DEPLOYING WEB APPLICATION & MODEL ON AWS CLOUD USING VARIOUS SERVICES

By:
Arisha Akhtar - A041
Tina Narsinghani - A040

Introduction : Traditional on-premises web architectures require complex solutions and accurate reserved capacity forecast in order to ensure reliability. Dense peak traffic periods and wild swings in traffic patterns result in low utilization rates of expensive hardware. This yields high operating costs to maintain idle hardware, and an inefficient use of capital for underused hardware. Amazon Web Services (AWS) provides a reliable, scalable, secure, and highly performing infrastructure for the most demanding web applications. This infrastructure matches IT costs with customer traffic patterns in near-real time.AWS provides seamless and cost effective solutions.

Services used for this Case study:

1. Web application was deployed on AWS Amplify , Amazon S3 .
2. ML Model XGboost was deployed using Sagemaker, stored data in S3 , ran the code using Lambda Function for prediction using the model.

Case - 1 : Deploying webpage on cloud using AWS Amplify

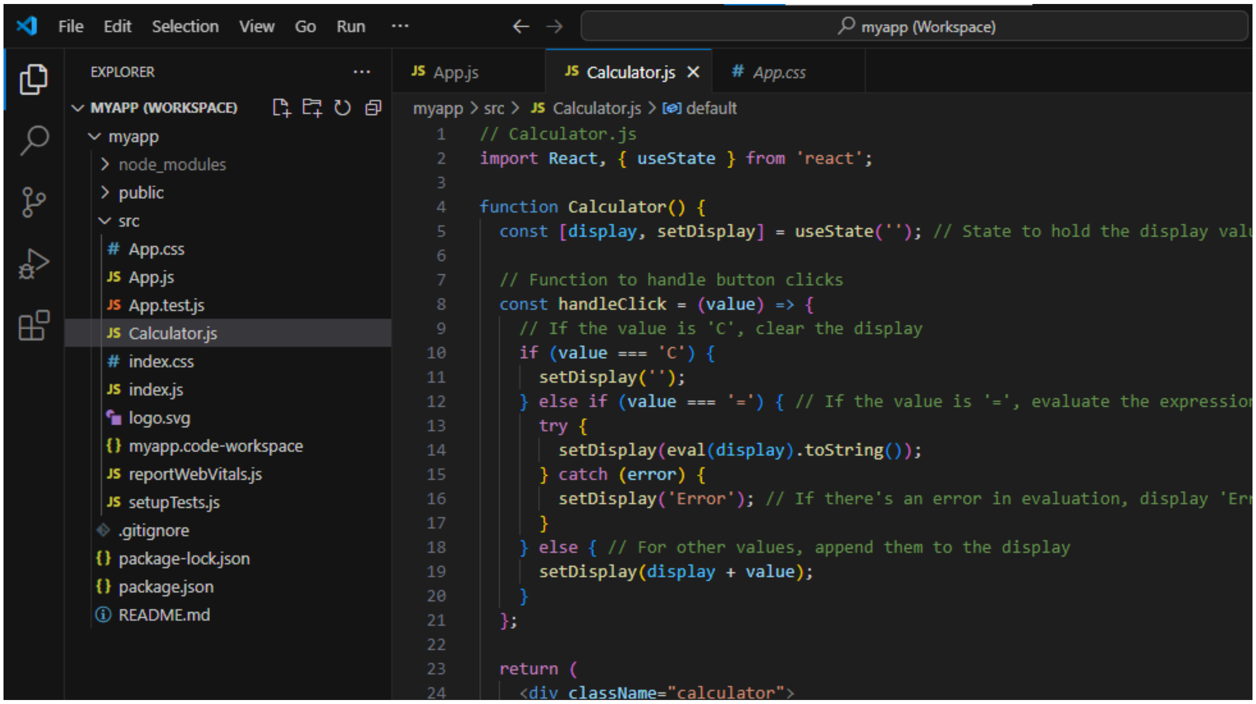
Amplify: Amplify is a framework with a set of tools and services that help you build scalable full stack applications, powered by AWS services. AWS Amplify is a set of tools and services that can be used together or on their own, to help front-end web and mobile developers build scalable full stack applications, powered by AWS. With Amplify, you can configure app backends and connect your app in minutes, deploy static web apps in a few clicks, and easily manage app content outside the AWS console.

For an example we deployed a React js web application on AWS Amplify

Methodology / Process of deployment:

Created a web application using react js on visual studio code . For deploying the webapp on Amplify we first connected to Github repository & we used Git app . Visual studio and git commands were used to push files to the github repository.

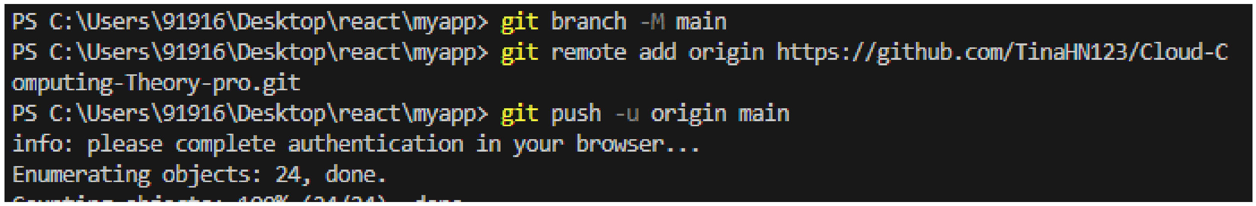
Step 1: We had created react js files on vscode:



Step 2: Downloaded git and wrote these commands in Visual studio code to push the files on github repository.

```
PS C:\Users\91916\Desktop\react\myapp> git init
Initialized empty Git repository in C:/Users/91916/Desktop/react/myapp/.git/
PS C:\Users\91916\Desktop\react\myapp> git add .
warning: in the working copy of '.gitignore', LF will be replaced by CRLF the next time Git touch
es it
warning: in the working copy of 'README.md', LF will be replaced by CRLF the next time Git touche
s it
```

```
PS C:\Users\91916\Desktop\react\myapp> git commit -m "first commit"
[master (root-commit) 894d783] first commit
 20 files changed, 18802 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 README.md
```



Here are the Git commands that were used to connect to our Github repository:

git init

git add .

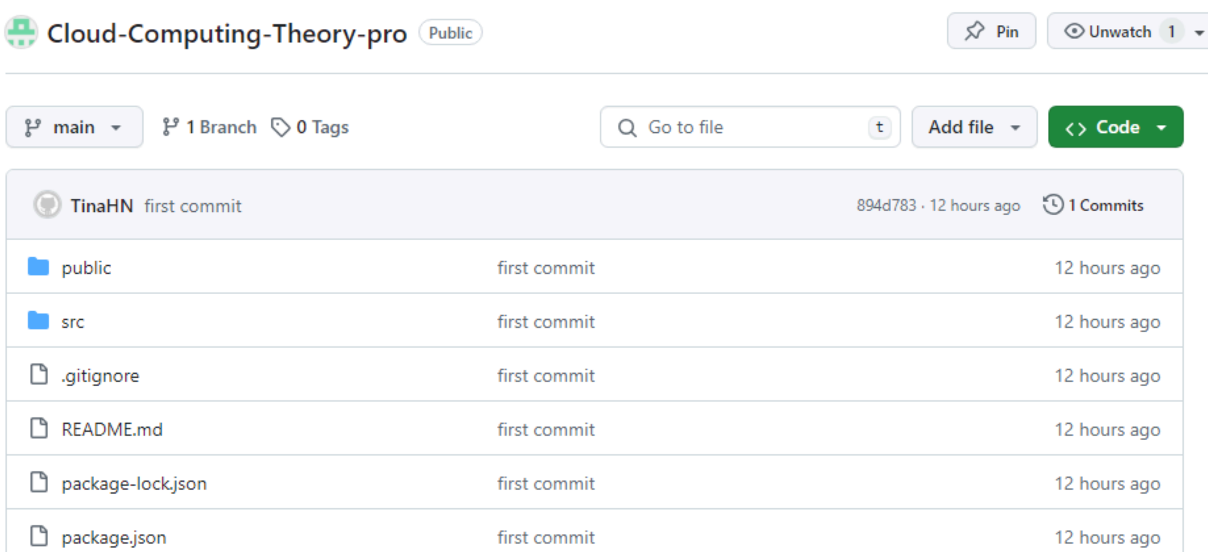
git commit -m "first commit"

git branch -M main

git remote add origin https://github.com/TinaHN123/Practice.git

git push -u origin main

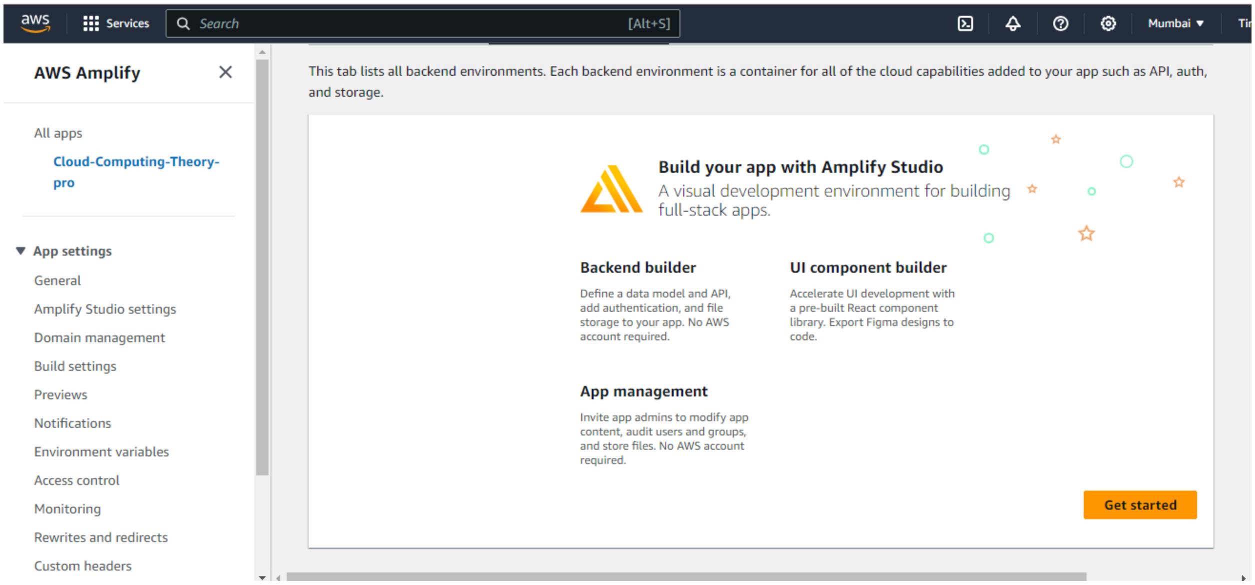
Step 3: FIles were successfully pushed into Github repository



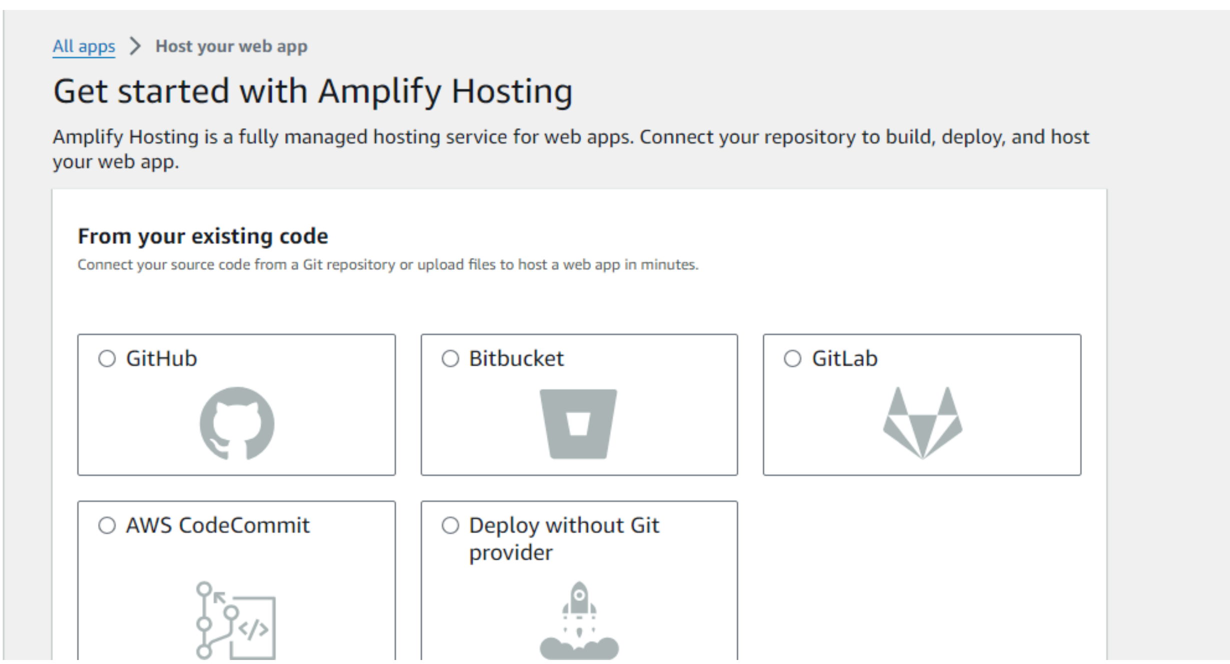
Link to the repository: https://github.com/TinaHN123/Cloud-Computing-Theory-pro

Step 4: Connecting source code and uploading web application files from our Github Repository

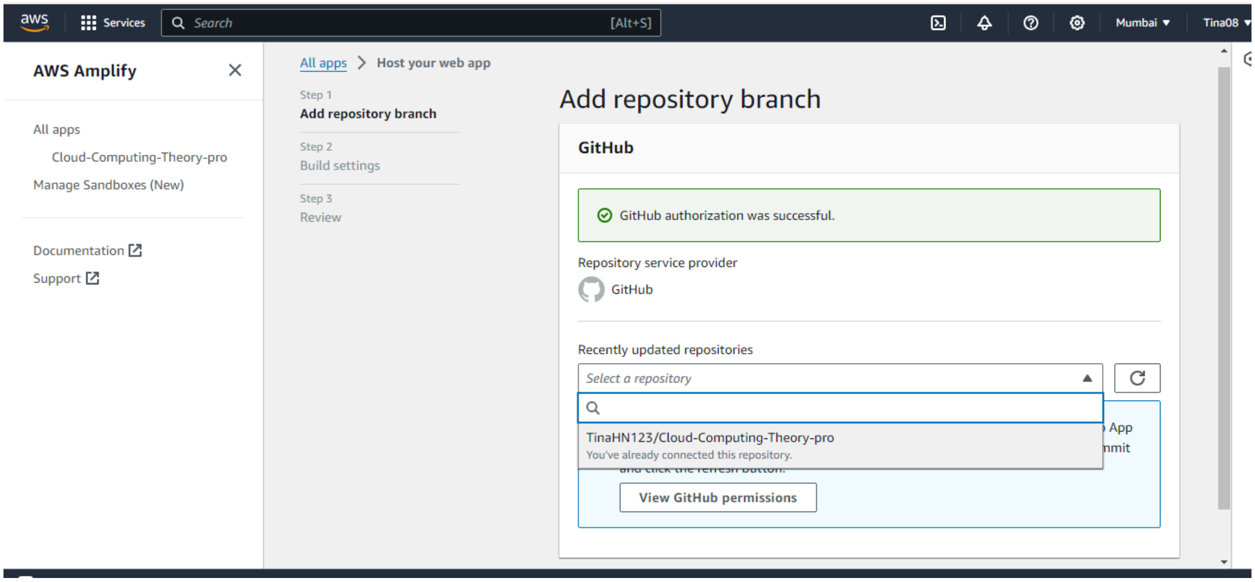
Amazon Amplify Homepage:



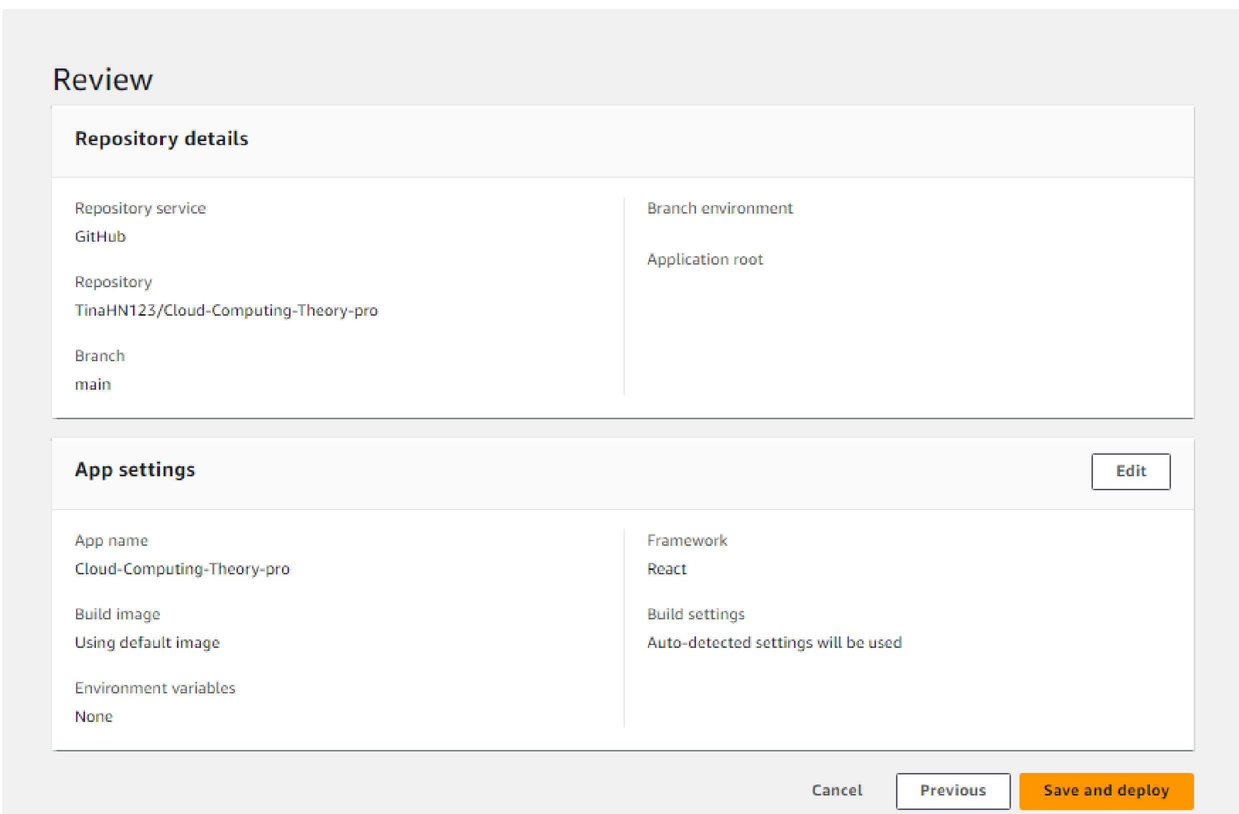
Connecting to Github repository:



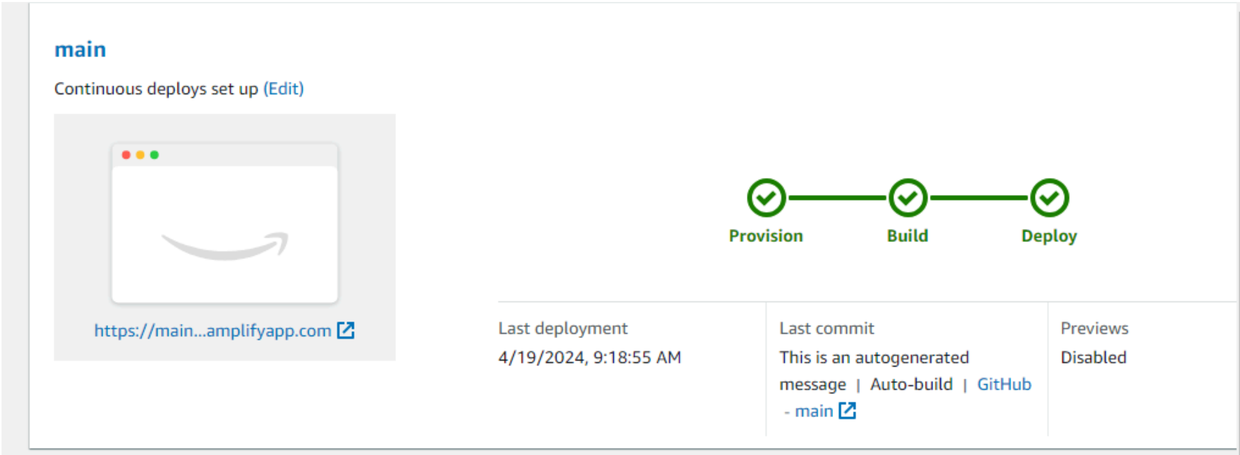
Step 5: Authorization of Github was done after this



Step 6: Review , Save and deploy

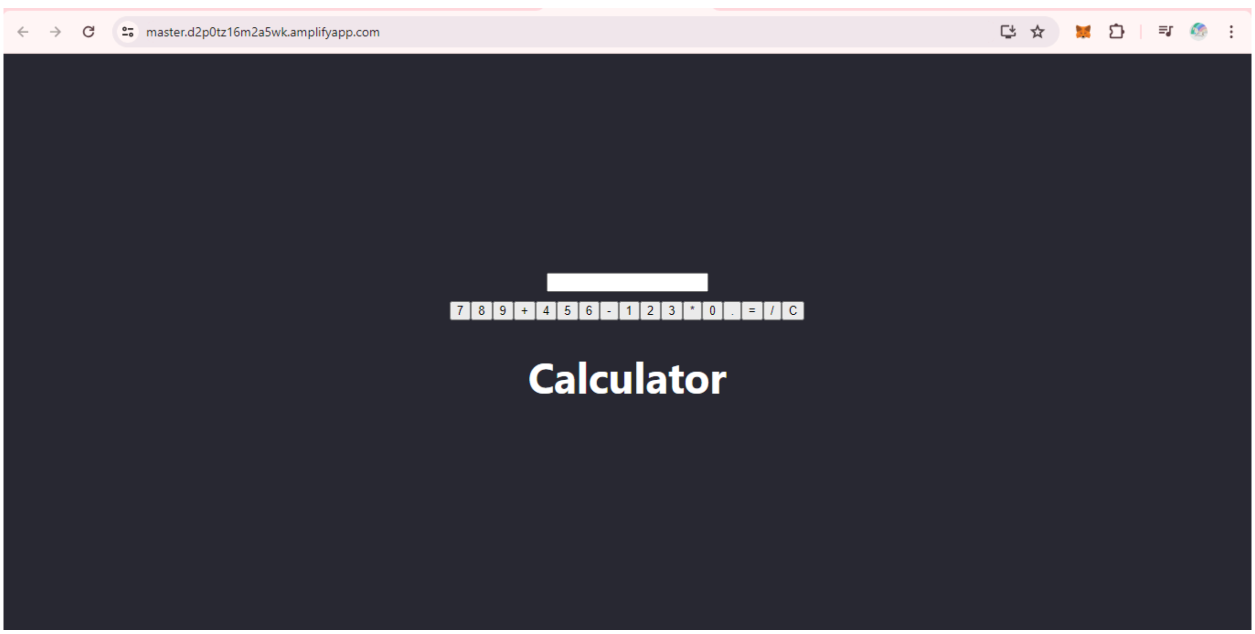


Step 7: The application was deployed on web



Link to webapp: https://master.d2p0tz16m2a5wk.amplifyapp.com/

Frontend Webapp:



This is the use case of Platform as a service provided by Amazon AWS Cloud.

Case 2 : Deploying webapp using Amazon S3

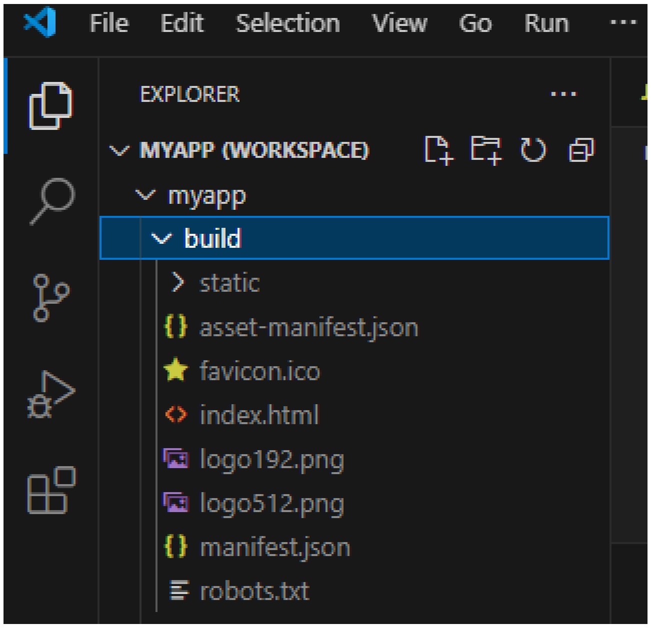
Amazon Simple Storage Service (Amazon S3): Amazon Simple Storage Service (Amazon S3) is an object storage service offering industry-leading scalability, data availability, security, and performance. Customers of all sizes and industries can store and protect any amount of data for virtually any use case, such as data lakes, cloud-native applications, and mobile apps. With cost-effective storage classes and easy-to-use management features, you can optimize costs, organize data, and configure fine-tuned access controls to meet specific business, organizational, and compliance requirements.

For an example we deployed a React js web application on Amazon S3

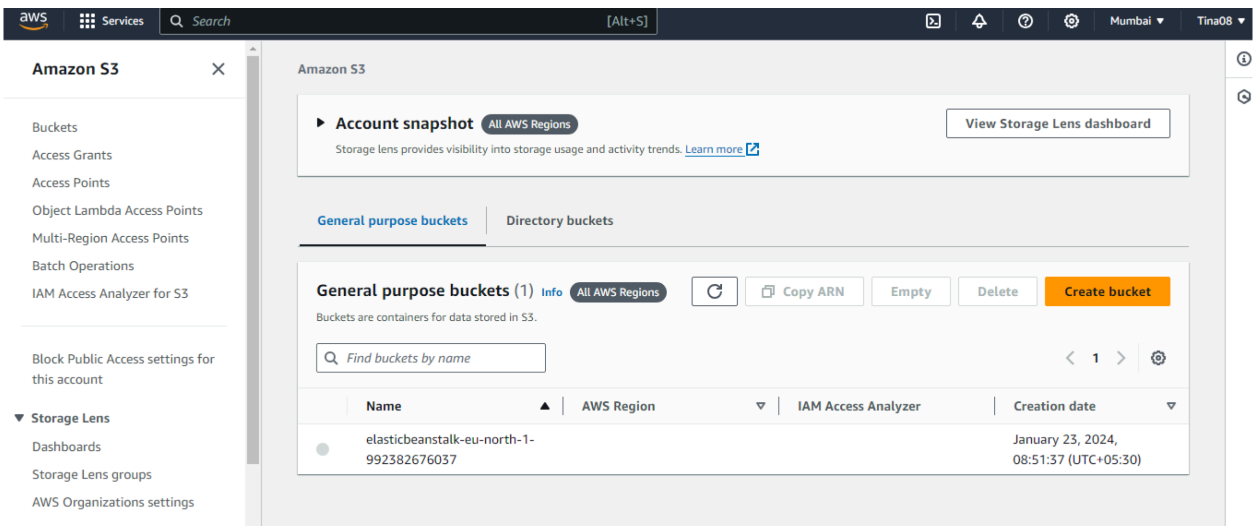
Methodology / Process of deployment:

Step 1: Created a build folder using npm run command on Vscode

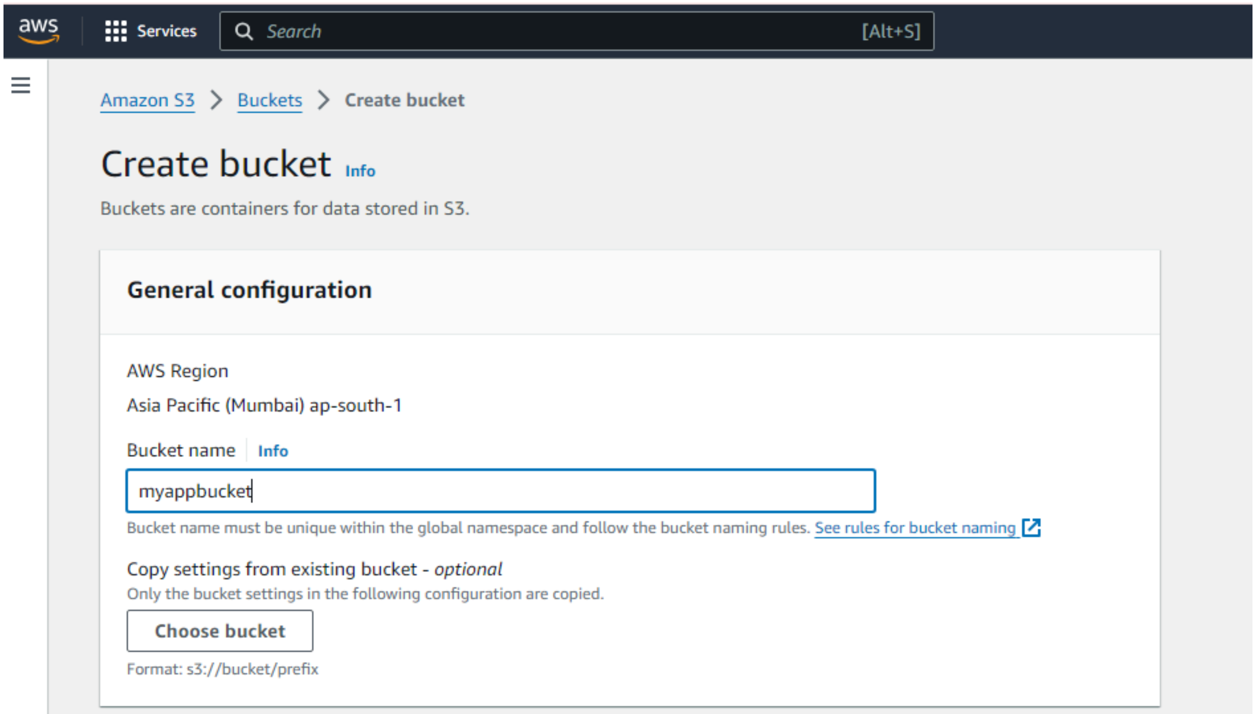
```
PS C:\Users\91916\Desktop\react\myapp> npm run build

> myapp@0.1.0 build
> react-scripts build

Creating an optimized production build...
One of your dependencies, babel-preset-react-app, is importing the
"@babel/plugin-proposal-private-property-in-object" package without
declaring it in its dependencies. This is currently working because
"@babel/plugin-proposal-private-property-in-object" is already in your
node_modules folder for unrelated reasons, but it may break at any time.

babel-preset-react-app is part of the create-react-app project, which
is not maintianed anymore. It is thus unlikely that this bug will
ever be fixed. Add "@babel/plugin-proposal-private-property-in-object" to
your devDependencies to work around this error. This will make this message
go away.

Compiled with warnings.
```



Step 2: In AWS S3 bucket , we created a new bucket

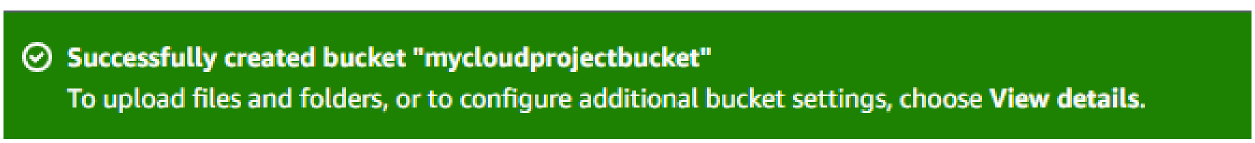


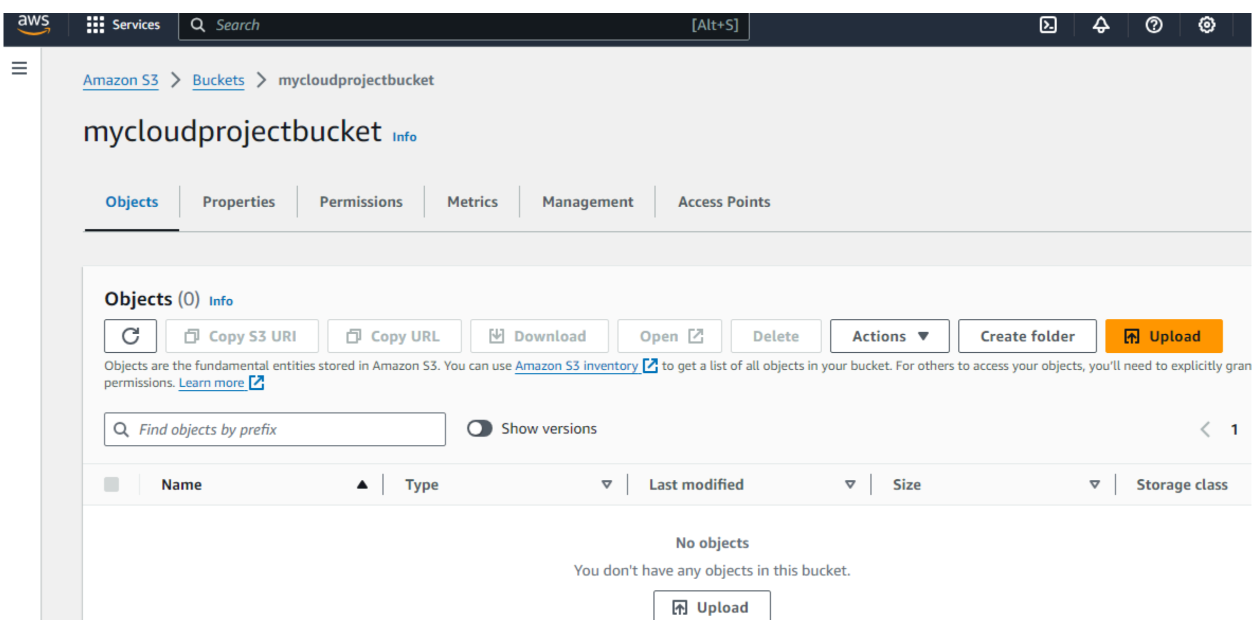
Created a Bucket:



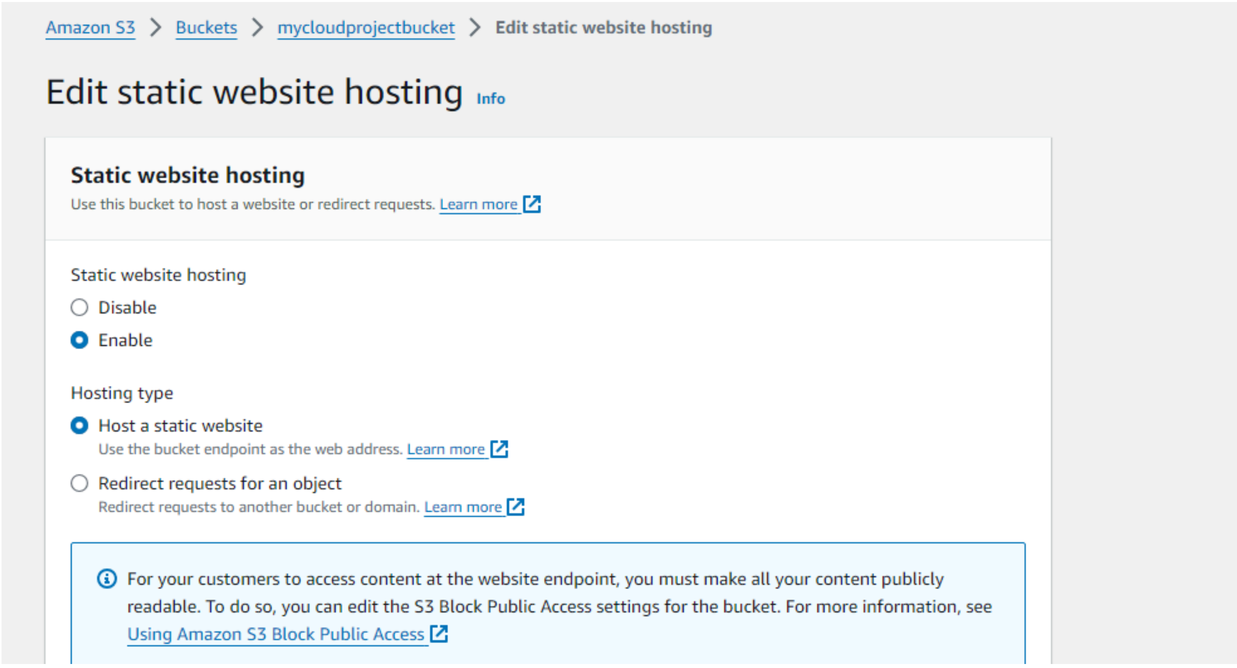
Note: Bucket versioning must be enabled

Then the bucket was created:

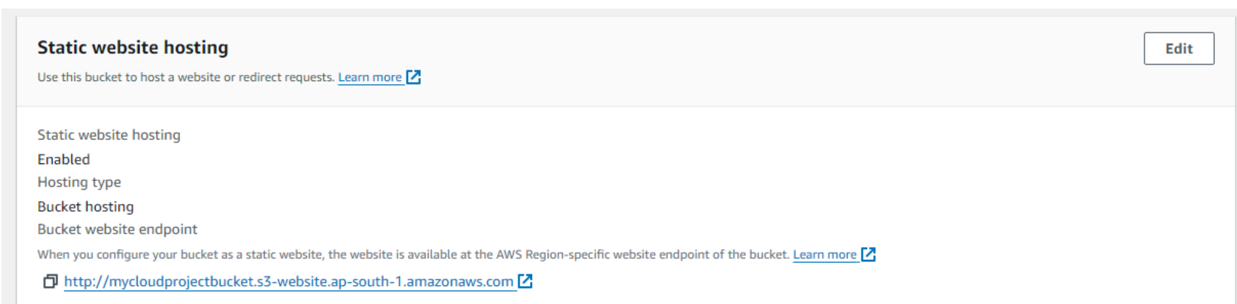




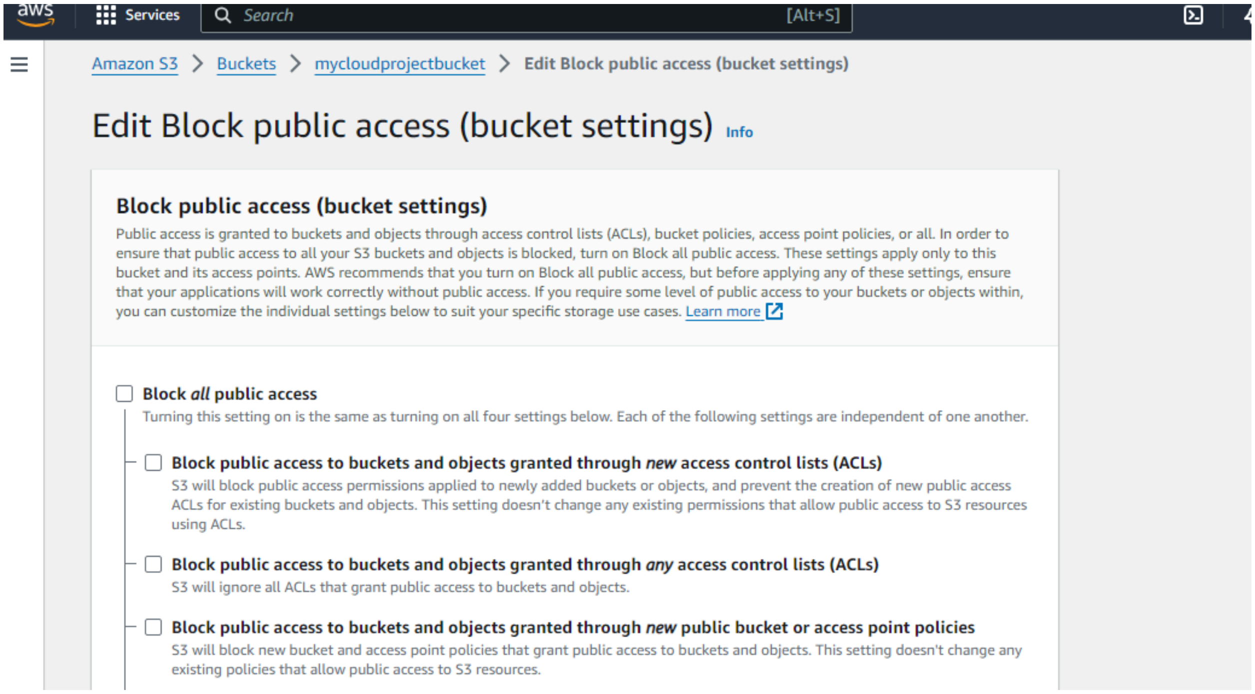
Step 3: Enabled static website hosting in the bucket



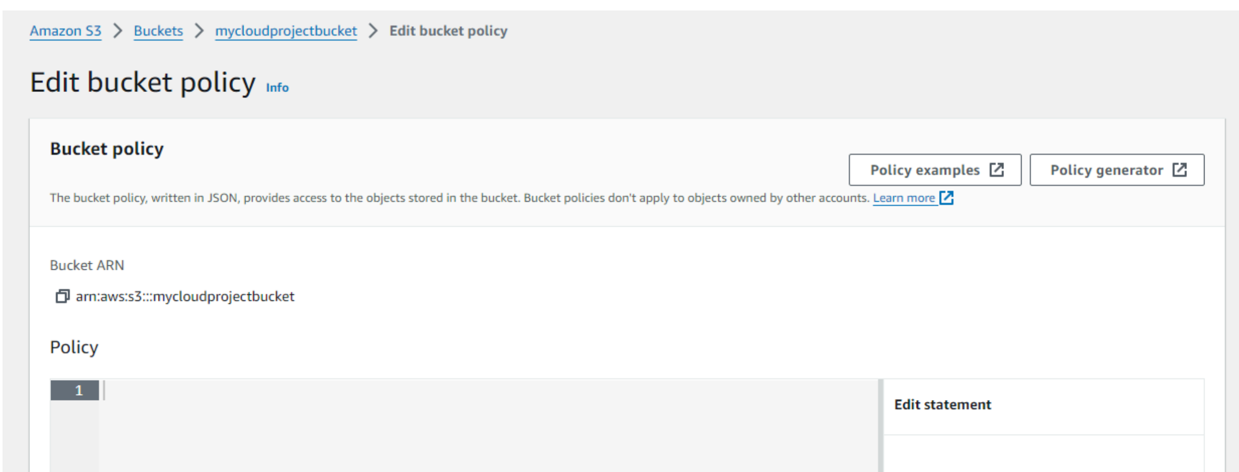
Step 4: After it was enabled , we got the url , which was used to deploy and host our web app.



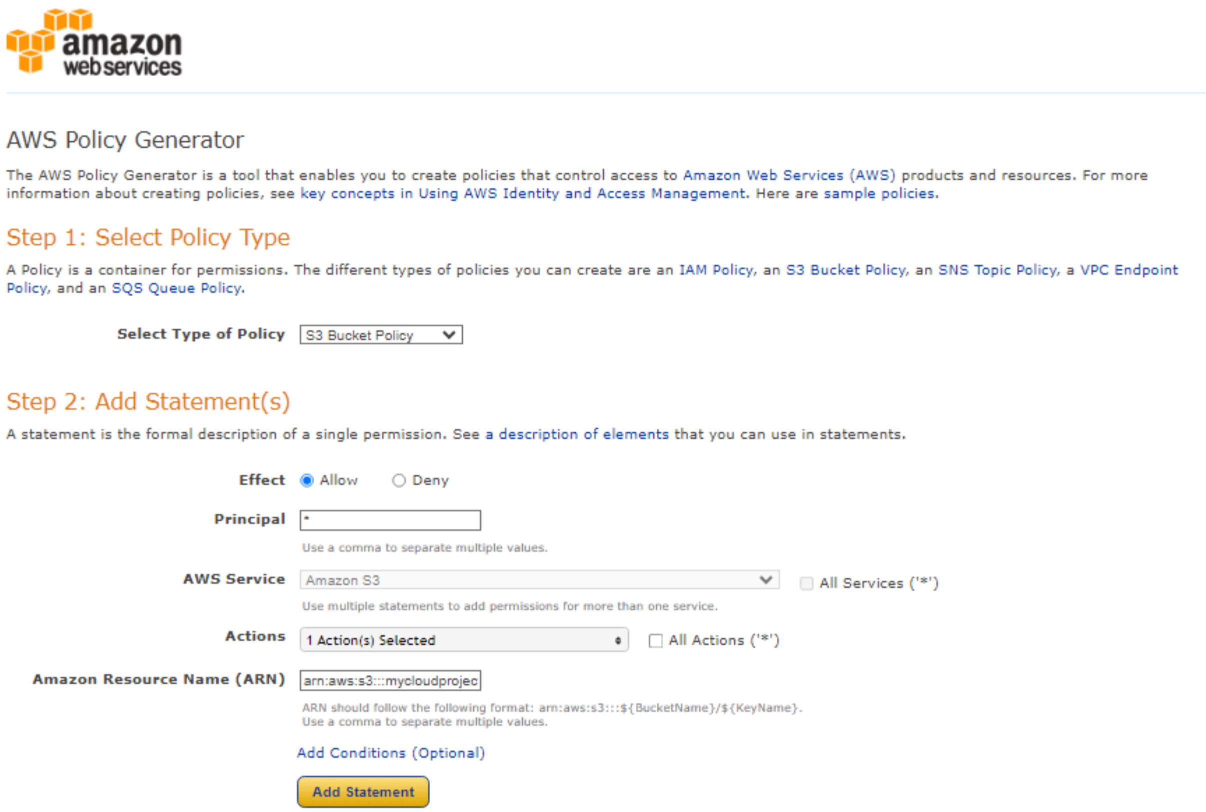
Step 5: In permissions , Disabled Block Public access to make the web application link publicly accessible



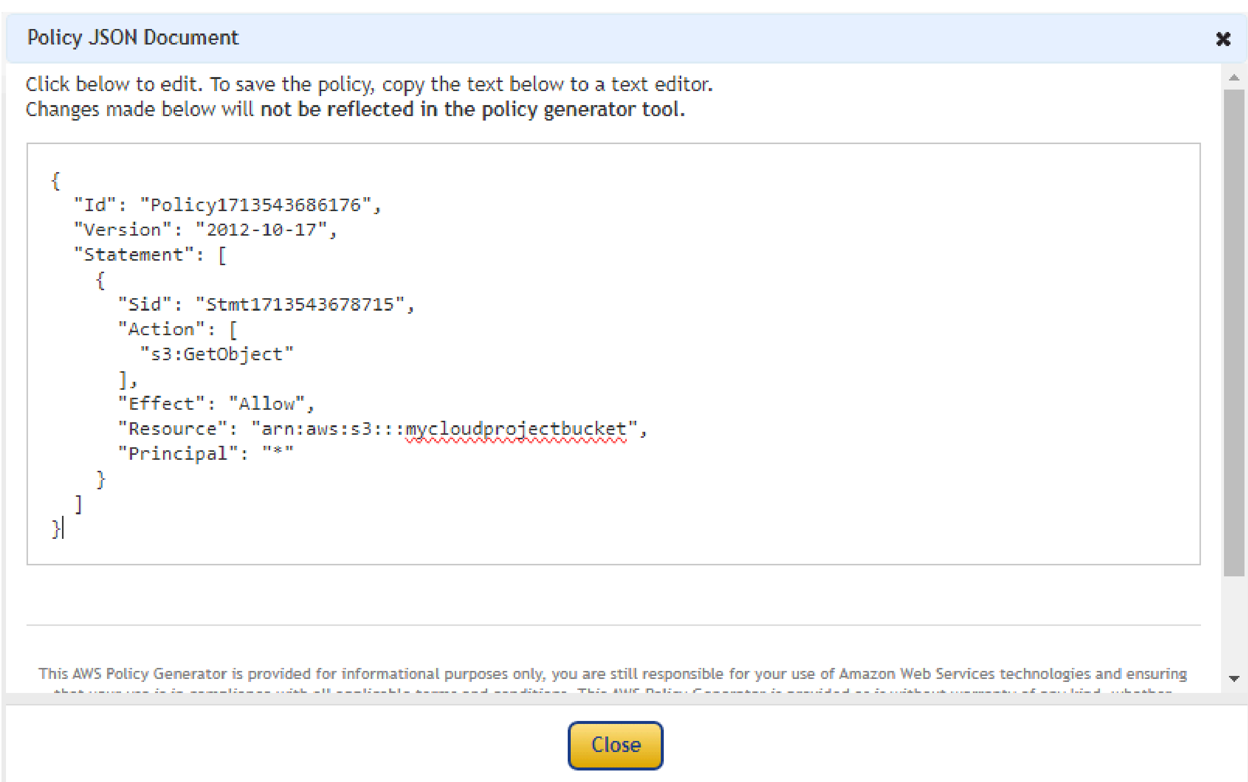
Step 6: To Generate a Bucket policy we used Policy generator



Amazon policy generator:



The policy was generated: This is the policy statement



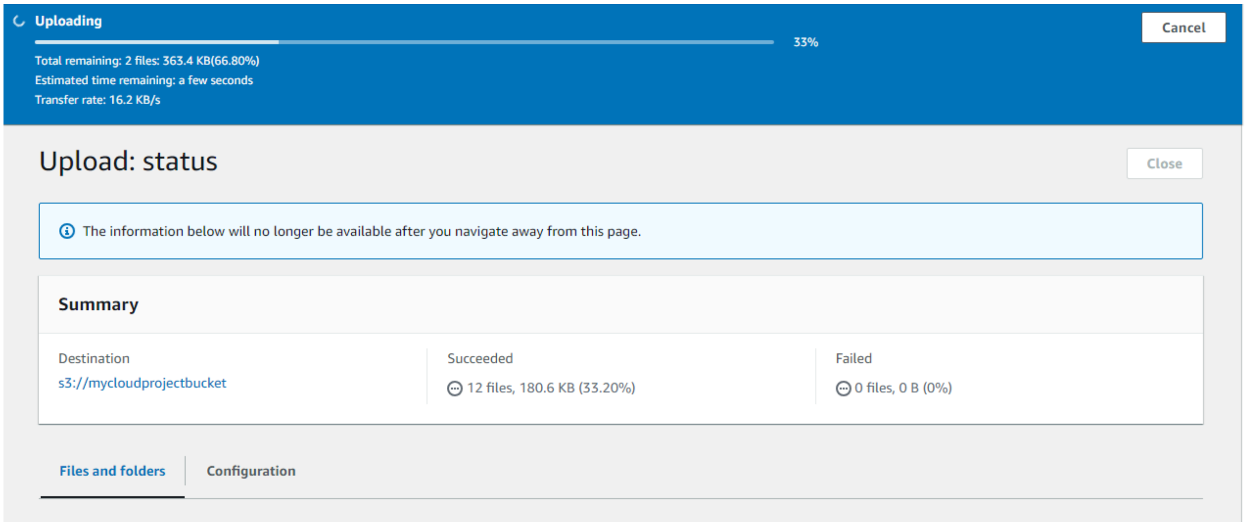
Policy

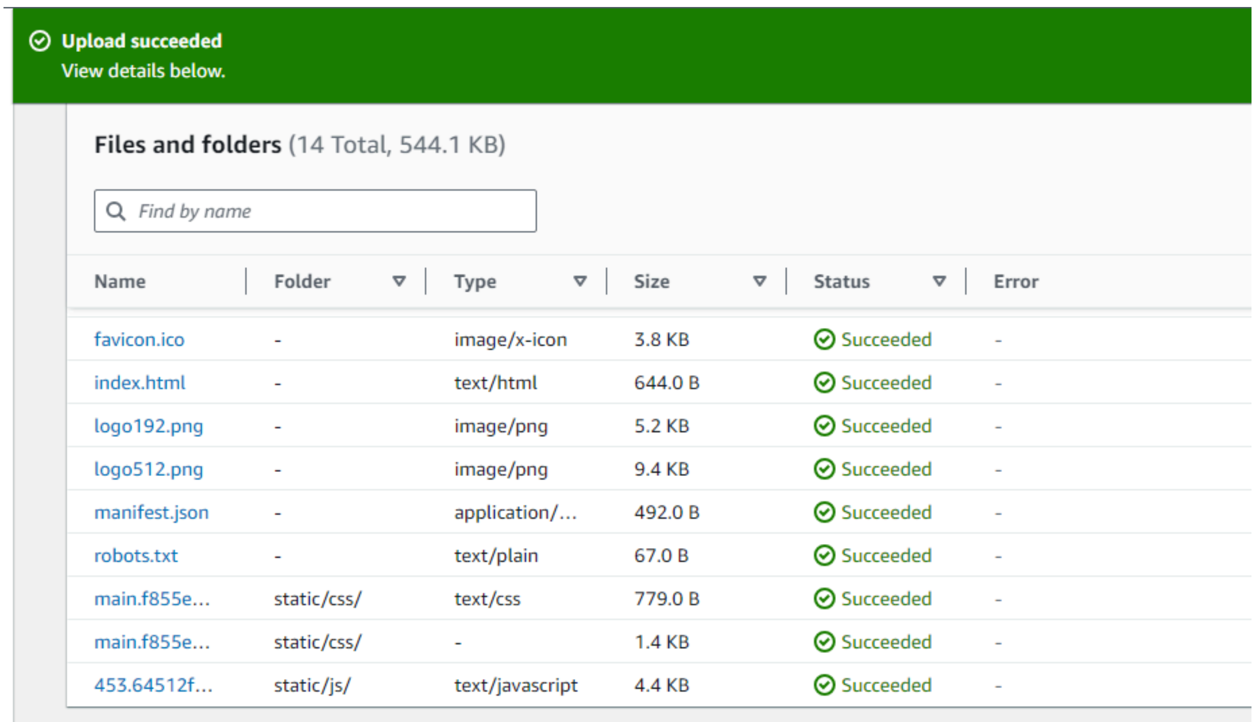
```
{
  "Id": "Policy1713543686176",
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1713543678715",
      "Action": [
        "s3:GetObject"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::mycloudprojectbucket/*",
      "Principal": "*"
    }
  ]
}
```

Step 7: Uploaded files in Objects in bucket from local machine build folder:

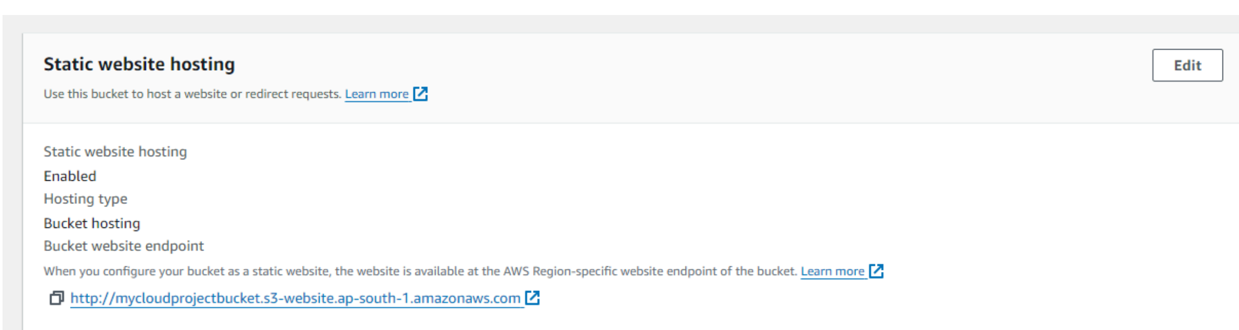
« Desktop › react › myapp › build ›   Search build

Name	Date modified	Type
static	19-04-2024 21:40	File folder
asset-manifest	19-04-2024 21:40	JSON Source File
favicon	18-04-2024 21:19	Icon
index	19-04-2024 21:40	Chrome HTML Do
logo192	18-04-2024 21:19	PNG File
logo512	18-04-2024 21:19	PNG File
manifest	18-04-2024 21:19	JSON Source File
robots	18-04-2024 21:19	Text Document

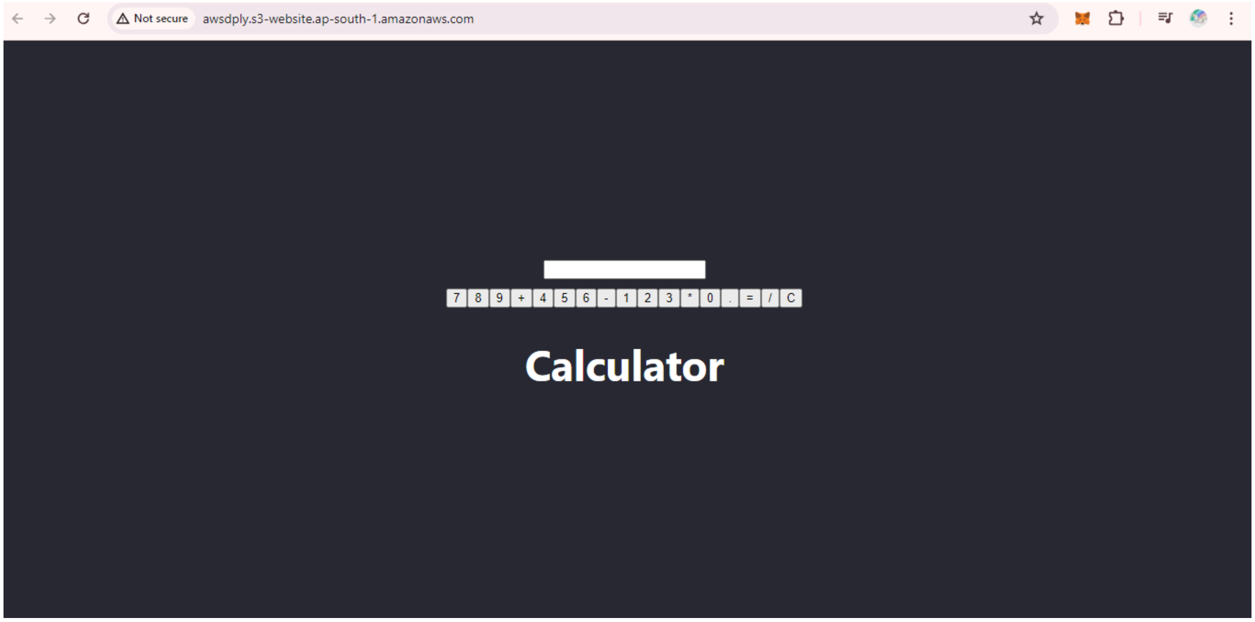




Step 8: The App was successfully deployed, we can view the frontend using static website link in the bucket:



Frontend looks like this:



Link to the website: http://awsdply.s3-website.ap-south-1.amazonaws.com/

Case 3: Deploying Machine learning model using Sagemaker, S3 bucket and Lambda function

Amazon SageMaker: Amazon SageMaker is a fully managed service that brings together a broad set of tools to enable high-performance, low-cost machine learning (ML) for any use case. With SageMaker, you can build, train and deploy ML models at scale using tools like notebooks, debuggers, profilers, pipelines, MLOps, and more – all in one integrated development environment (IDE). SageMaker supports governance requirements with simplified access control and transparency over your ML projects. In addition, you can build your own FMs, large models that were trained on massive datasets, with purpose-built tools to fine-tune, experiment, retrain, and deploy FMs. SageMaker offers access to hundreds of pretrained models, including publicly available FMs, that you can deploy with just a few clicks.

AWS Lambda: AWS Lambda is a compute service that runs your code in response to events and automatically manages the compute resources, making it the fastest way to turn an idea into a modern, production, serverless applications.

Both Sagemaker and Lambda are Platform as a Service products.

XGBoost: XGBoost stands for eXtreme Gradient Boosting, and it's an open-source library that provides an efficient and scalable implementation of gradient boosting. Gradient boosting is a machine learning technique used for regression and classification tasks. It is an implementation of gradient boosting decision trees designed for speed and performance. It's an ensemble learning method, which means it combines the predictions of multiple individual models (typically decision trees) to produce a more accurate and robust final prediction. It's particularly well-suited for structured/tabular data and is known for its effectiveness in handling large datasets.

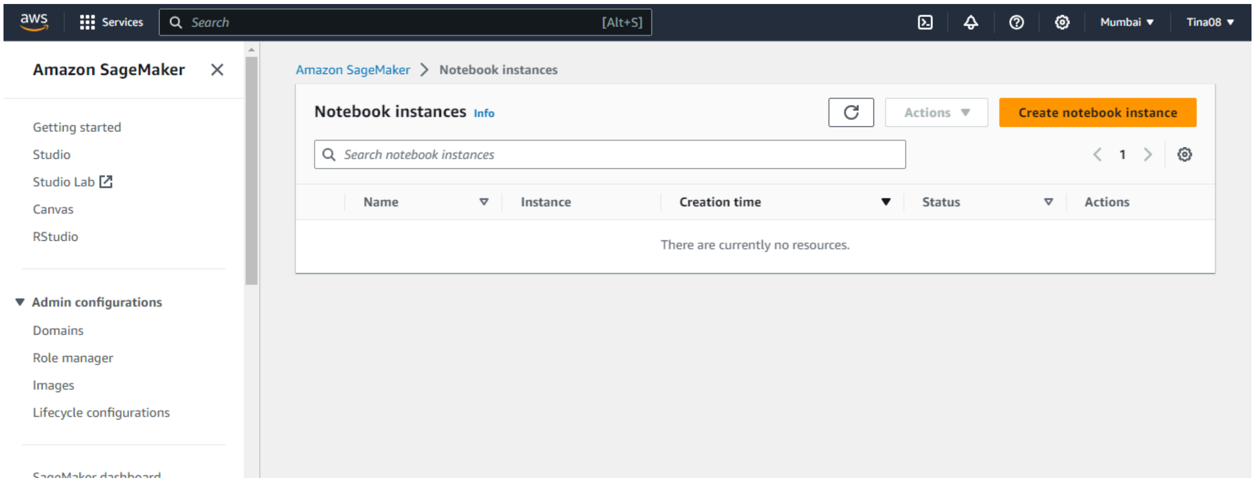
Overview of deployment:

We have deployed our Machine learning XGboost model which is a Gradient boosting algorithm using Sagemaker , S3 bucket for storage and Lambda function to run the code. We created a SageMaker notebook, thereafter creating a SageMaker endpoint, and executing a Lambda function.

Methodology / Process of deployment:

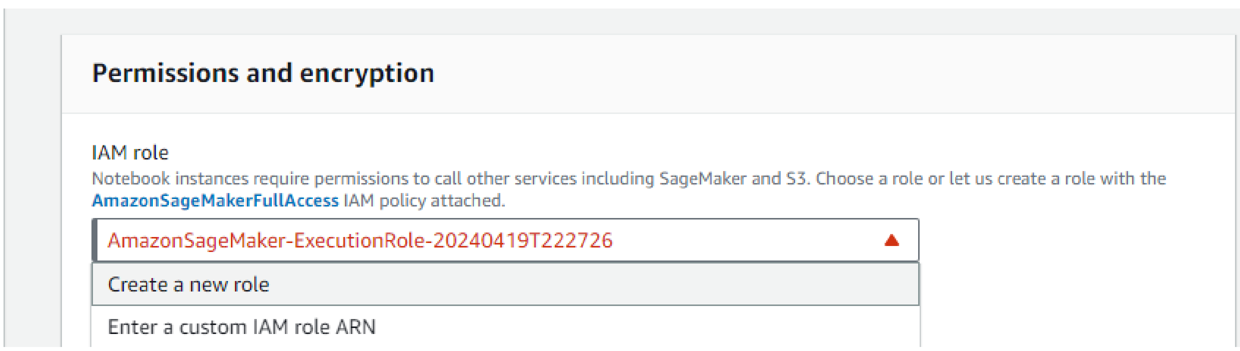
Step 1: Go to Amazon Sagemaker

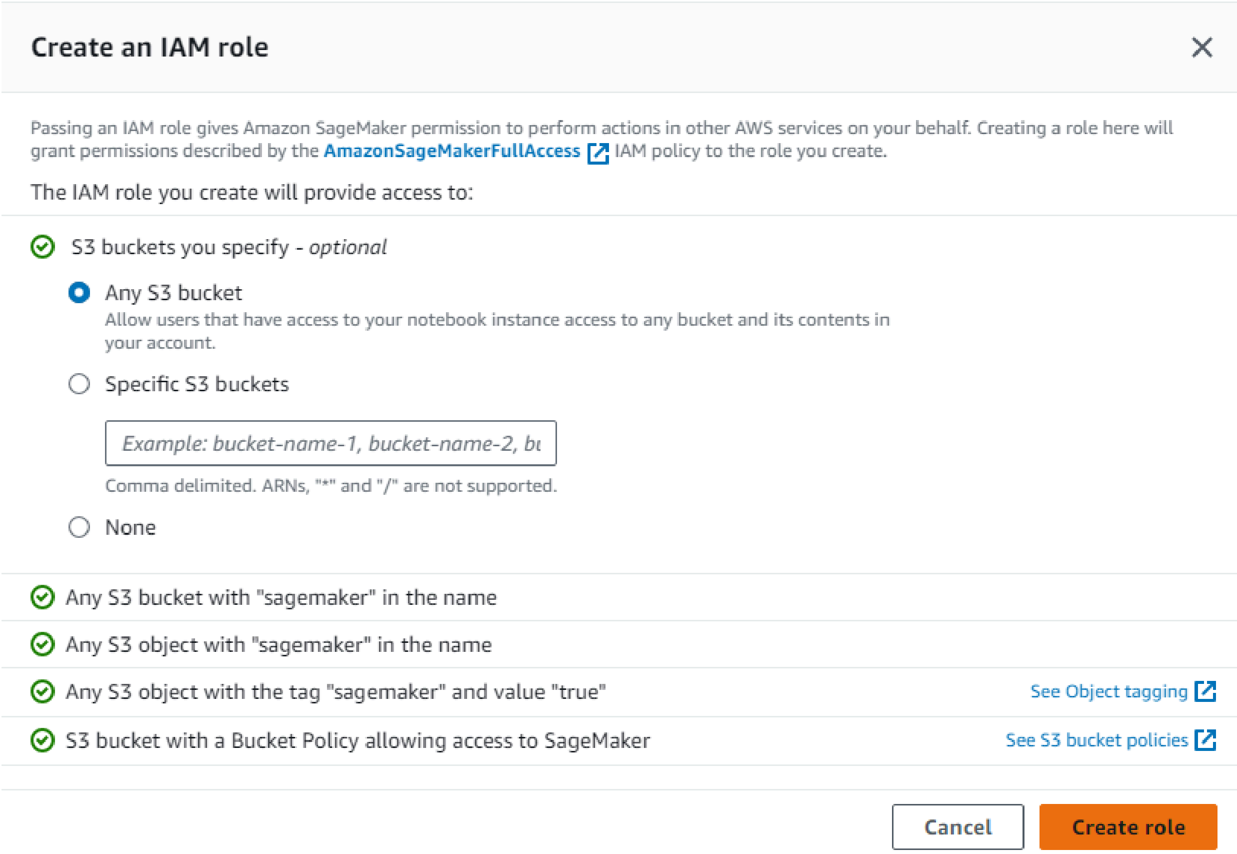
Step 2: Click on Notebook instance to create notebook instance

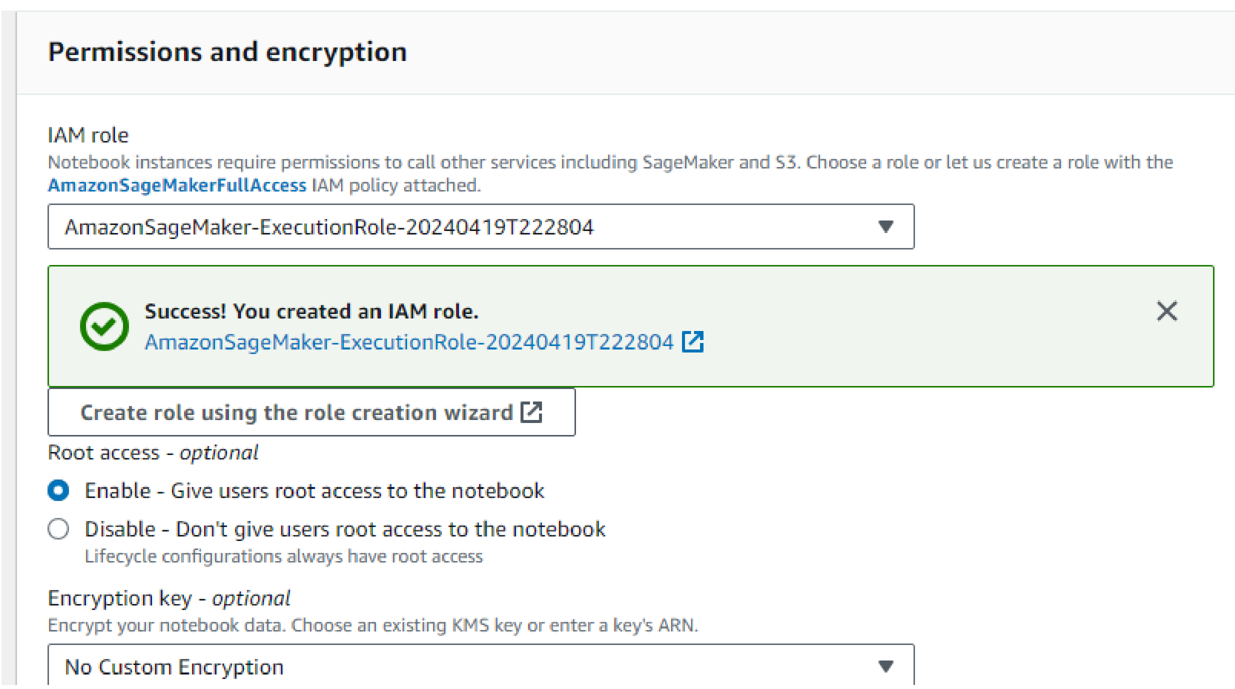


Step 3: We created the notebook in order to build and deploy the machine learning model in sagemaker.

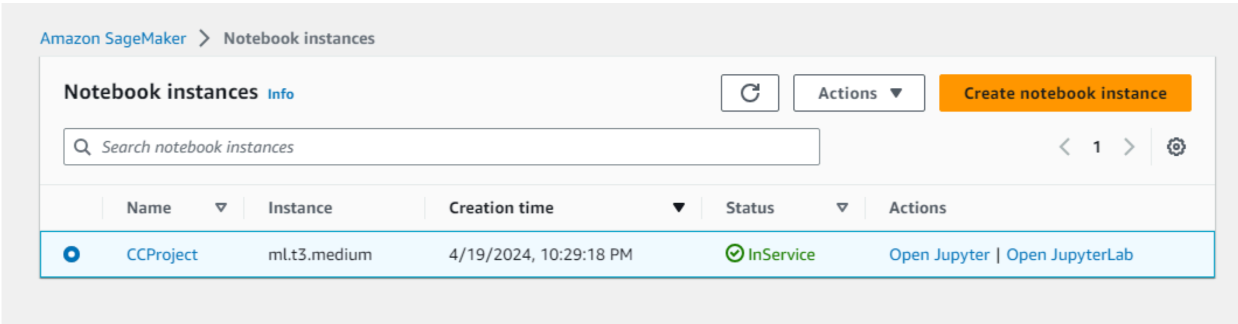
Step 4: Created a new role in IAM:



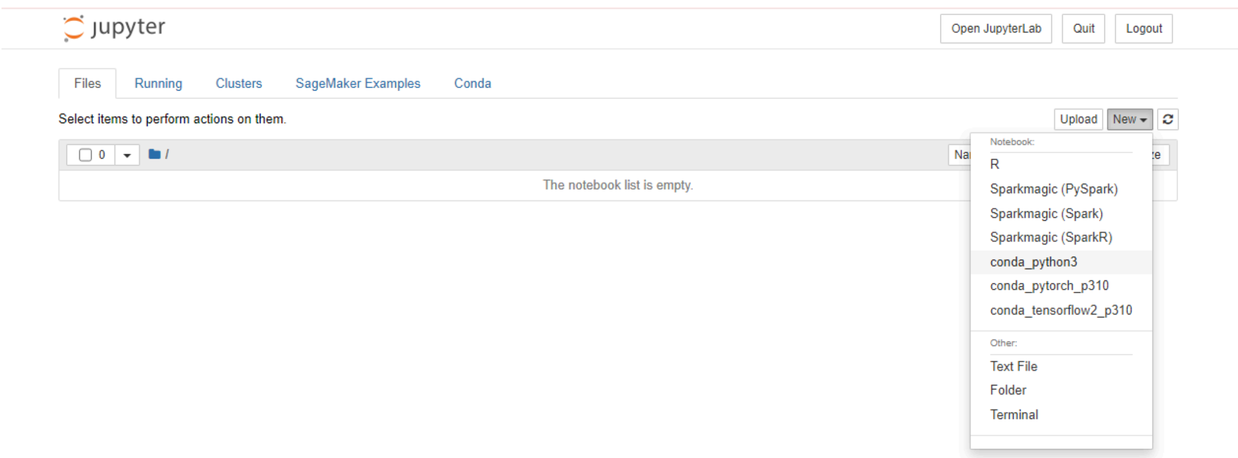

Create an IAM role
Passing an IAM role gives Amazon SageMaker permission to perform actions in other AWS services on your behalf. Creating a role here will grant permissions described by the AmazonSageMakerFullAccess IAM policy to the role you create.
The IAM role you create will provide access to:
S3 buckets you specify - optional
Any S3 bucket
Allow users that have access to your notebook instance access to any bucket and its contents in your account.
Specific S3 buckets
Example: bucket-name-1, bucket-name-2, bu
Comma delimited. ARNs, "*" and "/" are not supported.
None
Any S3 bucket with "sagemaker" in the name
Any S3 object with "sagemaker" in the name
Any S3 object with the tag "sagemaker" and value "true"
See Object tagging
S3 bucket with a Bucket Policy allowing access to SageMaker
See S3 bucket policies
Cancel
Create role


Permissions and encryption
IAM role
Notebook instances require permissions to call other services including SageMaker and S3. Choose a role or let us create a role with the AmazonSageMakerFullAccess IAM policy attached.
AmazonSageMaker-ExecutionRole-20240419T222804
Success! You created an IAM role.
AmazonSageMaker-ExecutionRole-20240419T222804
Create role using the role creation wizard
Root access - optional
Enable - Give users root access to the notebook
Disable - Don't give users root access to the notebook
Lifecycle configurations always have root access
Encryption key - optional
Encrypt your notebook data. Choose an existing KMS key or enter a key's ARN.
No Custom Encryption

Step 5: A Notebook instance was created. Open Jupyter notebook link redirects us to the jupyter notebook.

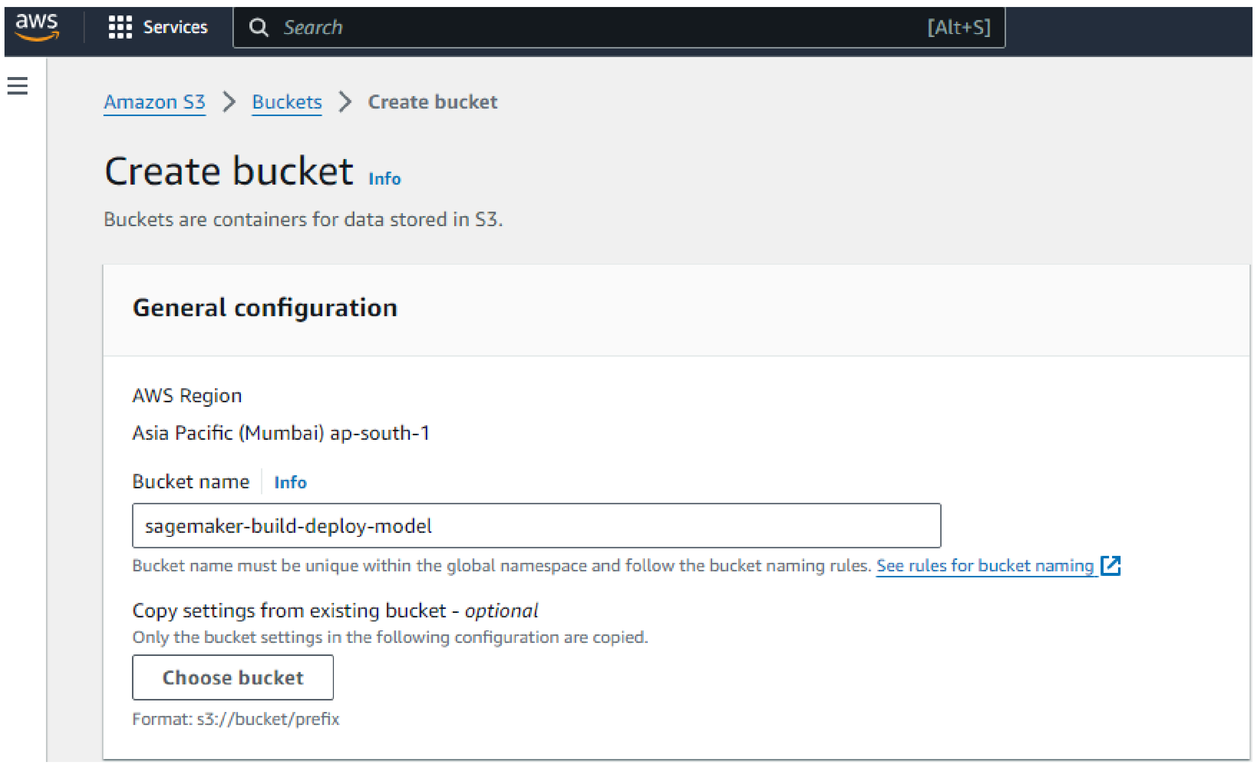


Step 6: After opening jupyter notebook : New>Conda python 3

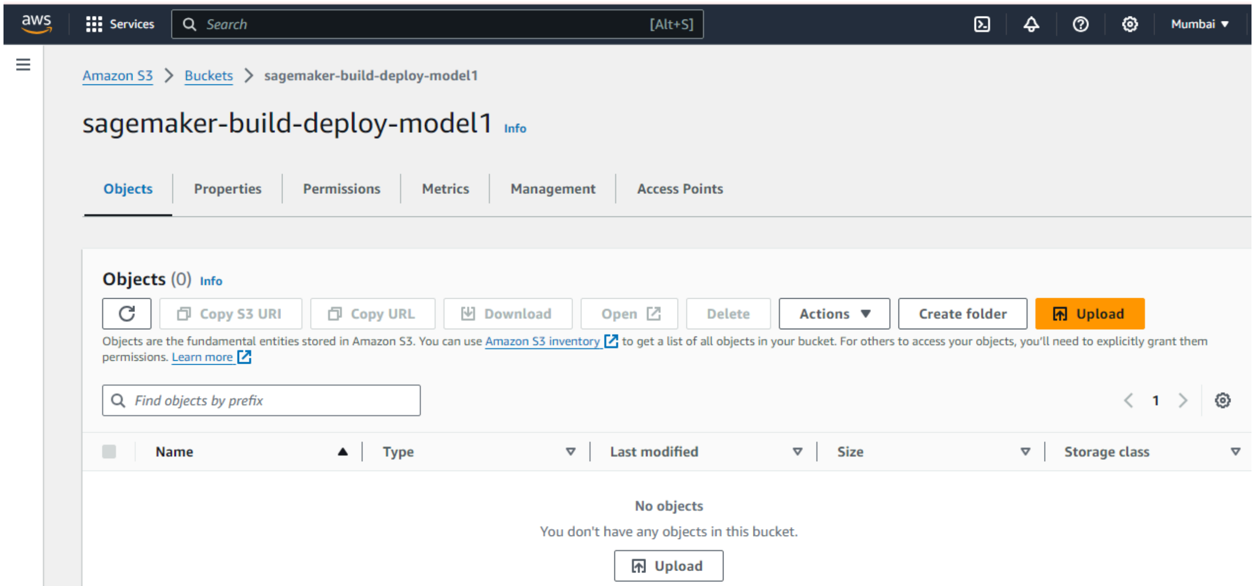


After running this code , our data was imported in the notebook.

Step 7: Created an S3 Bucket to store the data:



Bucket is created

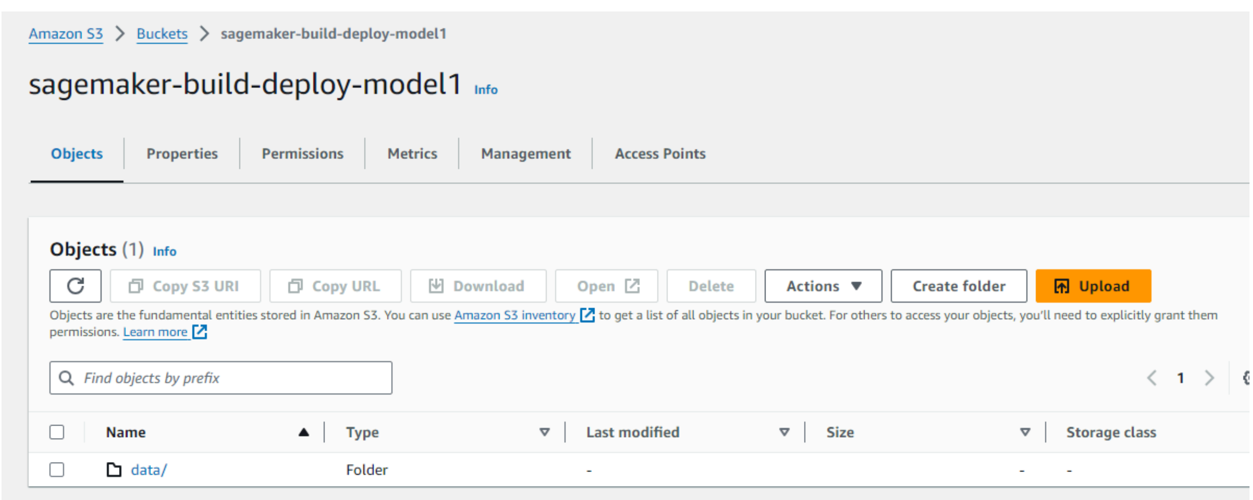


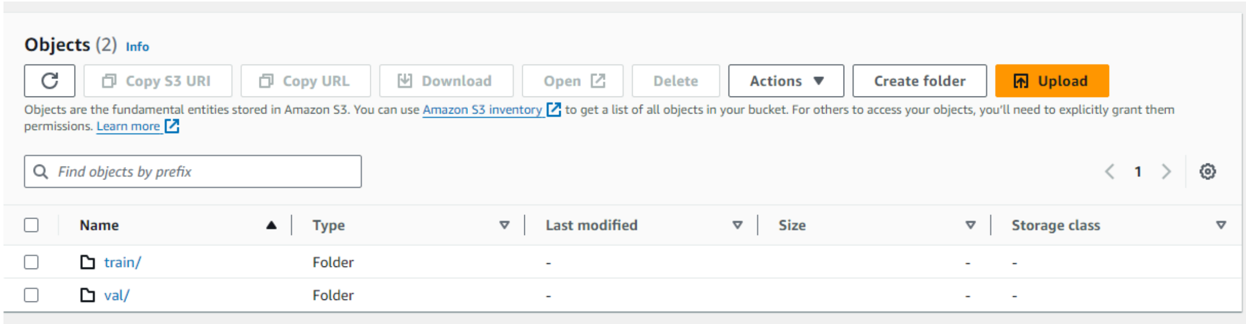
Step 8: The data imported in the notebook will be now stored in this bucket:

Moving Data Into S3 Bucket

In [4]:

```python
import boto3
bucket_name='sagemaker-build-deploy-model1'

train_data.to_csv('data.csv',header=False,index=False)
key = 'data/train/data'
url = 's3://{}/{}'.format(bucket_name,key)
boto3.Session().resource('s3').Bucket(bucket_name).Object(key).upload_file('data.csv')

val_data.to_csv('data.csv',header=False,index=False)
key = 'data/val/data'
url = 's3://{}/{}'.format(bucket_name,key)
boto3.Session().resource('s3').Bucket(bucket_name).Object(key).upload_file('data.csv')
```





Step 9: Created the Model:

Model Creation

```
import sagemaker
from sagemaker.amazon.amazon_estimator import get_image_uri
from sagemaker import get_execution_role

key='model/xgb_model'
s3_output_location=url='s3://{}/{}'.format(bucket_name,key)

xgb_model=sagemaker.estimator.Estimator(
    get_image_uri(boto3.Session().region_name,'xgboost'),
    get_execution_role(),
    train_instance_count=1,
    train_instance_type='ml.m4.xlarge',
    train_volume_size=5,
    output_path=s3_output_location,
    segemaker_session=sagemaker.Session()
)

xgb_model.set_hyperparameters(
        max_depth=5,
        eta=0.2,
        gamma=4,
        min_child_weight=6,
        silent=0,
        objective='multi:softmax',
        num_class=3,
```

Step 10: Then we Trained and Deployed our model inside the notebook:

Training the Model

```
[*]: train_data='s3://{}/{}'.format(bucket_name,'data/train')
val_data='s3://{}/{}'.format(bucket_name,'data/val')

train_channel=sagemaker.session.s3_input(train_data,content_type='text/csv')
val_channel=sagemaker.session.s3_input(val_data,content_type='text/csv')

data_channels={'train':train_channel,'validation':val_channel}

xgb_model.fit(inputs=data_channels)
```

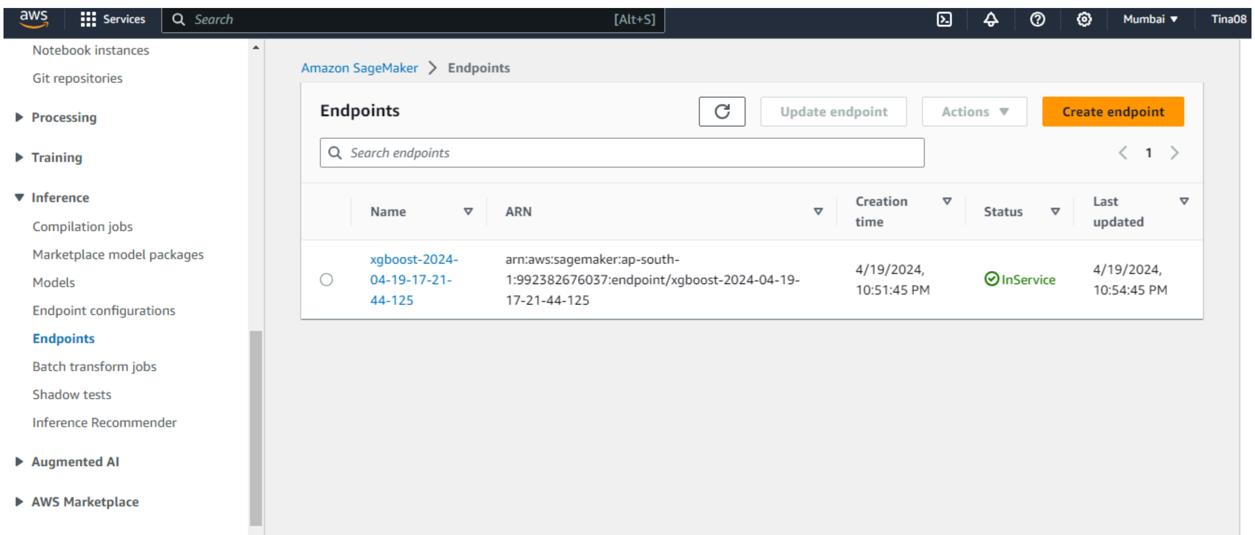
```
The class sagemaker.session.s3_input has been renamed in sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
The class sagemaker.session.s3_input has been renamed in sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
INFO:sagemaker:Creating training-job with name: xgboost-2024-04-19-17-18-31-617

2024-04-19 17:18:32 Starting - Starting the training job...
2024-04-19 17:18:47 Starting - Preparing the instances for training......
```

Deploying the Model

```
[*]: xgb_predictor=xgb_model.deploy(initial_instance_count=1,
                               instance_type='ml.m4.xlarge')
```

Step 11: After Model was deployed we could check the endpoints created in Sagemaker

Click on inference > Click on Endpoints



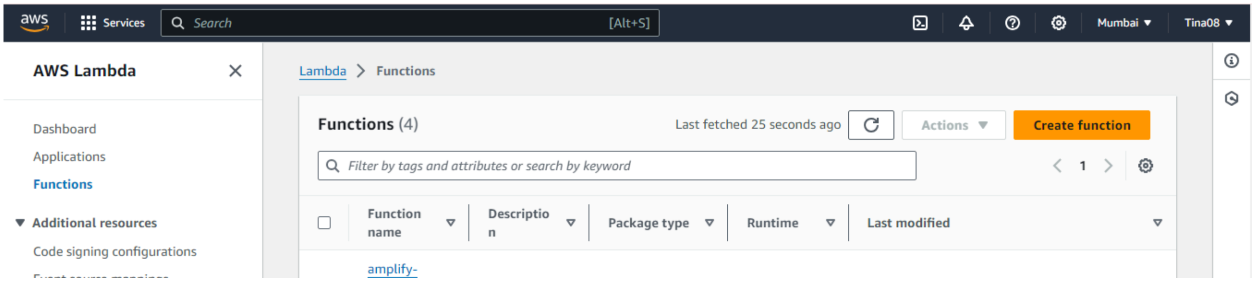
Note: Endpoints were created for predictions

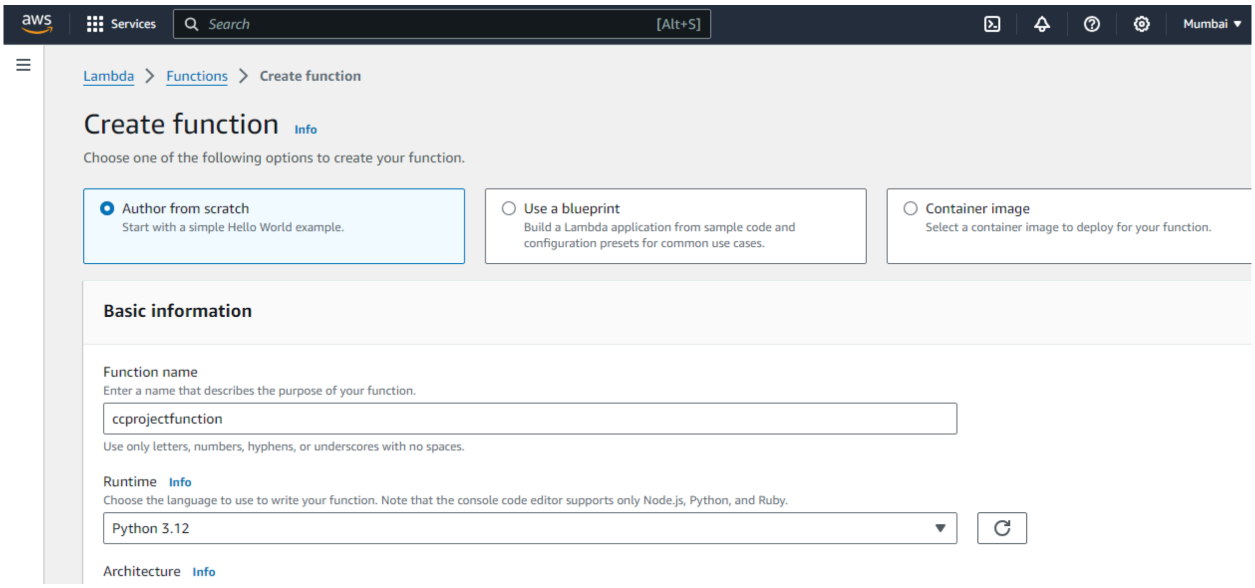
<u>To create an API , we used Lambda function:</u>

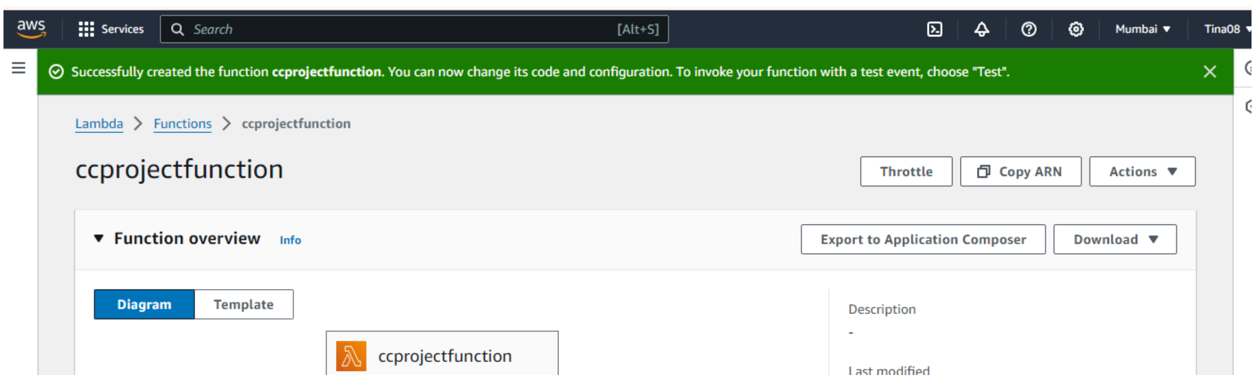
Steps are as follows :

Step 12: Created a new function

Lambda Homepage:

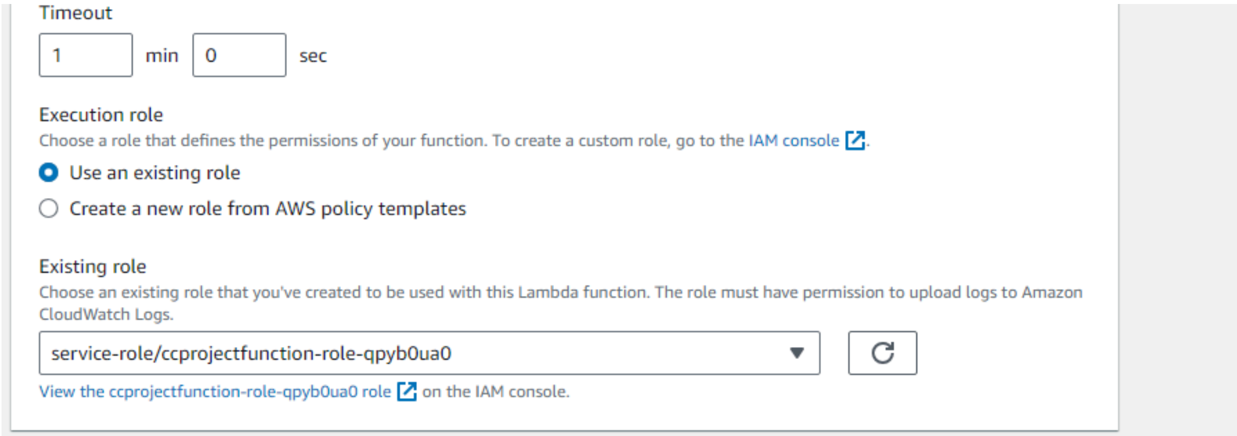






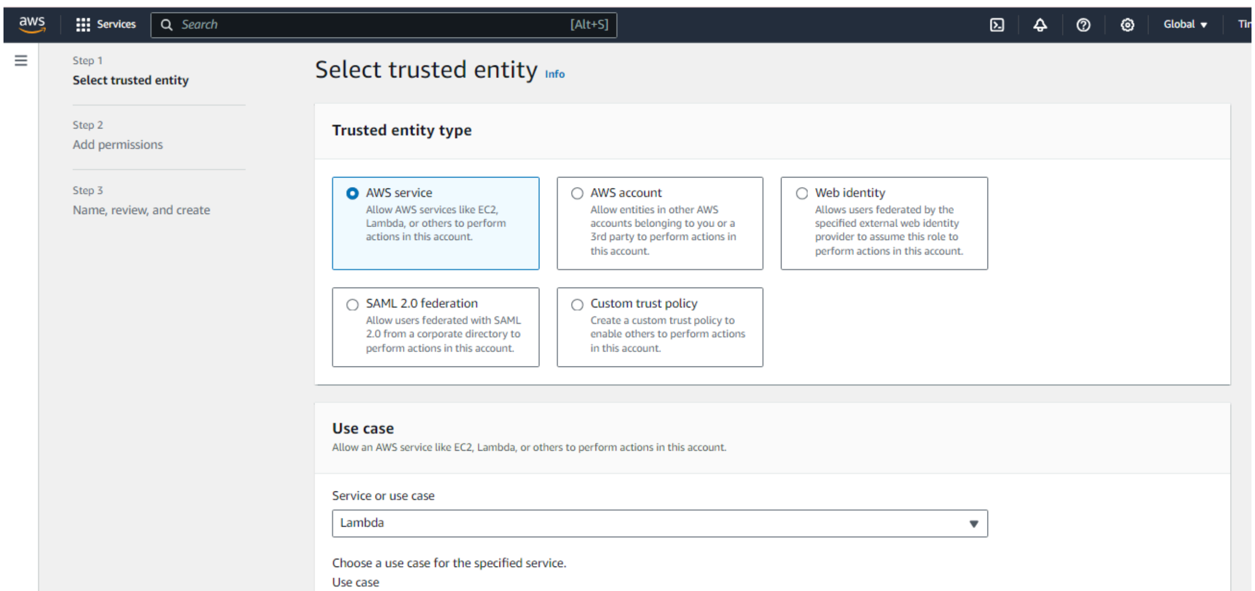
Step 13: Then we wrote code for invoking endpoints inside lambda

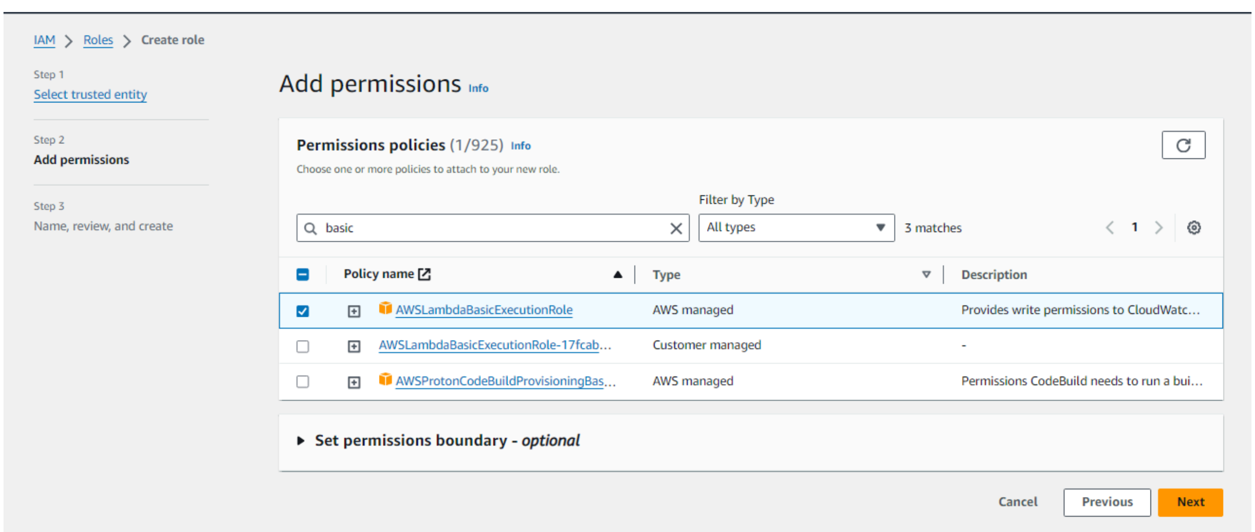
Step 14: We will first increase the time out period
Lambda > Functions > ccprojectfunction > Edit basic settings

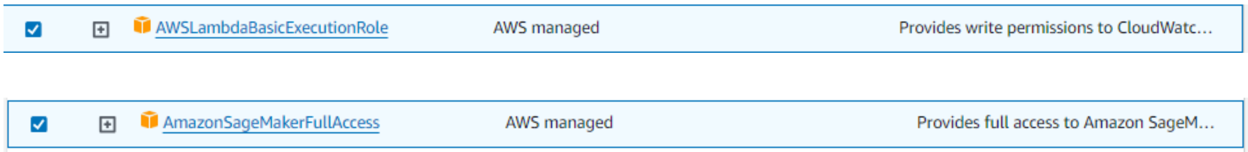


Step 15: For giving necessary permissions , we created a New role in IAM

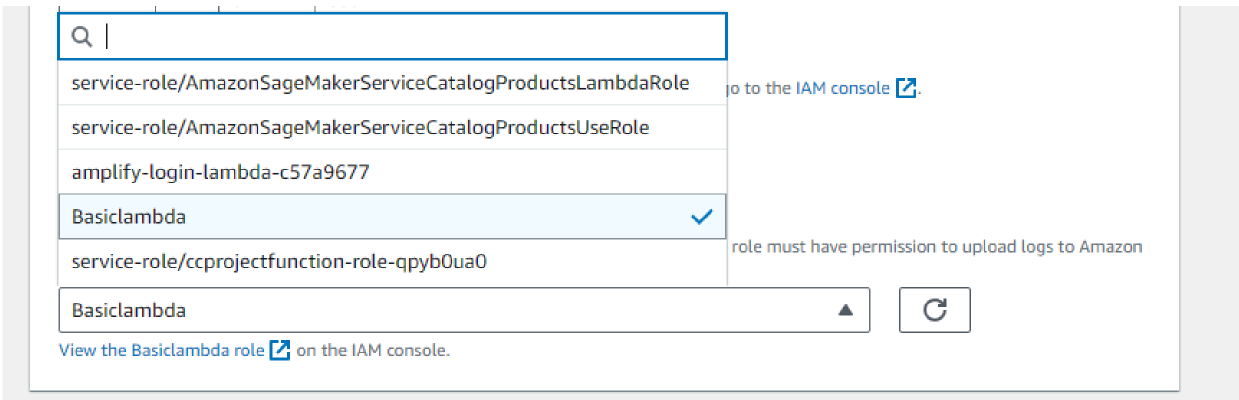
Selected lambda as the service



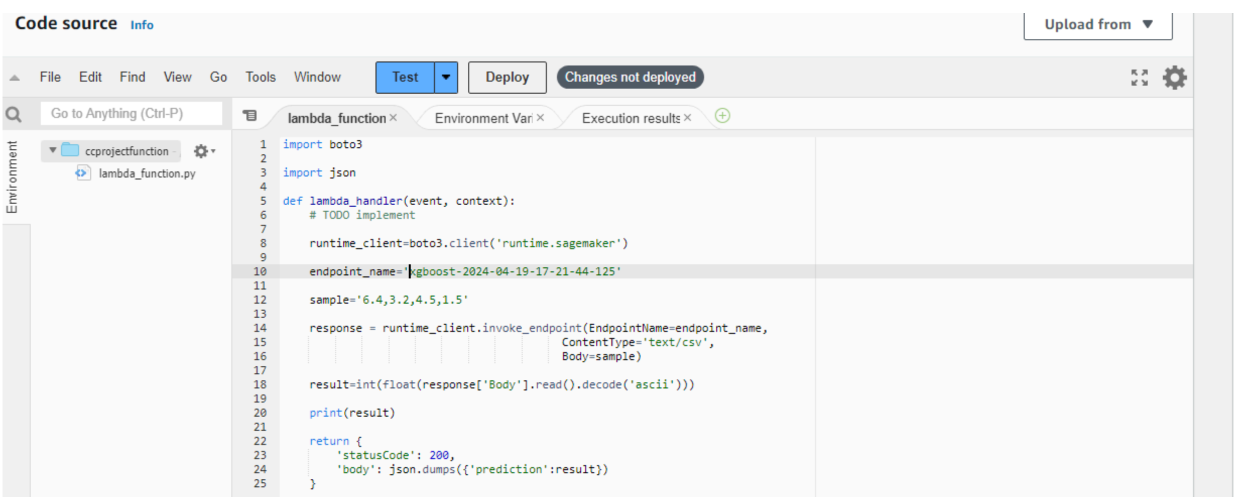


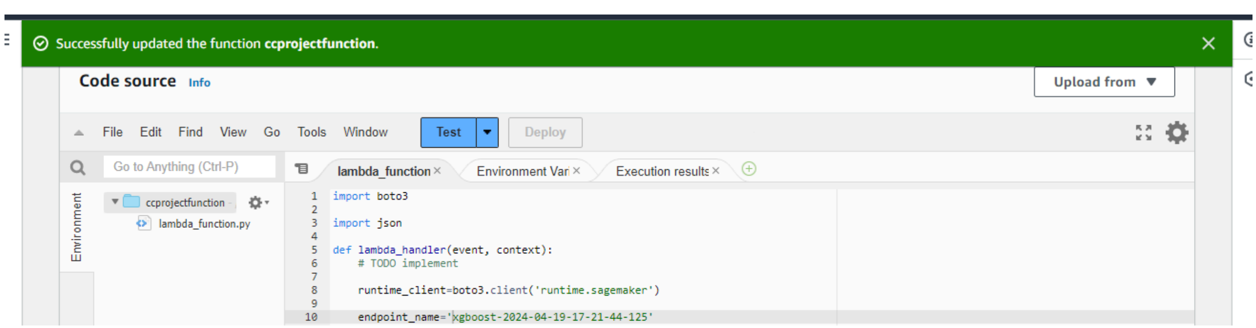


Step 16: Assigned the role Basiclambda:

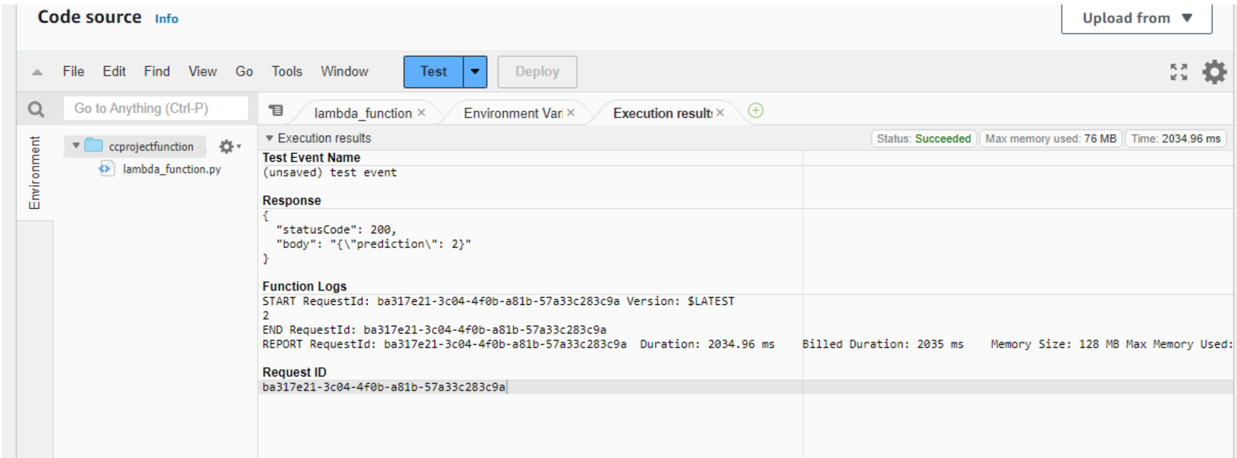


Step 17: Then we deployed the code and then tested it inside the Lambda function:



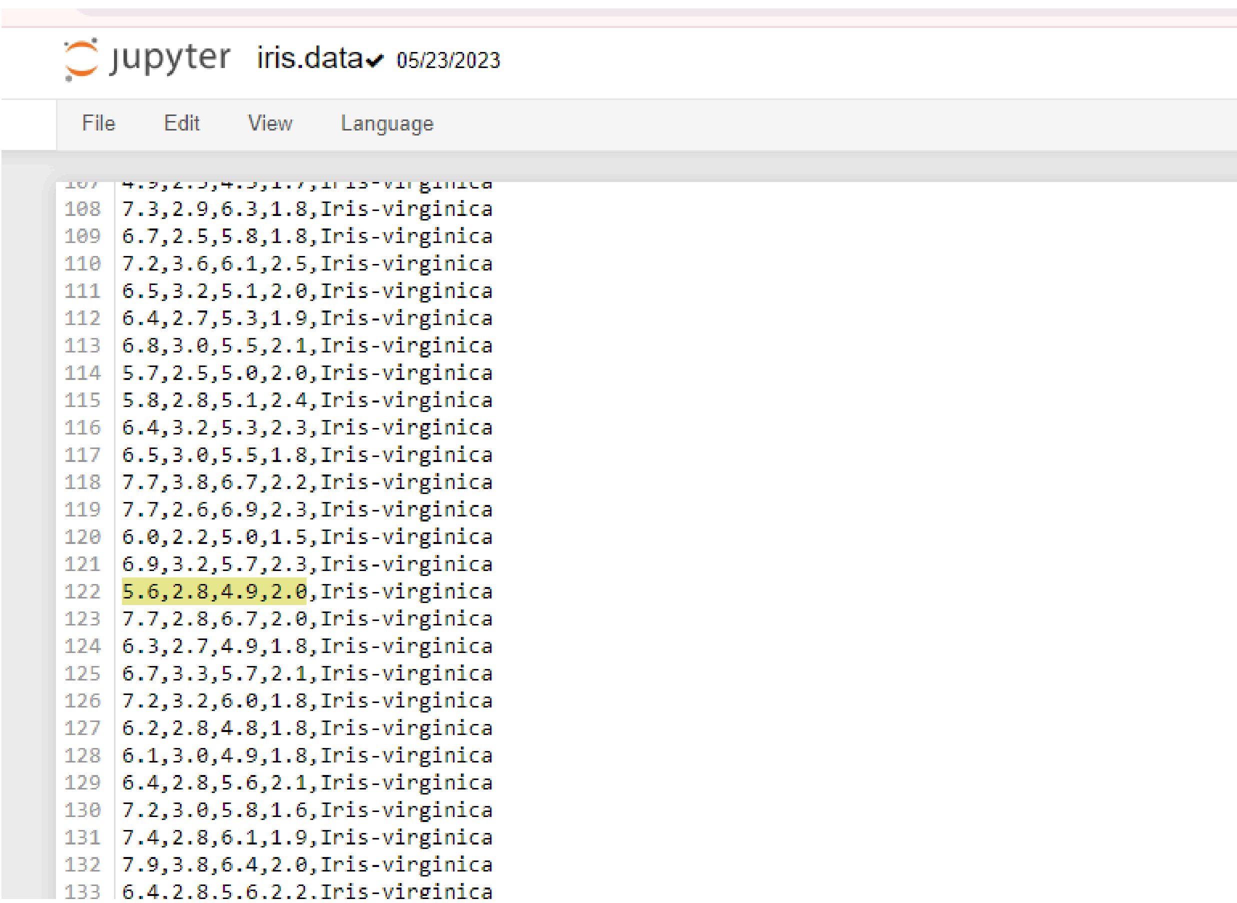


18. Execution result:



19. In the code of lambda - function when we manually enter the sample points, from iris dataset , it predicted the correct class

<u>Dataset:</u>



<u>Sample points:</u>

```
lambda_function ×   Environment Var ×   Execution results ×
```

```python
import boto3

import json

def lambda_handler(event, context):
    # TODO implement

    runtime_client=boto3.client('runtime.sagemaker')

    endpoint_name='xgboost-2024-04-19-17-21-44-125'

    sample='6.6,2.8,4.9,2.0'

    response  = runtime_client.invoke_endpoint(EndpointName=endpoint_name,
                                               ContentType='text/csv',
                                               Body=sample)

    result=int(float(response['Body'].read().decode('ascii')))

    print(result)

    return {
        'statusCode': 200,
        'body': json.dumps({'prediction':result})
    }
```

Prediction:

```
lambda_function ×   Environment Var ×   Execution result ×

▾ Execution results                                              Status: Succeeded   Max
Test Event Name
(unsaved) test event

Response
{
  "statusCode": 200,
  "body": "{\"prediction\": 1}"
}

Function Logs
START RequestId: ba8e03d1-4fbb-4a10-8ff7-928be780668e Version: $LATEST
1
END RequestId: ba8e03d1-4fbb-4a10-8ff7-928be780668e
REPORT RequestId: ba8e03d1-4fbb-4a10-8ff7-928be780668e  Duration: 1964.34 ms    Billed Duration: 1965 ms    Me

Request ID
ba8e03d1-4fbb-4a10-8ff7-928be780668e
```

Note : Iris-setosa was assigned the index 0 , iris-virginica was assigned the index as 1 and iris-versicolor as 2. The model correctly predicted the class iris-virginica using the sample points.

```
In [3]:  # read data
         import pandas  as pd
         data=pd.read_csv('data/iris.data',header=None)

         # convert to numerical values
         data[4]=data[4].replace('Iris-setosa', 0)
         data[4]=data[4].replace('Iris-virginica',1)
         data[4]=data[4].replace('Iris-versicolor',2)

         # shuffle
```

Inference: Storing the dataset in Amazon S3 provided easy access to data for training, validation, and inference. Using Amazon SageMaker for model training offered a scalable and managed environment. SageMaker made it straightforward to deploy trained models as endpoints, allowing for real-time inference. Deploying the XGBoost model as an endpoint enabled us to make predictions on new data with low latency. Creating a Lambda function for predictions enabled us serverless and scalable inference. Lambda functions can be triggered by various events, such as HTTP requests or scheduled events, making it suitable for a wide range of applications. By combining SageMaker, S3, and Lambda, we built an integrated and automated pipeline for machine learning inference. New data can be uploaded to S3, triggering the Lambda function to make predictions using the deployed model. This setup offers flexibility, scalability, and cost-effectiveness for real-world machine learning applications.

References:

https://docs.aws.amazon.com/sagemaker/

https://docs.aws.amazon.com/AmazonS3/latest/userguide/tutorials.html

https://docs.aws.amazon.com/lambda/latest/dg/welcome.html

https://docs.aws.amazon.com/sagemaker/latest/dg/automatic-model-tuning-ex-bucket.html