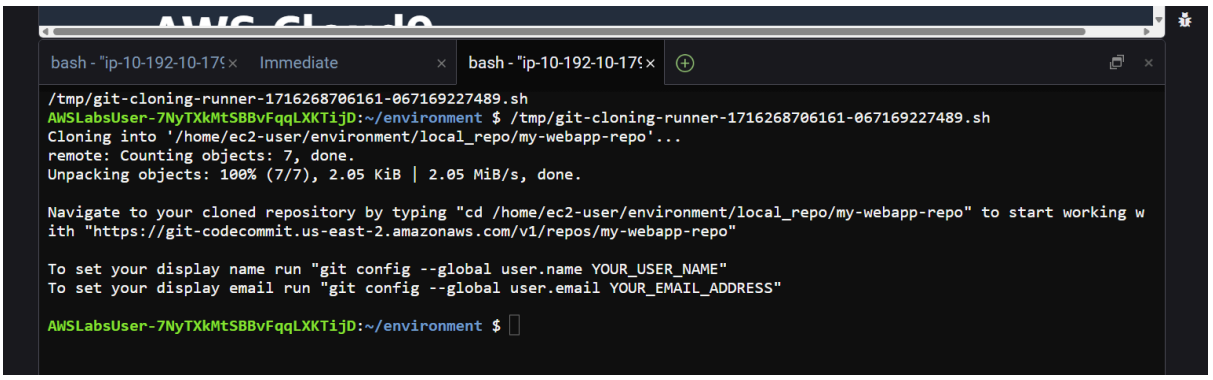
Lab 5: Performing Blue/Green Deployments with CI/CD Pipelines and Amazon ECS

Objectives: Task unclear to me - ECS, and how to configure it.

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.

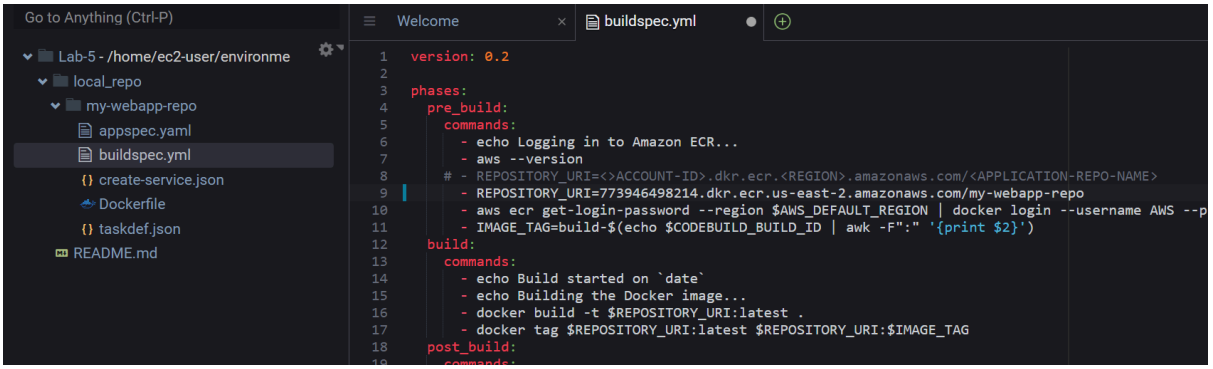
The Docker image built is based on the Dockerfile provided to you, which installs all basic dependencies needed for your application to host a simple web page in the blue background color.

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

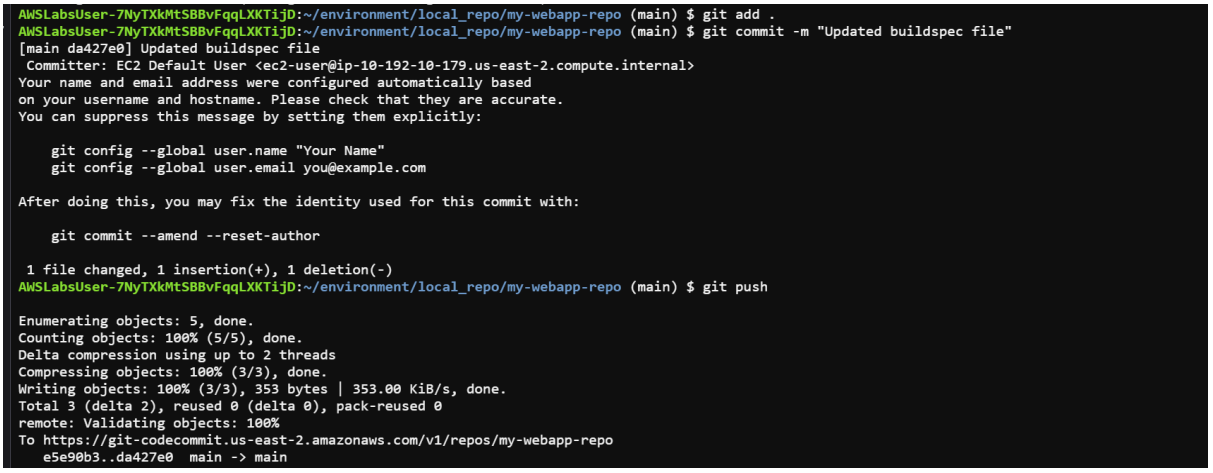
The buildspec.yml file tells AWS CodeBuild how to build and push your Docker image:

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

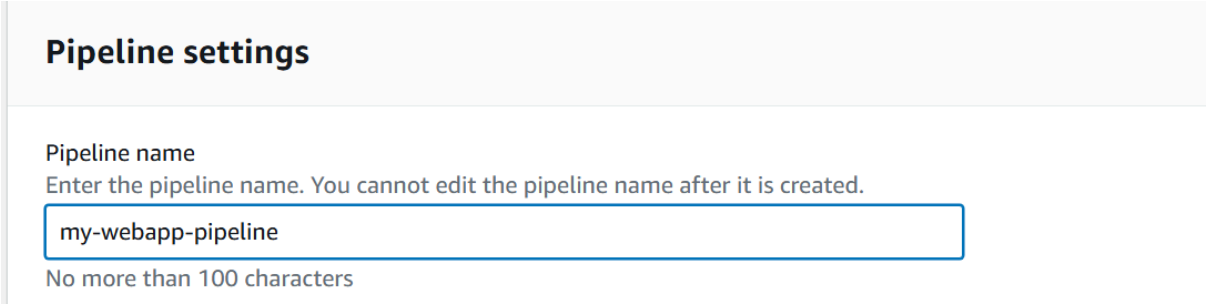


2.3 push the changes you made to the build specification file to the codecommit repo my-webapp-repo.

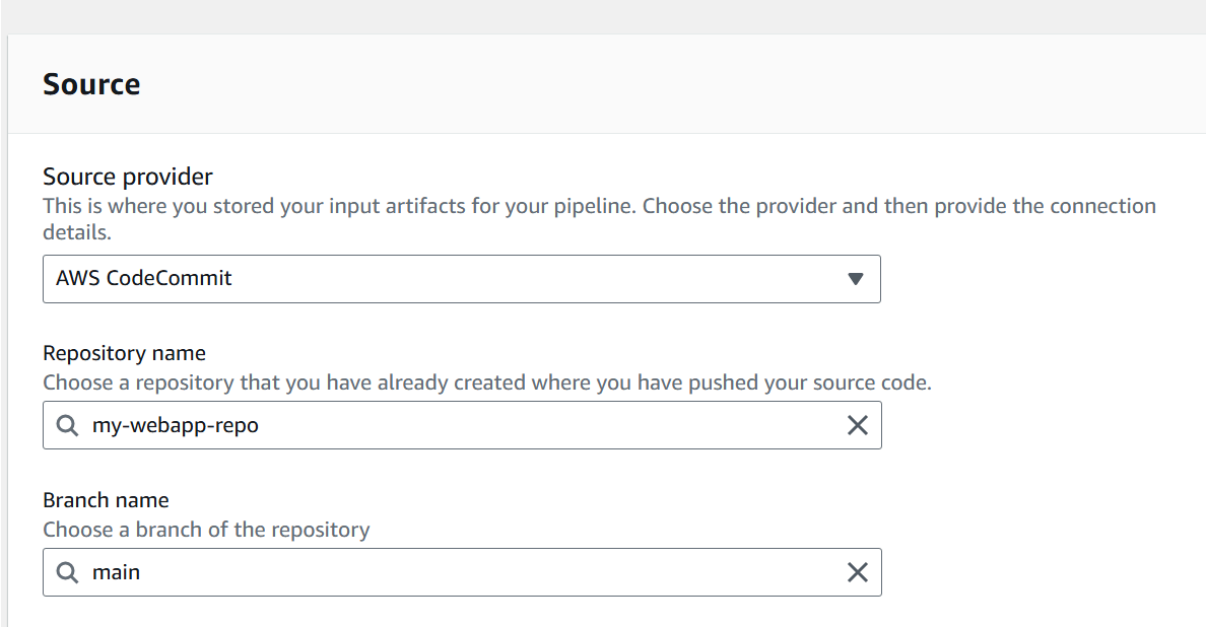


Objective 3 - Create a CI/CD pipeline named my-webapp-pipeline for automating the creation of the application image based on the build specification file.

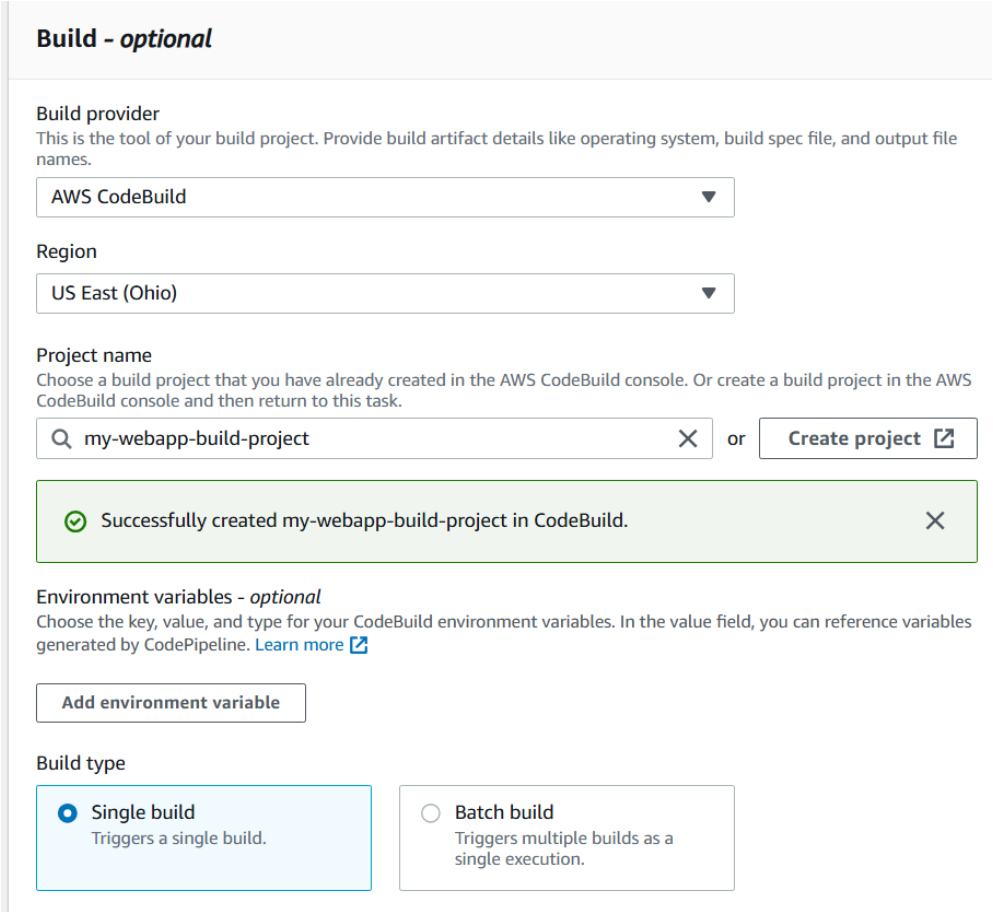
3.1 Open CodePipline, create a new pipeline with following values



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

- For **Project name** , choose Create project

Note: The browser opens a new tab for creating a CodeBuild project.

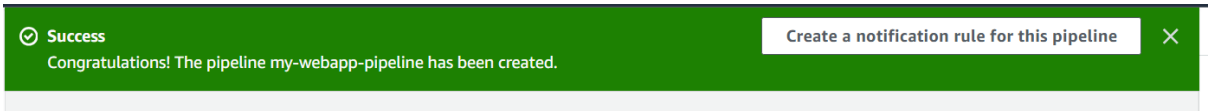
21. On the **Create build project** page, in the **Project configuration** section:

- For **Project name** , enter `my-webapp-build-project`

22. In the **Environment** section:

- For **Environment image** , select **Managed image**
- For **Operating system** , choose **Ubuntu**
- For **Runtime(s)** , choose **Standard**
- For **Image** , choose **aws/codebuild/standard:7.0**
- For **Image version** , choose **Always use the latest image for this runtime**
- For **Privileged** , select the check box **Enable this flag if you want to build**
- For **Service role** , select **Existing service role**
- For **Role ARN** , choose **CodeBuild_Service_Role**

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.

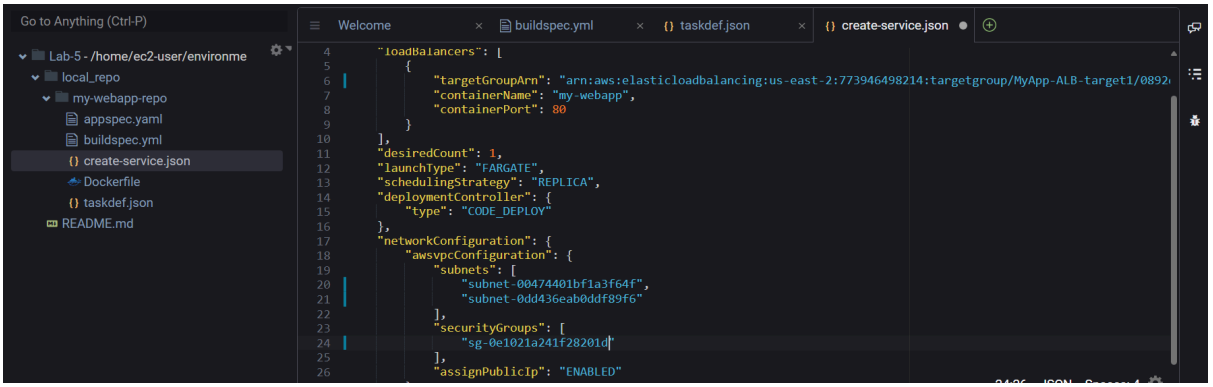
```
aws ecs register-task-definition --cli-input-json file://taskdef.json
```

```
{
    "taskDefinition": {
        "taskDefinitionArn": "arn:aws:ecs:us-east-2:773946498214:task-definition/my-webapp:1",
        "containerDefinitions": [
            {
                "name": "my-webapp",
                "image": "773946498214.dkr.ecr.us-east-2.amazonaws.com/my-webapp-repo",
                "cpu": 0,
                "portMappings": [
                    {
                        "containerPort": 80,
                        "hostPort": 80,
                        "protocol": "tcp"
                    }
                ],
                "essential": true,
                "environment": [],
                "mountPoints": [],
                "volumesFrom": [],
                "systemControls": []
            }
        ],
        "family": "my-webapp",
        "executionRoleArn": "arn:aws:iam::773946498214:role/Task_Definition_Role",
        "networkMode": "awsvpc",
        "revision": 1,
        "volumes": [],
        "status": "ACTIVE",
        "requiresAttributes": [
```


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

```
aws ecs create-service --service-name MyApp-Web-service --cli-input-json file://create-service.json
```

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

4.5 Push Changes to the source CodeCommit Repo

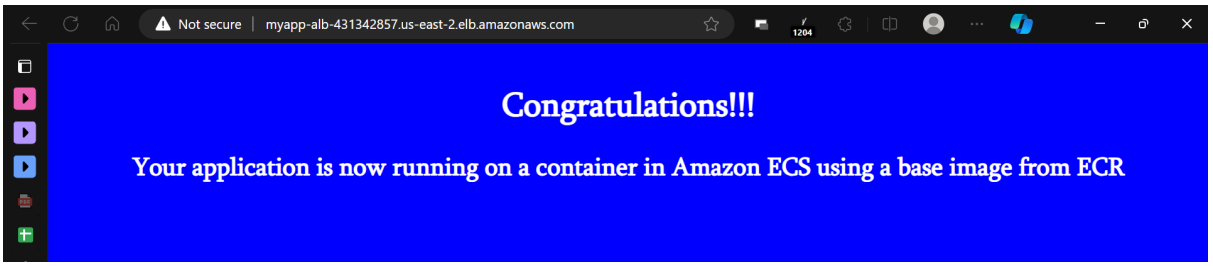
```
AWSLabsUser-7NyTXkMtSBBvFqqLXKTijD:~/environment/local_repo/my-webapp-repo (main) $ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   create-service.json
        modified:   taskdef.json
```

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

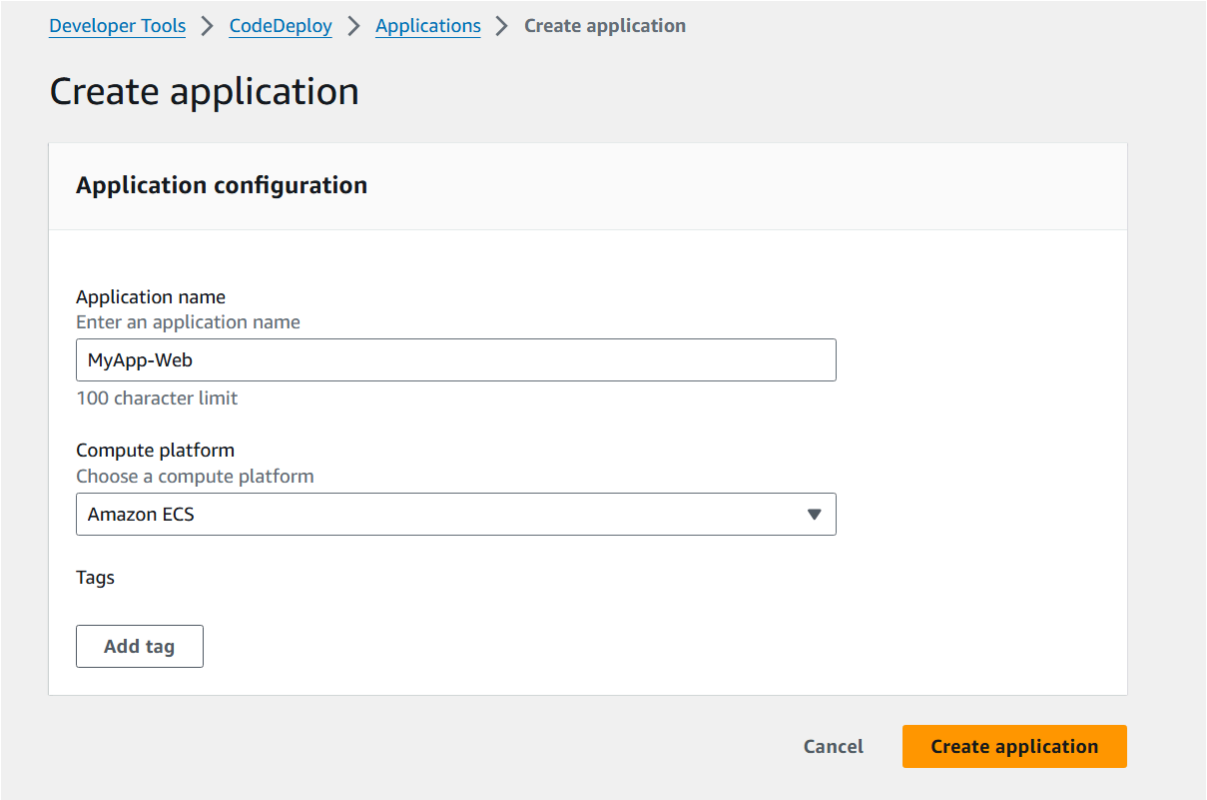


Objective 6 - Configure the Blue/green deployment

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

58. On the **Create deployment group** page, for the **Deployment group name** section:

- For **Enter a deployment group name** , enter `MyApp-Web-ECS-Group`

59. In the **Service role** section:

- For **Enter a service role** , choose **CodeDeploy_Service_Role**

Note: Automatically updates with the **ARN** .

60. In the **Environment configuration** section:

- For **Choose an ECS cluster name** , choose **my-webapp-cluster**
- For **Choose an ECS service name** , choose **MyApp-Web-service**

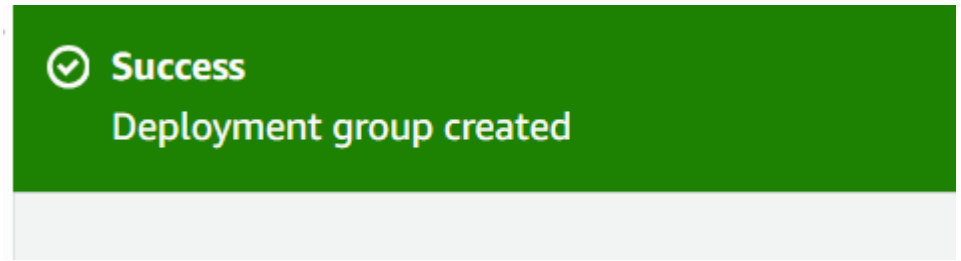
61. In the **Load balancers** section:

- For **Choose a load balancer** , choose **MyApp-ALB**
- For **Production listener port** , choose **HTTP: 80**
- For **Test listener port - *optional*** , leave blank
- For **Target group 1 name** , choose **MyApp-ALB-target1**
- For **Target group 2 name** , choose **MyApp-ALB-target2**

62. In **Deployment Settings** section:

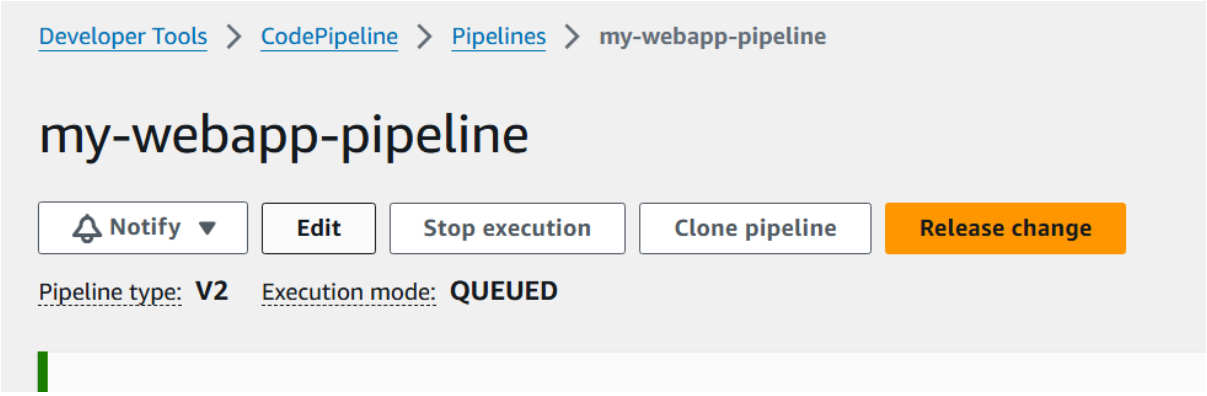
- For **Traffic rerouting** , select **Reroute traffic immediately**
- For **Deployment configuration** , choose **CodeDeployDefault.ECSAllAtOnce**
- For **Original revision termination** section, make the following selections:
  - **Days** 0 **Hours** 0 **Minutes** 0

Create the deployment group

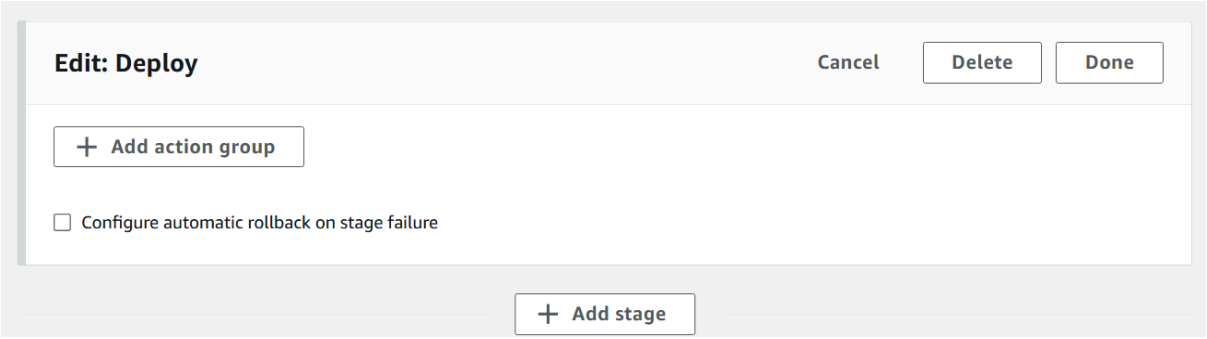


6.2 Adding Deploy Stage

Open the one given pipeline, and click on edit button



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



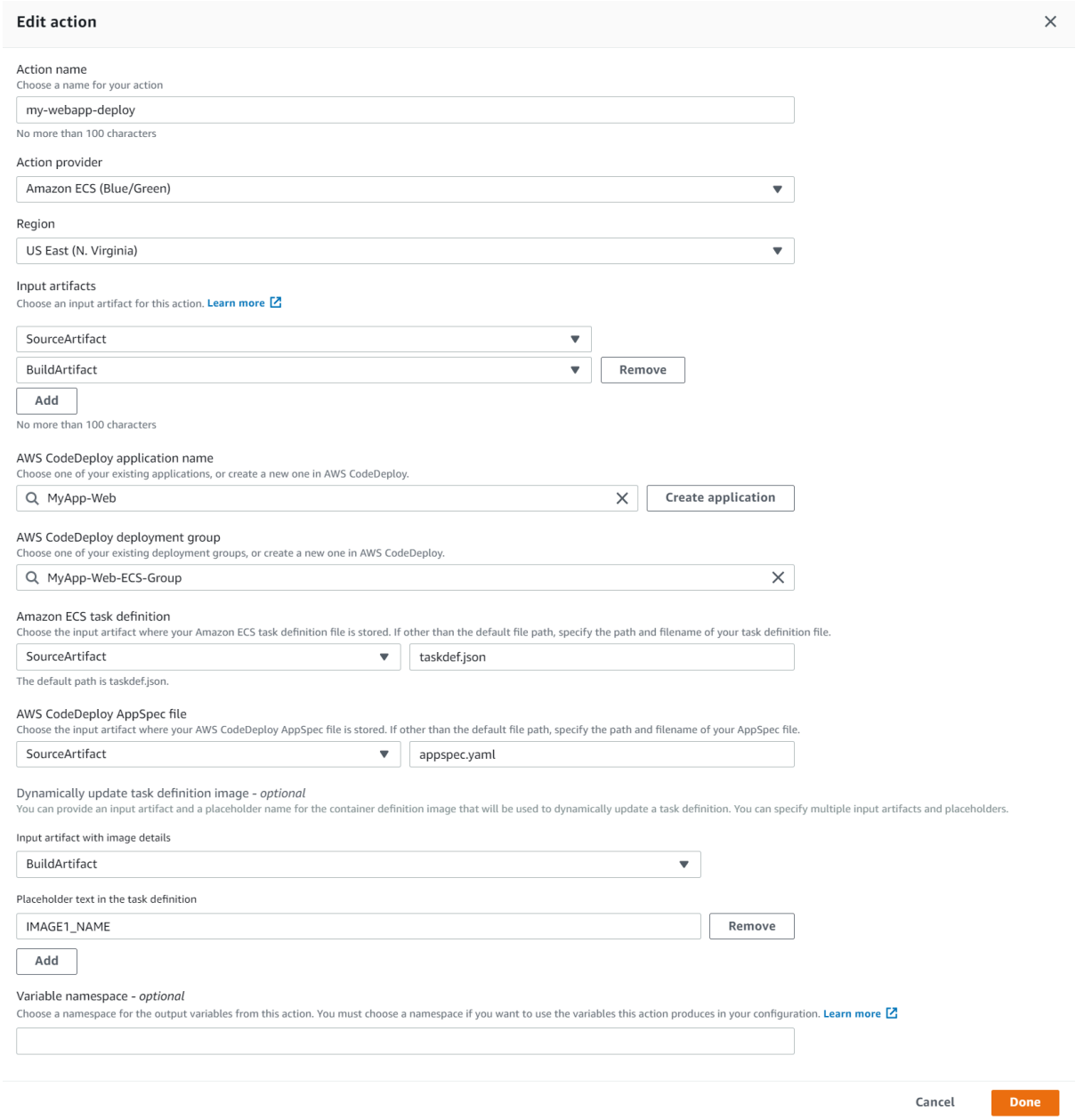
For creating the stage, give the following values

68. In the **Edit action** pop-up window:

- For **Action name** , enter `my-webapp-deploy`
- For **Action provider** , choose **Amazon ECS (Blue/Green)**

Note: The browser refreshes the page with additional options.

- For **Region** , ensure it matches the **AwsRegionCode** value from the **Resources** panel to the
- For **Input artifacts** , choose **SourceArtifact**
- Choose Add and add a second artifact:
- For **Input artifacts** , choose **BuildArtifact**
- For **AWS CodeDeploy application name** , choose **MyApp-Web**
- For **AWS CodeDeploy deployment group** , choose **MyApp-Web-ECS-Group**
- For **Amazon ECS task definition** , choose **SourceArtifact** and enter `taskdef.json`
- For **AWS CodeDeploy AppSpec file** , choose **SourceArtifact** and enter `appspec.yaml`
- For **Input artifact with image details** , choose **BuildArtifact**
- For **Placeholder text in the task definition** , enter `IMAGE1_NAME`



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.

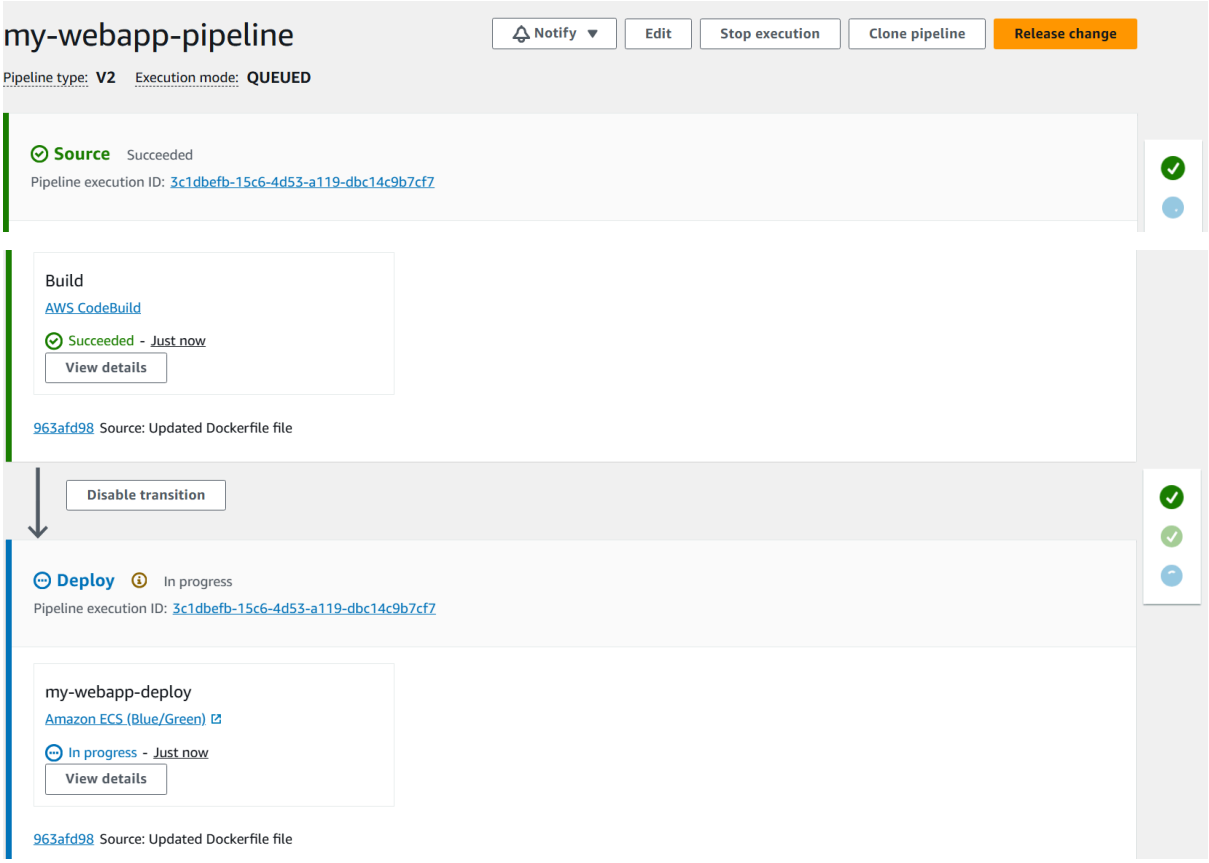
The deploy stage creates the green environment, shifts the traffic to it, and removes the blue environment once deployment is successful.

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

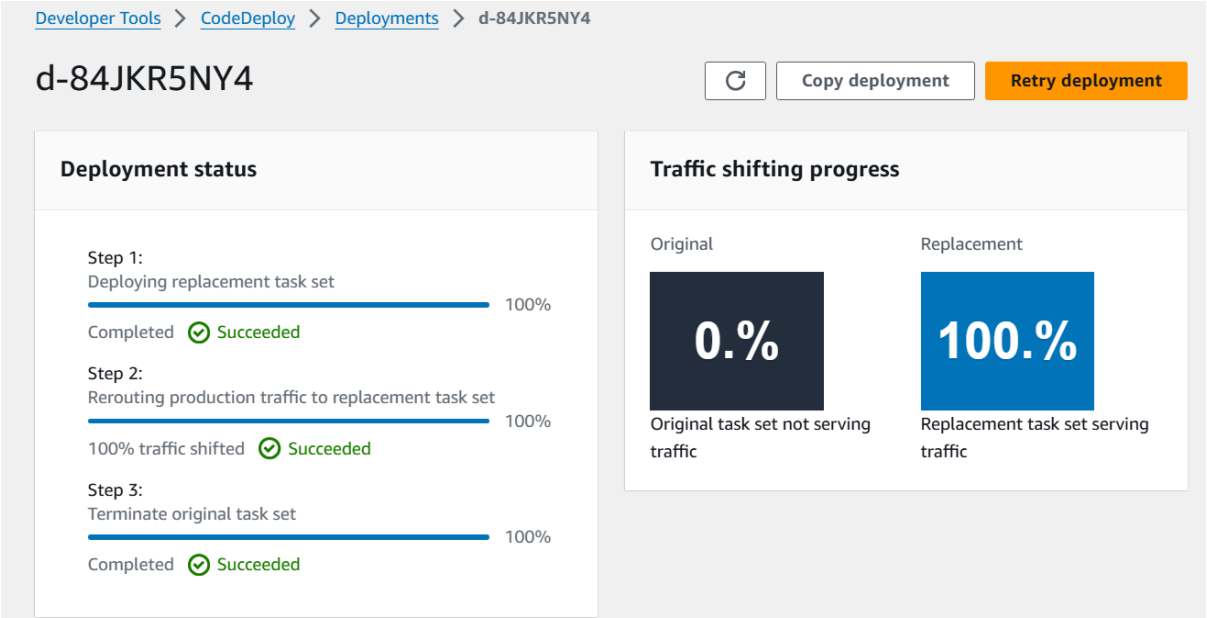
```
argin-top: 40px; background-color:green;}
```

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