😊 Building Code with AWS CodeBuild

AWS CodeBuild is a fully managed continuous integration service that helps developers compile source code, run tests, and generate deployable artifacts. It automates the building process, providing a scalable and cost-effective way to ensure code is ready for deployment. Here are the key details:

Key Features

1. **Fully Managed Service:**
    - AWS CodeBuild takes care of provisioning, managing, and scaling the build servers. This means that users don't need to maintain or configure their own infrastructure for building and testing code.
2. **Build Environment Customization:**
    - Users can select predefined build environments or create custom ones using Docker images. This flexibility supports various programming languages and tools, making it versatile for different development stacks.
3. **Scalability:**
    - CodeBuild scales automatically to handle the build load, enabling it to run multiple builds in parallel. This is especially useful for large teams or projects with frequent code changes.
4. **Integration with AWS Services:**
    - CodeBuild integrates seamlessly with other AWS services such as:
        - **AWS CodePipeline:** For automating the CI/CD pipeline.
        - **AWS CodeCommit:** For source control.
        - **AWS CodeDeploy:** For automating application deployments.
        - **Amazon S3:** For storing build artifacts.
    - This integration helps create a cohesive workflow from source control to deployment.
5. **Security:**
    - AWS CodeBuild offers robust security features, including:
        - **Encryption:** Data can be encrypted both in transit and at rest.
        - **IAM Roles and Policies:** Fine-grained access control can be implemented to restrict who can start builds, view logs, and access other resources.
6. **Pay-as-You-Go Pricing:**

- Users are billed based on the compute resources consumed during the build process, making it a cost-effective solution, particularly for small or sporadic build requirements.

7. **Build Specifications:**
    - Builds are defined using a buildspec.yml file, which outlines the phases and commands to be executed during the build. This file can specify environment variables, build commands, and more.
8. **Logging and Monitoring:**
    - Detailed logs and metrics are available for each build, which can be accessed via the AWS Management Console, AWS CLI, or programmatically via AWS SDKs. This helps in troubleshooting and optimizing the build process.

How It Works

1. **Source Code Fetching:**
    - CodeBuild retrieves source code from sources like AWS CodeCommit, GitHub, Bitbucket, or Amazon S3.
2. **Build Execution:**
    - The build is executed according to the instructions defined in the buildspec.yml file. This includes compiling code, running tests, and generating artifacts.
3. **Artifact Storage:**
    - The resulting artifacts are stored in Amazon S3 or can be pushed to a container registry like Amazon ECR.
4. **Notifications and Reporting:**
    - Build status and results can be communicated via Amazon CloudWatch, SNS, or other notification services, helping keep teams informed about build outcomes.

Use Cases

- **Continuous Integration:** Automatically build and test code whenever changes are pushed to the source repository.
- **Automated Testing:** Run unit, integration, and other automated tests to ensure code quality.
- **Multi-Platform Builds:** Compile and test code for different platforms, such as mobile, web, and desktop.

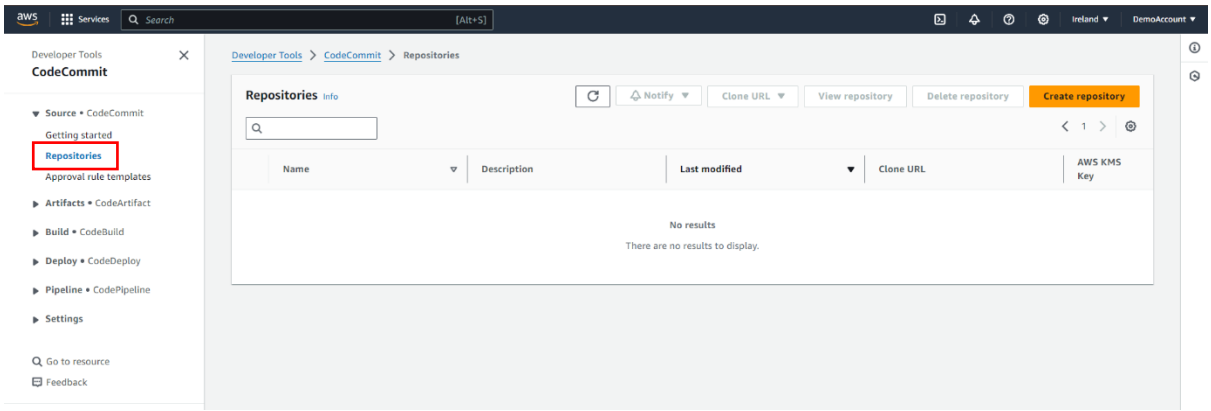
AWS CodeBuild simplifies the process of compiling, testing, and packaging code, allowing developers to focus on writing quality software rather than managing build infrastructure.

End Goal: The end goal is to demonstrate how to set up an automated build and deployment pipeline using AWS CodeBuild and CodePipeline. By following the steps provided, you'll create a repository, push code to AWS CodeCommit, configure a buildspec file, set up a CodePipeline, and deploy a static website to an S3 bucket. This process ensures that any changes made to the code are automatically built, tested, and deployed, streamlining development workflows and enhancing productivity.

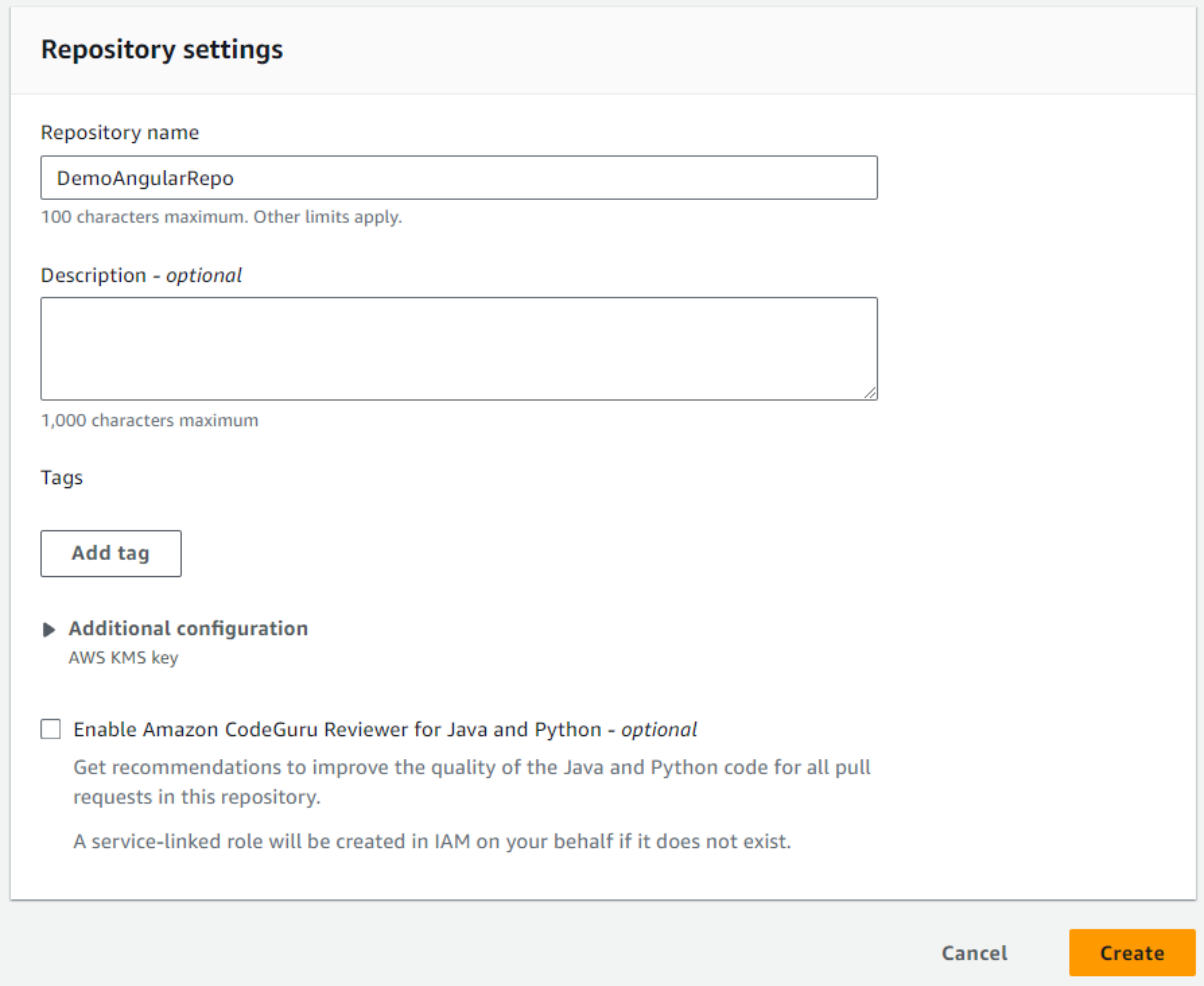
😁 To begin with the Lab:

😁 Step 1: Creating Repo and deployment Bucket

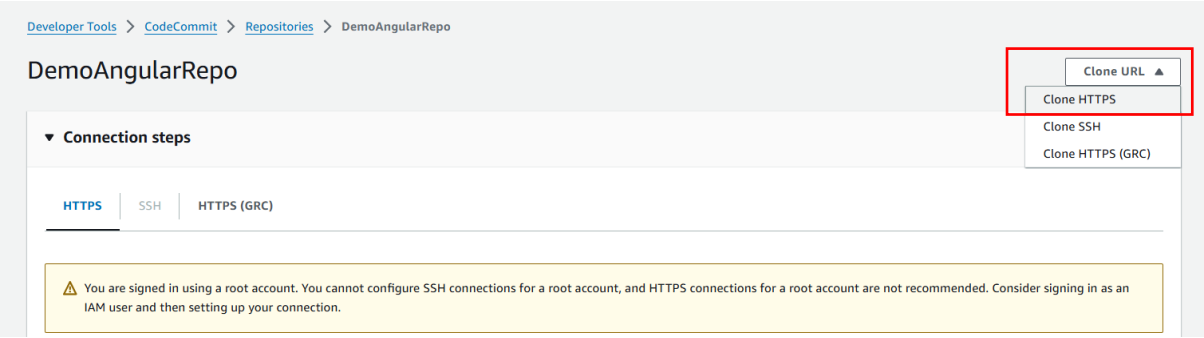
1. In your AWS Console, from the AWS Developer tools you need to expand the source and go to Repositories. Now you need to click on Create Repository.



2. Here you just need to give it a name and click on create.



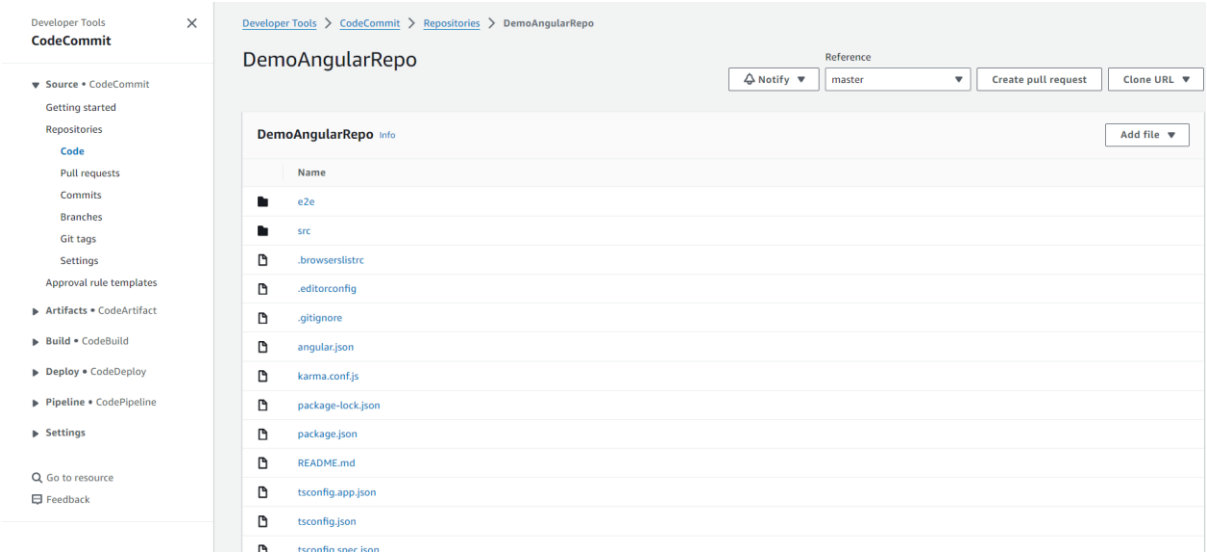
3. Once your repository is created then you need to click on the clone URL and choose HTTPS



4. Now you must download a zipped folder named my-angular-project from our GitHub. After that unzip the folder and open the whole folder in VS Code.
5. When you create an Angular project using Angular CLI, it initializes a local git repository by default. I already made an initial Git commit in our project package, as we did in the previous labs. The only thing we need to do is push it to our new CodeCommit repository. So first, let's set our new CodeCommit repository as the remote repository by typing the 'git remote add origin' command, and pasting the HTTPS URL we cloned from the CodeCommit Console.
6. Below you can see that we have cloned our project to our repository

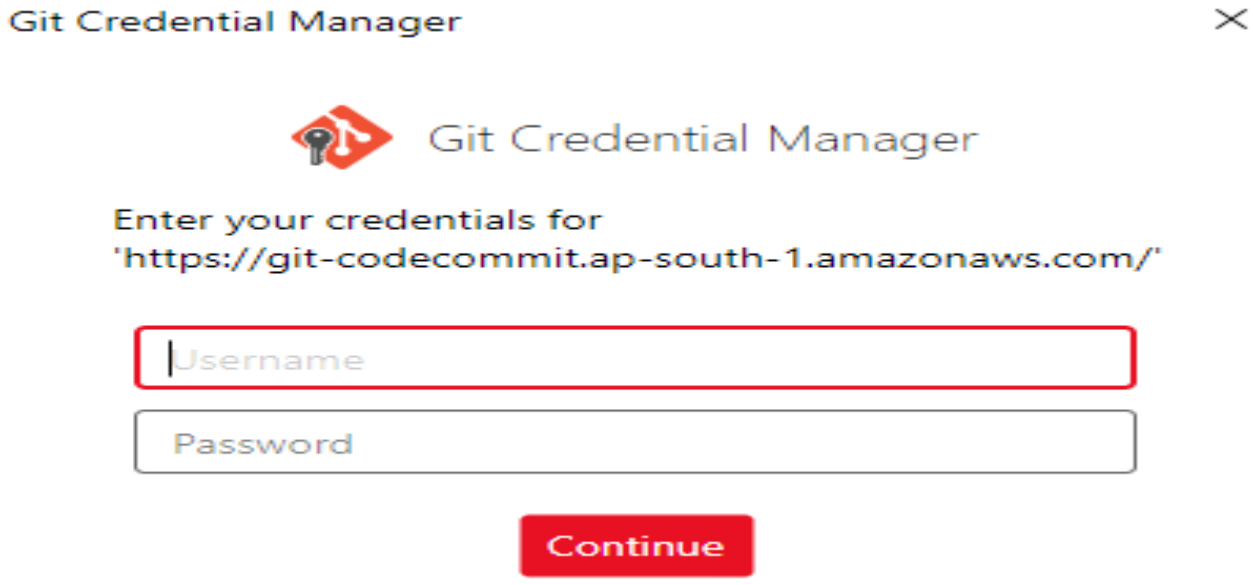
```
PS C:\Users\PULKIT\Downloads\my-angular-project> git remote add origin https://git-codecommit.eu-west-1.amazonaws.com/v1/repos/DemoAngularRepo
PS C:\Users\PULKIT\Downloads\my-angular-project> git push origin master
Enumerating objects: 69, done.
Counting objects: 100% (69/69), done.
Delta compression using up to 12 threads
Compressing objects: 100% (67/67), done.
Writing objects: 100% (69/69), 255.66 KiB | 8.25 MiB/s, done.
Total 69 (delta 22), reused 0 (delta 0), pack-reused 0
remote: Validating objects: 100%
To https://git-codecommit.eu-west-1.amazonaws.com/v1/repos/DemoAngularRepo
 * [new branch]      master -> master
PS C:\Users\PULKIT\Downloads\my-angular-project>
```

7. Now if you go to your repository and refresh the page you will see all your project data here.

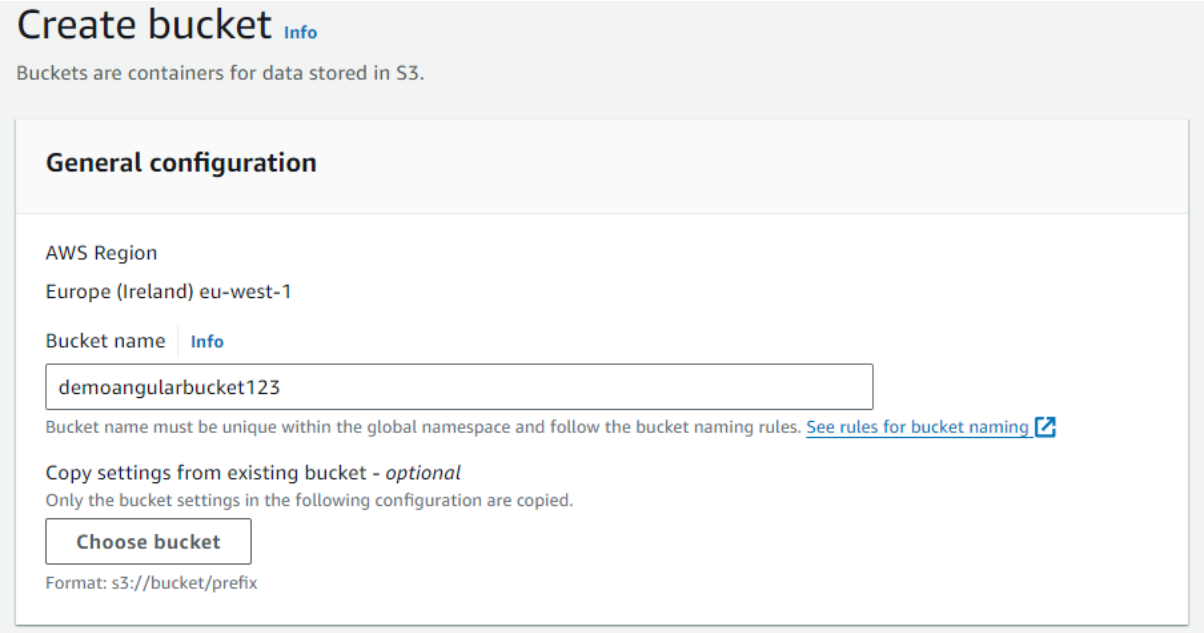


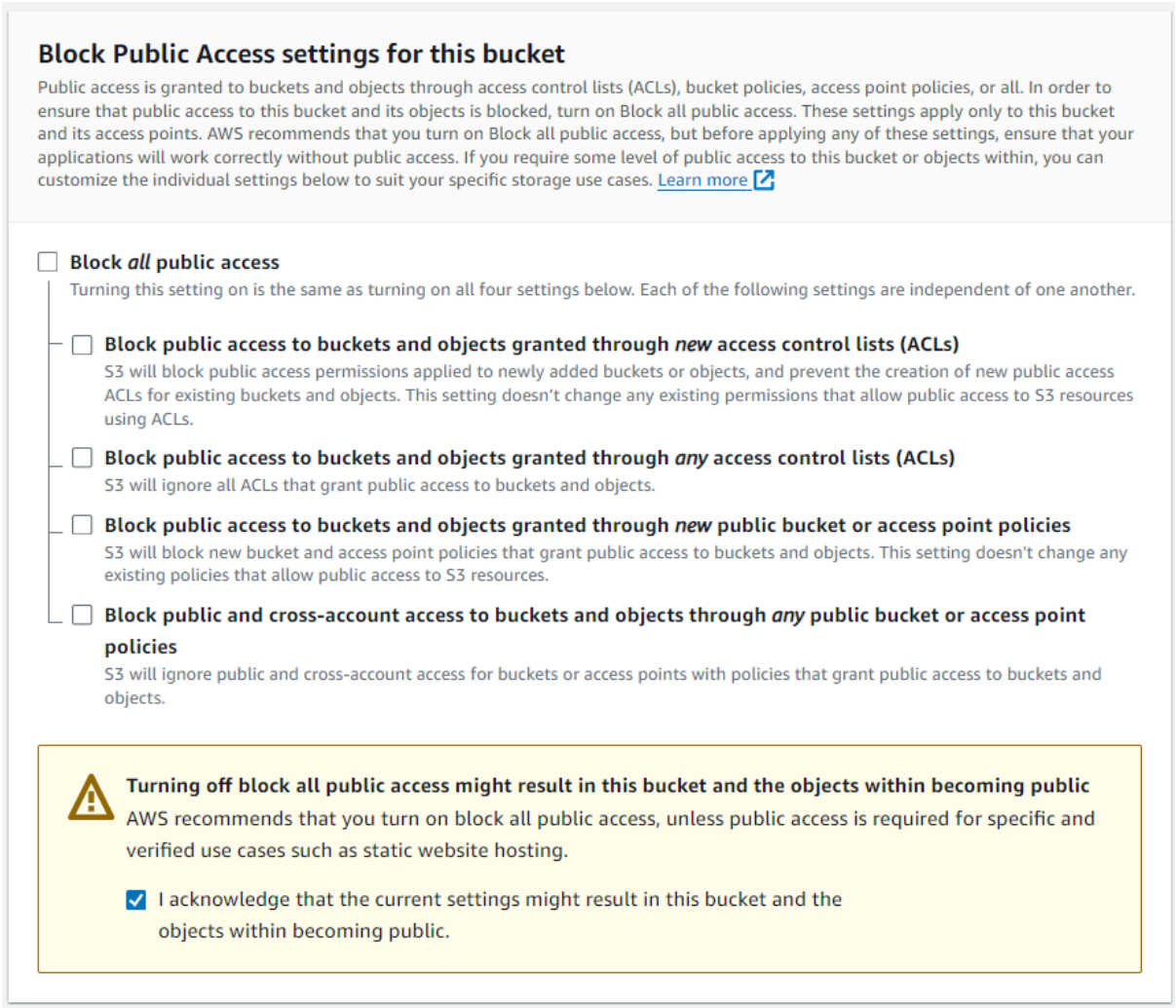
8. Also, if you get this type of error while cloning the project to the repository, you can visit the link below.

   https://docs.aws.amazon.com/codecommit/latest/userguide/setting-up-gc.html?icmpid=docs_acc_console_connect_np



9. Let’s create a new bucket for static website hosting as before. We will use it to deploy our sample website after building with CodeBuild. We are creating this bucket in the 'Europe (Ireland)' region.
10. So, basically, we are going to create a public bucket, click on create bucket give it a unique name, and turn off block public access, then create your bucket.

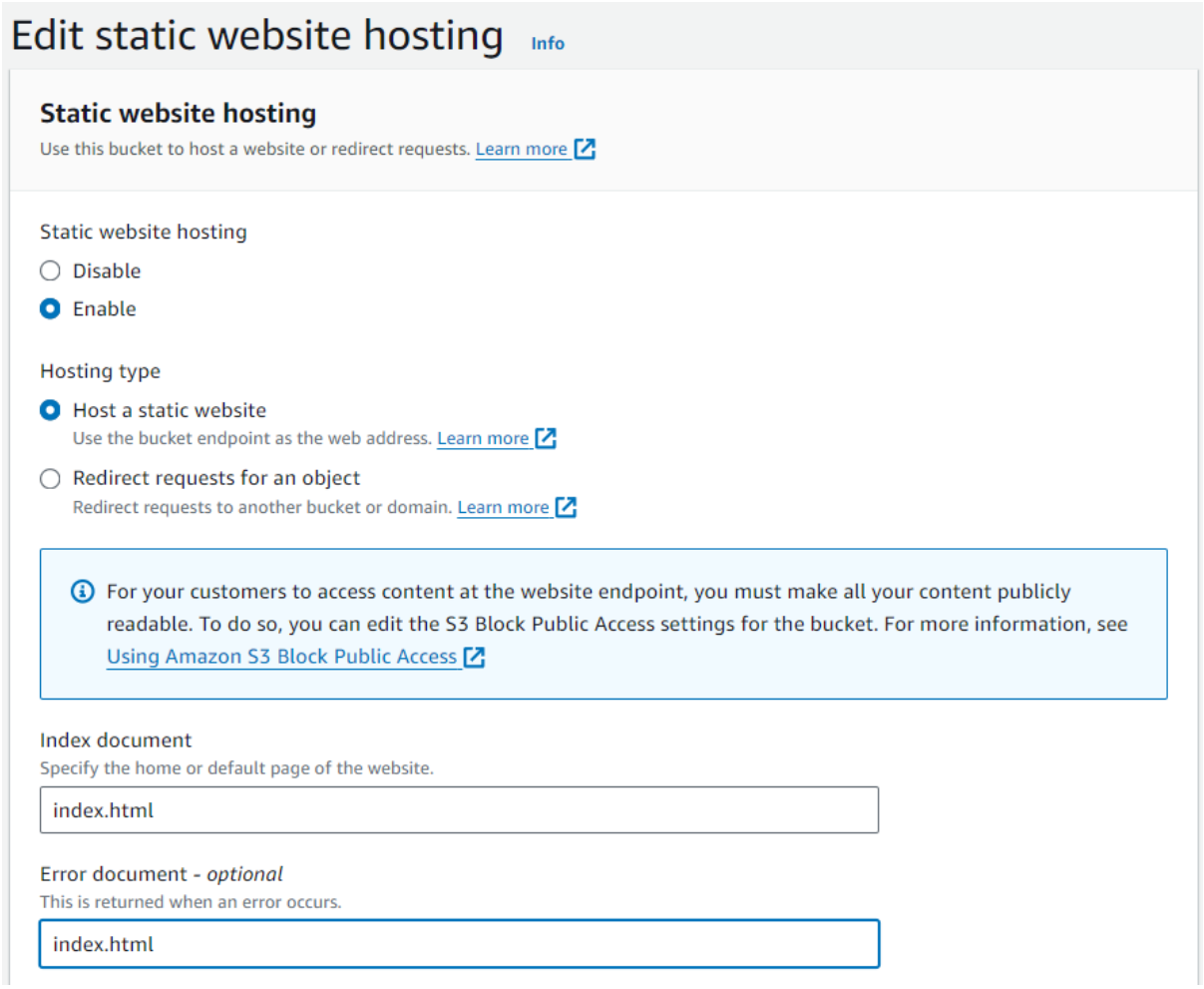


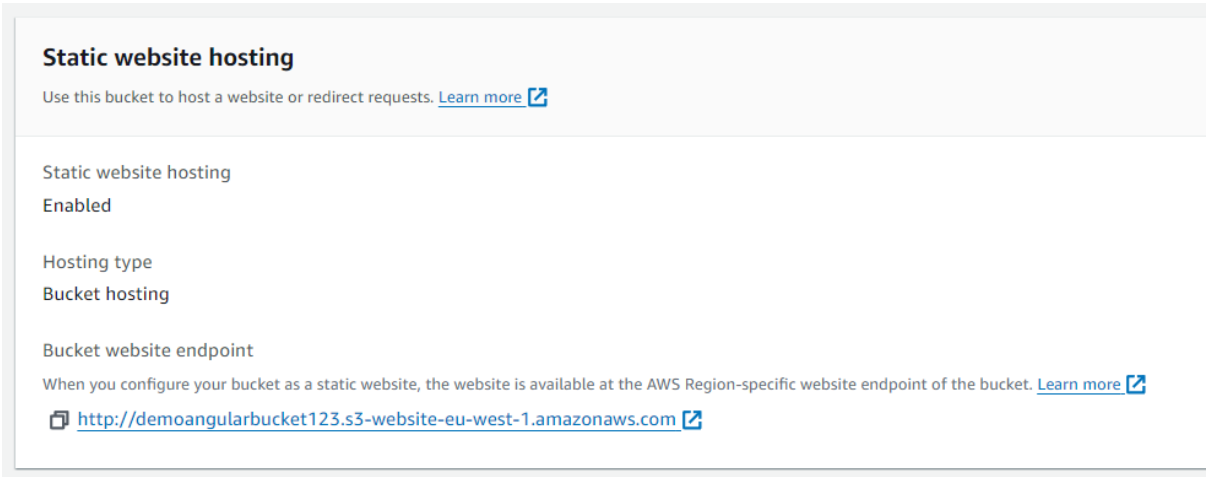


11. Once your bucket is created now should go to bucket policies and update its policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublicReadGetObject",
      "Effect": "Allow",
      "Principal": "*",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::YOUR-BUCKET-NAME/*"
      ]
    }
  ]
}
```

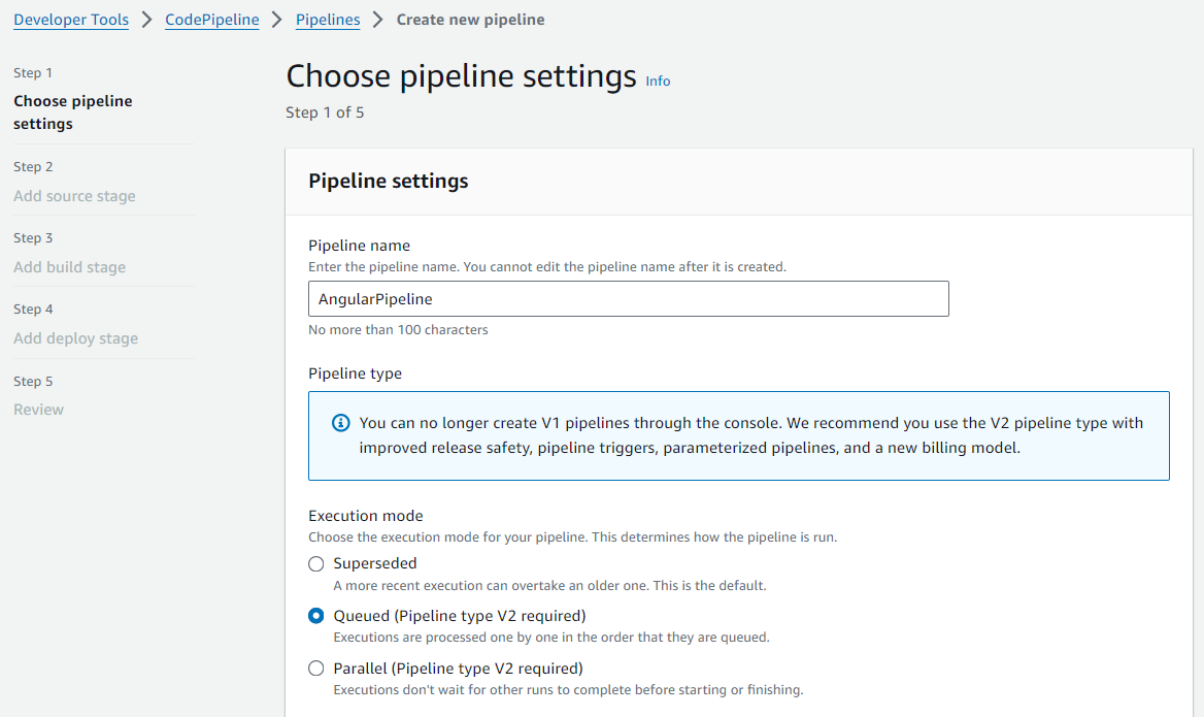
12. After that you need to enable static website hosting. Now when you host a single-page application, like Angular, on an S3 bucket, you need to configure both index and error document settings as 'index.html', the main page of your app.



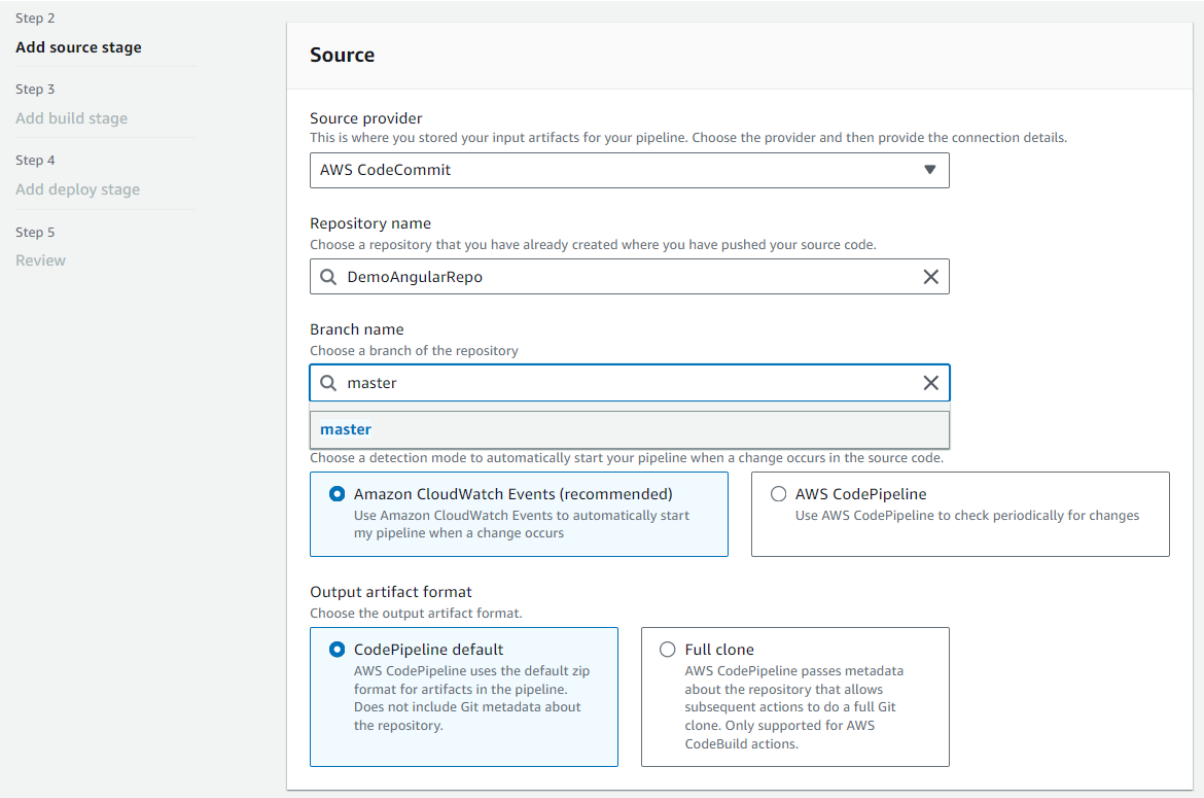


Step 2: Create a pipeline with the build stage

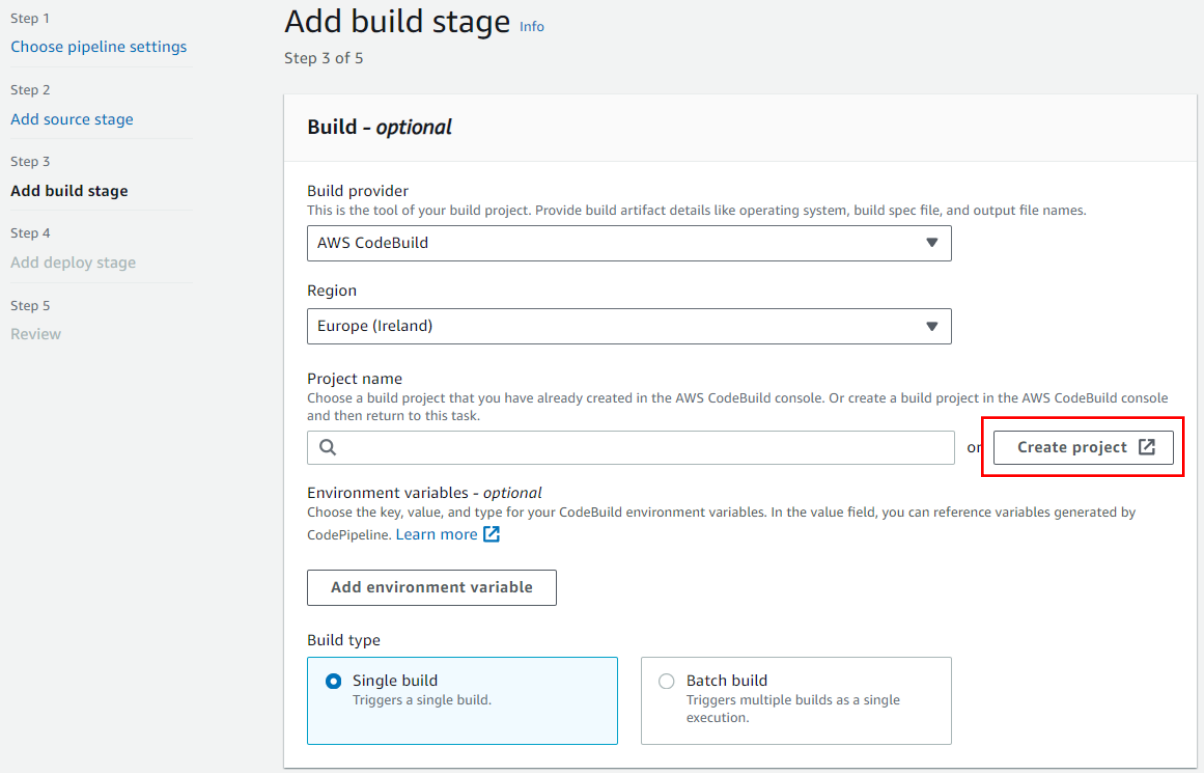
1. Now go to code pipeline and click on create pipeline. Now you need to give your pipeline a name and then move to the next step.



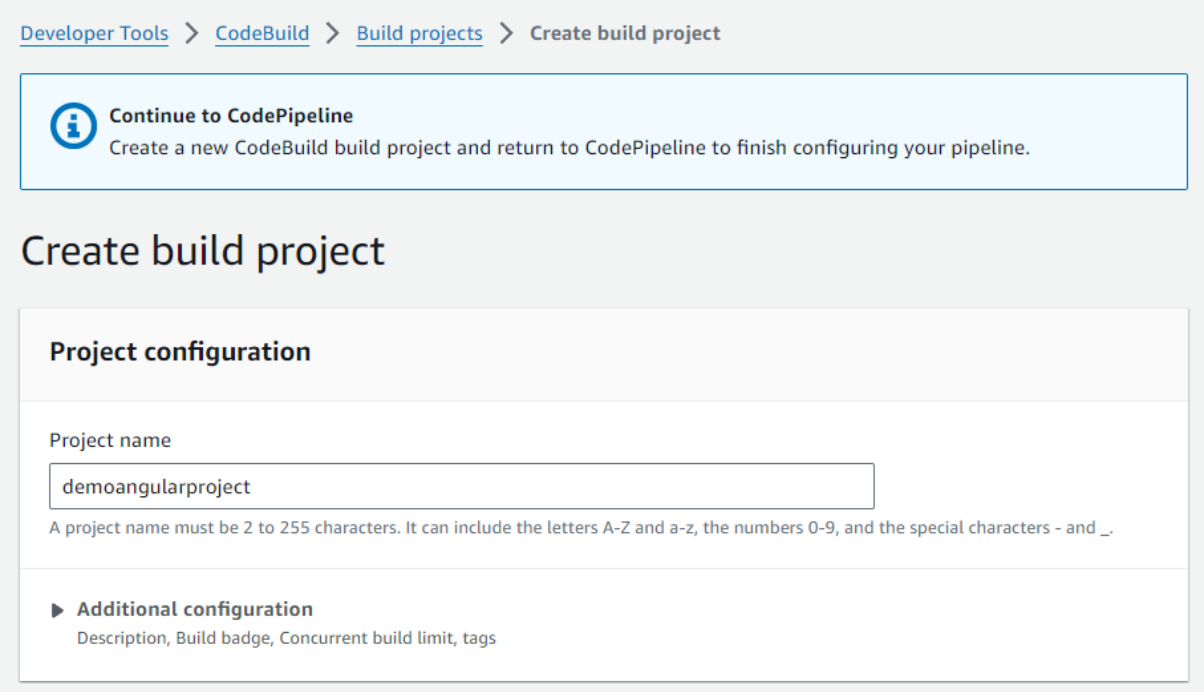
2. In step 2 we need to add the source provider, which is AWS code commit, choose it, and then select the repository name and choose your branch after that keep everything to default and move to the next step.



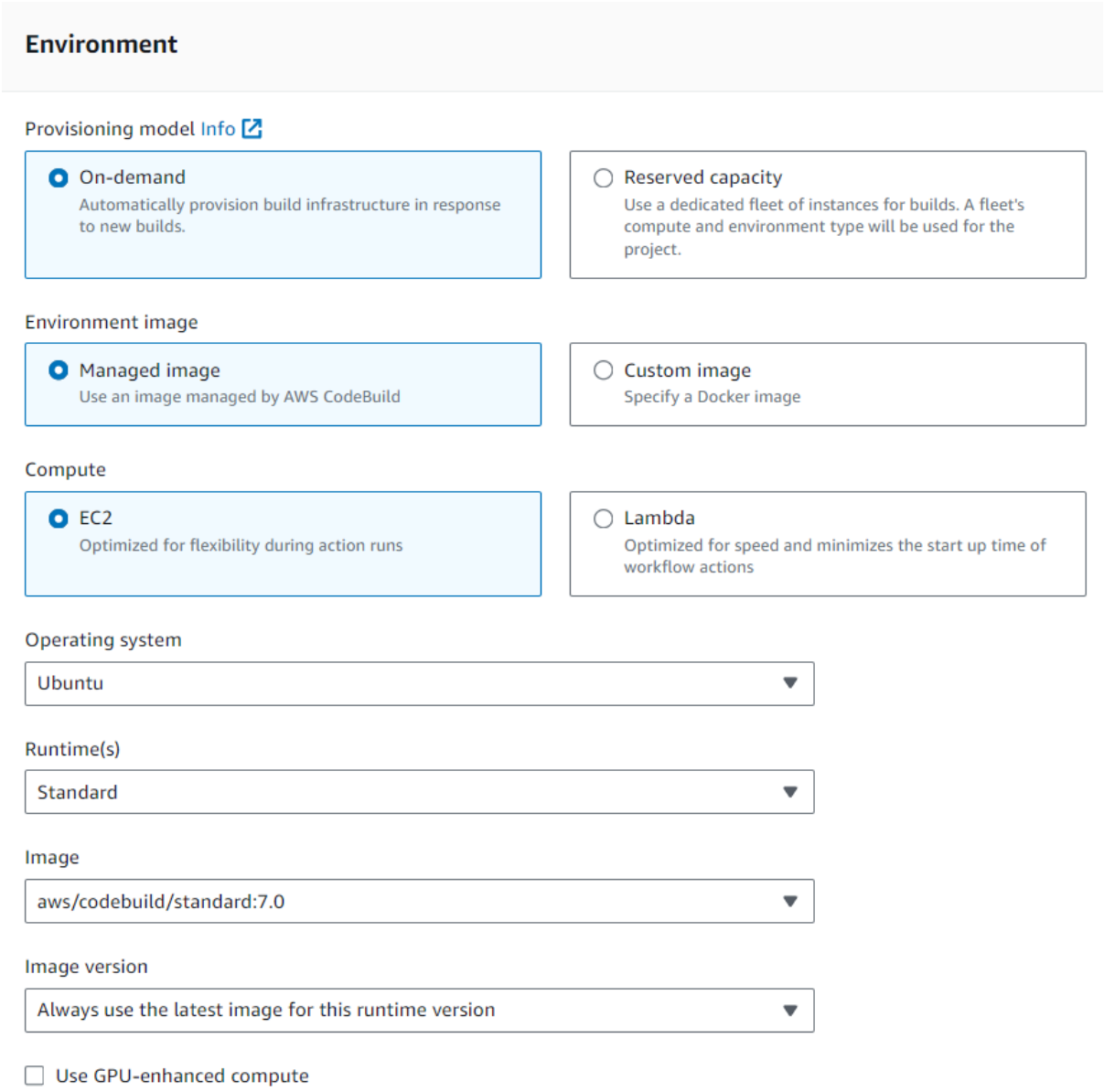
3. In step 3 you need to add a build stage, for that first you need to choose your build provider which is AWS Code build. Then choose your region, you need to create a new project for that click on Create a project and it will take you to a new tab.



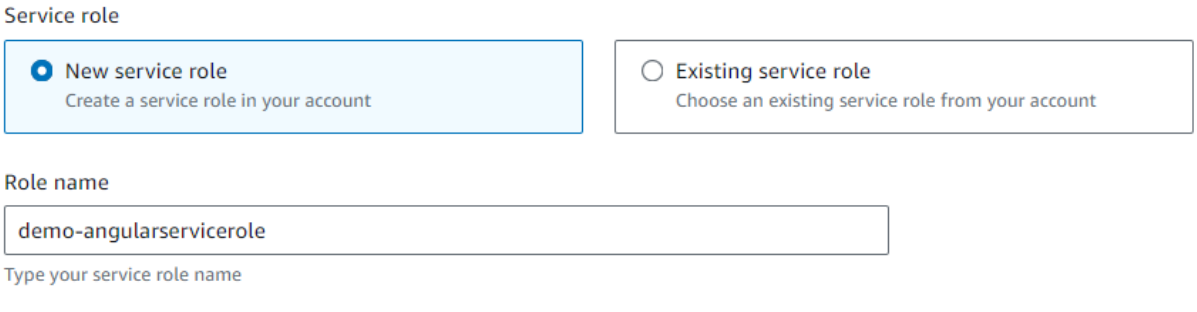
4. So, here first you need to give a project name and scroll down.

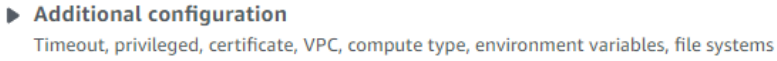


5. Now for the environment choose the same settings as shown below.

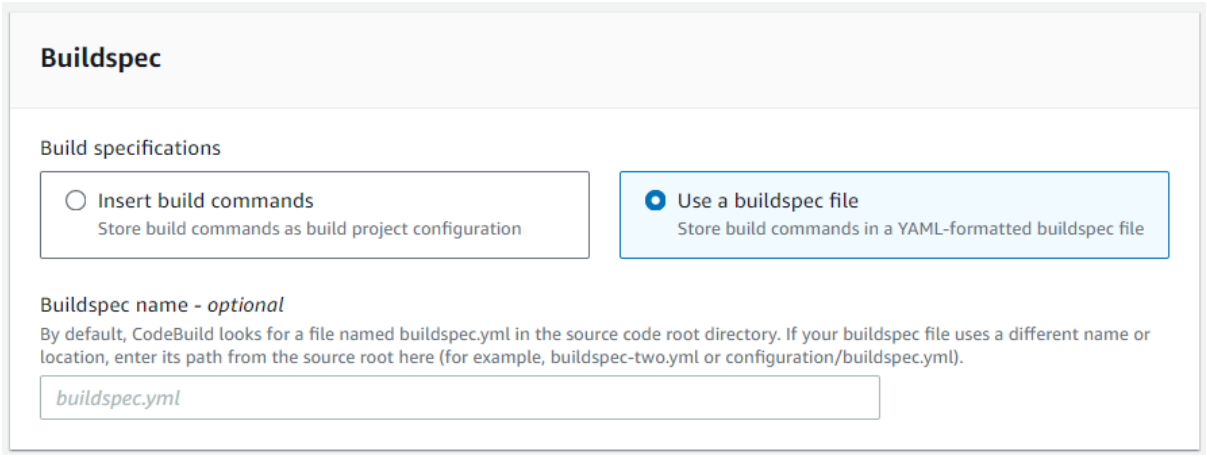


6. After that choose to create a new service role and give it a role name then scroll down and keep everything to default and click on continue to pipeline.

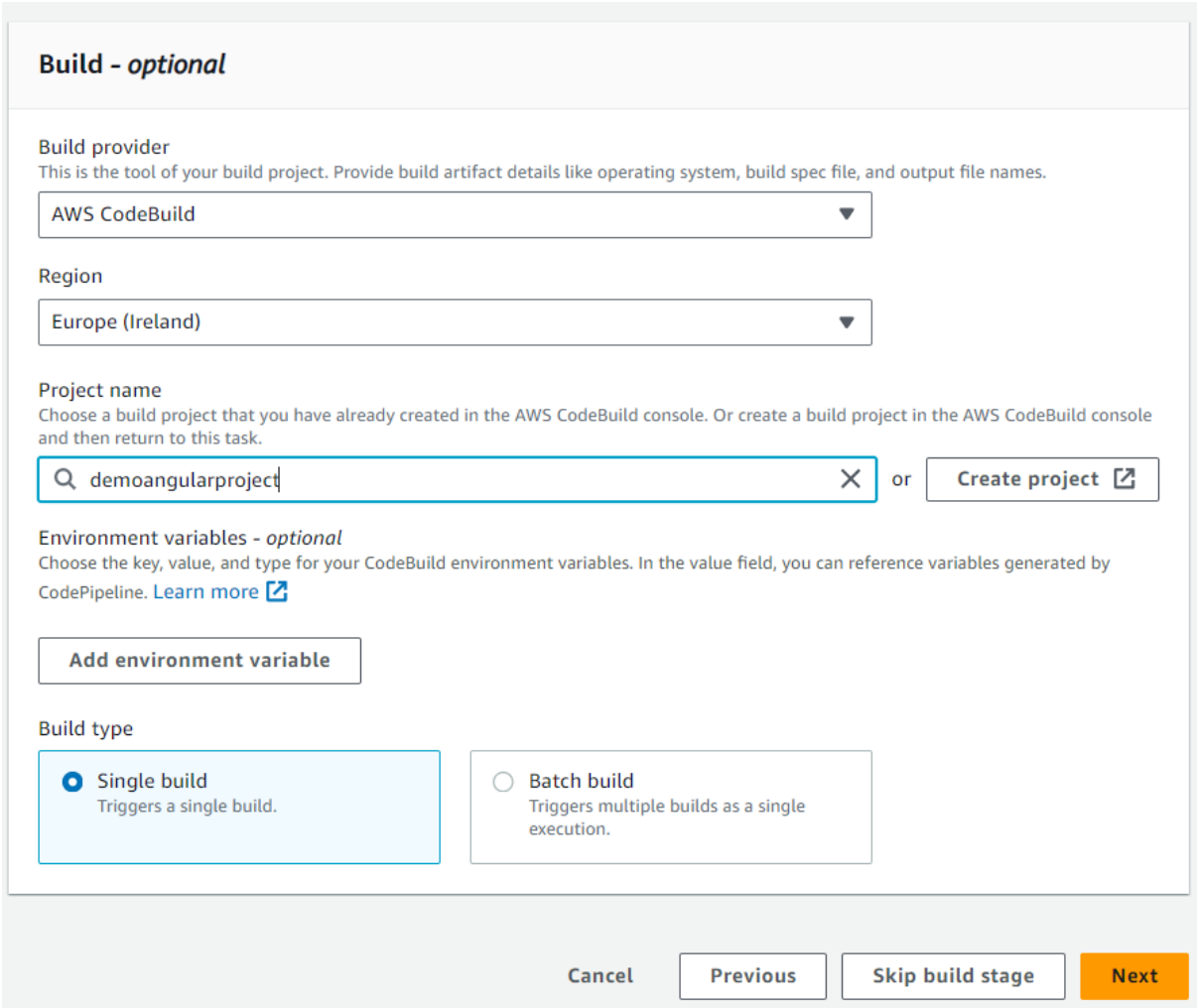




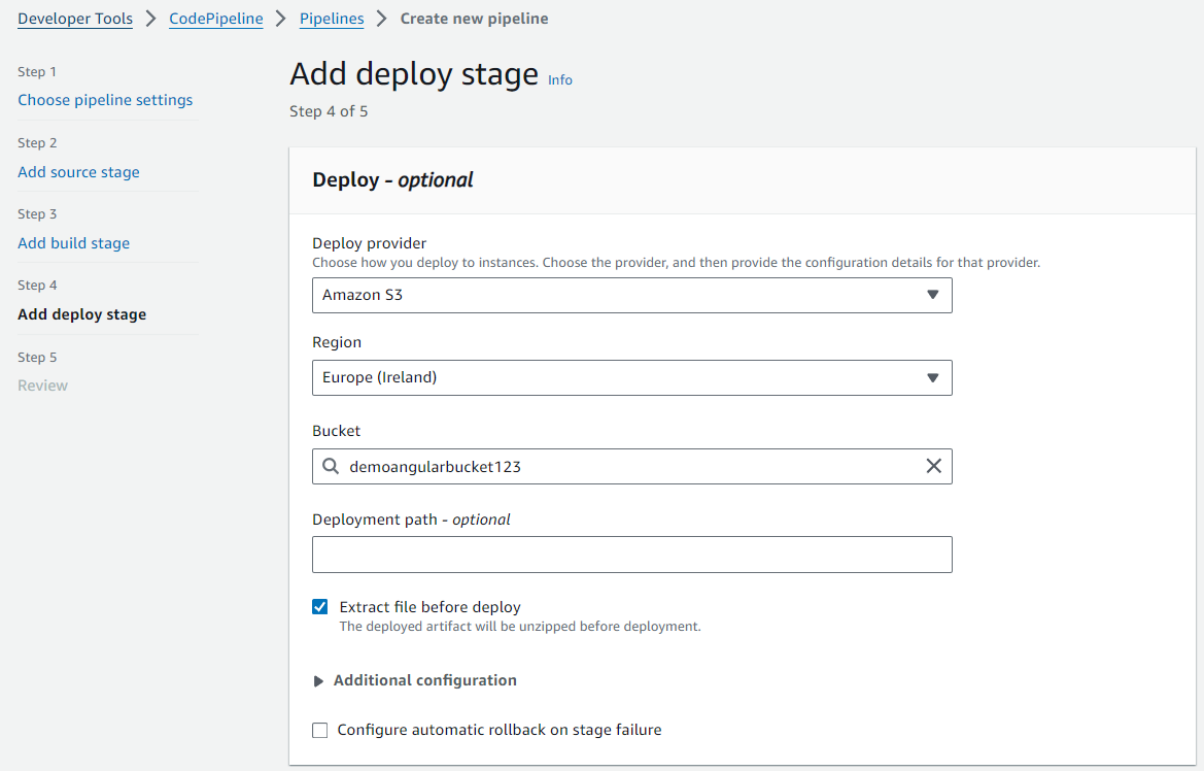
7. Also in the buildspec choose to use a buildspec file.



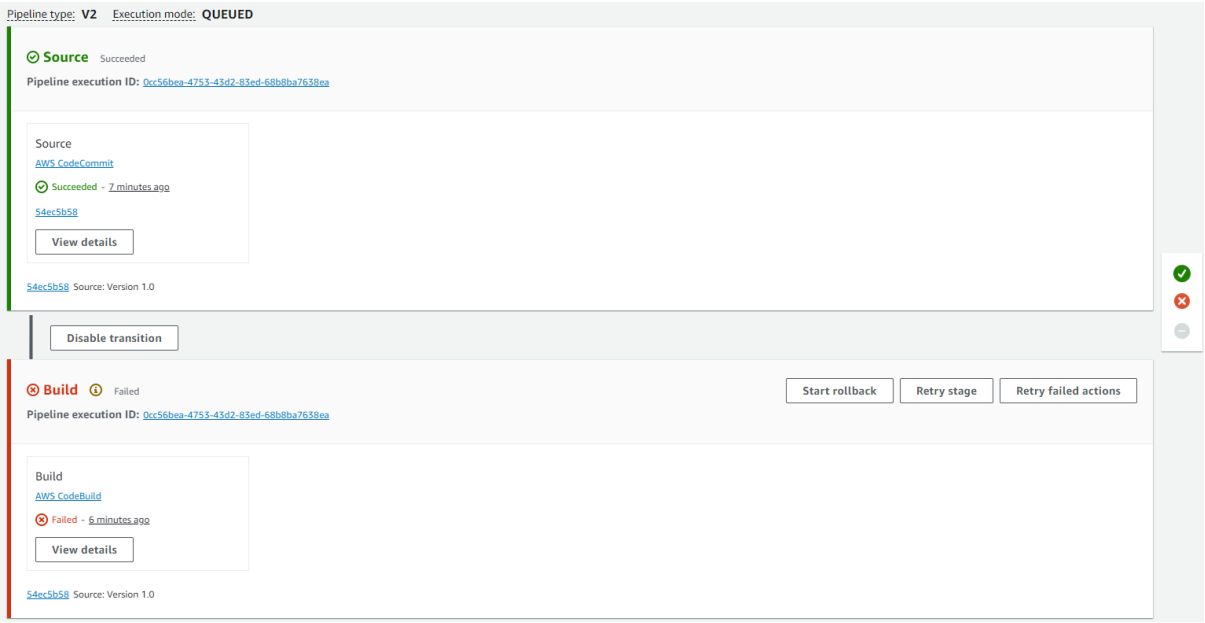
8. Below you can see that we can see our project name now click on next.



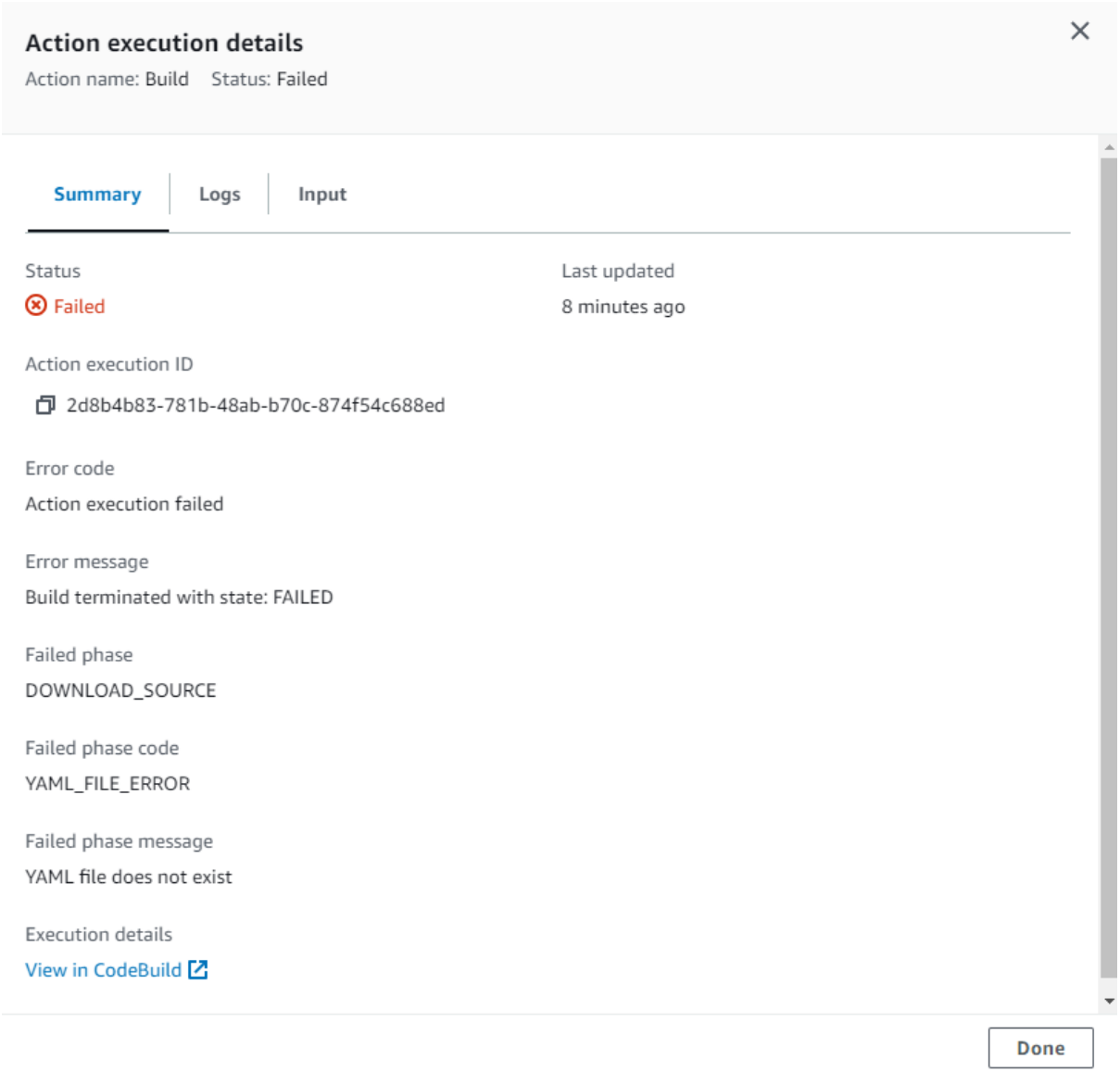
9. Now in step 4 you need to choose where you want to deploy your project, so we need to choose Amazon S3 and then choose our bucket then choose to extract file before deploy and move to review page then click to create your pipeline.



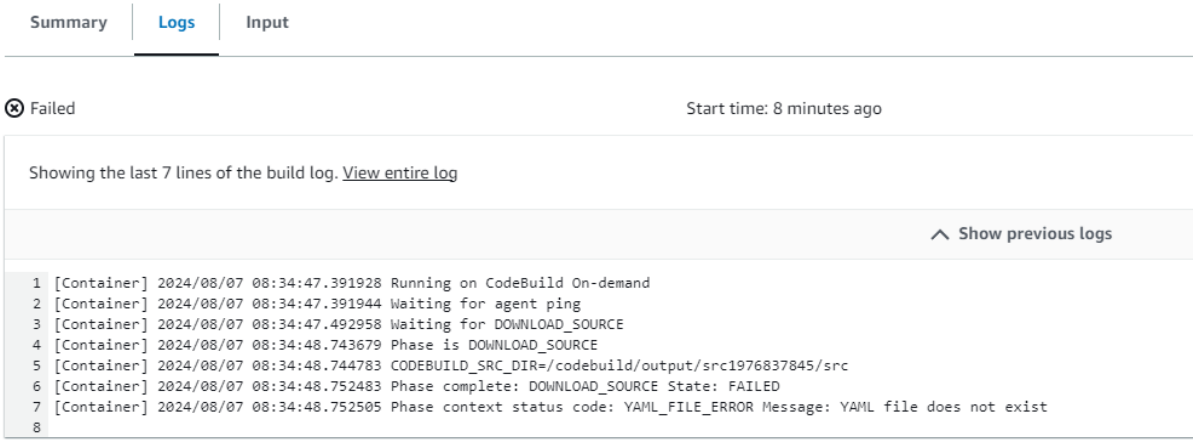
10. Below you can see that we created our pipeline and in the source stage it was successfully executed but during the build stage, it failed.



11. So, to check why it happened we need to click on the view details option available in the build stage. Here you can see that the failure was related to the YAML file.



12. Also, if you go to the logs section here you can see the context of failure.



13. This happens because CodeBuild looks for a buildspec file in our source code, which we have not yet provided. Hence, it doesn't know how to build the code.

😊 Creating Buildspec file

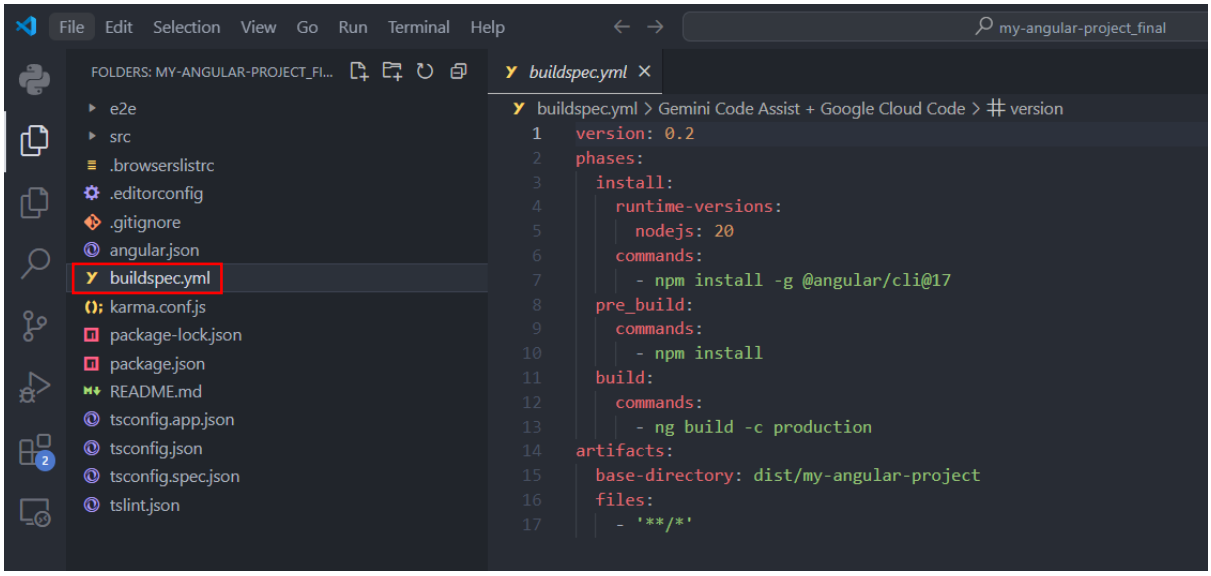
A build specification (build spec) file is a configuration file that defines how a software build process should be carried out. It contains instructions and parameters that guide the build system or tool in compiling and packaging the software application. Build spec files are used in various build automation tools and systems to streamline and standardize the build process.

Key Components of a Build Spec File

1. **Environment Setup**:
    - **Dependencies**: Specifies libraries, frameworks, and other dependencies needed for the build.
    - **Environment Variables**: Defines any required environment variables.
2. **Build Instructions**:
    - **Commands**: Specifies the sequence of commands to compile, test, and package the software.
    - **Scripts**: References scripts that need to be executed during the build process.
3. **Source Code**:
    - **Source Directories**: Defines where the source code is located.
    - **Exclusions**: Lists files or directories to be excluded from the build.
4. **Build Artifacts**:
    - **Output Directories**: Specifies where the build outputs (artifacts) should be stored.
    - **Artifact Management**: Defines how artifacts are managed, such as versioning and storage.
5. **Build Triggers**:
    - **Conditions**: Specifies conditions under which the build should be triggered (e.g., code commits, schedule).

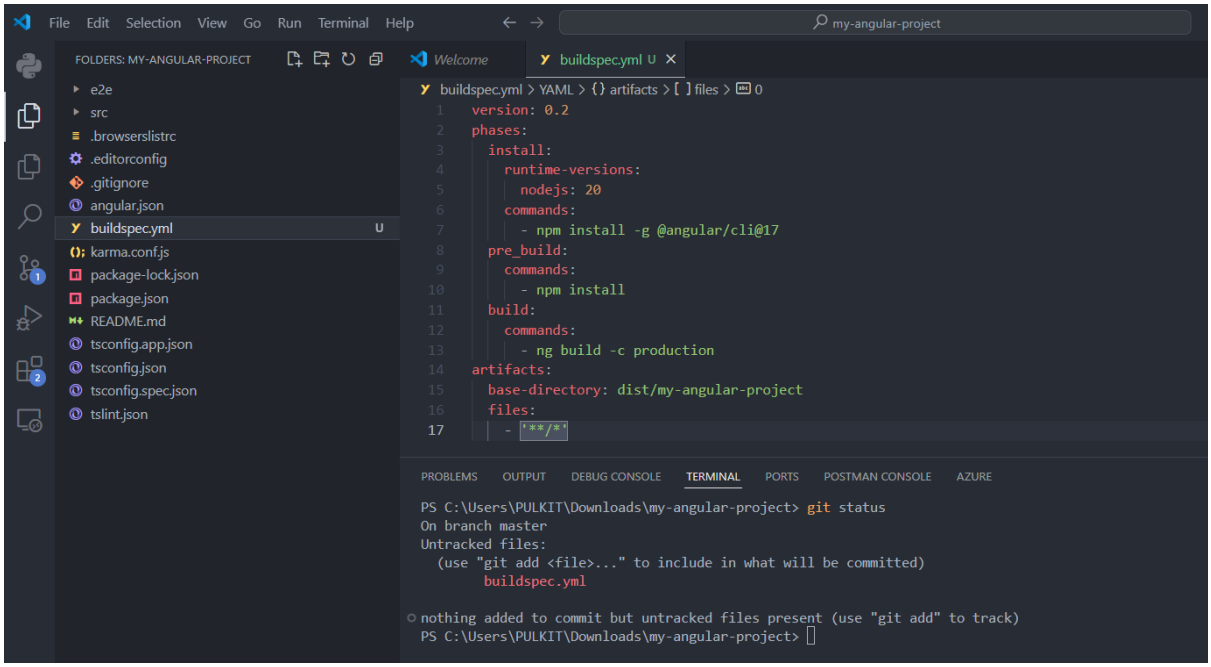
😄 To begin with the Lab:

1. First you need to download a zipped folder from our GitHub named my-angular-project_final. Then extract it and open it in VS Code.
2. So, in this project you will find a YAML file named buildspec.yaml. So, in our previous lab, our build stage failed because we didn't have this file in our source code.

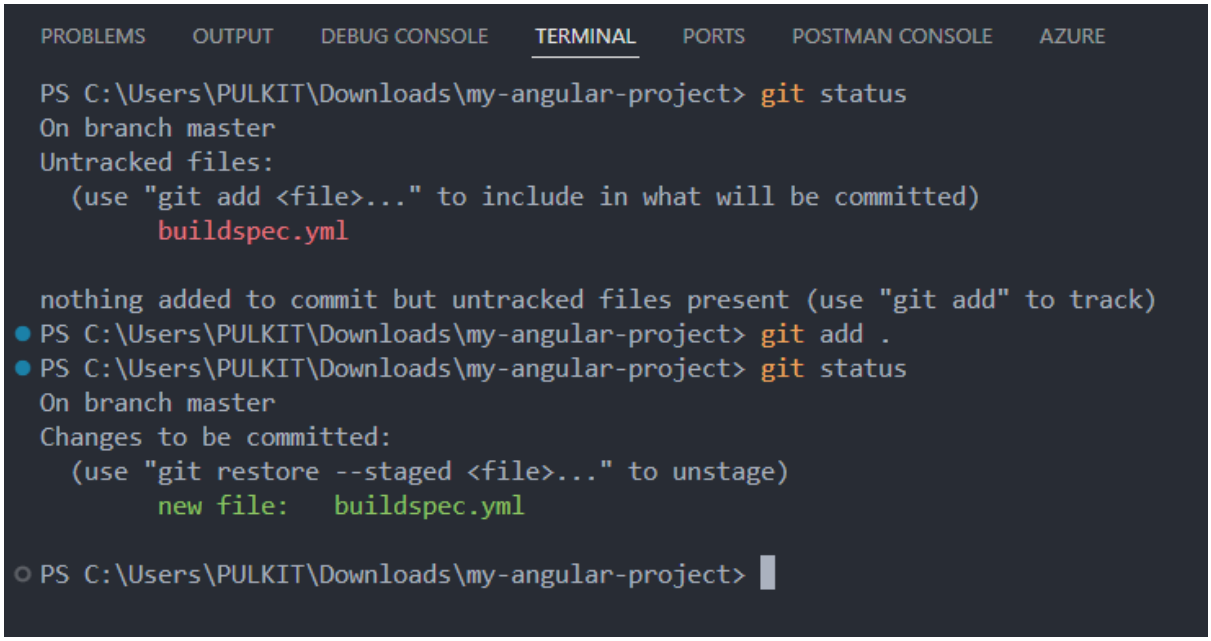


3. Now you need to copy the content from this file and create a similar file in the previous project which we had uploaded to our repository.
4. Once you have done that then you need to open the terminal and run the git status command to check what needs to be committed. After that run the git add command.

git status
git add .

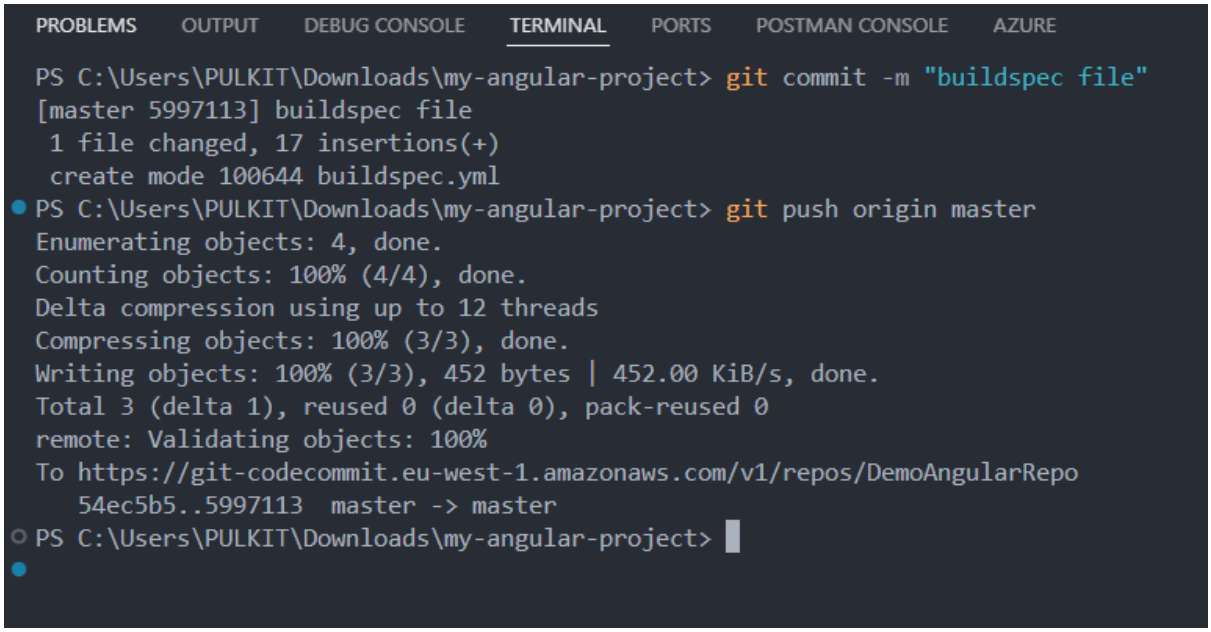


5. Below you can see that our new file has been staged now we need to commit our changes and push the updates to our repository.

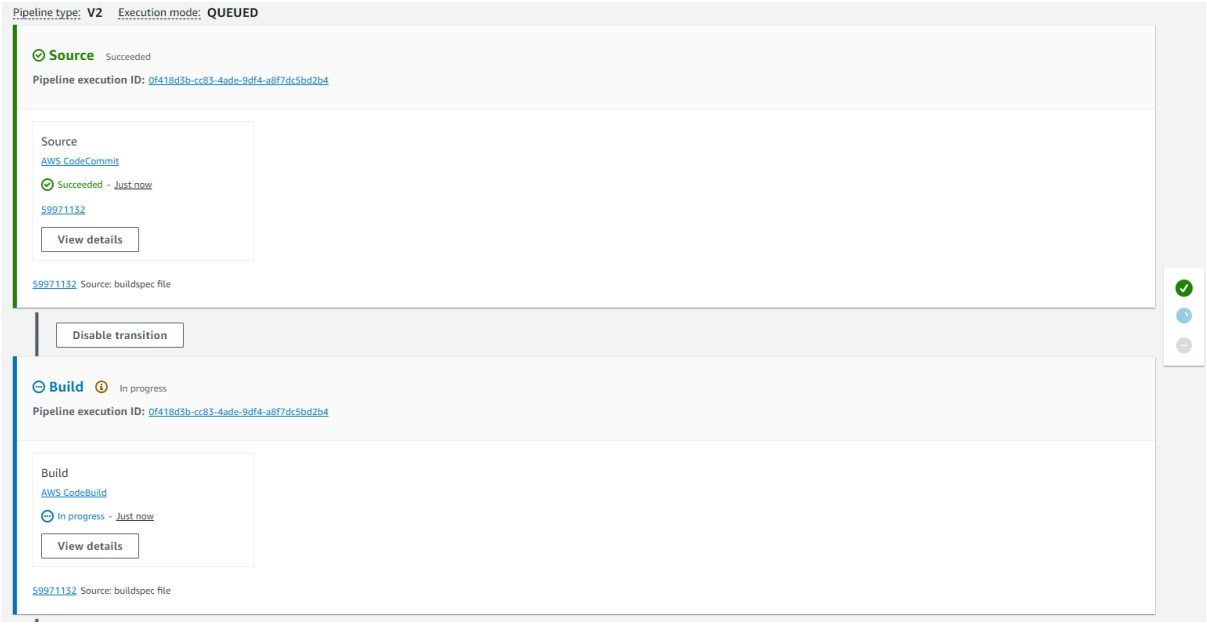


6. For that you need to run the below command.

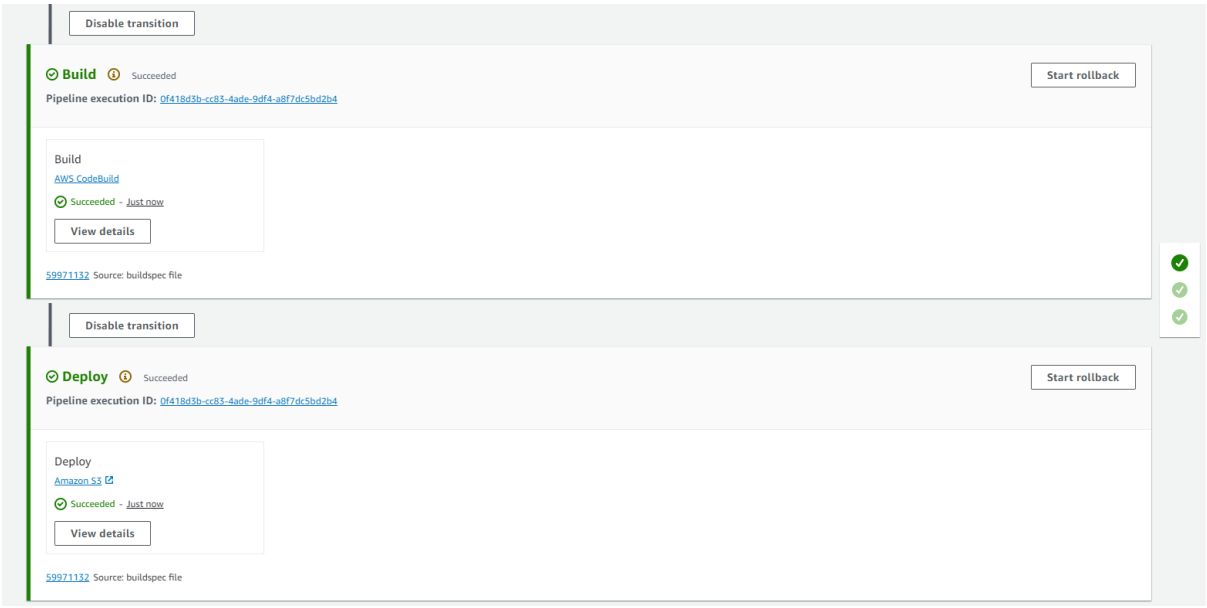
   git commit -m “buildspec file”
   git push origin master.
7. Below you can see that we have committed our file and pushed it onto our repository.



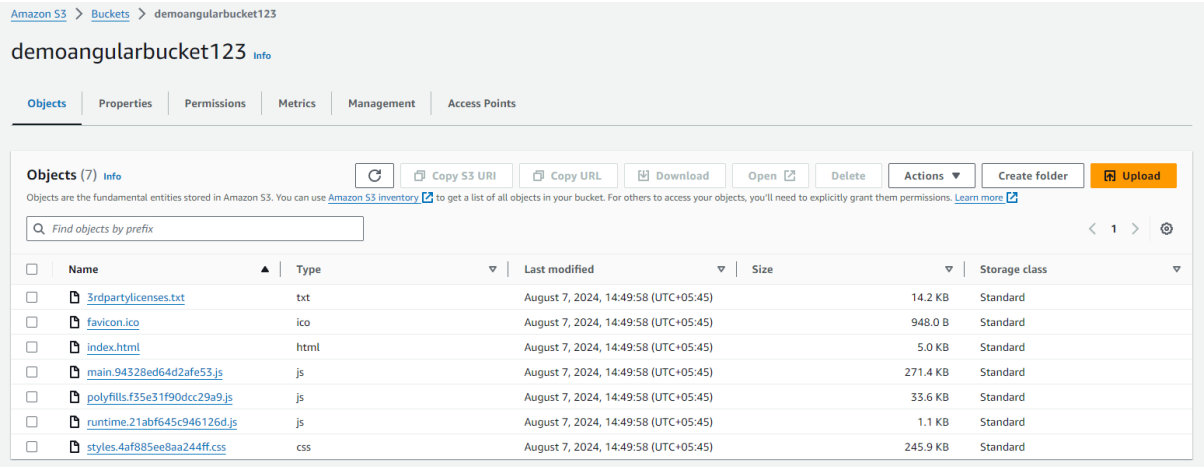
8. Now if you go back to the code pipeline console you will see that our pipeline has started running and you can see that our build stage is in progress. You can also see the same commit message in the build stage. This build stage will take some time.



9. And below you can see that our pipeline has been executed successfully. All three stages has been executed. Source, build and deploy.



10. Now if you go to the S3 bucket you will see that the objects have been added here from the repository. Now go to properties scroll down to the bottom and open your static website link.



11. Below you can see that our static website is working perfectly.