

CI/CD Pipeline Implementation for Scalable and Reliable Deployments

Introduction

This project aims to improve its software development and deployment process by implementing a CI/CD pipeline. This pipeline automates key stages such as code build, testing, and deployment, ensuring faster and more reliable delivery of updates. The solution integrates AWS services, including CodePipeline, CodeBuild, and Elastic Beanstalk, to streamline operations and reduce manual intervention. Additionally, a manual approval step before production deployment ensures quality control. This document details the implementation process and steps taken to set up the CI/CD pipeline as Proof of Concept (POC).

Table of Contents

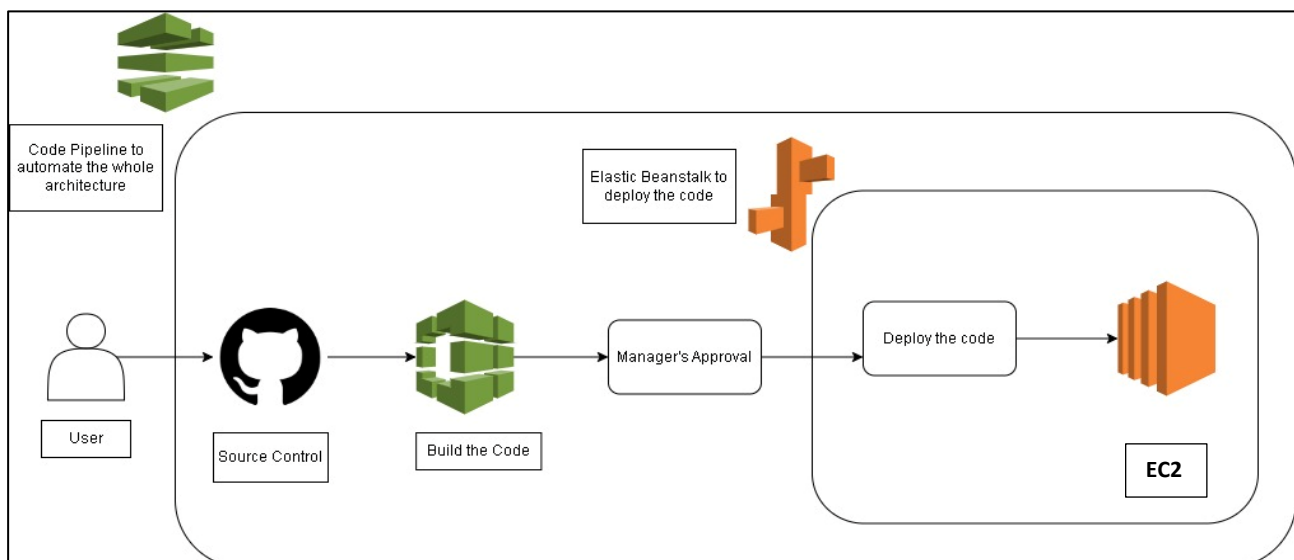
1. Project Overview
2. Architecture
3. Key Components
4. Tools and Technologies
5. Implementation Steps
6. Testing Changes
7. Troubleshooting IAM Permissions for Code Pipeline Role Creation.
8. Conclusion

1. Project Overview

Ample Technologies is implementing a CI/CD pipeline to automate the build, test, and deployment processes, ensuring faster, error-free delivery. The pipeline integrates AWS services and includes a manual approval step before production rollouts. This document outlines the tools, architecture, and implementation steps involved in this Proof of Concept (POC).

2. Architecture

The CI/CD pipeline architecture is depicted below:



3. Key Components

1. **Source Control (GitHub):** Repository to store and manage application code and provide version control for collaborative development.
2. **Build Automation (AWS CodeBuild):** Builds the application code using a buildspec.yml file for build instructions.
3. **Deployment Platform (AWS Elastic Beanstalk):** Hosts the application in a web server environment configured for Java applications.
4. **Pipeline Orchestration (AWS CodePipeline):** Automates the CI/CD process integrating source, build, approval, and deployment stages.
5. **Manual Approval from the Manager:** Ensures manager's approval before deploying to production.

4. Tools and Technologies

1. **AWS CodePipeline:** Automates the entire CI/CD workflow, integrating stages like source, build, approval, and deployment.
2. **GitHub:** A platform to store and manage the source code, enabling collaboration and version control.
3. **AWS CodeBuild:** Compiles source code, runs tests, and generates build artifacts for deployment.
4. **AWS Elastic Beanstalk:** Simplifies the deployment process by providing a managed environment for hosting applications.

5. Implementation Steps

1. Code Preparation:

- Obtain the Java code.
- Create a GitHub repository and upload the code.

1. Create a README File:

```
echo "# Order-service" >>
```

2. Initialize a New Git Repository

```
git init
```

3. Add All Files to Staging

```
git add .
```

4. Commit the Changes

```
git commit -m "first commit"
```

5. Rename the Default Branch to master

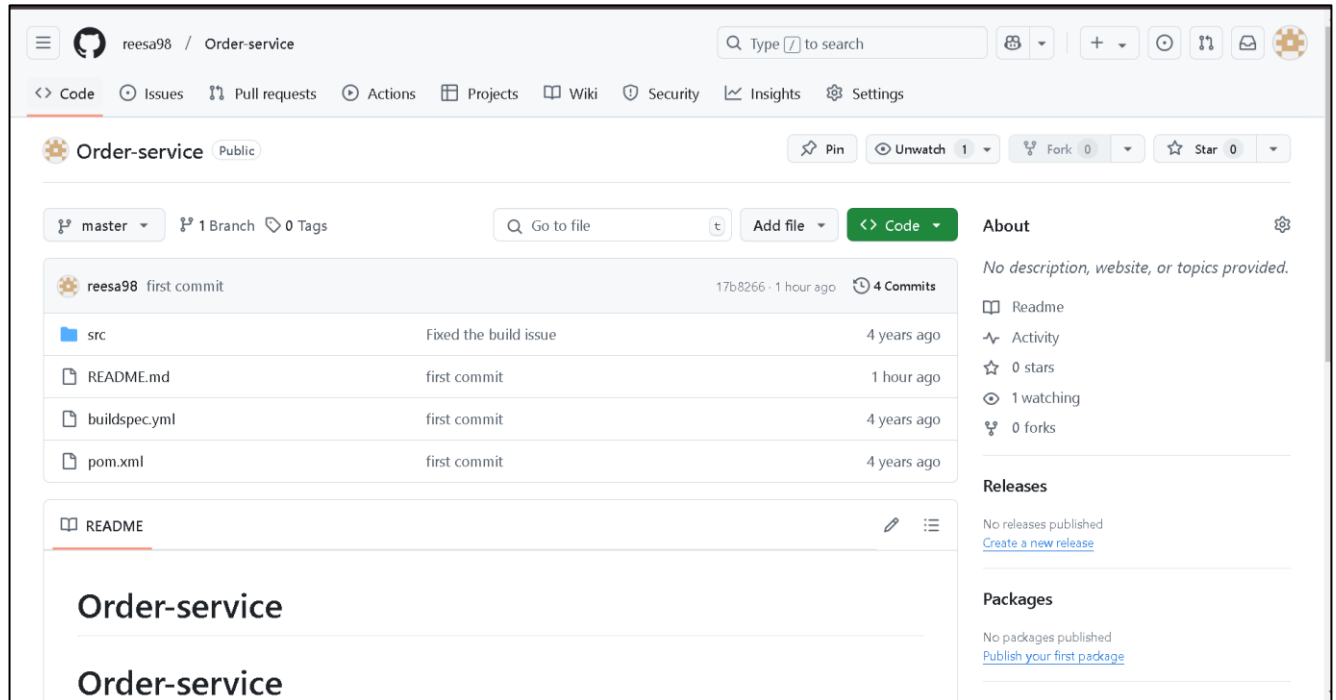
```
git branch -m master
```

6. Add a Remote Repository

```
git remote add origin https://github.com/reesa98/Order-service.git
```

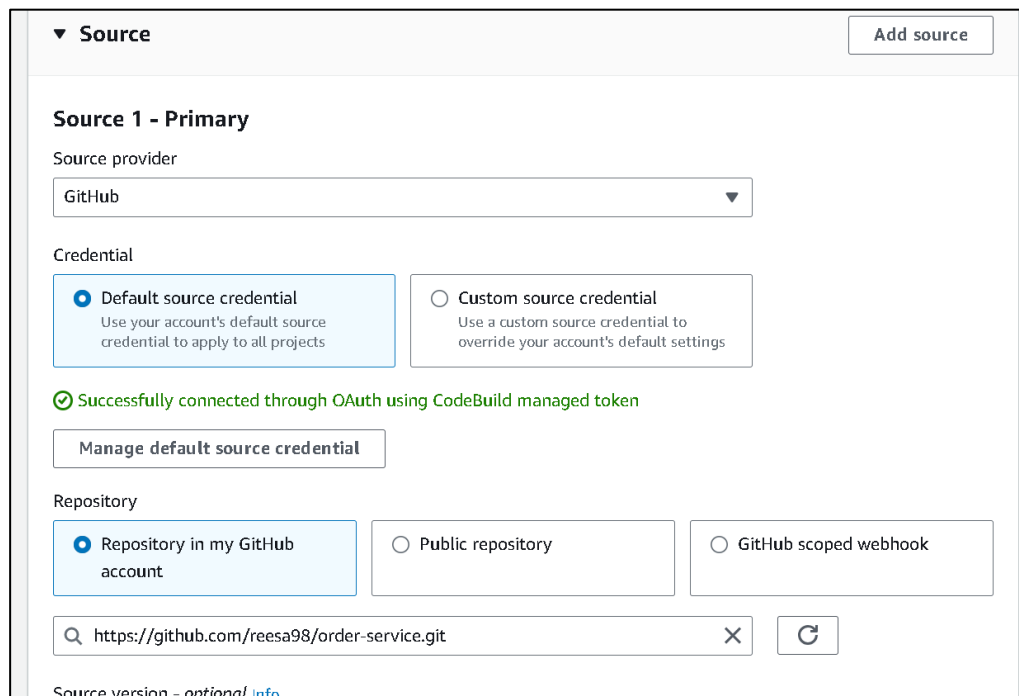
7. Push the Local Repository to GitHub

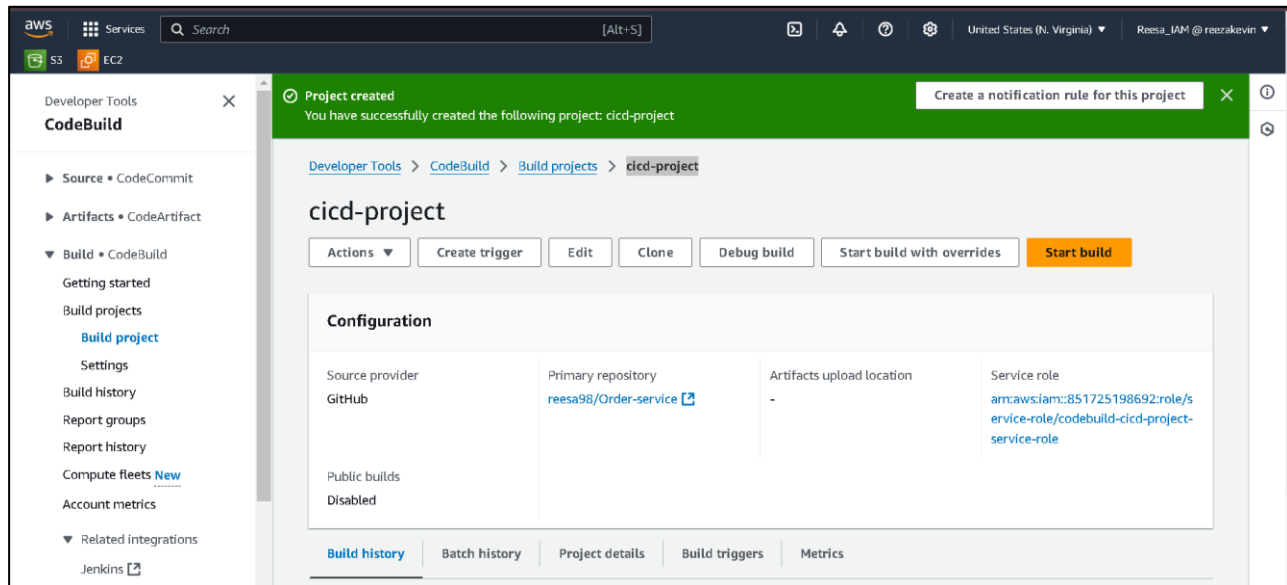
```
git push -u origin master
```



2. Create Build Project:

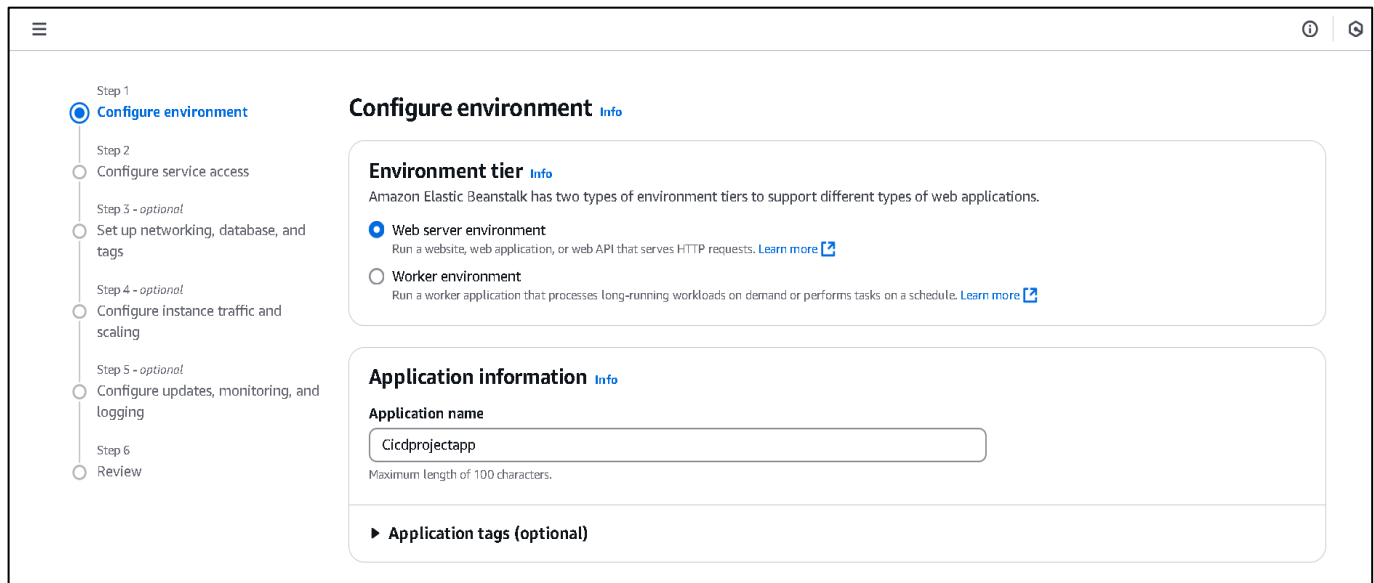
- Set up an AWS CodeBuild project.
- Use a buildspec.yml file for build instructions.





3. Set Up Elastic Beanstalk Application:

- Create an application with a Java platform.
- Configure roles and permissions for deployment.



Platform

Info

Platform type

☒ Managed platform

Platforms published and maintained by Amazon Elastic Beanstalk. [Learn more](#)

☐ Custom platform

Platforms created and owned by you. This option is unavailable if you have no platforms.

Platform

Java

Platform branch

Corretto 8 running on 64bit Amazon Linux 2

Platform version

3.7.10 (Recommended)

Step 1

Configure environment

Step 2

Configure service access

Step 3 - optional

Set up networking, database, and tags

Step 4 - optional

Configure instance traffic and scaling

Step 5 - optional

Configure updates, monitoring, and logging

Step 6

Review

Configure service access

Info

Service access

IAM roles, assumed by Elastic Beanstalk as a service role, and EC2 instance profiles allow Elastic Beanstalk to create and manage your environment. Both the IAM role and instance profile must be attached to IAM managed policies that contain the required permissions. [Learn more](#)

Service role

☐ Create and use new service role

☒ Use an existing service role

Existing service roles

Choose an existing IAM role for Elastic Beanstalk to assume as a service role. The existing IAM role must have the required IAM managed policies.

aws-elasticbeanstalk-service-role

EC2 key pair

Select an EC2 key pair to securely log in to your EC2 instances. [Learn more](#)

demo

EC2 instance profile

Choose an IAM instance profile with managed policies that allow your EC2 instances to perform required operations.

ElasticBeanstalkRole

View permission details

Cancel

Skip to review

Previous

Next

Elastic Beanstalk

Applications

Environments

Change history

Application: cicdprojectapp

Application versions

Saved configurations

Environment: Cicdprojectapp-env

Go to environment

Configuration

Events

Health

Logs

Monitoring

Alarms

Managed updates

orderserviceproject application is being deleted

Environment successfully launched.

Cicdprojectapp-env

Info

Environment overview

Health

Ok

Environment ID

e-vum2xmim8p

Domain

Cicdprojectapp-env.eba-kef9phim.us-east-1.elasticbeanstalk.com

Application name

cicdprojectapp

Platform

Change version

Platform

Corretto 8 running on 64bit Amazon Linux 2/3.7.10

Running version

-

Platform state

Supported

Events

Health

Logs

Monitoring

Alarms

Managed updates

Tags

Events (13)

Info

Filter events by text, property or value

CloudShell

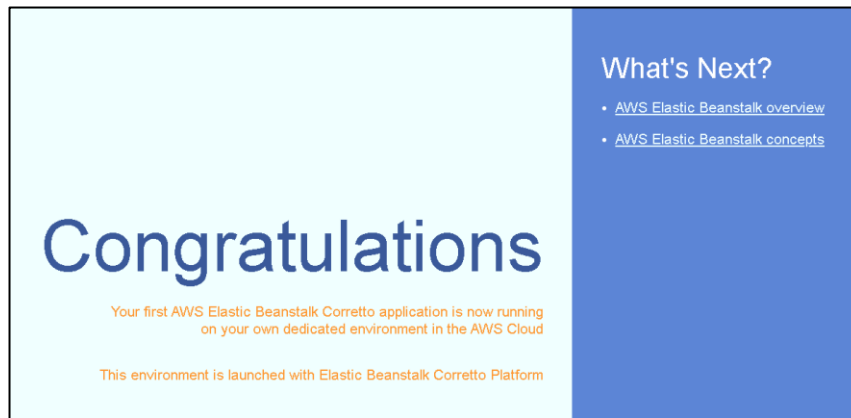
Feedback

© 2025, Amazon Web Services, Inc. or its affiliates.

Privacy

Terms

Cookie preferences



4. Configure CodePipeline:

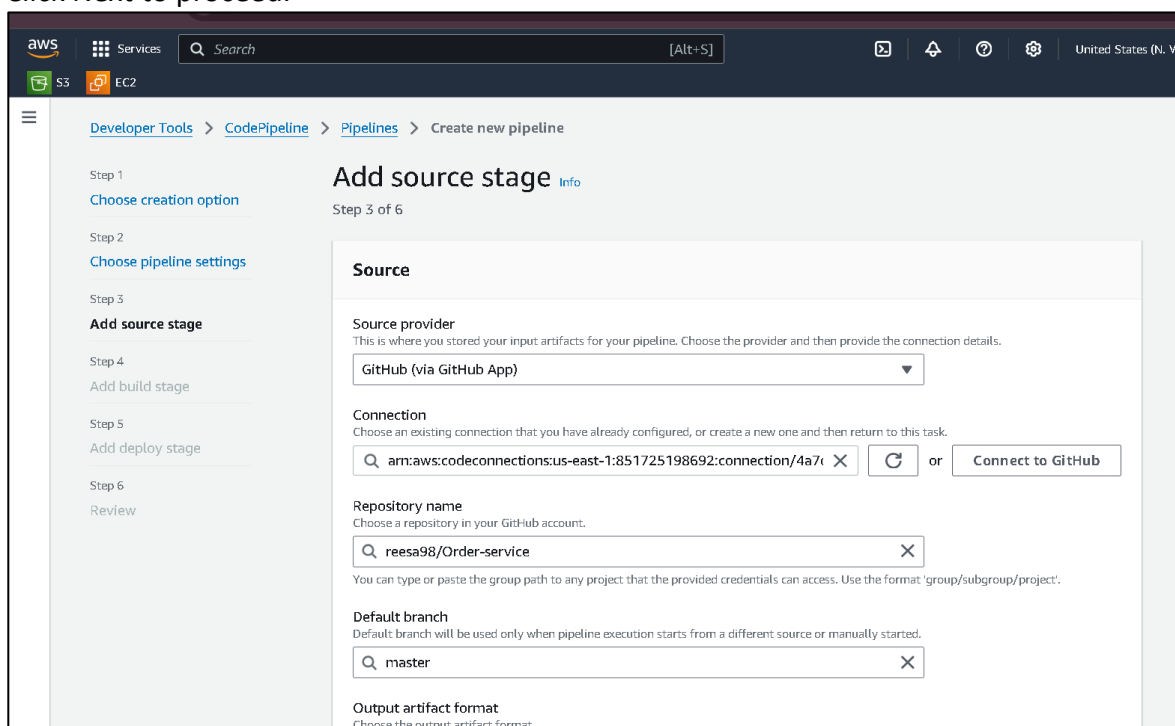
This step involves setting up an AWS CodePipeline to automate the CI/CD workflow. The pipeline integrates stages for source retrieval, build execution, manual approval, and deployment to the target environment.

4.1 Set Up the Pipeline:

- Navigate to the AWS CodePipeline service in the AWS Management Console.
- Click on Create Pipeline.
- Enter a name for the pipeline and select a Service Role (choose an existing role or create a new one).

4.2 Configure Source Stage:

- Select GitHub as the source provider.
- Authenticate your GitHub account and select the repository and branch containing your application code.
- Click Next to proceed.



4.3 Add Build Stage:

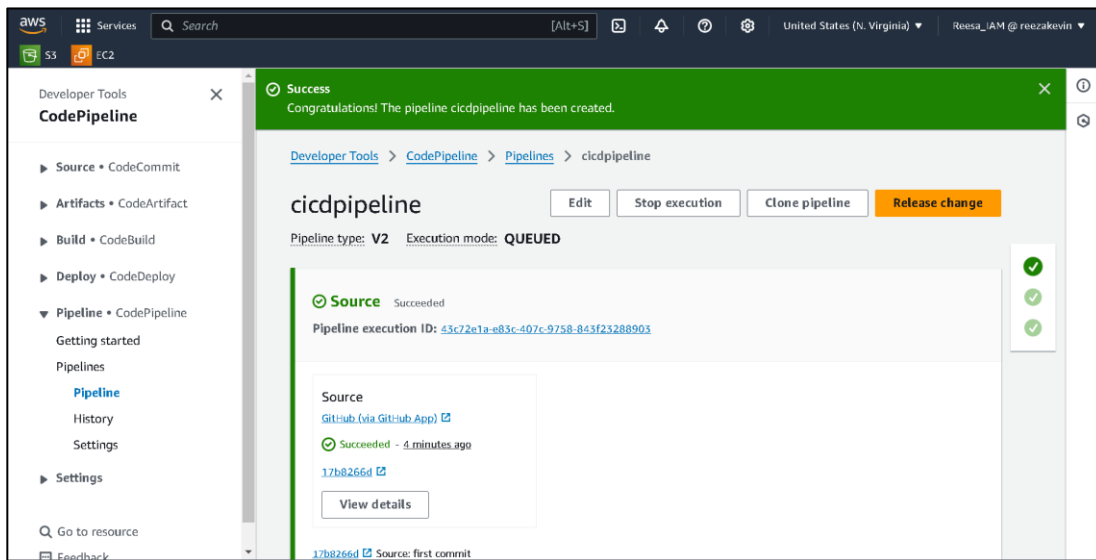
- In the Build Stage, select AWS CodeBuild as the build provider.
- Choose the previously created CodeBuild project.

The screenshot shows the 'Add build stage' screen in the AWS CodePipeline console. The breadcrumb trail is 'Developer Tools > CodePipeline > Pipelines > Create new pipeline'. The left sidebar shows a progress list: Step 1 (Choose creation option), Step 2 (Choose pipeline settings), Step 3 (Add source stage), Step 4 (Add build stage - selected), Step 5 (Add deploy stage), Step 6 (Review). The main content area is titled 'Add build stage' with an 'Info' link and 'Step 4 of 6'. The section is 'Build - optional'. Under 'Build provider', 'Other build providers' is selected. The 'AWS CodeBuild' provider is chosen from a dropdown. The 'Project name' field contains 'cicd-project' with a search icon and a 'Create project' link. Below, there's a section for 'Environment variables - optional' with an 'Add environment variable' button. At the bottom, the 'Build type' section is partially visible.

4.4 Add Deployment Stage:

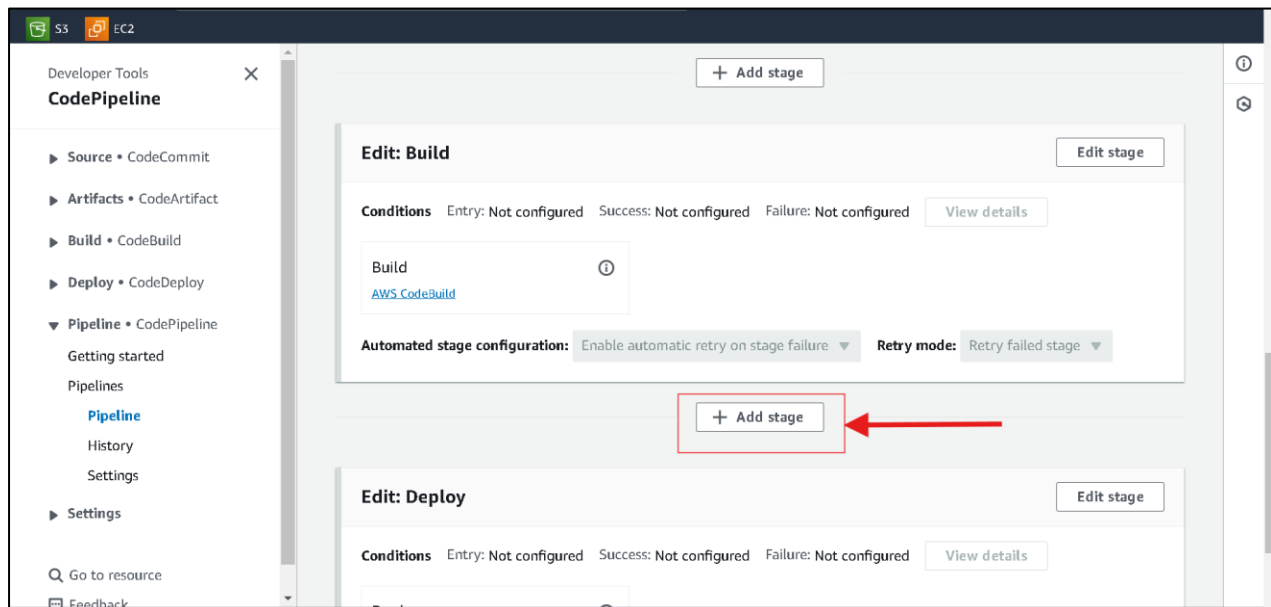
- Select 'AWS Elastic Beanstalk' as the deployment provider.
- Specify the Elastic Beanstalk environment where the application will be deployed.
- Click **Next** to proceed.

The screenshot shows the 'Add deploy stage' screen in the AWS CodePipeline console. The breadcrumb trail is 'Developer Tools > CodePipeline > Pipelines > Create new pipeline'. The left sidebar shows a progress list: Step 1 (Choose creation option), Step 2 (Choose pipeline settings), Step 3 (Add source stage), Step 4 (Add build stage), Step 5 (Add deploy stage - selected), Step 6 (Review). The main content area is titled 'Add deploy stage' with an 'Info' link and 'Step 5 of 6'. The section is 'Deploy - optional'. Under 'Deploy provider', 'AWS Elastic Beanstalk' is selected. The 'Region' dropdown shows 'US East (N. Virginia)'. The 'Input artifacts' section has a dropdown showing 'BuildArtifact' defined by 'Build'. The 'Application name' field contains 'cicdprojectapp'. The 'Environment name' field contains 'Cicdprojectapp-env'. There are checkboxes for 'Configure automatic rollback on stage failure' (checked) and 'Enable automatic retry on stage failure' (unchecked). At the bottom, there are buttons for 'Cancel', 'Previous', 'Skip deploy stage', and 'Next'.



4.5 Add Manual Approval Stage:

- After creating the pipeline, edit it to include a manual approval stage.
- In the **Pipeline Editor**, add a new stage after the **Build Stage**.
- Name the stage **Manual Approval** and add an action for **Approval**.



4.6 Finalize and Save the Pipeline:

- Review the stages and configurations in the pipeline.
- Click Save Pipeline to complete the setup.

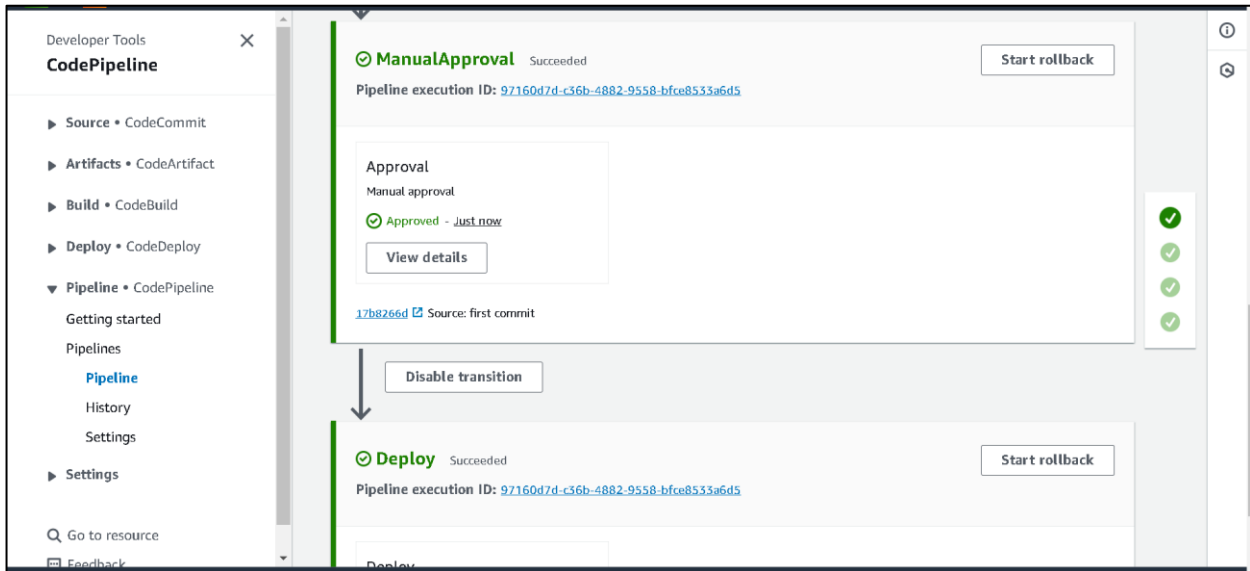
The screenshot shows a dialog box titled "Edit: ManualApproval" with "Cancel", "Delete", and "Done" buttons in the top right. The main area contains three buttons: "Add entry condition", "Add success condition", and "Add failure condition". Below these is a "+ Add action group" button. A central panel displays "Approval" with "Manual approval" below it, an information icon, and edit/delete icons. To the right of this panel is a "+ Add action" button. At the bottom, there is another "+ Add action group" button and a label "Automated stage configuration:" followed by a "None" dropdown menu.

5. Finalize Deployment:

- The pipeline will automatically retrieve the source code, execute the build, and pause after the build stage for manual approval.

The screenshot shows a "Review" dialog box with a close button in the top right. It displays "Action name: Approval" and "Status: Waiting for approval". There are two tabs: "Details" (selected) and "Revisions". Under "Details", the "Trigger" is "StartPipelineExecution - Reesa_IAM" with a link icon. The "Comments about this action" field is empty. The "URL for review" field is also empty. Under "Decision", the "Approve" radio button is selected, with the text "Approving will resume the pipeline execution." below it. The "Reject" radio button is unselected, with the text "Rejecting will stop the pipeline execution with a failed status." below it. At the bottom, there is a "Comments - optional" label, a "Preview markdown" toggle (which is turned off), a "Learn more" link, and a text input field. The bottom right corner has "Cancel" and "Submit" buttons.

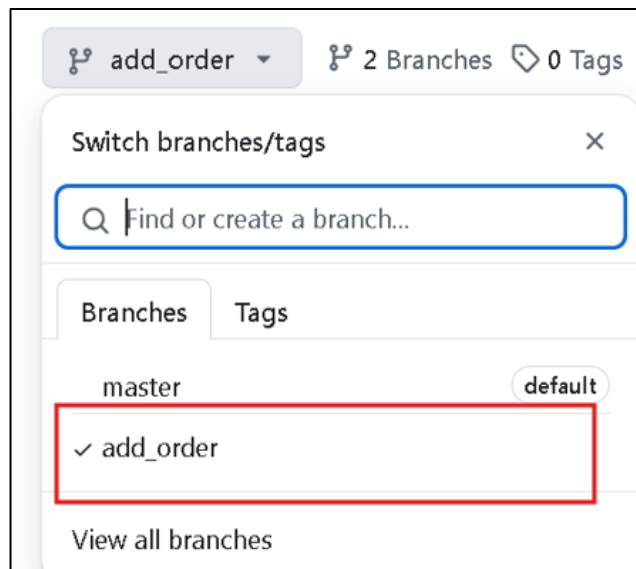
- Once the approval is provided, the pipeline will deploy the application to the specified Elastic Beanstalk environment.



6. Testing Changes: Updating Local Repository and Pushing to Remote

This section describes how to test changes by creating a new branch in your GitHub repository, modifying files locally, and merging the updates into the main branch to trigger the CI/CD pipeline.

1. Create a branch in your GitHub repository



2. Check Current Working Branch

- Use the following command to verify the current branch in your local repository

```
git branch
```

3. Pull a New Branch from the Remote Repository

- To fetch a new branch (e.g., add_order) from the remote repository, use

```
git pull
```

4. Switch to the New Branch

- Change your working branch to the newly fetched branch (e.g., add_order):

```
git checkout add_order
```

5. Fetch All Changes

- Ensure your local branch is up to date with the remote repository:

```
git pull
```

6. Make Changes Locally

- Open the required file (e.g., DAO.java) in your local repository.
- Implement the necessary changes (e.g., adding new orders in the DAO file).
- Save the file after making changes.

7. Stage the Changes

- Add the modified files to the staging area

```
git add .
```

8. Commit the Changes

- Commit the changes with an appropriate message

```
git commit -m "Made changes in the DAO file"
```

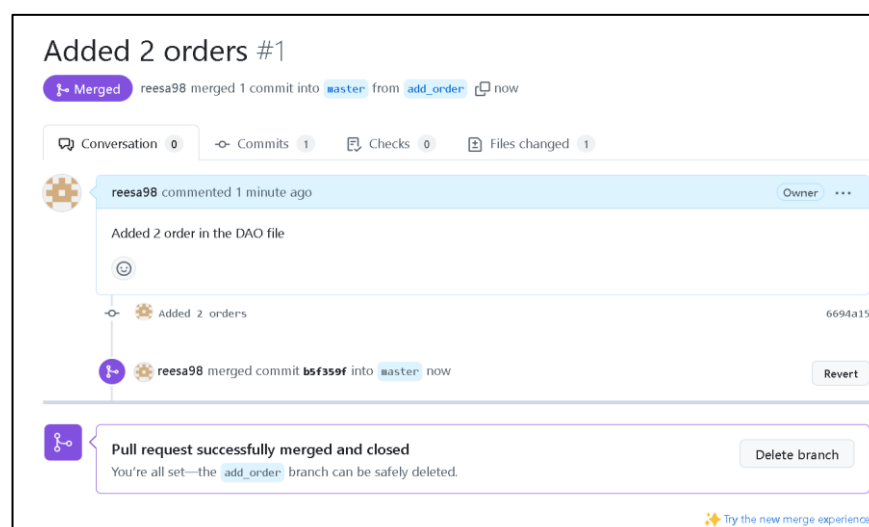
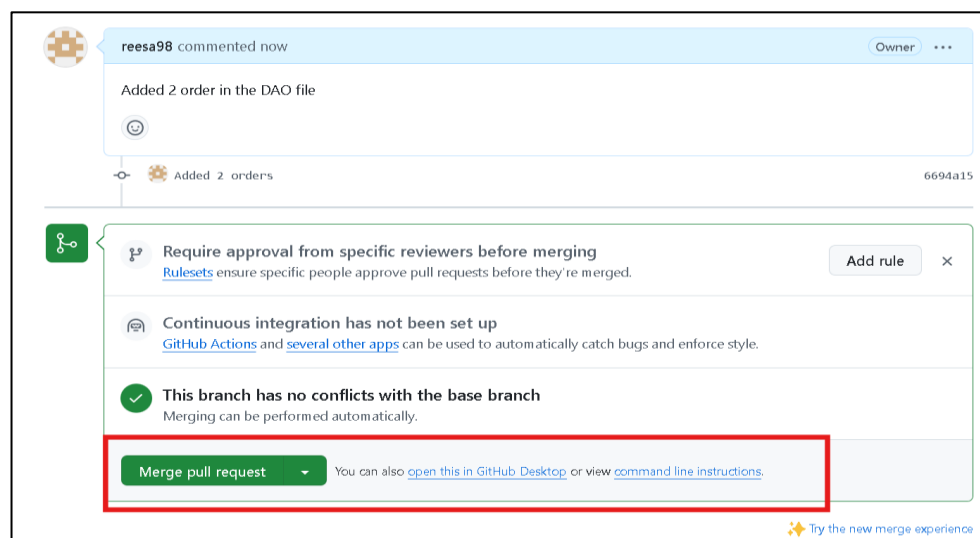
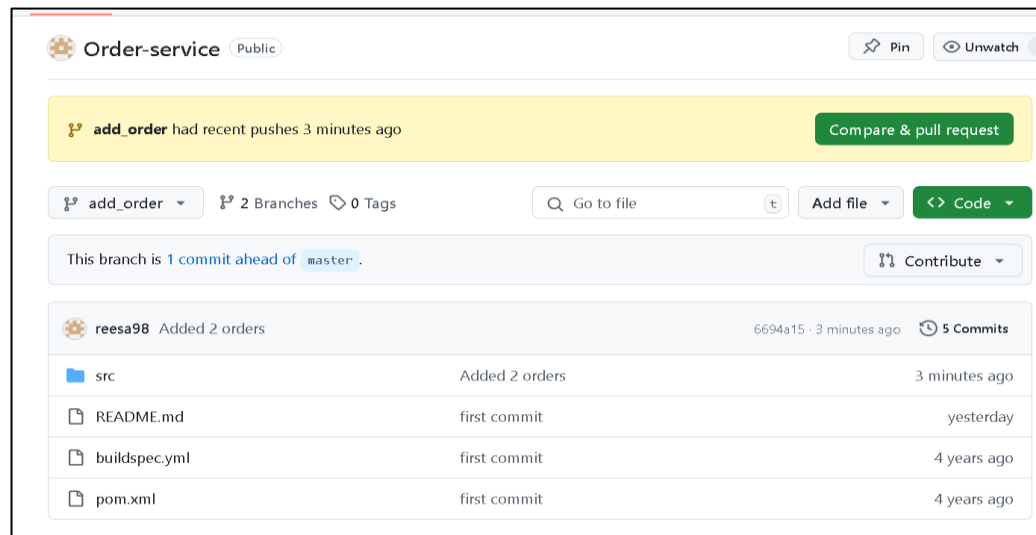
9. Push Changes to the Remote Branch

- Push the changes from your local branch to the corresponding remote branch:

```
git push origin add-order
```

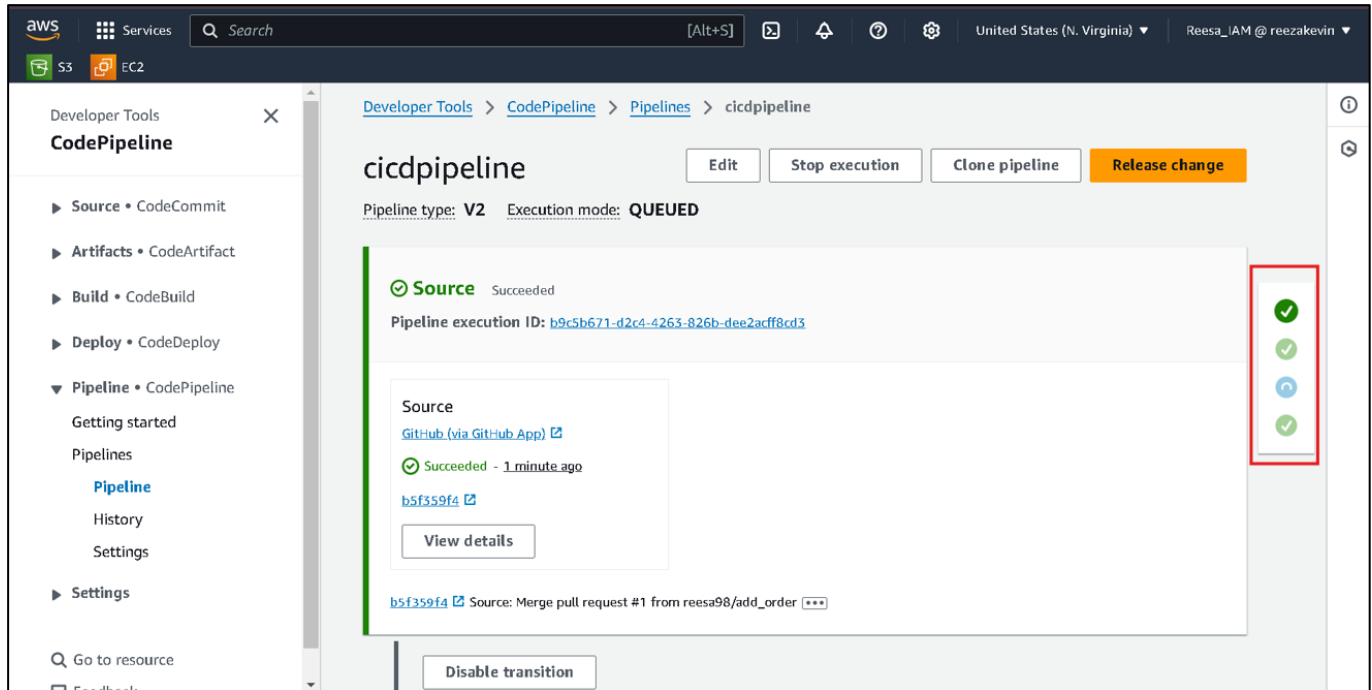
10. Merge the Branch to Main

- In the GitHub repository, initiate a pull request to merge the changes from the add-order branch to the main branch.
- Review the changes and complete the pull request to merge.



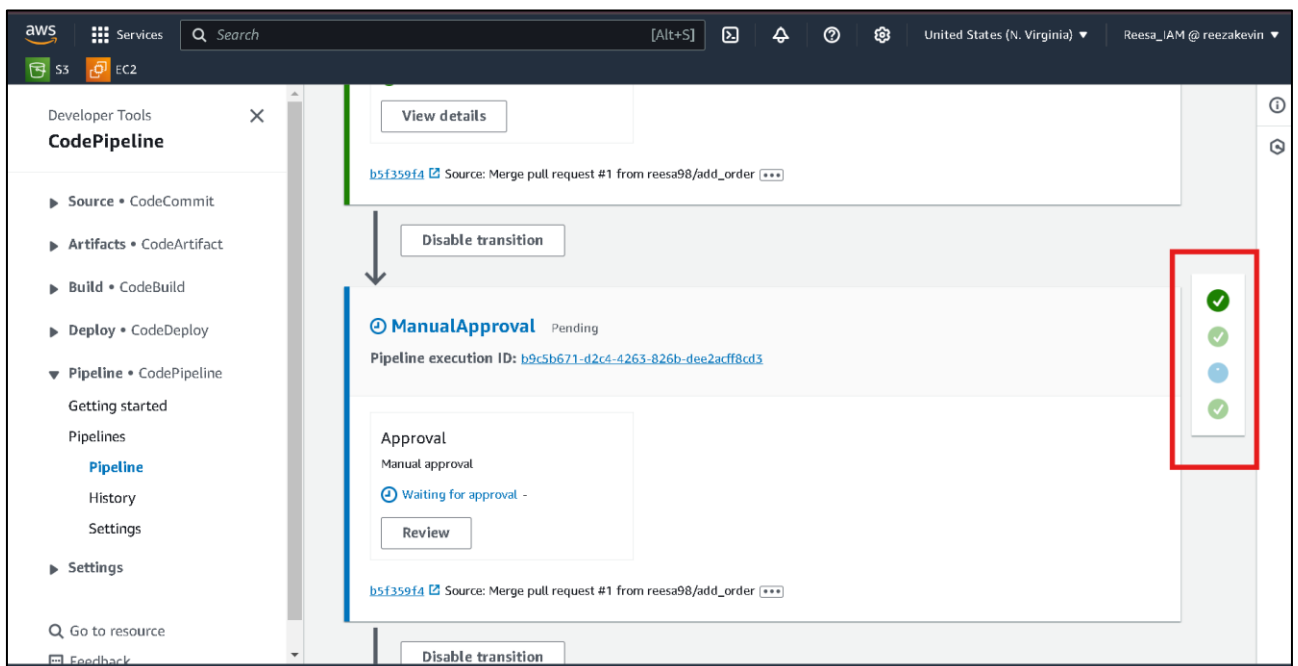
11. Pipeline Triggered and Build Started

- The screenshot demonstrates that the CI/CD pipeline was successfully triggered after merging the changes into the main branch. It shows the build stage being initiated automatically as part of the pipeline workflow.



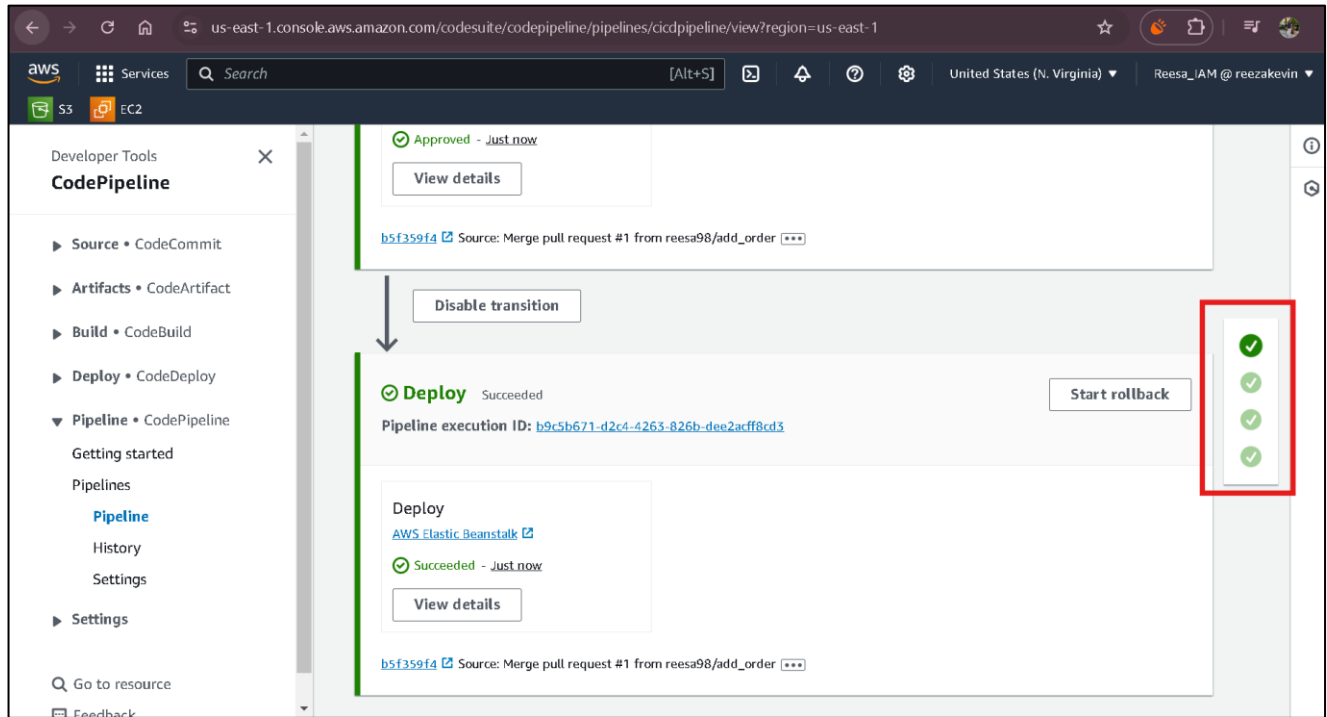
12. Manual Approval Stage

- This screenshot captures the pipeline pausing at the manual approval stage, awaiting the manager's confirmation before proceeding with the deployment to the production environment.



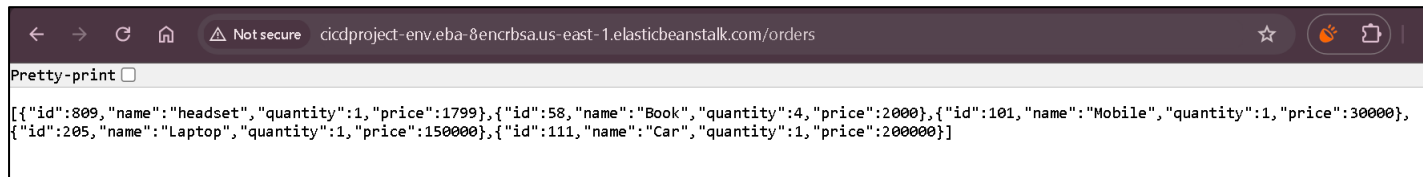
13. Deployment Successful

- The final screenshot confirms a successful deployment of the updated application after the manual approval was provided, showcasing the end-to-end execution of the pipeline.

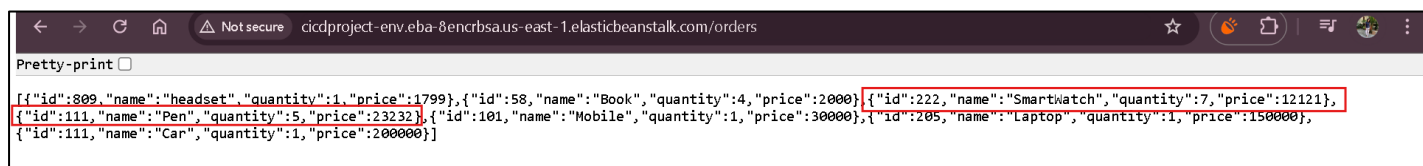


14. Order Details (Before and After)

- Two screenshots display the before and after states of the orders in the application.
 - The **before** image reflects the state of the orders before the code change was implemented.



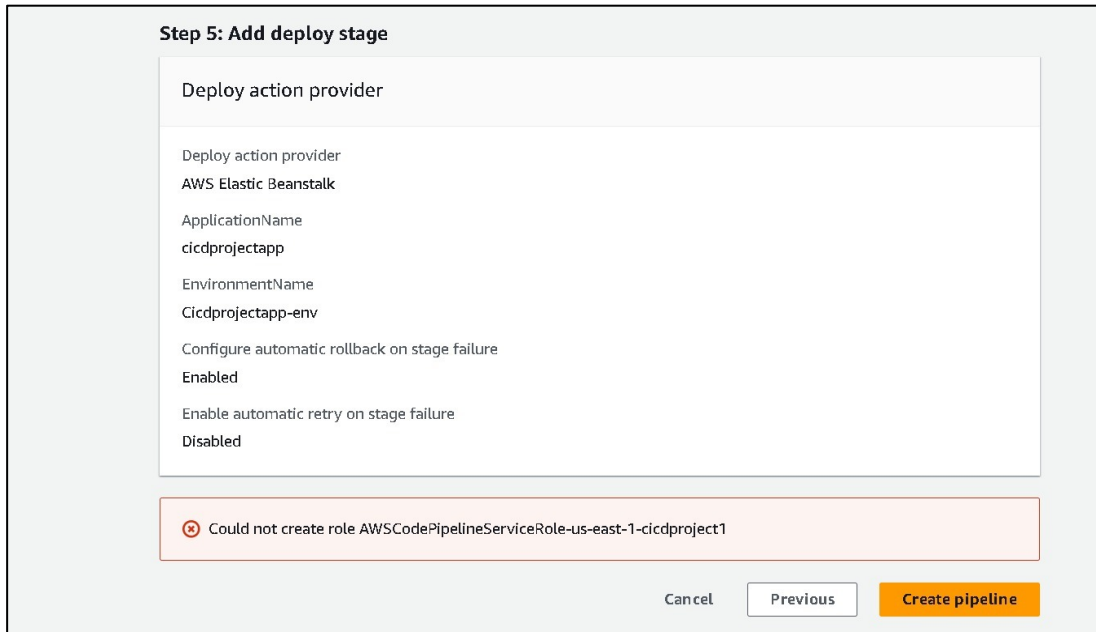
- The **after** image highlights the successful addition of new orders, demonstrating the impact of the changes made to the DAO file and validated through the CI/CD pipeline.



7. Troubleshooting IAM Permissions for Code Pipeline Role Creation

Error Encountered

- While creating an AWS CodePipeline, encountered the error:



The screenshot shows the 'Step 5: Add deploy stage' configuration window in the AWS CodePipeline console. The 'Deploy action provider' is set to 'AWS Elastic Beanstalk'. The 'ApplicationName' is 'cicdprojectapp' and the 'EnvironmentName' is 'Cicdprojectapp-env'. The 'Configure automatic rollback on stage failure' option is 'Enabled', and the 'Enable automatic retry on stage failure' option is 'Disabled'. A red error message at the bottom states: 'Could not create role AWSCodePipelineServiceRole-us-east-1-cicdproject1'. At the bottom right, there are three buttons: 'Cancel', 'Previous', and 'Create pipeline'.

- To resolve the error by attaching a custom IAM policy to the user, allowing the creation of the required role for CodePipeline

Steps to Resolve

Step 1: Create a Custom IAM Policy

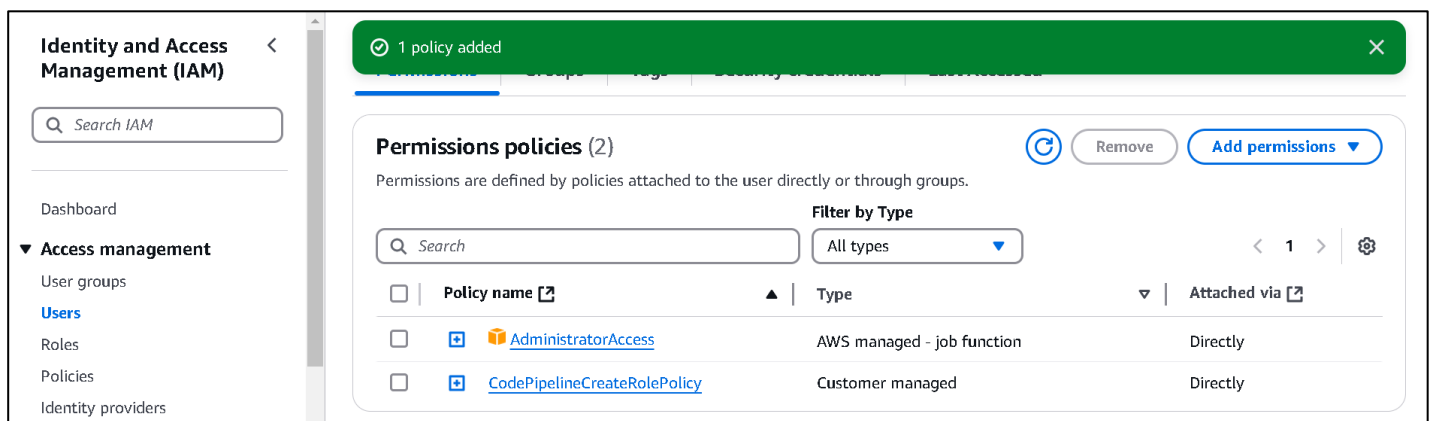
- Log in to AWS Management Console.
- Navigate to IAM from the AWS services dashboard.
- Click Policies on the left menu and choose Create Policy.
- Select the JSON tab and paste the following policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Statement1",
      "Effect": "Allow",
      "Action": "iam:CreateRole",
      "Resource": "arn:aws:iam::*:role/AWSCodePipeline*"
    }
  ]
}
```

- Click Next: Tags (optional) > Next: Review.
- Provide a name, “CodePipelineCreateRolePolicy”, and click Create Policy.

Step 2: Attach the Policy to the IAM User

- Navigate back to the IAM Dashboard.
- Select **Users** from the left menu and choose the user requiring the permission.
- Under the **Permissions** tab, click **Add Permissions > Attach policies** directly.
- Search for and select CodePipelineCreateRolePolicy.
- Click **Next: Review > Add Permissions**.



Step 3: Retry the AWS CodePipeline Creation

- Return to the AWS CodePipeline setup process.
- Retry creating the pipeline. The error should now be resolved, and the necessary role (AWSCodePipelineServiceRole) will be created automatically.

8. Conclusion

The implementation of the CI/CD pipeline represents a significant advancement in streamlining Ample Technologies' software development and deployment processes. By leveraging AWS services and adhering to best practices, the pipeline ensures an efficient, automated, and reliable workflow. It incorporates critical features like automated builds, continuous testing, and a manual approval step to maintain high code quality and secure production rollouts.

Extensive testing was performed to validate the pipeline, including code changes, branch management, and deployment scenarios, ensuring the system's robustness and reliability. This Proof of Concept not only highlights the value of CI/CD pipelines in modern software development but also provides a solid foundation for scaling and optimizing future deployments.