😊 Creating our first Code Pipeline

AWS CodePipeline is a fully managed continuous integration and continuous delivery (CI/CD) service provided by Amazon Web Services (AWS). It automates the steps required to release software changes, such as building, testing, and deploying, enabling rapid and reliable application updates.

Key Features of AWS CodePipeline:

1. **Pipeline Automation**: Automates the entire software release process.
2. **Integration**: Integrates seamlessly with AWS services (like CodeCommit, CodeBuild, and Elastic Beanstalk) and third-party tools (like GitHub, Jenkins, and Bitbucket).
3. **Continuous Delivery**: Supports workflows for continuous integration and continuous deployment.
4. **Custom Actions**: Allows defining custom actions using Lambda or other tools for unique requirements.
5. **Parallel and Sequential Steps**: Supports defining pipelines with multiple stages and steps, executed sequentially or in parallel.
6. **Pay-as-you-go Pricing**: Charges based on pipeline usage, with no upfront costs.

How AWS CodePipeline Works:

1. **Source**: CodePipeline listens for changes in the source repository (e.g., CodeCommit, GitHub, or S3).
2. **Build**: Automatically triggers a build process (e.g., via AWS CodeBuild or another build tool).
3. **Test**: Runs automated tests to validate changes.
4. **Deploy**: Deploys the application to specified environments (e.g., Elastic Beanstalk, EC2, or Lambda).

Example Workflow:

1. **Source Stage**:
    - A developer pushes code changes to a GitHub repository.
    - CodePipeline detects the change and triggers the pipeline.
2. **Build Stage**:

- CodePipeline invokes AWS CodeBuild to compile the source code, run tests, and produce an artifact.

3. **Test Stage**:
   - Artifacts are deployed to a test environment where integration tests are run.
4. **Deploy Stage**:
   - Validated artifacts are deployed to production (e.g., an EC2 instance, an Elastic Beanstalk environment, or a Lambda function).

Benefits of AWS CodePipeline:

1. **Automation**: Reduces manual intervention in the release process.
2. **Speed**: Enables fast, frequent, and reliable software updates.
3. **Scalability**: Handles multiple pipelines and scales automatically based on the workload.
4. **Flexibility**: Supports custom actions, integration with third-party tools, and multi-cloud deployments.
5. **Security**: Built-in integration with AWS IAM for secure access control.

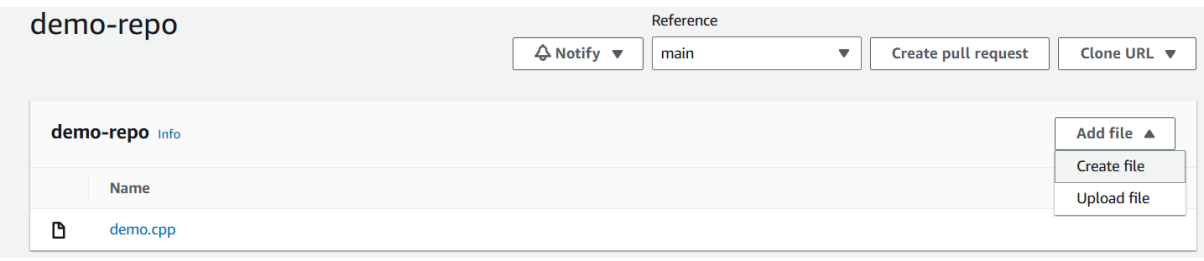
Common Use Cases:

- **Continuous Delivery**: Streamline deployments to production environments.
- **Multi-Environment Deployments**: Deploy applications to different environments (e.g., development, staging, production).
- **Testing Pipelines**: Automate testing for every code change.
- **Microservices**: Manage independent pipelines for different microservices.

AWS CodePipeline is an essential tool for implementing DevOps practices and ensuring efficient, automated, and reliable software delivery pipelines.

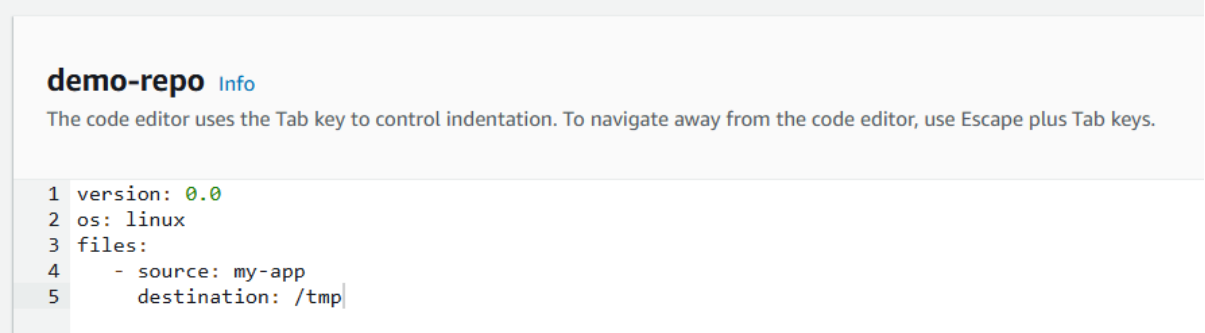
😄 To begin with the Lab:

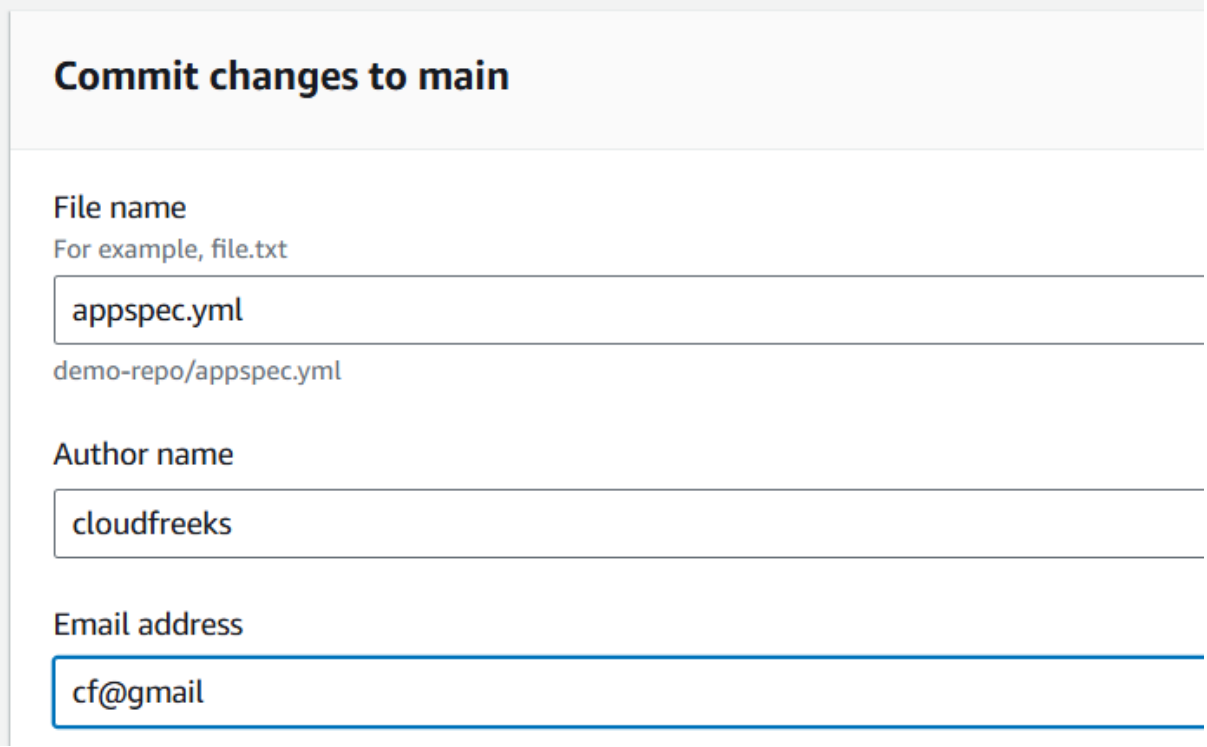
1. The first step for us is to create an appspec.yml file in the code commit repository because the pipeline expects it to be there. So, basically this appspec.yml file should be added to the root of our repository.
2. So, go to your repository and click on Create file.



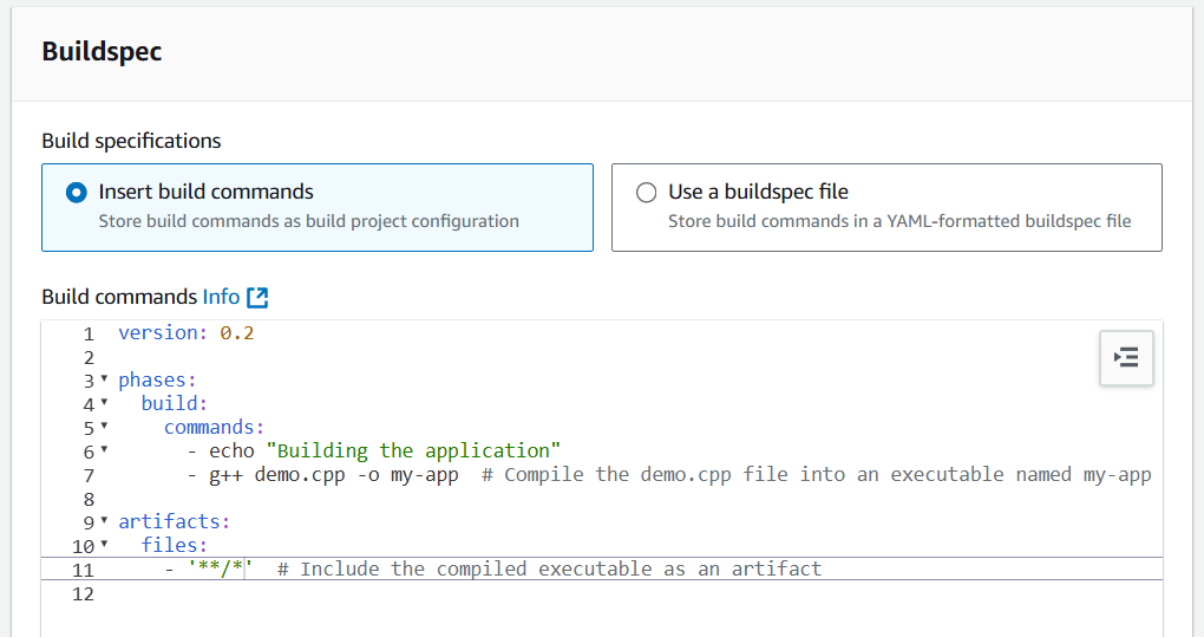
3. Here you need to use the below code and give the details then click on commit changes.

version: 0.0
os: linux
files:
- source: my-app
destination: /tmp

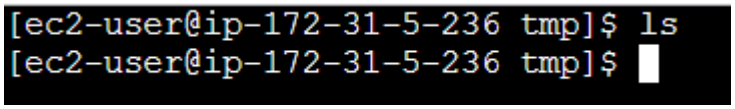


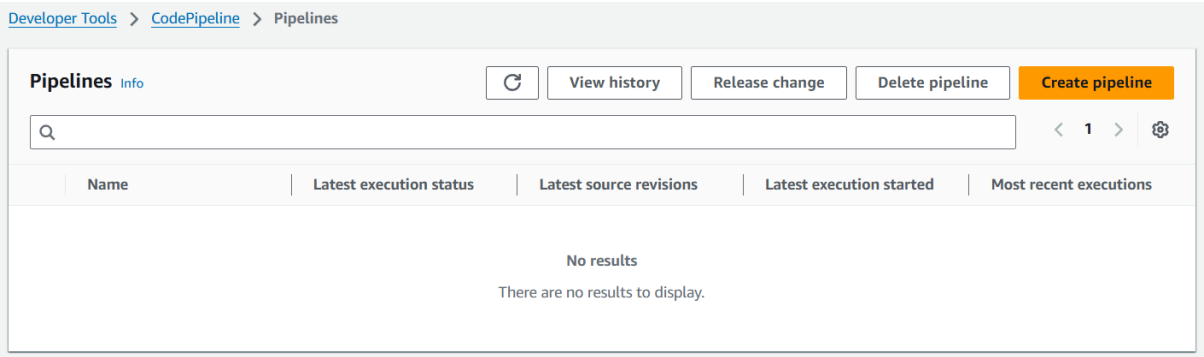


4. After that from code build open your project and click on Edit your project. Then scroll down to build spec file.
5. Here on line 11, you need to add the wildcard characters so that it picks up both of your files from the repository.

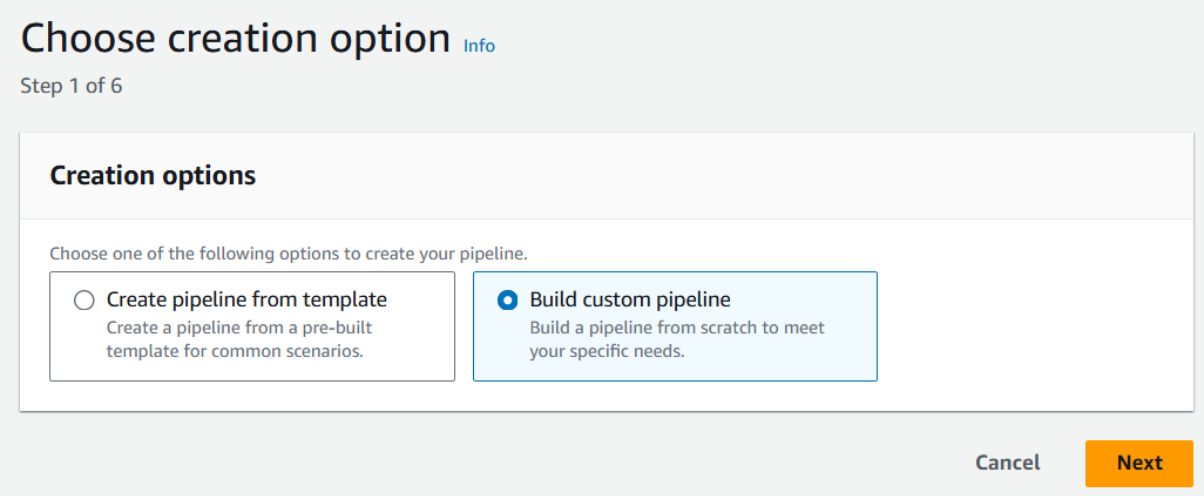


6. Now we are ready to create our code pipeline so click on Create Pipeline.
7. Also, before that we can just clear out the tmp directory from our EC2 instance as well.

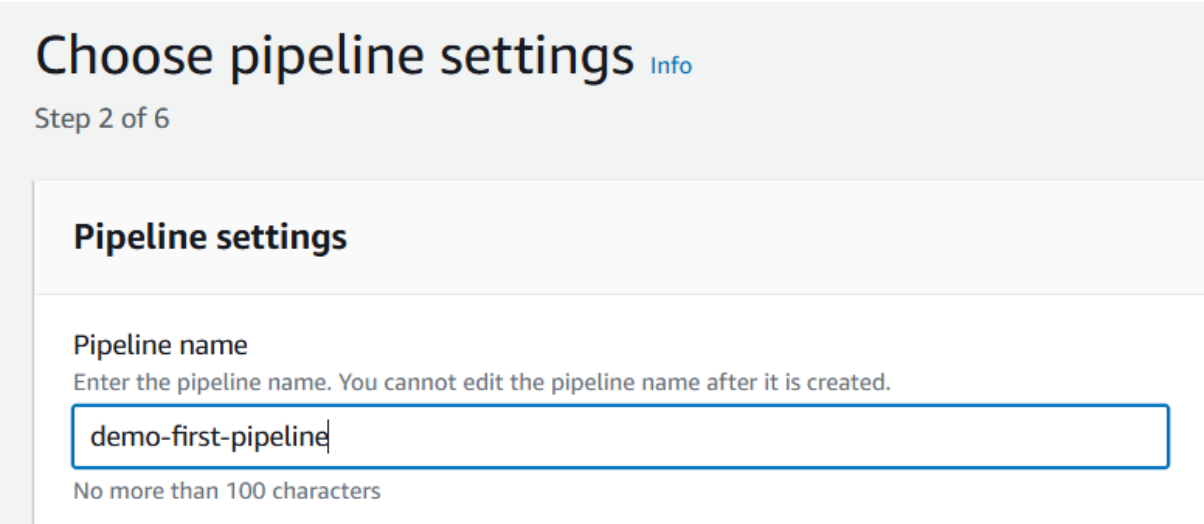




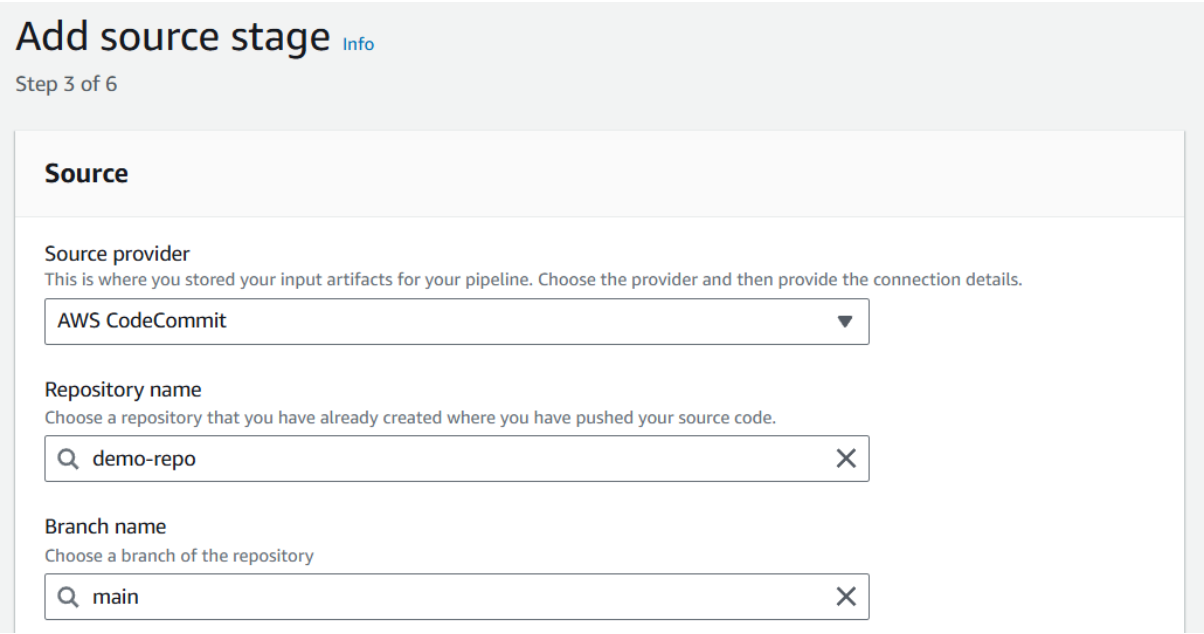
8. On step 1 choose build custom pipeline and click on Next.



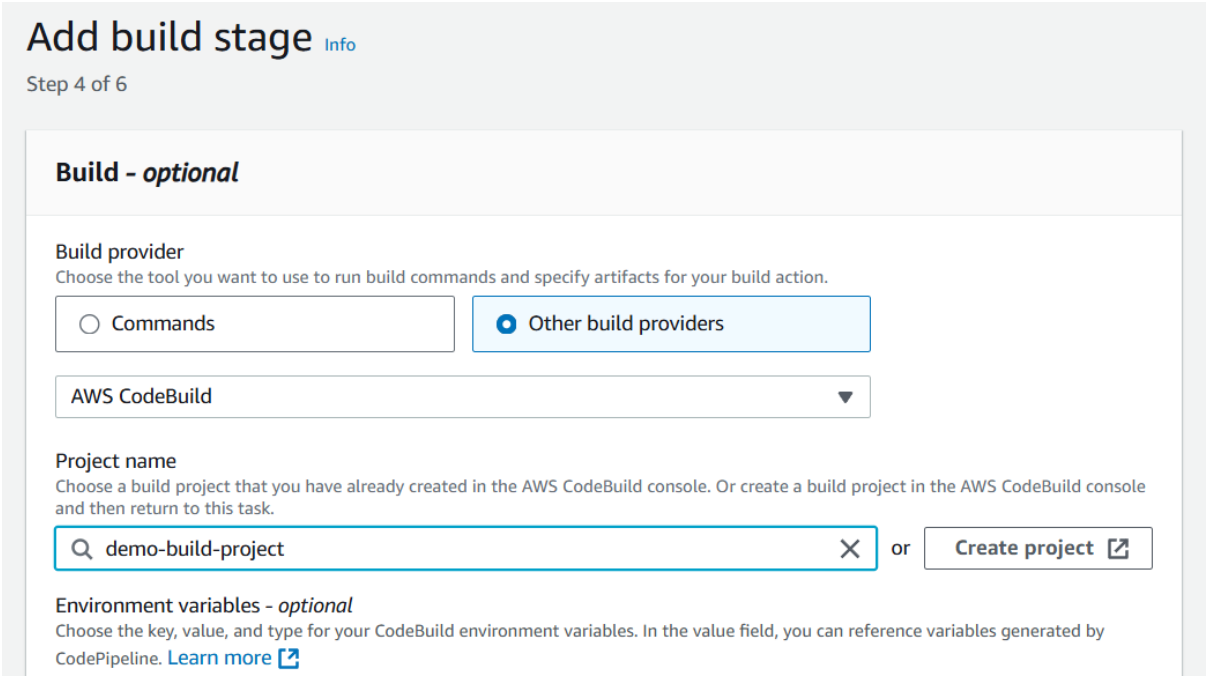
9. On step 2 just give it a name and move to next step.



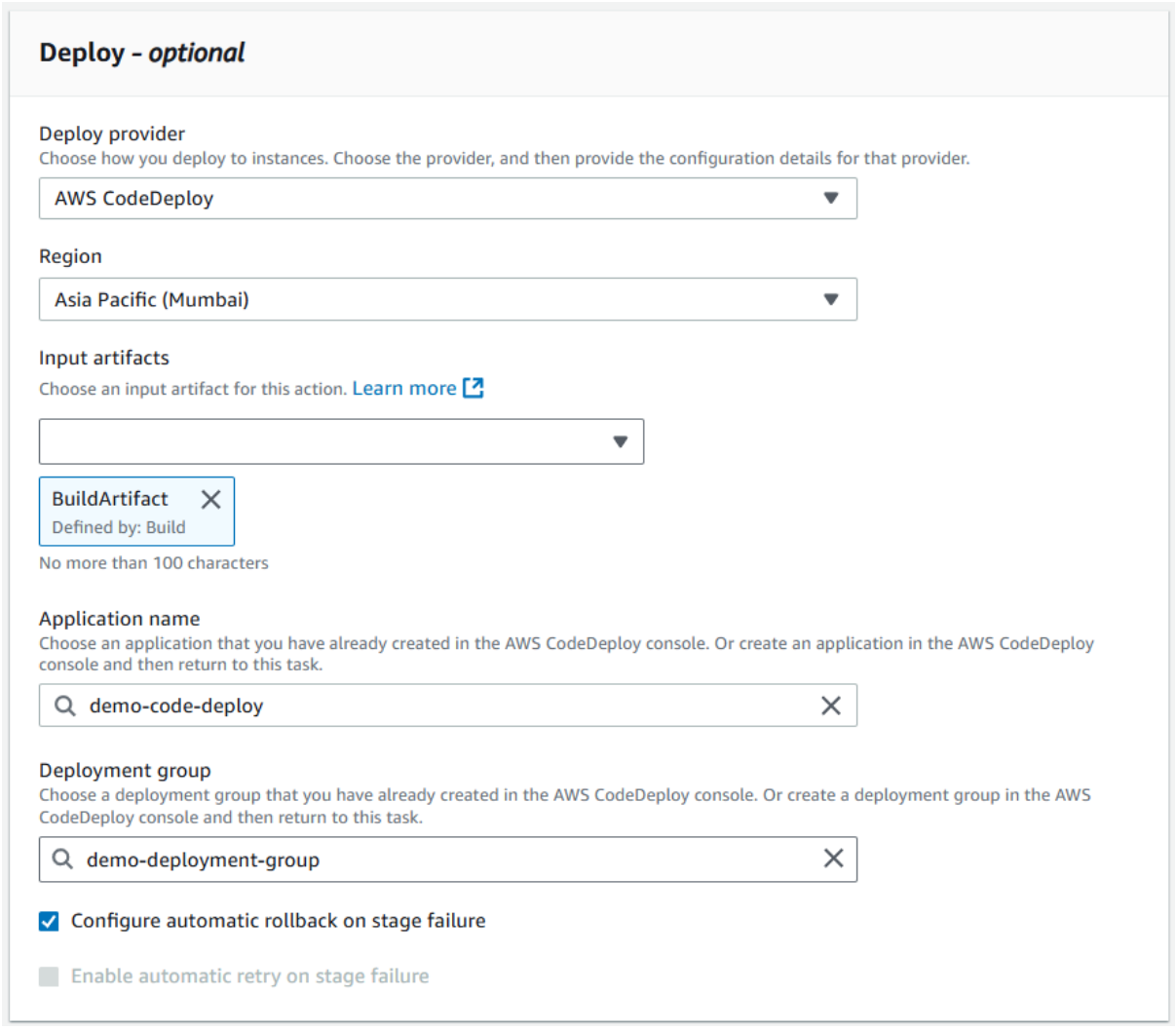
10. Here you need to choose code commit as your source provider and choose your repository.



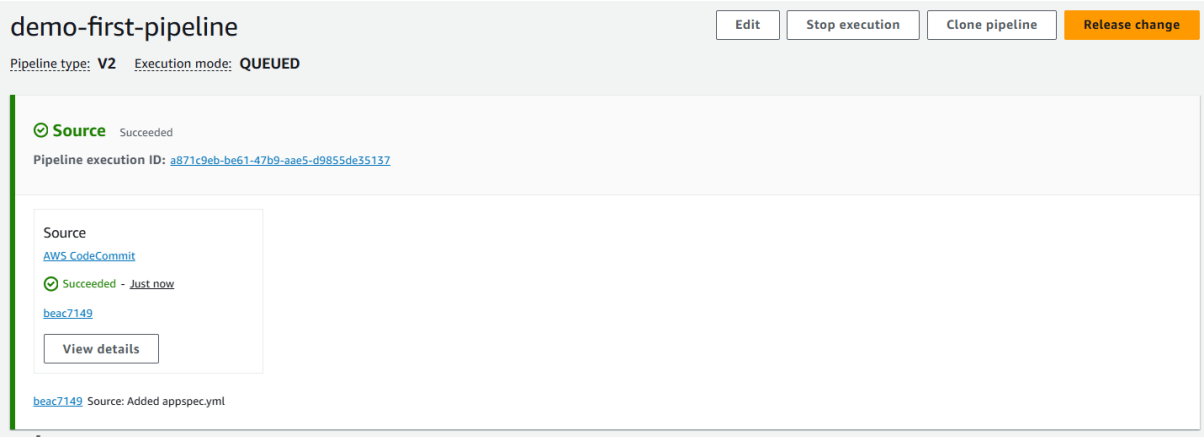
11. On step 4 choose other build providers and choose AWS Code build then choose your project. Move to step 5.



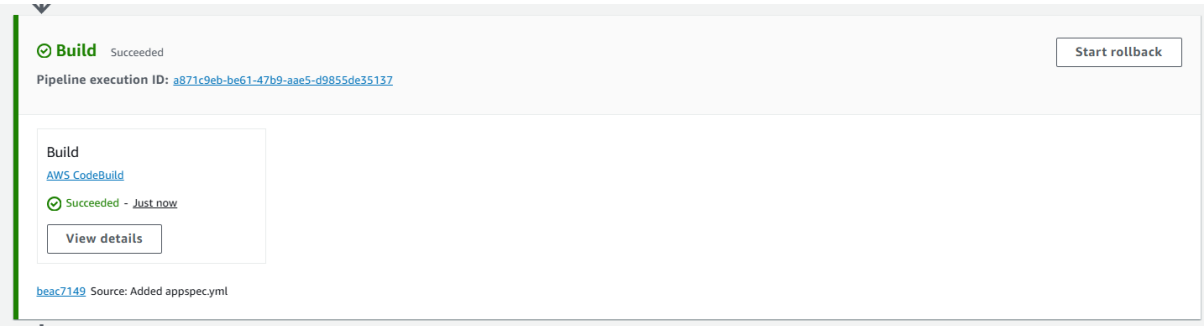
12. On the last step we have to choose code deploy as our deploy provider and choose the build artifact as input artifact then choose our application name and deployment group.



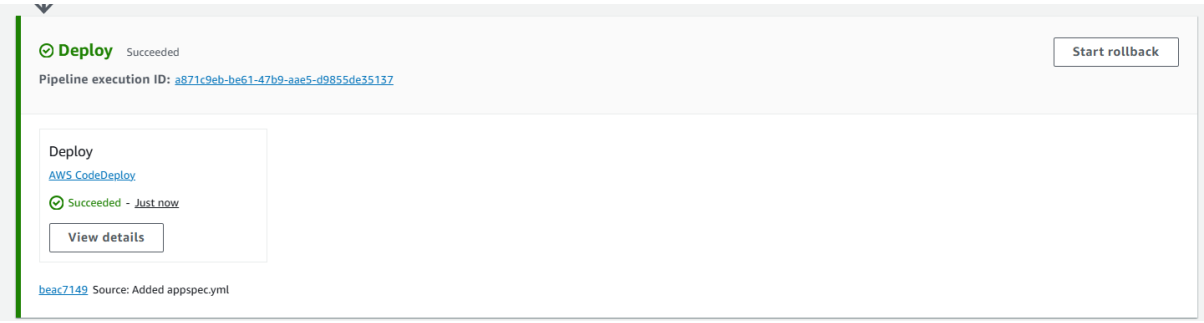
13. Move to the review page and create your pipeline. Below you can see that our first step has been completed.



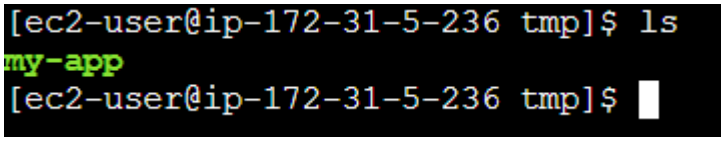
14. Our second step has also been completed for the build stage.



15. The last step for code deploy has also been completed.



16. Also, if you go to your instance and do the object listing in your tmp directory then you will see the my-app binary there.



17. In the end if you go to the code commit and open your repository, the demo.cpp file and edit file then save it.

18. So, the code pipeline will detect the change and it will start to run again. Below you can see that once it has detected the changes the execution is in progress.