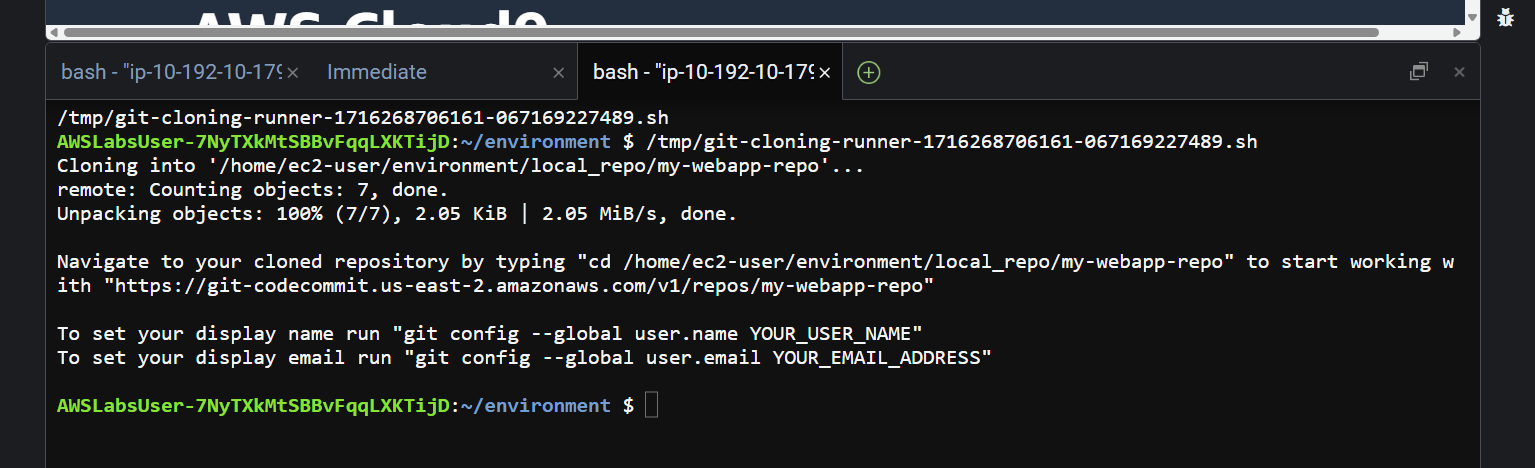
## Lab 5: Performing Blue/Green Deployments with CI/CD Pipelines and Amazon ECS

Objectives:

1. Build custom container images using AWS CodeBuild and store them in Amazon ECR.
2. Set up a CI/CD to build and automate application container image creation using AWS CodePipeline.
3. Host and run a containerized web application using Amazon ECS and AWS Fargate.
4. Configure AWS CodeDeploy to perform a blue/green deployment.
5. Set up a CI/CD to automate the build and deployment of a containerized web application.

Objective 1 - Set up the source files

1.1 Open Cloud9 from URL in the lab, and source files will be updated automatically



Objective 2 - Use AWS CodeBuild and Docker to build your web application image and push the image to Amazon ECR.

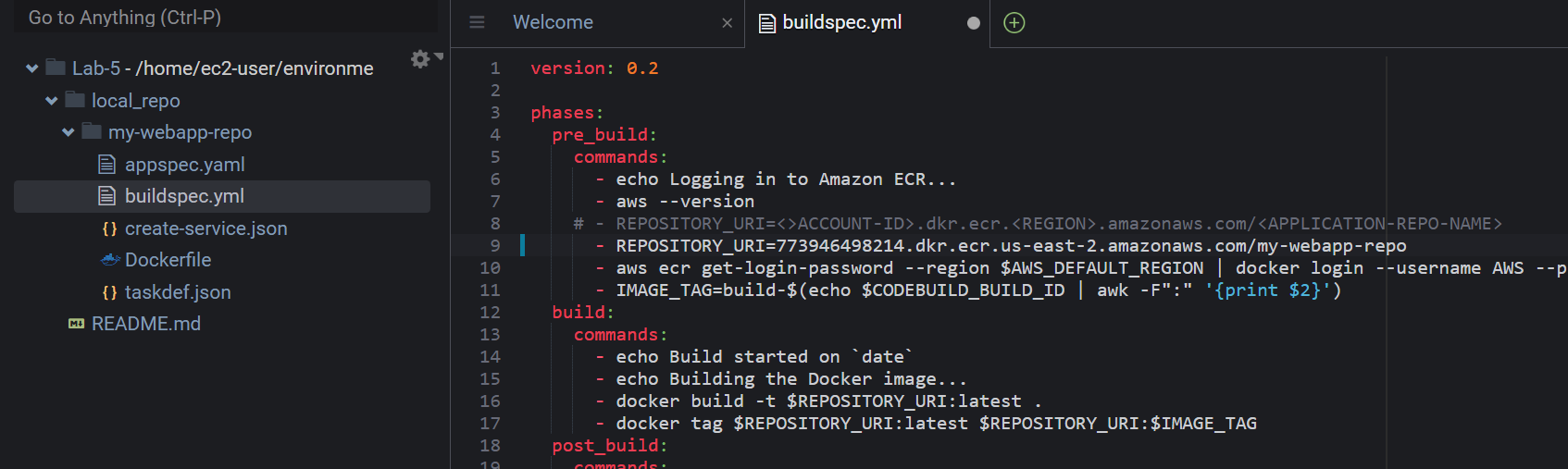
The Docker image built is based on the Dockerfile provided to you, which installs all basic dependencies needed for your application to host a simple web page in the blue background color.

2.1 There is a buildspec.yml file to your source code repository to tell CodeBuild how to build the application image.

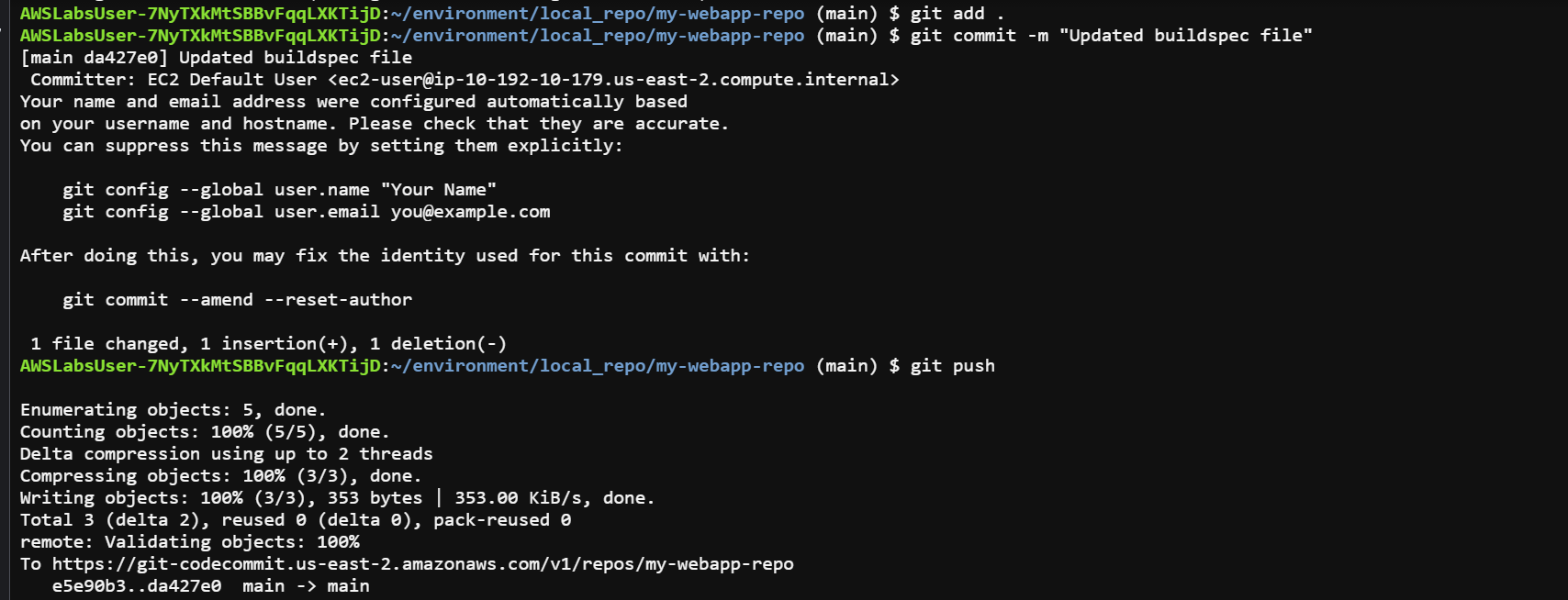
The buildspec.yml file tells AWS CodeBuild how to build and push your Docker image:

1. Pre-build stage: Logs in to Amazon ECR and sets up variables.
2. Build stage: Builds and tags the Docker image.
3. Post-build stage: Pushes the image to Amazon ECR.

2.2 Change the URI value in that file from the value in lab

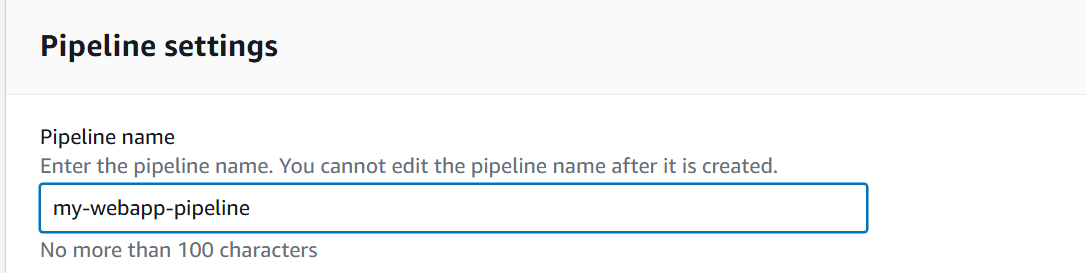


2.3 push the changes you made to the build specification file to the codecommit repo my-webapp-repo.

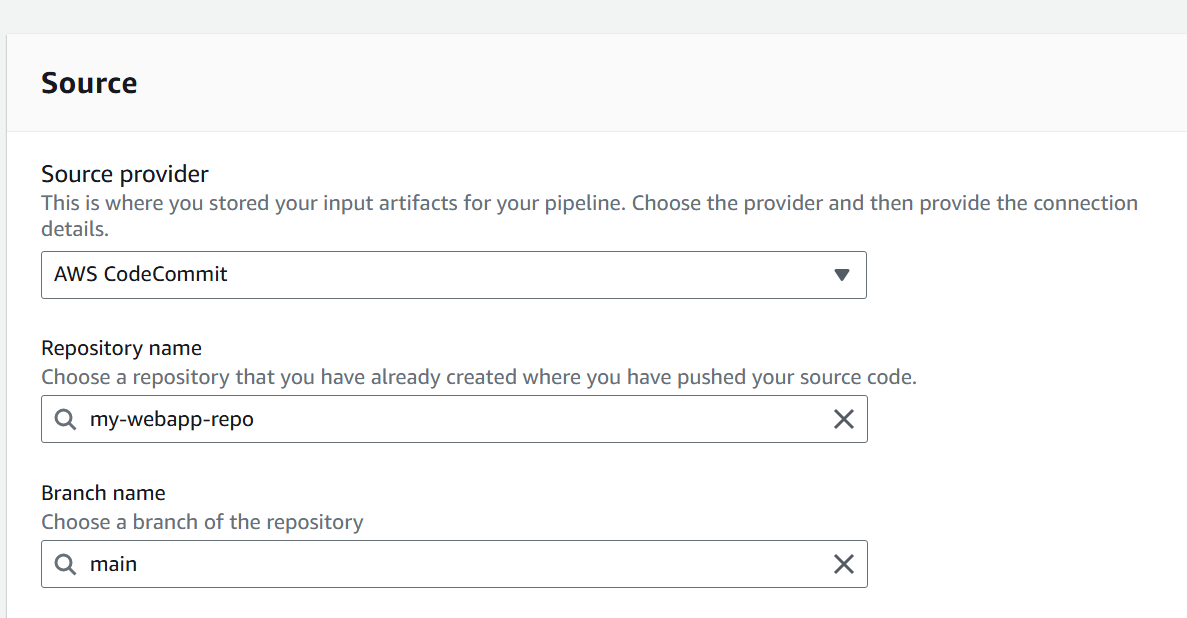


Objective 3 - Create a CI/CD pipeline named my-webapp-pipeline for automating the creation of the application image based on the build specification file.

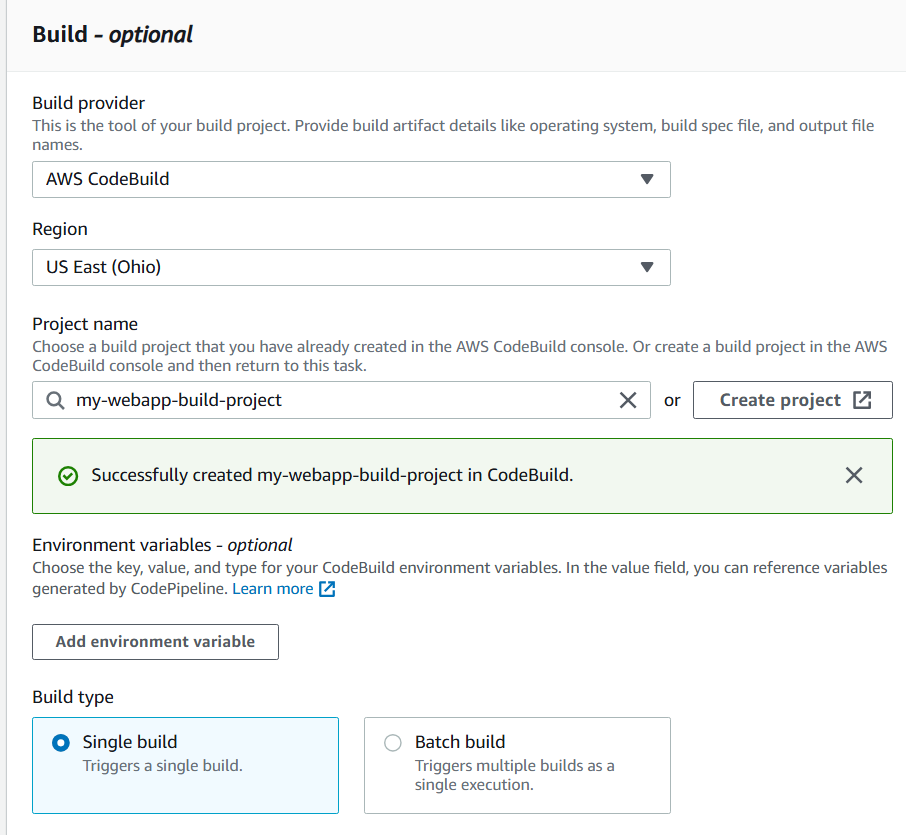
3.1 Open CodePipline, create a new pipeline with following values



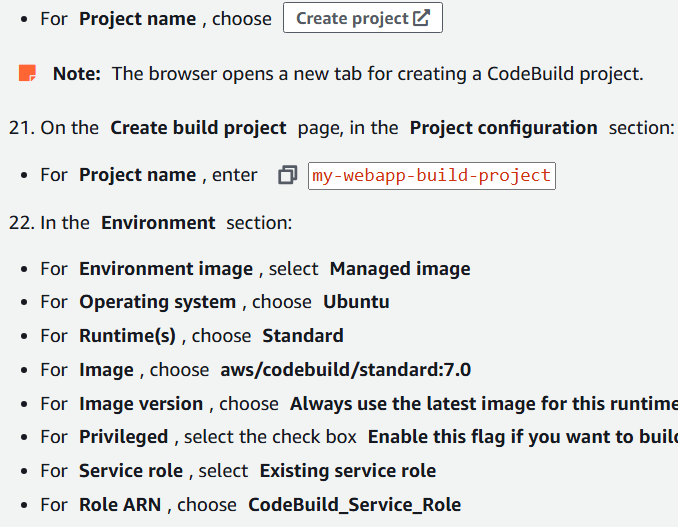
3.2 By adding this stage, any changes you make to the source code is automatically detected and continuously integrated into the pipeline workflow.



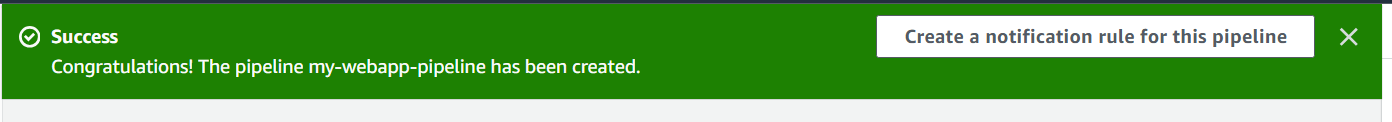
3. 3 add a stage to build your application artifacts using AWS CodeBuild. AWS CodeBuild refers to the source artifacts and builds the application based on the commands you provided in build specification file



In creating a new project section, fill the following values



3.4 Click on create pipeline



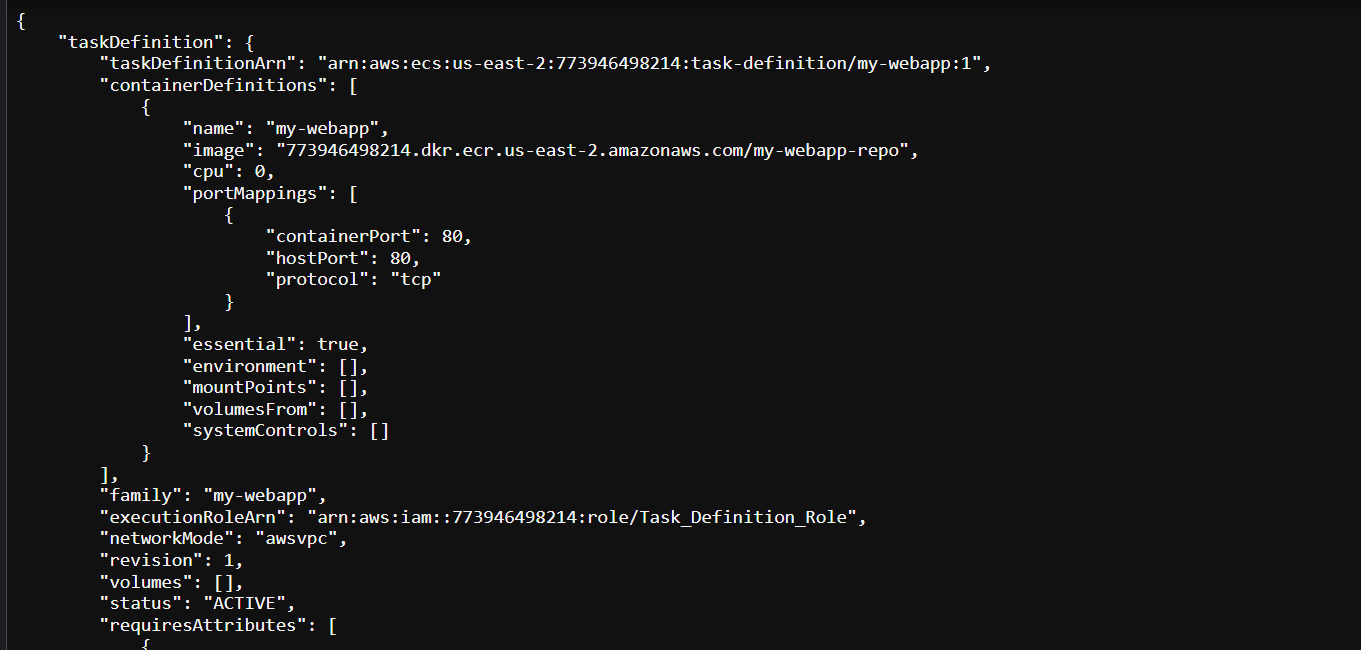
Pipeline created

**Objective 4 - Preparing the ECS cluster to support application deployment**

4.1 Create ECS task definition

Run the following command to create the task definition, it takes the input from taskdef.json file, pre-existing in the Cloud9 IDE.

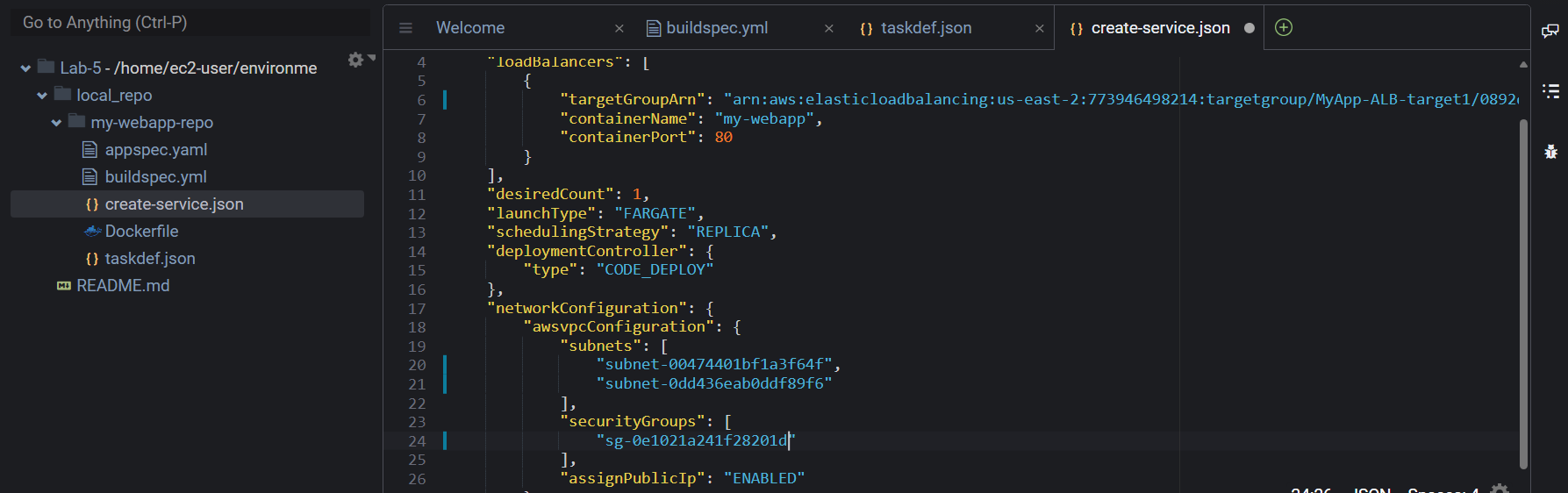




Successfully created an ECS task to define your container’s configurations.

4.2 Create an ECS service MyApp-Web-service based on the task you defined in the previous task.

Open create\_service.json file and insert the values for variables, that is suggested in lab manual

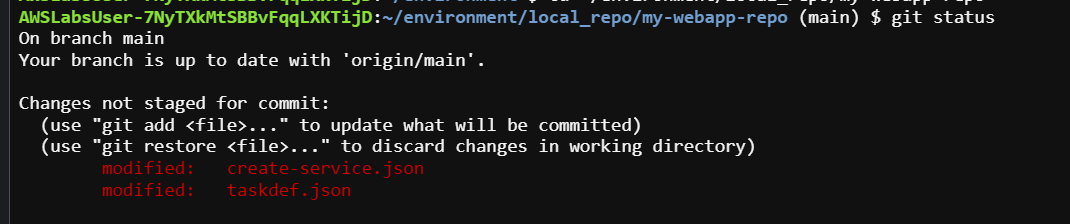


4.3 To create the service for your ECS cluster, run the following command



4.4 successfully created an Amazon ECS service and defined how to run it.

4.5 Push Changes to the source CodeCommit Repo

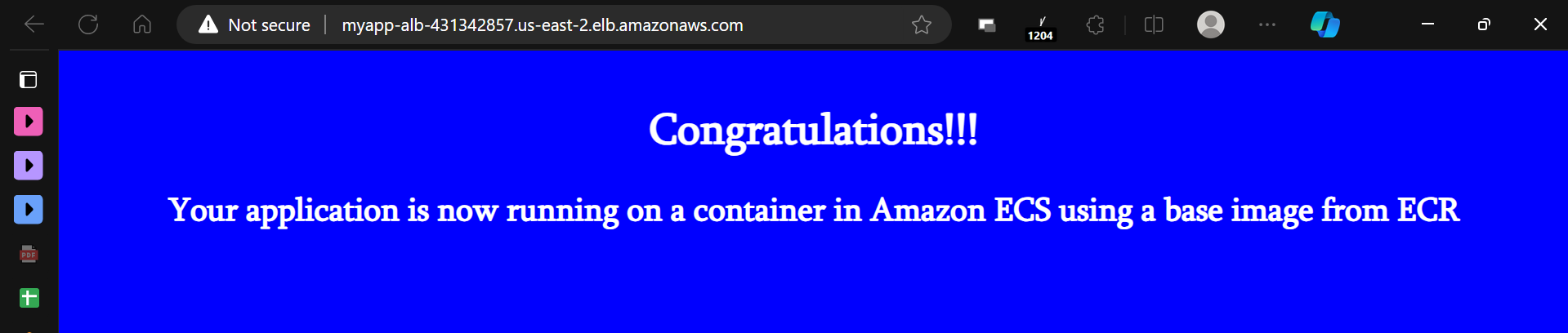


Successfully configured your ECS cluster to support the deployment of a demonstration application. ECS task and ECS service are now in effect to deploy application automatically in the next task.

NOTE - By making the required application source code available and configuring ECS service in the previous tasks, you have manually deployed the application

**Objective 5 - Review the current running state of your web application**

5.1 Put **LoadBalancerDNSName**  value from lab manual in new tab, and webapp opens

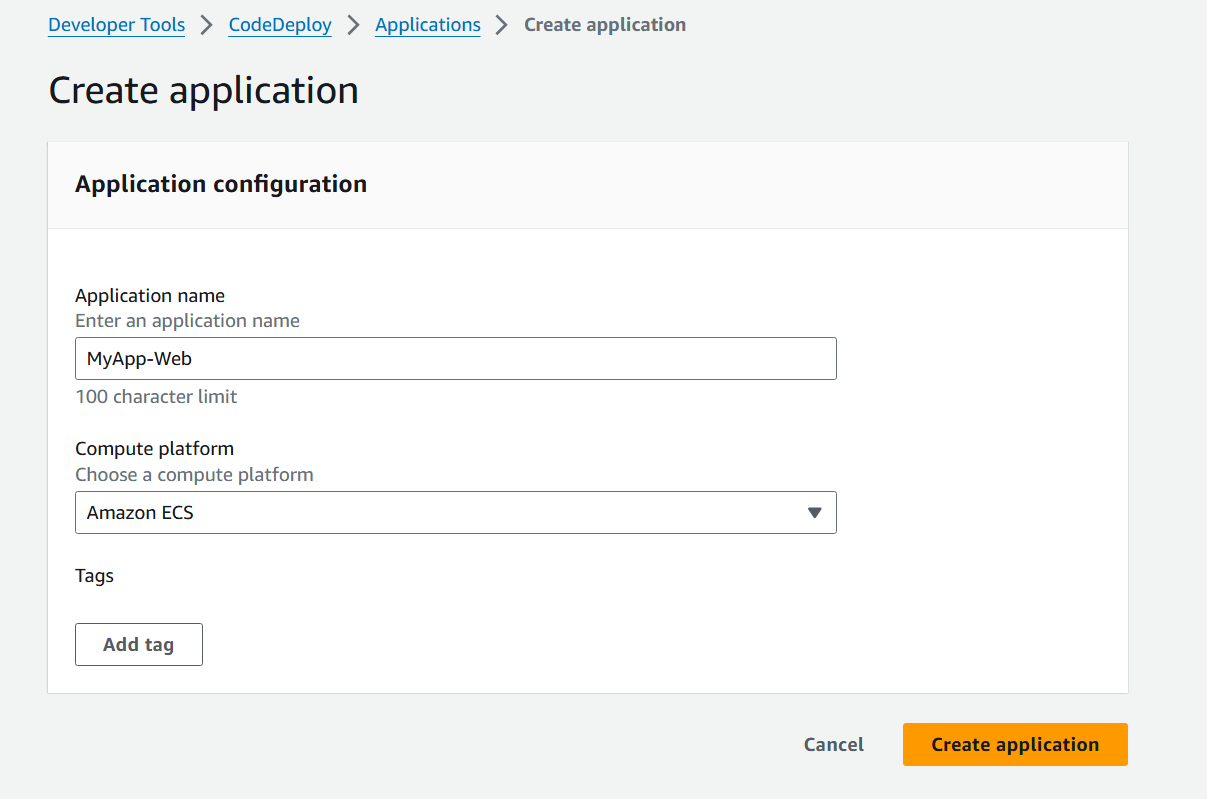


Objective 6 - Configure the Blue/green deployment

1. add the Deploy stage to the pipeline.
2. current running state of the application (blue).
3. you make a code change to start the pipeline to initiate an automatic application deployment. You notice AWS CodeDeploy creating the new application environment (green)

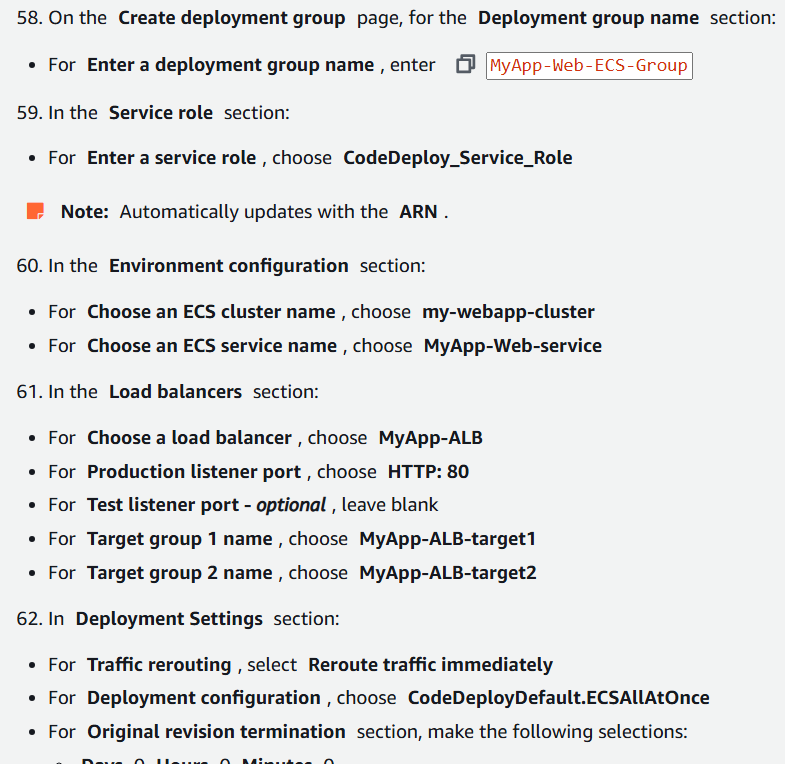
6.1 CREATE A CODEDEPLOY APPLICATION AND DEPLOYMENT GROUP

Open CodeDeploy, create a new application as follows:

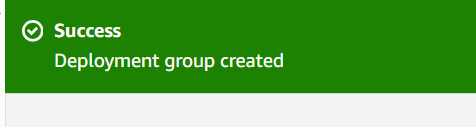


Create a new deployment group

Input the following values

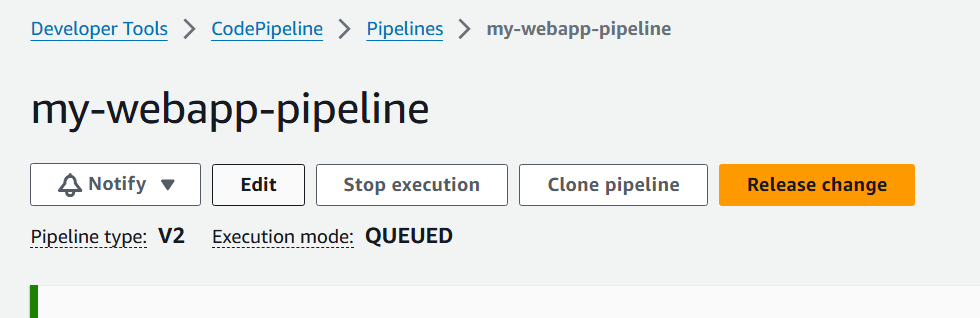


Create the deployment group

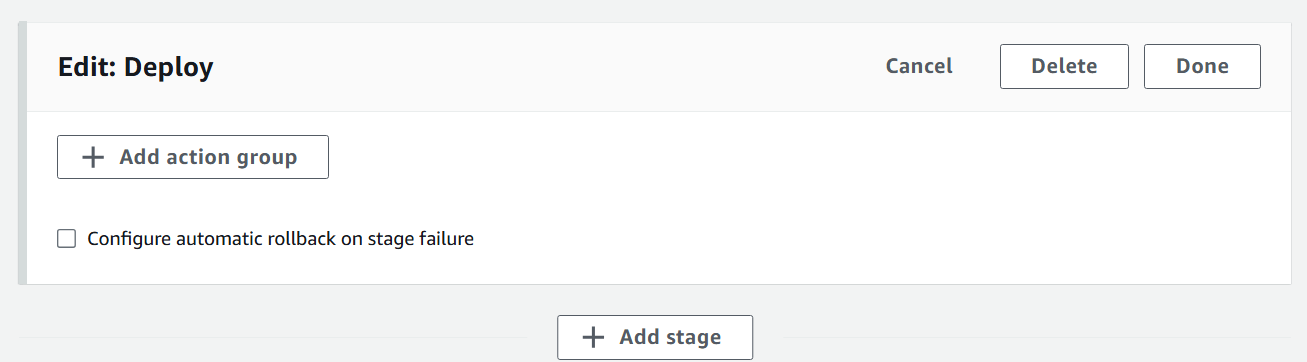


6.2 Adding Deploy Stage

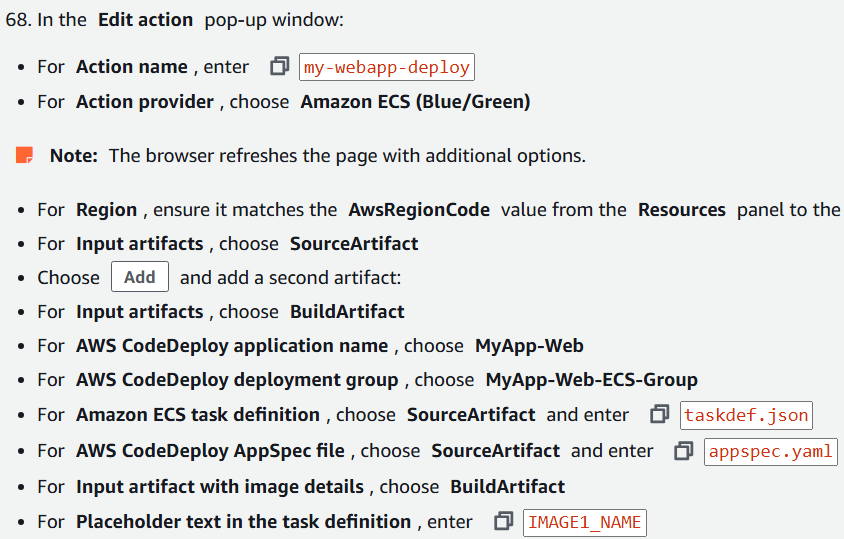
Open the one given pipeline, and click on edit button

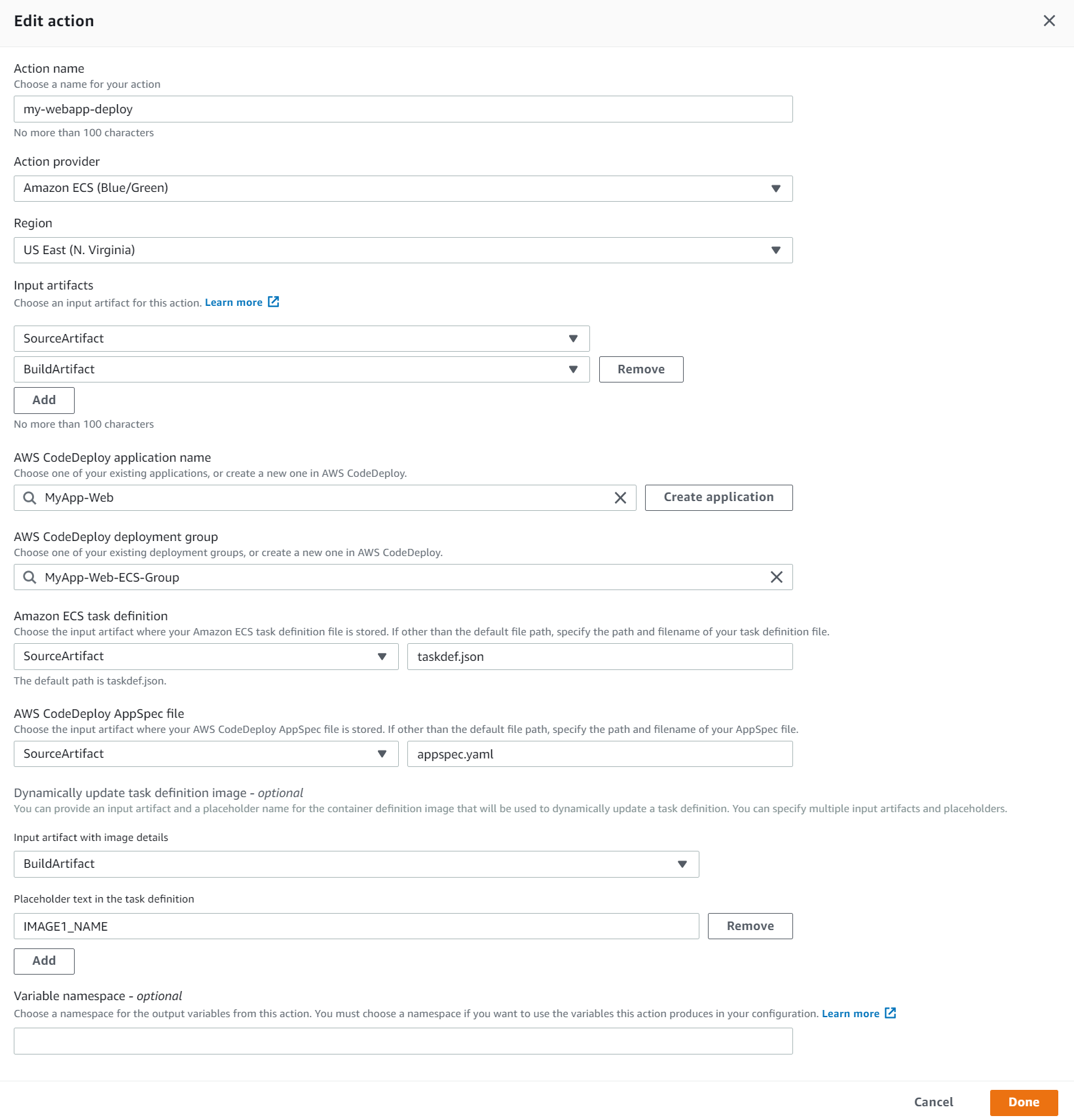


Using the add stage button, add a new stage - Deploy



For creating the stage, give the following values





Save the pipeline.

Successfully configured AWS CodeDeploy to handle Blue/Green deployments for the ECS environment and added a new stage to the pipeline to automated this.

The pipeline is now ready to automate application deployments using a blue/green deployment type when any changes made to the source code in my-webapp-repo.

**Objective 7 - : Testing the CodePipeline automation**

You verify that the my-webapp-pipeline is automatically performing blue/green deployments by eliminating the need for manual intervention for building new images and deploying to the containers.

7.1 Change the webpage’s background-color to green in the docker file and push the code changes to my-webapp-repo.

The pipeline initiates the release workflow once changes are pushed.

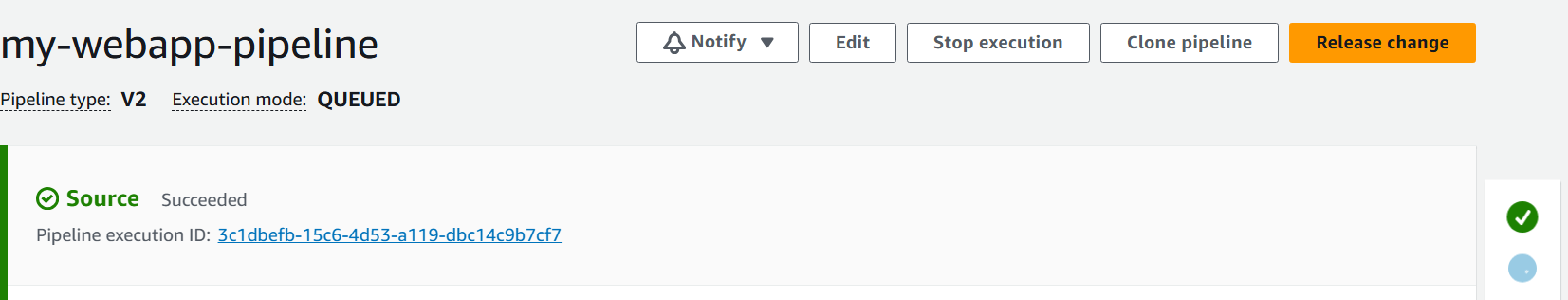
The deploy stage creates the green environment, shifts the traffic to it, and removes the blue environment once deployment is successful.

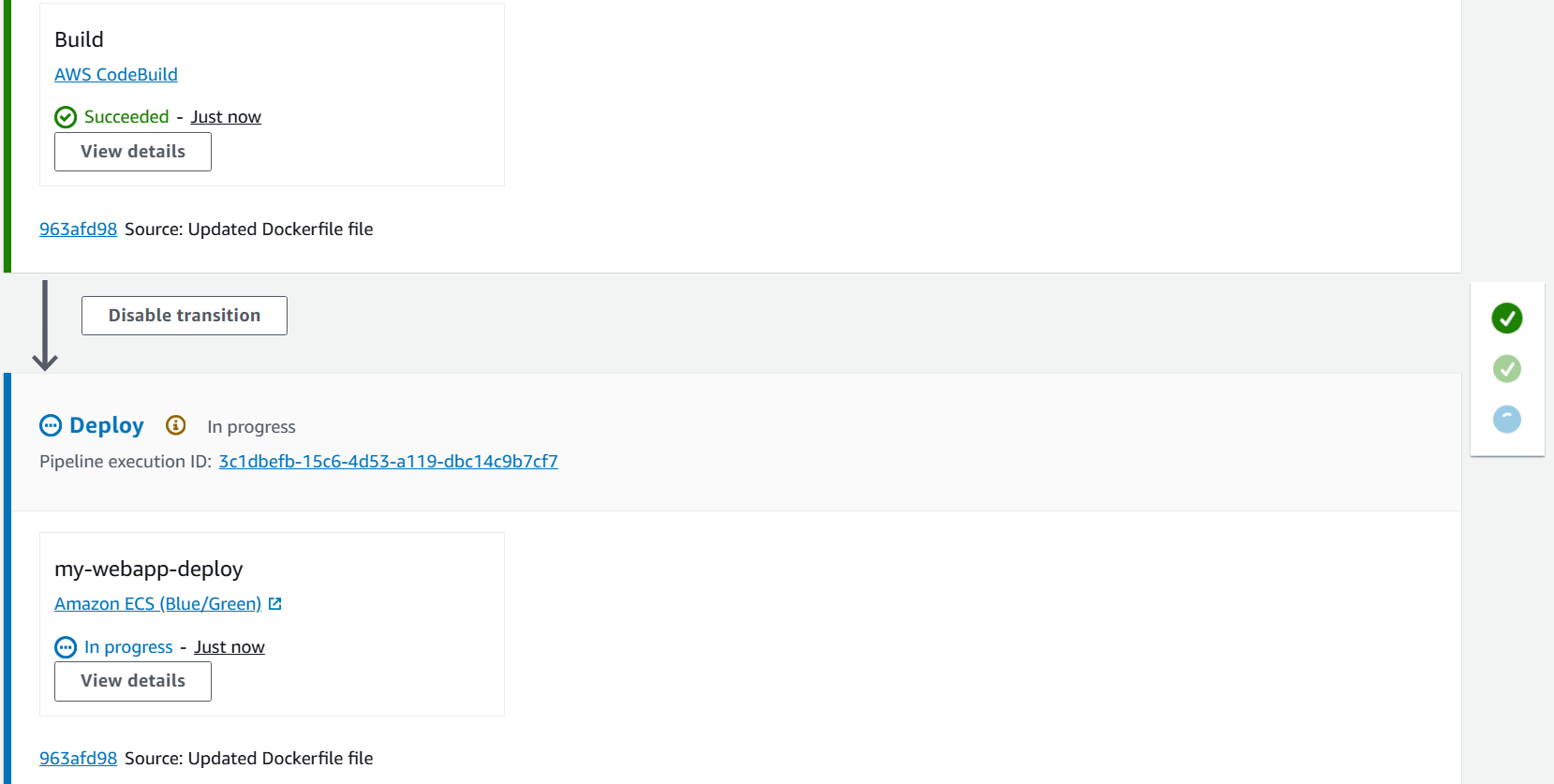
**NOTE - To change color to green, just replace blue with green in the app code**

****

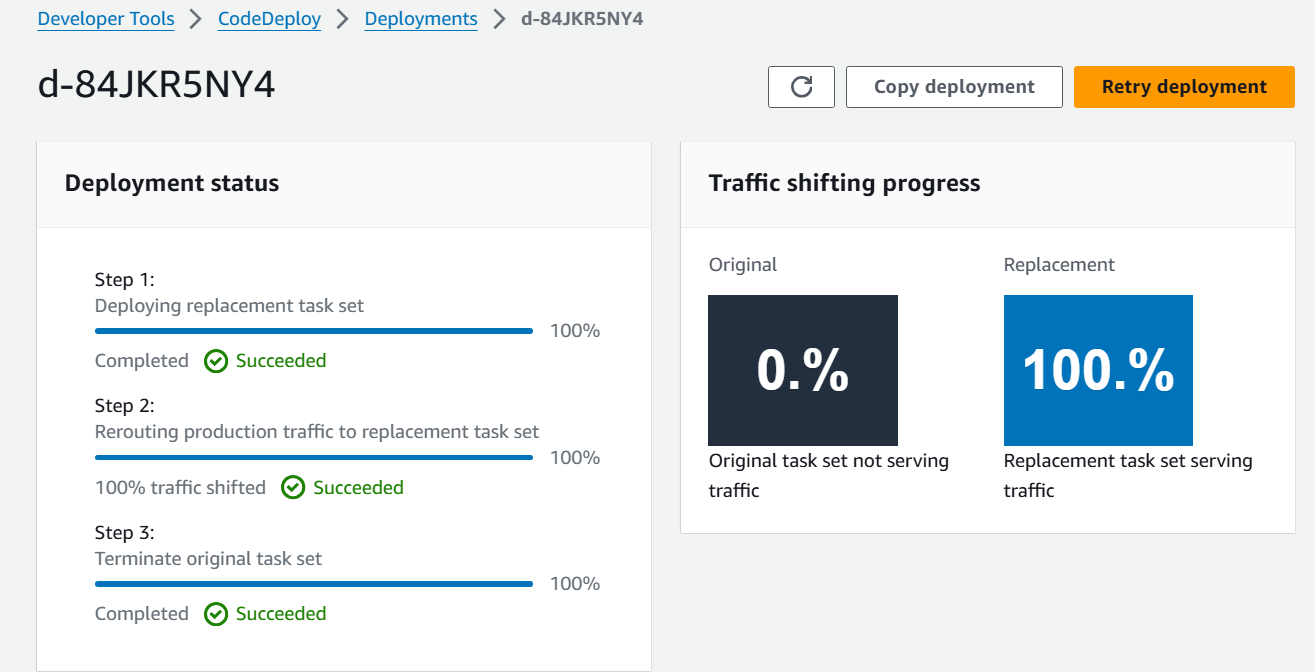
**Push this change in code.**

Now, open the pipeline window, and we see that the changes are being deployed (see the checklist on right side of screenshot)





When deploy stage is being processed, click on the deployment id (in the deployment window), the following is shown



It depicts the deployment status once completed with traffic shifting to the replacement instance.

Now when we visit the web-app, bg color is green

