😊 Adding a test action with Code Build

In this exercise, we intentionally introduced a mistake in our code to see how it affects our pipeline and then added automated tests to prevent such issues. Initially, we changed a calculator operator to cause a deliberate error and verified it by pushing the code, which updated the website incorrectly. This highlighted the need for automated testing.

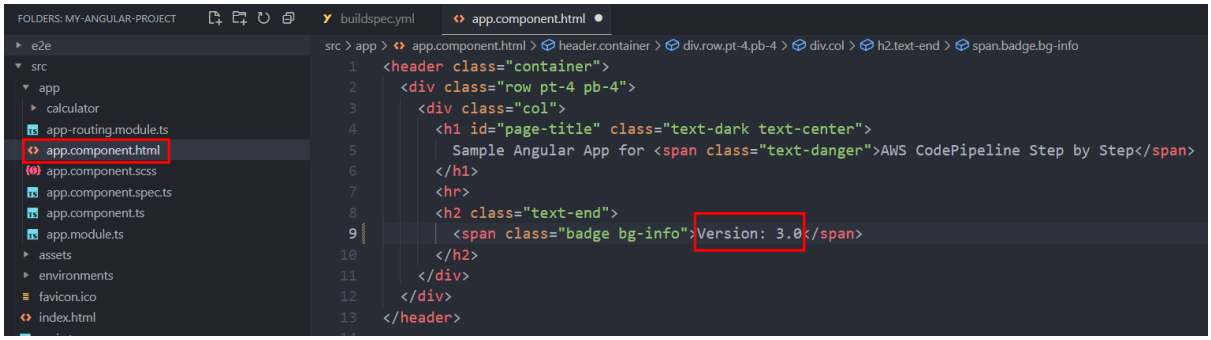
We then created a unit-test-buildspec.yml file to run unit tests using CodeBuild before the build stage. By editing the pipeline to include this new test action, we ensured that tests run first and the build proceeds only if tests pass. After reintroducing the mistake and pushing the code, the pipeline failed at the test stage, preventing the faulty build from deploying. We then corrected the error, re-ran the pipeline, and confirmed everything worked correctly.

End Goal

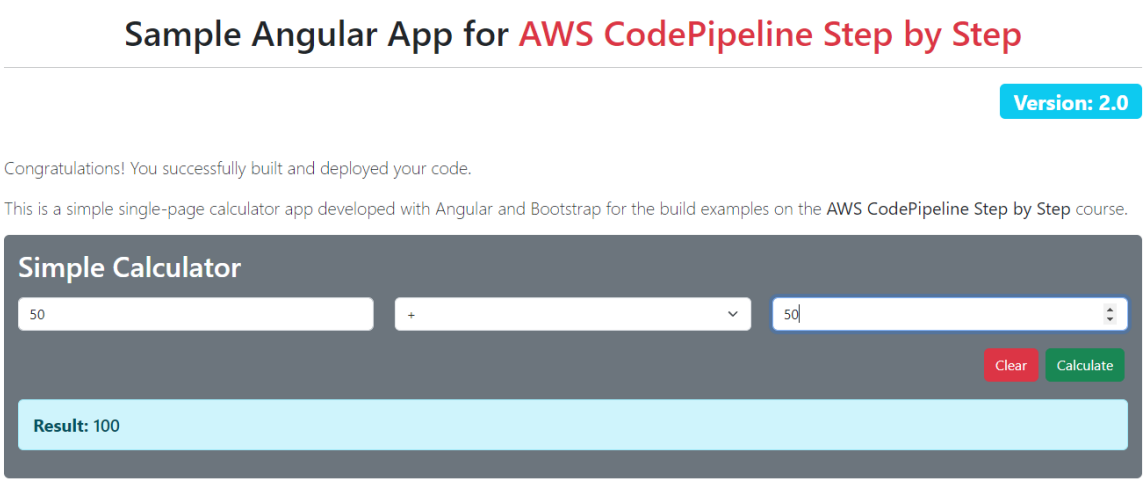
The aim was to demonstrate the importance of automated testing in a CI/CD pipeline to catch and fix errors before deployment, ensuring reliable and error-free software delivery.

😄 To begin with the Lab:

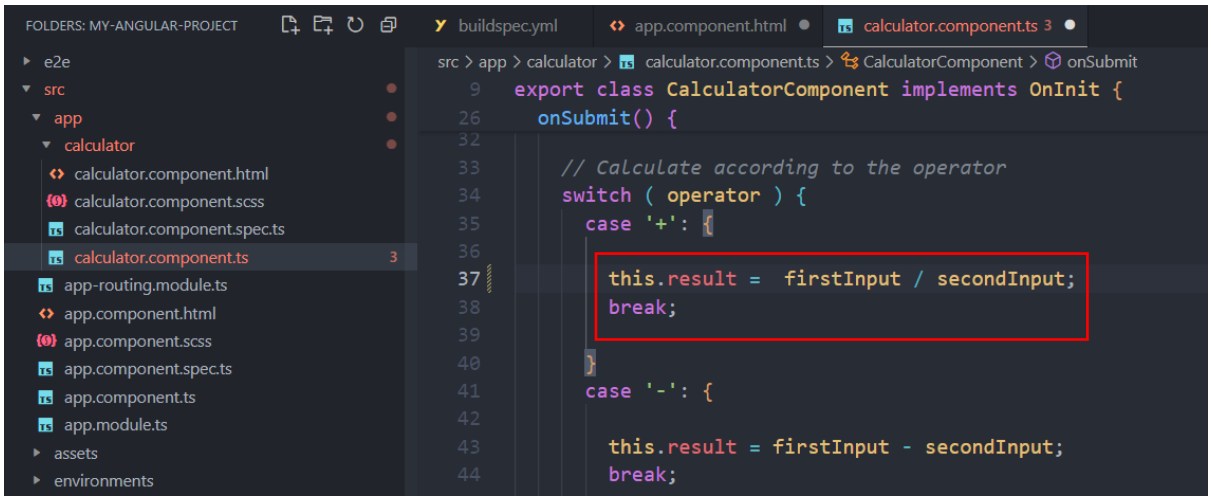
1. In this lab first we are going to change the version name of our website. For that expand the app folder and you will see a file named app.component.html. from there you need to change the version of your website.



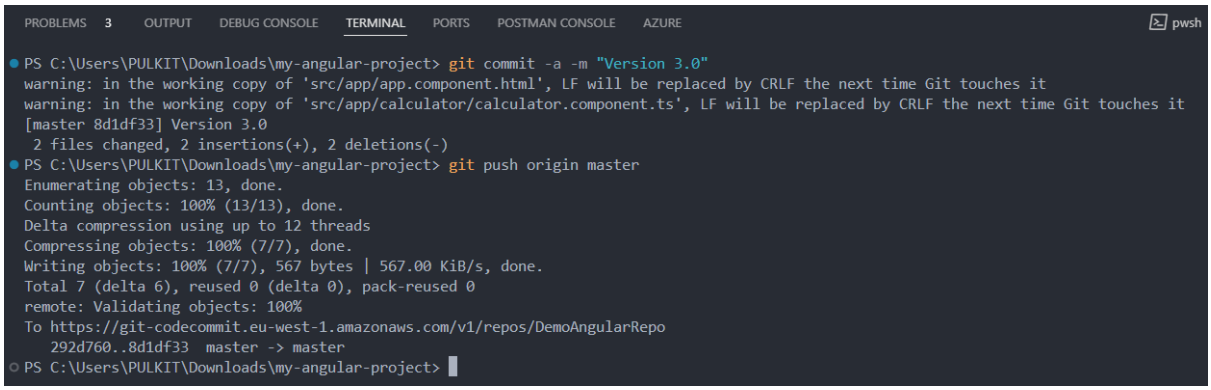
2. Now before doing anything else let's go to our website and check how it is working. So, below you can see that our website is currently working fine, there are no issues with it.



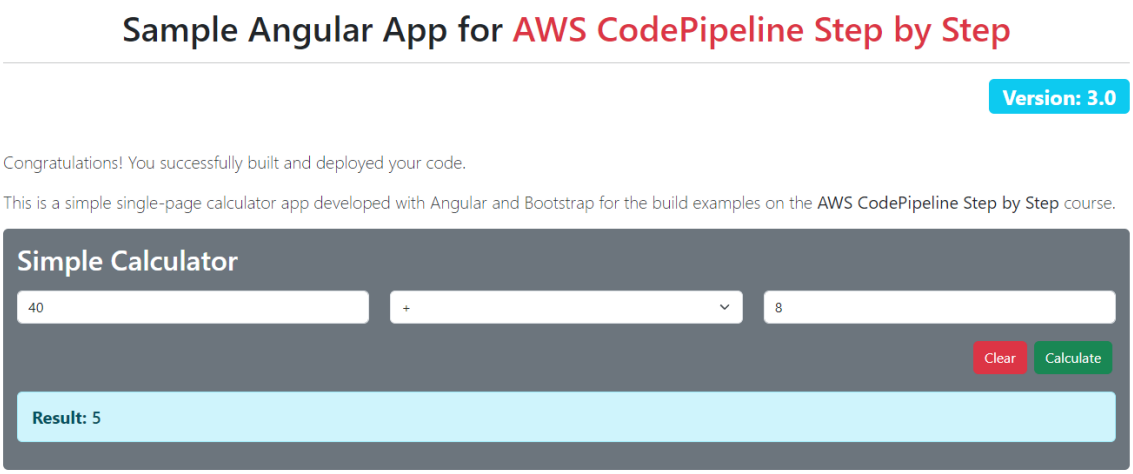
3. Now come back to VS Code and expand the calculator folder now let's say a mistake has been made by us that is we forgot to put the right operator in the highlighted part. You see here instead of the divide symbol, it should be plus but intentionally for this lab I'd changed it to division, you should also do this. Then save everything.
4. Now let's push our code to the repository. Now, if you push this change the plus operator will act like a division, right? Well, is there anything on our pipeline to avoid this?



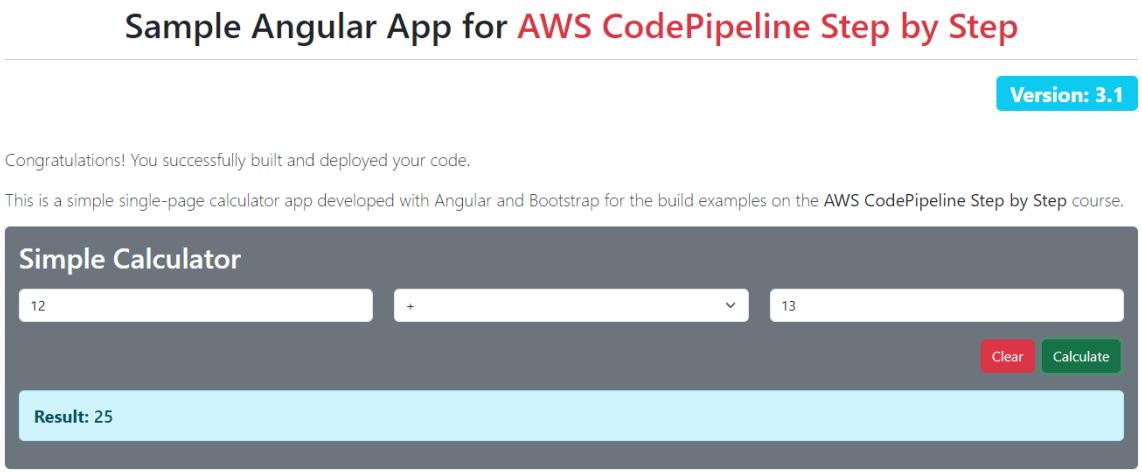
5. Below you can see that we committed to our changes and pushed it all to the repository. And as expected our pipeline has been started.



6. Once your pipeline has been executed successfully then you need to go to the website refresh it check the version and try to do the addition on the calculator.
7. Below you can see that our version has been changed which means that the pipeline was executed properly, also when we try to do addition it performed division.



8. We just broke the application. The bad thing is if you have many users, they notice these types of mistakes before you do. Then it will have a bad impact on you.
9. But first we will fix our application so go back to VS Code and change the operator to its original state which is addition. Then push the code to the repository, wait for the pipeline to run, and check if our calculator is working properly or not.
10. Also, you can change the version of your website. Below you can see that we have changed everything back to normal.

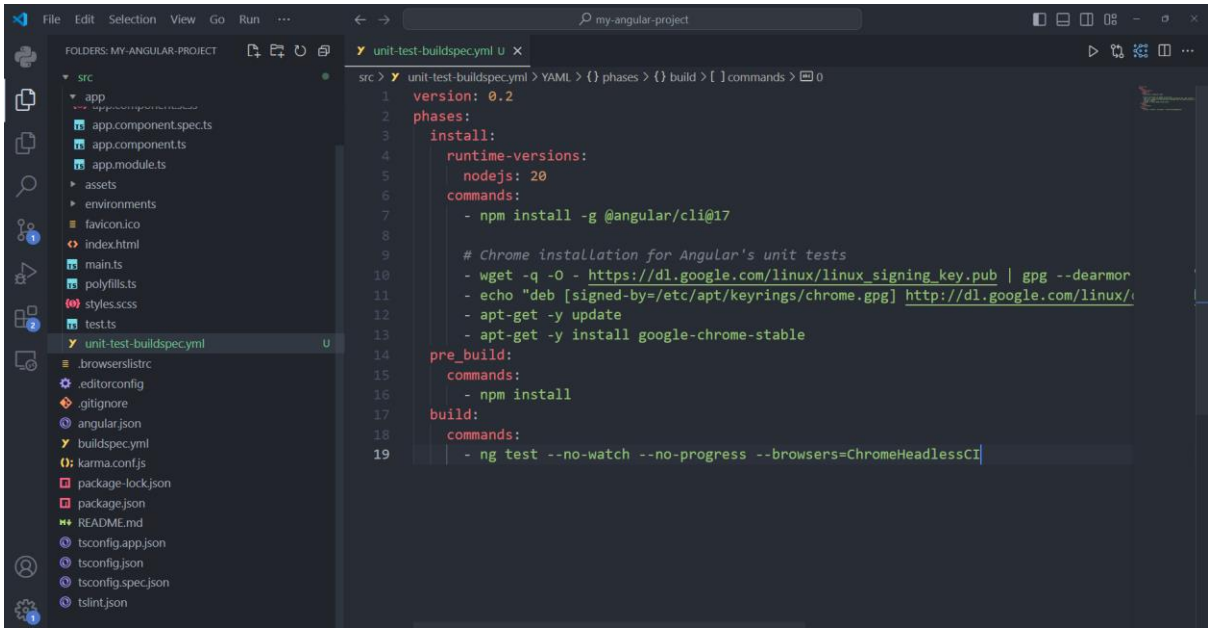


11. Hence, we need the tools to avoid these types of failures before the deployment. So, how can we achieve that? Well, we use automated tests for this purpose. We can have unit tests in our code and run them before the build command. So, we can catch them at the building stage. In addition, we can also have a staging environment and run our integration tests there before deploying to production.

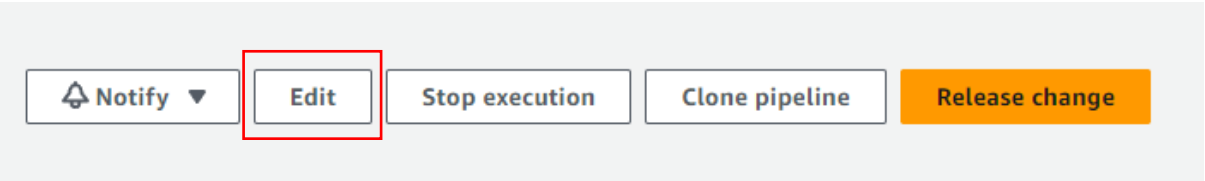
😁 Adding a test action to the code build

1. Now open VS Code and create a new YAML file named unit-test-buildspec.yml, we are creating this buildspec file so that it can perform the checking. You can use the code below and paste it into the file, just remember to check the indentation and typo errors.

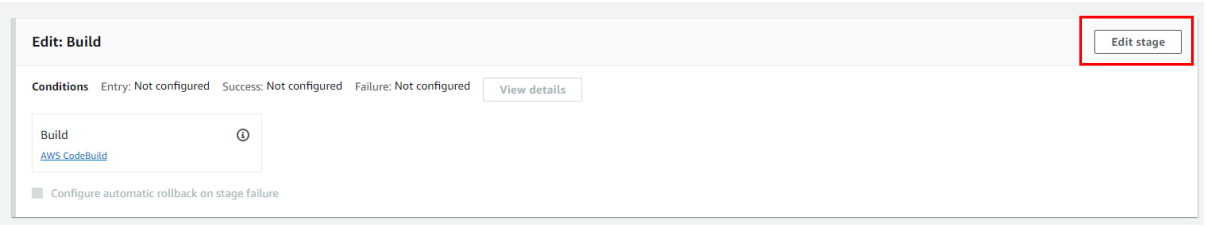
```
version: 0.2
phases:
  install:
    runtime-versions:
      nodejs: 20
    commands:
      - npm install -g @angular/cli@17

      # Chrome installation for Angular's unit tests
      - wget -q -O - https://dl.google.com/linux/linux_signing_key.pub | gpg --dearmor | tee /etc/apt/keyrings/chrome.gpg
      - echo "deb [signed-by=/etc/apt/keyrings/chrome.gpg] http://dl.google.com/linux/chrome/deb/ stable main" >> /etc/apt/sources.list.d/google-chrome.list
      - apt-get -y update
      - apt-get -y install google-chrome-stable
  pre_build:
    commands:
      - npm install
  build:
    commands:
      - ng test --no-watch --no-progress --browsers=ChromeHeadlessCI
```



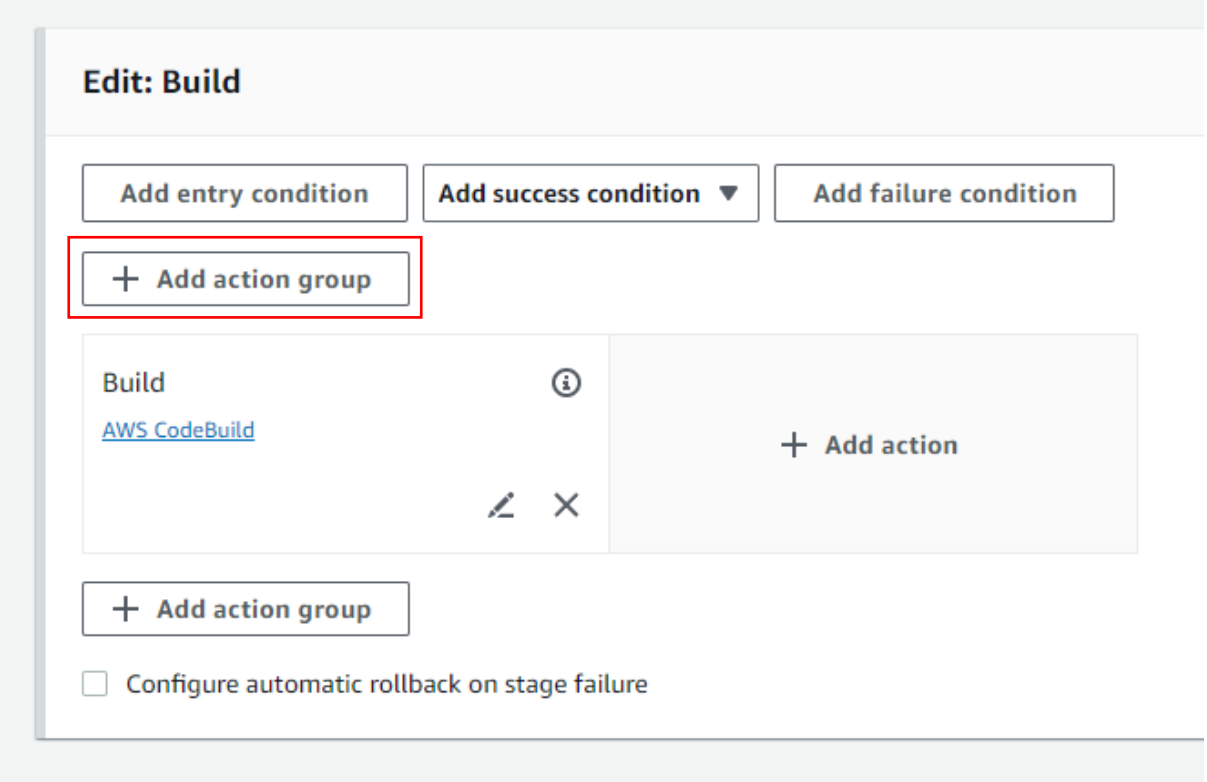
2. Now go to your pipeline and we will create a sequential action along with our stage for that we need to click on edit.



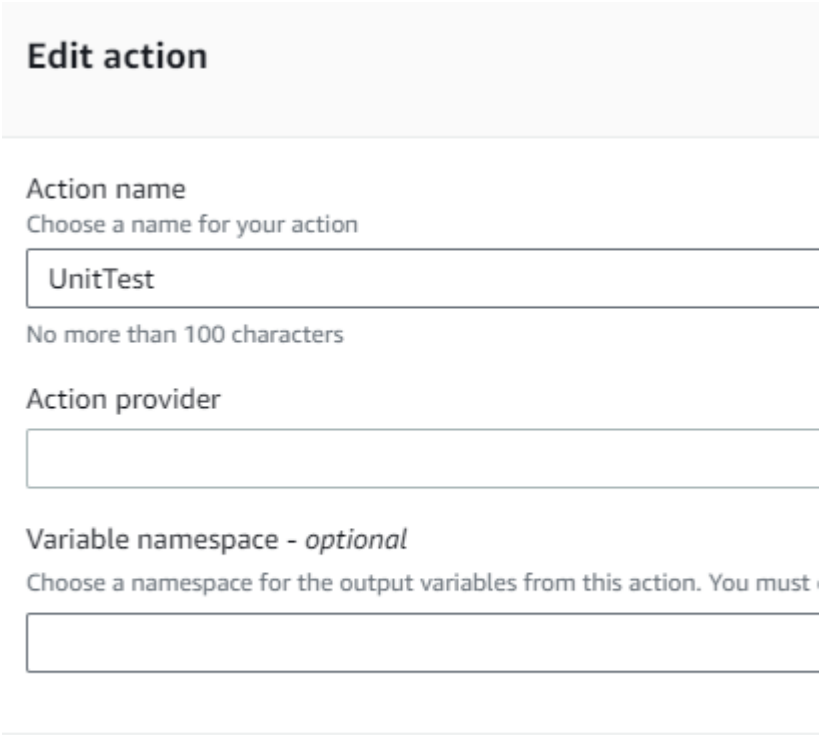
3. Now we will add a new action to our build stage. So, click on the edit stage button.



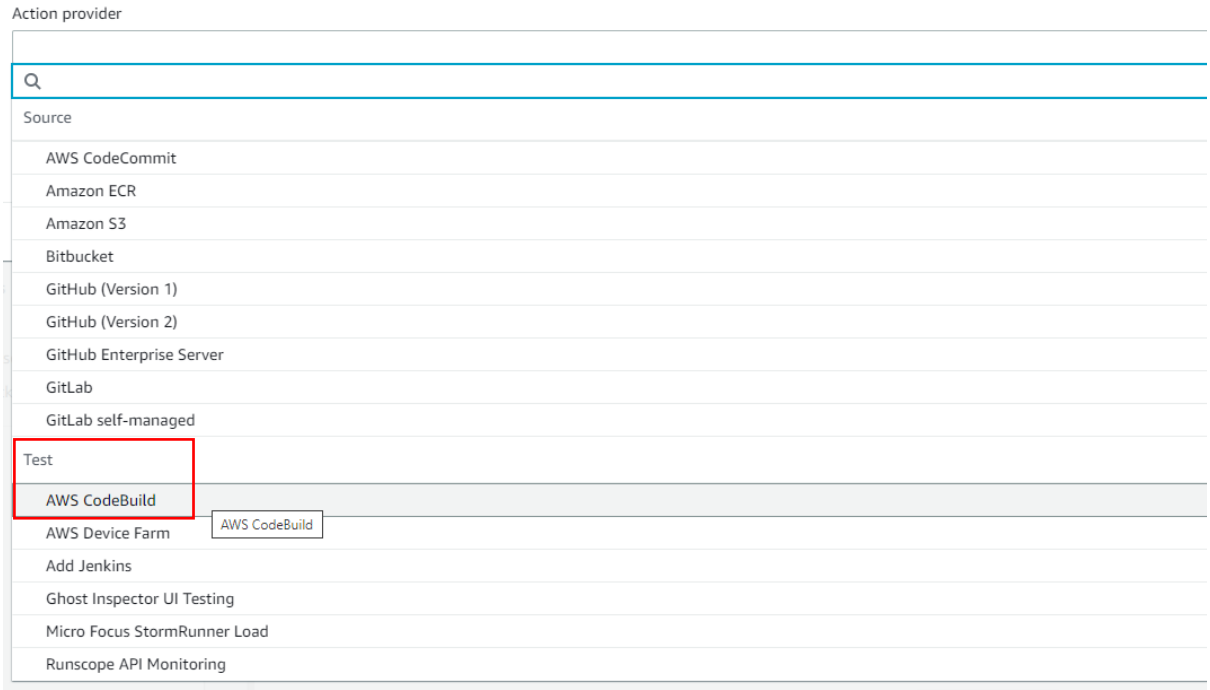
4. Now click on the add action group button.



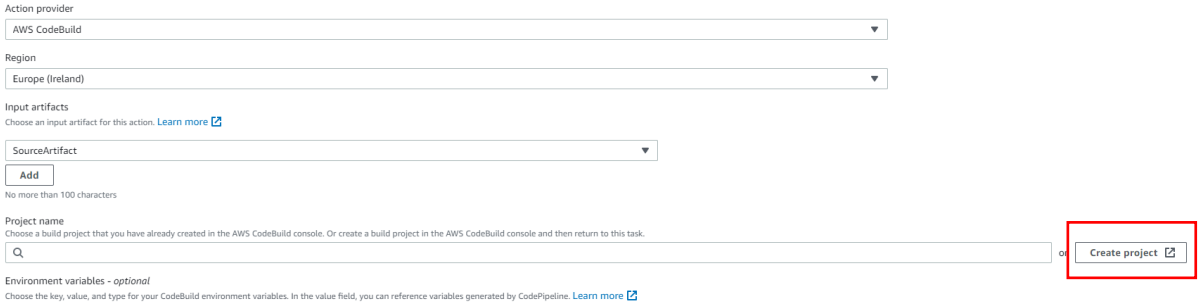
5. Now you need to give it a name and expand the bar for action provider



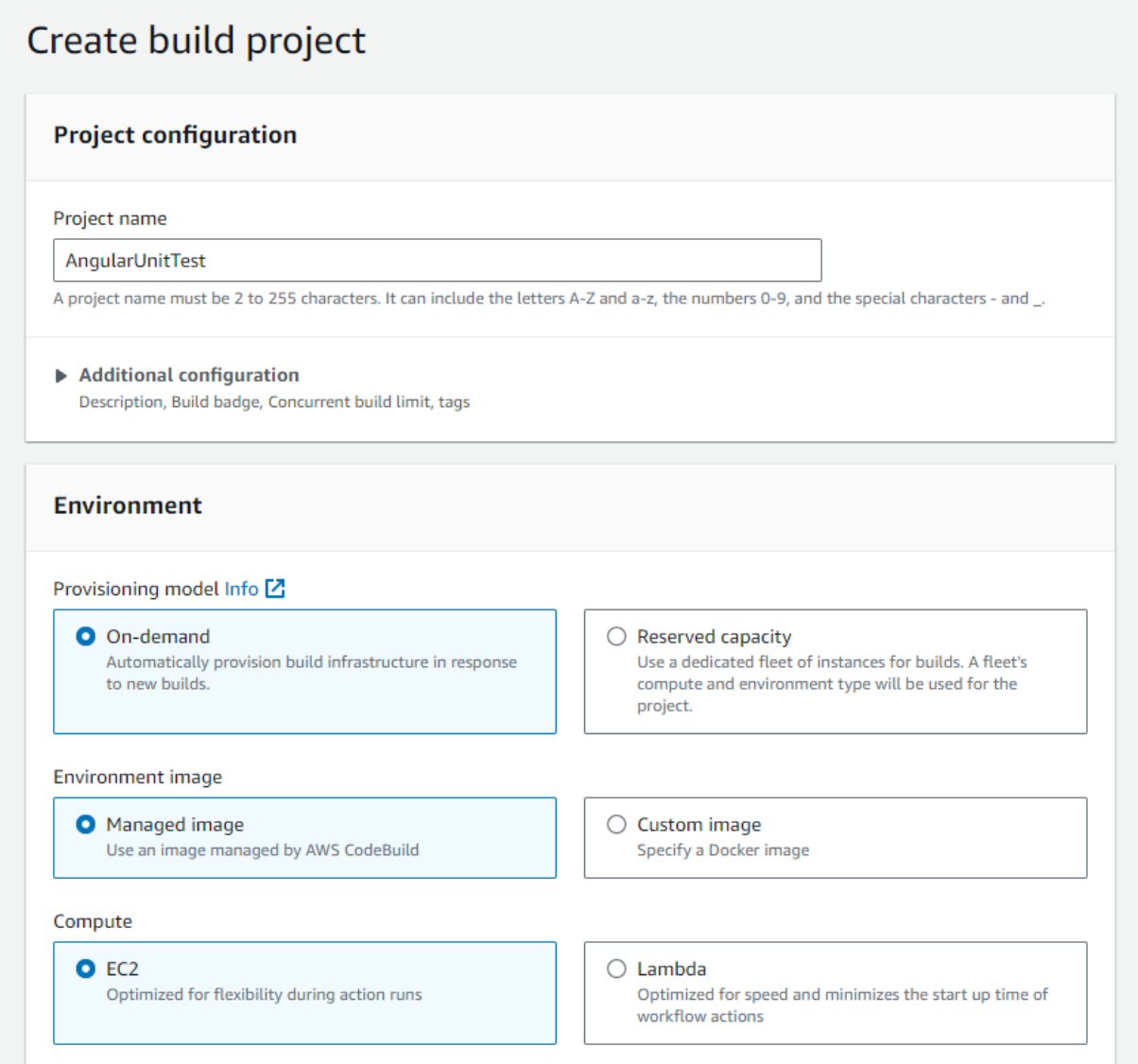
6. Here you need to choose AWS Code Build from the Test group.



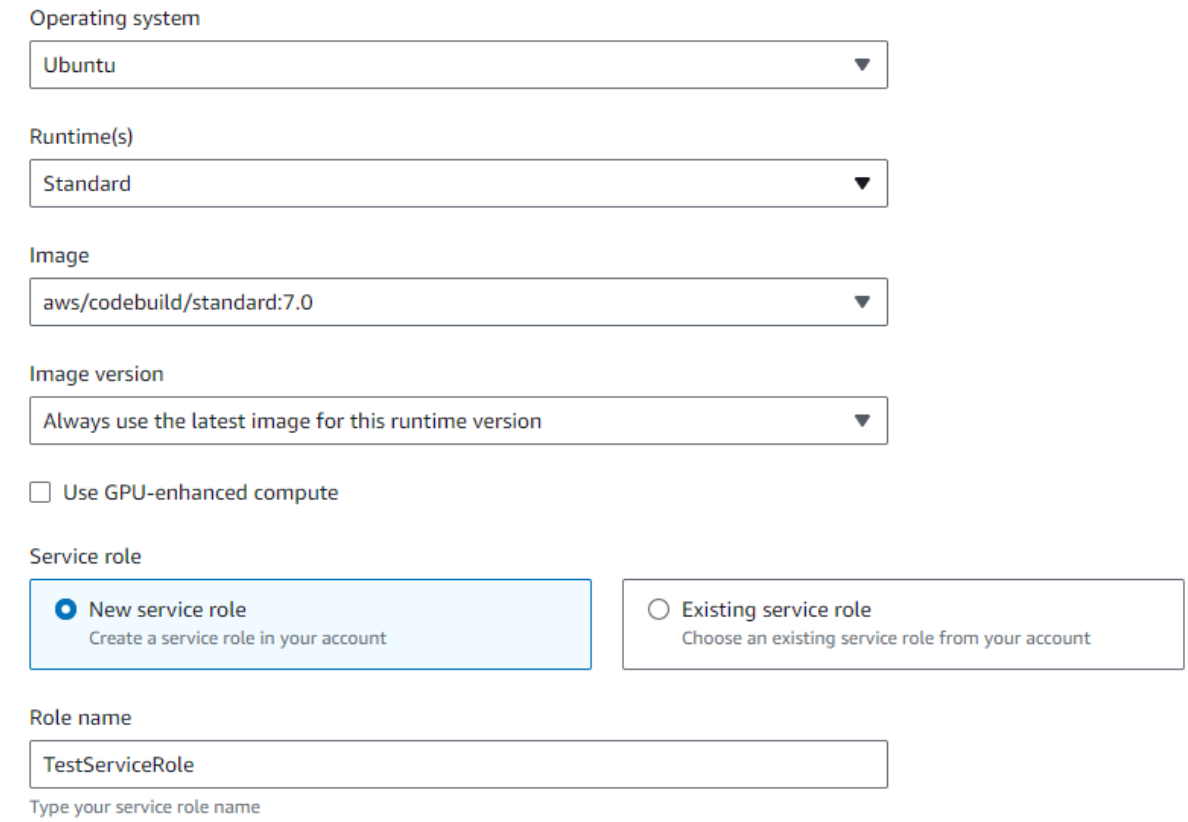
7. Here we need to choose the same region we are working on and then we need to create a new project because our new action will use the new buildspec file we just created. So, click on create project.



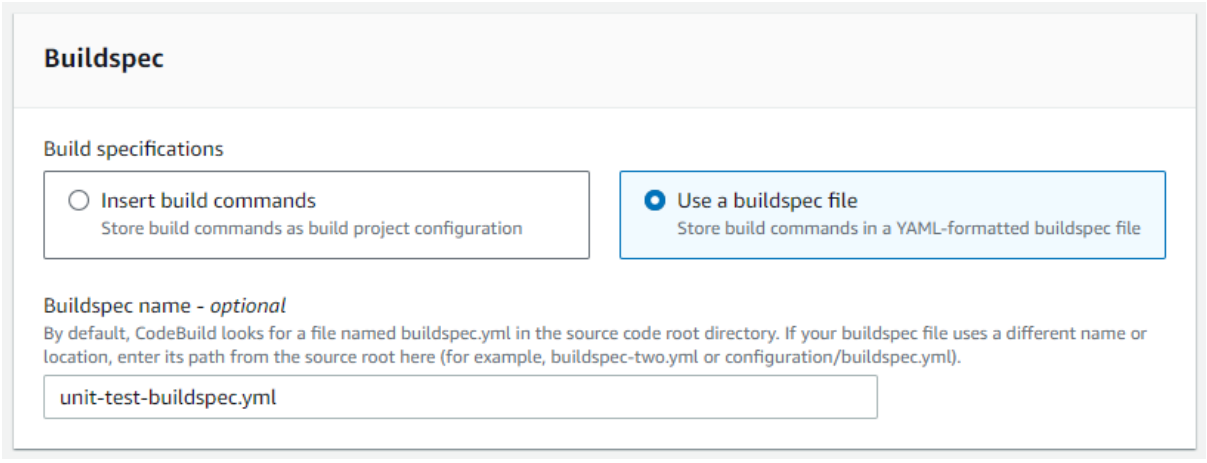
8. First, give it a name and keep the rest of the settings to default and scroll down.



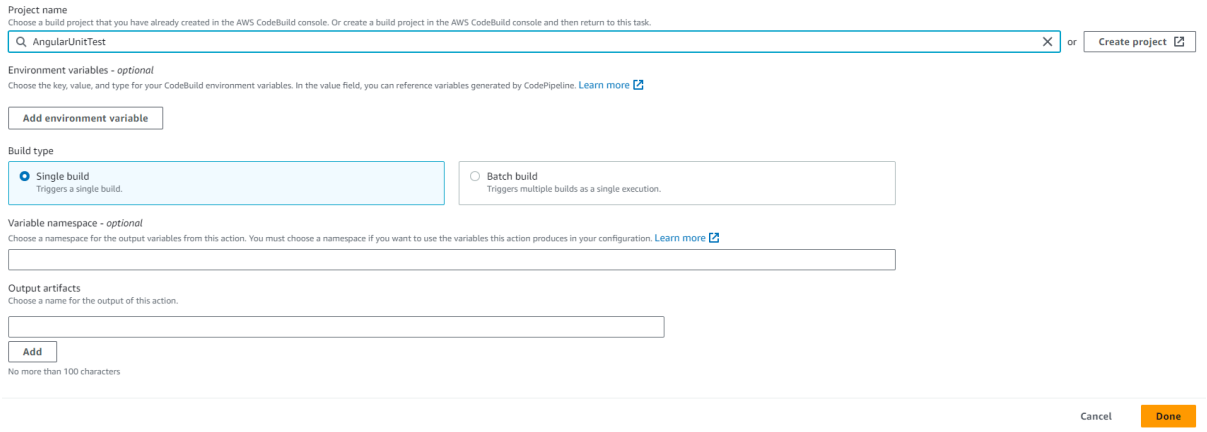
9. Then you need to choose the OS as Ubuntu and then in the service role choose Create New Role, scroll down.



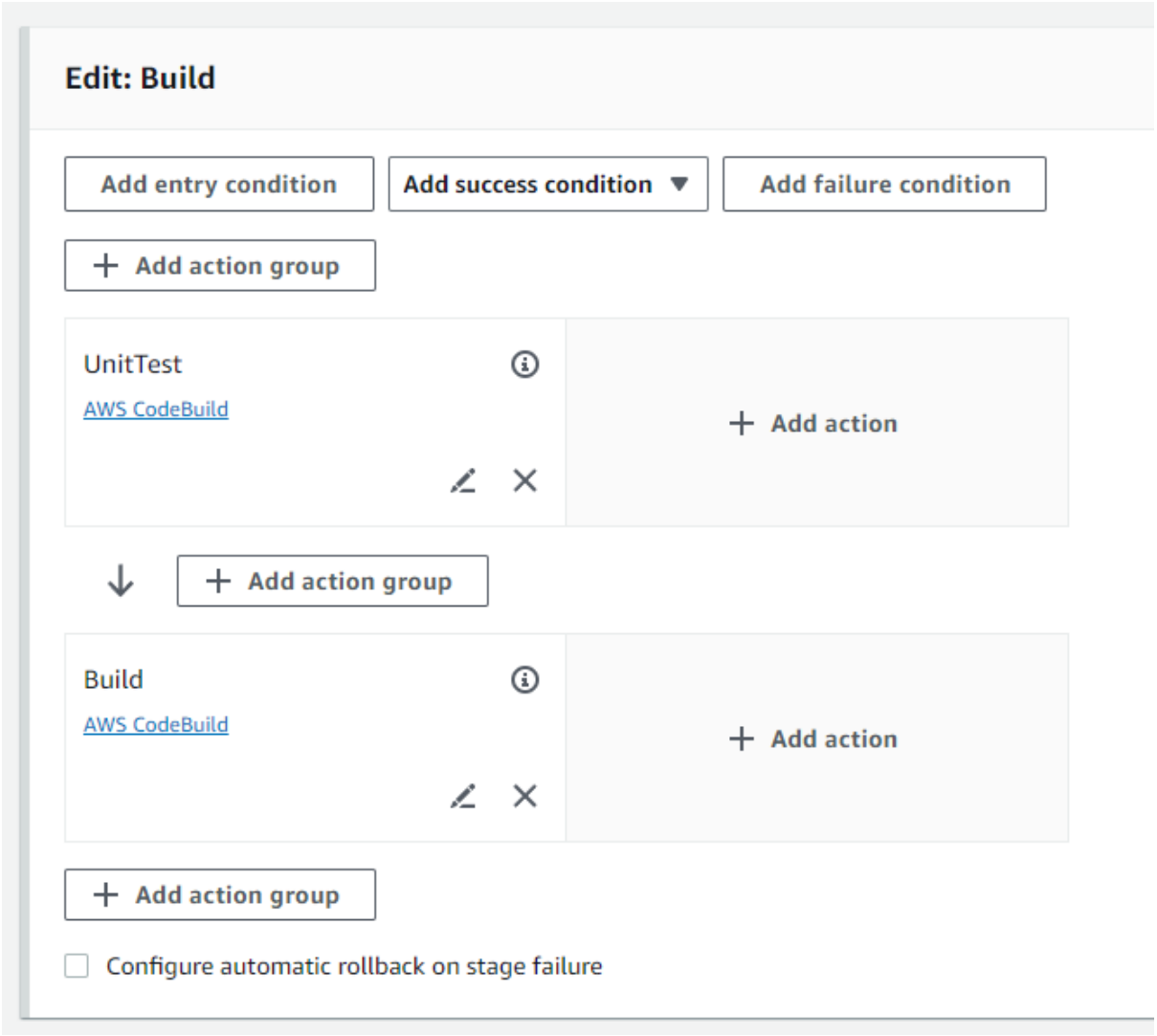
10. Now choose to use a buildspec file and then specify the name of your file. If you have created your buildspec file inside of a subfolder then you need to provide its relative path here, but we have created that file in the root of the project folder.
11. After that click on continue to code pipeline.



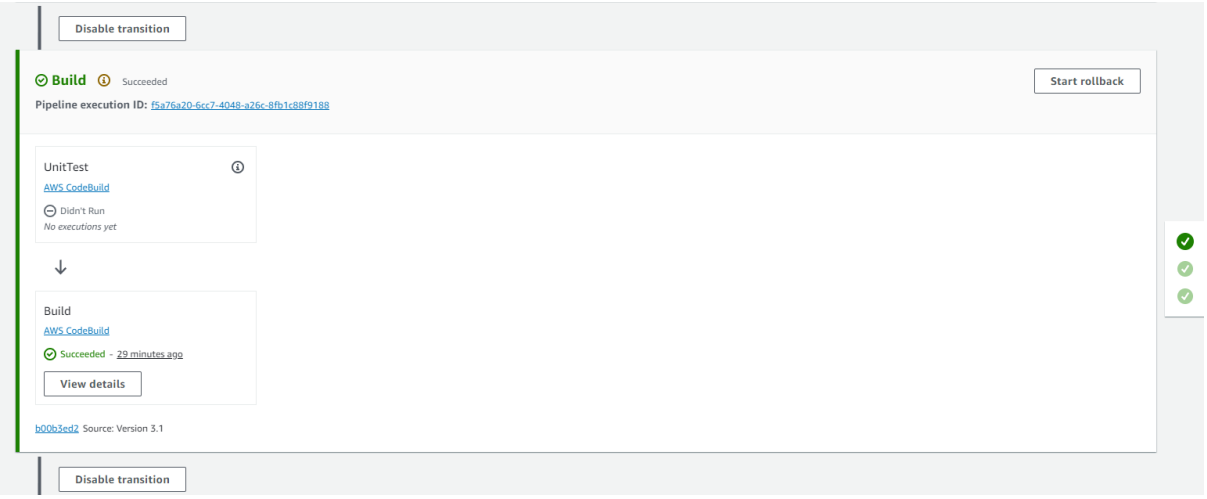
12. We can see our project name here and this time we don't need to provide any output artifact because we don't need to produce any output. So, click on done.



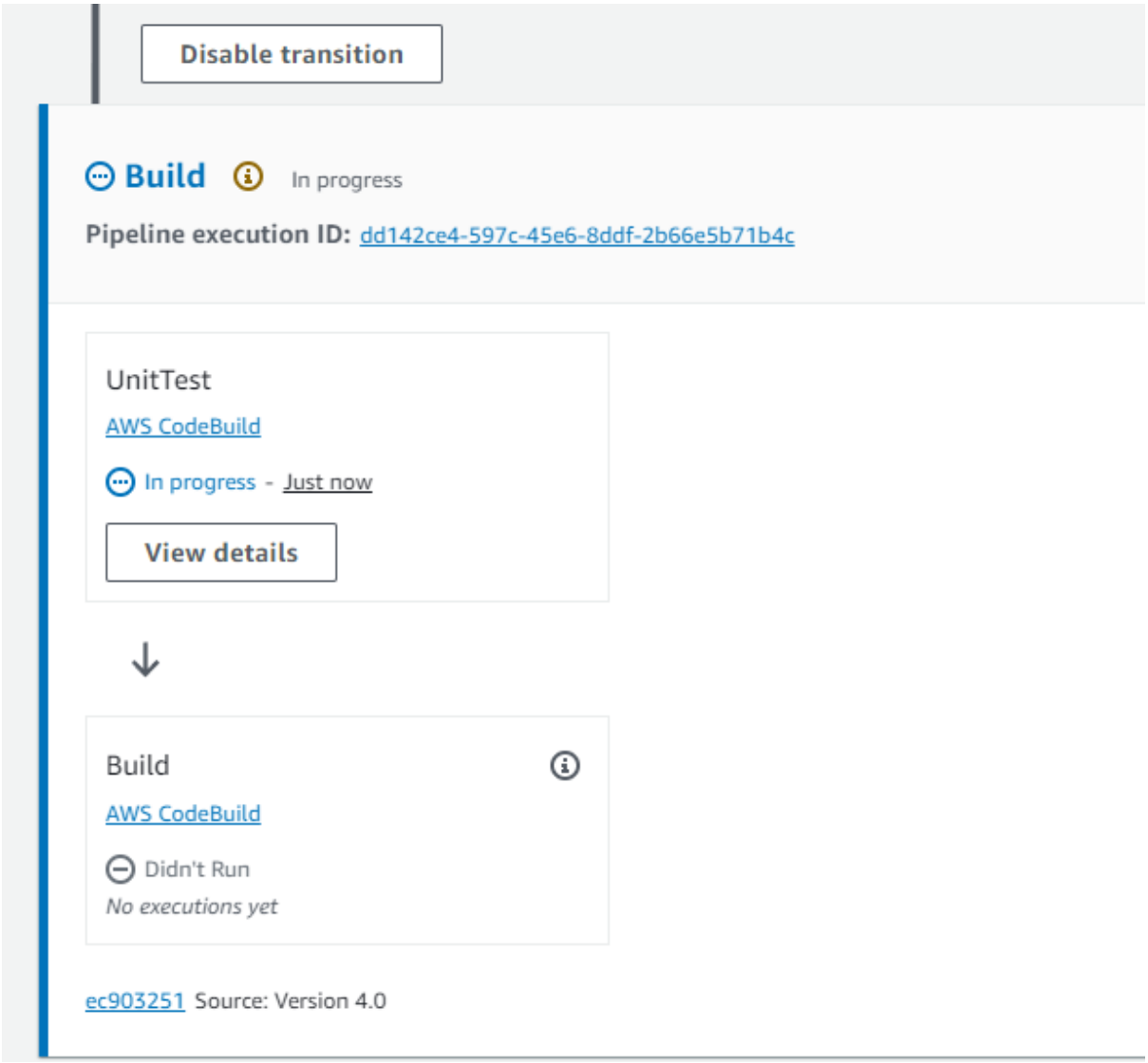
13. Below you can see that we have the unit test stage above the build stage. Now here also click on done and close the editing mode of the build stage.
14. Then you need to scroll up to the top of the page and save your changes.



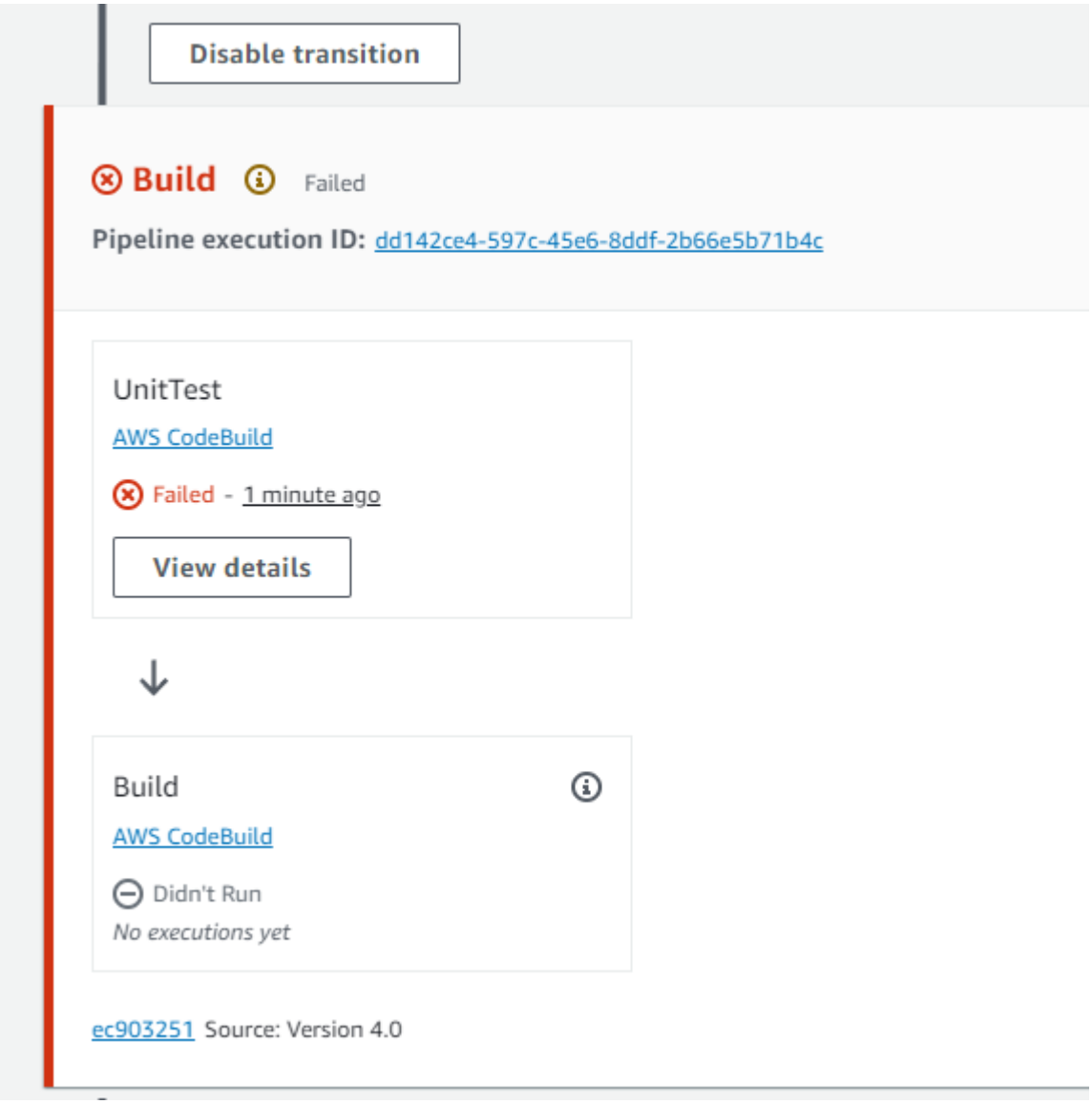
15. As you see, our new test action was added to the Build stage before the build action. Its current state is 'Didn't Run', but it will run in the next pipeline execution. So, let's go back to our code and try to break it again.



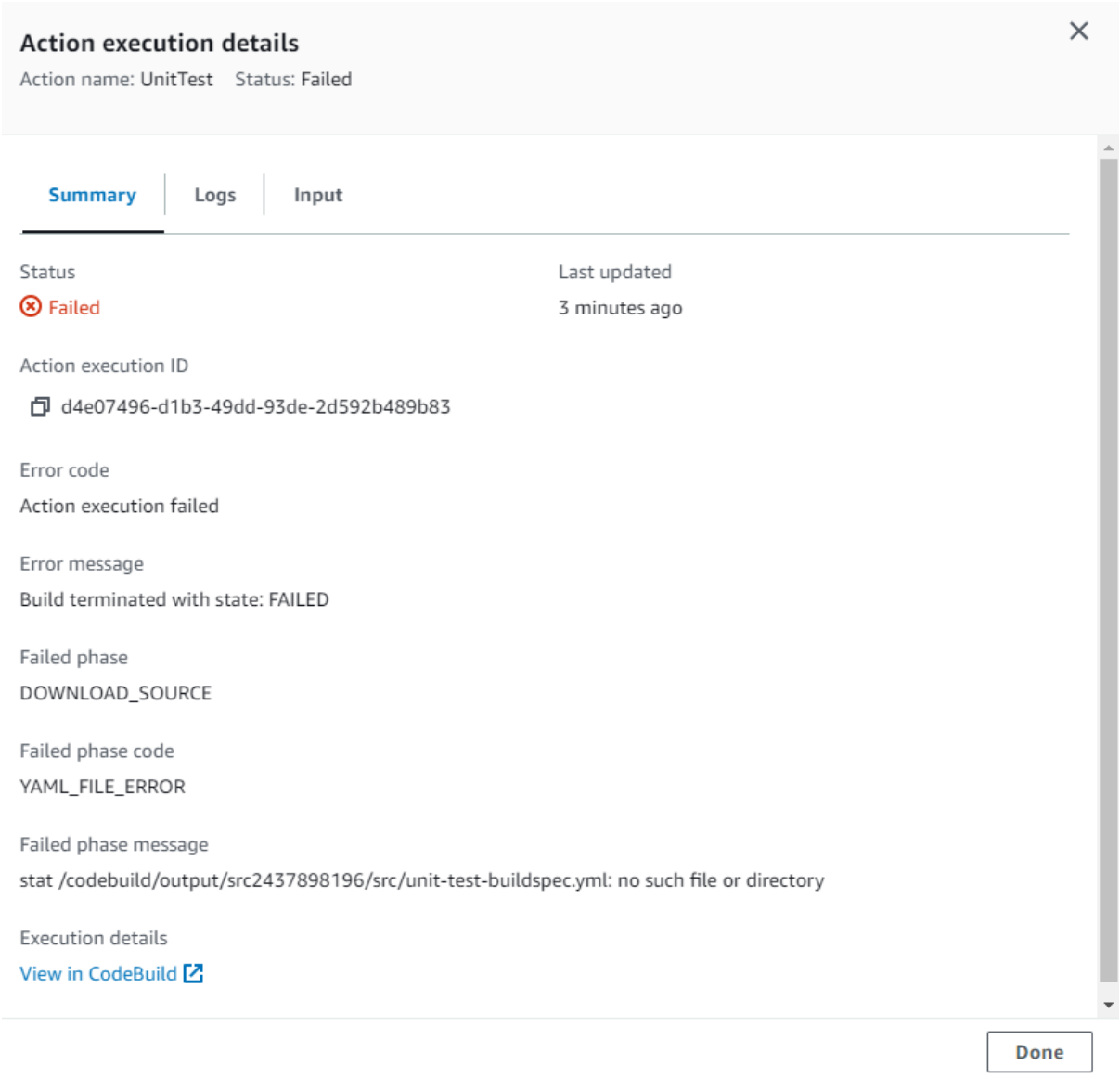
16. Like we did at the start, change the version of your website to 4.0 and then change the operator from addition to division. So, we have simulated the same mistake.
17. Now we will push our code to the repository and soon after that our pipeline will start executing, so this time we will see the preventive method which we have set up to check whether the mistake is getting caught or not.
18. Yes, the pipeline execution is in progress. As you see, the source stage was finished, and its source version is 4.0. The build stage is currently in progress. Its source version is 4.0, too. But this time, our new Unit Test action is running first in the build stage. The build action has not been triggered yet because it will wait for the test action to succeed beforehand. This will take time because CodeBuild will provision a new container and make the necessary installations we configured as before.



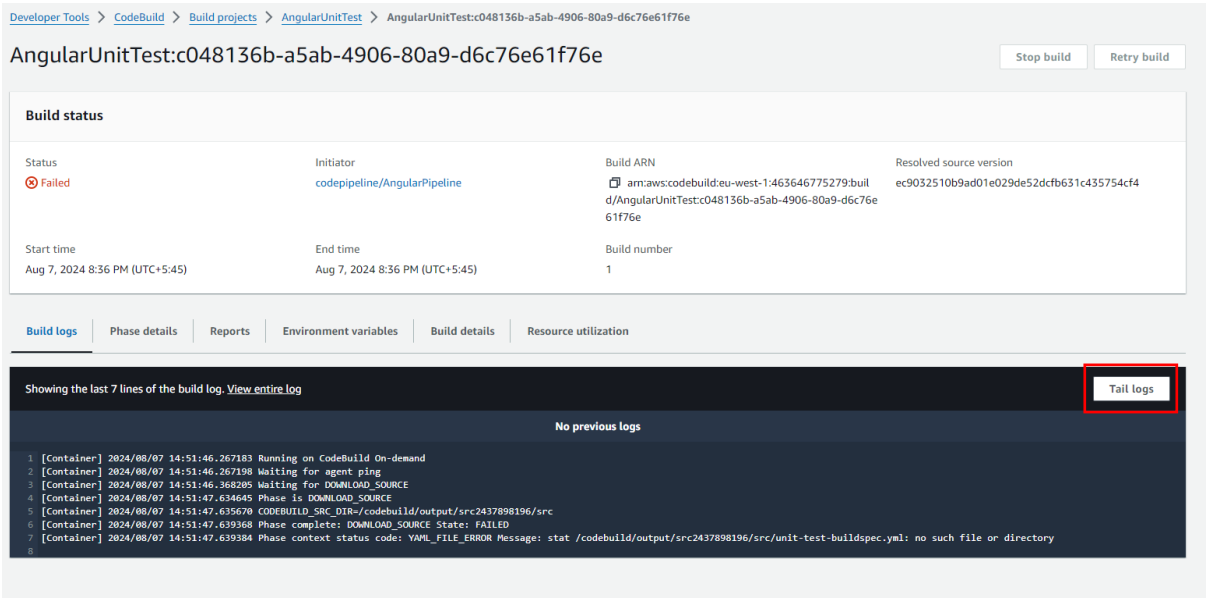
19. Below you can see that the pipeline execution has failed on the build stage. As you see, the UnitTest action failed as we expected, and the build action did not execute as a result.



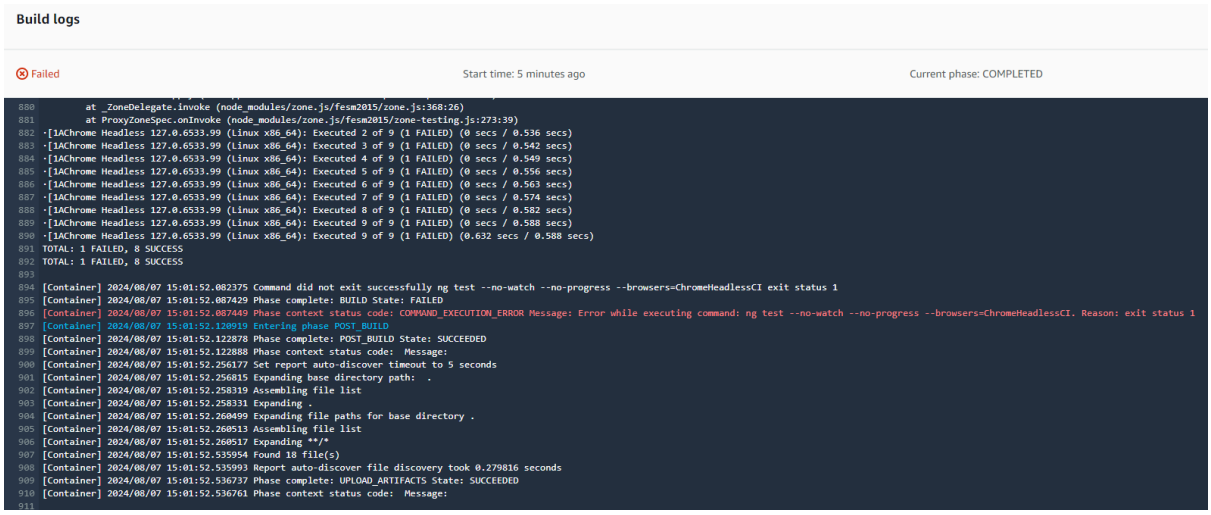
20. Now click on view details then click on view in code build.



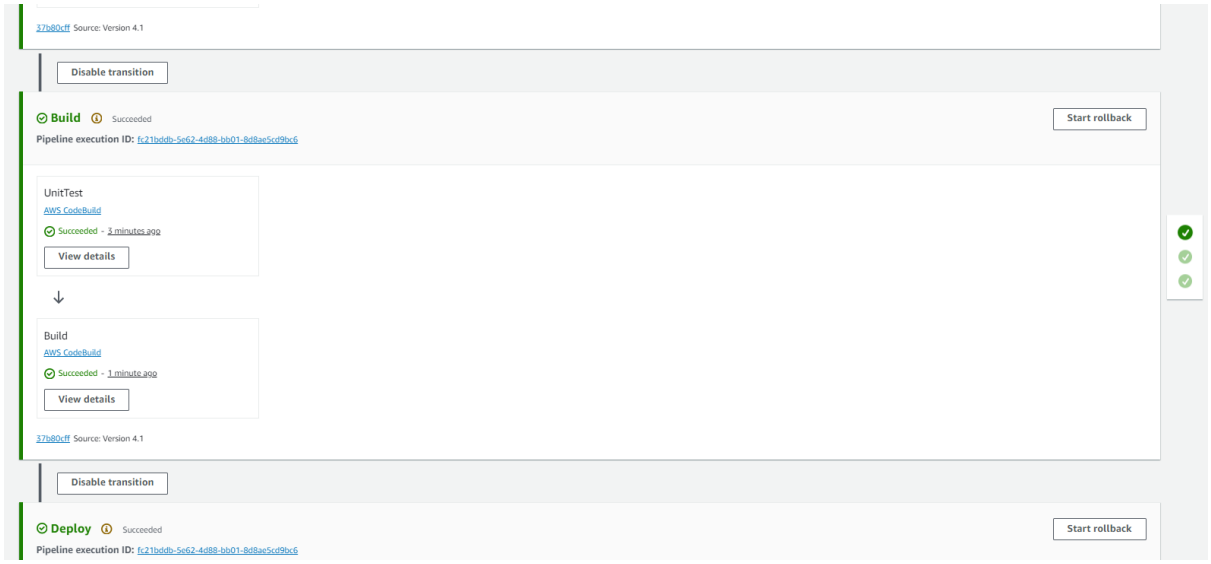
21. Now click on the tail logs button.



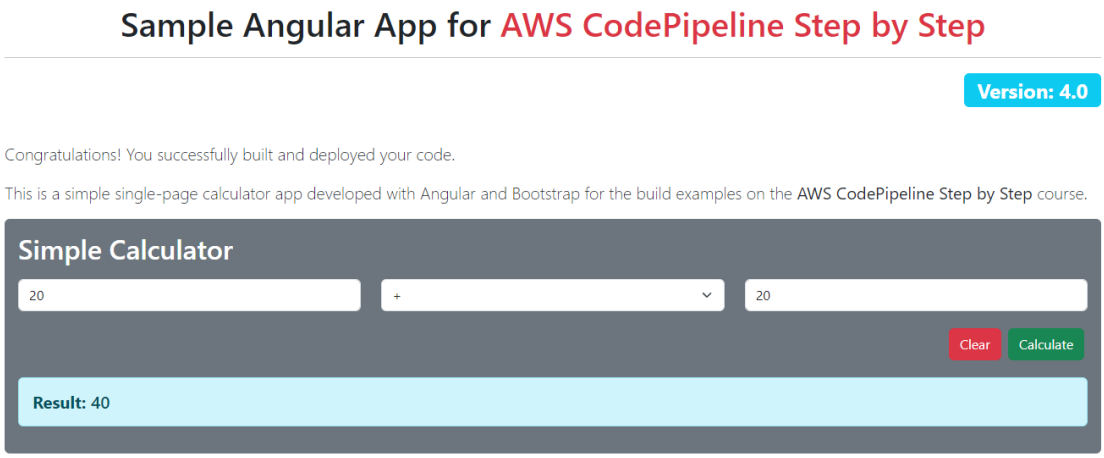
22. Here you can see that proper error that caused the failure.



23. Also, if you go to the website and try to calculate something then you will see that it is working fine because this time there was no pipeline execution. The pipeline stopped at the Unit Test stage.
24. Now go back to VS Code, change the operation back to its original state, and push the code once again.
25. Below you can see that our build was successfully executed.



26. Now if you go to your website and refresh it then you can see the new version and your addition operator is working properly.



27. Now do not delete your pipeline or the repository because we will be using our current pipeline in the next set of labs.