AdvDevops Case Study 1: Cloud Deployment with Automation

- **Concepts Used:** EC2, S3, CodeBuild, CodePipeline, CodeDeploy.
- **Problem Statement:** "Build a simple web application using AWS CodeBuild and deploy it to an S3 bucket. Then, automate the deployment process using AWS CodePipeline, ensuring the application is deployed on an EC2 instance. Use a sample index.html page for demonstration."
- **Tasks:**
  - Set up AWS CodeBuild for the web app.
  - Create a pipeline that deploys to an S3 bucket.
  - Use AWS CodeDeploy to push updates to an EC2 instance.

1. Introduction

Case study overview:

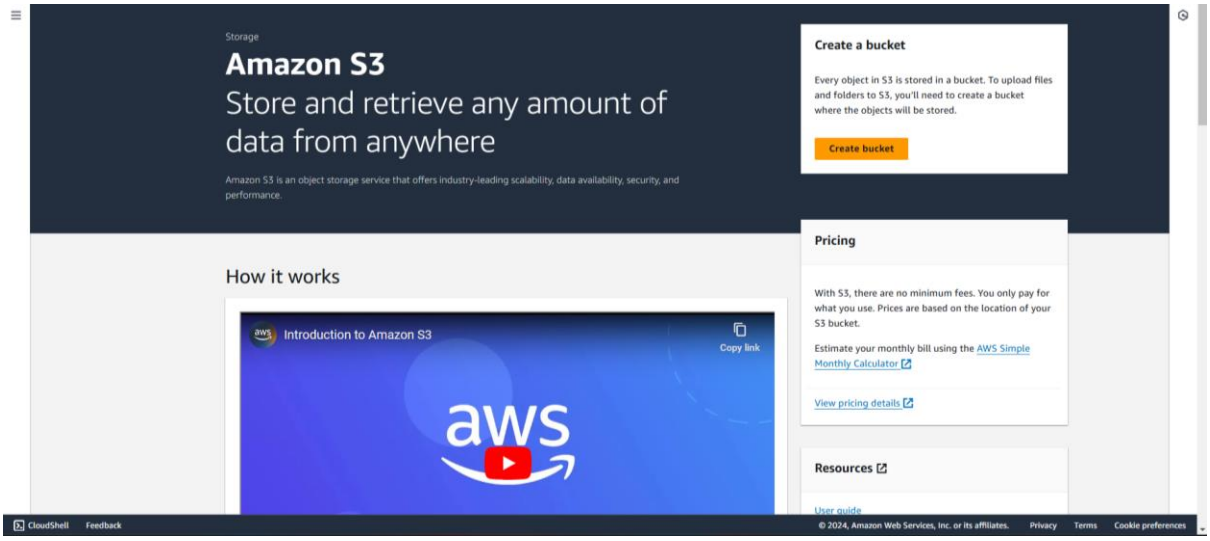
This case study involves storing a web application in an S3 bucket, building it through AWS CodeBuild, automation using AWS CodePipeline to deploy the web app to an EC2 instance via CodeDeploy.

Key features and Applications:

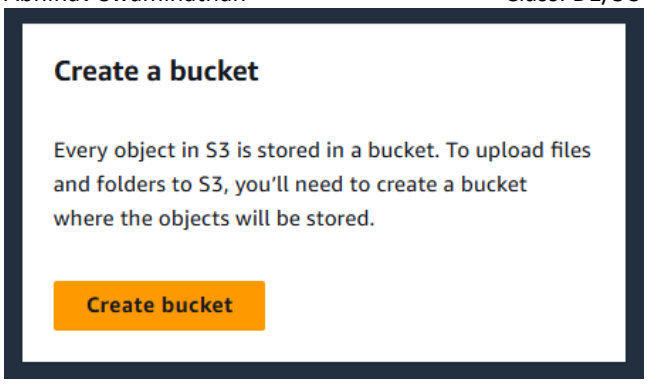
Automated Deployment: AWS CodePipeline automates code deployment to S3 and EC2.
CI/CD Integration: Continuous integration and deployment with CodeBuild and CodeDeploy streamlines updates.
Scalability: EC2 provides scalable compute power and S3 offers scalable storage.
Version Control: CodeDeploy enables easy rollback to previous versions in case of failure
Web App Hosting: Ideal for hosting dynamic and static web apps.

Step 1: Create an S3 Bucket

In the AWS Management Console, navigate to **S3** by searching for it in the search bar at the top.

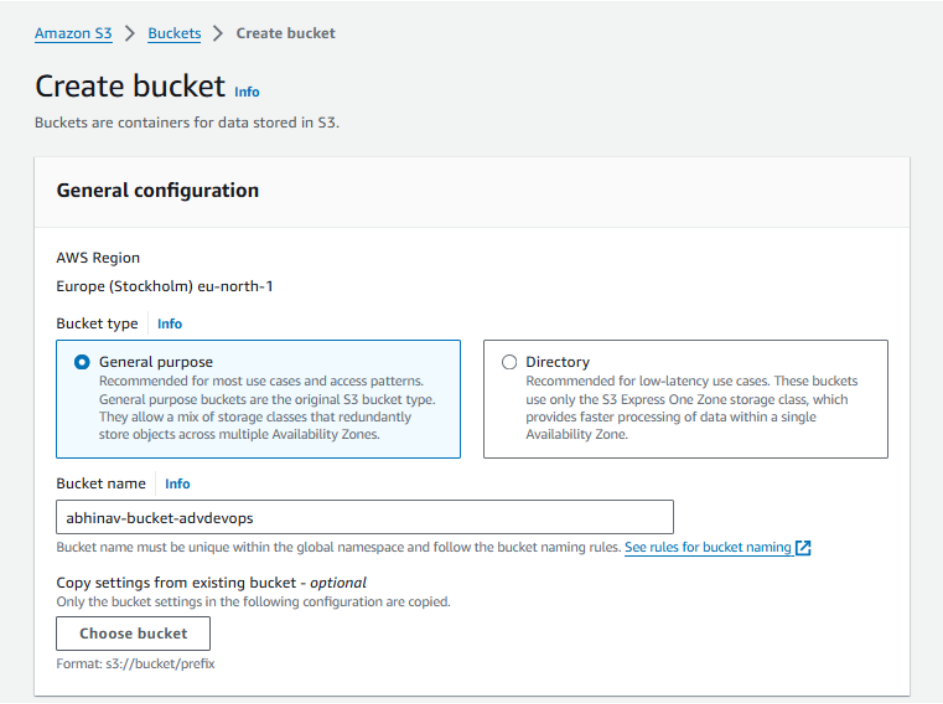


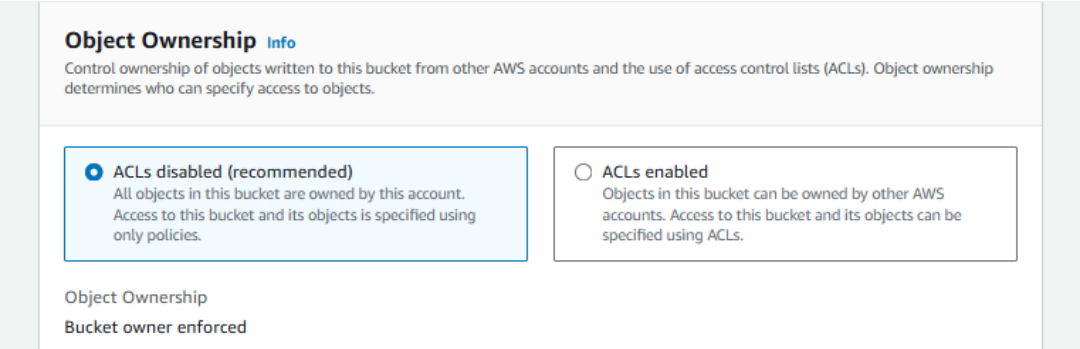
Click Create bucket.

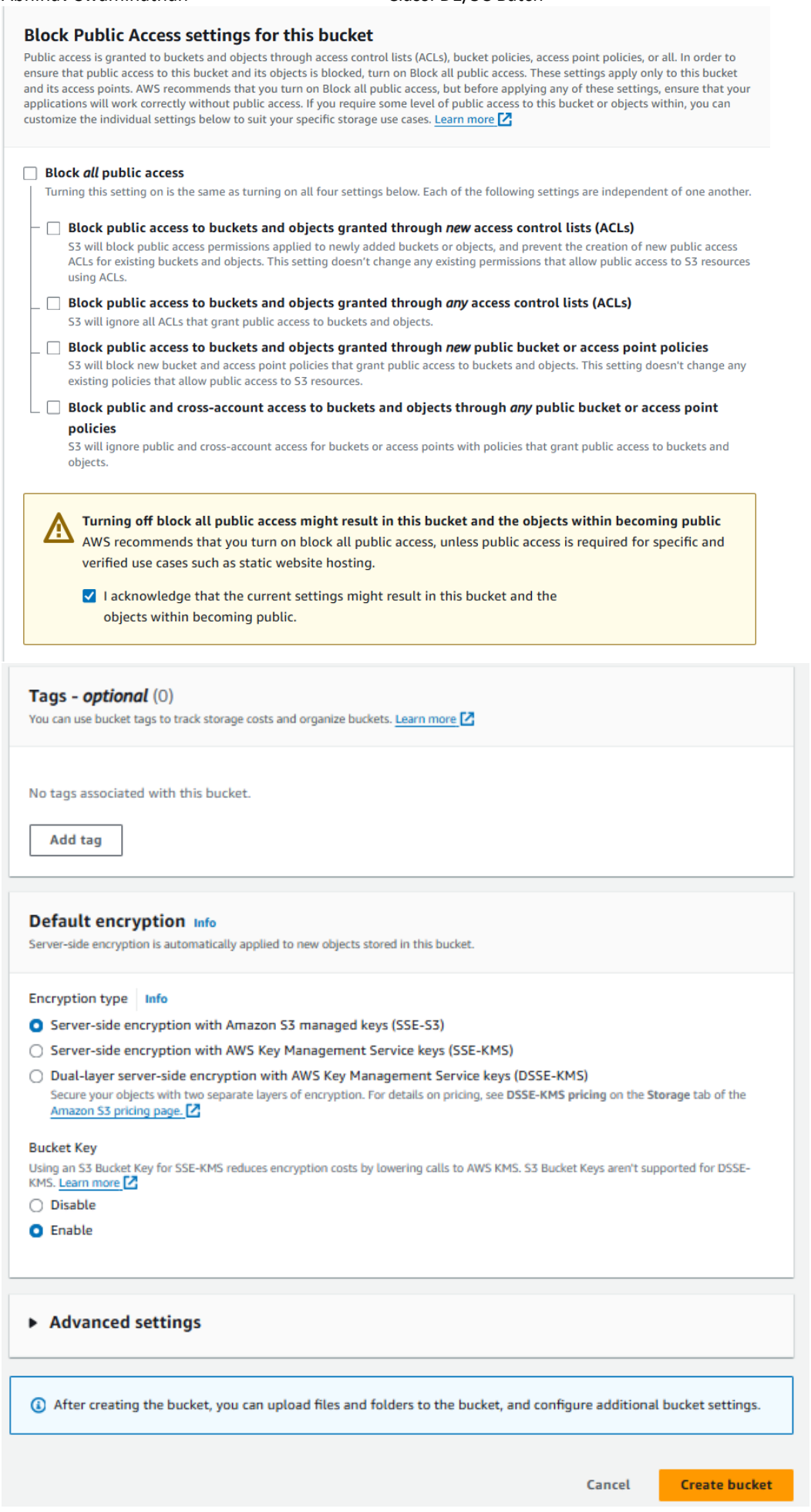


Give your bucket a unique name (e.g., my-app-bucket). If you want the content of your web app (like the index.html file) to be publicly accessible, uncheck Block all public access.

For testing/deployment, you can allow public access





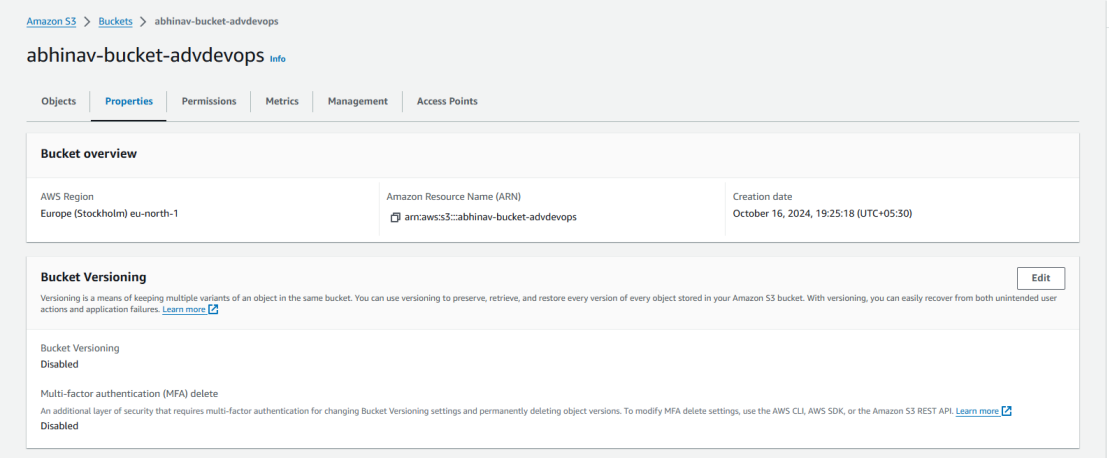

Block Public Access settings for this bucket
Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to this bucket or objects within, you can customize the individual settings below to suit your specific storage use cases. Learn more
Block all public access
Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.
Block public access to buckets and objects granted through new access control lists (ACLs)
S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.
Block public access to buckets and objects granted through any access control lists (ACLs)
S3 will ignore all ACLs that grant public access to buckets and objects.
Block public access to buckets and objects granted through new public bucket or access point policies
S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.
Block public and cross-account access to buckets and objects through any public bucket or access point policies
S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.
Turning off block all public access might result in this bucket and the objects within becoming public
AWS recommends that you turn on block all public access, unless public access is required for specific and verified use cases such as static website hosting.
I acknowledge that the current settings might result in this bucket and the objects within becoming public.
Tags - optional (0)
You can use bucket tags to track storage costs and organize buckets. Learn more
No tags associated with this bucket.
Add tag
Default encryption Info
Server-side encryption is automatically applied to new objects stored in this bucket.
Encryption type Info
Server-side encryption with Amazon S3 managed keys (SSE-S3)
Server-side encryption with AWS Key Management Service keys (SSE-KMS)
Dual-layer server-side encryption with AWS Key Management Service keys (DSSE-KMS)
Secure your objects with two separate layers of encryption. For details on pricing, see DSSE-KMS pricing on the Storage tab of the Amazon S3 pricing page.
Bucket Key
Using an S3 Bucket Key for SSE-KMS reduces encryption costs by lowering calls to AWS KMS. S3 Bucket Keys aren't supported for DSSE-KMS. Learn more
Disable
Enable
Advanced settings
After creating the bucket, you can upload files and folders to the bucket, and configure additional bucket settings.
Cancel
Create bucket

Enable Versioning on Your S3 Bucket

To do this

Select Your Bucket: Click on your bucket named (abhinav-bucket-advdevops.)

Go to Properties: In the bucket settings, click on the "Properties" tab.

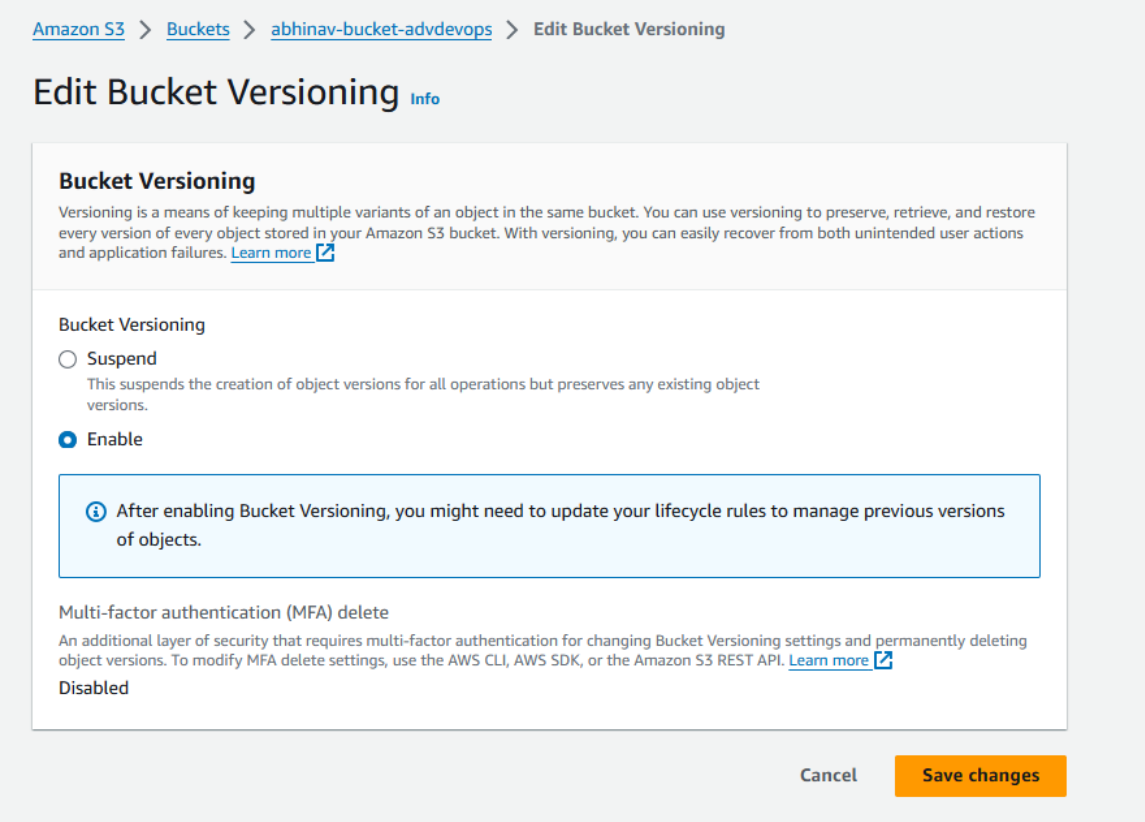


Scroll down to the "Bucket Versioning" section.

Click on "Edit".

Select "Enable" for versioning.

Click "Save changes".



Step 2: Create a Simple Web Application (index.html)

On your local machine, create a new file called index.html.

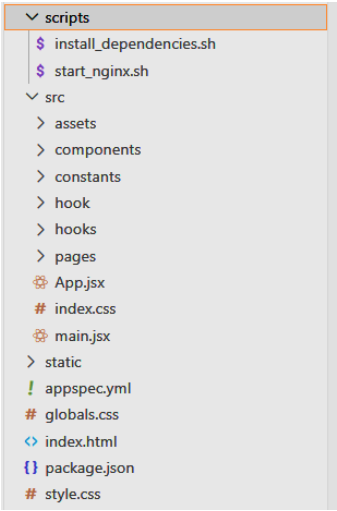
Add the following simple HTML code into the file: index.html

This is necessary for a basic web app. I implemented my Project Integration (shop.co)

index.html

File Edit View

```
<!DOCTYPE html>
<html>
<head>
  <title>Simple Web App</title>
</head>
<body>
  <h1>Hello, I am Abhinav</h1>
</body>
</html>
```



Create the following

appspec.yml (configuration file listing instructions to execute when the application is deployed)

```
version: 0.0
os: linux
files:
  - source: /
    destination: /usr/share/nginx/html
    overwrite: true
hooks:
  BeforeInstall:
    - location: scripts/install_dependencies.sh
      timeout: 300
      runas: ec2-user
  AfterInstall:
    - location: scripts/start_nginx.sh
      timeout: 300
      runas: ec2-user
```

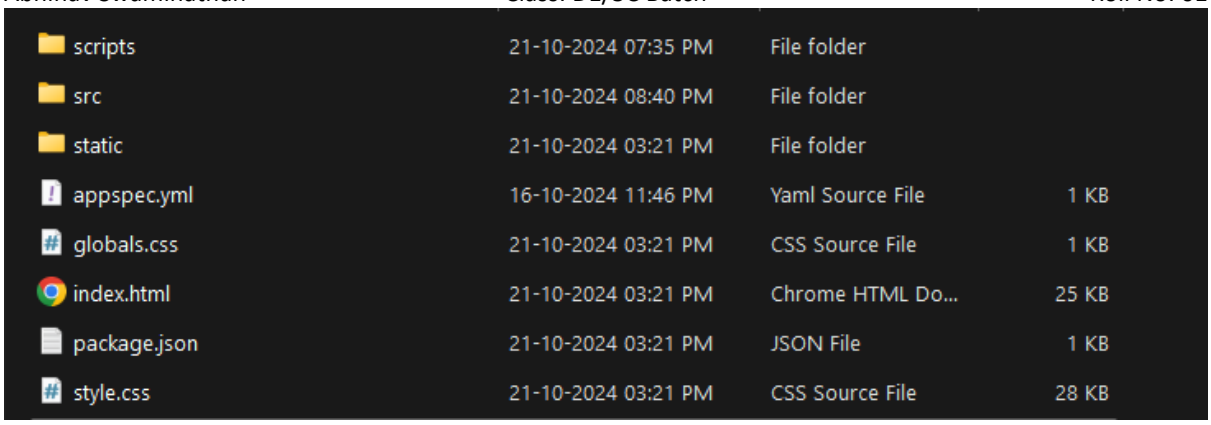
scripts\install_dependencies.sh (update your system & install nginx)

```
#!/bin/bash
sudo yum update -y
sudo yum install -y nginx
```

scripts\start_nginx.sh (restart nginx service)

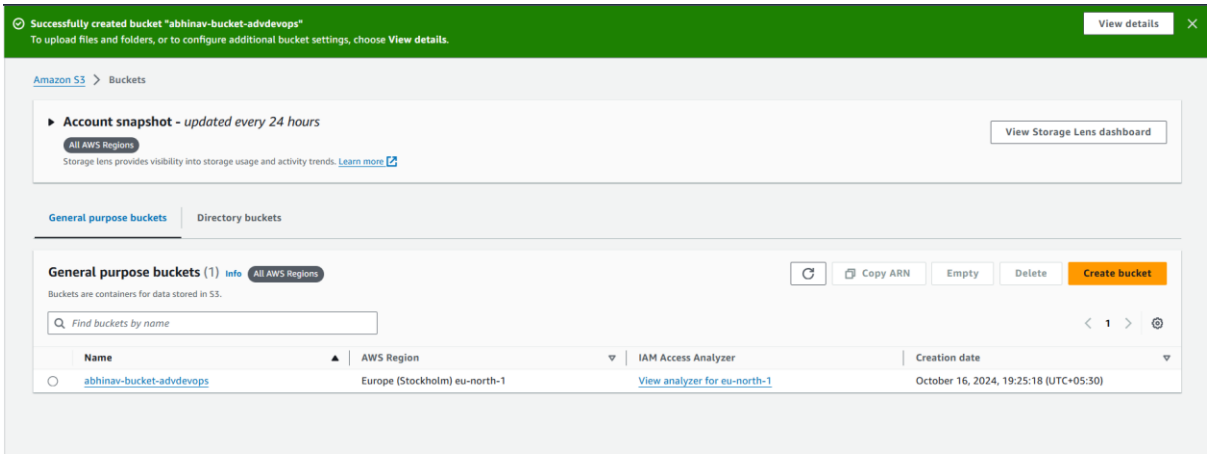
```
#!/bin/bash
sudo service nginx restart
```

Zip all into webapp.zip (project files for deployment).

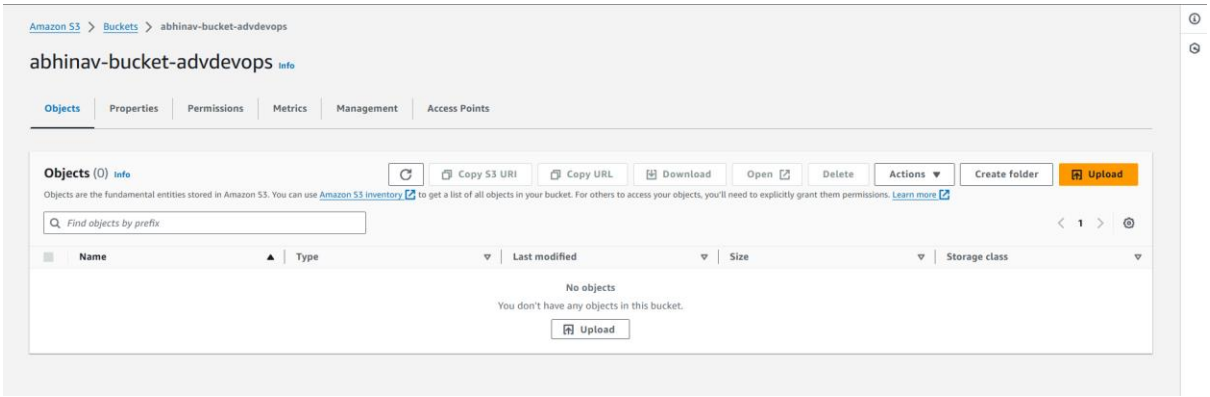


Step 3: **Upload the Web App (webapp.zip) to S3**

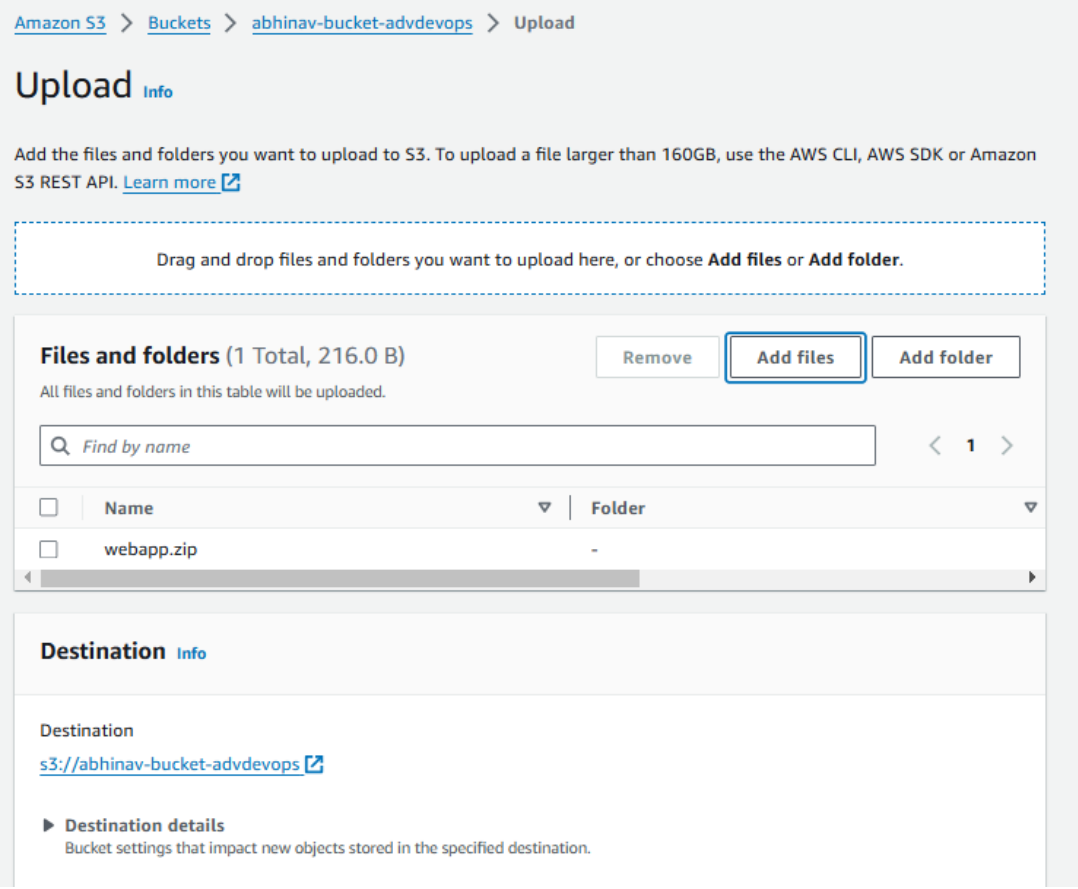
Go back to the S3 Console. Click on the bucket you created in Step 1

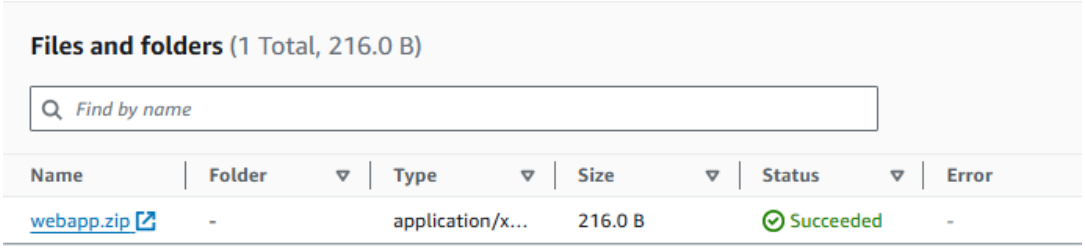


In the bucket, click **Upload**.



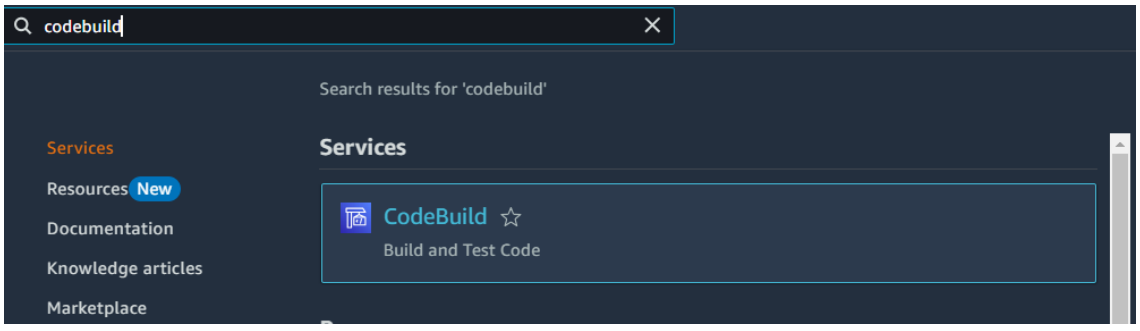
Click **Add files**, then select the webapp.zip file from your local machine. Leave all other options as default and click **Upload**.





Step 4: Set Up AWS CodeBuild for the Web App

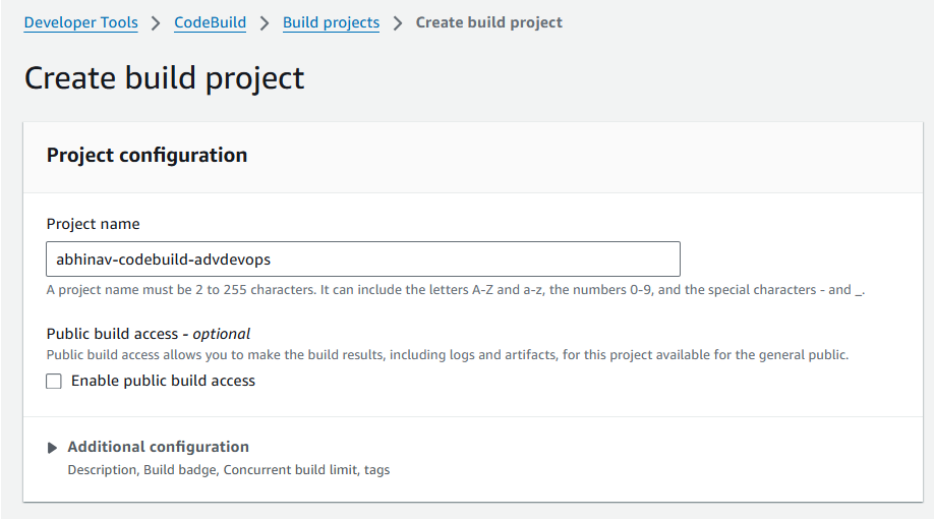
In the AWS Management Console, navigate to **CodeBuild** by searching for it in the top search bar.



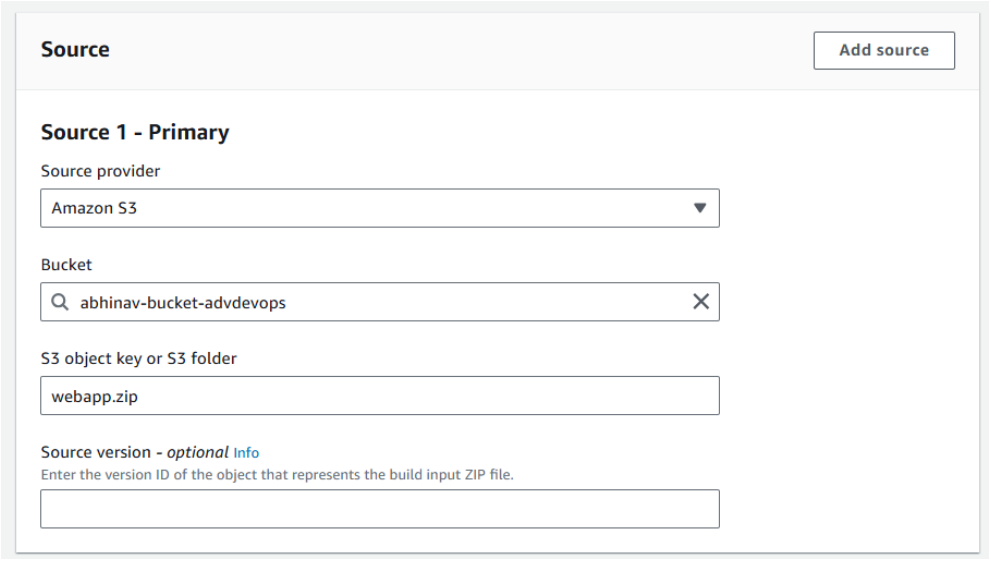
Click Create build project**.**

Project Configuration**:**

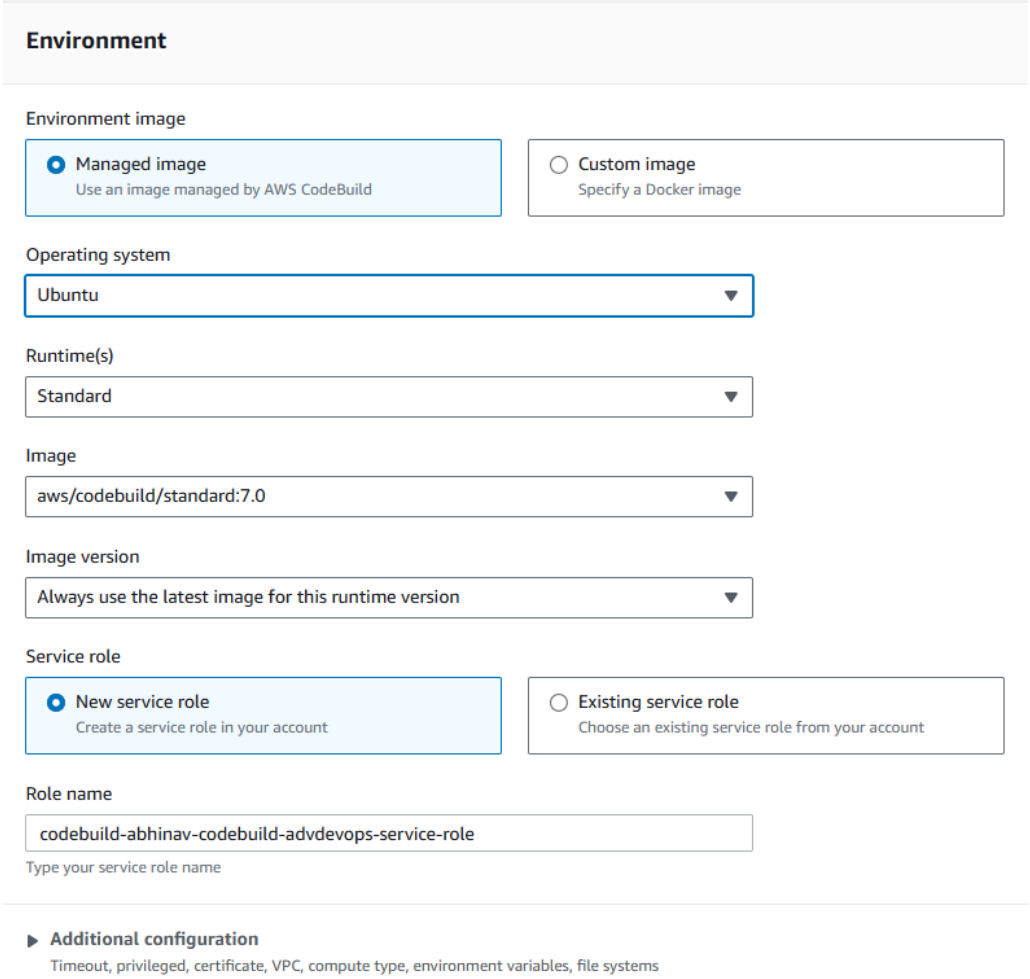
- Project Name**:** Enter a name



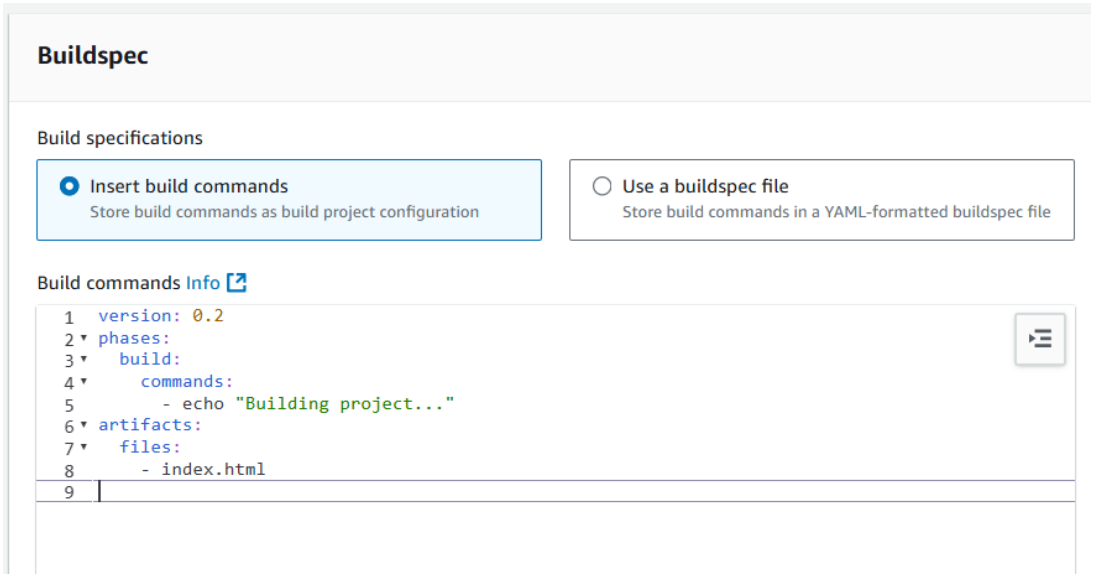
- Source Provider: Select Amazon S3**.**
- Bucket**:** Select the bucket (your bucket name) you created earlier.
- Object Key**:** Select the webapp.zip file you uploaded.



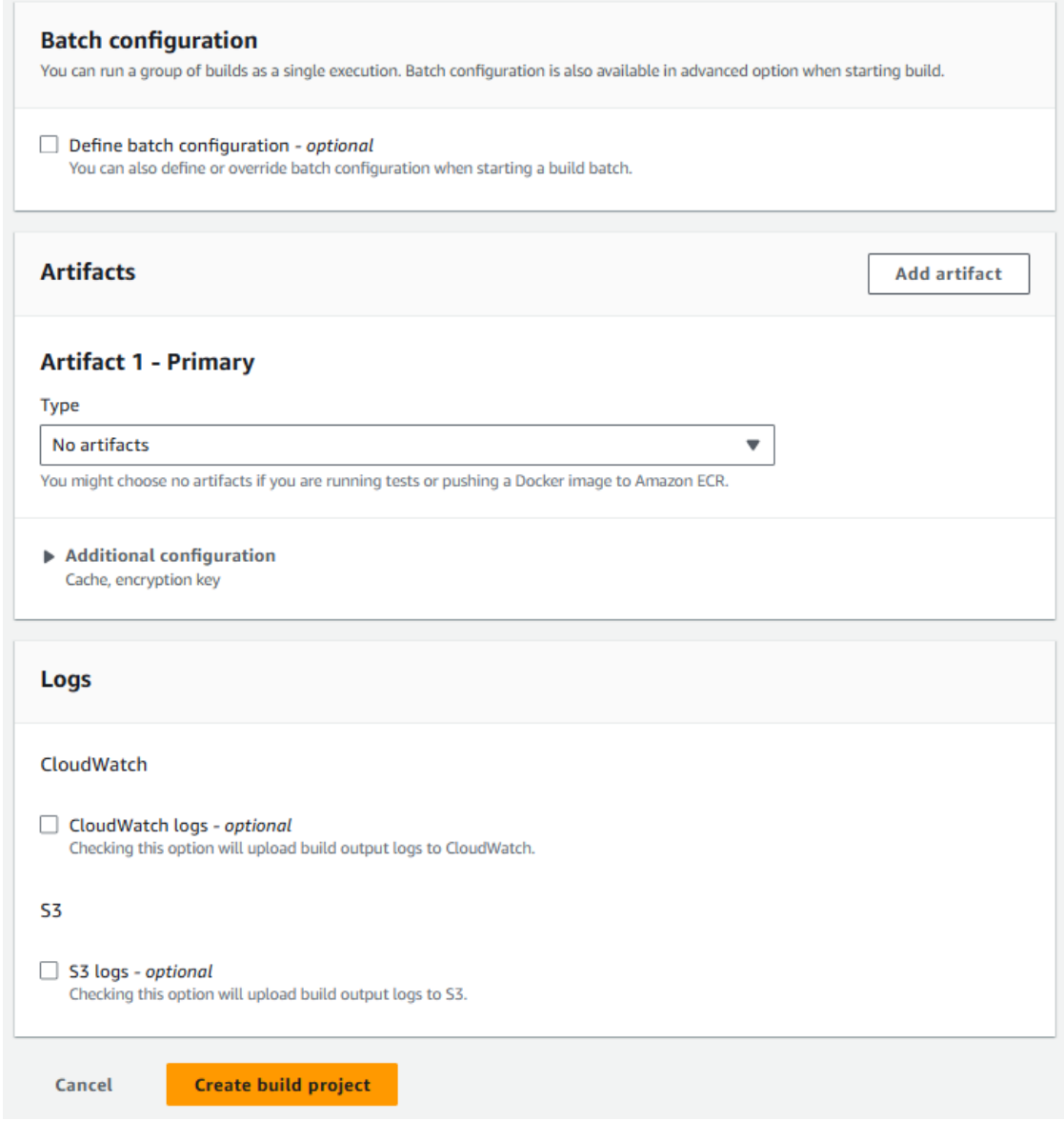
- Operating System: Select Ubuntu.

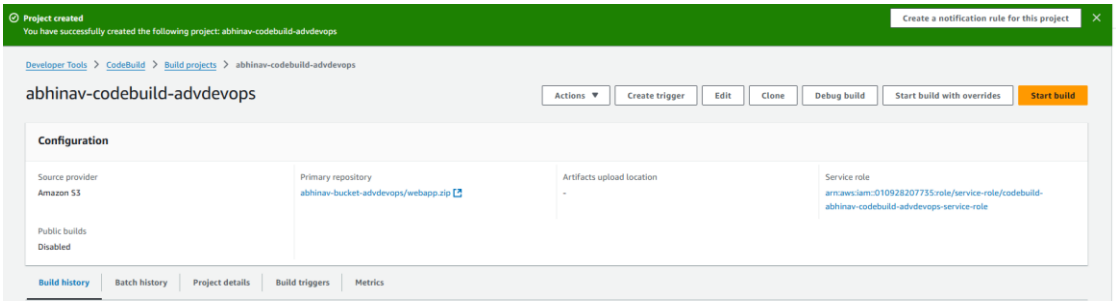


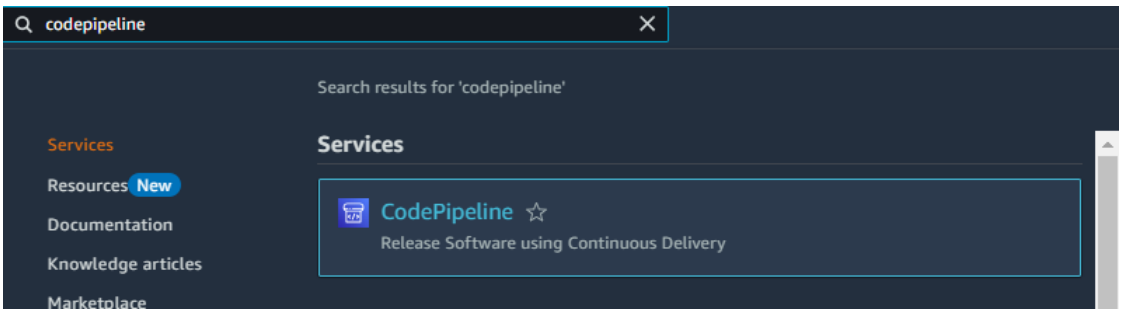
Buildspec**: Select "Insert build commands" and enter the following YAML script: (Switch to editor)**



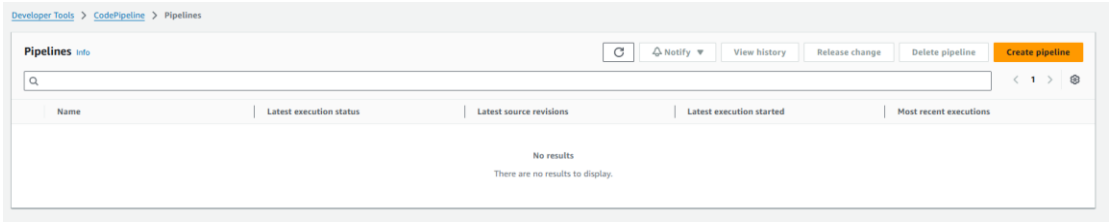
- Uncheck uploading logs to CloudWatch as it unnecessary

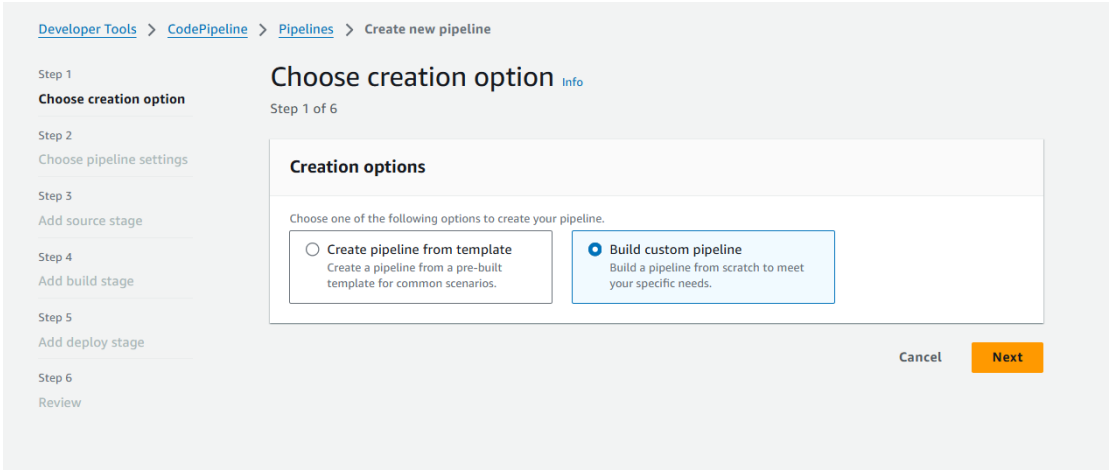




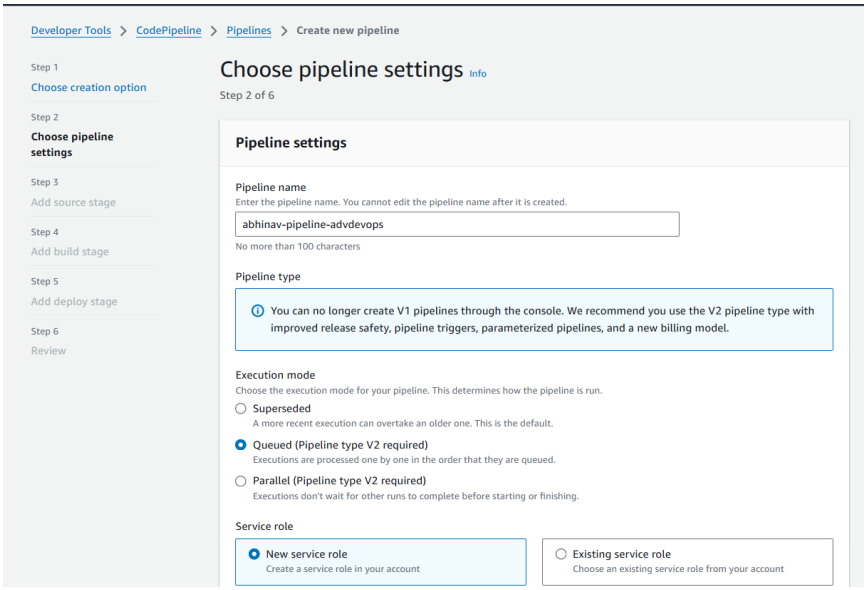


Click Create Pipeline.

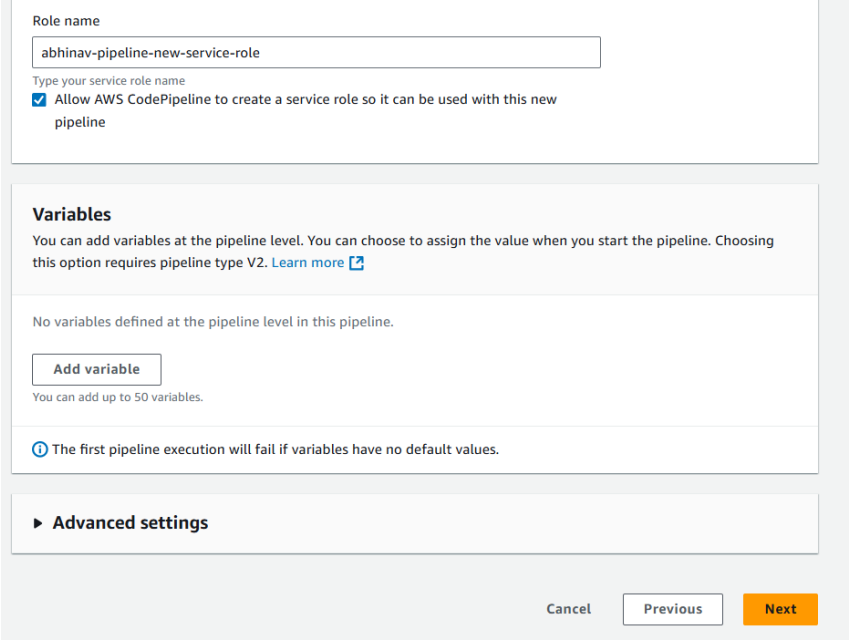




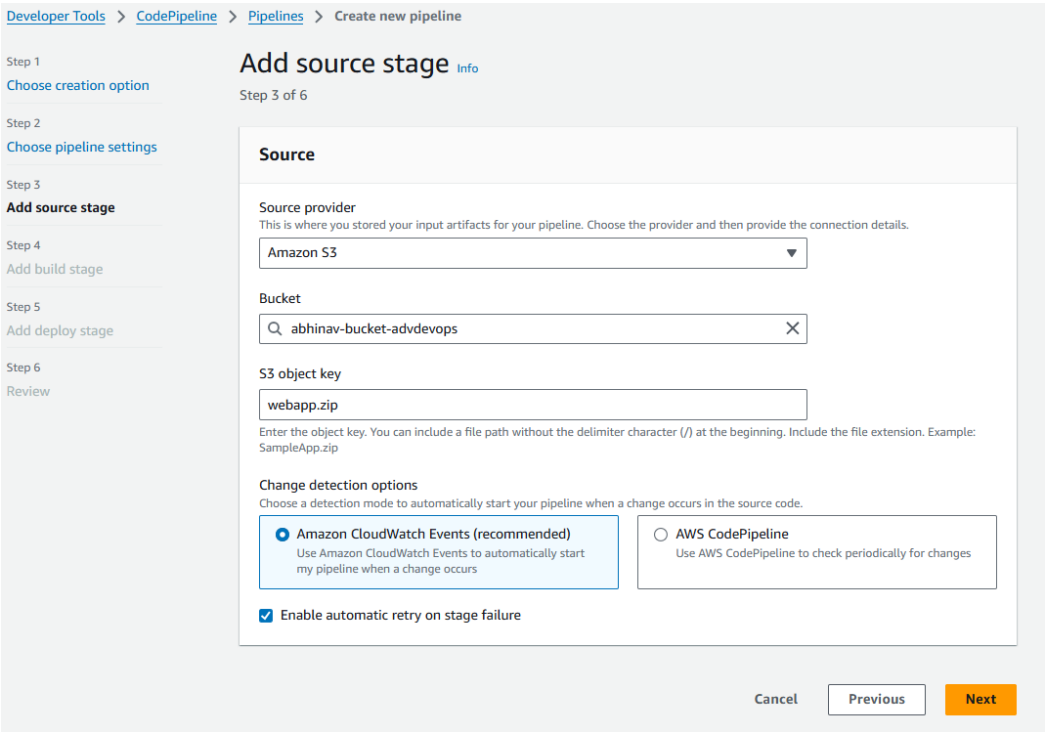
- Pipeline Name: Enter a name



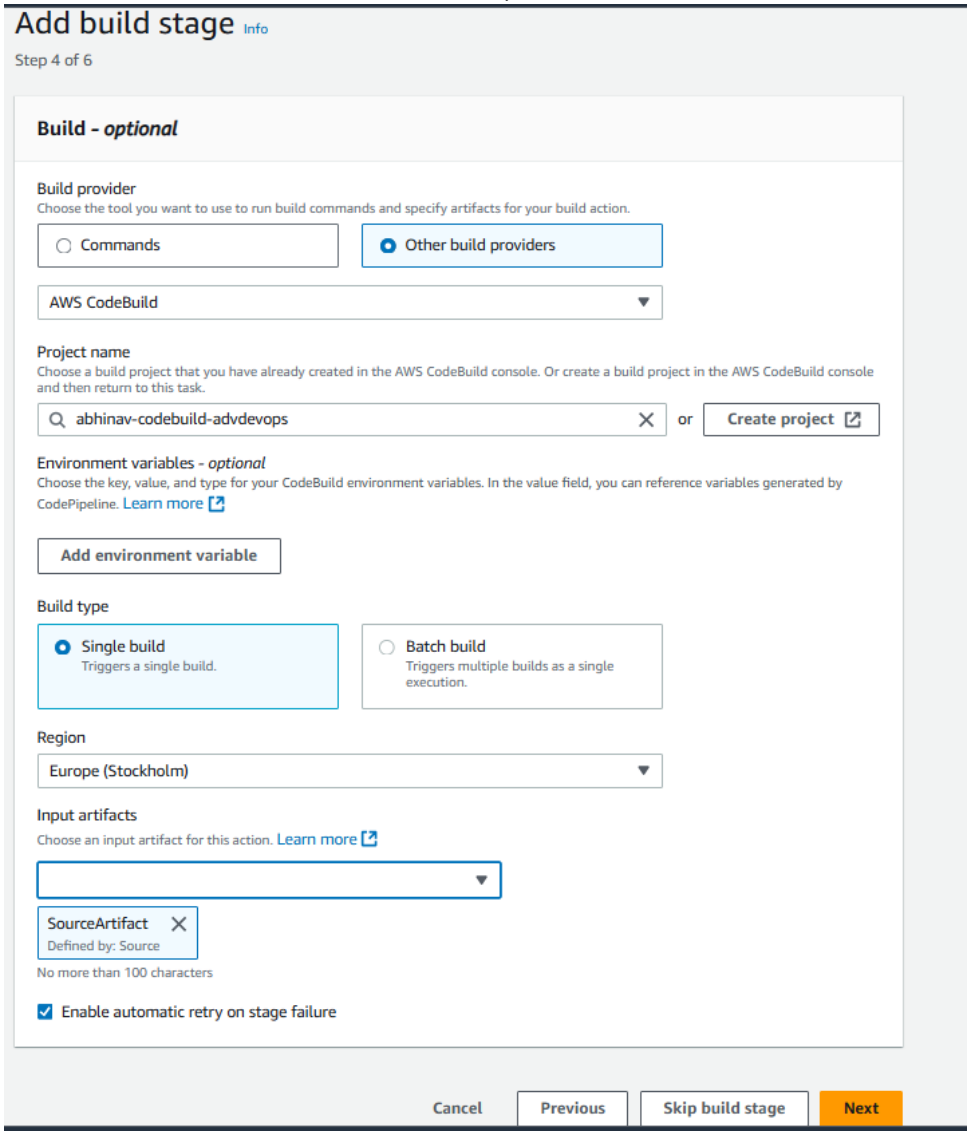
- Service Role: Select New service role to create a new role for this pipeline.



Source Provider: Select Amazon S3.
Bucket: Select the same S3 bucket you used earlier.
S3 object key: Enter webapp.zip as the source file.



Build Provider: Select Other Build Providers Then select AWS Codebuild
Project Name: Select the build project you created in Step 4



Deploy Provider: Select Amazon S3.
Bucket: Choose the same S3 bucket.
For Extract file before deploy, select Yes.

Add deploy stage Info

Step 5 of 6

Deploy - *optional*

Deploy provider

Choose how you deploy to instances. Choose the provider, and then provide the configuration details for that provider.

Amazon S3

Region

Europe (Stockholm)

Input artifacts

Choose an input artifact for this action. Learn more

BuildArtifact
Defined by: Build

No more than 100 characters

Bucket

abhinav-bucket-advdevops

Deployment path - *optional*

Extract file before deploy
The deployed artifact will be unzipped before deployment.

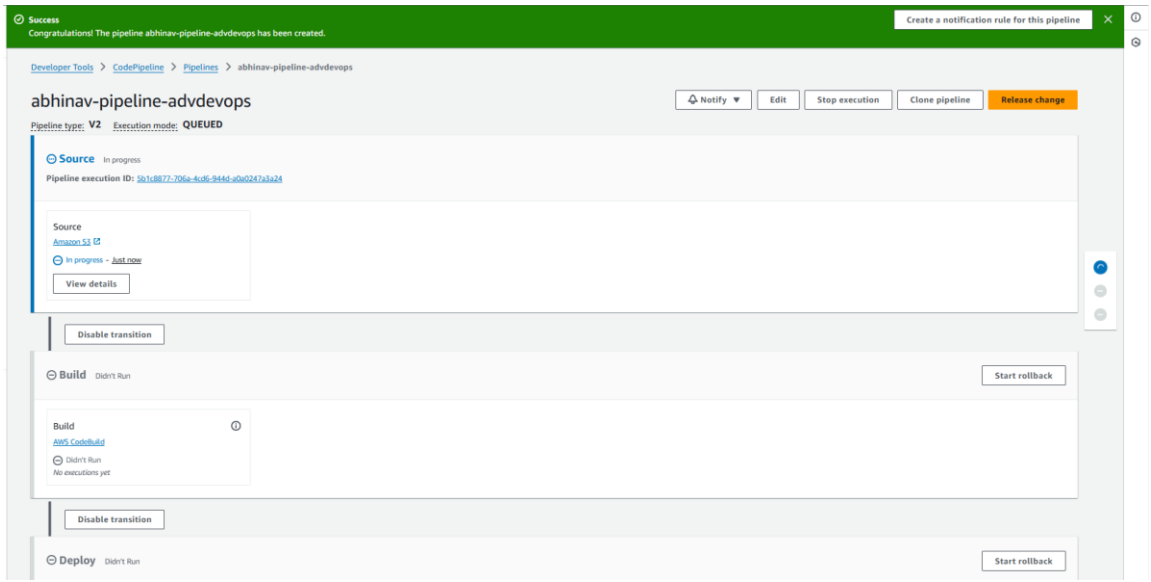
Additional configuration

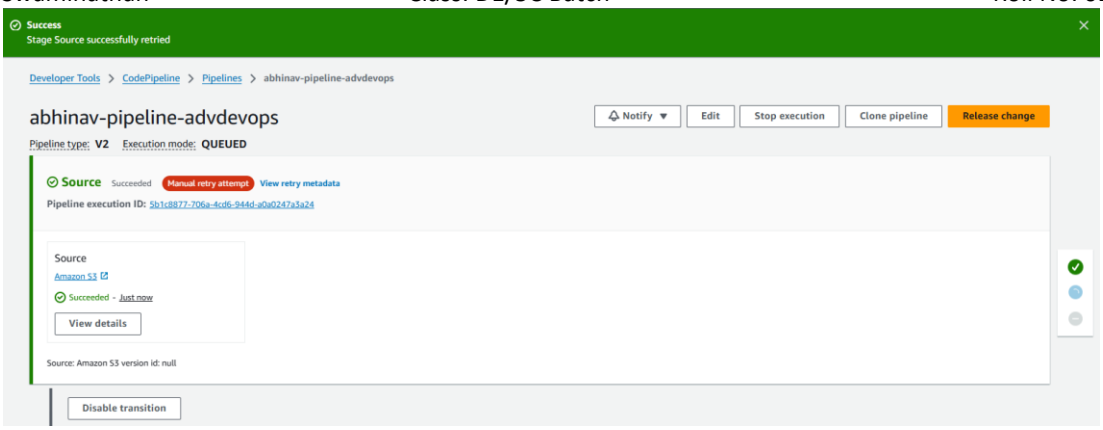
Configure automatic rollback on stage failure

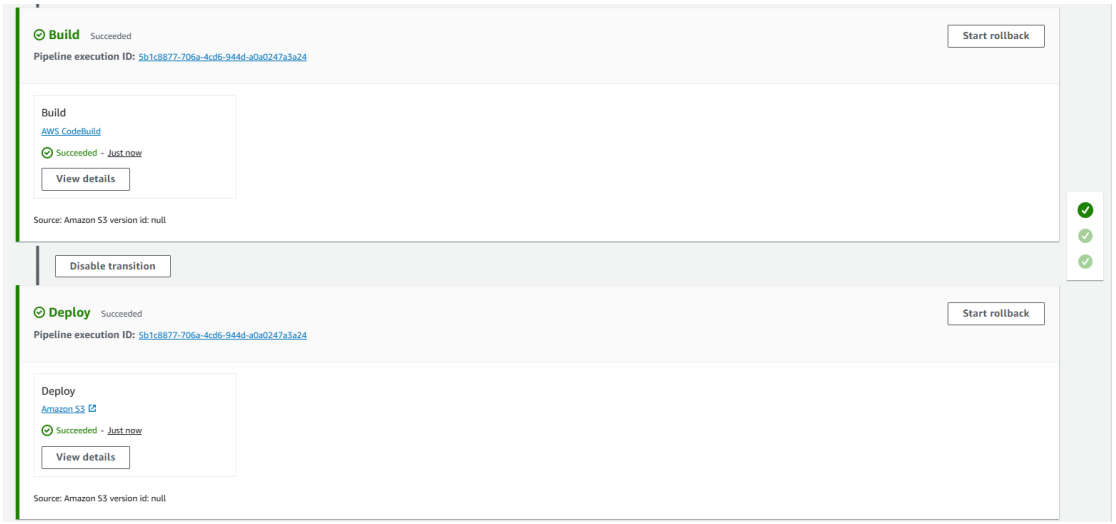
Enable automatic retry on stage failure

Cancel Previous Skip deploy stage Next

Review your bucket and create the pipeline



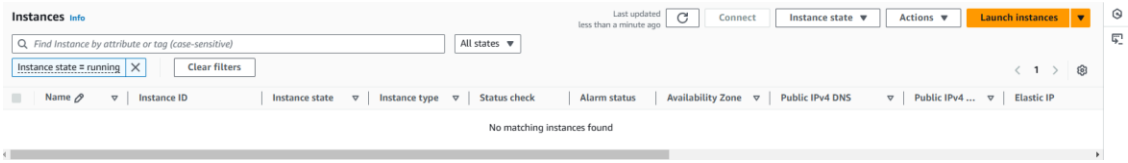


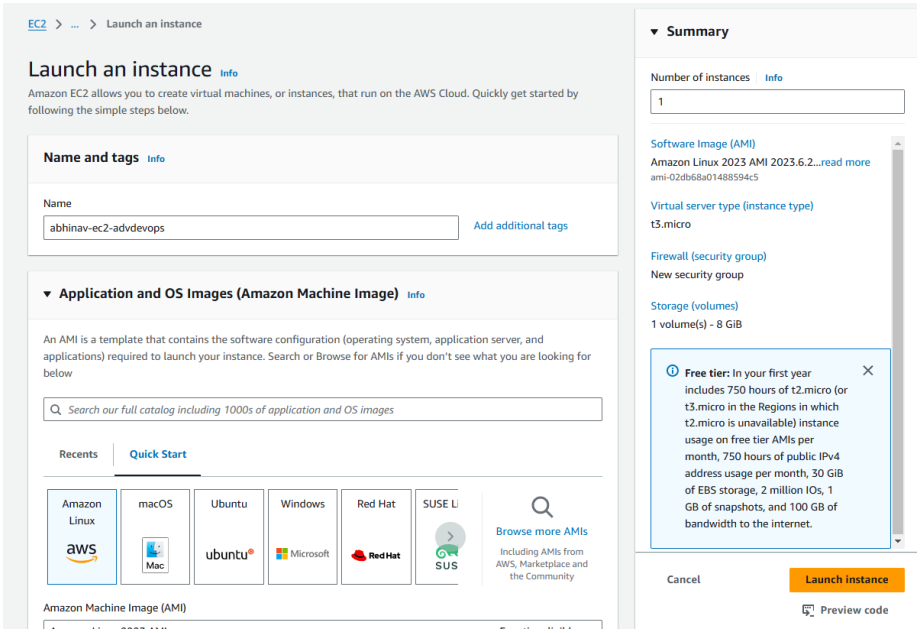


Step 6: Create an EC2 Instance

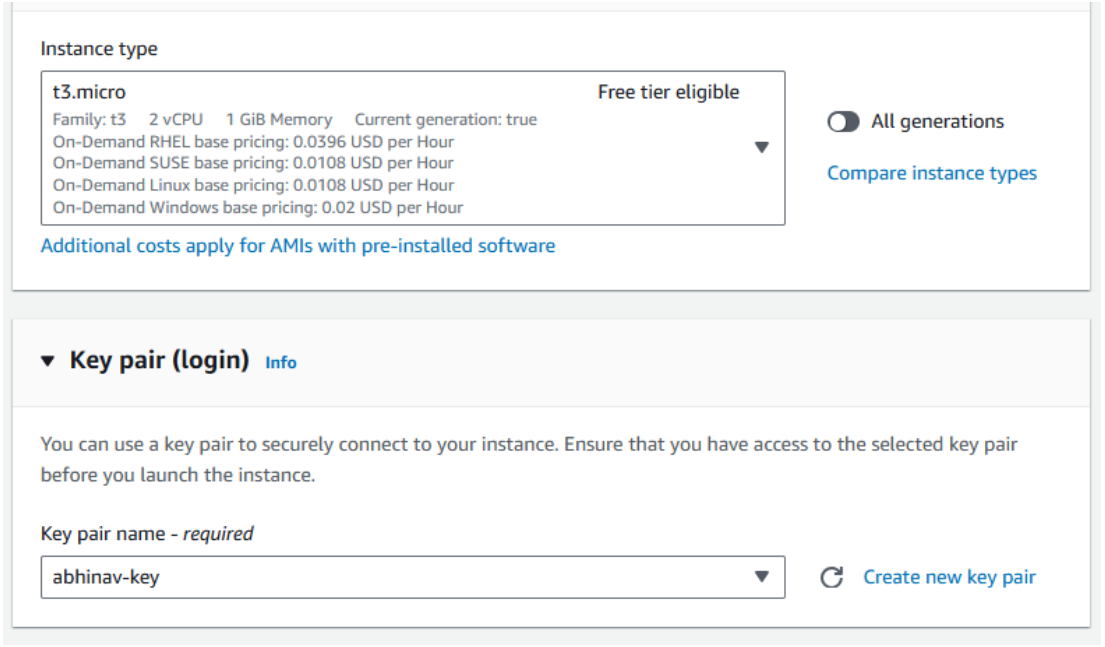
Click on **Launch Instance**.

Select an instance type (e.g., t3.micro for free tier eligibility).

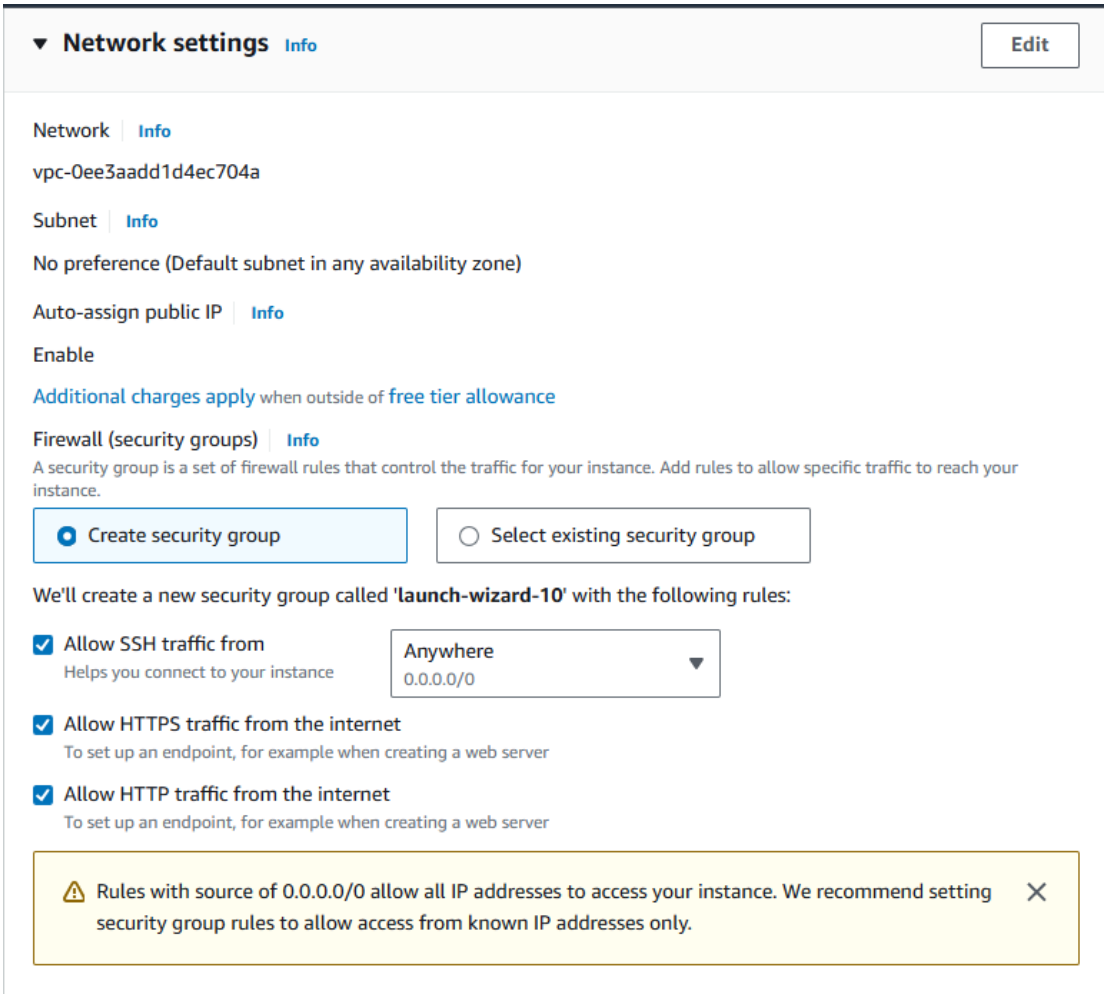


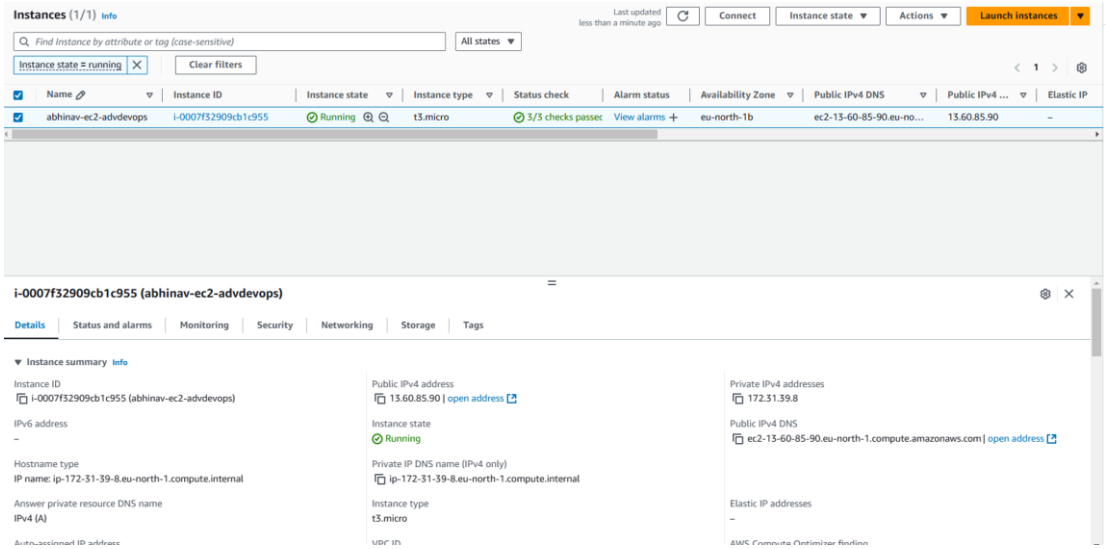


Create a new key-pair of RSA type. (Will be used to SSH into our instance later)

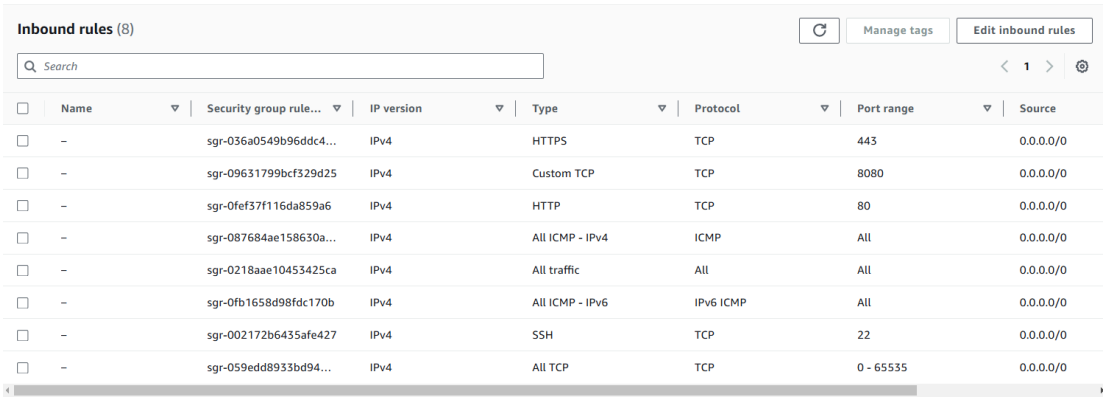


Allow HTTP/HTTPS traffic from anywhere





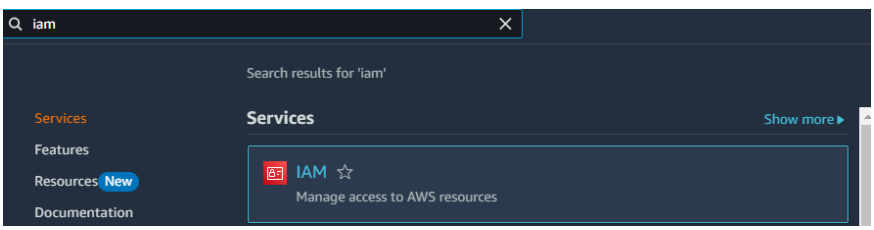
Edit inbound rules in security groups



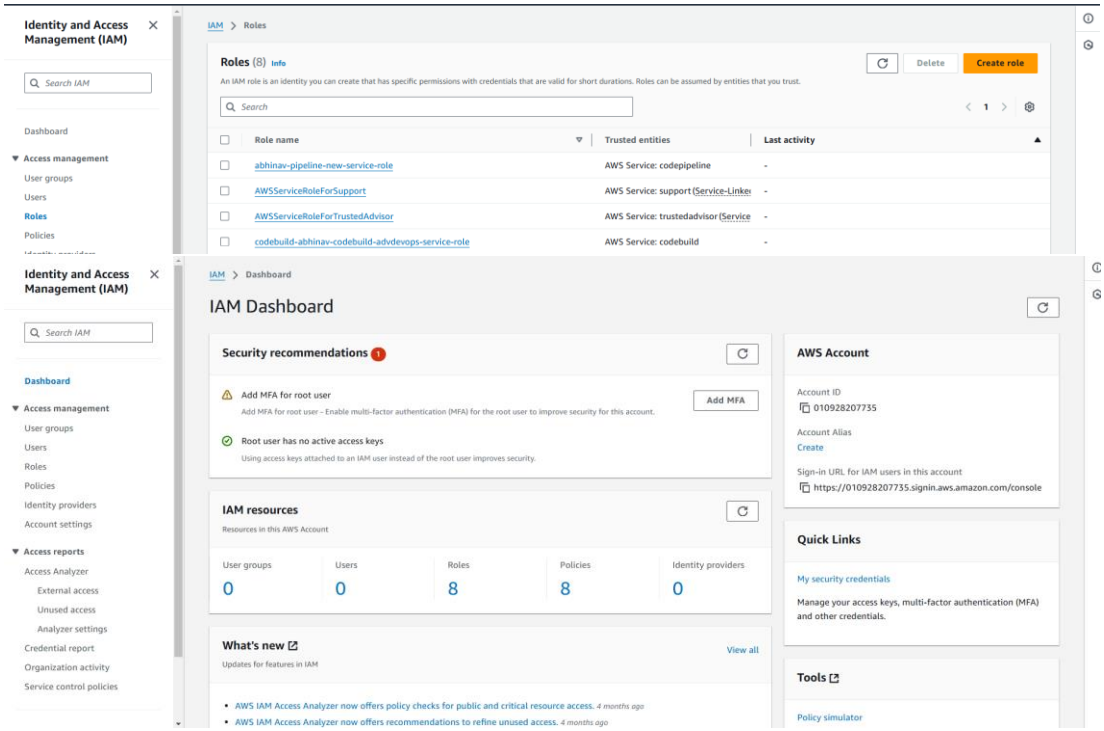
Take note of tag of instance

Step 7: Create an IAM Role for EC2 with CodeDeploy Permissions

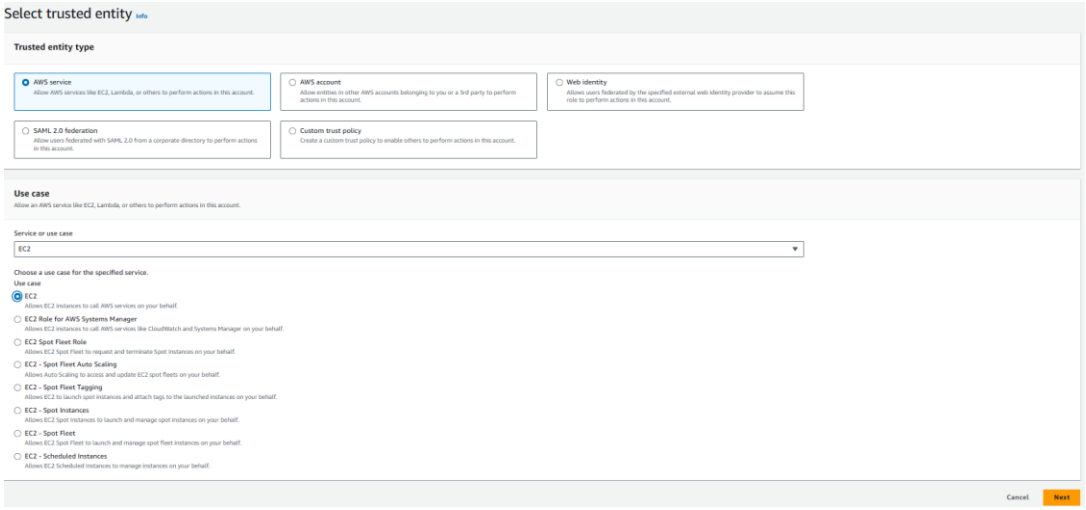
In the AWS Management Console, go to **Services** and select **IAM**.



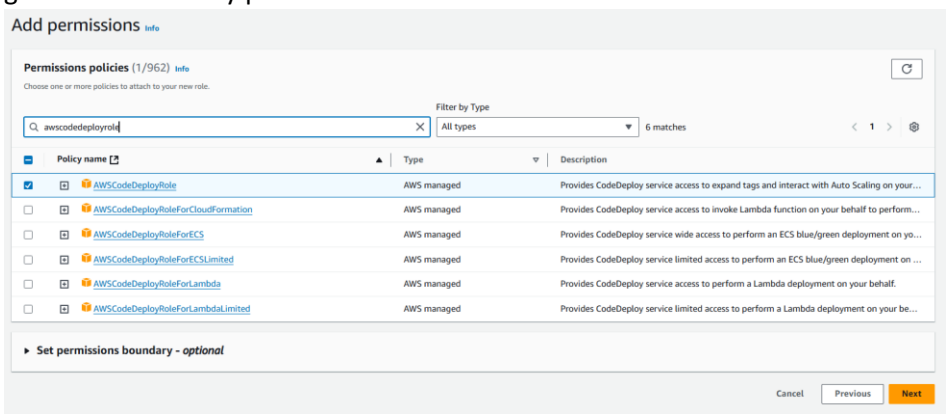
Click on **Roles** in the left sidebar and then click **Create role**.

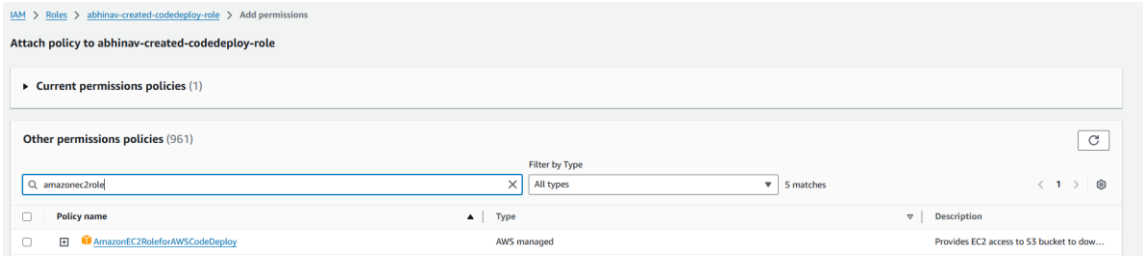


Choose **AWS service** and then select **EC2**.

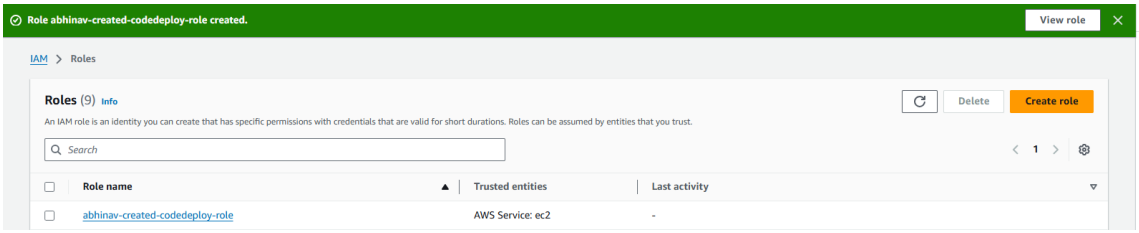


Search for and attach the AWSCodeDeployRole, AmazonEC2RoleforCodeDeploy policy to grant the necessary permissions.





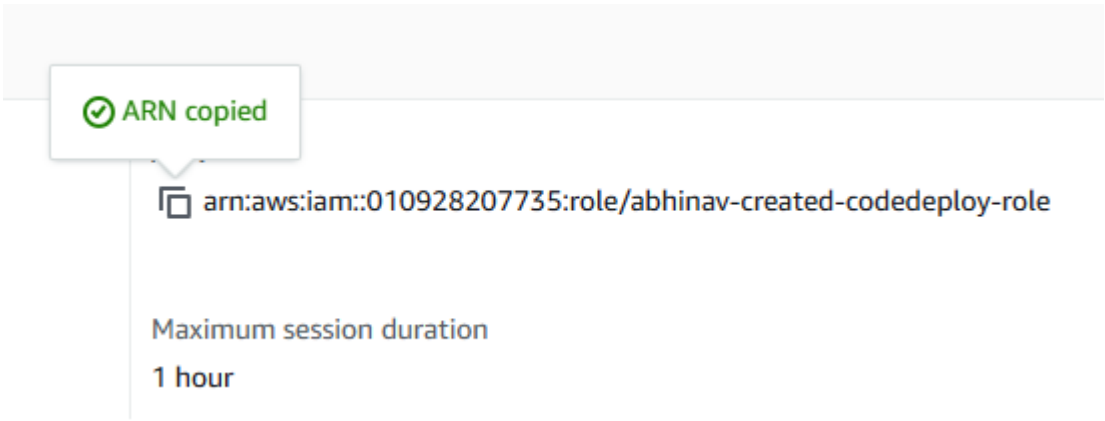
Name your role and click **Create role**.



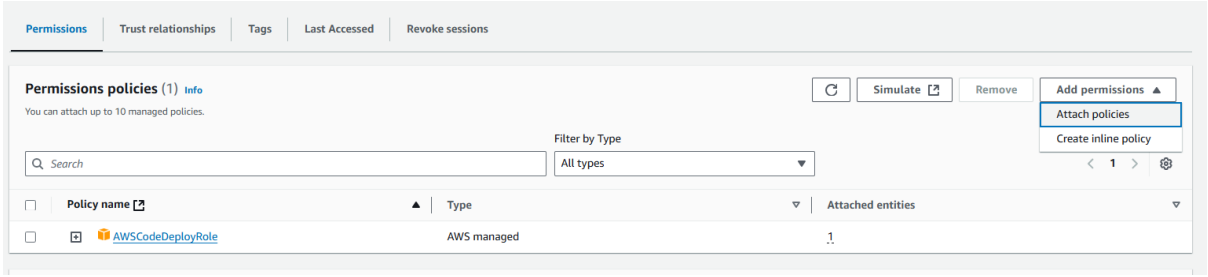
Goto trusted relationships and edit trust policy with code

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Principal": {
    "Service": "codedeploy.amazonaws.com"
   },
   "Action": "sts:AssumeRole"
  }
 ]
}
```

Copy the arn as it will be needed later

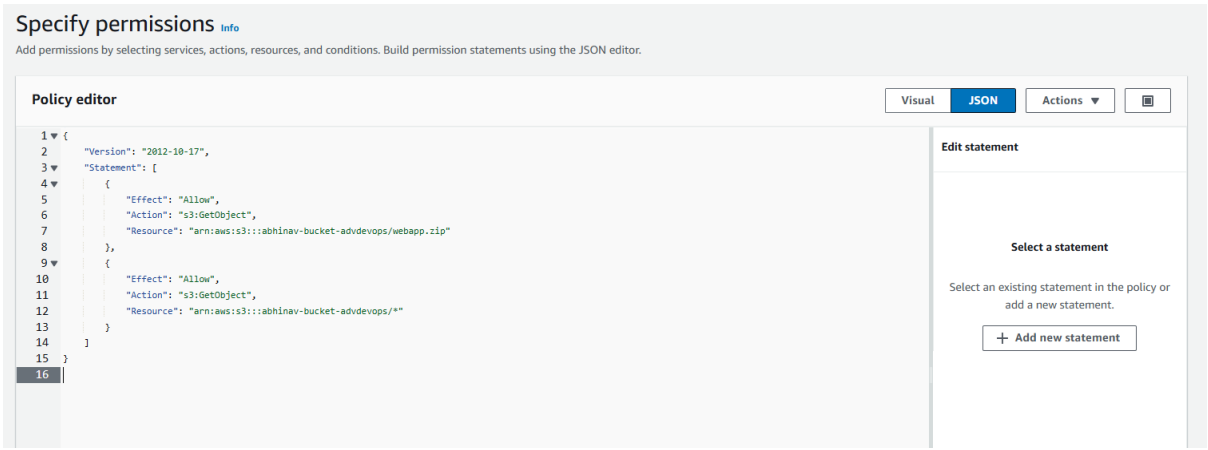


Add inline policy

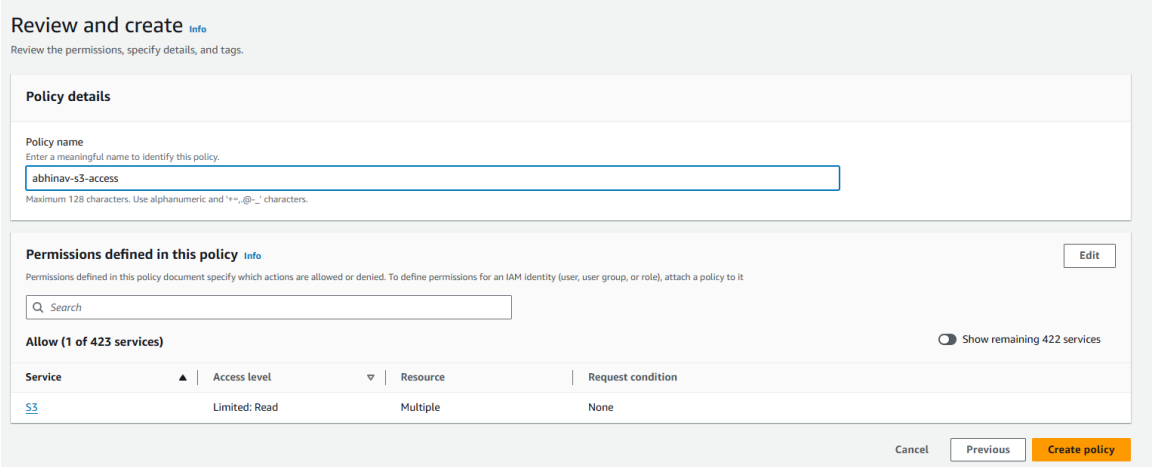


In Json editor

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": "s3:GetObject",
         "Resource": "arn:aws:s3:::abhinav-bucket-advdevops/webapp.zip"
      },
      {
         "Effect": "Allow",
         "Action": "s3:GetObject",
         "Resource": "arn:aws:s3:::abhinav-bucket-advdevops/*"
      }
   ]
}
```



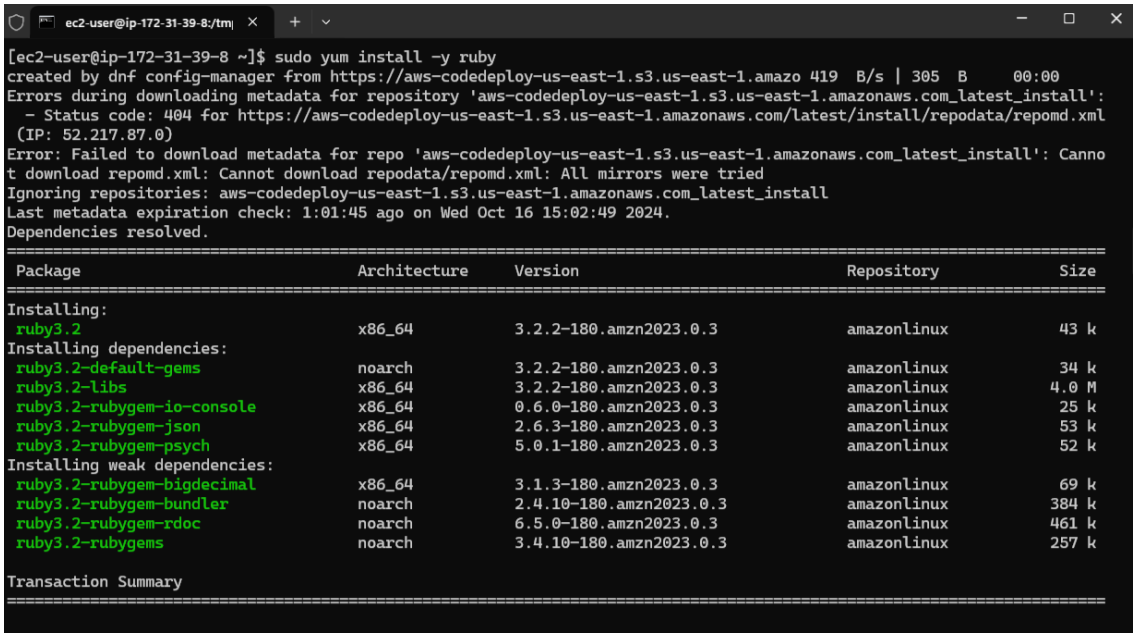
Review and create



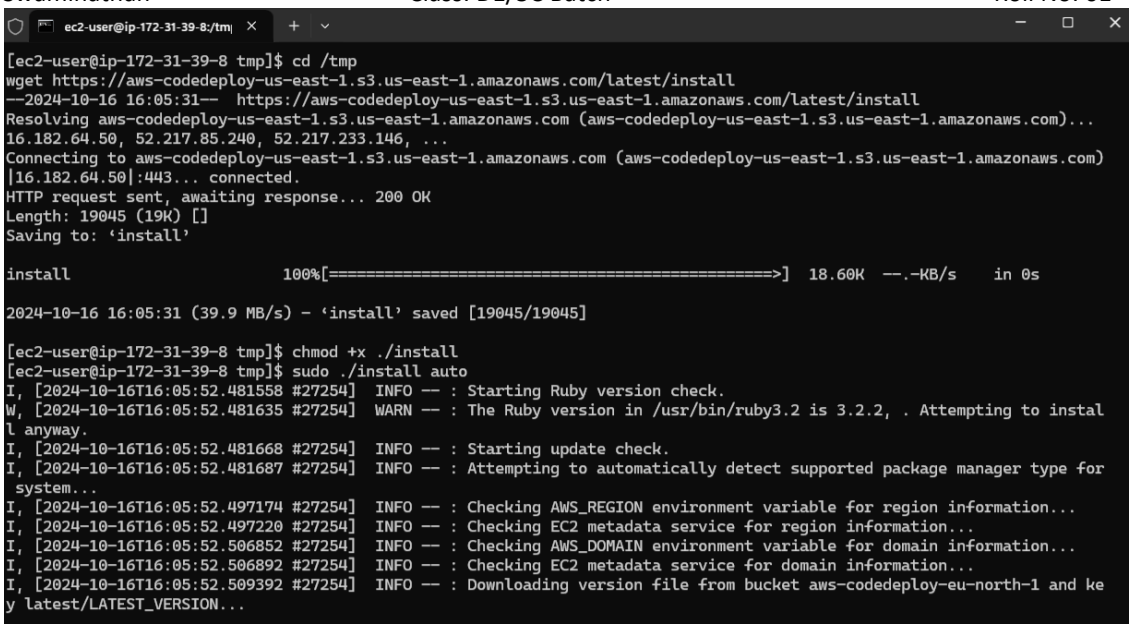
Step 8: Install codedeploy-agent and nginx on ec2

Refer to the documentation to install
https://docs.aws.amazon.com/codedeploy/latest/userguide/codedeploy-agent-operations-install-linux.html

sudo yum install -y ruby

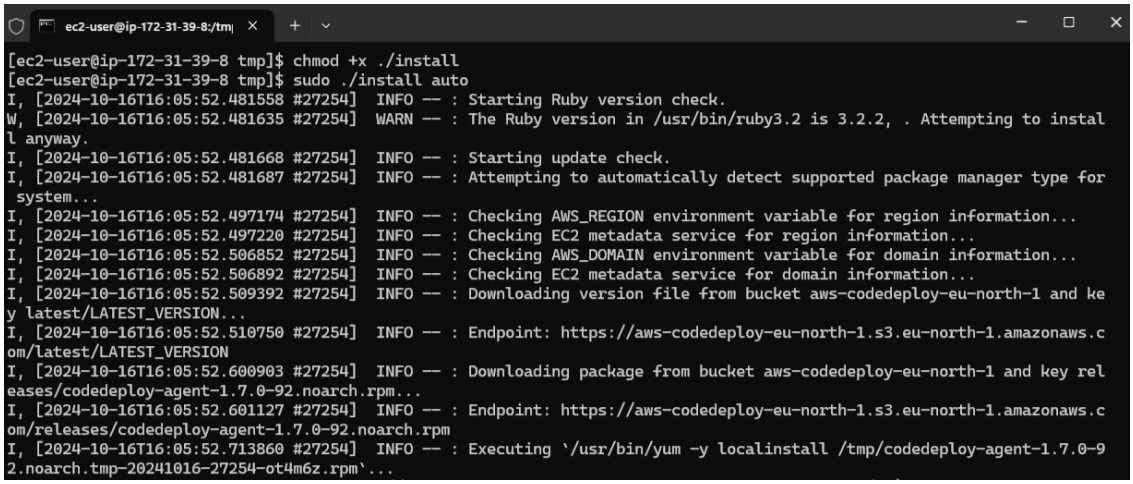


cd /tmp
(wget https://**bucket-name**.s3.**region-identifier**.amazonaws.com/latest/install)
wget https://aws-codedeploy-us-east-1.s3.us-east-1.amazonaws.com/latest/install



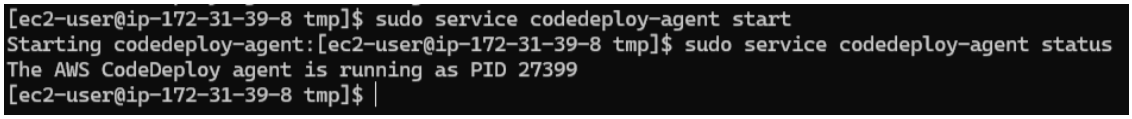
chmod +x ./install

sudo ./install auto

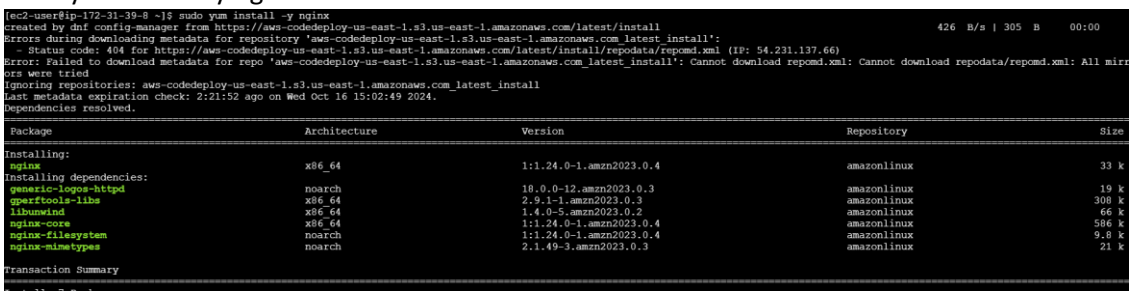


sudo service codedeploy-agent start

sudo service codedeploy-agent status



sudo yum install -y nginx



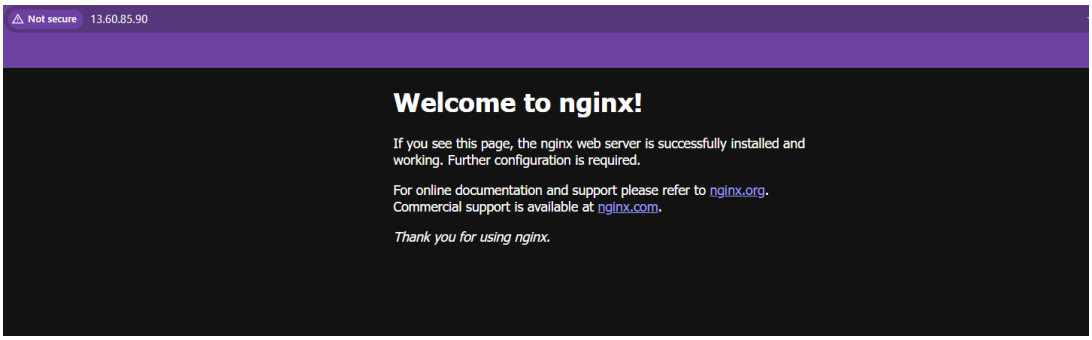
sudo service nginx start

sudo service nginx status

```
[ec2-user@ip-172-31-39-8 ~]$ sudo service nginx status
Redirecting to /bin/systemctl status nginx.service
● nginx.service - The nginx HTTP and reverse proxy server
     Loaded: loaded (/usr/lib/systemd/system/nginx.service; disabled; preset: disabled)
     Active: active (running) since Wed 2024-10-16 17:25:47 UTC; 16s ago
    Process: 33739 ExecStartPre=/usr/bin/rm -f /run/nginx.pid (code=exited, status=0/SUCCESS)
    Process: 33740 ExecStartPre=/usr/sbin/nginx -t (code=exited, status=0/SUCCESS)
    Process: 33741 ExecStart=/usr/sbin/nginx (code=exited, status=0/SUCCESS)
   Main PID: 33742 (nginx)
      Tasks: 3 (limit: 1059)
     Memory: 2.9M
        CPU: 54ms
     CGroup: /system.slice/nginx.service
             ├─33742 "nginx: master process /usr/sbin/nginx"
             ├─33743 "nginx: worker process"
             └─33744 "nginx: worker process"

Oct 16 17:25:47 ip-172-31-39-8.eu-north-1.compute.internal systemd[1]: Starting nginx.service - The nginx HTTP and reverse proxy server...
Oct 16 17:25:47 ip-172-31-39-8.eu-north-1.compute.internal nginx[33740]: nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
Oct 16 17:25:47 ip-172-31-39-8.eu-north-1.compute.internal nginx[33740]: nginx: configuration file /etc/nginx/nginx.conf test is successful
Oct 16 17:25:47 ip-172-31-39-8.eu-north-1.compute.internal systemd[1]: Started nginx.service - The nginx HTTP and reverse proxy server.
[ec2-user@ip-172-31-39-8 ~]$
```

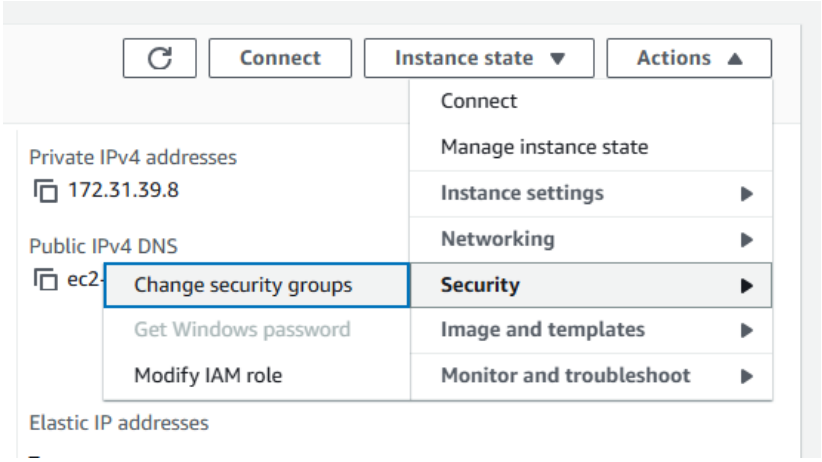
Check if nginx installed by opening the instance's public ipv4 ip



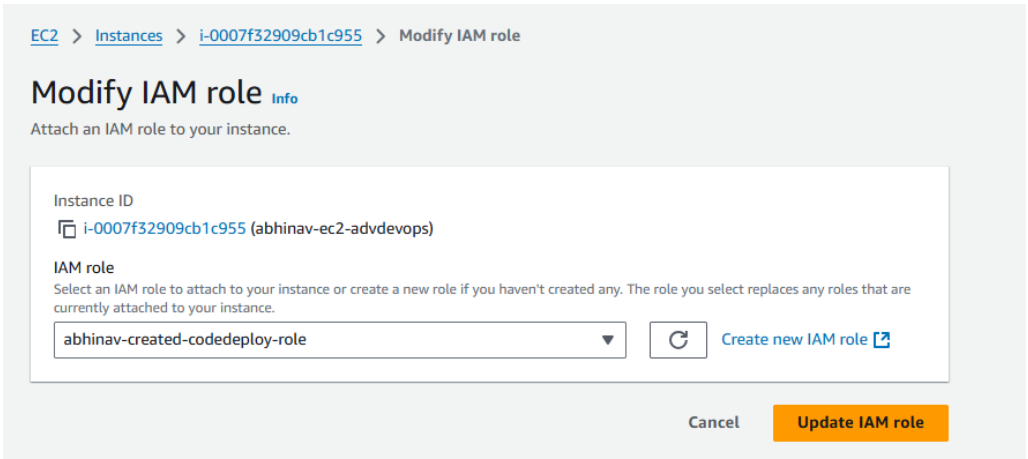
Step 9: Attach the IAM Role to the EC2 Instance

Go back to the EC2 Dashboard.

Select your Instance and click on Actions > Security > Modify IAM Role.

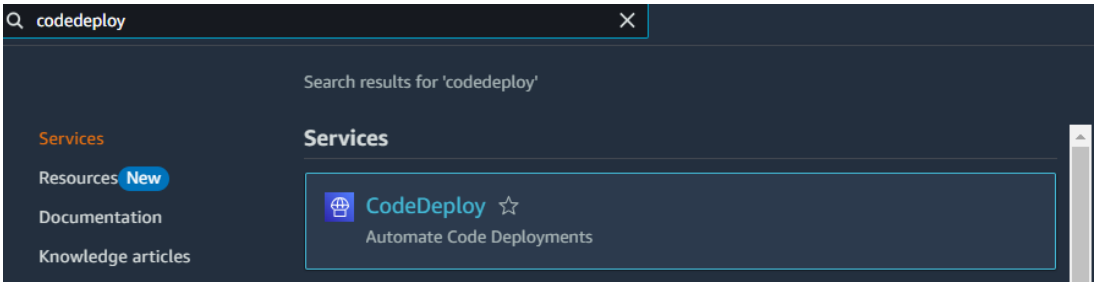


Attach the IAM Role created earlier.

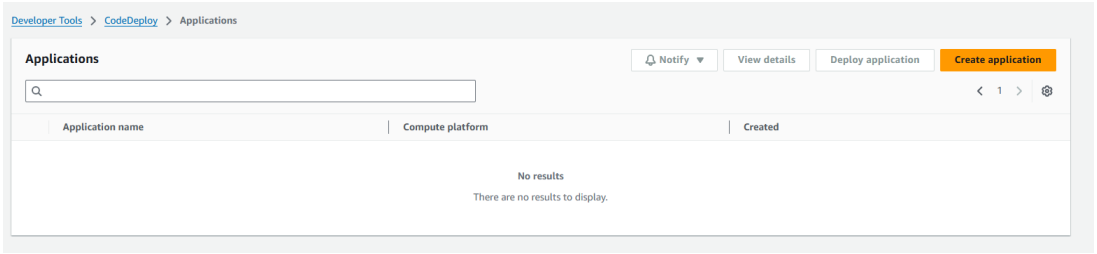


Step 10: Set Up AWS CodeDeploy

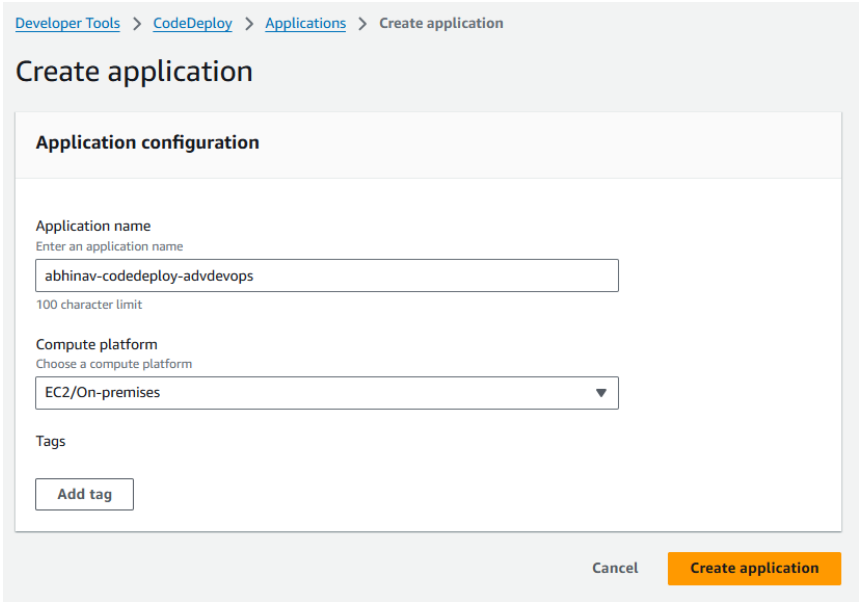
Open the CodeDeploy Console: In the AWS Management Console, navigate to the **CodeDeploy** service.



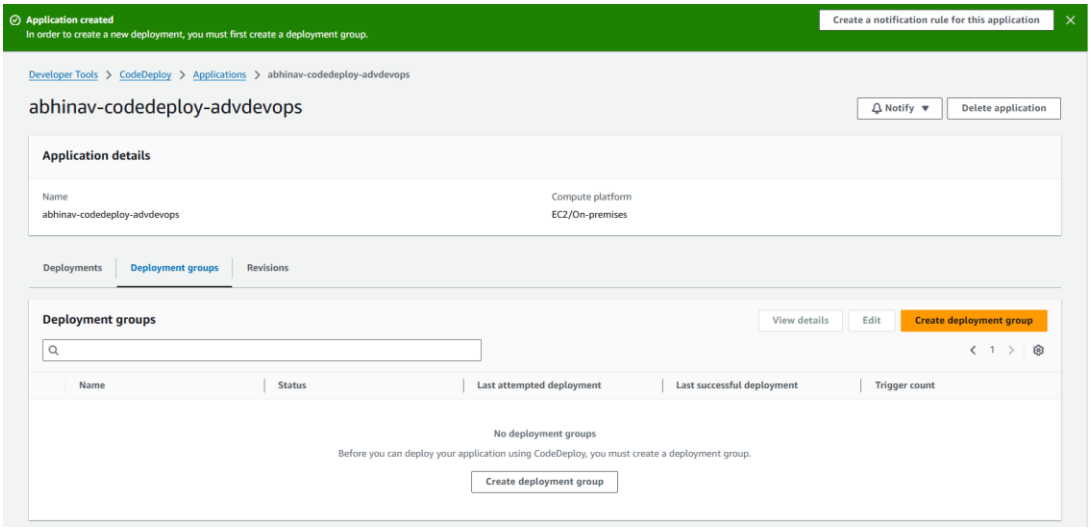
Click on **"Create application"**.

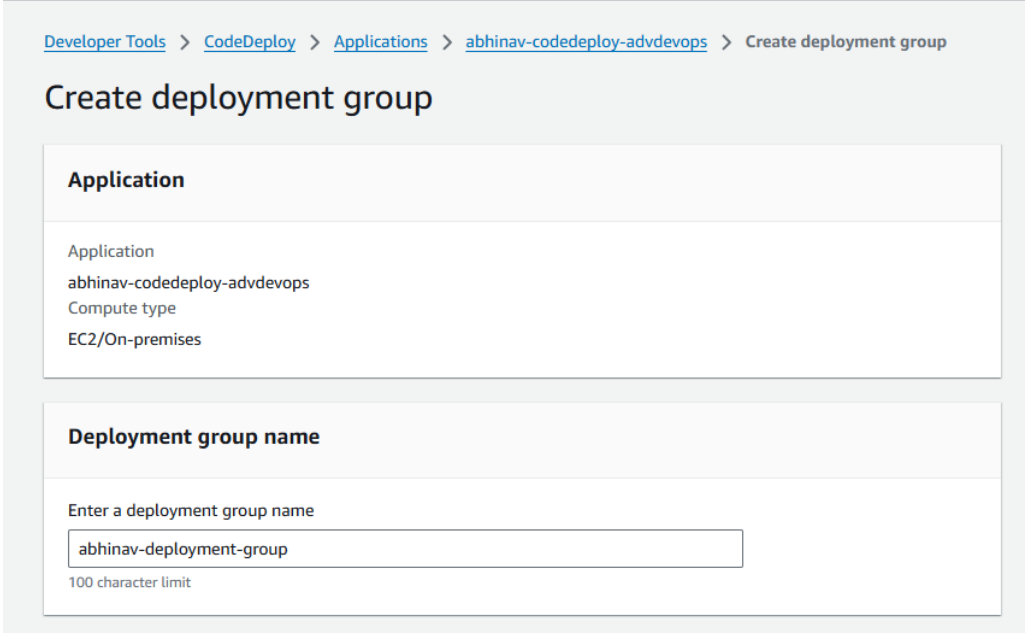


Enter an application name , For Compute platform, select EC2/On-premises. Click "Create application"

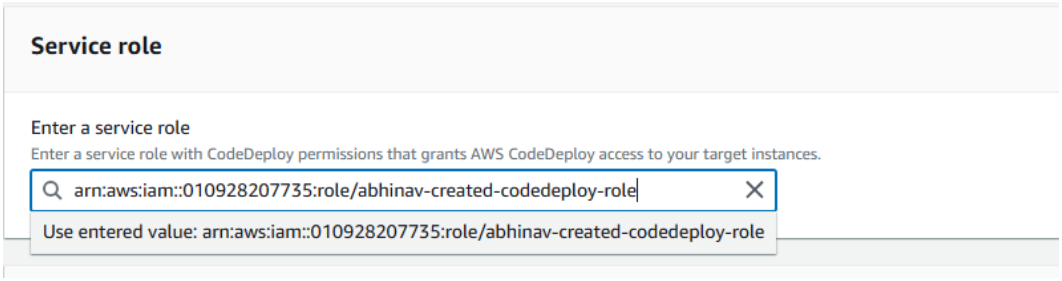


After creating the application, you will be prompted to create a deployment group.
Click on "Create deployment group".

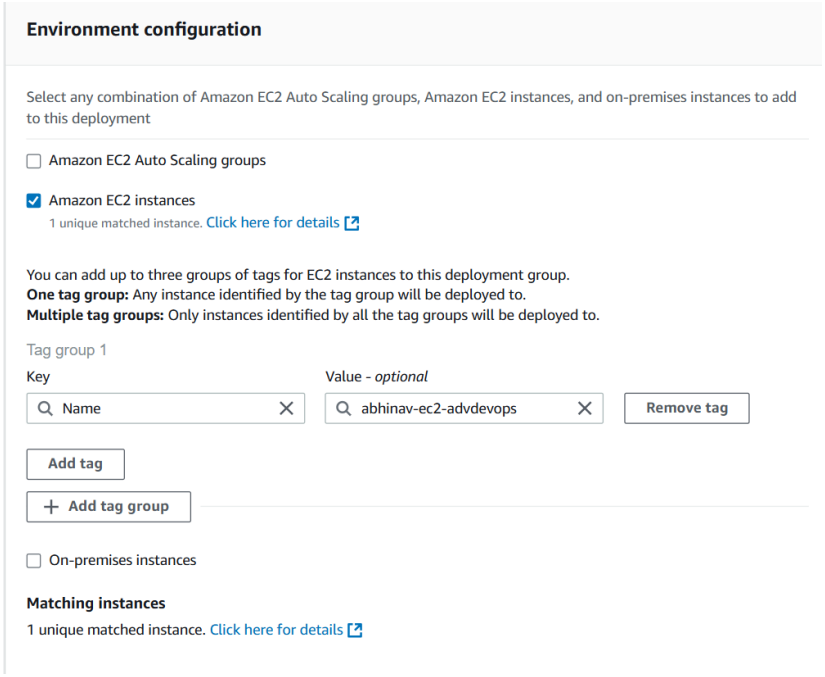




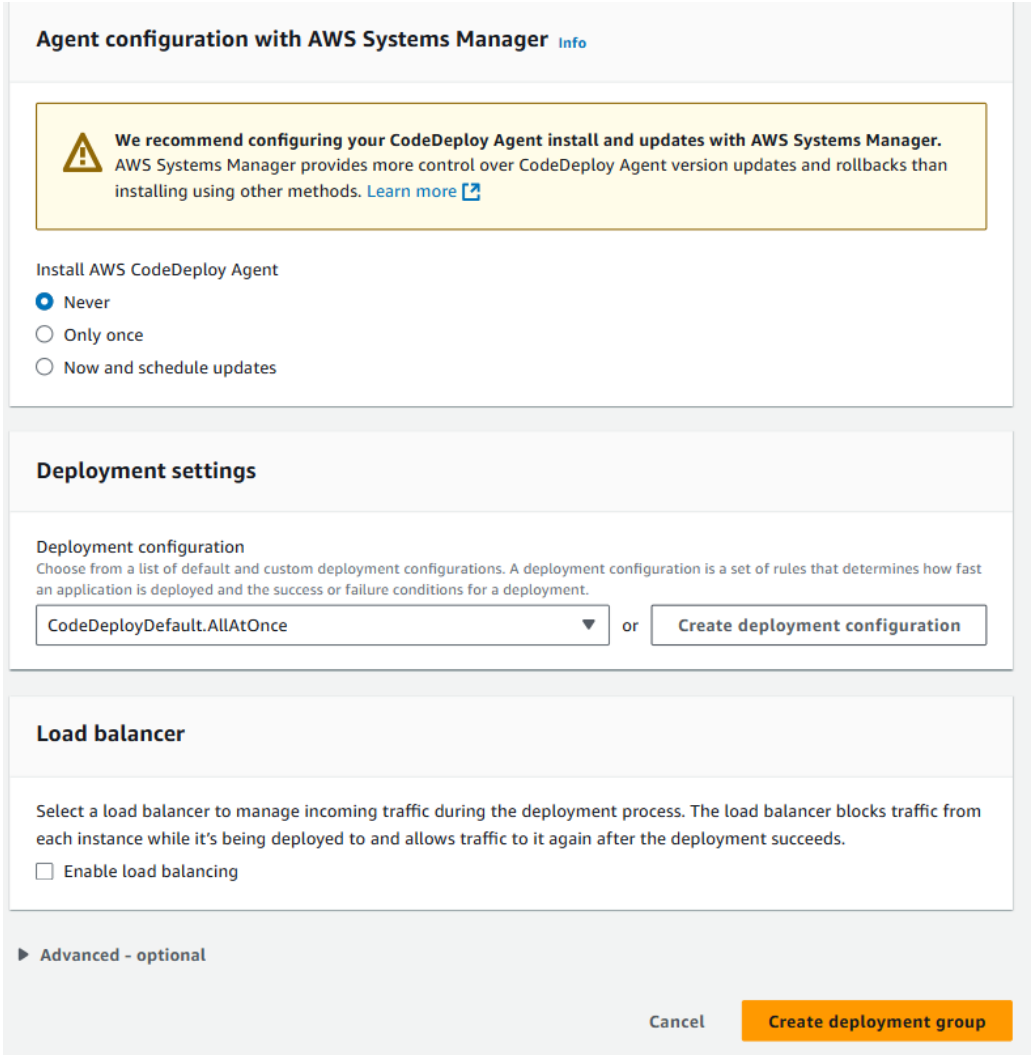
Enter the **ARN** of the service role you created for CodeDeploy (the one with permissions for CodeDeploy).
arn:aws:iam::010928207735:role/abhinav-created-codedeploy-role



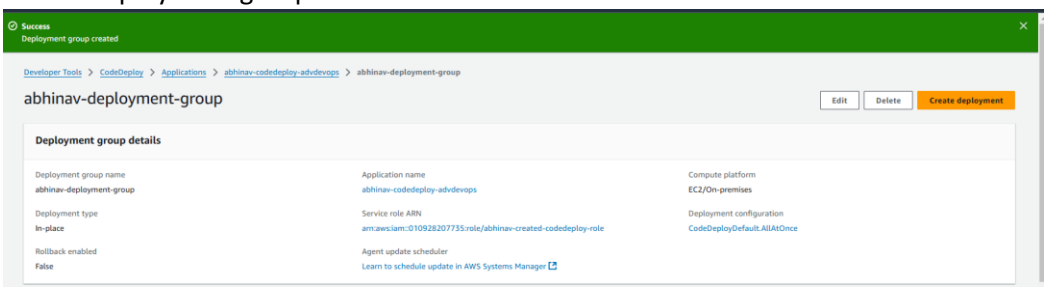
For **Environment configuration**, choose **EC2 instances**. Enter your key value pair which was noted from under tags



Disable load balancing.

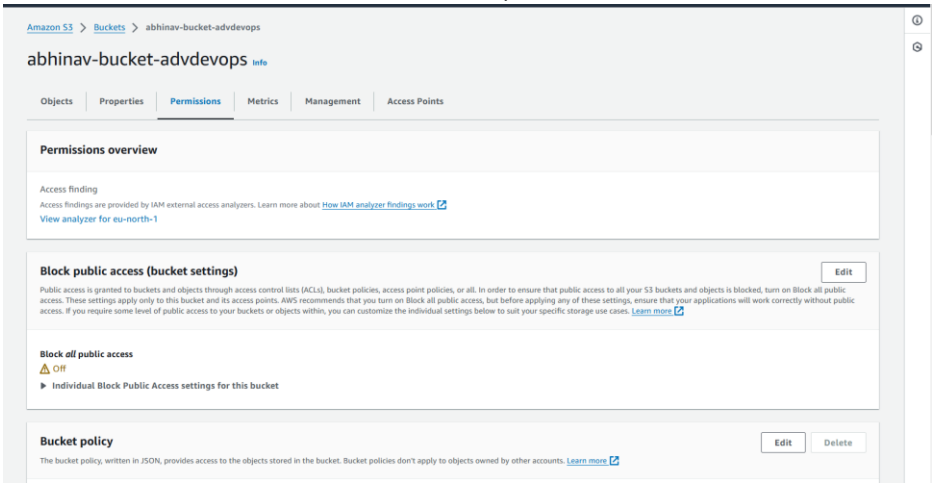


Create deployment group



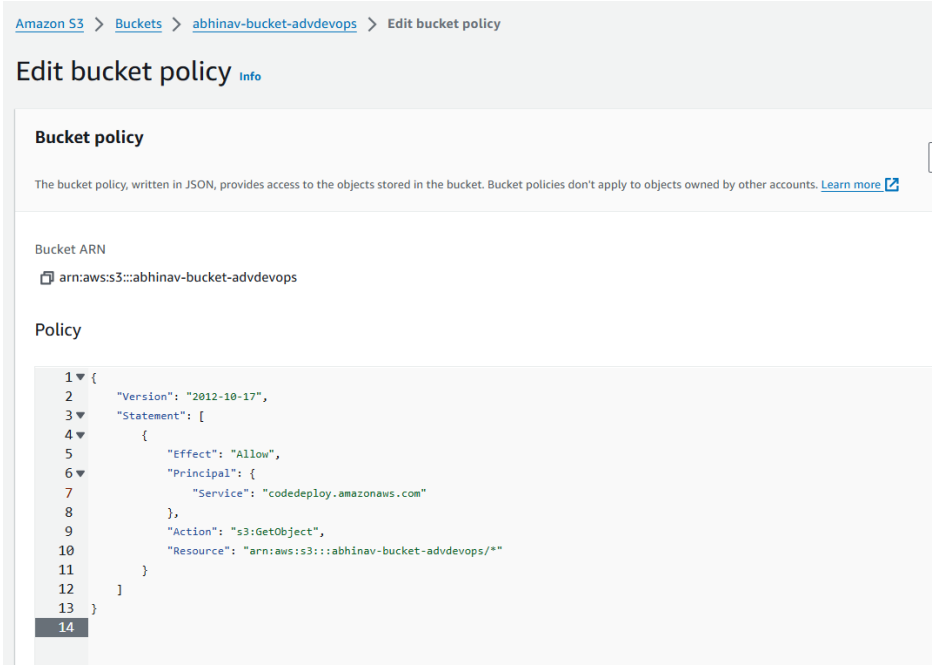
Step 11: Give access of S3 bucket to codedeploy

Navigate to the **S3 service** in the AWS Management Console.
Find and select your bucket (e.g., abhinav-bucket-advdevops).
Go to the **Permissions** tab and check the **Bucket Policy**.

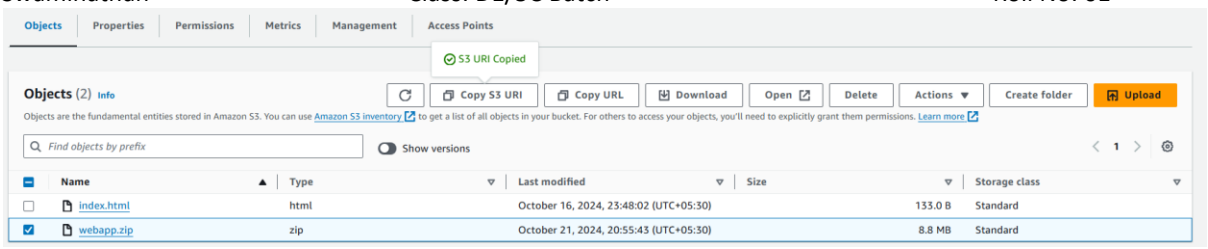


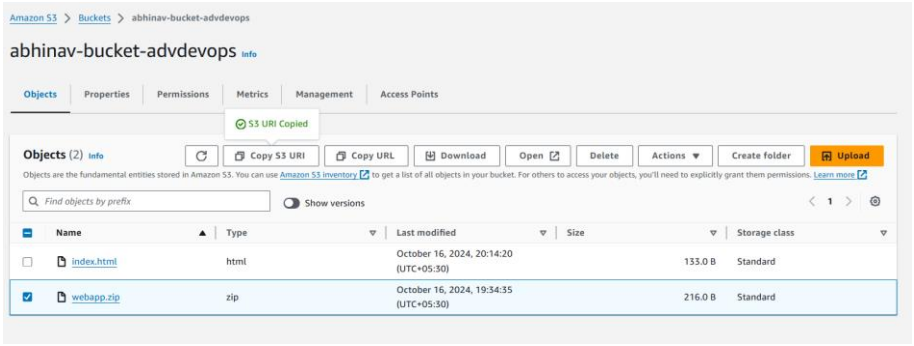
Go to the Permissions tab and check the Bucket Policy.
Ensure that the policy allows CodeDeploy to access the bucket

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Principal": {
            "Service": "codedeploy.amazonaws.com"
         },
         "Action": "s3:GetObject",
         "Resource": "arn:aws:s3:::{your_bucket_name}/*"
      }
   ]
}
```



Copy the S3 bucket's URI

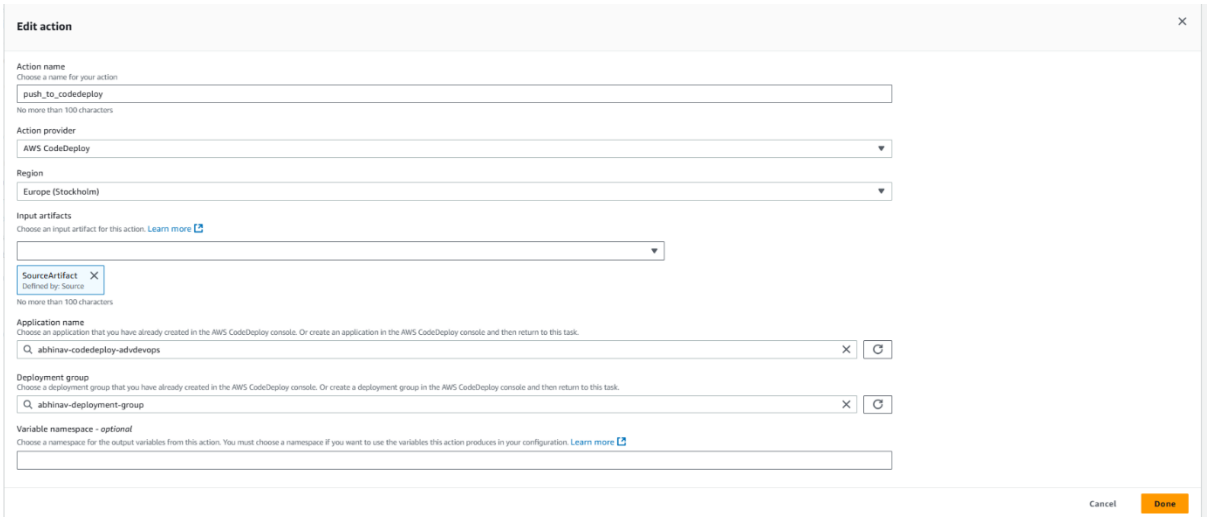




Step 12: Automate CodeDeploy using CodePipeline

.

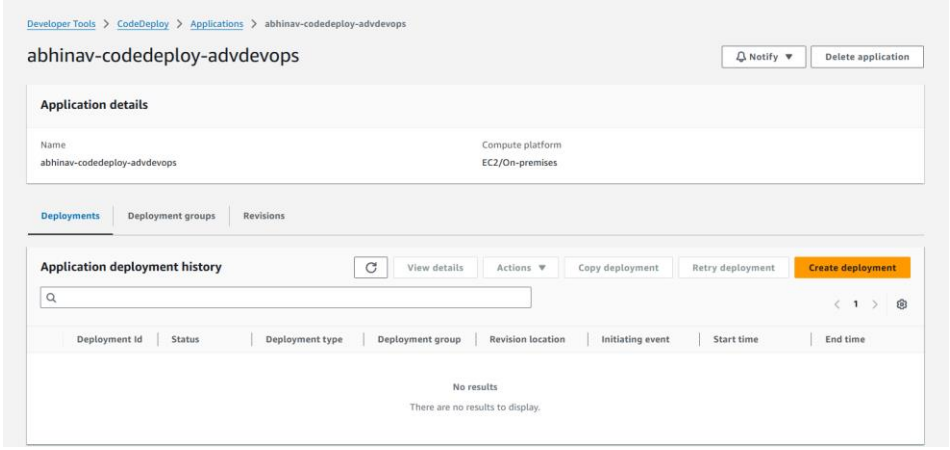
Under your CodePipeline > Deploy > Edit > Add an Action > Set Action Provider as "**AWS Codedeploy**" > configure our created application and deployment groups > **Save**



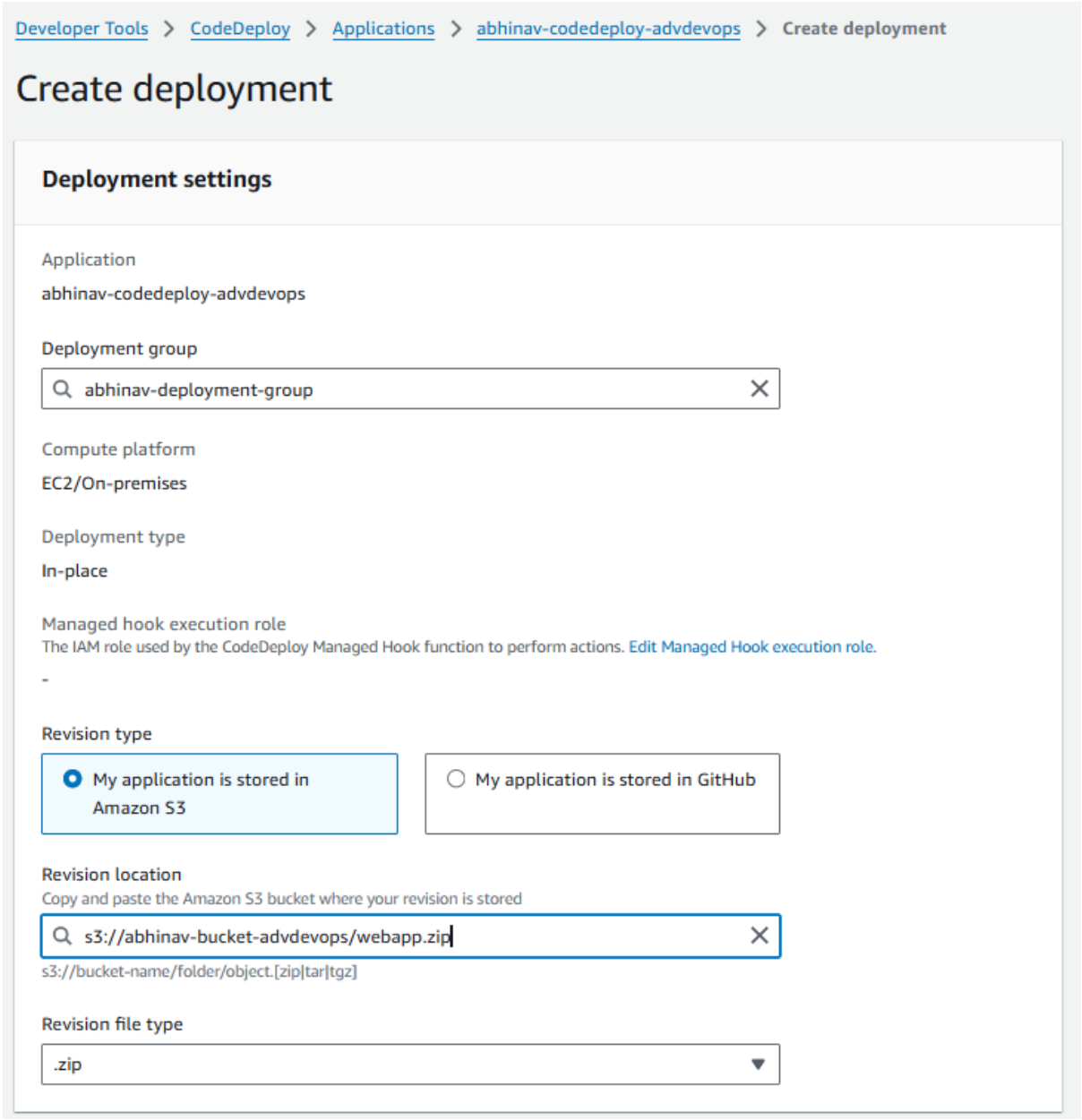
Step 13: Create a Deployment for Your Application

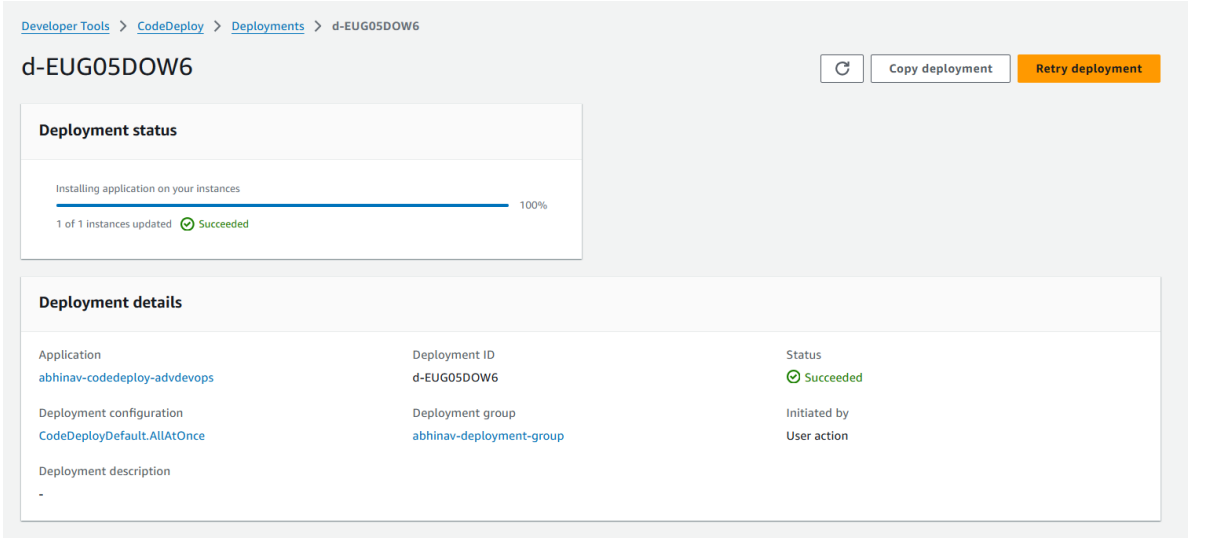
Click on the Deployments tab.

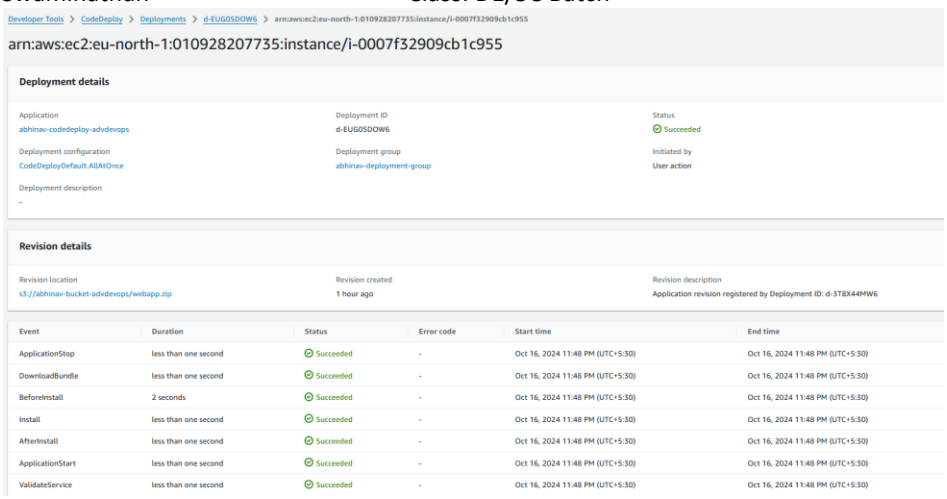
Click on Create deployment.



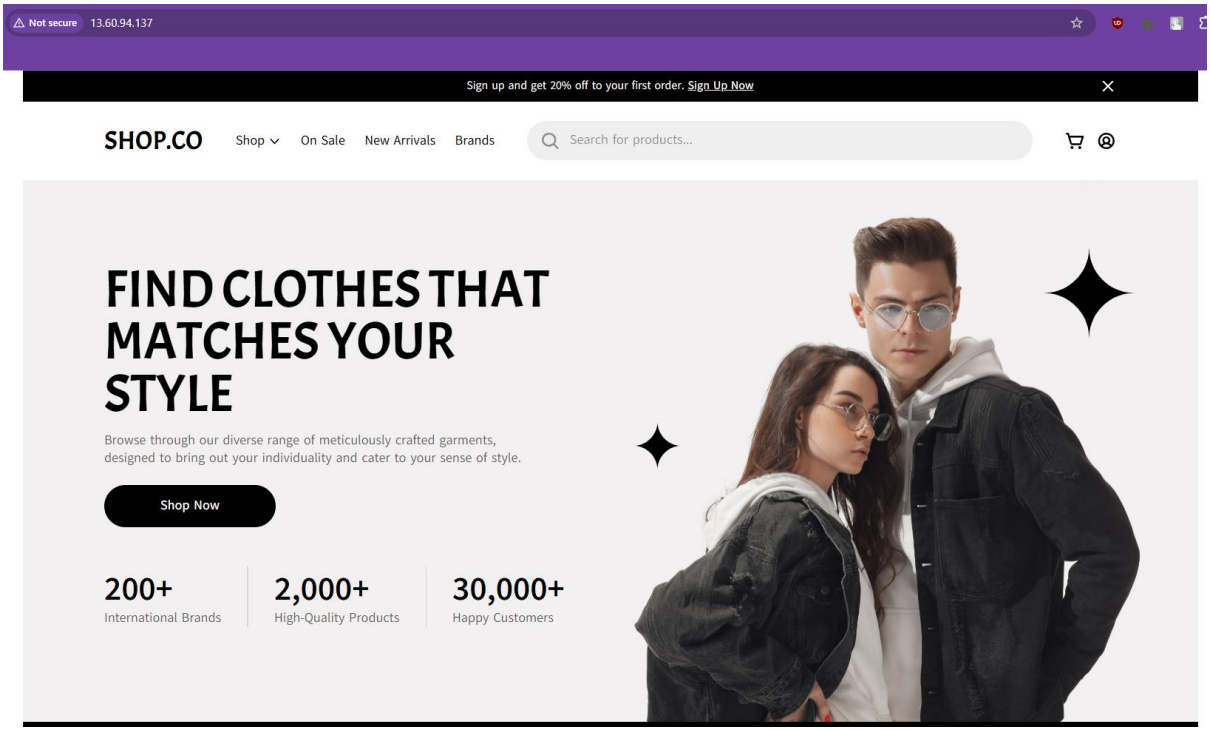
Selet your deployment group and paste the URI you copied







Check if your page is hosted on your public ipv4 address of your EC2 bucket



Conclusion

In this project, we successfully automated the deployment of a web application. The application was hosted on an EC2 instance and managed through S3 for storage. Codebuild created the Build Artifact of the project. Code Pipeline stored the source & build artifacts on an S3 bucket. It then handled automation of the project linking with AWS CodeDeploy which deployed the project to our EC2 instance. The instance had nginx setup as a web server which enabled rendering of the web application over the public IPV4 address.