# AWS Task-5

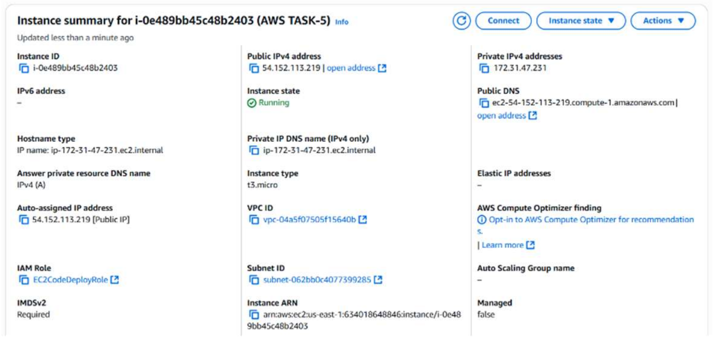
## TASKS

### Work Flow:

- Create an EC2 instance with the help of AWS Management Console with linux OS of required configuration.
- Now, Connect an EC2 instance with an help of  Windows Terminal or Gitbash or Vbox.
- To connect an EC2 instance the command is:
    - ssh -i "**key_file**" ec2-user@"**Public_IP_address**"

**Key_file** --- Key file of the instance with the extension .pem

**Public_IP_address ---** Public IP address of the instance.



Instance summary for i-0e489bb45c48b2403 (AWS TASK-5)

Updated less than a minute ago

| | | |
|---|---|---|
| Instance ID<br>i-0e489bb45c48b2403 | Public IPv4 address<br>54.152.113.219 \| open address | Private IPv4 addresses<br>172.31.47.231 |
| IPv6 address<br>– | Instance state<br>Running | Public DNS<br>ec2-54-152-113-219.compute-1.amazonaws.com \| open address |
| Hostname type<br>IP name: ip-172-31-47-231.ec2.internal | Private IP DNS name (IPv4 only)<br>ip-172-31-47-231.ec2.internal | |
| Answer private resource DNS name<br>IPv4 (A) | Instance type<br>t3.micro | Elastic IP addresses<br>– |
| Auto-assigned IP address<br>54.152.113.219 [Public IP] | VPC ID<br>vpc-04a5f07505f15640b | AWS Compute Optimizer finding<br>Opt-in to AWS Compute Optimizer for recommendations. \| Learn more |
| IAM Role<br>EC2CodeDeployRole | Subnet ID<br>subnet-062bb0c4077399285 | Auto Scaling Group name<br>– |
| IMDSv2<br>Required | Instance ARN<br>arn:aws:ec2:us-east-1:634018648846:instance/i-0e489bb45c48b2403 | Managed<br>false |

## 1. Deploy a simple web application using AWS code commit, code build and deploy & access via browser and automate via codepipeline.

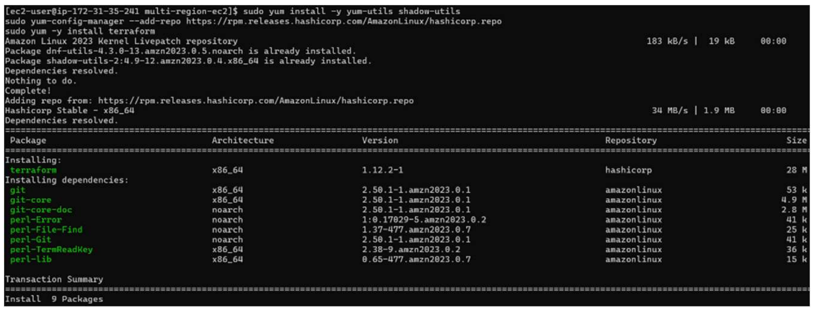
### Step 1: Create a CodeCommit Repository

### Install An Terraform :

- ✓ In this step, we created a new Git-based repository using AWS CodeCommit to store our application source code.
- ✓ We navigated to the CodeCommit service in the AWS Management Console and created a repository named **MyWebAppRepo.**
- ✓ After the repository was created, we cloned it to our local machine using the HTTPS Git URL provided by AWS.
- ✓ This allowed us to add, update, and manage our application's source code locally and push changes directly to the CodeCommit repository.
- ✓ Clone the repo locally using Git:

- **git clone https://git codecommit.<region>.amazonaws.com/v1/repos/MyWebAppRepo**

✓ We created a new AWS CodeCommit repository named MyWebAppRepo. This repository will store our web application source code and deployment configuration.



```
[ec2-user@ip-172-31-35-241 multi-region-ec2]$ sudo yum install -y yum-utils shadow-utils
sudo yum-config-manager --add-repo https://rpm.releases.hashicorp.com/AmazonLinux/hashicorp.repo
sudo yum -y install terraform
Amazon Linux 2023 Kernel Livepatch repository                                   183 kB/s |  19 kB     00:00
Package dnf-utils-4.3.0-13.amzn2023.0.5.noarch is already installed.
Package shadow-utils-2:4.9-12.amzn2023.0.4.x86_64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
Adding repo from: https://rpm.releases.hashicorp.com/AmazonLinux/hashicorp.repo
Hashicorp Stable - x86_64                                                        34 MB/s | 1.9 MB     00:00
Dependencies resolved.
===========================================================================================================
 Package                  Architecture       Version                          Repository              Size
===========================================================================================================
Installing:
 terraform                x86_64             1.12.2-1                         hashicorp               28 M
Installing dependencies:
 git                      x86_64             2.50.1-1.amzn2023.0.1            amazonlinux             53 k
 git-core                 x86_64             2.50.1-1.amzn2023.0.1            amazonlinux            4.9 M
 git-core-doc             noarch             2.50.1-1.amzn2023.0.1            amazonlinux            2.8 M
 perl-Error               noarch             1:0.17029-5.amzn2023.0.2         amazonlinux             41 k
 perl-File-Find           noarch             1.37-477.amzn2023.0.7            amazonlinux             25 k
 perl-Git                 noarch             2.50.1-1.amzn2023.0.1            amazonlinux             41 k
 perl-TermReadKey         x86_64             2.38-9.amzn2023.0.2              amazonlinux             36 k
 perl-lib                 x86_64             0.65-477.amzn2023.0.7            amazonlinux             15 k

Transaction Summary
===========================================================================================================
Install  9 Packages
```

## Step 2: Add Your Application Code.

✓ We added our simple web application files to the local CodeCommit repository directory.

✓ To define the build instructions for AWS CodeBuild, we created a file named buildspec.yml in the project root.

✓ This file includes the basic phases like install and build, and specifies that all files should be included in the output artifact.

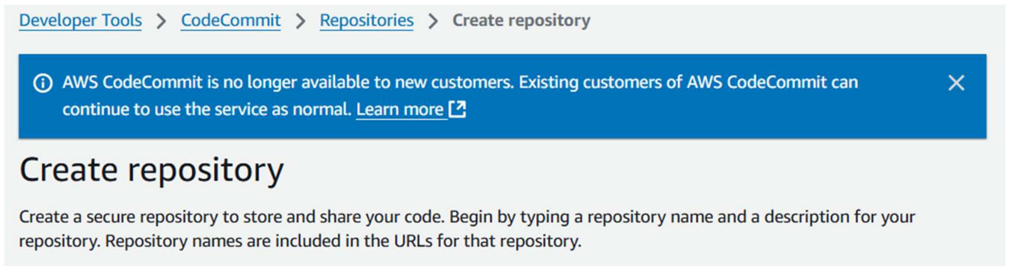
✓ After setting up the files, we committed and pushed the code to the CodeCommit repository using Git commands

✓ This completed the source code setup needed for the CI/CD pipeline.

✓ Added our application files to the CodeCommit repo. Also included a buildspec.yml file for AWS CodeBuild to understand the build process.

✓ **CodeCommit is not available for your AWS account (common for new customers), using GitHub as the source repository is absolutely fine — and AWS CodePipeline supports it directly.**

✓ **In this case , Let's proceed with your setup using GitHub as the source.**

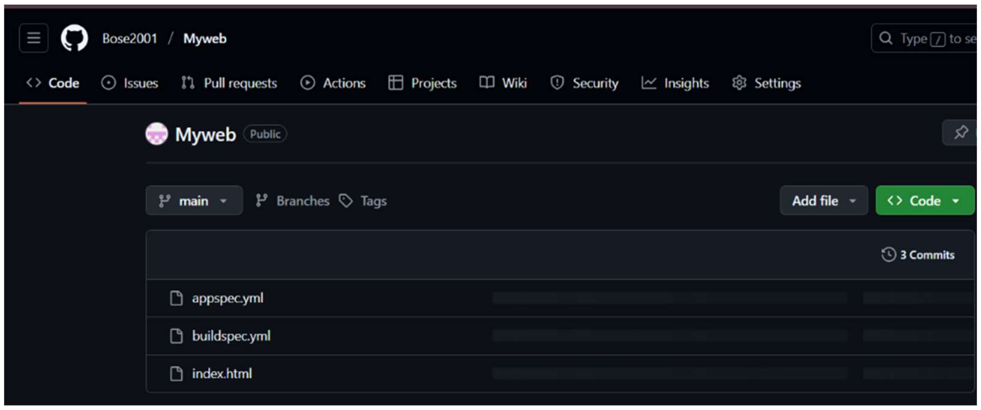


Developer Tools > CodeCommit > Repositories > Create repository

ⓘ AWS CodeCommit is no longer available to new customers. Existing customers of AWS CodeCommit can continue to use the service as normal. Learn more

Create repository

Create a secure repository to store and share your code. Begin by typing a repository name and a description for your repository. Repository names are included in the URLs for that repository.


Bose2001 / Myweb

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

Myweb Public

main Branches Tags Add file Code

3 Commits

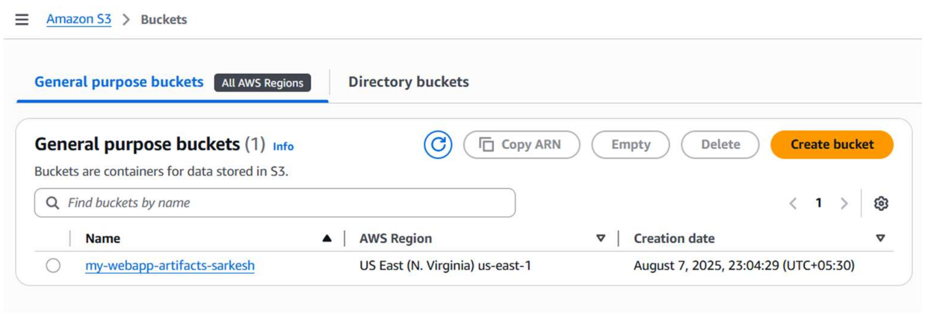
appspec.yml

buildspec.yml

index.html

## Step 3: Create an S3 Bucket for CodeDeploy Artifacts

- ✓ To manage the deployment process efficiently, we created an Amazon S3 bucket that will be used to store the build artifacts generated by AWS CodeBuild.
- ✓ These artifacts are packaged application files that AWS CodeDeploy will use to deploy the application to the target EC2 instances.
- ✓ This ensures a centralized and secure location for managing application versions and makes the deployment process smoother and more automated.
- ✓ We created an S3 bucket to store deployment artifacts generated by CodeBuild.


Amazon S3 > Buckets

General purpose buckets (All AWS Regions) | Directory buckets

General purpose buckets (1) Info

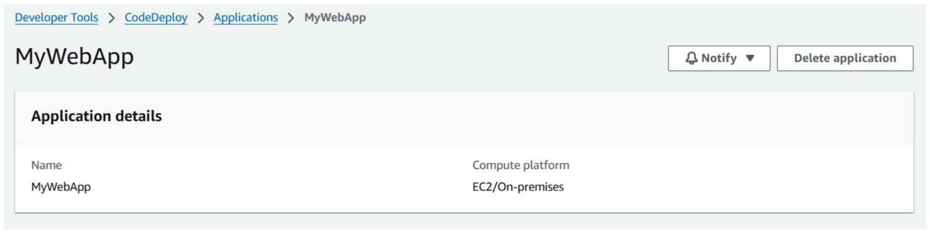
Buckets are containers for data stored in S3.

| Name | AWS Region | Creation date |
|---|---|---|
| my-webapp-artifacts-sarkesh | US East (N. Virginia) us-east-1 | August 7, 2025, 23:04:29 (UTC+05:30) |

## Step 4: Create a CodeDeploy Application and Deployment Group.

- ✓ In this step, we set up AWS CodeDeploy to handle the deployment of our application to an EC2 instance.
- ✓ First, we created a new CodeDeploy application named MyWebAppDeploy and selected EC2/On-premises as the compute platform.
- ✓ Next, we created a **deployment group** called **MyWebAppGroup**, where we specified the target EC2 instance for deployment.
- ✓ We ensured the instance was tagged correctly or selected manually.
- ✓ The deployment type was set to **in-place**, which updates the application directly on the running EC2 instance.
- ✓ Additionally, we verified that the **CodeDeploy agent** was installed and running on the EC2 instance, which is required for the deployment to work properly.

- ✓ Created a CodeDeploy application and a deployment group that points to our target EC2 instance where the app will be deployed.

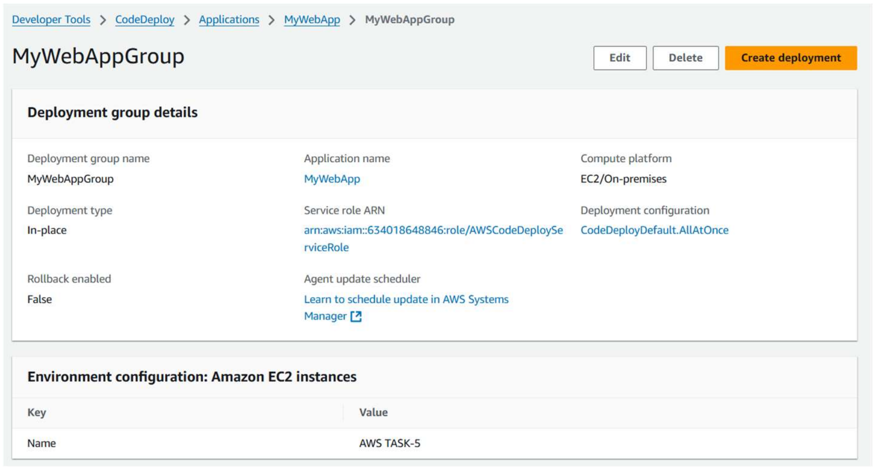


Developer Tools > CodeDeploy > Applications > MyWebApp

MyWebApp

**Application details**

| Name | Compute platform |
|---|---|
| MyWebApp | EC2/On-premises |



Developer Tools > CodeDeploy > Applications > MyWebApp > MyWebAppGroup

MyWebAppGroup

**Deployment group details**

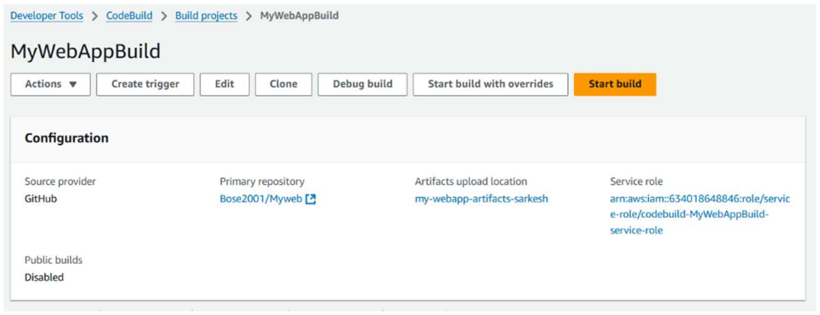
Deployment group name: MyWebAppGroup
Application name: MyWebApp
Compute platform: EC2/On-premises
Deployment type: In-place
Service role ARN: arn:aws:iam::634018648846:role/AWSCodeDeployServiceRole
Deployment configuration: CodeDeployDefault.AllAtOnce
Rollback enabled: False
Agent update scheduler: Learn to schedule update in AWS Systems Manager

**Environment configuration: Amazon EC2 instances**

| Key | Value |
|---|---|
| Name | AWS TASK-5 |

## Step 5: Create a CodeBuild Project.

- ✓ We created a new AWS CodeBuild project named MyWebAppBuild. In this project, we selected CodeCommit as the source provider and chose the repository MyWebAppRepo that contains our application code.
- ✓ For the build environment, we used a managed image with Ubuntu as the operating system, and selected the latest standard runtime environment provided by AWS.
- ✓ We instructed CodeBuild to use the buildspec.yml file already present in our repository, which defines the steps needed to build the application.
- ✓ Finally, we configured the build artifacts to be stored in our previously created S3 bucket named my-webapp-deploy-bucket. This allows the build output to be used in the next stage of the deployment process.
- ✓ Configured CodeBuild to automatically build the application and package the output for deployment. It reads instructions from the buildspec.yml file.



Developer Tools > CodeBuild > Build projects > MyWebAppBuild

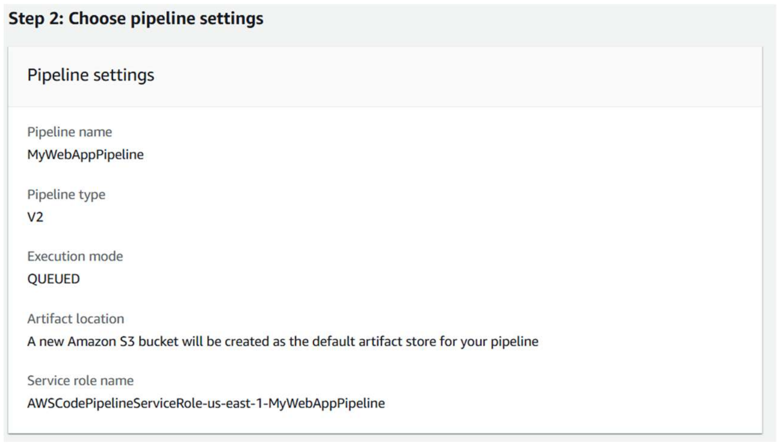
MyWebAppBuild

**Configuration**

Source provider: GitHub
Primary repository: Bose2001/Myweb
Artifacts upload location: my-webapp-artifacts-sarkesh
Service role: arn:aws:iam::634018648846:role/service-role/codebuild-MyWebAppBuild-service-role
Public builds: Disabled

## Step 6: Create a CodePipeline

- ✓ In this step, we created a new CodePipeline named MyWebAppPipeline to automate the entire CI/CD process.
- ✓ We selected **AWS CodeCommit** as the source provider and linked it to our repository **MyWebAppRepo**. This ensures that any code pushed to the repository automatically triggers the pipeline.
- ✓ For the build stage, we selected **CodeBuild** and used the build project **MyWebAppBuild**, which compiles and packages our application code according to the instructions defined in the buildspec.yml file.
- ✓ For the deploy stage, we chose **AWS CodeDeploy** and configured it to use the application **MyWebAppDeploy** and deployment group **MyWebAppGroup**, which is linked to our EC2 instance.
- ✓ Once the pipeline was created, it connected all the services—CodeCommit, CodeBuild, and CodeDeploy—allowing the application to be built and deployed automatically each time a change is pushed to the repository.


**Step 2: Choose pipeline settings**

Pipeline settings

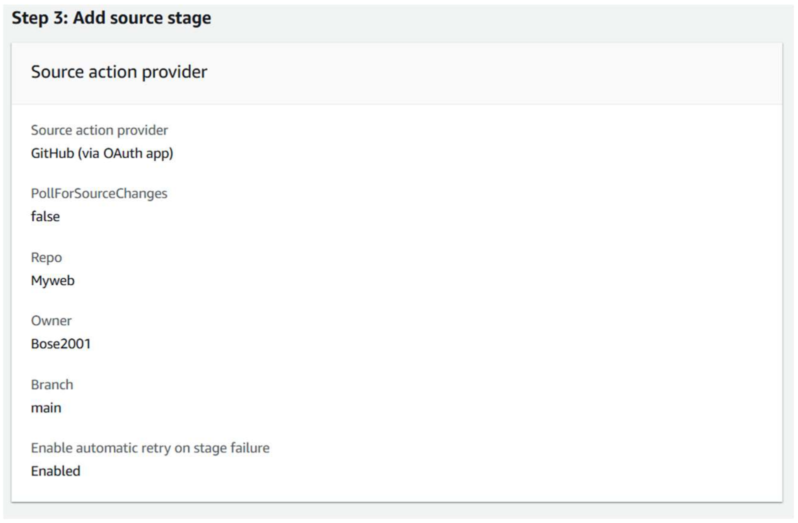
Pipeline name
MyWebAppPipeline

Pipeline type
V2

Execution mode
QUEUED

Artifact location
A new Amazon S3 bucket will be created as the default artifact store for your pipeline

Service role name
AWSCodePipelineServiceRole-us-east-1-MyWebAppPipeline


**Step 3: Add source stage**

Source action provider

Source action provider
GitHub (via OAuth app)

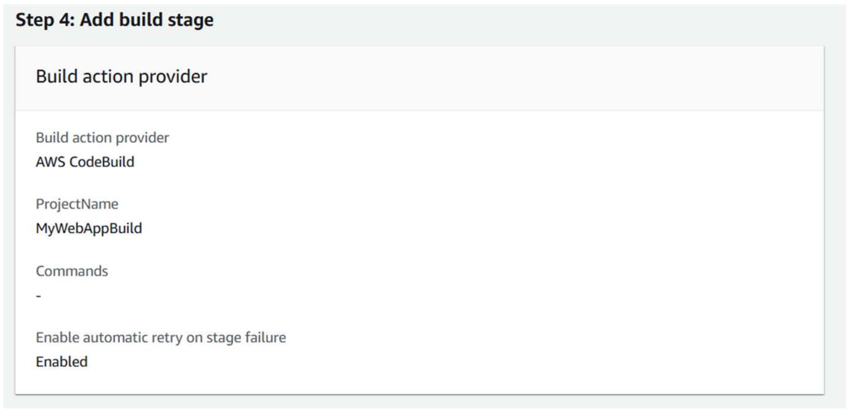
PollForSourceChanges
false

Repo
Myweb

Owner
Bose2001

Branch
main

Enable automatic retry on stage failure
Enabled



**Step 4: Add build stage**

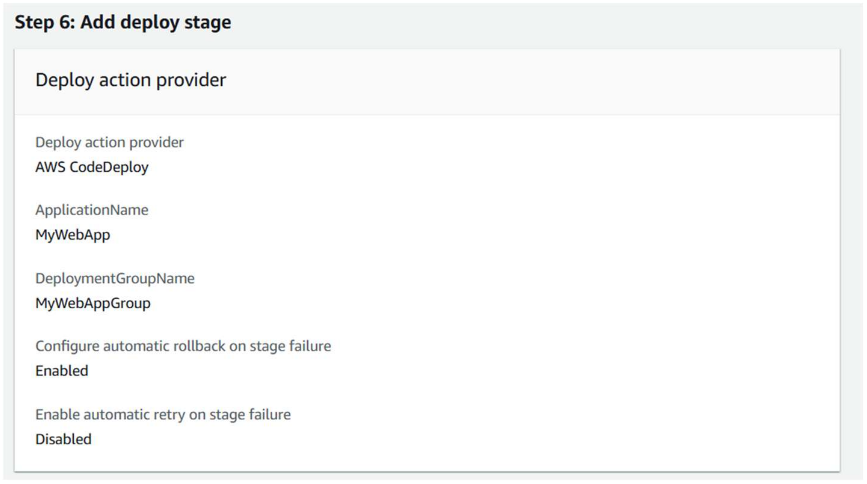
Build action provider

Build action provider
AWS CodeBuild

ProjectName
MyWebAppBuild

Commands
-

Enable automatic retry on stage failure
Enabled



**Step 6: Add deploy stage**

Deploy action provider

Deploy action provider
AWS CodeDeploy

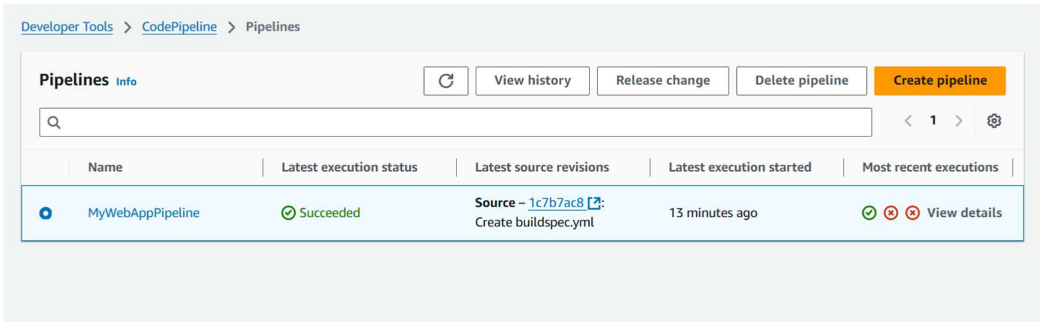
ApplicationName
MyWebApp

DeploymentGroupName
MyWebAppGroup

Configure automatic rollback on stage failure
Enabled

Enable automatic retry on stage failure
Disabled

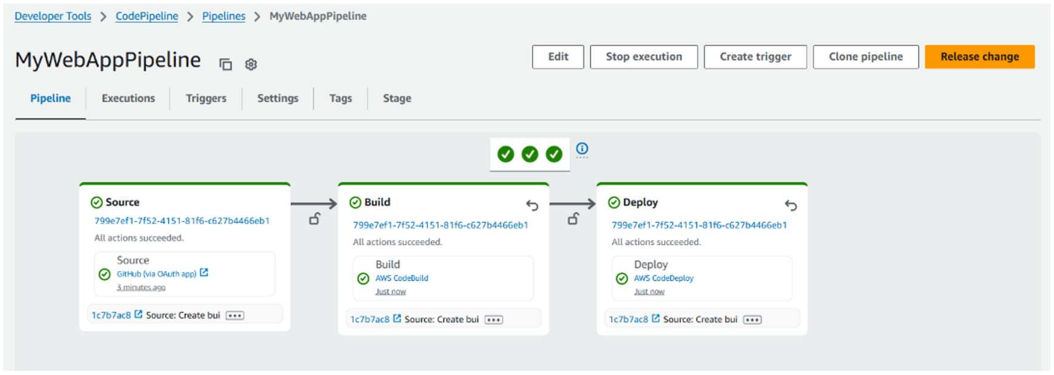
✓ Set up a CI/CD pipeline using CodePipeline. The pipeline automatically fetches code from CodeCommit, builds using CodeBuild, and deploys to EC2 using CodeDeploy.



Developer Tools > CodePipeline > Pipelines

**Pipelines** Info

| | Name | Latest execution status | Latest source revisions | Latest execution started | Most recent executions |
|---|---|---|---|---|---|
| ● | MyWebAppPipeline | Succeeded | Source – 1c7b7ac8: Create buildspec.yml | 13 minutes ago | ✓ ✗ ✗ View details |



Developer Tools > CodePipeline > Pipelines > MyWebAppPipeline

MyWebAppPipeline

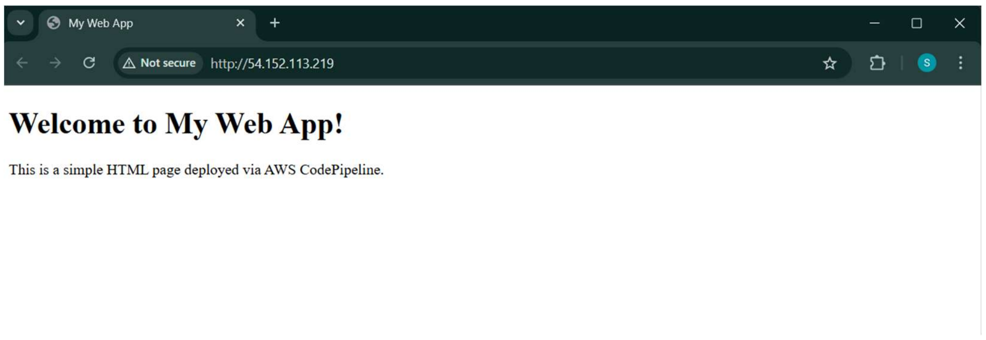
Source
799e7ef1-7f52-4151-81f6-c627b4466eb1
All actions succeeded.
Source
GitHub (via OAuth app)
3 minutes ago
1c7b7ac8 Source: Create bui

Build
799e7ef1-7f52-4151-81f6-c627b4466eb1
All actions succeeded.
Build
AWS CodeBuild
Just now
1c7b7ac8 Source: Create bui

Deploy
799e7ef1-7f52-4151-81f6-c627b4466eb1
All actions succeeded.
Deploy
AWS CodeDeploy
Just now
1c7b7ac8 Source: Create bui

## Step 7: Deploy and Access the Application via Browser.

- ✓ After completing the pipeline setup, we tested the deployment process. We made a small change in our application code and pushed it to the AWS CodeCommit repository using Git. This action automatically triggered the CodePipeline.
- ✓ The pipeline then executed all defined stages:
    - **Source stage** fetched the latest code from CodeCommit.
    - **Build stage** used CodeBuild to package the application as per the buildspec.yml instructions.
    - **Deploy stage** used CodeDeploy to install the application on the target EC2 instance defined in the deployment group.
- ✓ Once the deployment completed successfully, we accessed the deployed application by opening the **public IP address of the EC2 instance** in a web browser.
- ✓ The application loaded correctly, confirming that the end-to-end CI/CD pipeline worked as expected.
- ✓ After pushing a code change, CodePipeline automatically executed the entire CI/CD flow and deployed the app to EC2. We were able to access the application using the EC2 public IP via browser.



Welcome to My Web App!

This is a simple HTML page deployed via AWS CodePipeline.

## Conclusion

- ✓ In this task, we implemented a complete CI/CD pipeline on AWS using CodeCommit, CodeBuild, CodeDeploy, and CodePipeline. This setup ensures every code change gets automatically built, tested, and deployed to production without manual intervention. It simplifies the release process and ensures faster and reliable deployments.

************** **TASK COMPLETED** **************