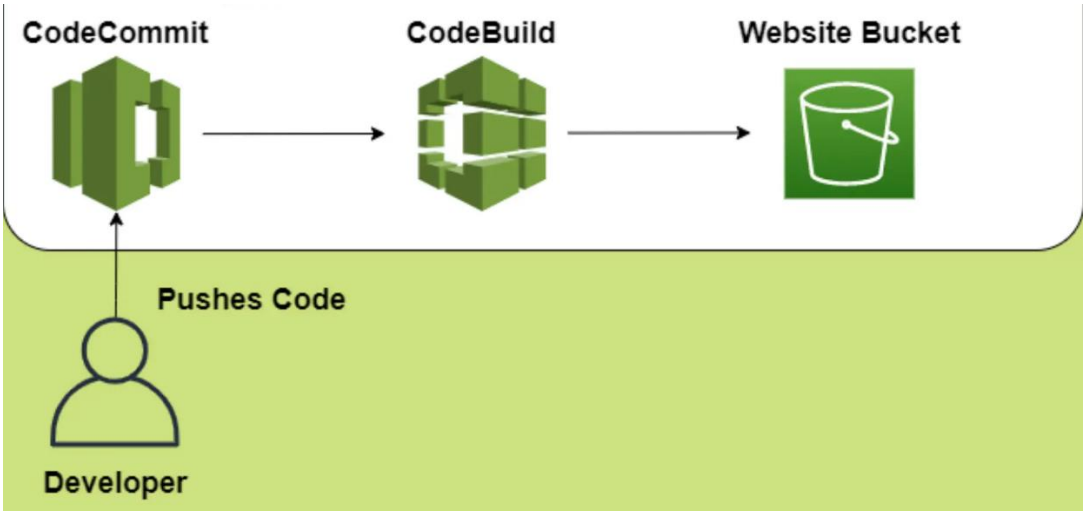
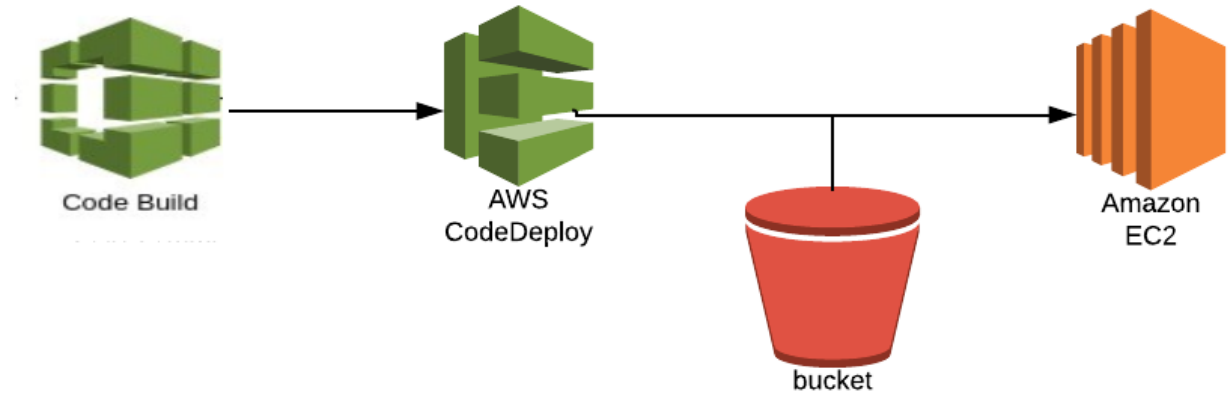
Step-by-Step Guide to DevOps Implementation on AWS – Jasvitha Buggana

Contents

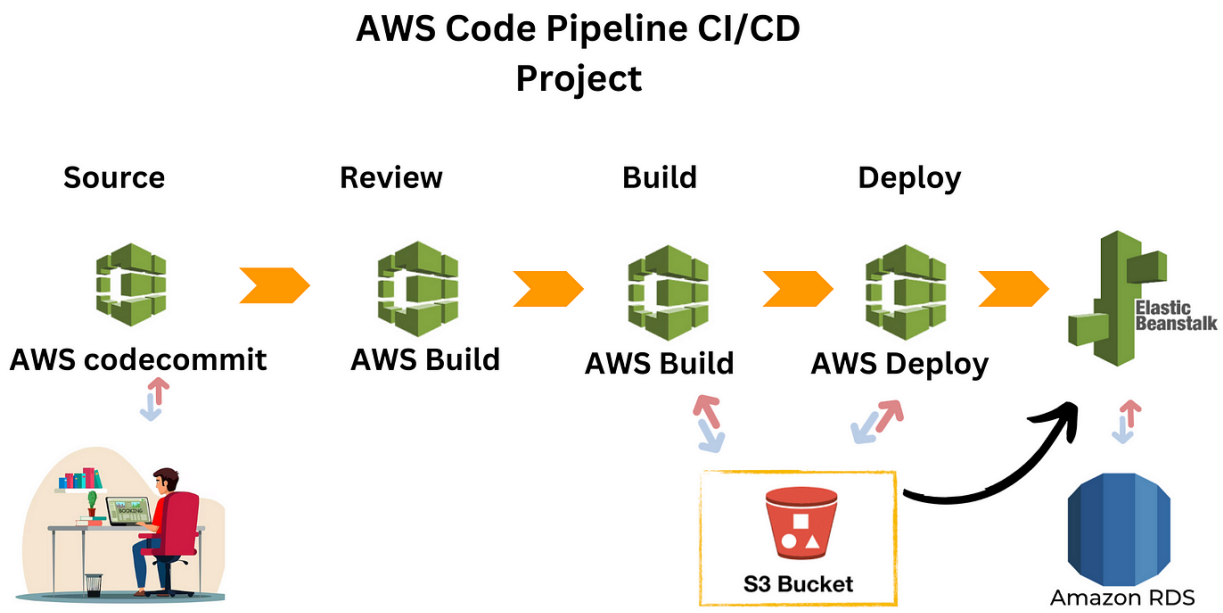
Implementing DevOps Practices with AWS Infrastructure



Project flow: A developer pushes code to a website stored in an **AWS CodeCommit** repository. Subsequently, **AWS CodeBuild** automatically builds the code and deploys it to an S3 bucket.



In the second stage, the code built and tested in **CodeBuild** is uploaded to an Amazon S3 bucket, which can be configured to function as a website. This means that users can access the website by visiting the S3 bucket's URL.

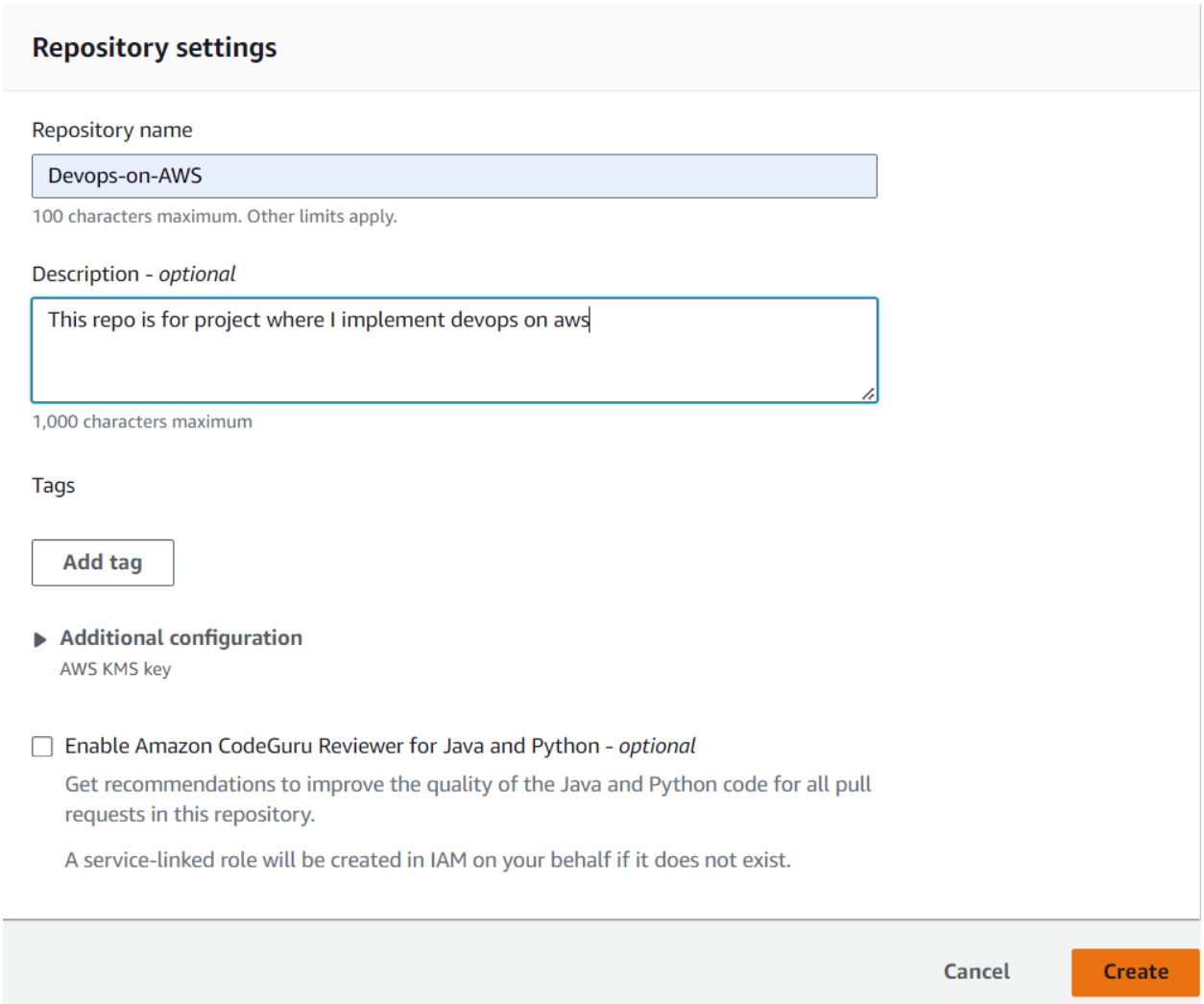


After the code was built and tested in **CodeBuild**.This diagram shows how the built code is shipped to production. **AWS CodeDeploy** takes the tested code from CodeBuild and deploys it to an Elastic Beanstalk environment. Elastic Beanstalk is a service that lets you easily deploy and manage web applications in the cloud.

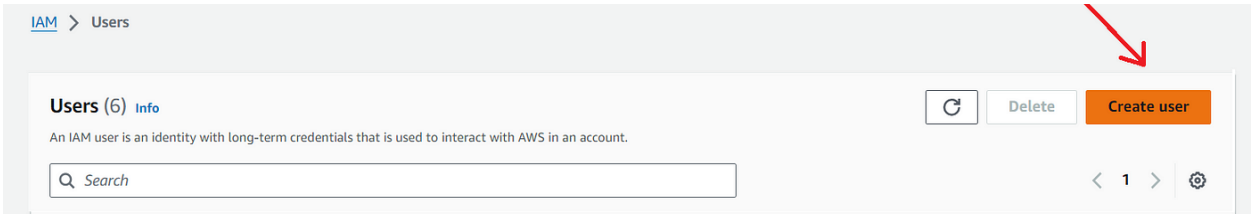
1. Setting up the AWS and Git environment:

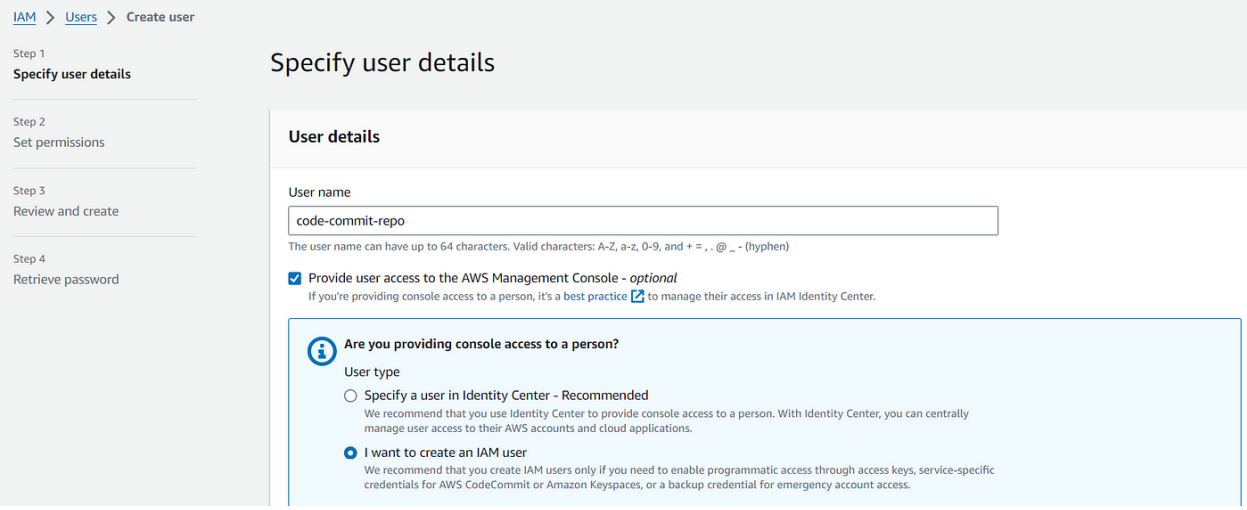
i) Create a new repository on AWS:

- Log in to the AWS Management Console.
- Navigate to the **CodeCommit** service.
- Click on '**Create repository**' and follow the prompts to set up your repository.

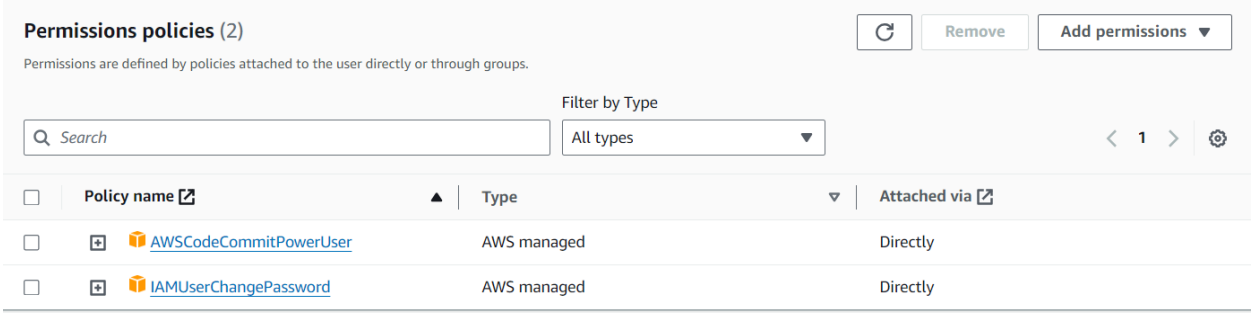


- Create an **IAM user** for this repo (Devops-on-AWS) because it can't be logged in with the root user.

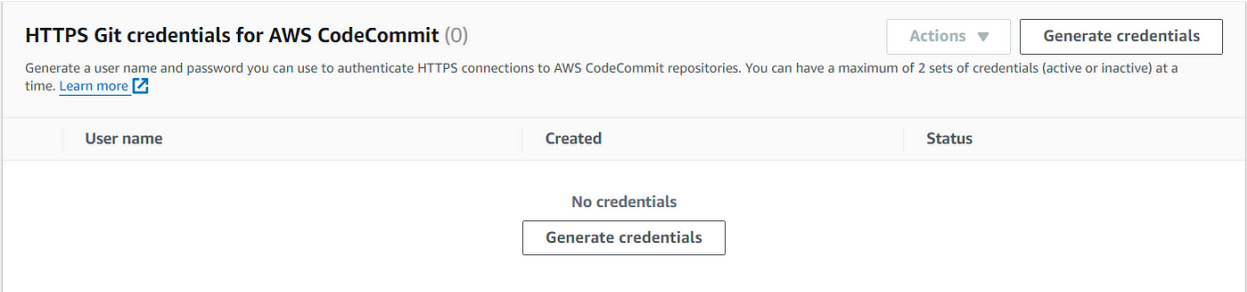




- Add the following permissions to this user.



- Generate credentials for this user. Navigate to the section ‘security credentials’ > HTTPS Git credentials for **AWS CodeCommit**.

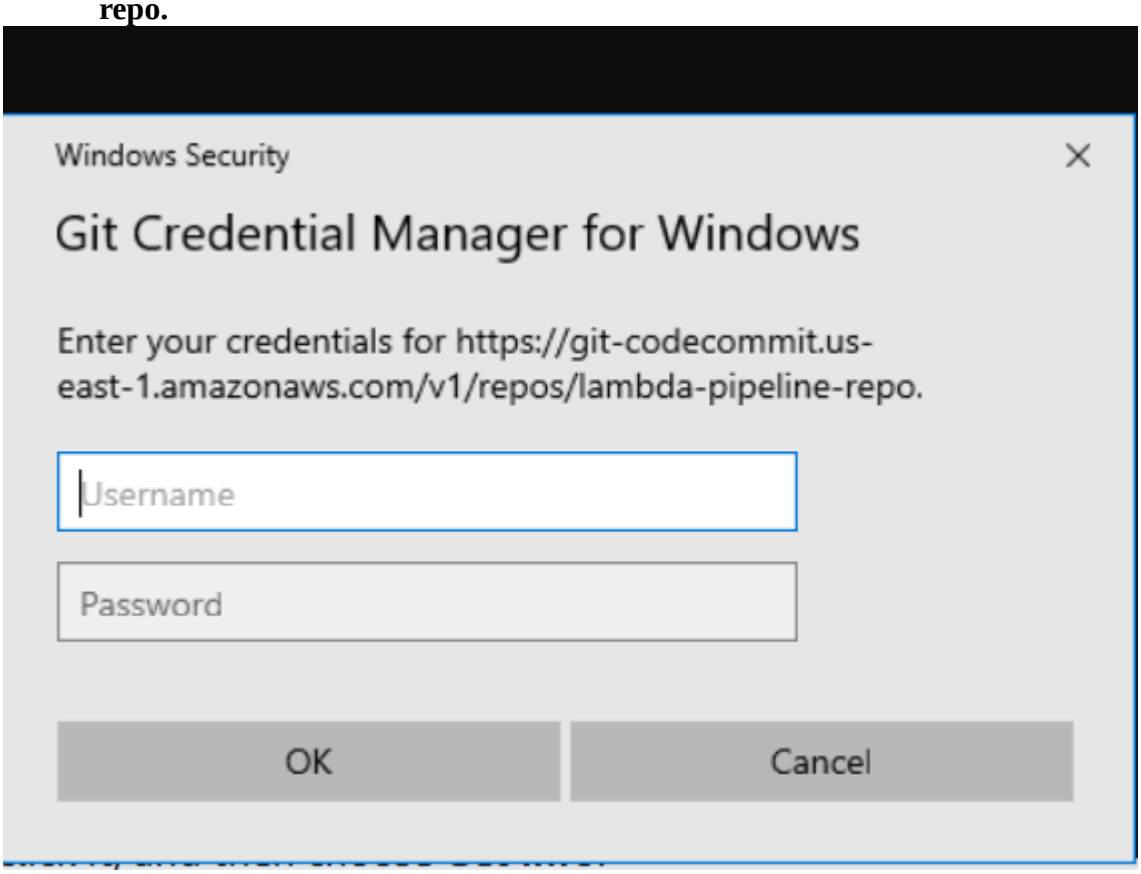


- Download the credentials after creating it from here.

ii) Clone the repository:

- Once the repository is created, clone it to your local machine using Git.
- Command: **git clone <repo-url>**
- Note: Git should be installed on your local machine.

- It will ask for credentials; copy the credentials we have downloaded for user **code-commit-repo.**



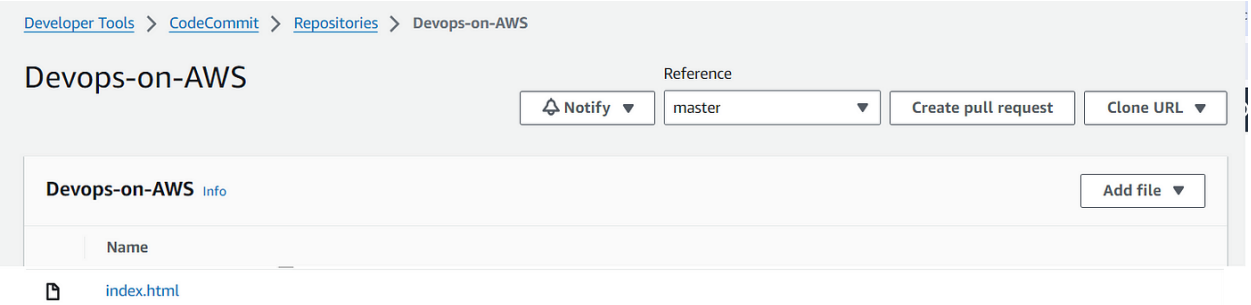
iii) Add files and commit changes:

- Add the first project files and commit the changes. We have an **index.html** file here.

```
cd Devops-on-AWS/
vim index.html
```

```
git add .
git commit -m "index.html added"
git push origin master
```

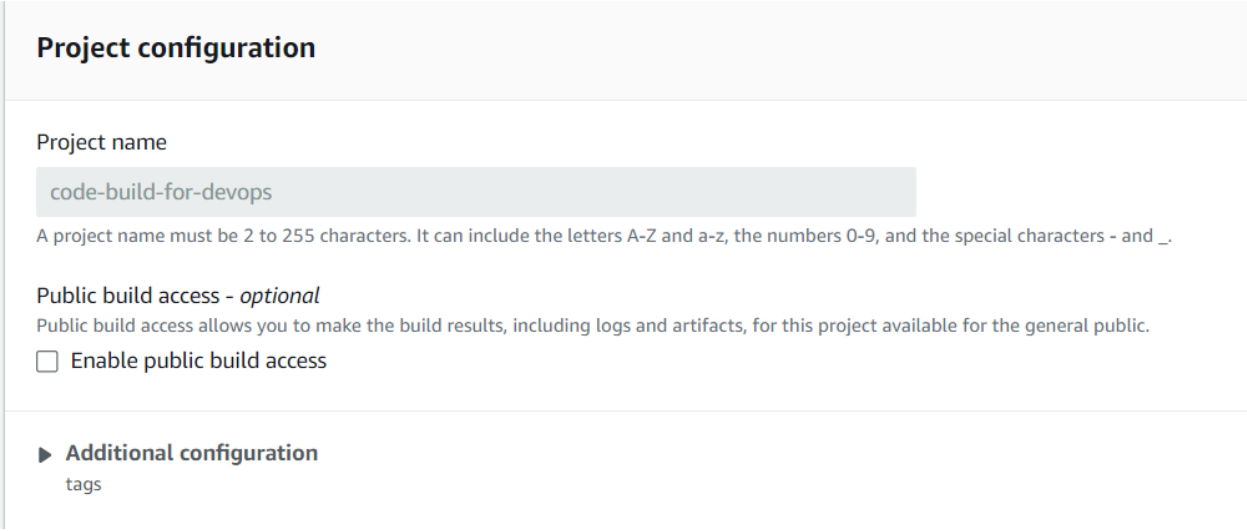
- You can observe the file pushed to the AWS repository.

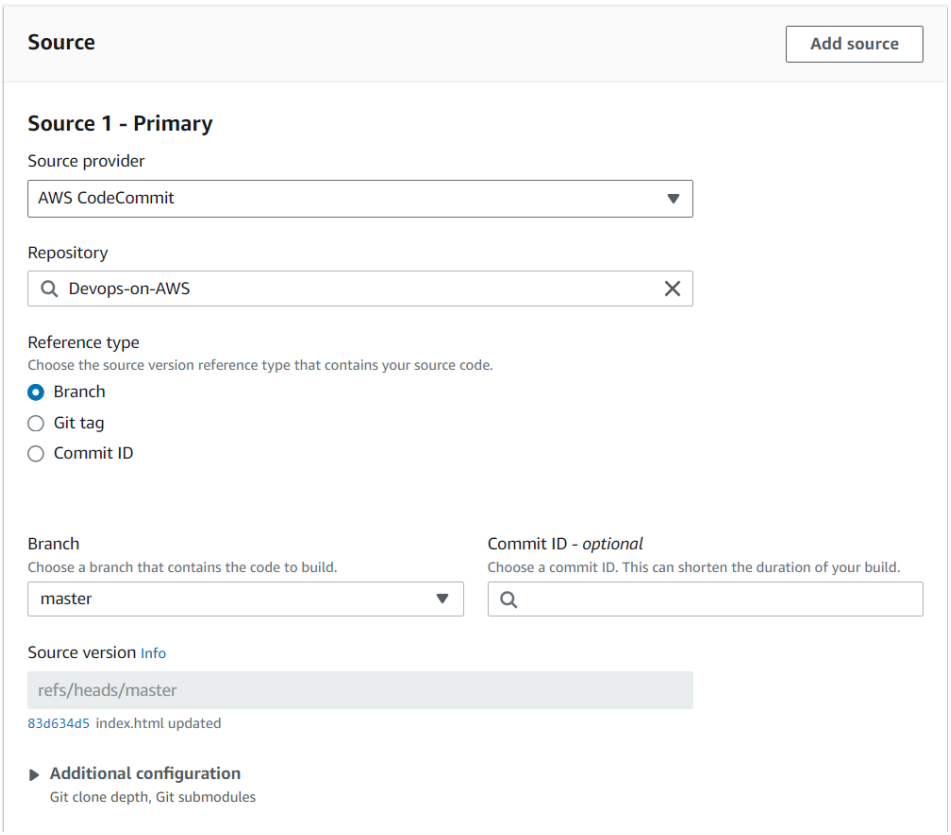


2. **Code Build:** CodeBuild streamlines your development process by compiling your source code, executing unit tests, and generating deployable artifacts. It removes the necessity of setting up, maintaining, and scaling your build servers.

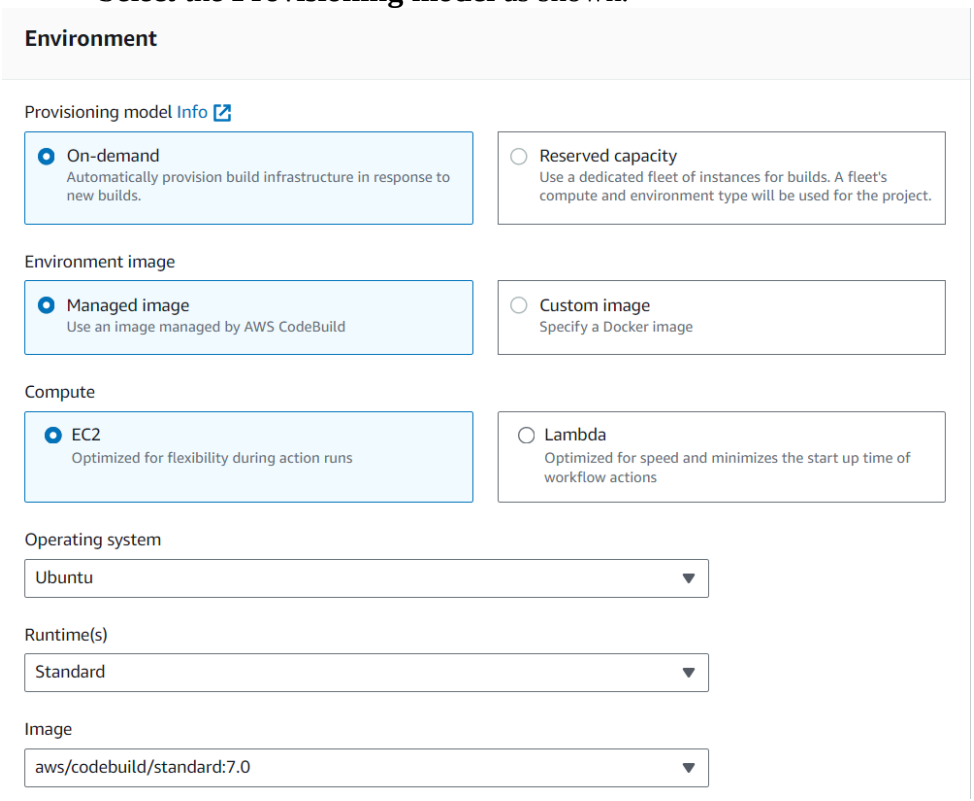
i) Create a build project:

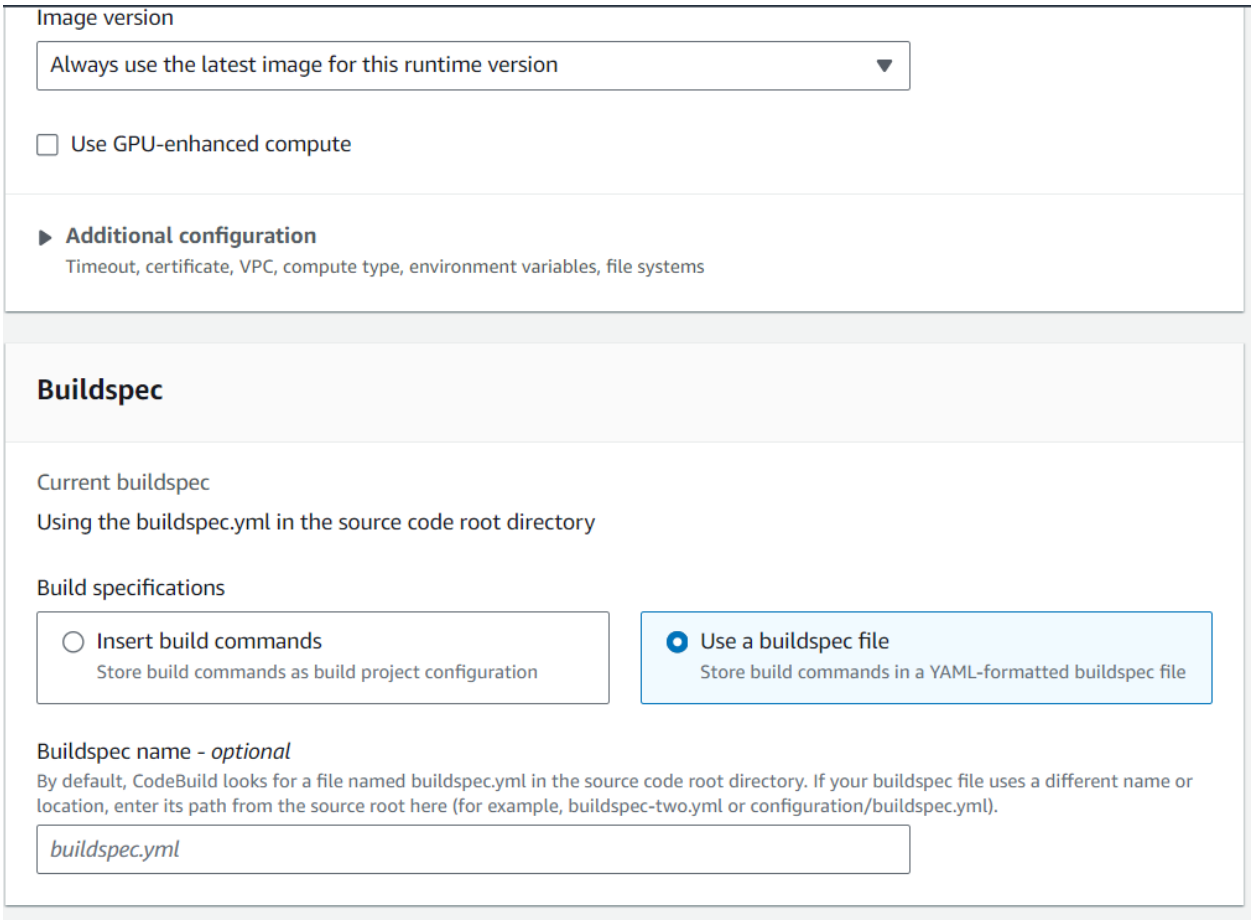
- Navigate to the **CodeBuild** service in the AWS Management Console.
- Click on 'Create build project'.
- Configure the build settings including source provider (CodeCommit), environment, and buildspec file.



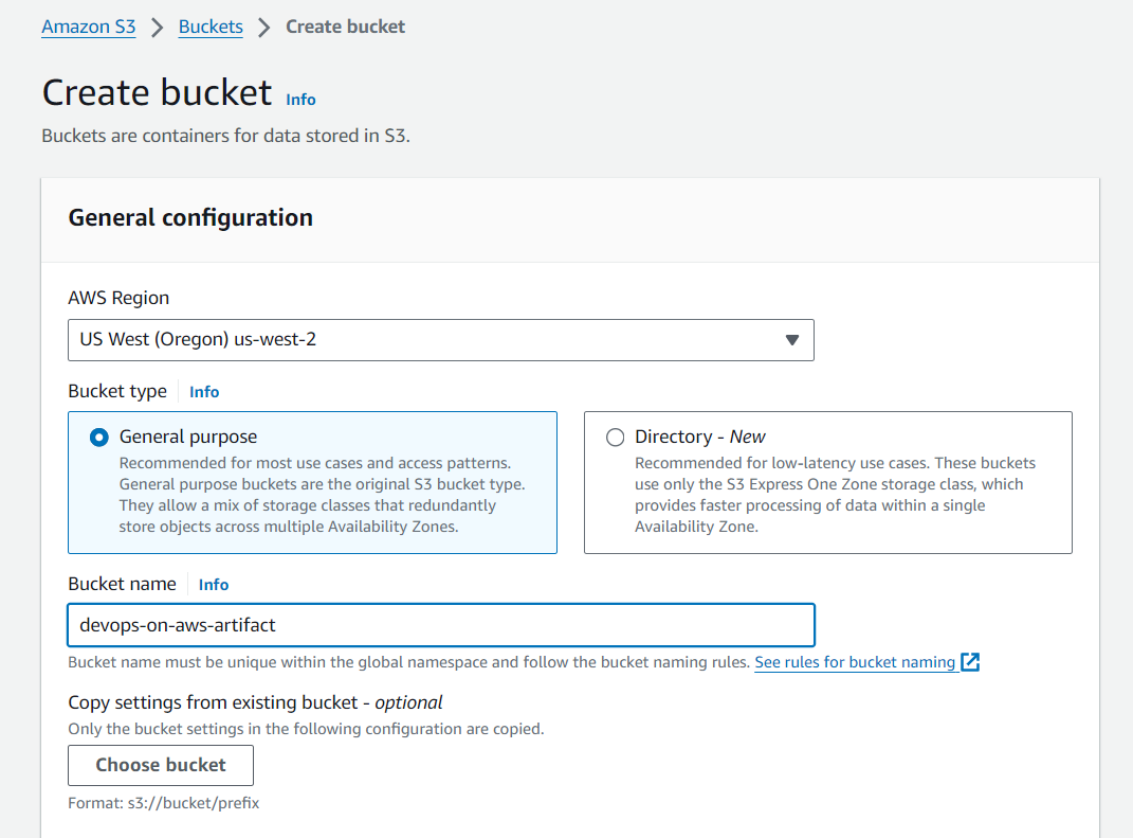


- Select the **Provisioning model** as shown.

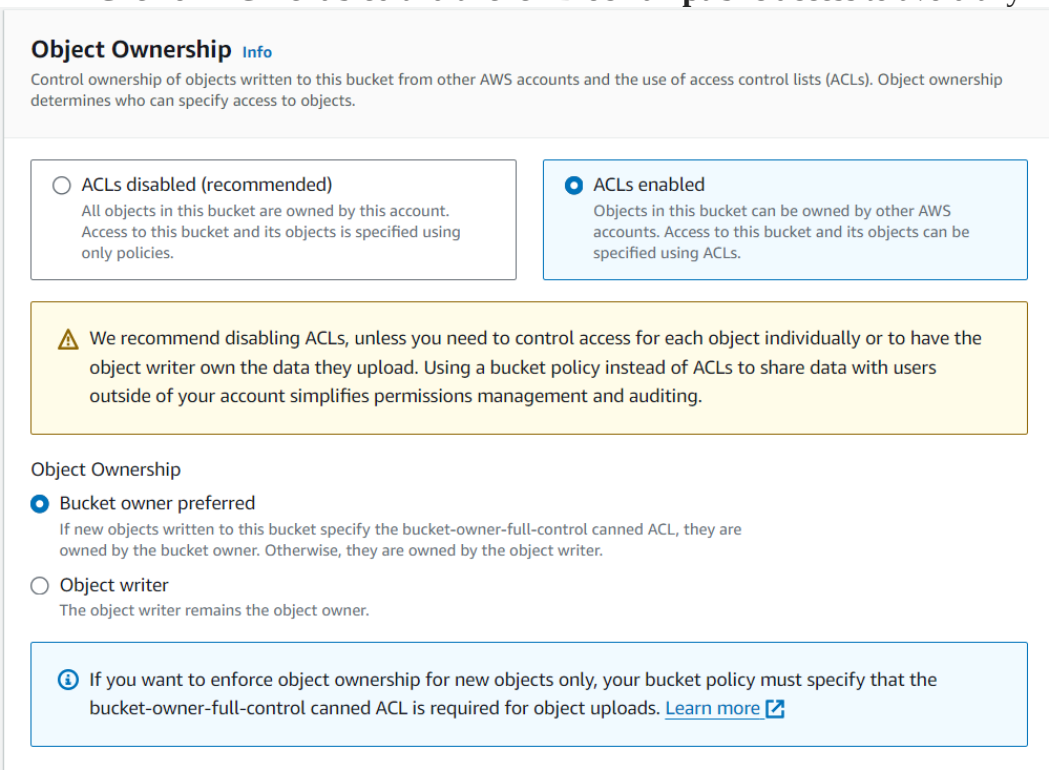


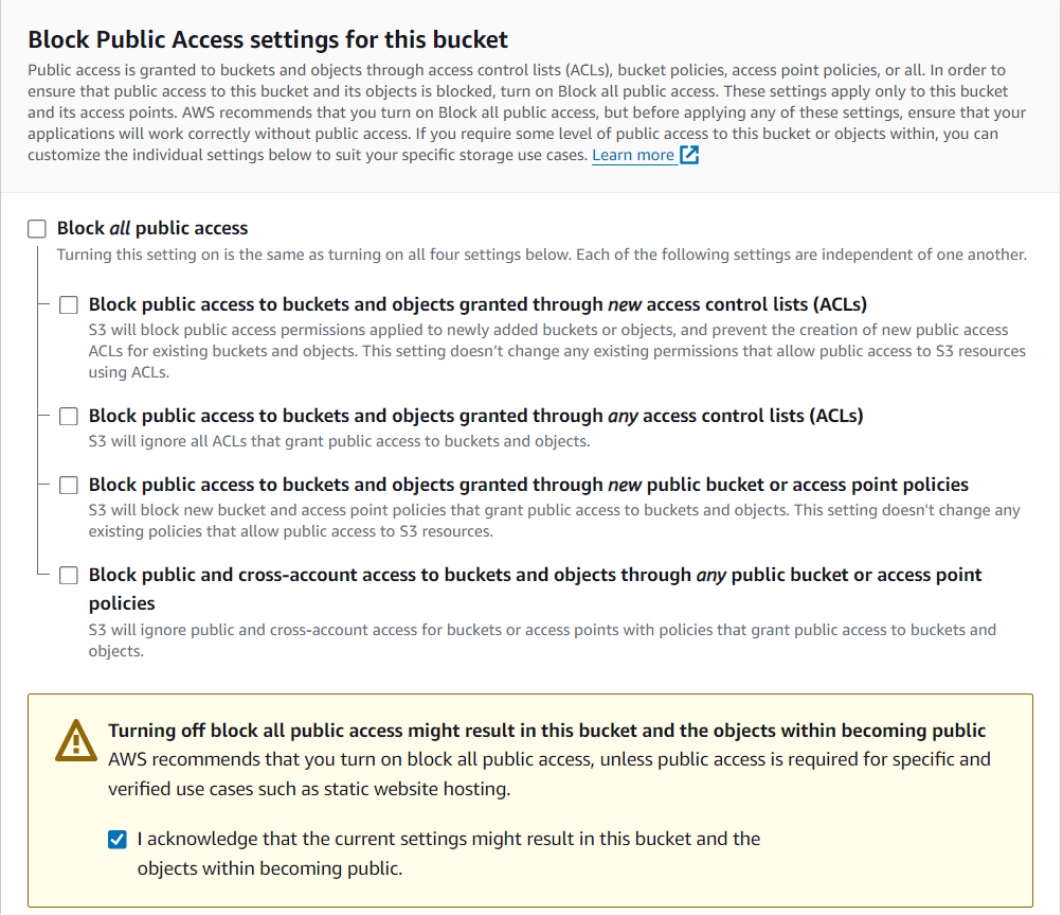


- Now for the next section, which is artifacts, you must create an S3 **bucket**.
- Go to the Amazon S3 dashboard and click on '**Create Bucket**'.

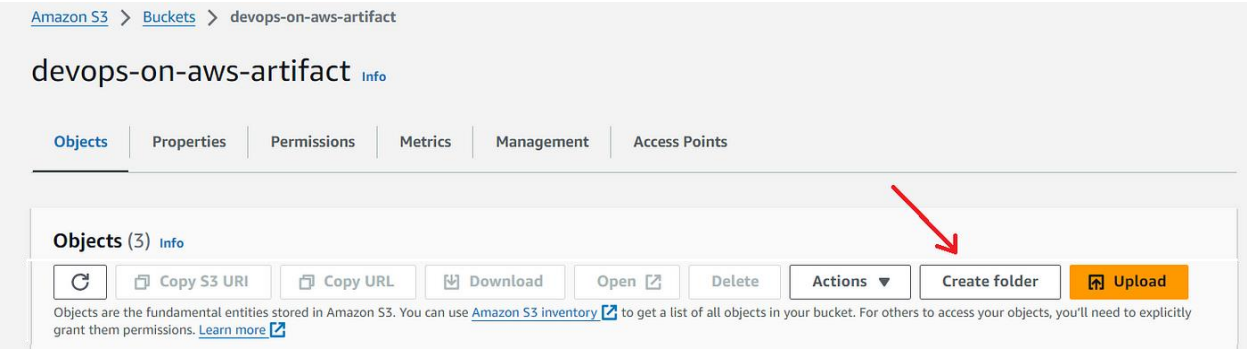


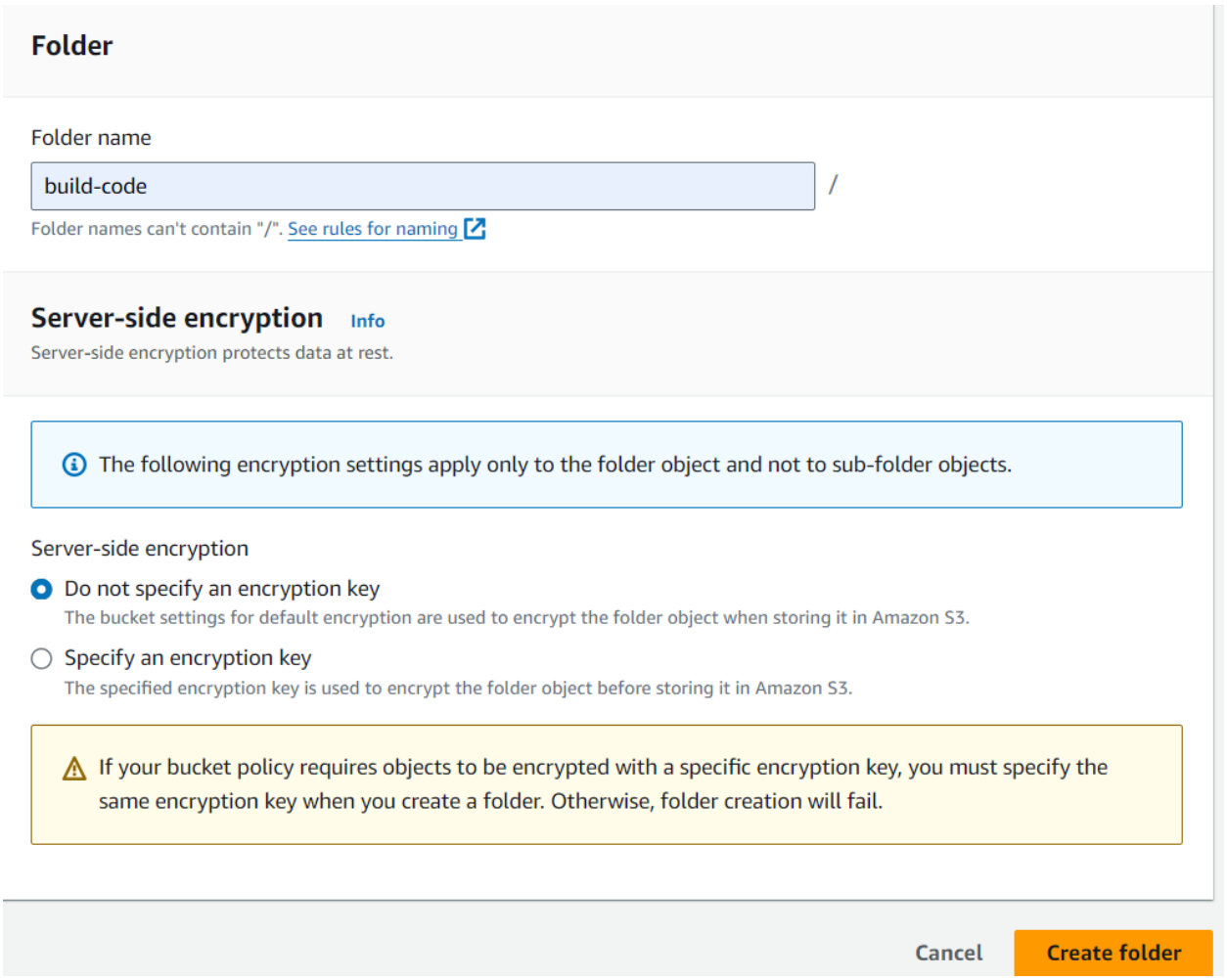
- Click on **ACL enabled** and unclick **Block all public access** to avoid any kind of errors.



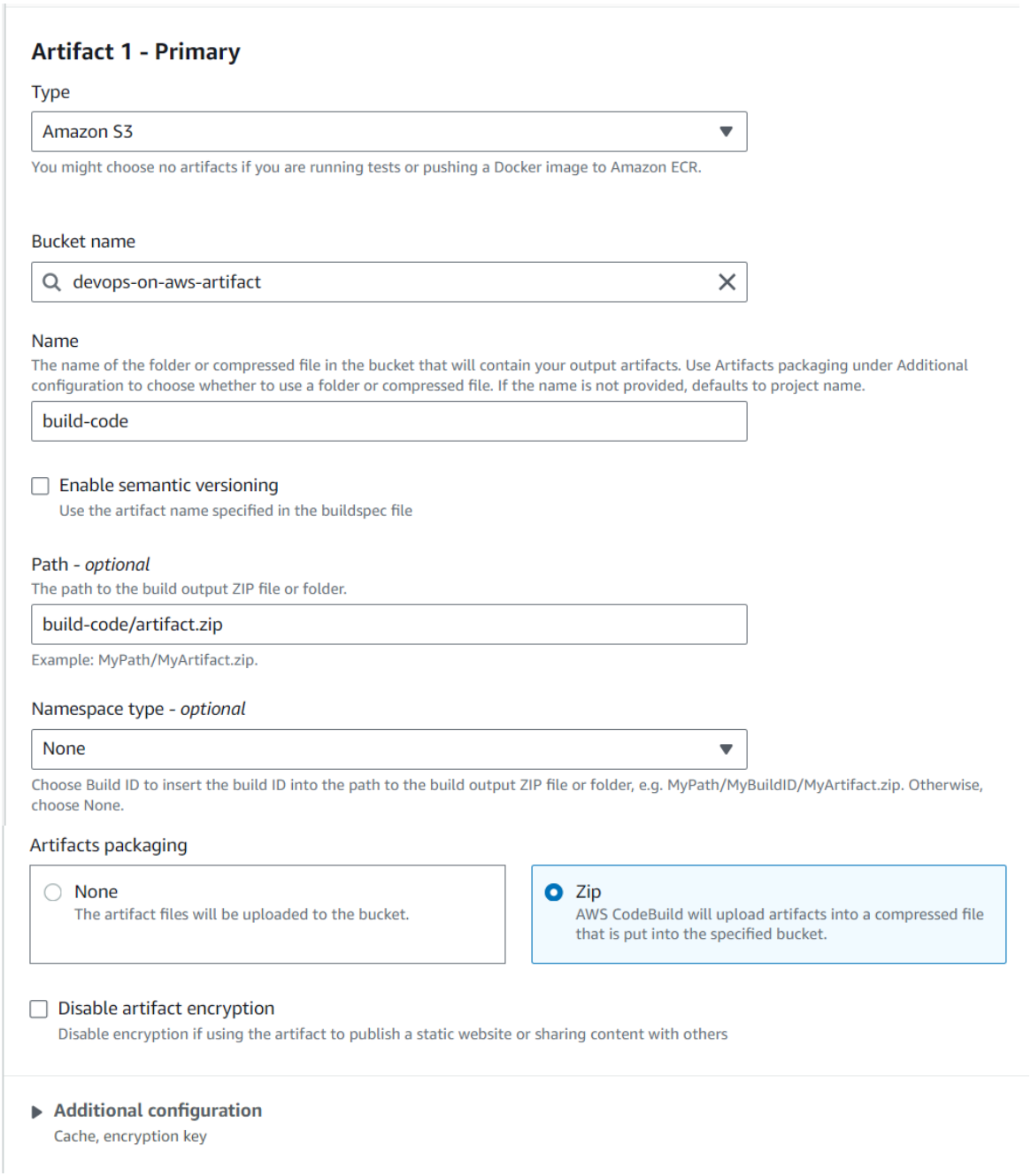


- After creating the bucket, create one folder inside it where **codebuild** will store the build files.





- Now add this bucket information into the **Artifacts** section in the code build configuration.



ii) Define Build Specifications:

A buildspec serves as the blueprint for CodeBuild to execute your build process effectively. It's crafted in YAML format and comprises essential build commands and configurations. Devoid of a build spec, CodeBuild lacks the instructions necessary to transform your input into output and identify the resultant artifact in the build environment for uploading to your designated output bucket.

- Create a buildspec.yml file in the root of your CodeCommit repository.

- Define the build phases such as install, pre-build, build, and post-build commands in the buildspec.yml.

```
MINGW64 ~/Documents/Devops-AWS/Devops-on-AWS (master)
```

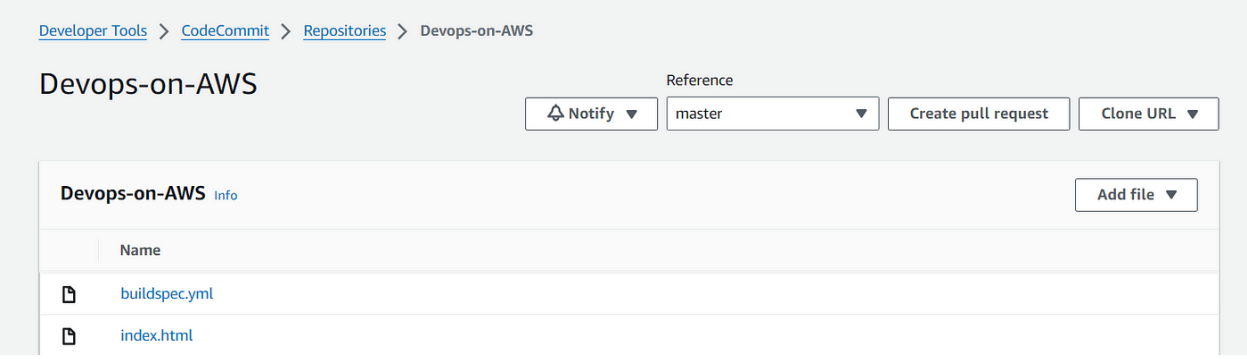
- Command: **$ vim buildspec.yml** (In Git Bash, the `$vim` command launches the Vim text editor. Vim is a powerful and highly configurable text editor that operates within the terminal. When you run `$vim` in Git Bash, it opens the Vim editor, allowing you to create, edit, and manipulate text files directly from the command line.)

In git bash:

```
version: 0.2

phases:
  install:
    commands:
      - echo Installing NGINX
      - sudo apt-get update
      - sudo apt-get install nginx -y

  build:
    commands:
      - echo Build started on `date`
      - cp index.html /var/www/html

  post_build:
    commands:
      - echo Configuring NGINX

artifacts:
  files:
    - '**/*'
```

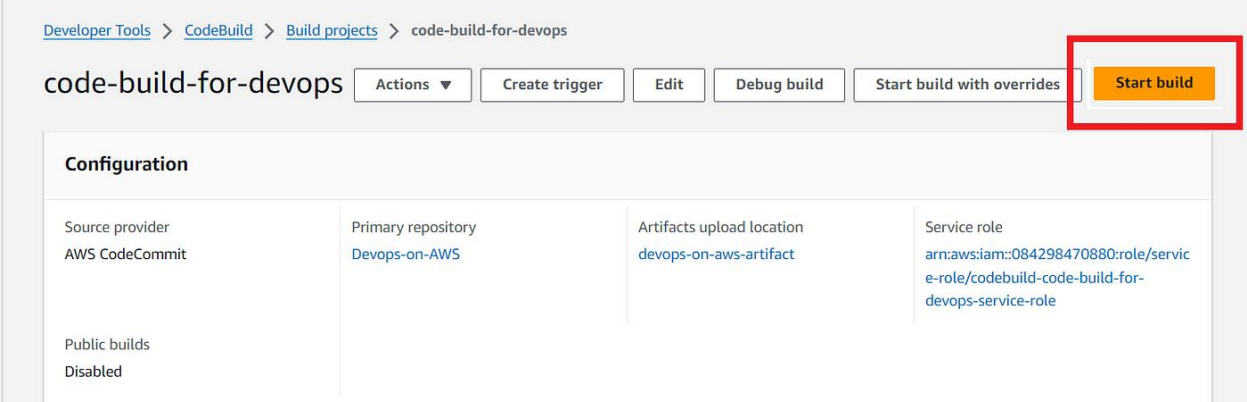
- After creating this, add the file, commit the changes, and push it into your code **commit** repo.

```
ls
git add .
git commit -m "adding buildspec file"
git push
vim appsec.yml
```

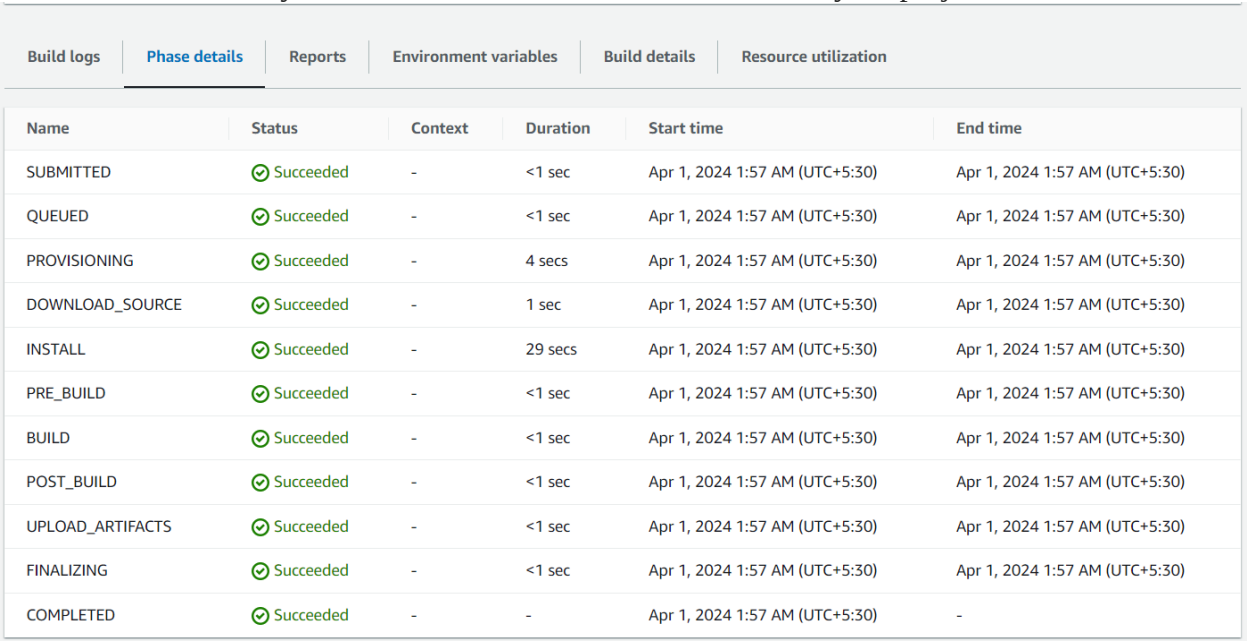


3. Start A Build:

- After uploading your **buildspec.yml** file, start your build.

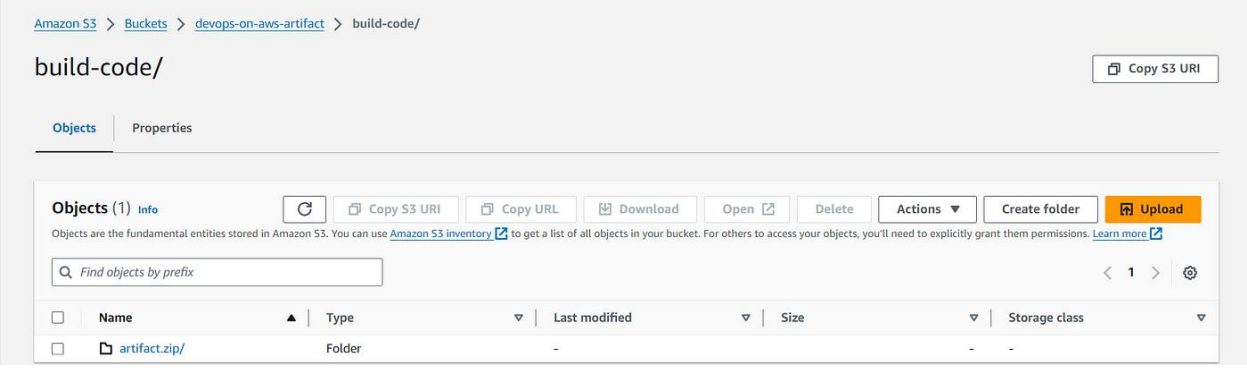


- You can trace your build in the **'Phase details'** section of your project.



Name	Status	Context	Duration	Start time	End time
SUBMITTED	Succeeded	-	<1 sec	Apr 1, 2024 1:57 AM (UTC+5:30)	Apr 1, 2024 1:57 AM (UTC+5:30)
QUEUED	Succeeded	-	<1 sec	Apr 1, 2024 1:57 AM (UTC+5:30)	Apr 1, 2024 1:57 AM (UTC+5:30)
PROVISIONING	Succeeded	-	4 secs	Apr 1, 2024 1:57 AM (UTC+5:30)	Apr 1, 2024 1:57 AM (UTC+5:30)
DOWNLOAD_SOURCE	Succeeded	-	1 sec	Apr 1, 2024 1:57 AM (UTC+5:30)	Apr 1, 2024 1:57 AM (UTC+5:30)
INSTALL	Succeeded	-	29 secs	Apr 1, 2024 1:57 AM (UTC+5:30)	Apr 1, 2024 1:57 AM (UTC+5:30)
PRE_BUILD	Succeeded	-	<1 sec	Apr 1, 2024 1:57 AM (UTC+5:30)	Apr 1, 2024 1:57 AM (UTC+5:30)
BUILD	Succeeded	-	<1 sec	Apr 1, 2024 1:57 AM (UTC+5:30)	Apr 1, 2024 1:57 AM (UTC+5:30)
POST_BUILD	Succeeded	-	<1 sec	Apr 1, 2024 1:57 AM (UTC+5:30)	Apr 1, 2024 1:57 AM (UTC+5:30)
UPLOAD_ARTIFACTS	Succeeded	-	<1 sec	Apr 1, 2024 1:57 AM (UTC+5:30)	Apr 1, 2024 1:57 AM (UTC+5:30)
FINALIZING	Succeeded	-	<1 sec	Apr 1, 2024 1:57 AM (UTC+5:30)	Apr 1, 2024 1:57 AM (UTC+5:30)
COMPLETED	Succeeded	-	-	Apr 1, 2024 1:57 AM (UTC+5:30)	-

- Verify by going to your artifacts manually, that is S3 bucket whether your build is uploading there or not.



The build is finally completed.

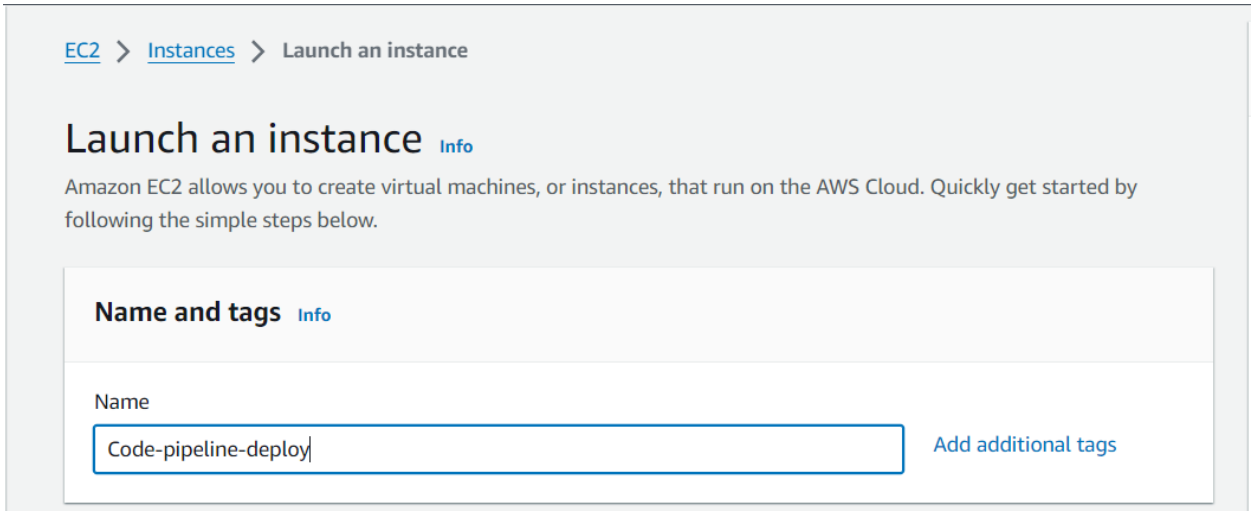
i)Build Environment: CodeBuild offers pre-configured environments like Ubuntu, Amazon Linux, and Windows, so pick the one that fits your project. You can also create custom environments using Docker images for more specialized requirements.
ii) Build Caching: Improve build performance by enabling build caching. CodeBuild caches dependencies and intermediate build artifacts, reducing build times for subsequent runs.

4. Creating EC2 Instance & Set-Up Agent:

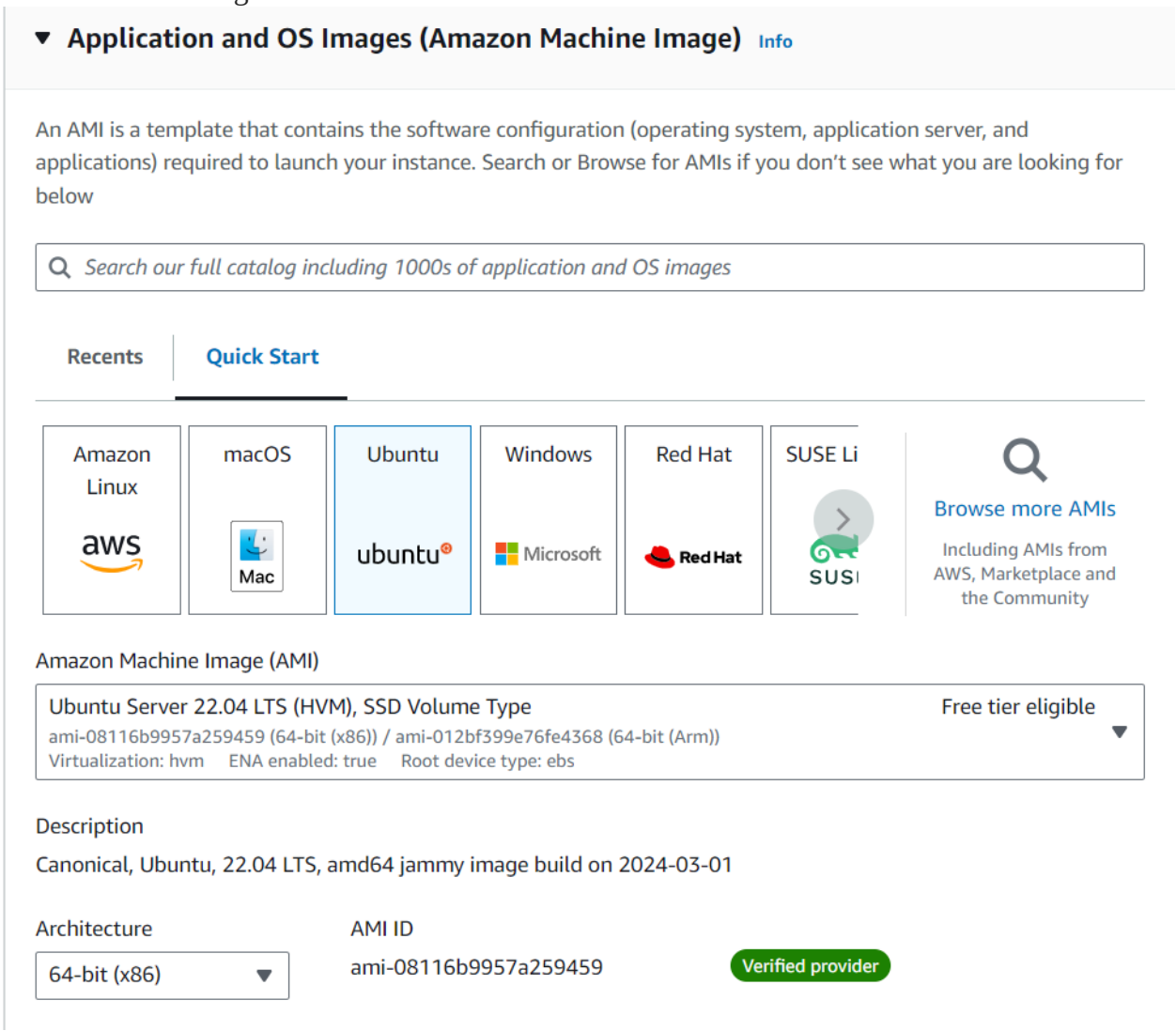
- Before proceeding with AWS CodeDeploy to automate software deployments, it's essential to set up an EC2 instance where your application will be deployed. This instance serves as the target environment for your deployments. Therefore, the initial step involves creating an EC2 instance within your AWS environment.

i) Create an EC2 Instance:

- Go to EC2 dashboard > instances > launch instance.



- Choose image **‘Ubuntu 22.04’**.



- Choose instance type **‘t2.micro’**, it is enough for this project.

Instance type Info | Get advice

Instance type

t2.micro — Free tier eligible

Family: t2 1 vCPU 1 GiB Memory Current generation: true
On-Demand Linux base pricing: 0.0116 USD per Hour
On-Demand SUSE base pricing: 0.0116 USD per Hour
On-Demand Windows base pricing: 0.0162 USD per Hour
On-Demand RHEL base pricing: 0.0716 USD per Hour

All generations

Compare instance types

Additional costs apply for AMIs with pre-installed software

Key pair (login) Info

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - *required*

- Allow all **'SSH Traffic'** and **'HTTPS Traffic'** in the firewall.

Network settings Info

Edit

Network Info

vpc-0c8b7481e2980ed0d

Subnet Info

No preference (Default subnet in any availability zone)

Auto-assign public IP Info

Enable

Firewall (security groups) Info

A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

(•) Create security group ( ) Select existing security group

We'll create a new security group called '**launch-wizard-3**' with the following rules:

[x] Allow SSH traffic from — Anywhere 0.0.0.0/0
Helps you connect to your instance

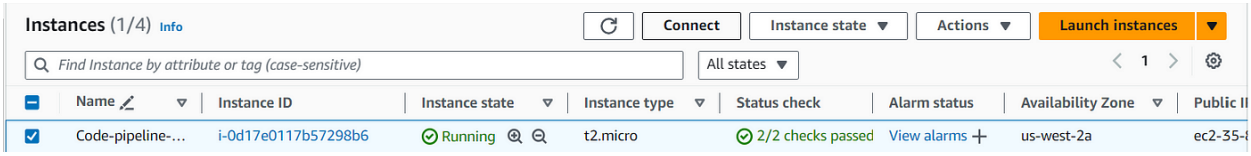
[x] Allow HTTPS traffic from the internet
To set up an endpoint, for example when creating a web server

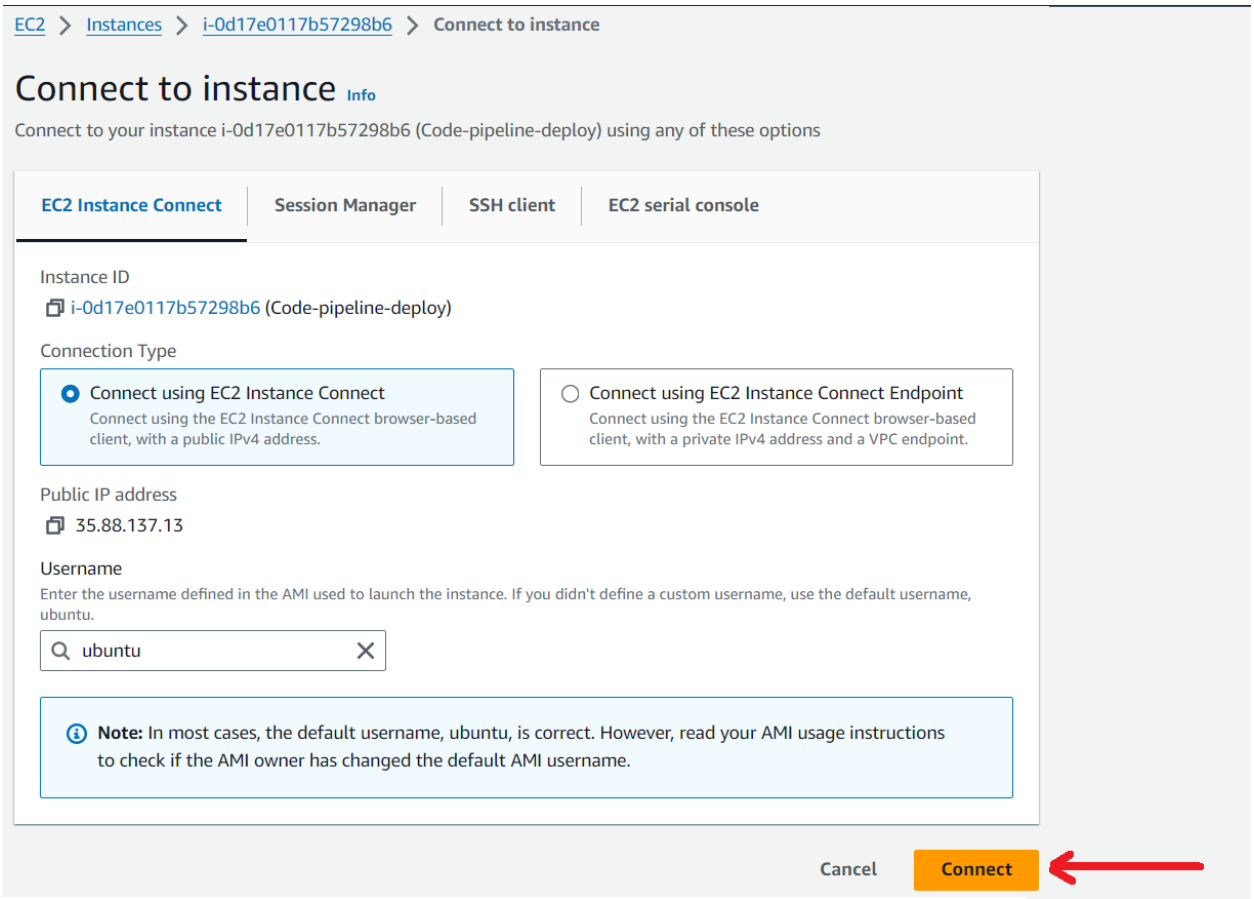
[x] Allow HTTP traffic from the internet
To set up an endpoint, for example when creating a web server

⚠ Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only. ✕

ii) Setting Up 'CodeDeploy' Agent in EC2:

- The AWS CodeDeploy agent plays a crucial role in facilitating deployments by acting as a bridge between your EC2 instances and the CodeDeploy service. Once installed and configured on an instance, this software package enables seamless integration with CodeDeploy, allowing the instance to participate in deployment processes.
- Connect to your instance.





- Now create a script on the instance, **'agent-install.sh'**, and copy the below content in that file.

```
#!/bin/bash
# This installs the CodeDeploy agent and its prerequisites on Ubuntu 22.04.
sudo apt-get update
sudo apt-get install ruby-full ruby-webrick wget -y
cd /tmp
```

```
wget https://aws-codedeploy-us-west-2.s3.us-west-2.amazonaws.com/releases/codedeploy-agent_1.3.2-1902_all.deb
mkdir codedeploy-agent_1.3.2-1902_ubuntu22
dpkg-deb -R codedeploy-agent_1.3.2-1902_all.deb codedeploy-agent_1.3.2-1902_ubuntu22
sed 's/Depends:.*/Depends:ruby3.0/' -i ./codedeploy-agent_1.3.2-1902_ubuntu22/DEBIAN/control
dpkg-deb -b codedeploy-agent_1.3.2-1902_ubuntu22/
sudo dpkg -i codedeploy-agent_1.3.2-1902_ubuntu22.deb
systemctl list-units --type=service | grep codedeploy
sudo service codedeploy-agent status
```

- If your region is different than **Oregon,** then replace us-west-2 in the above script with your region code.
- Now run the script with the following command.

```
bash agent-install.sh
```

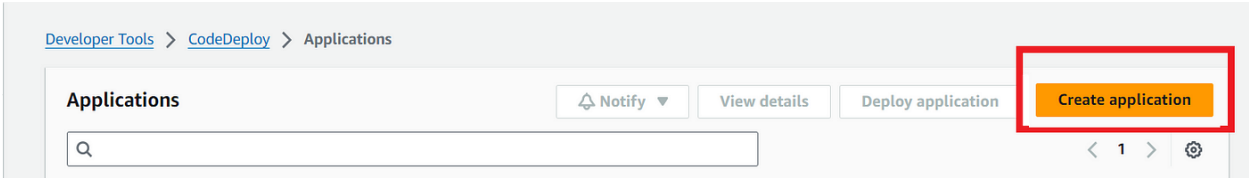
- This command will install an agent inside the instance.

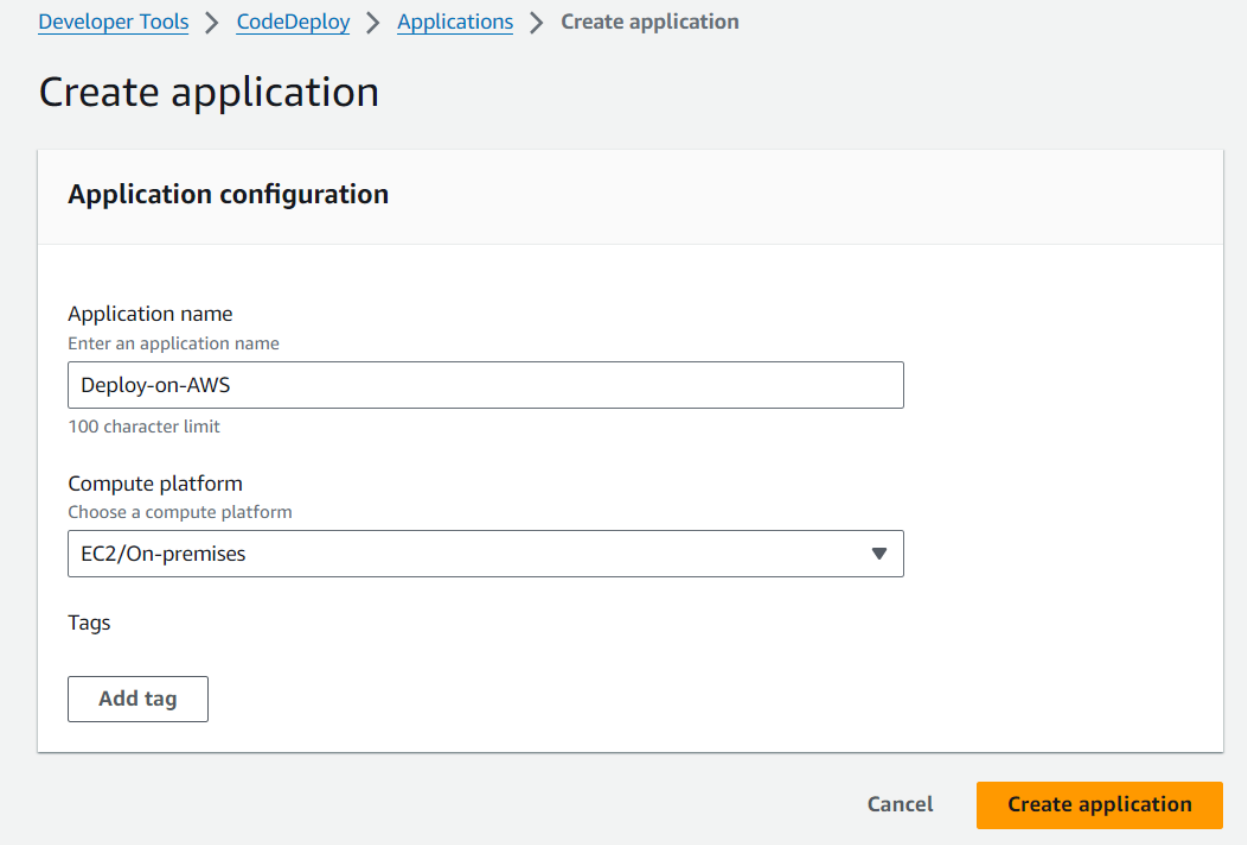
5. CodeDeploy:

- CodeDeploy automates application deployments across Amazon EC2 instances, on-premises servers, Lambda functions, and ECS services, streamlining the deployment process.

i) Prepare your application:

- Prepare your application for deployment by packaging it into a .zip file containing all necessary code and dependencies.

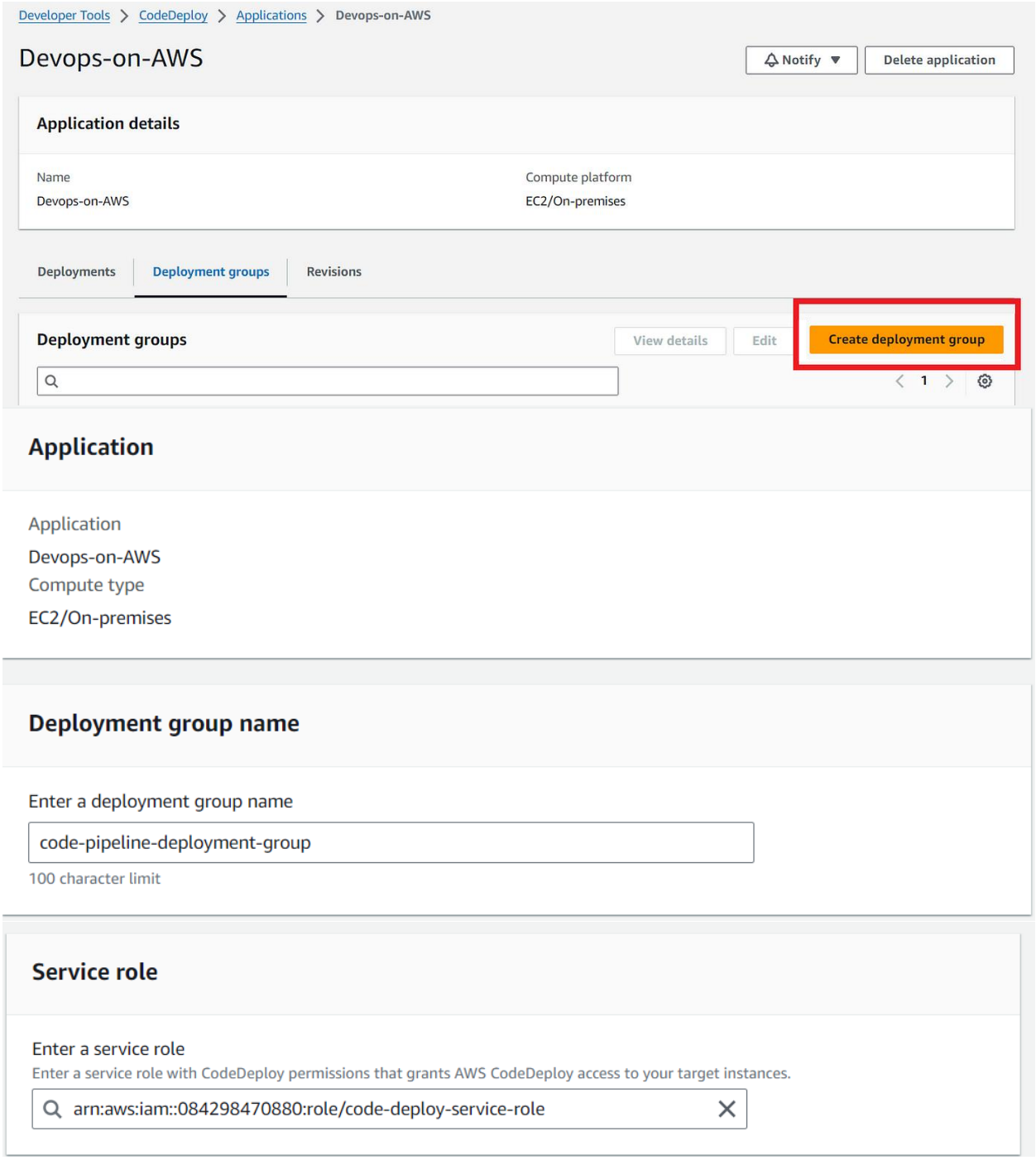




- After creating the Application, now create the deployment group.

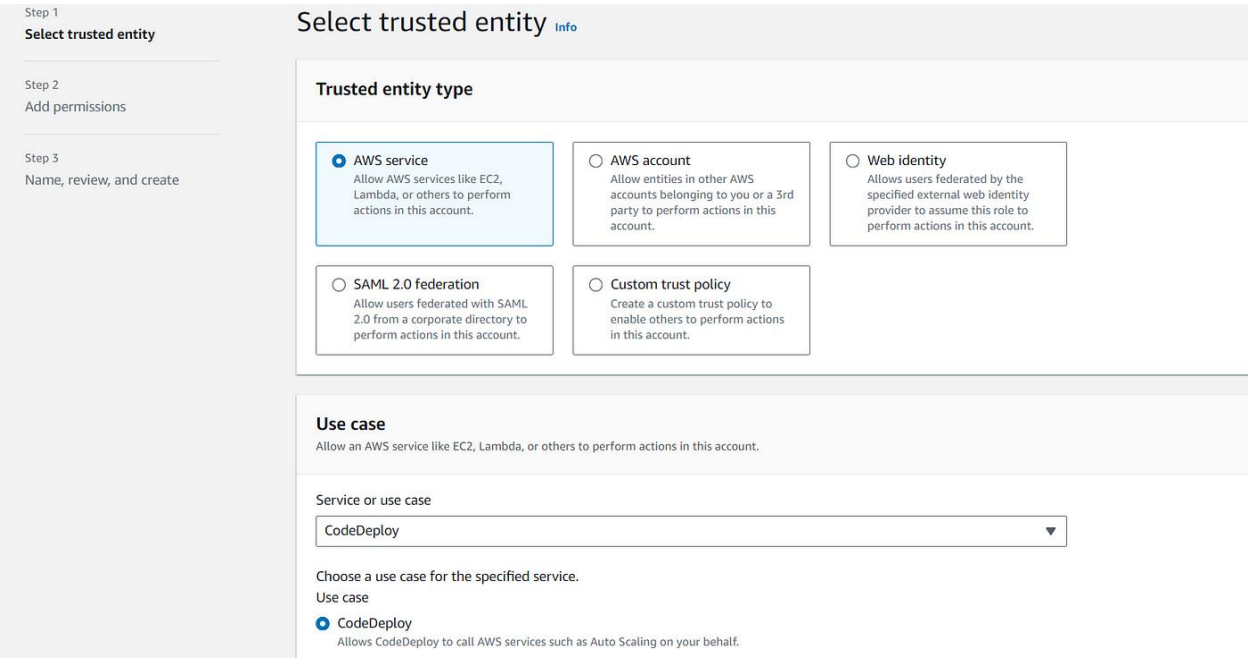
ii) Deployment Group: In an EC2/On-Premises deployment, a deployment group serves as a collection of specific instances designated for a deployment. This group comprises individually tagged instances, Amazon EC2 instances, or those within Amazon EC2 Auto Scaling groups. It provides a means to organize and manage the deployment process, ensuring that updates are applied precisely to the intended instances.

- Navigate to the **'CodeDeploy'** service in the AWS Management Console.
- Click **'Create deployment group'** and specify details such as deployment configuration, EC2 instances, and deployment type.

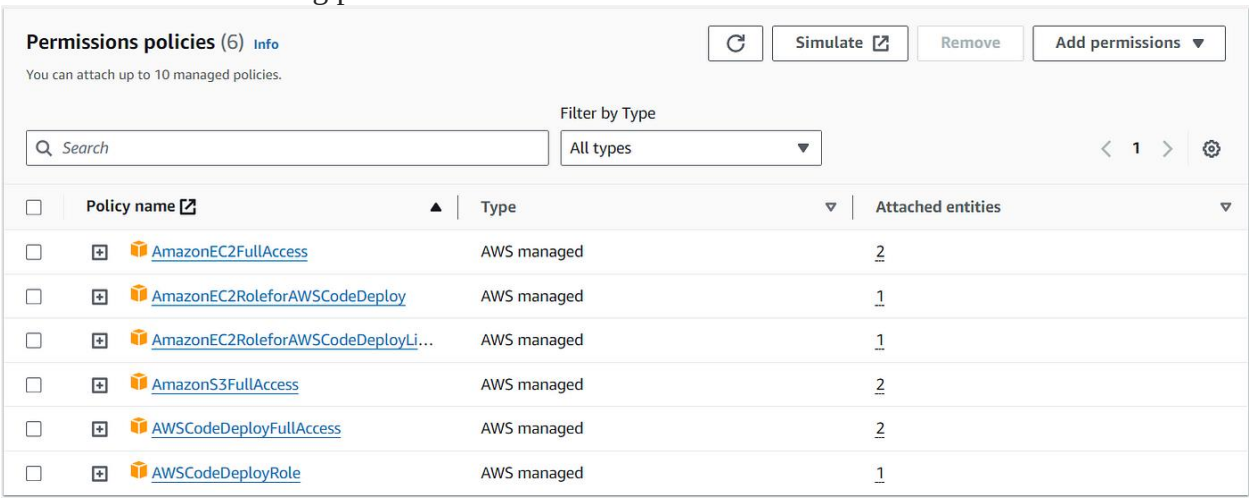


iii) Service Role:

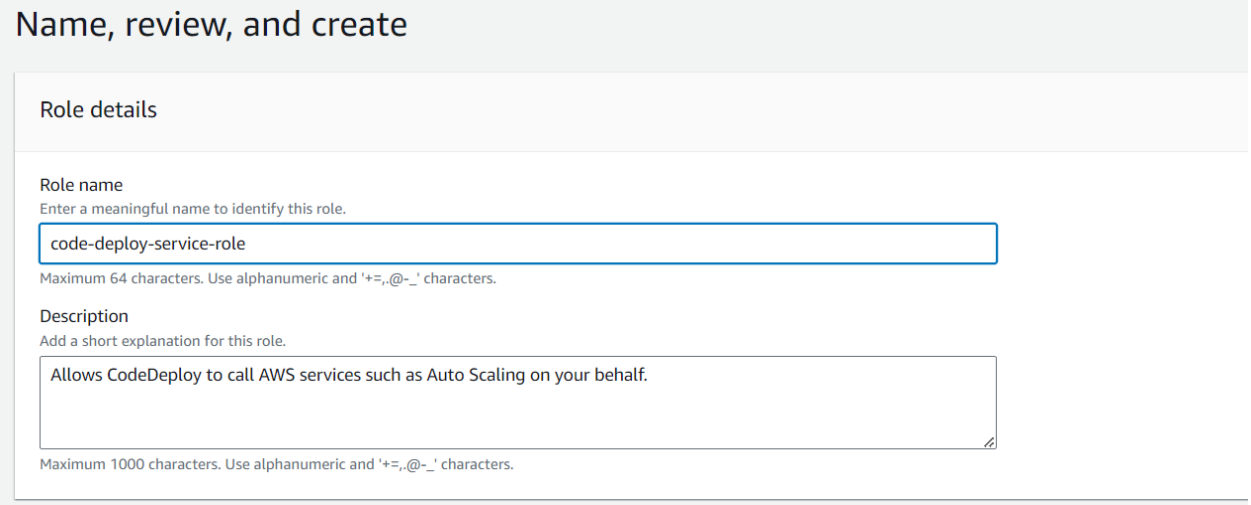
- Now to enter this service role we must create it first.
- Go to **IAM** > Roles > Create Role.



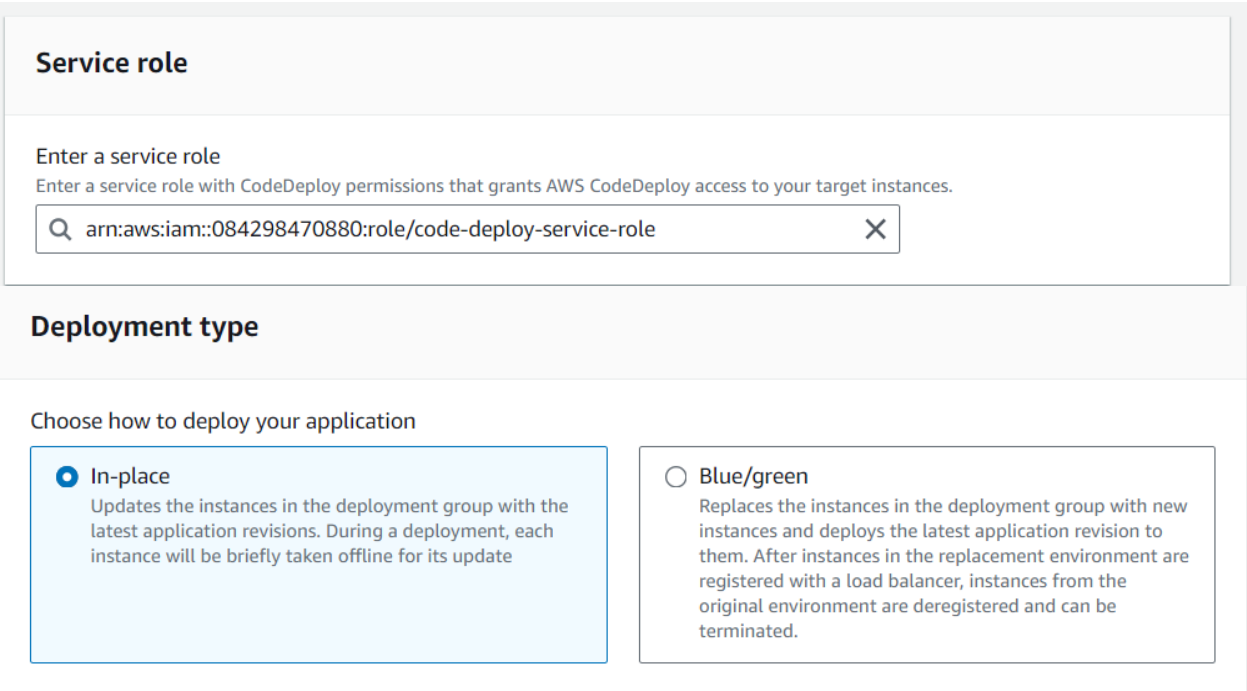
- Add the following permission to this Role.



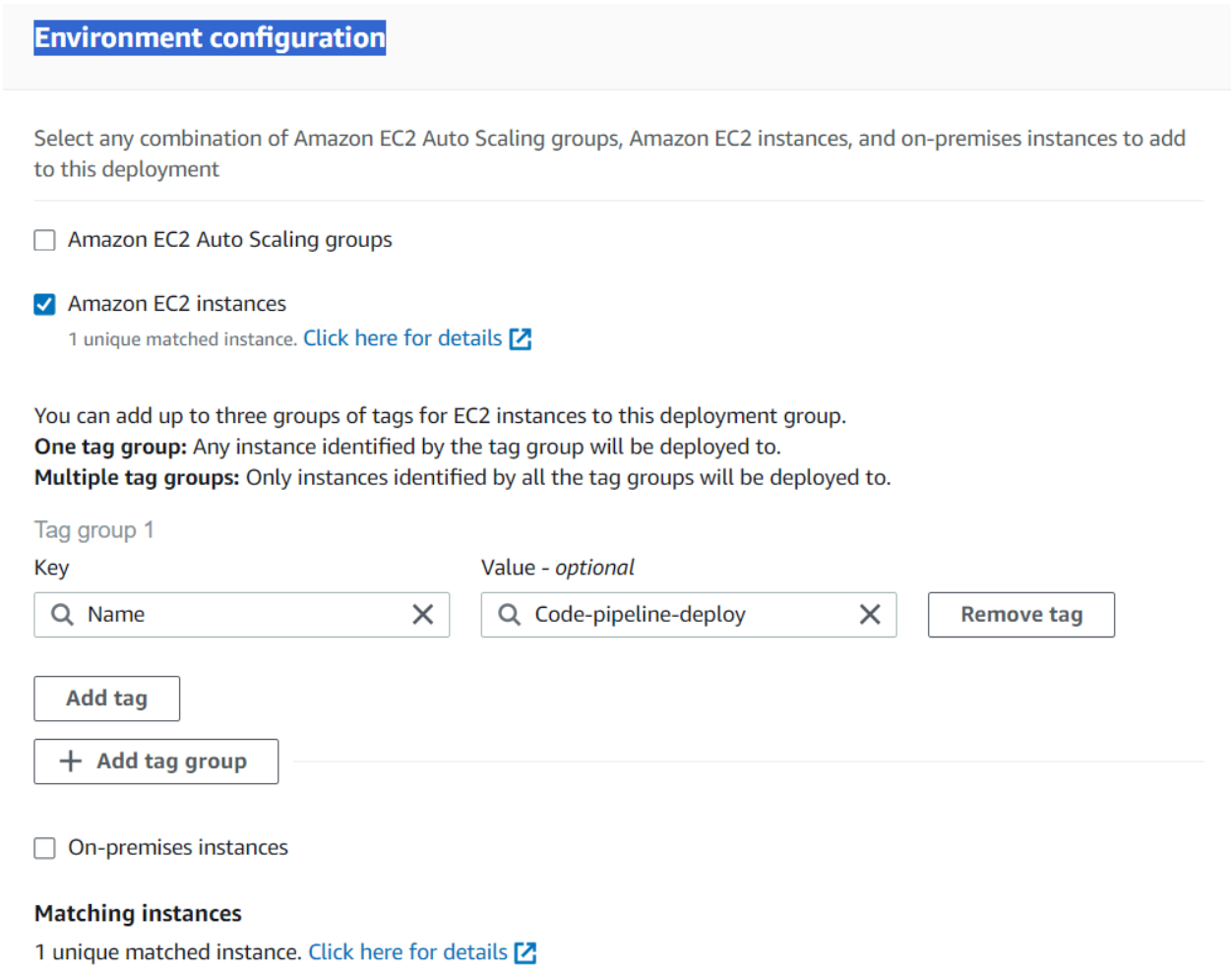
- Give a name and create it.



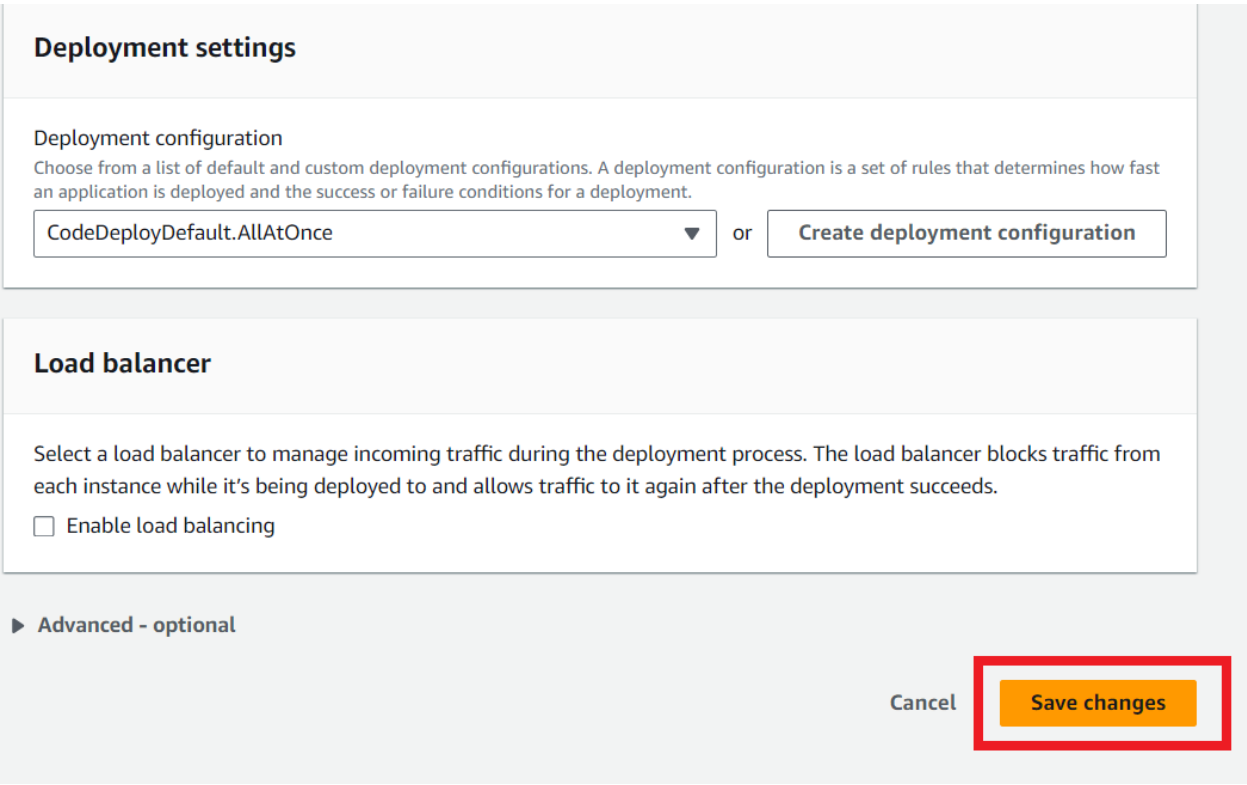
- After creating a role, return to your codeDeploy configuration and add this role to the service role section.



- Now we must enter our EC2-instance in '**Environment configuration'.**

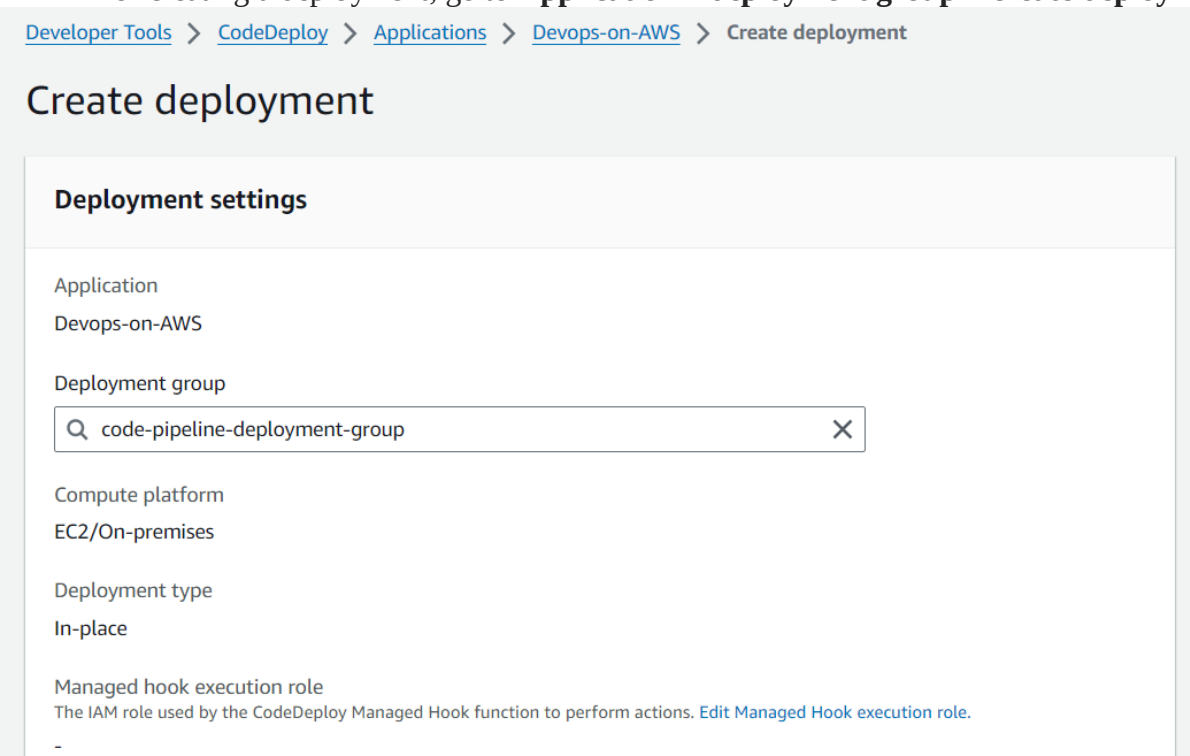


- Save the changes.

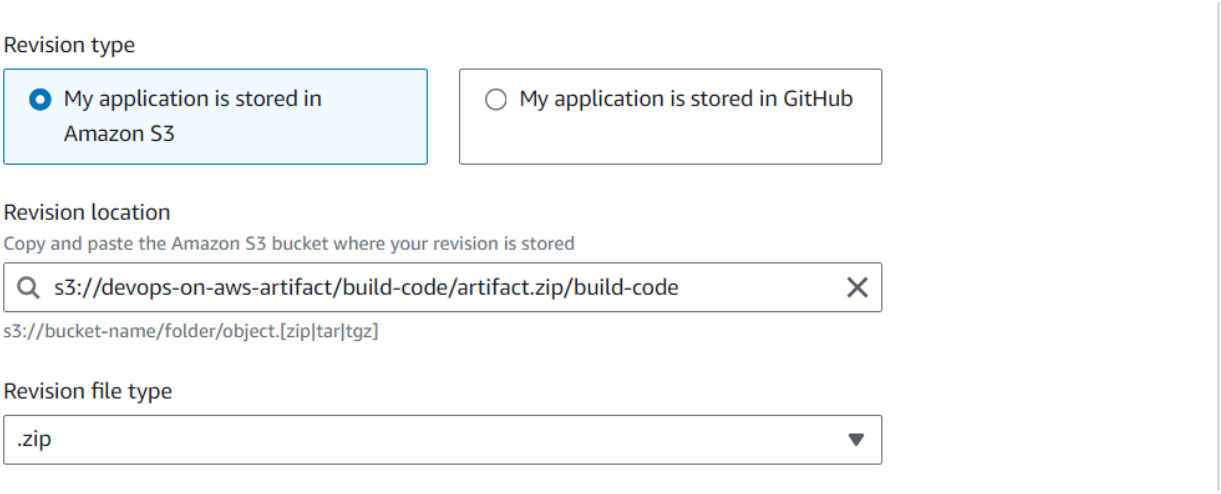


6. Create Deployment:

- For creating a deployment, go to **Application > deployment group > create deployment.**



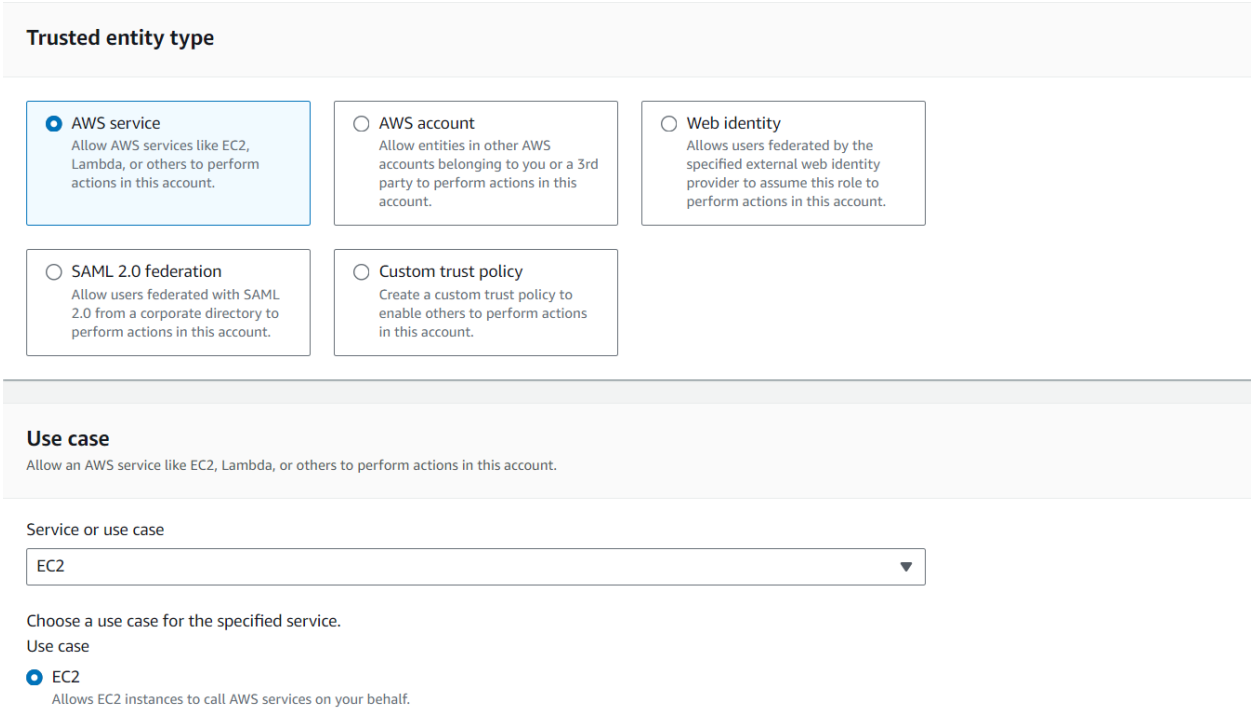
- Paste the link of your S3 artifact where your build files are stored by code build, in the below column.



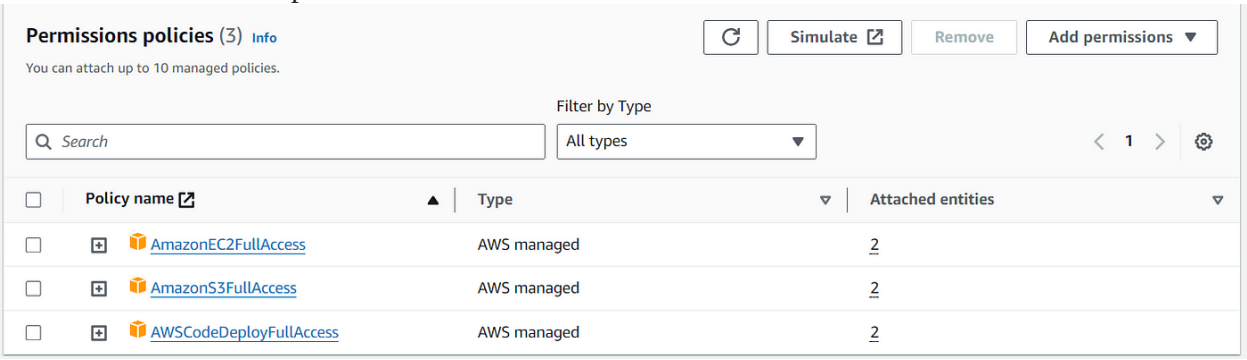
- And create deployment.

i)EC2-ROLE:

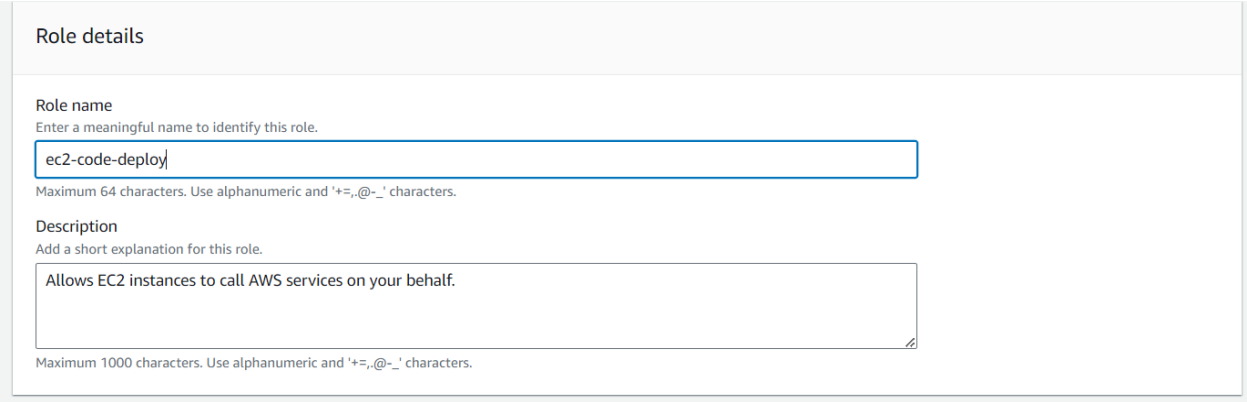
- After creating a deployment, it's essential to set up an additional role for the EC2 instance. This role facilitates seamless communication between the EC2 instance and both CodeDeploy and Amazon S3. This setup ensures that the instance can securely access and interact with the necessary services during the deployment process.
- Create Role. Go to **IAM > Roles > Create role.**



- Add the below permissions to this role.

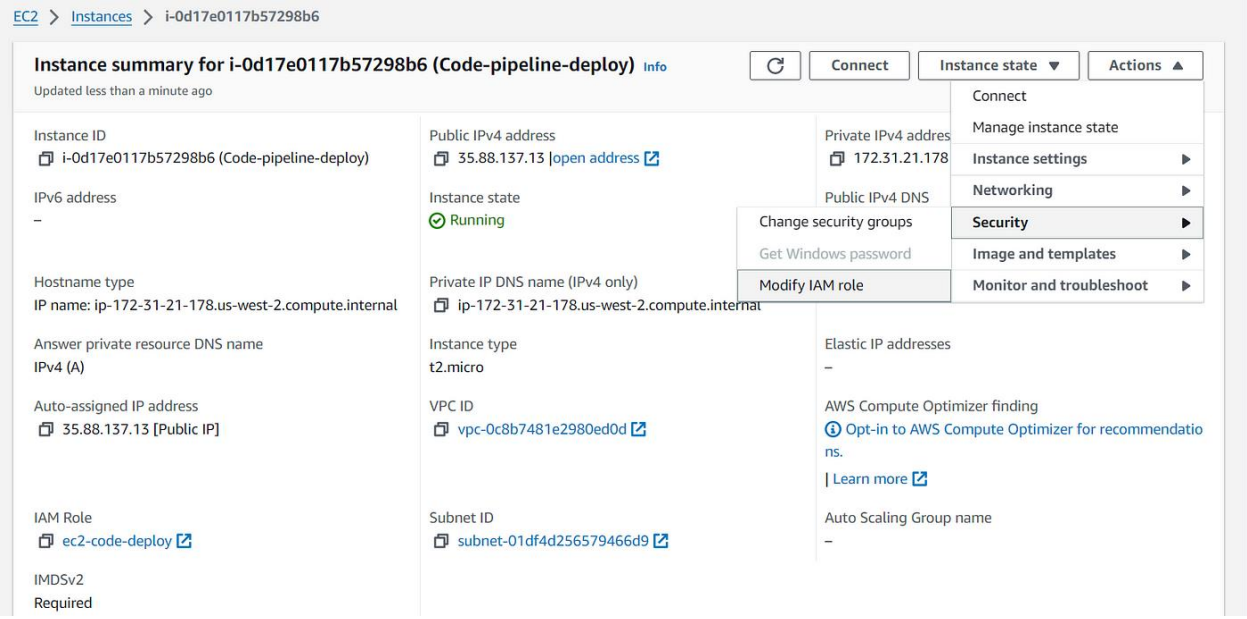


- Give a name and create.

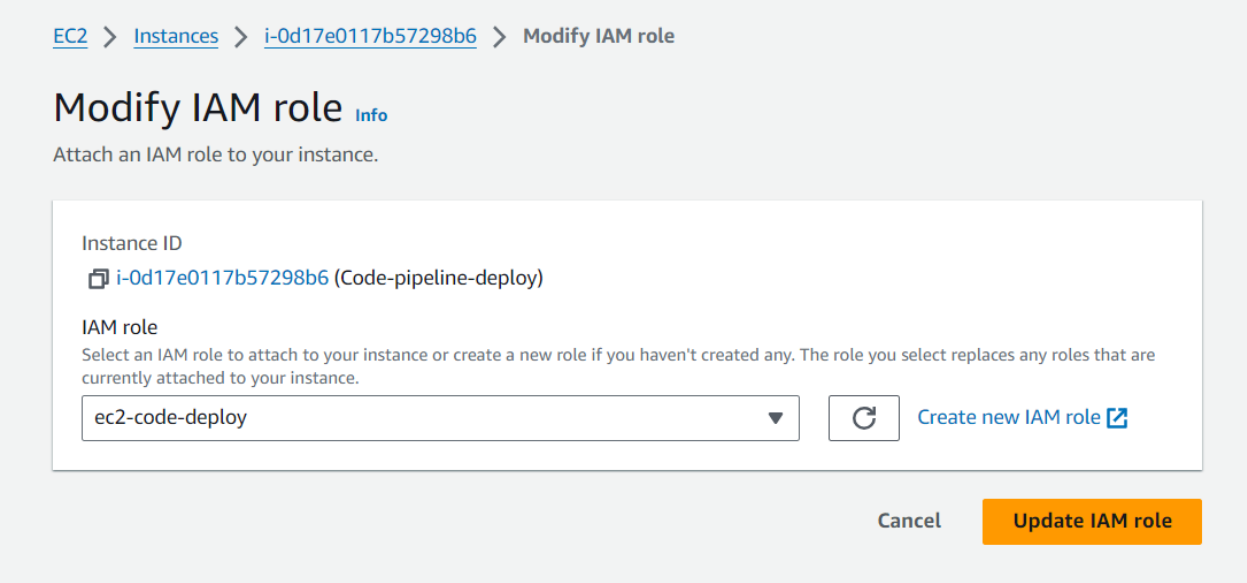


ii)Attach to EC2 instance:

- After creating this role attach it to your EC2-instance.

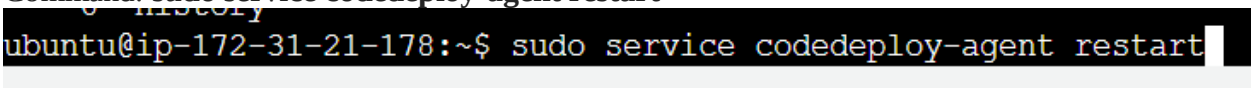


- Add the service role here to give required permissions to EC2-instance.



After this, restart the code deploy agent service in your instance.

Command: **sudo service codedeploy-agent restart**



iii)Adding App Specification File:

The Application Specification file (AppSpec file) is a YAML or JSON formatted configuration file utilized by CodeDeploy to orchestrate deployments. It contains instructions for CodeDeploy to manage the deployment process, including how to handle various lifecycle events, such as application installation, code deployment, and cleanup tasks.

iv)Create Required Files:

- Add the **'appspec.yml'** file with other required files to our code commit repository.
- $vim appspec.yml

```
version: 0.0
os: linux
files:
  - source: /
    destination: /var/www/html
hooks:
  AfterInstall:
    - location: scripts/install_nginx.sh
      timeout: 300
      runas: root
  ApplicationStart:
    - location: scripts/start_nginx.sh
      timeout: 300
      runas: root
~
~
~
~
appspec.yml [unix] (00:26 01/04/2024)
```

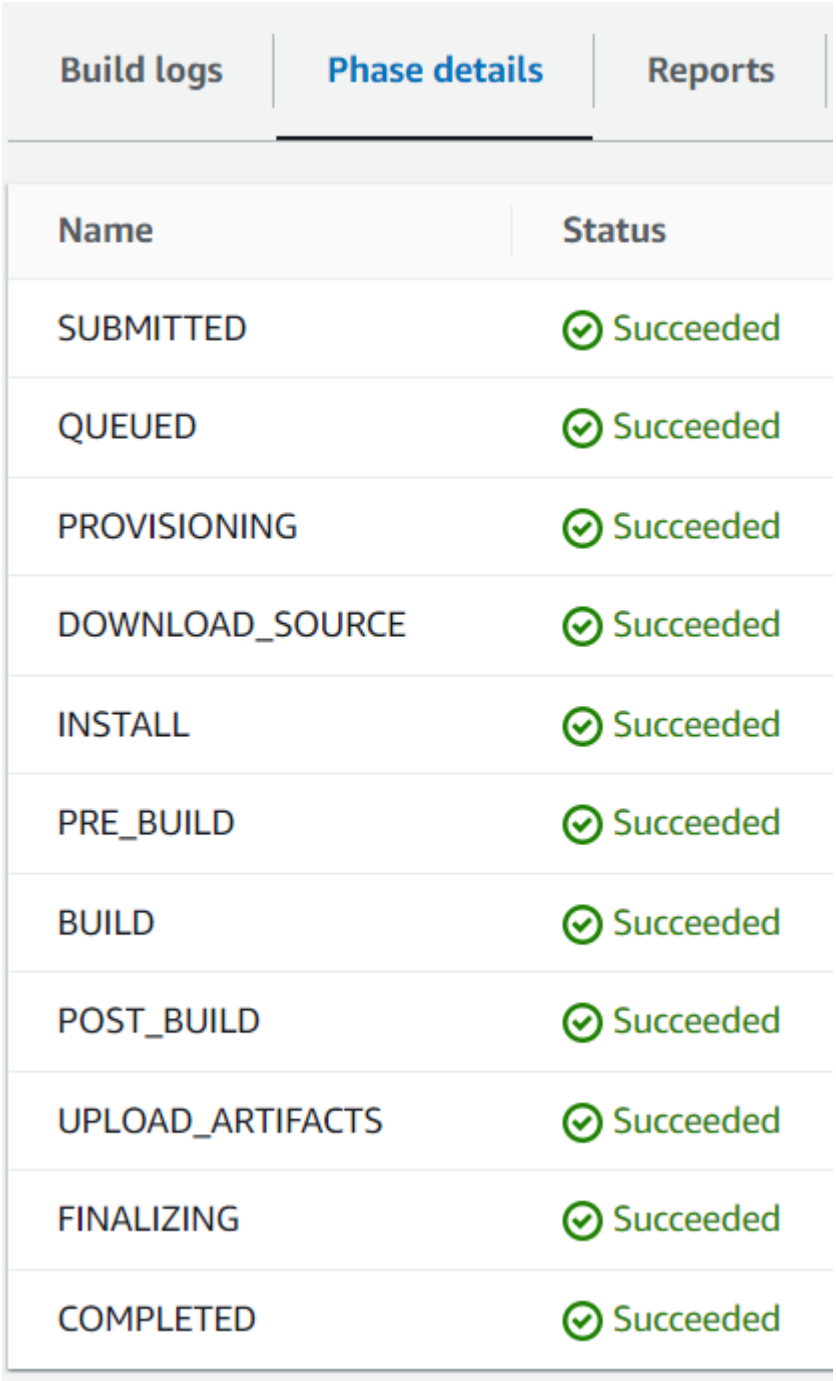
- Also add some scripts in the script folder that will perform the requires task on the instance.
- Commands:
  - $ mkdir scripts/ (Under Devops-on-AWS folder)
  - $ cd scripts/
  - $ cat install_nginx.sh
  - $ cat start_nginx.sh

- Add and commit all the changes:

```
$ ls
appspec.yml  buildspec.yml  index.html  scripts/
```

```
git add .
git commit -m "add appspec.yml"
git push
```

v)Start the Build:

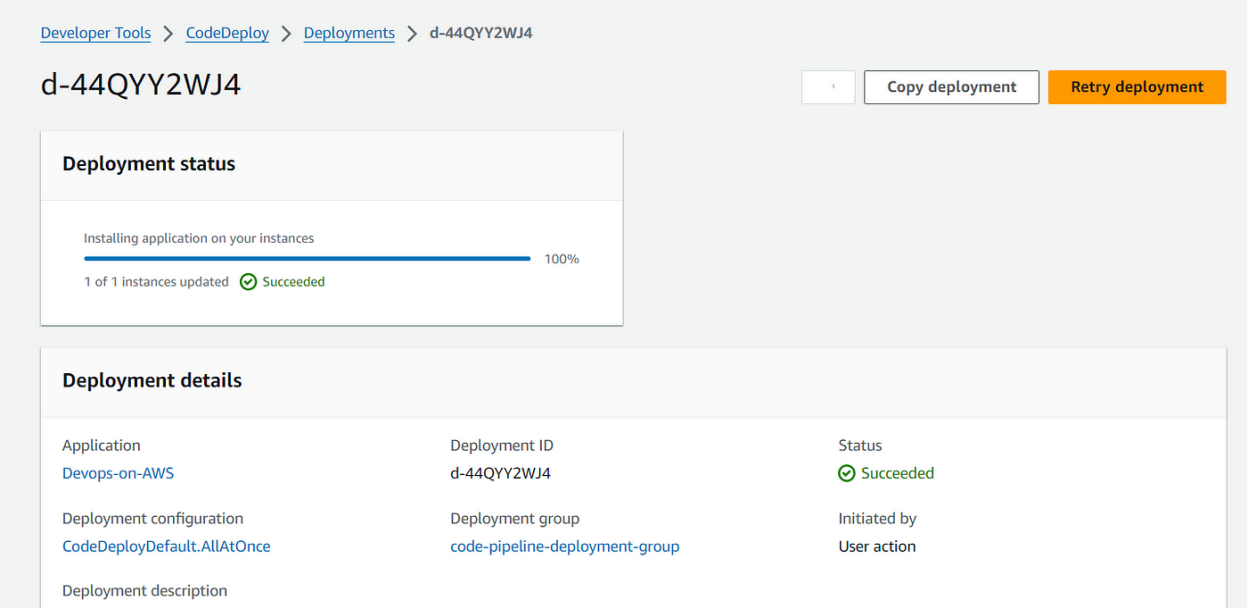


- Now your artifact is uploaded to the targeted bucket and path.

vi)Deploy your application:

- Start the deployment.

- CodeDeploy will automatically deploy your application to the specified EC2 instances, ensuring minimal downtime and rollback capabilities.



RESULT:

- Access your EC2 instance by copying its IP address and pasting it into your web browser. This action allows you to view the content of your **'index.html'** file hosted on the instance.

Best Practices:

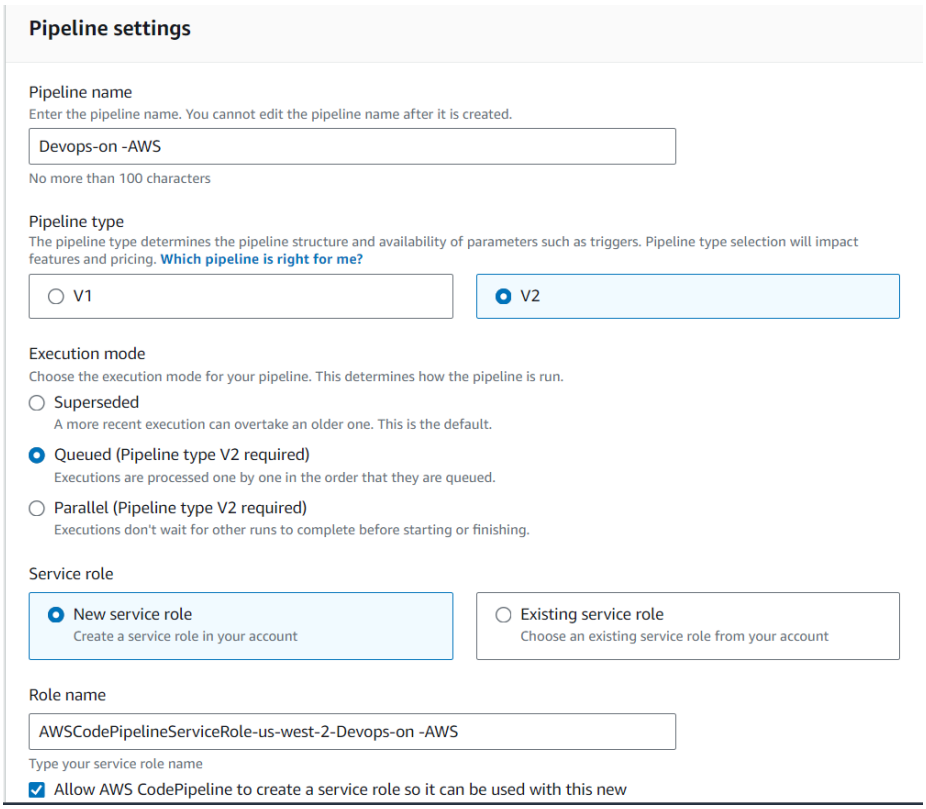
- Use blue-green deployments to minimize downtime and risk during deployments.
- Leverage deployment hooks to run custom scripts before and after deployment.
- Monitor deployment health using **CloudWatch** metrics and alarms.

7. Creating a Complete CI/CD Pipeline with CodePipeline:

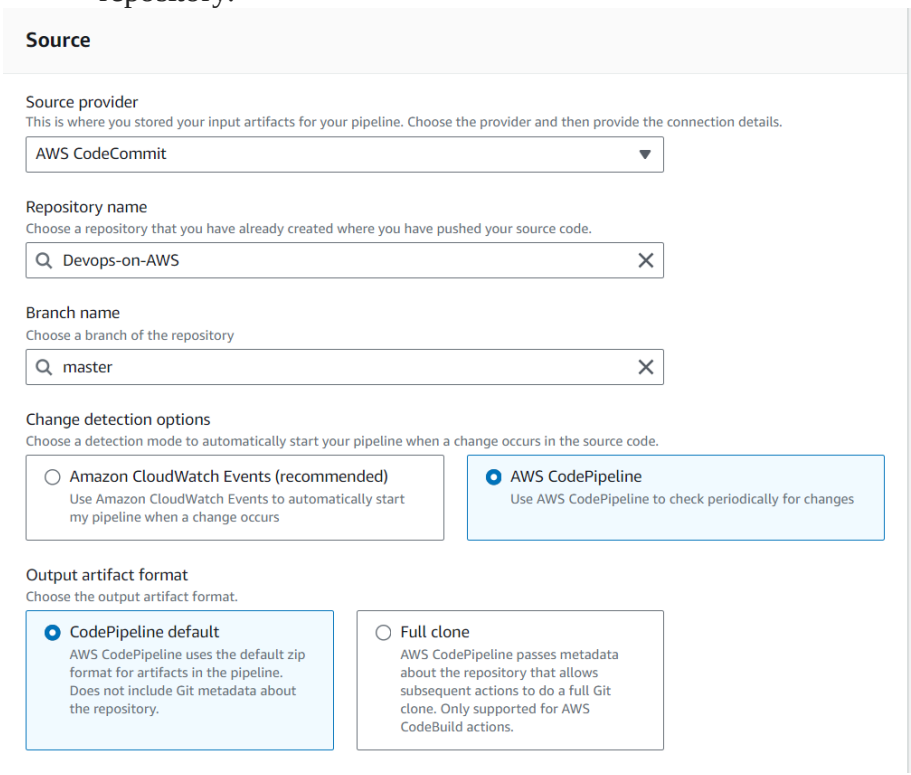
CodePipeline: AWS CodePipeline is a comprehensive continuous integration and continuous delivery service that automates the build, test, and deployment stages of your software release process. Setting up a CI/CD pipeline involves defining the workflow for your application's lifecycle, including source code management, build automation, testing, and deployment, all orchestrated seamlessly within CodePipeline.

i)Create a pipeline:

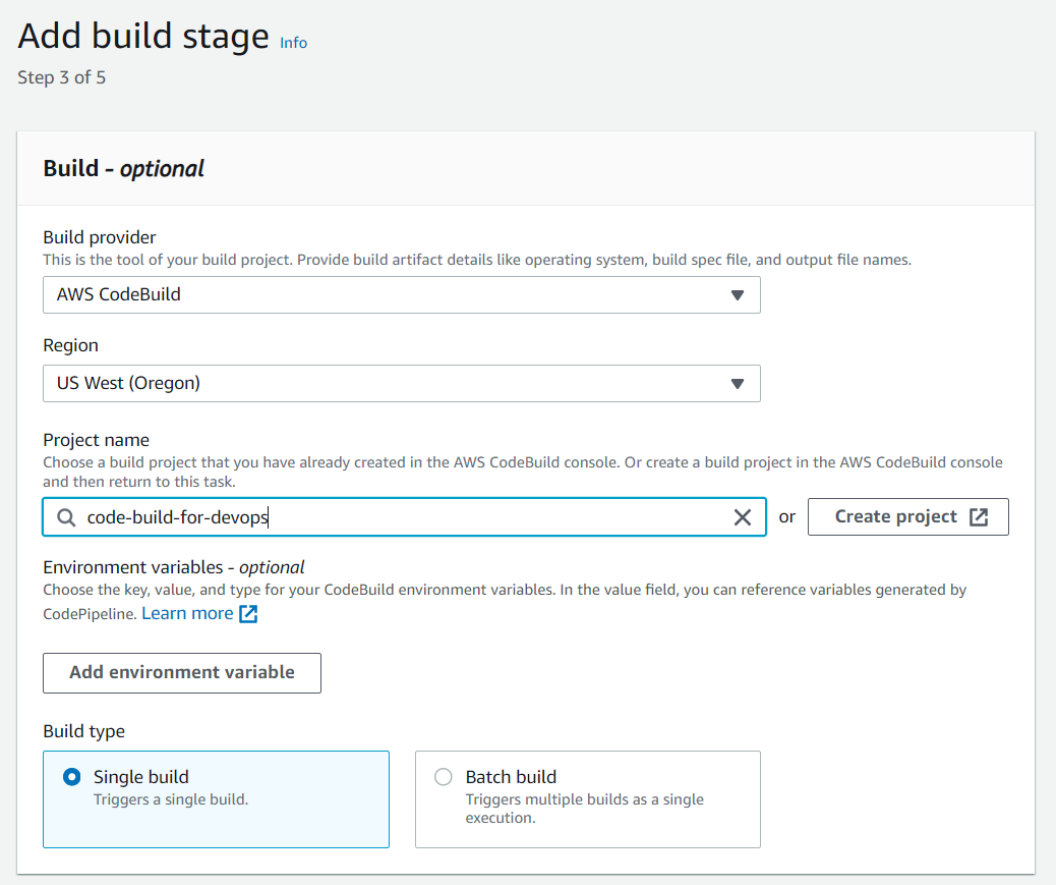
- Navigate to the '**CodePipeline'** service in the AWS Management Console.
- Click on **'Create pipeline'** and provide a name for your pipeline.



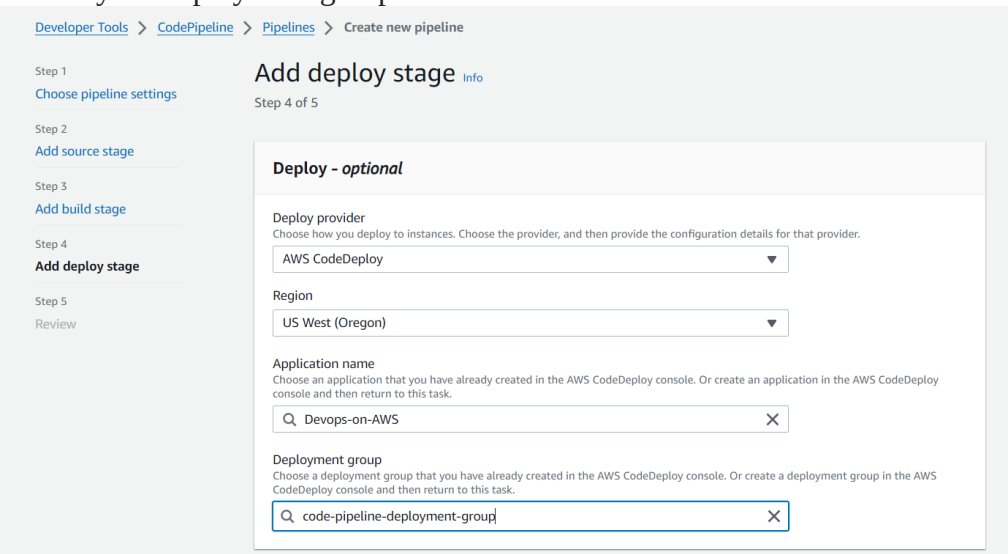
- **Define the source stage:** Choose **'CodeCommit'** as the source provider and select your repository.



- **Define the build stage:** Choose **'CodeBuild'** as the build provider and select your build project.



- **Define the deploy stage:** Choose **'CodeDeploy'** as the deployment provider and specify your deployment group.



- **Review and create it:** Review the pipeline configuration and click **'Create pipeline'** to initiate the pipeline creation process.

i)Pipeline Execution:

Once the pipeline is created, it will automatically trigger when you commit to your code commit repo.

Pipeline execution flow works:

1. Source stage:

- **CodePipeline** retrieves the source code from the specified **CodeCommit** repository.

2. Build stage:

- **CodePipeline** triggers the **CodeBuild** project, which compiles the source code, runs tests, and produces build artifacts.

3. Deploy stage:

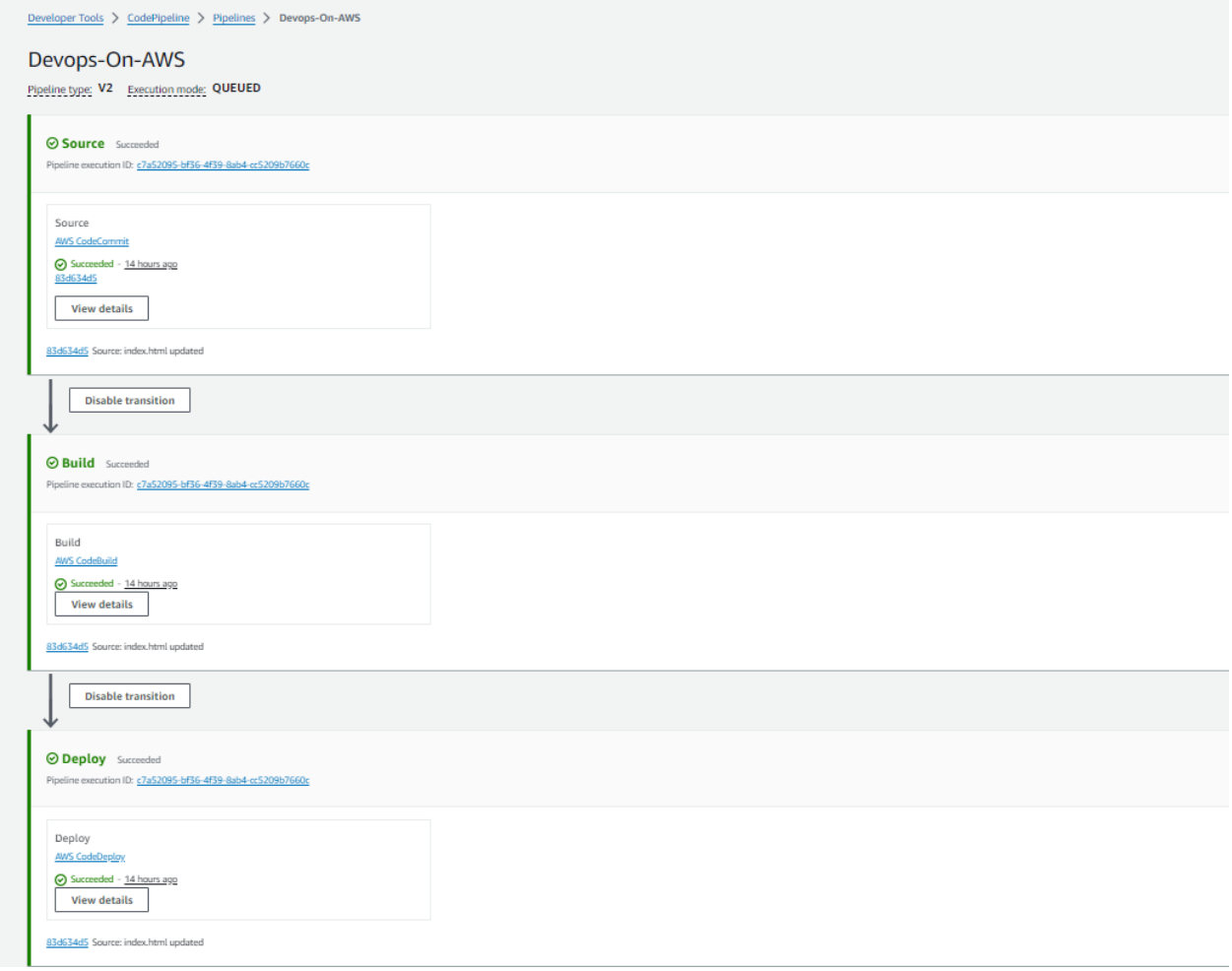
- **CodePipeline** deploys the build artifacts to the specified **CodeDeploy** deployment group, automating the deployment process to EC2 instances.

4. Monitor and manage:

- Monitor pipeline execution status and view detailed execution logs in the CodePipeline console.
- Manage pipeline stages, actions, and settings as per your project requirements.

VERIFICATION:

- Committing and pushing changes to the **'index.html'** file in the **CodeCommit** repository will trigger the CI/CD pipeline. This event initiates the automated workflow defined in the pipeline, encompassing processes like build, test, and deployment, ensuring that the latest changes are seamlessly integrated and deployed to the target environment.



You've successfully created a complete CI/CD pipeline using AWS CodePipeline. By automating the build, test, and deployment phases of your release process, you can accelerate software delivery and improve overall efficiency.