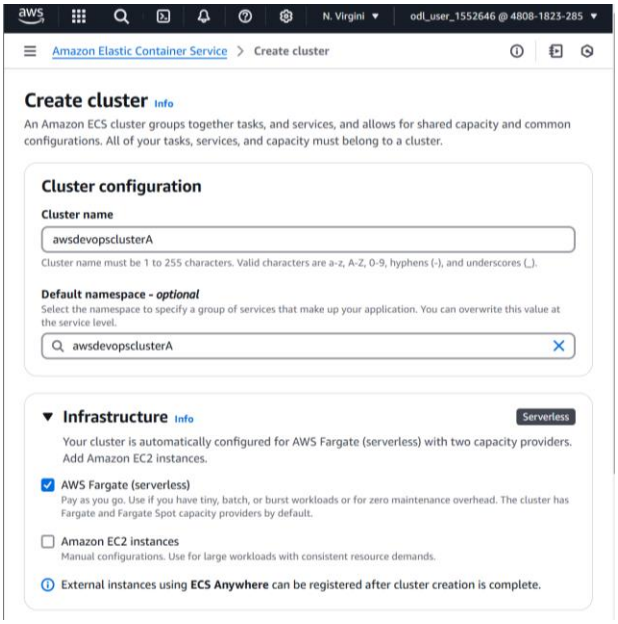
3. Elastic Container Service Cluster

3.1 Create ECS Cluster

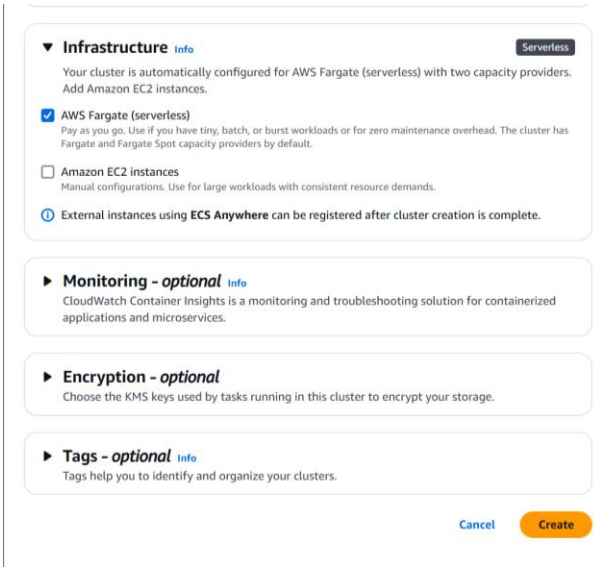
Go to Elastic Container Service (ECS). Go to Clusters.



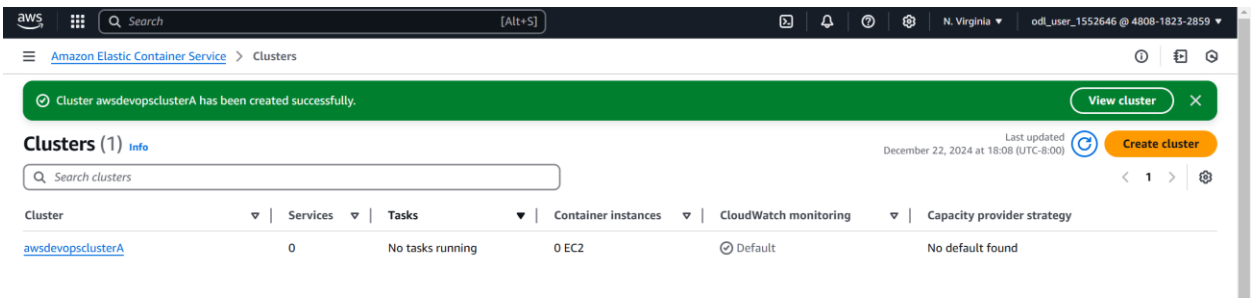
Select Create Cluster.



Create a name for your cluster “awsdevopsclusterA”.



Be sure to select AWS Fargate in Infrastructure. Then select Create.

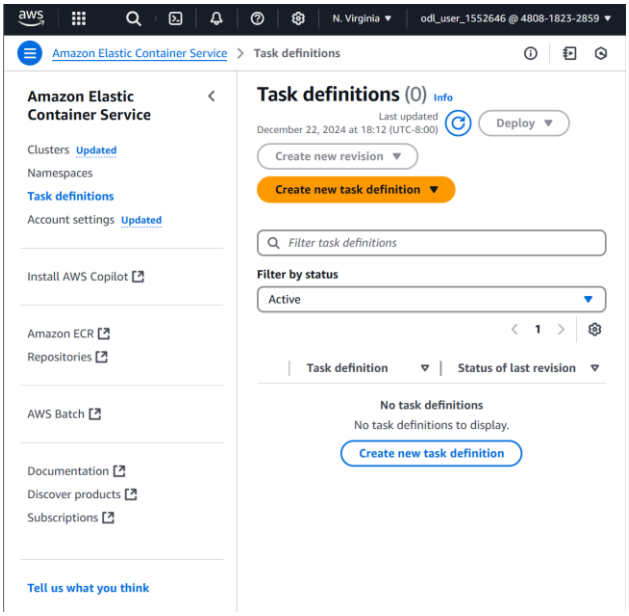


Cluster made.

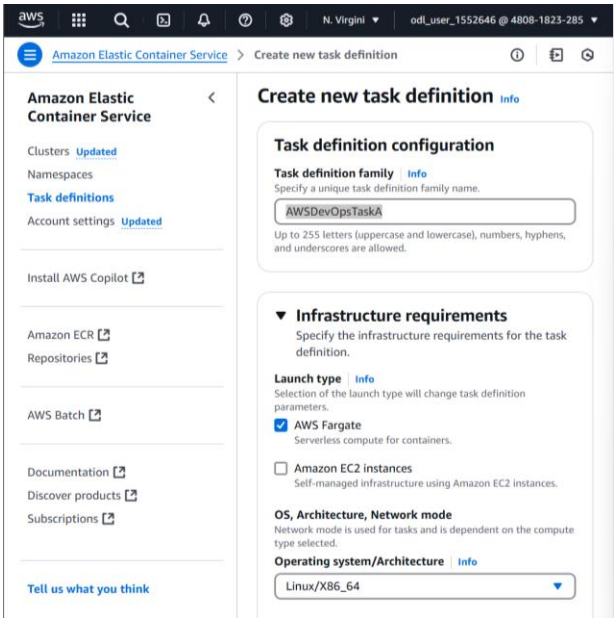
NOTE: You might get an error in creating your cluster. This could be due to the name or need to be redone again. Keep trying until it is successful.

3.2 Create Tasks

Go to Task definitions.



Click on Create new task defintion.



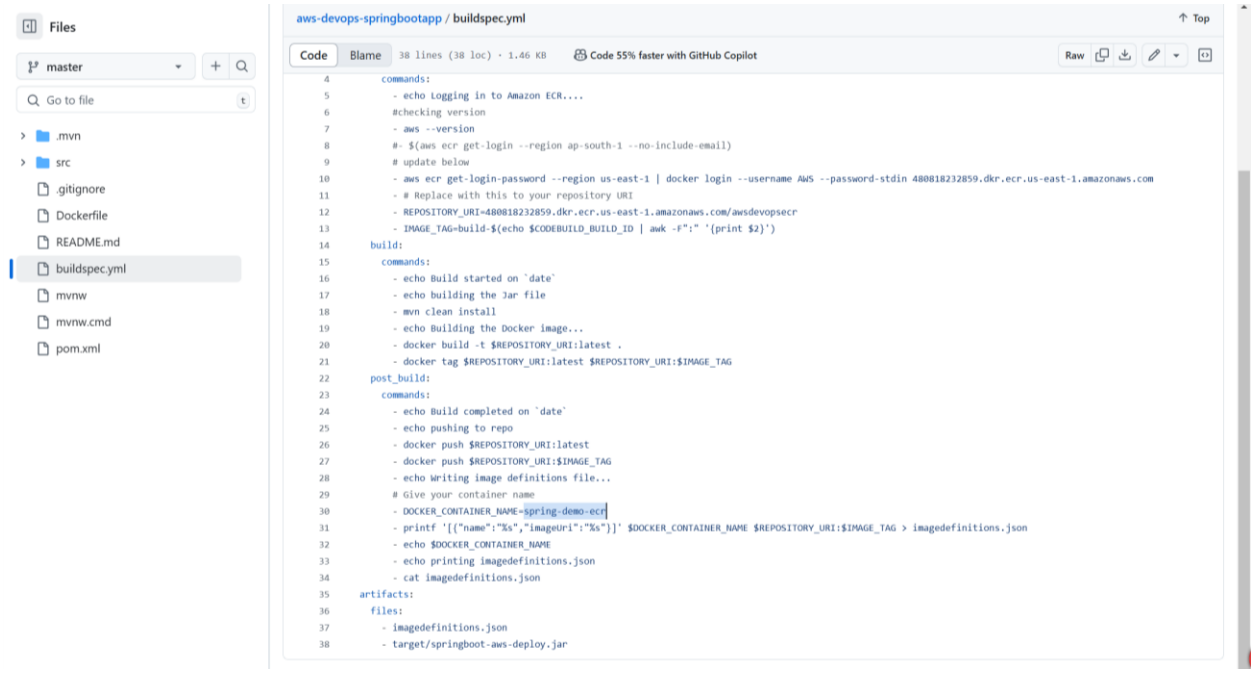
Give a name for the task defintion family. Be sure it is an unique name.

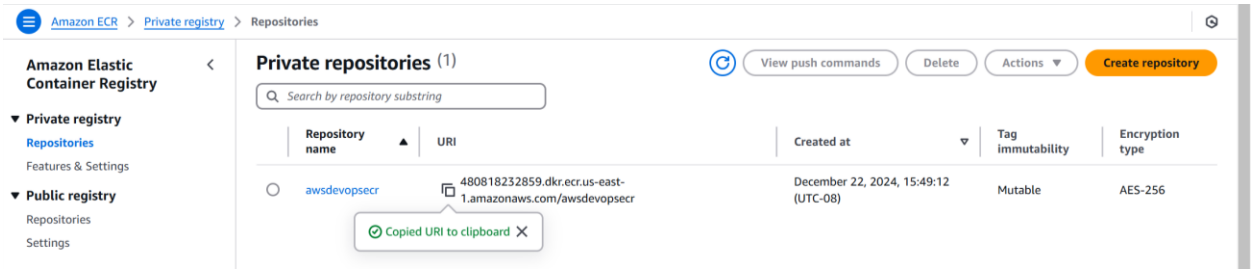
In Infrastructure requirements, select AWS Fargate.

Scroll down to Container -1.

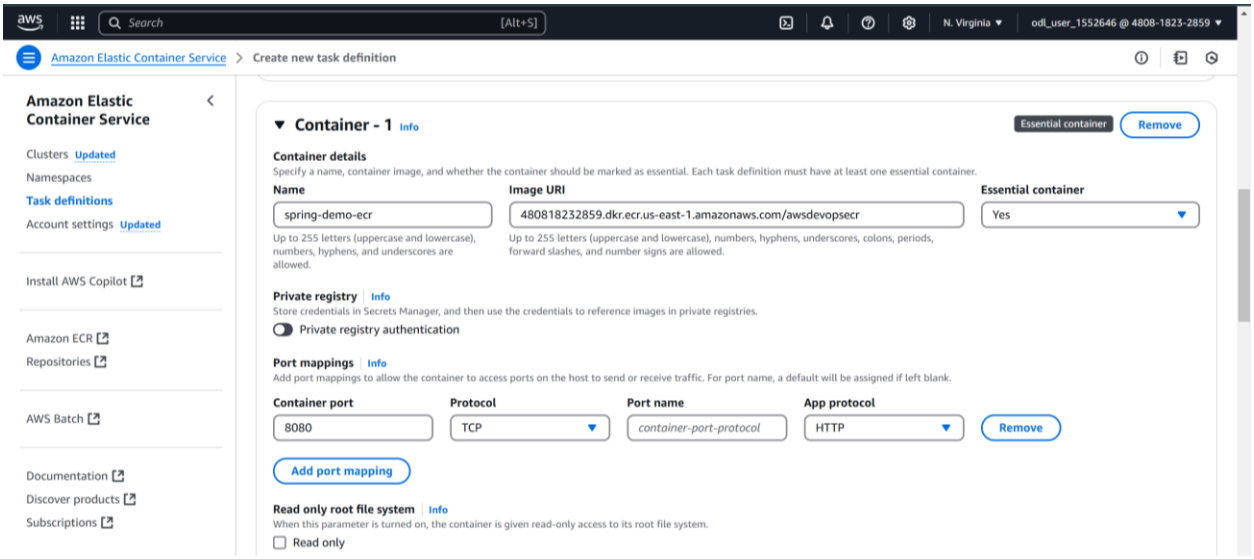
Provide the name of the Container: spring-demo-ecr

As shown in the buildspec.yml on GitHub.





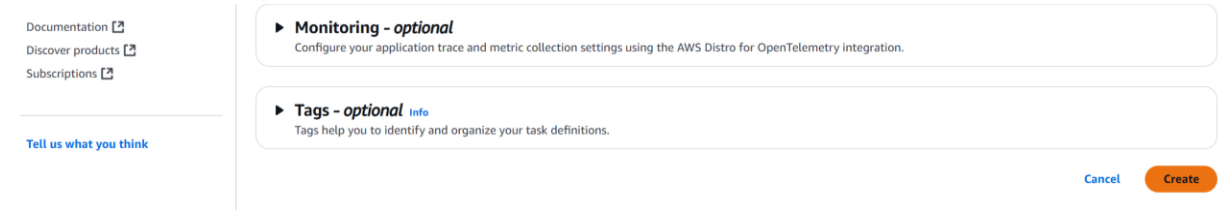
Copy the Image URI from the ECR repository you made.



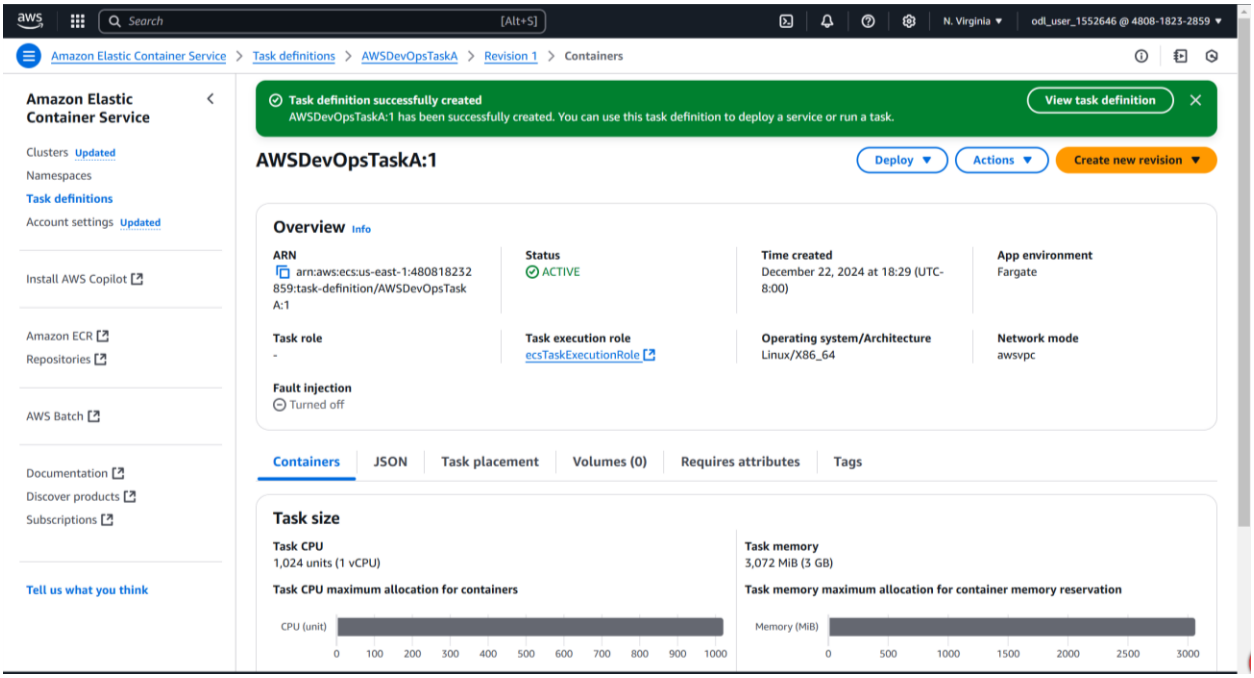
Copy ECR URI as your image URI for Task definition.

Be sure to change the container port to 8080.

This is how you will do Docker through AWS.

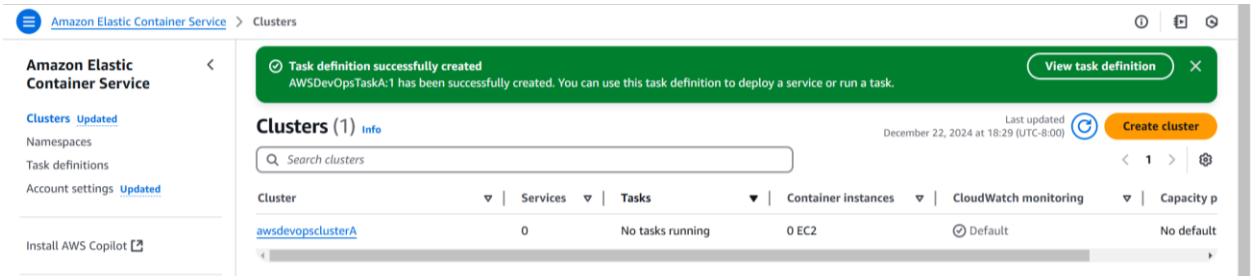


Click on Create.

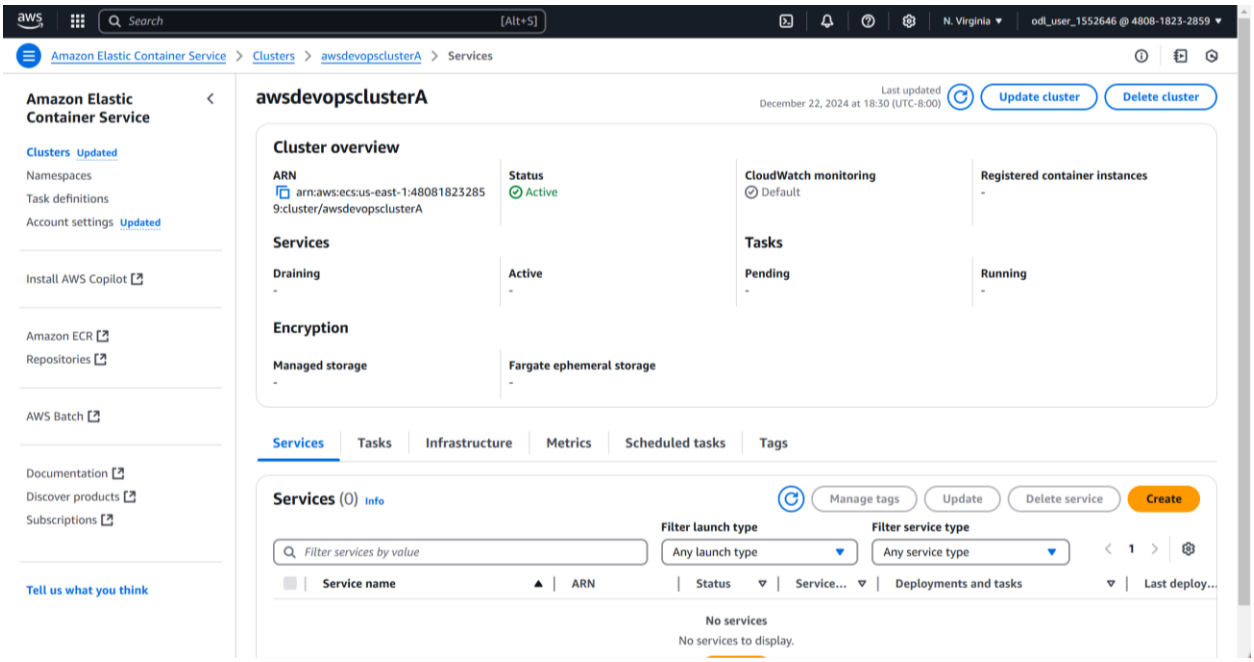


Task created.

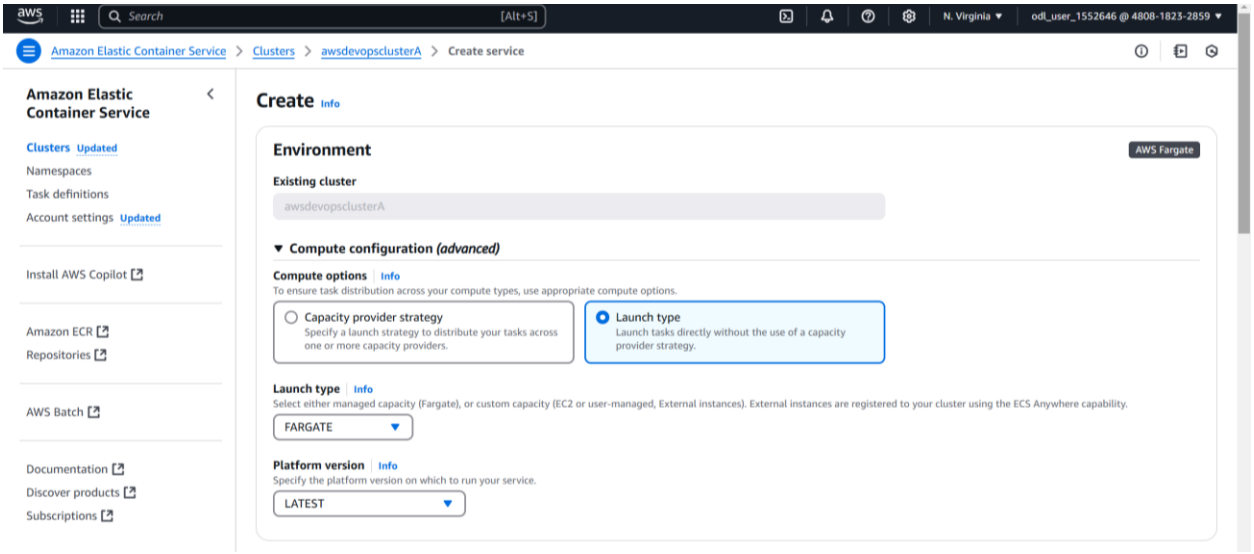
3.3 Services



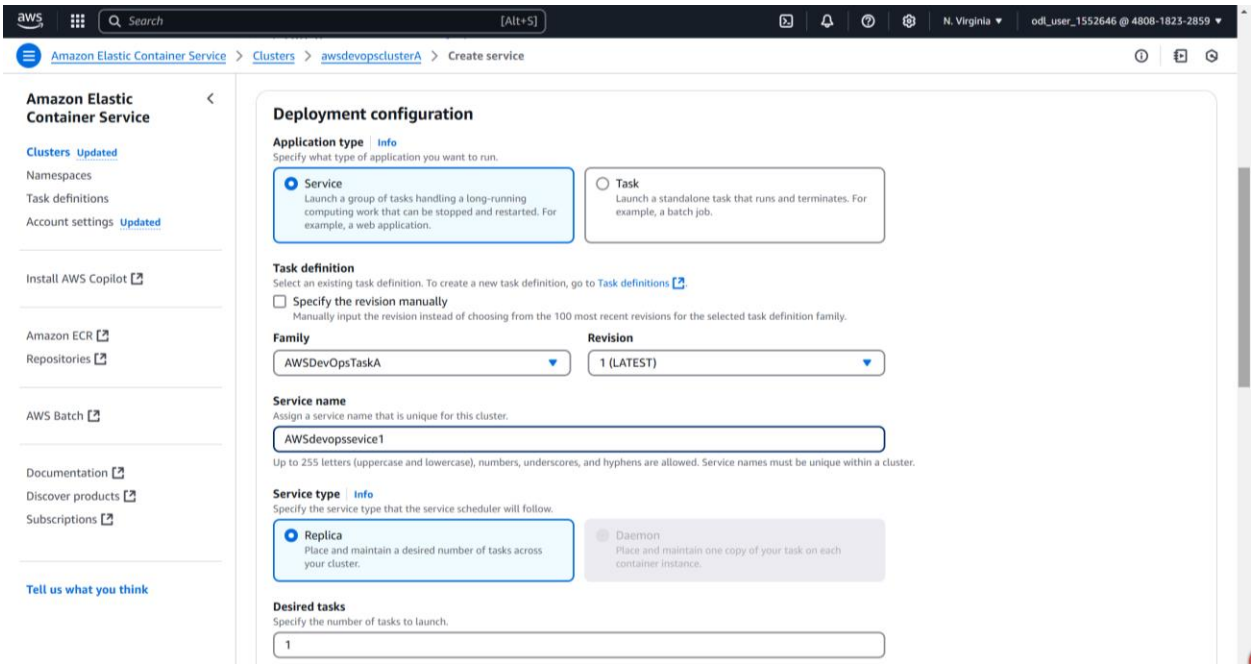
Go back to ECS and click your cluster.



In Services, click on Create.

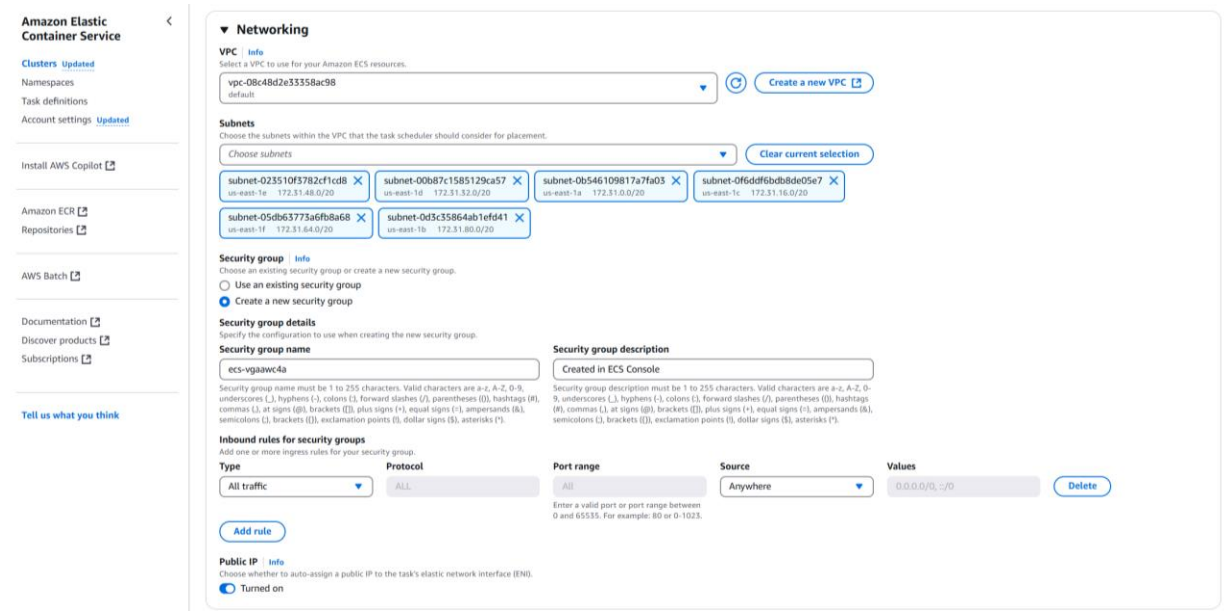


Select Launch Type. Be sure to have Launch type set to Fargate and Latest platform version.



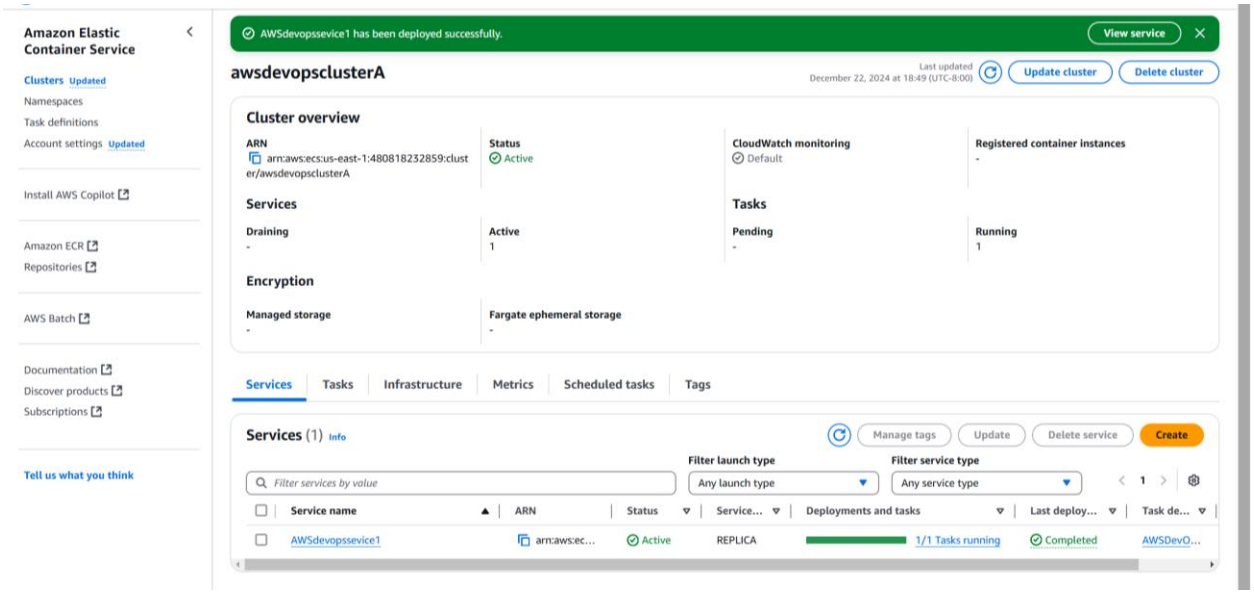
Select your task defintion you created and create a service name for it. AWSdevopssevice1

Under Network.

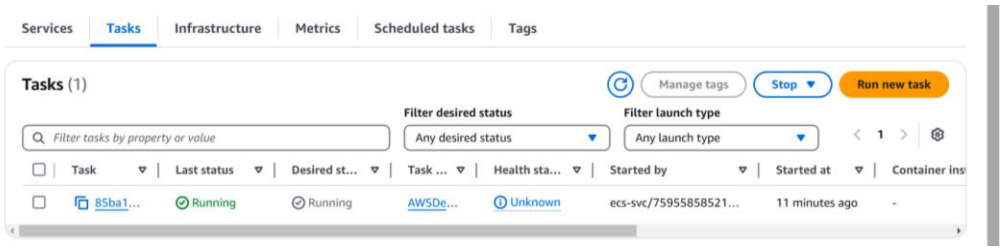


Create a new security group. Set inbound rules to Type: All traffic and Source: Anywhere.

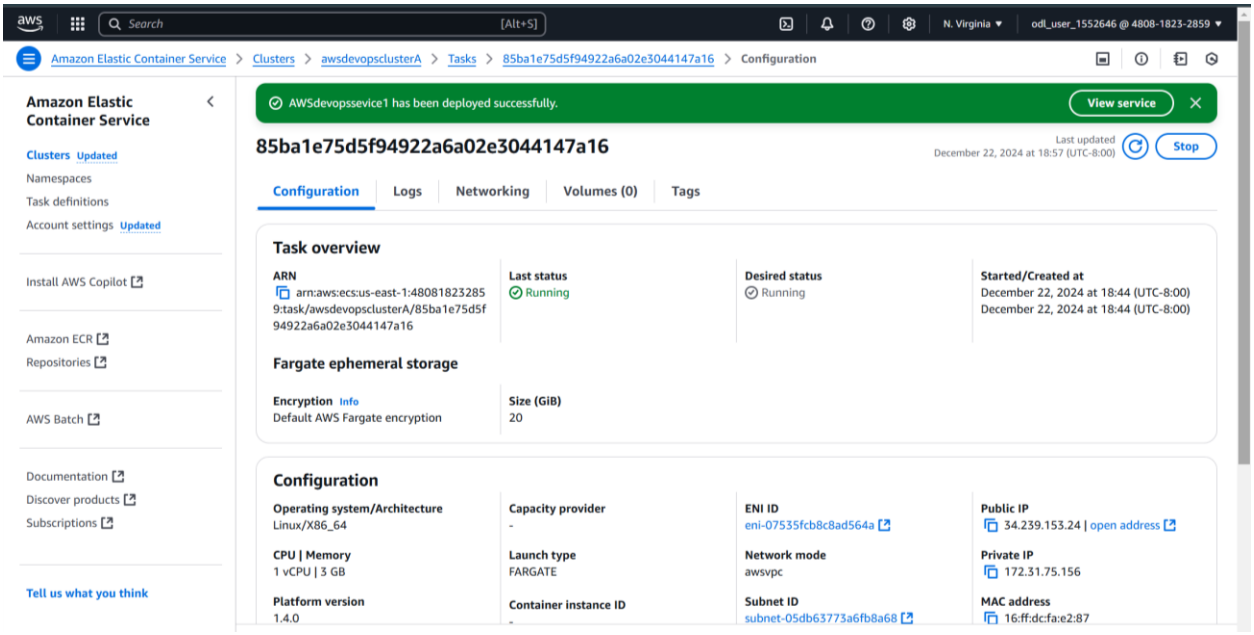
Now select Create. Give it a few minutes to develop.



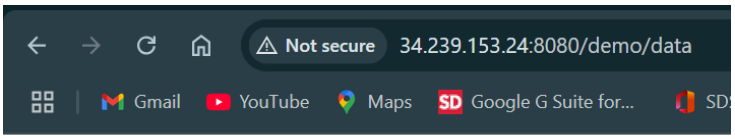
Service has been created.



Check tasks to see if it's running. As it is running select your task.



Copy Public IP to check if it is running. Be sure to add PublicIP:8080/demo/data

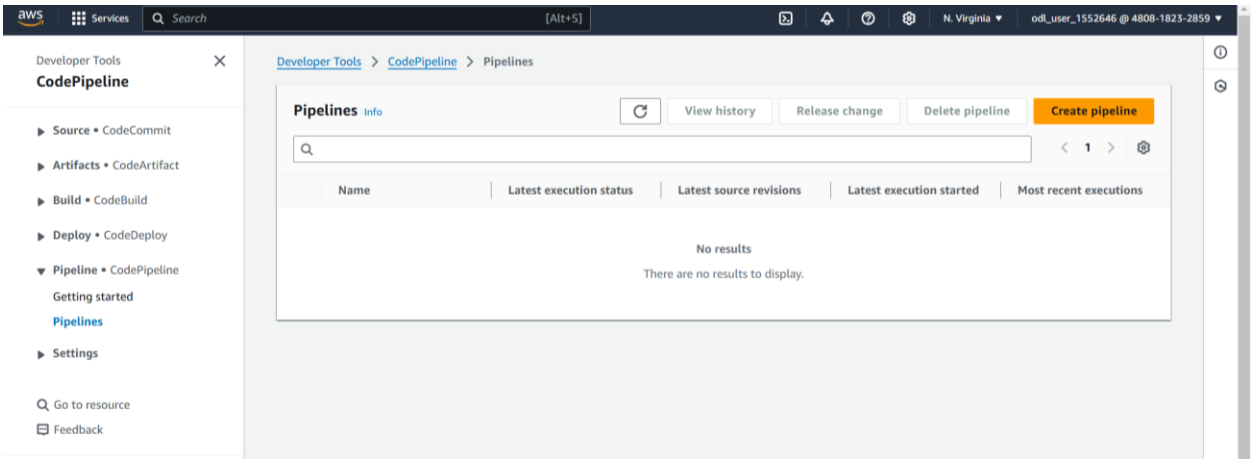


First message from AWS Ecs

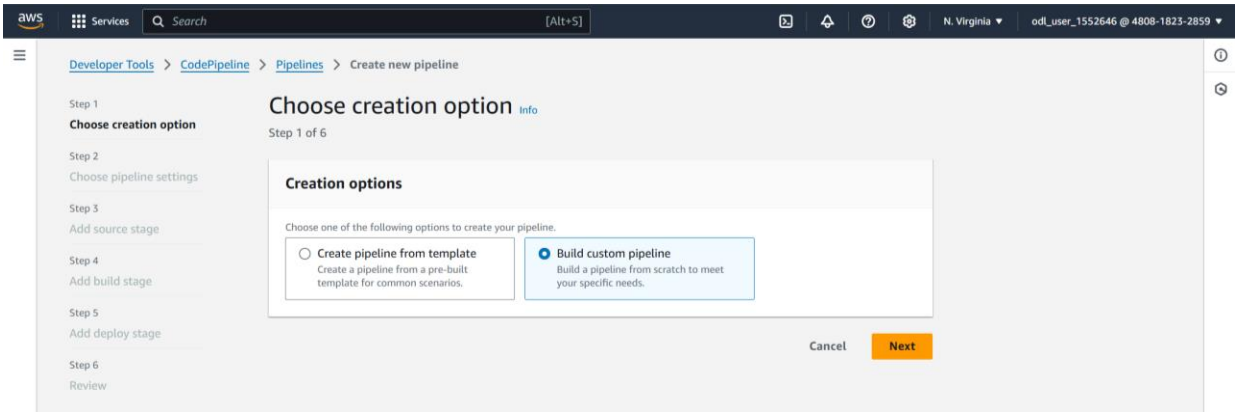
4. Code Pipeline

4.1 Create Pipeline

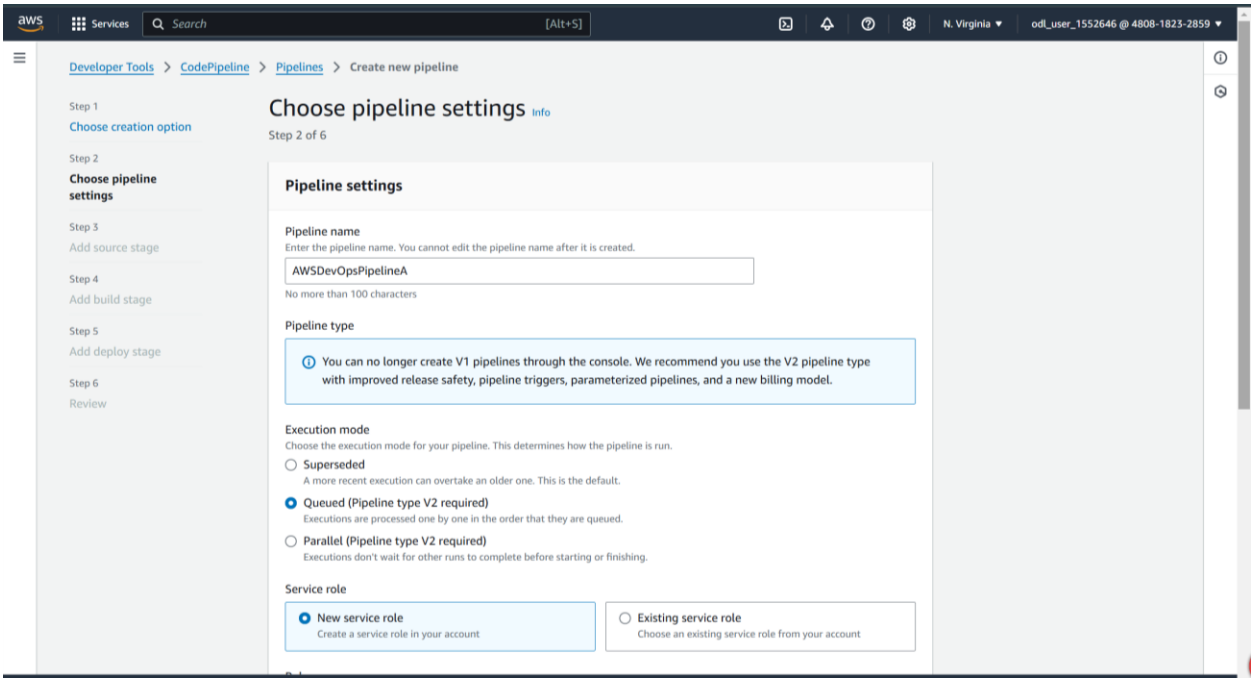
Go to AWS CodePipeline. Select Pipelines.



Select Create pipeline.



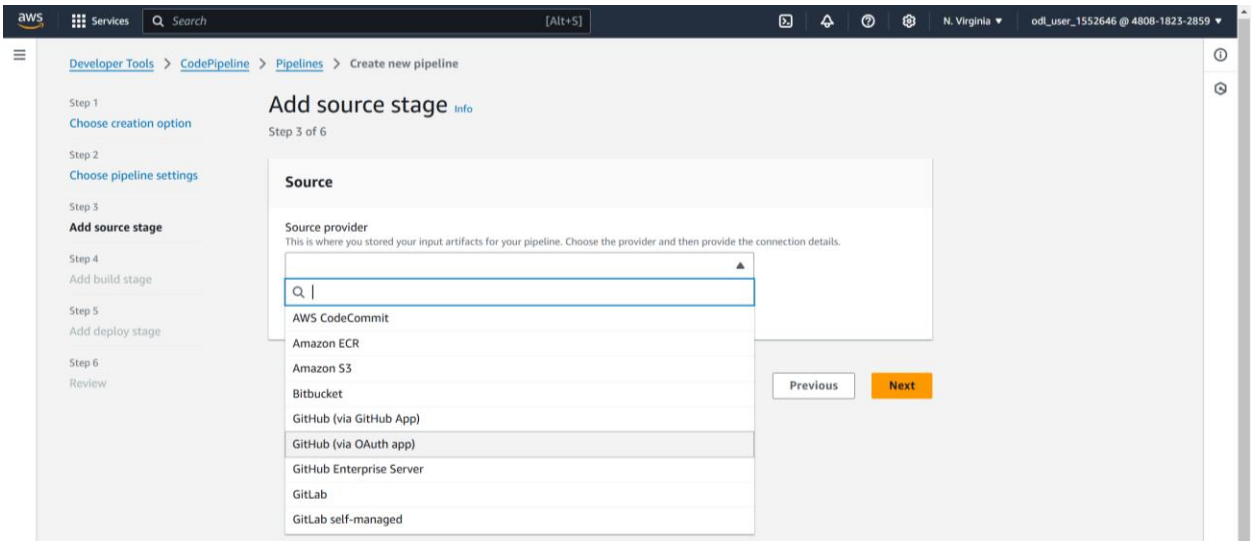
In the first step, select “Build custom pipeline”. Click Next.



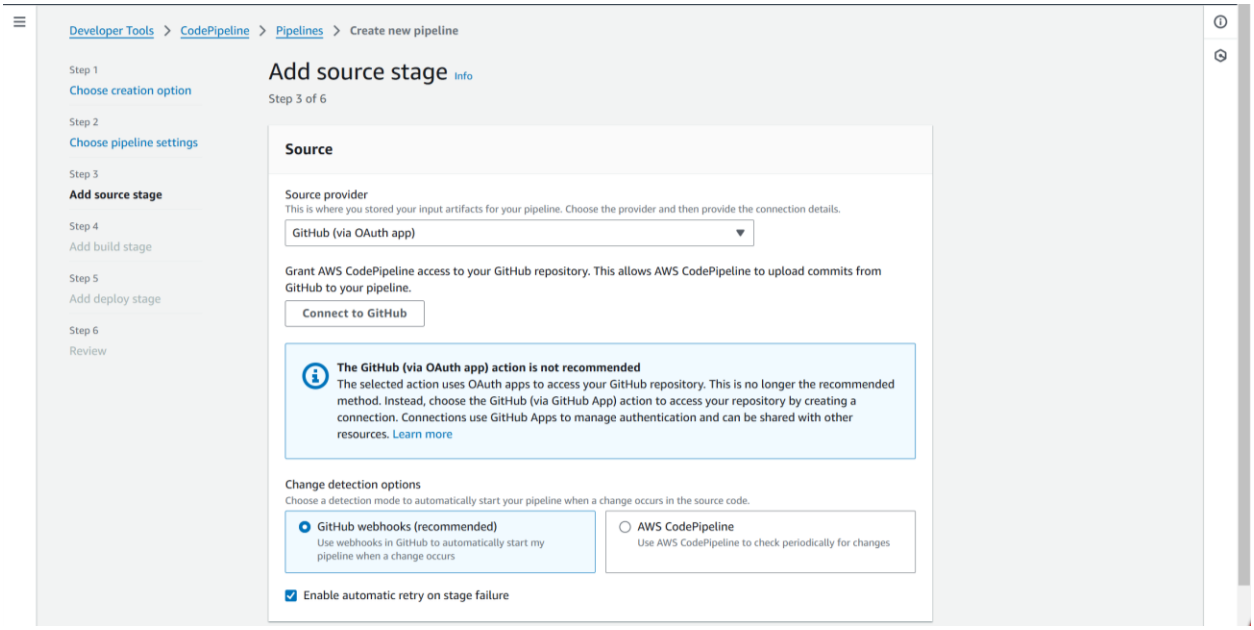
In second step, create a name for your pipeline. AWSDevOpsPipelineA.

Leave default settings. Click Next.

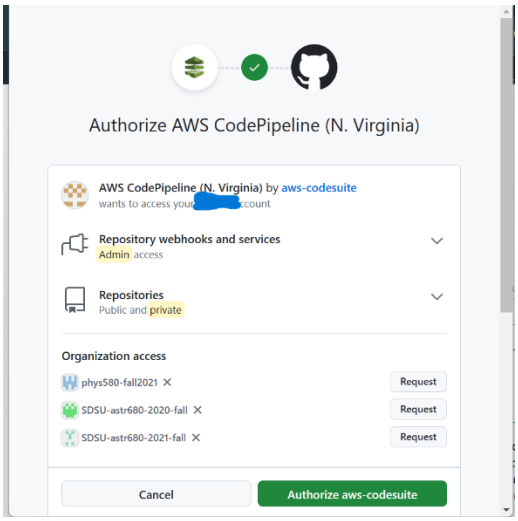
In step 3, Add source stage, you have two options on how to connect to Github.



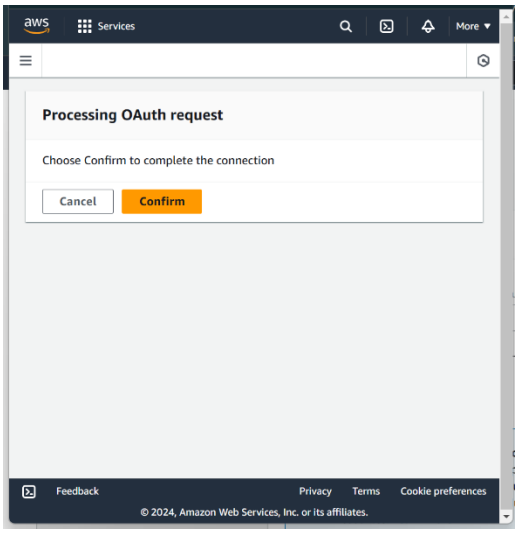
Either GitHub App or OAuth app. In this case, we will use OAuth app.



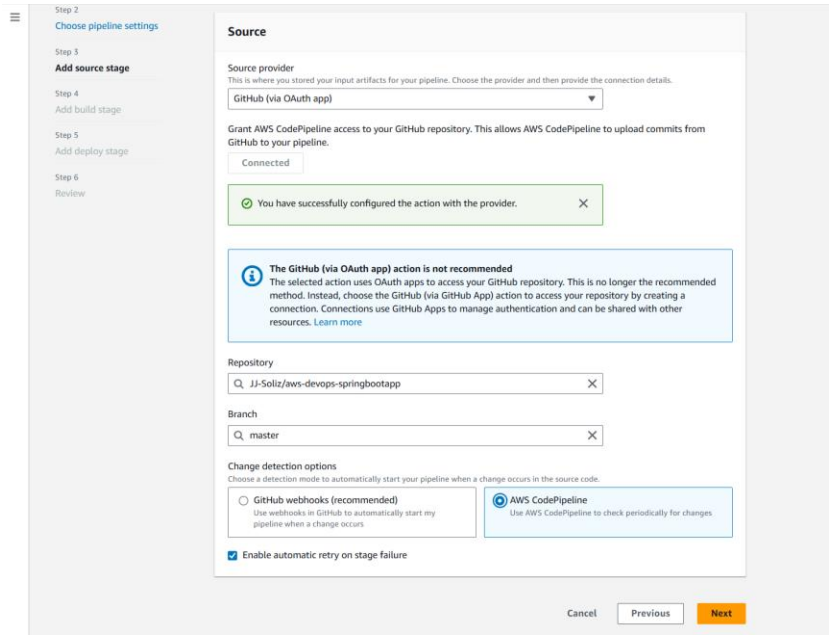
Click on Connect to GitHub. A popup window will appear to connect your GitHub account.



Click Authorize aws-codesuite. You will need to provide an passkey, authorization code or password. Once provide you will be sent to this window.



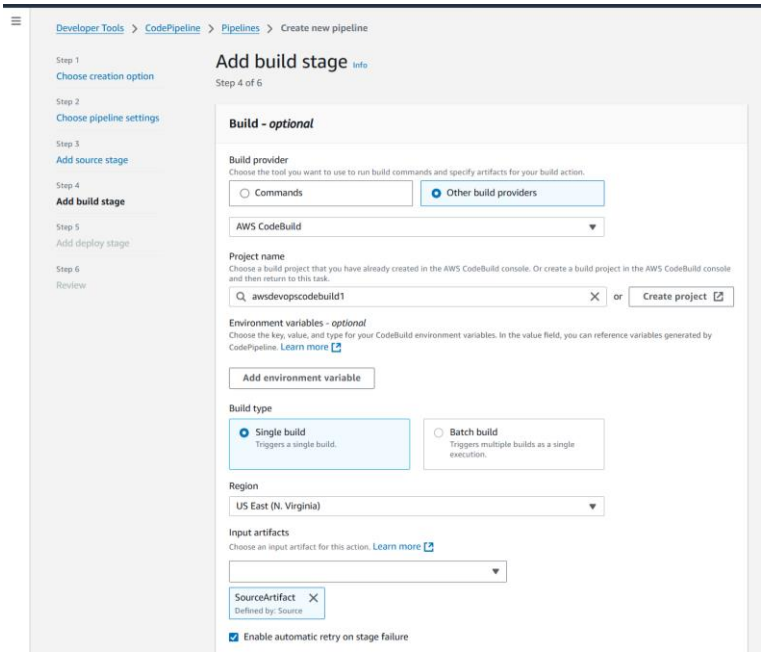
Click Confirm.



With your GitHub account connected, select your Spring Boot App repository and master branch. You have the option on detection mode. In this case, we will have it set to AWS CodePipeline. Select Next.

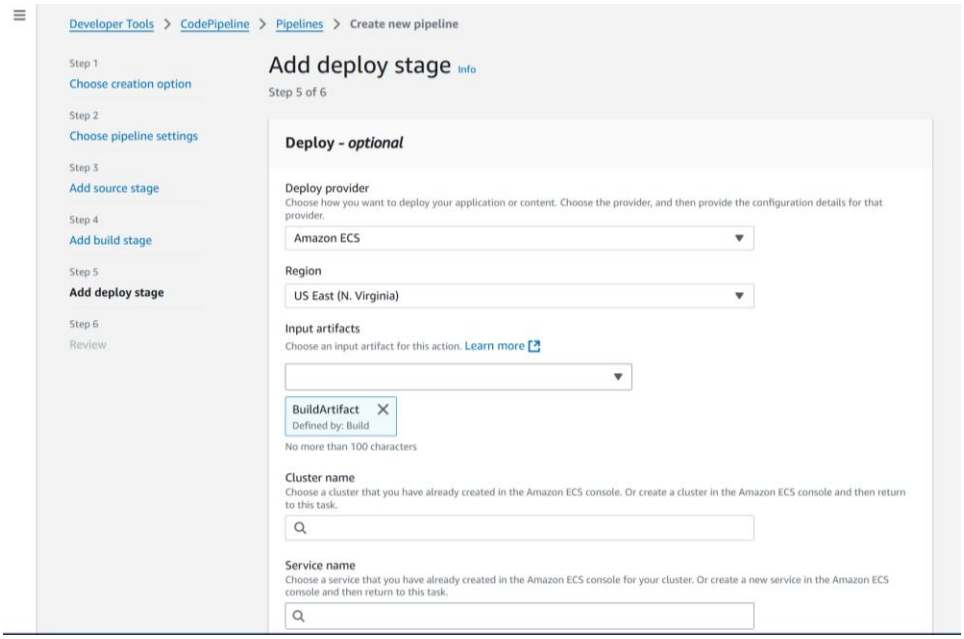
In step 4, Add build stage.

Select Other build providers. Select AWS CodeBuild then under project name, select your code build you created.

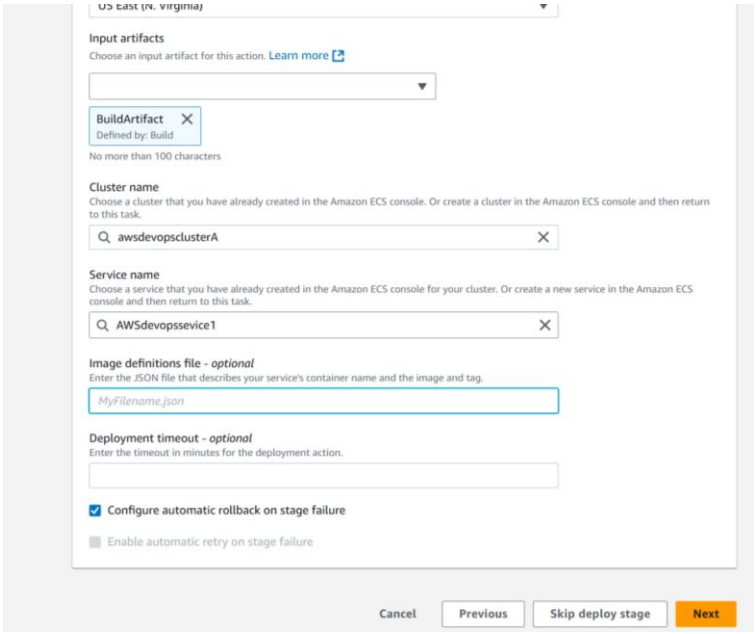


Click Next.

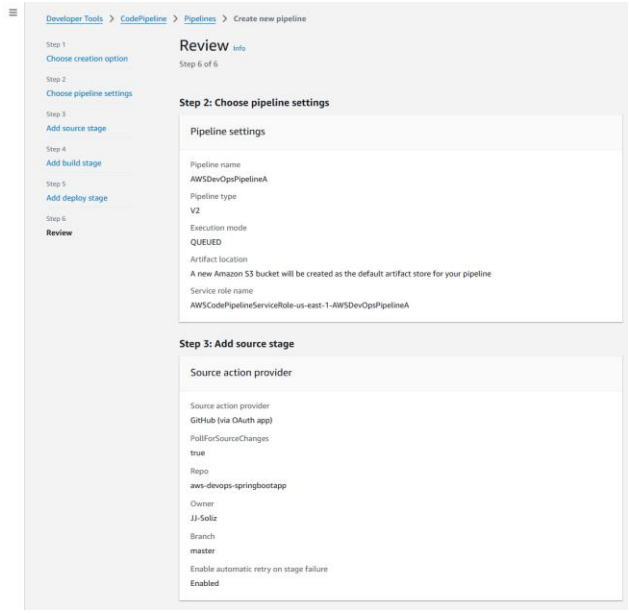
In step 5, Add deploy stage.

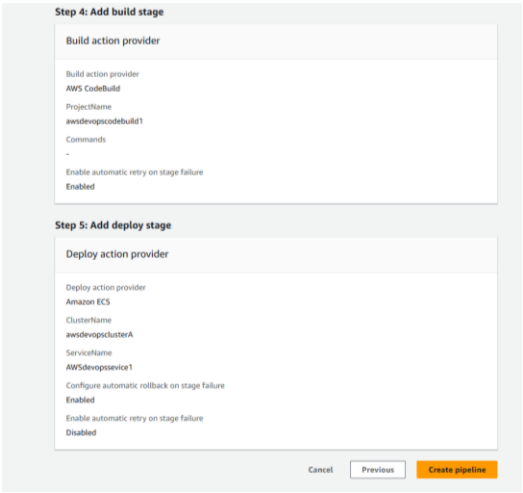


Select Amazon ECS in Deploy provider.

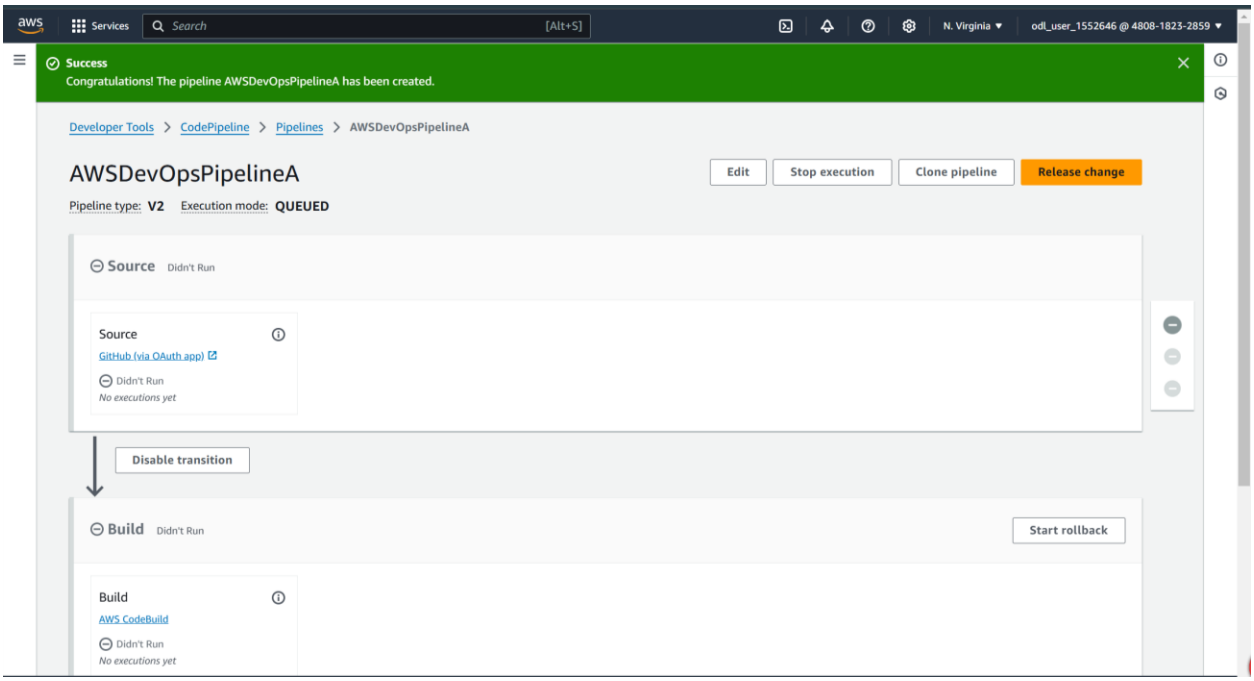


Select your cluster and service you created. Then click Next.





Review your pipeline. Lastly, click on Create pipeline.

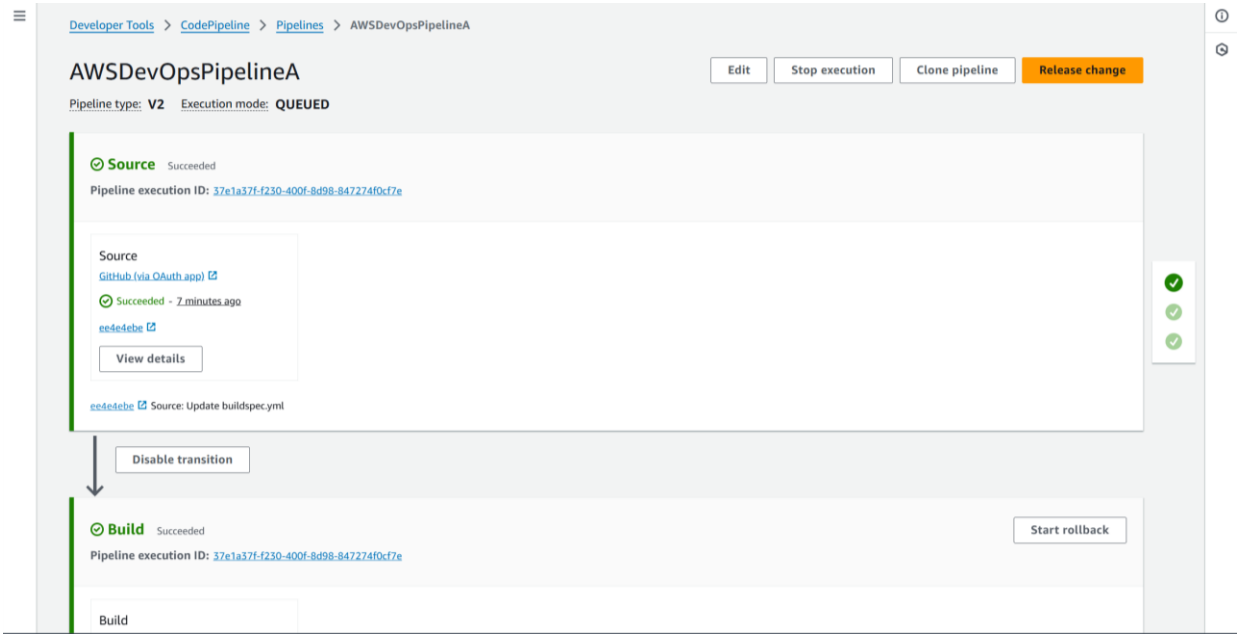


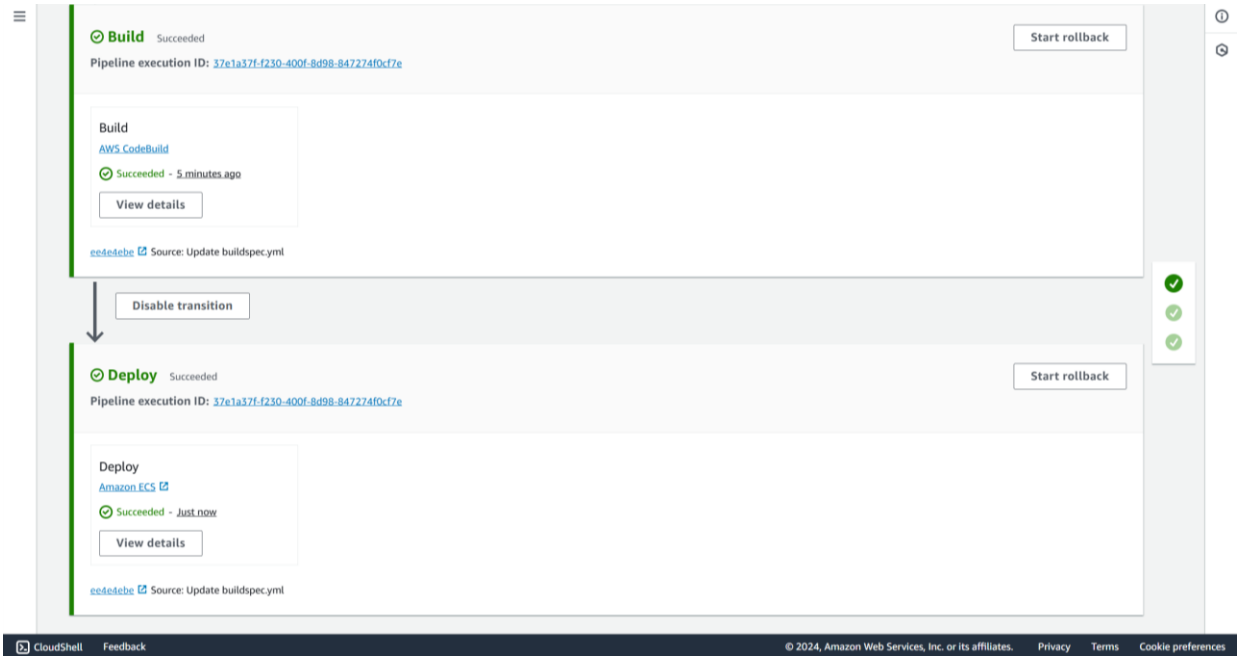
Pipeline created.

4.2 Run Pipeline

Now click on “Release change”.

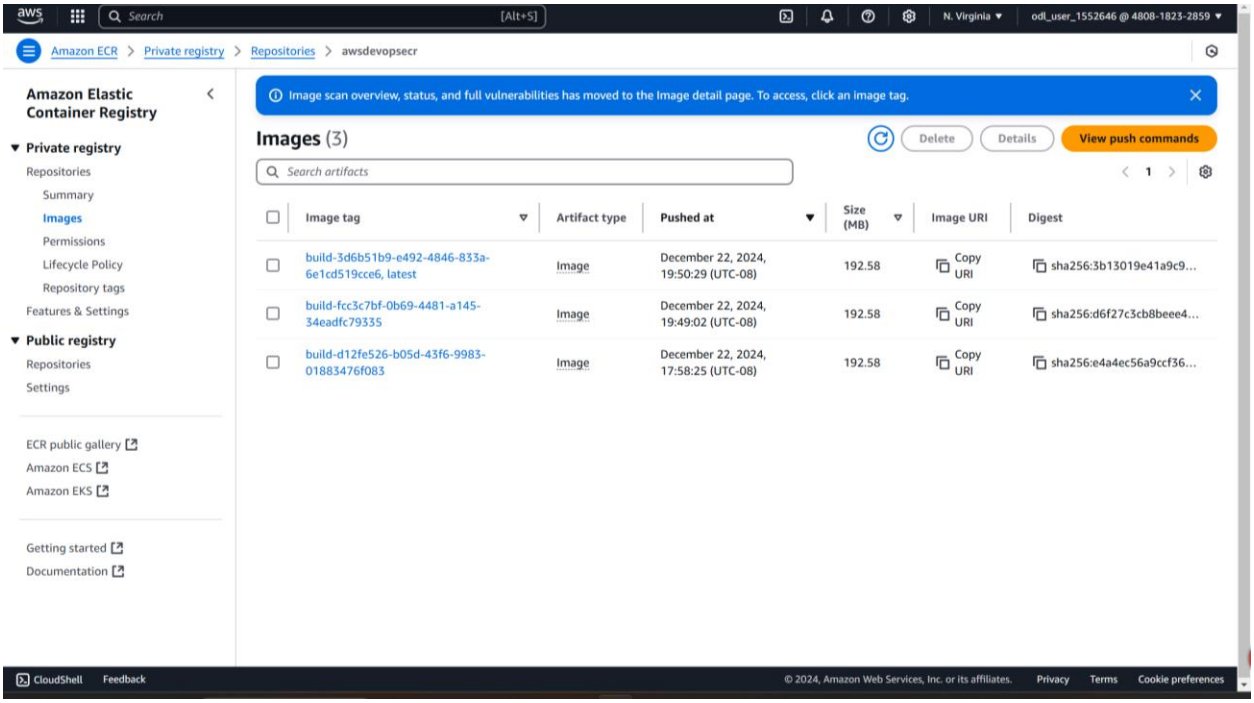
This will take about 10 minutes to complete.



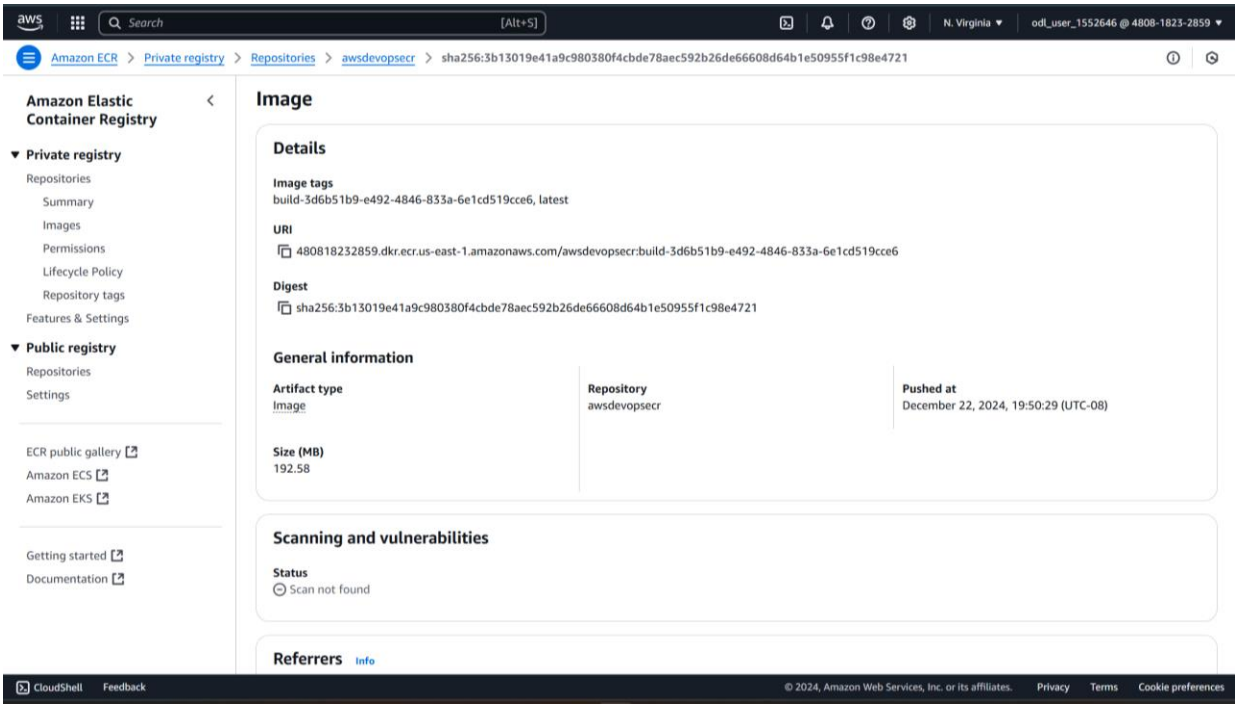


Pipeline has been successfully made.

Go to ECR and click on your repository.



You will see Docker images have been made.



Your CI/CD Pipeline for Spring Boot Application is Ready!

Finish