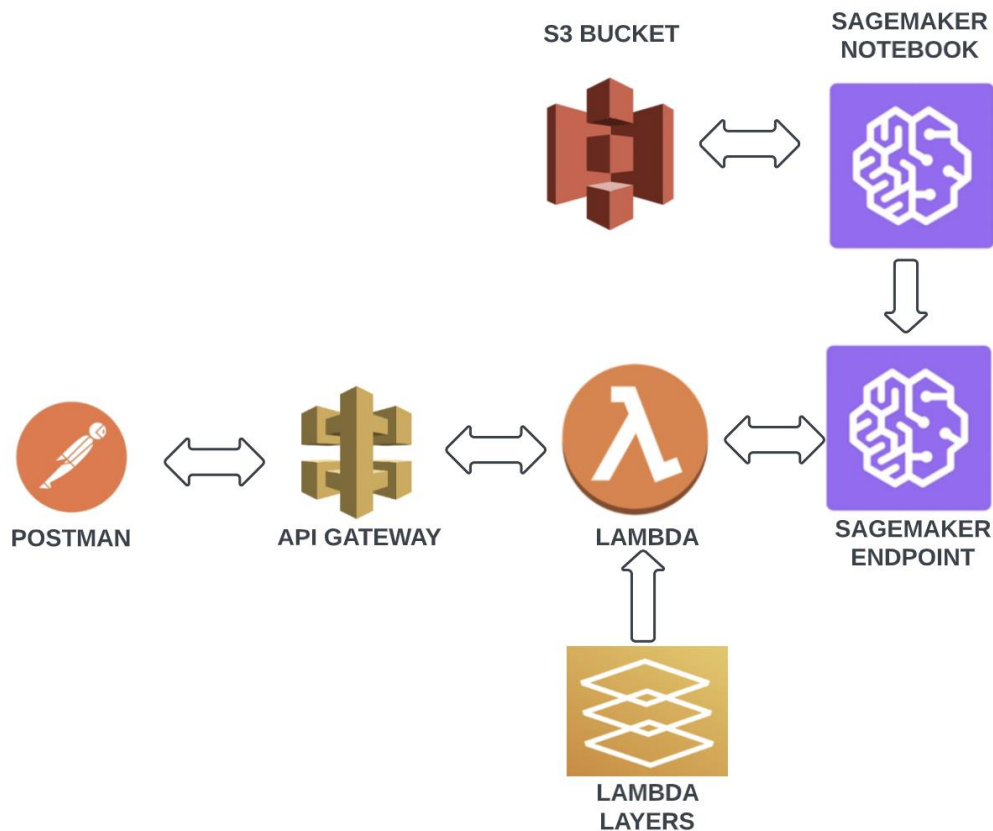


## Transfer Learning on MNIST using AWS Jagadish Arepalli

In this assignment we are going to do transfer learning on MNIST dataset and test with our own handwritten digits. So, to perform the above assignment we would use SageMaker, S3, Lambda, IAM and Amazon API Gateway.

### Architecture Diagram



### **Postman (User)**

It is an API building platform that simplifies development and testing of various API's. Here in the assignment, we use Postman to do a POST request to our API with image URL as the parameter.

### **AWS API Gateway**

Here we use this service to create a POST API which when called in return calls the lambda function.

## Lambda

Here our lambda function is configured in such a way that when it gets triggered with the image url it captures the url of the image and converts it to a numpy array which in return is given as the input to the sagemaker model endpoint. Then we get the response from the endpoint which in turn goes back to the API gateway and then to postman.

## AWS SageMaker

Here we use SageMaker Notebook instances to deploy our code and train our data and generate a machine learning model endpoint.

## AWS S3

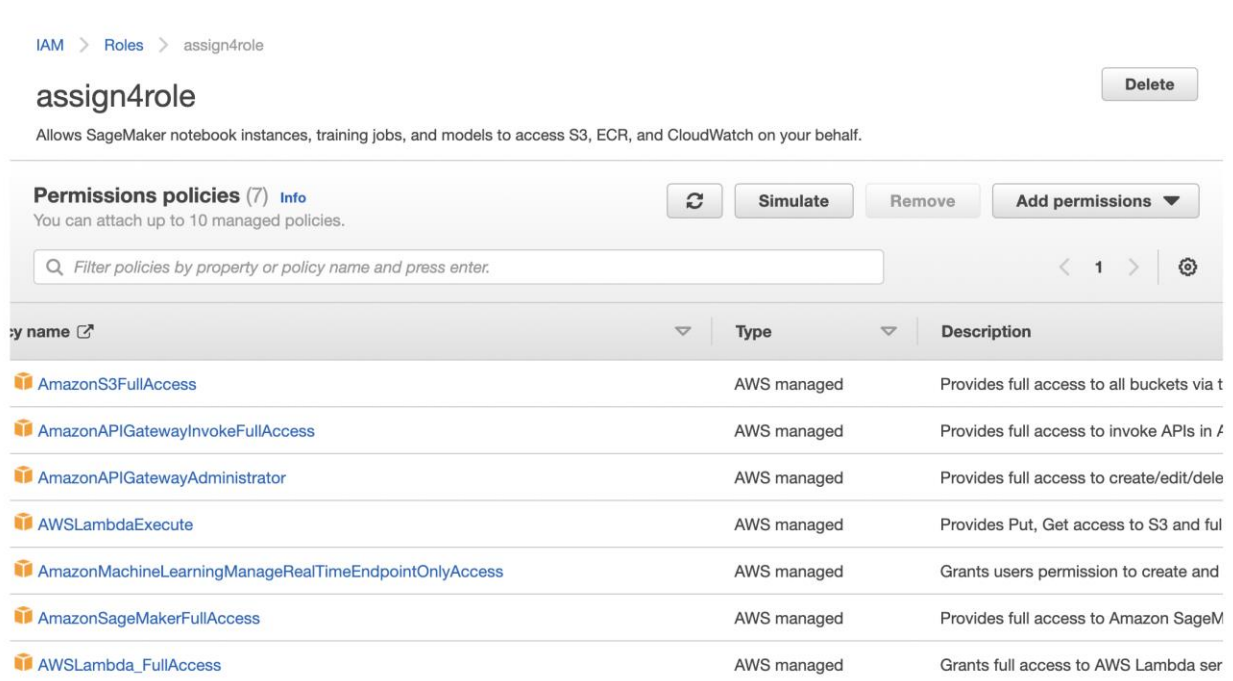
We use S3 to store our data that is required for training the model.

## Explanation of each step of the assignment

### Step -1 Create an IAM role

In this step we create an IAM role that gives us access to **SageMaker, S3, API Gateway and Lambda** and allows them to interact with each other.

For this assignment we created a role called **assign4role**.



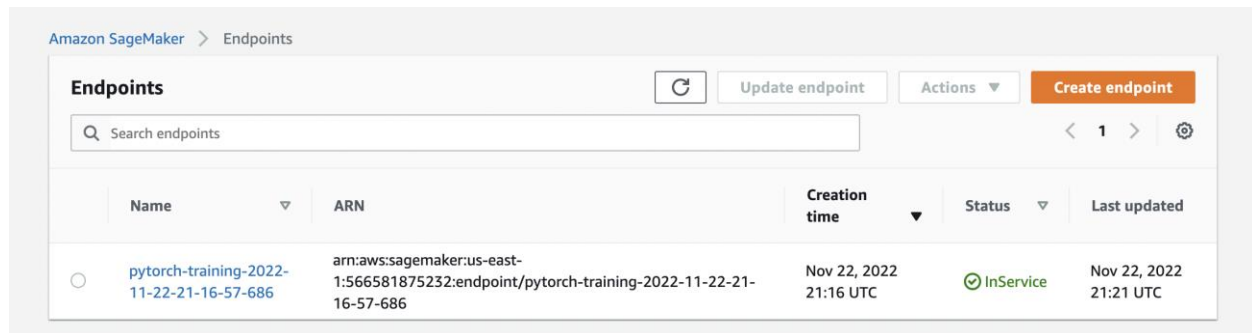
The screenshot shows the AWS IAM console for the role 'assign4role'. The role is described as 'Allows SageMaker notebook instances, training jobs, and models to access S3, ECR, and CloudWatch on your behalf.' It lists seven permissions policies, all of which are AWS managed. The policies are: AmazonS3FullAccess, AmazonAPIGatewayInvokeFullAccess, AmazonAPIGatewayAdministrator, AWSLambdaExecute, AmazonMachineLearningManageRealTimeEndpointOnlyAccess, AmazonSageMakerFullAccess, and AWSLambda\_FullAccess.

Policy name	Type	Description
AmazonS3FullAccess	AWS managed	Provides full access to all buckets via t
AmazonAPIGatewayInvokeFullAccess	AWS managed	Provides full access to invoke APIs in /
AmazonAPIGatewayAdministrator	AWS managed	Provides full access to create/edit/dele
AWSLambdaExecute	AWS managed	Provides Put, Get access to S3 and ful
AmazonMachineLearningManageRealTimeEndpointOnlyAccess	AWS managed	Grants users permission to create and
AmazonSageMakerFullAccess	AWS managed	Provides full access to Amazon SageM
AWSLambda_FullAccess	AWS managed	Grants full access to AWS Lambda ser

### Step -2 Deploy a ML code in SageMaker notebook instance

Here to do this job we have taken the reference from the github repository that performs a similar task ([https://github.com/aws/amazon-sagemaker-examples/blob/main/sagemaker-python-sdk/pytorch\\_mnist/pytorch\\_mnist.ipynb](https://github.com/aws/amazon-sagemaker-examples/blob/main/sagemaker-python-sdk/pytorch_mnist/pytorch_mnist.ipynb))

After running the above code, we can see that a SageMaker endpoint is created under **SageMaker -> Inference -> Endpoints**



The screenshot shows the Amazon SageMaker console's 'Endpoints' page. At the top, there are buttons for 'Update endpoint', 'Actions', and 'Create endpoint'. Below these is a search bar and a table of endpoints. The table has columns for Name, ARN, Creation time, Status, and Last updated. One endpoint is listed with the name 'pytorch-training-2022-11-22-21-16-57-686', ARN 'arn:aws:sagemaker:us-east-1:566581875232:endpoint/pytorch-training-2022-11-22-21-16-57-686', creation time 'Nov 22, 2022 21:16 UTC', status 'InService' (indicated by a green checkmark), and last updated time 'Nov 22, 2022 21:21 UTC'.

Name	ARN	Creation time	Status	Last updated
pytorch-training-2022-11-22-21-16-57-686	arn:aws:sagemaker:us-east-1:566581875232:endpoint/pytorch-training-2022-11-22-21-16-57-686	Nov 22, 2022 21:16 UTC	InService	Nov 22, 2022 21:21 UTC

## Explanation of the code

Here our code uses PyTorch to train our model. The dataset that we have here consists of grayscale images that are 28 x 28 in size and are handwritten.

- Firstly, we get the data from the online resource using torchvision and store it in an s3 location using `Sagemaker.Session.upload_data`.
- Then we use the same data from the s3 location to train our model. In order to train our model we have a training script `mnist.py` that runs and saves the model to `model_dir`.
- We run the training script with the PyTorch class and create a PyTorch object.
- Then we will fit the PyTorch object using the training data that will be made available locally by sagemaker to easily access and simply read the data from disk.
- After training the model we use PyTorch estimator to build the PyTorch model and deploy a PyTorchPredictor. This in turn creates a sagemaker endpoint that is a hosted prediction service that can be used for inference.

```
In [9]: predictor.endpoint
```

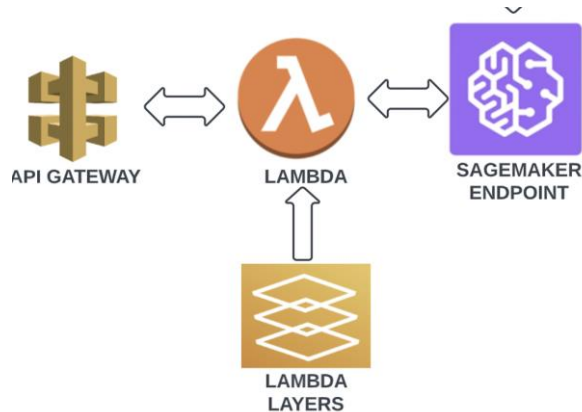
The endpoint attribute has been renamed in `sagemaker>=2`.  
See: <https://sagemaker.readthedocs.io/en/stable/v2.html> for details.

```
Out[9]: 'pytorch-training-2022-11-22-21-16-57-686'
```

Here we can see that an endpoint is created [pytorch-training-2022-11-22-21-16-57-686](#)

## Step -2 Creating a lambda function

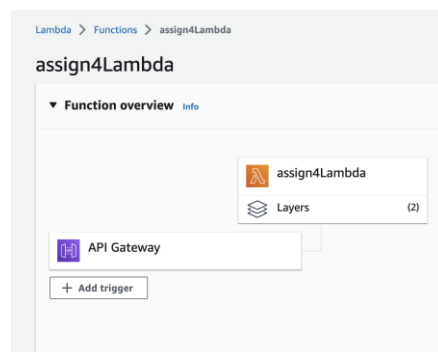
Here we create a lambda function(**assign4Lambda**) to call a sagemaker endpoint with the url that will be sent from the API Gateway. So, **Lambda acts as a bridge between API Gateway and SageMaker endpoint.**



Here we have to discuss about Layers. Layers in Lambda is used to add any packages that are required to process the lambda code. As our code needs python packages PIL and NUMPY. We added them to a zip file and added it to the layers in Lambda.

#### Functionality of Lambda code –

- Retrieves image url from the API gateway.
- Retrieves image object from image URL using PIL.
- Converts the image to required size.
- Converts the image to np array.
- Converts the image to grayscale.
- Expands the image over another dimension using axis.
- Changes the type to float32.
- Converts the array to JSON string and passes it to Sagemaker endpoint.
- The response of the sagemaker endpoint is raw data.
- So the data is filtered out using labeled\_predictions code in lambda.
- The most likely prediction is taken out and sent back as response.



#### Step -3 Setting up API gateway to trigger lambda

Here we create an **REST API** with the following details-

- **Method** = POST
- **Integration type** = Lambda Function
- **Lambda Function** = We select the Lambda function that we created.

Resources Actions ▾ / - POST - Setup

POST

Choose the integration point for your new method.

Integration type ☒ Lambda Function ⓘ  
☐ HTTP ⓘ  
☐ Mock ⓘ  
☐ AWS Service ⓘ  
☐ VPC Link ⓘ

Use Lambda Proxy integration ☐ ⓘ

Lambda Region us-east-1 ▾

Lambda Function assign4Lambda

Use Default Timeout ☒ ⓘ

Then we **deploy the API with stage as prod** and then we get the invoke URL –

APIs > Assign4 API (uiprenjrvk) > Stages > prod Show all hints ?

Stages Create prod Stage Editor Delete Stage Configure Tags

prod

Invoke URL: https://uiprenjrvk.execute-api.us-east-1.amazonaws.com/prod

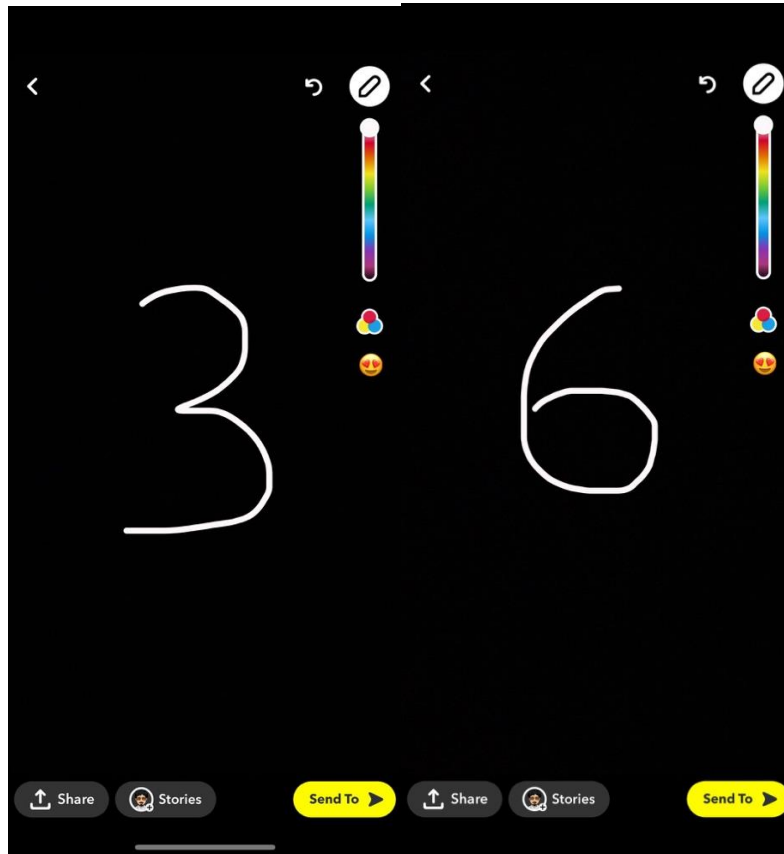
This URL will be used to trigger lambda which in turn calls the sagemaker endpoint and responds back the result of the predicted number.

This Invoke URL will be used by the user to send in a request in order to call the model. As this is a POST API the **parameters for the API would be the image URL** that needs to be predicted.

```
{  
  "url": "https://assign4handwrittenimage.s3.amazonaws.com/handwritten6.jpeg"  
}
```

### Generating Hand written images

All the handwritten images generated for testing the model were created using **Snapchat application** and were **uploaded to S3 bucket** and each of the **image object** was given **public access** to test them.



Amazon S3 > Buckets > assign4handwrittenimage

assign4handwrittenimage [Info](#)

[Objects](#) | [Properties](#) | [Permissions](#) | [Metrics](#) | [Management](#) | [Access Points](#)

**Objects (4)**

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

[Copy S3 URI](#) [Copy URL](#) [Download](#) [Open](#) [Delete](#) [Actions](#) [Create folder](#)

[Upload](#)

Find objects by prefix

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	<a href="#">handwritten1.jpeg</a>	jpeg	November 22, 2022, 20:57:19 (UTC-06:00)	19.0 KB	Standard
<input type="checkbox"/>	<a href="#">handwritten2.jpeg</a>	jpeg	November 22, 2022, 20:21:37 (UTC-06:00)	10.0 KB	Standard
<input type="checkbox"/>	<a href="#">handwritten3.jpeg</a>	jpeg	November 22, 2022, 20:51:15 (UTC-06:00)	18.2 KB	Standard
<input type="checkbox"/>	<a href="#">handwritten6.jpeg</a>	jpeg	November 22, 2022, 21:00:44 (UTC-06:00)	25.3 KB	Standard

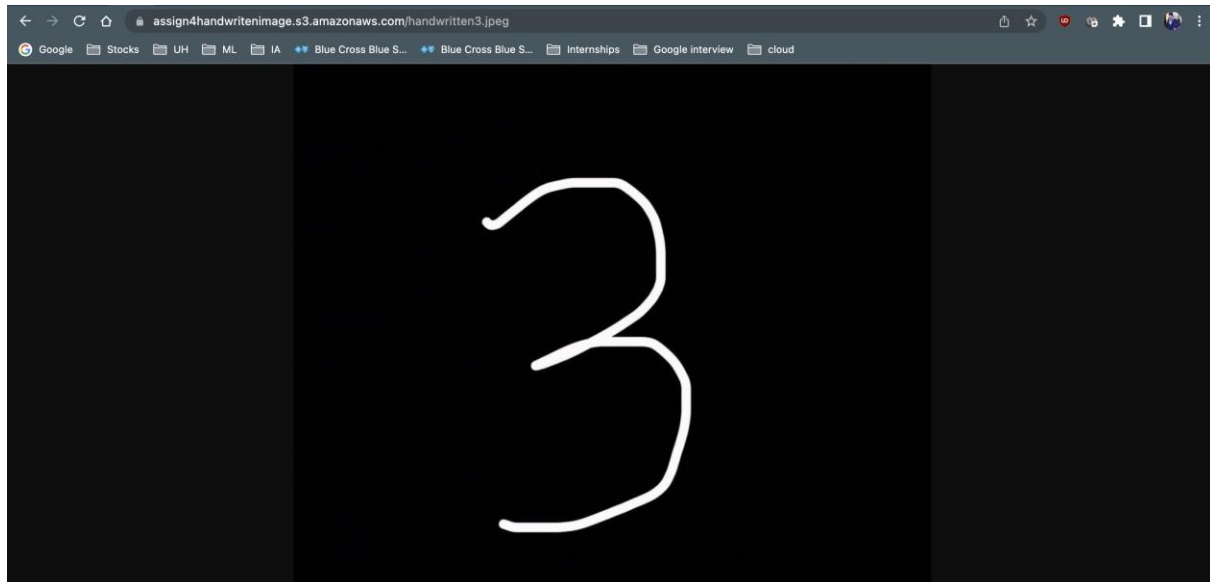
<https://assign4handwrittenimage.s3.amazonaws.com/handwritten3.jpeg>

<https://assign4handwrittenimage.s3.amazonaws.com/handwritten6.jpeg>

**Validation of the model using my own handwritten digits**

## Scenario - 1

A handwritten image 3 hosted in a S3 bucket was passed as an input to the API.

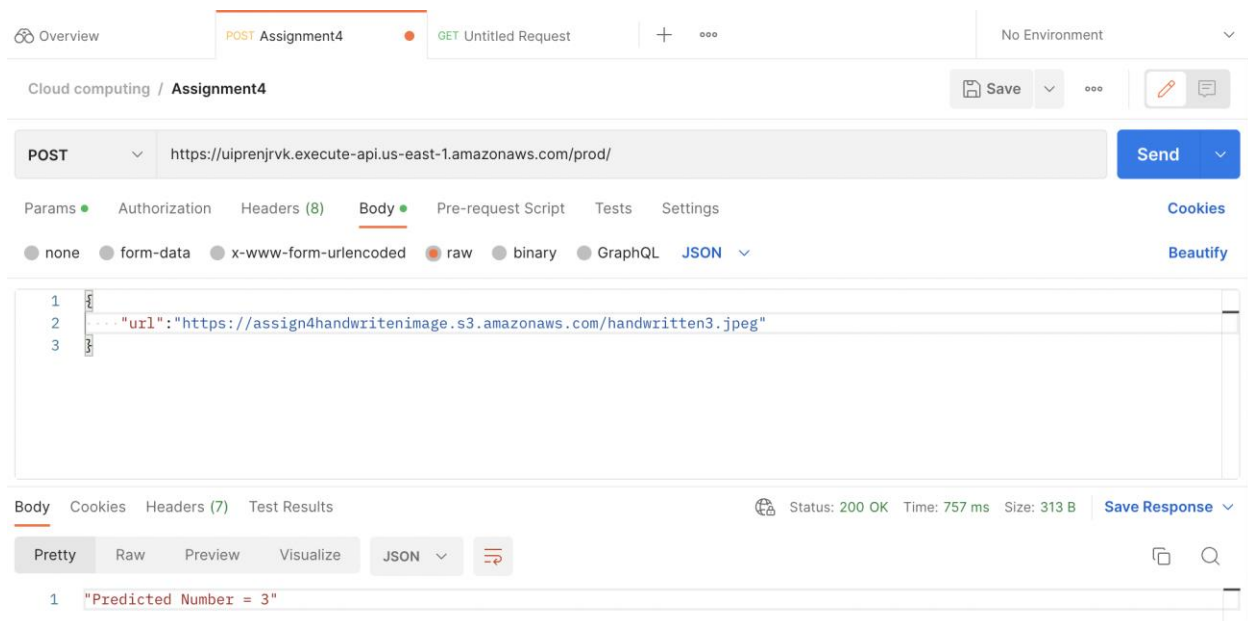


## CURL Output -

```
jagadisharepalli@Jagadishs-MacBook-Pro ~ % curl -i -X POST -H 'Content-Type: application/json' -d '{"url": "https://assign4handwrittenimage.s3.amazonaws.com/handwritten3.jpeg"}' https://uiprenjrvk.execute-api.us-east-1.amazonaws.com/prod/
HTTP/2 200
date: Wed, 23 Nov 2022 03:25:24 GMT
content-type: application/json
content-length: 22
x-amzn-requestid: 8353c6cb-9b87-4083-89c3-8fe71e325b55
x-amz-apigw-id: cCP21FGmIAMFThAs
x-amzn-trace-id: Root=1-637d92a3-43920a9679e6f47b5a24d330;Sampled=0

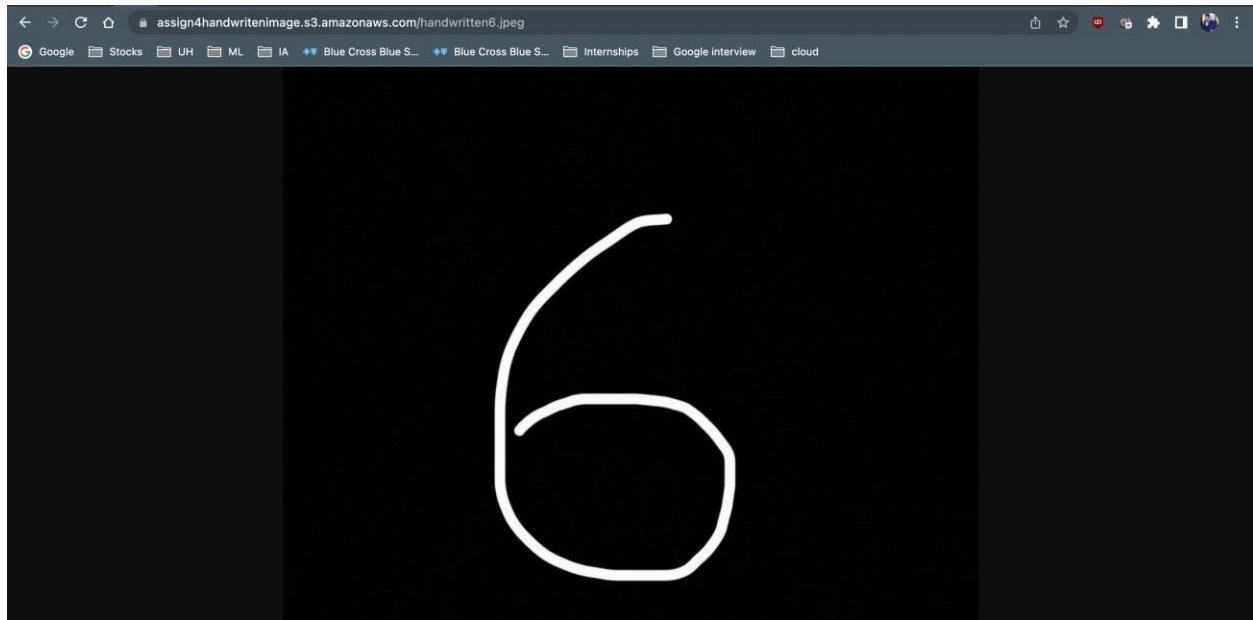
{"Predicted Number = 3"}
```

## Output from Postman



## Scenario-2

A **handwritten image 3** hosted in a S3 bucket was passed as an input to the API.



### CURL output-

```
jagadisharepalli@Jagadishs-MacBook-Pro ~ % curl -i -X POST -H 'Content-Type: application/json' -d '{"url":"https://assign4handwrittenimage.s3.amazonaws.com/handwritten6.jpeg"}' https://uiprenjrvk.execute-api.us-east-1.amazonaws.com/prod/
HTTP/2 200
date: Wed, 23 Nov 2022 03:21:57 GMT
content-type: application/json
content-length: 22
x-amzn-requestid: 5b5b3c3d-440b-423f-bdb7-25798834ad35
x-amz-apigw-id: c0043Er0IAMFU5A-
x-amzn-trace-id: Root=1-637d91d2-6c2097205e2ec848784e6b23;Sampled=0

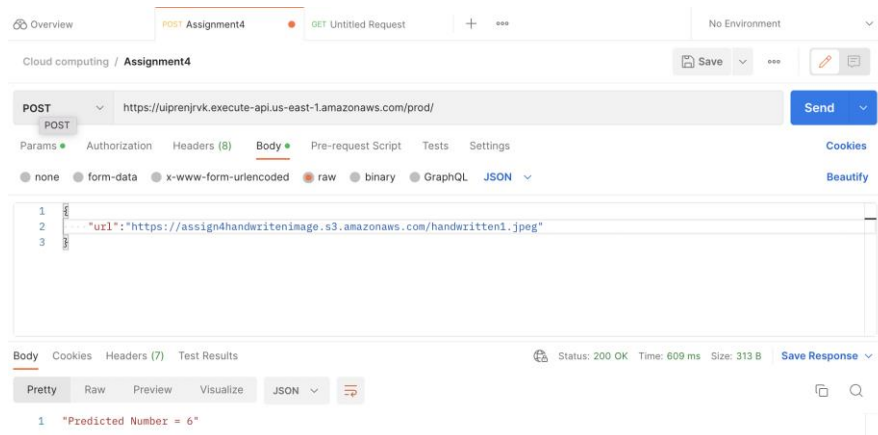
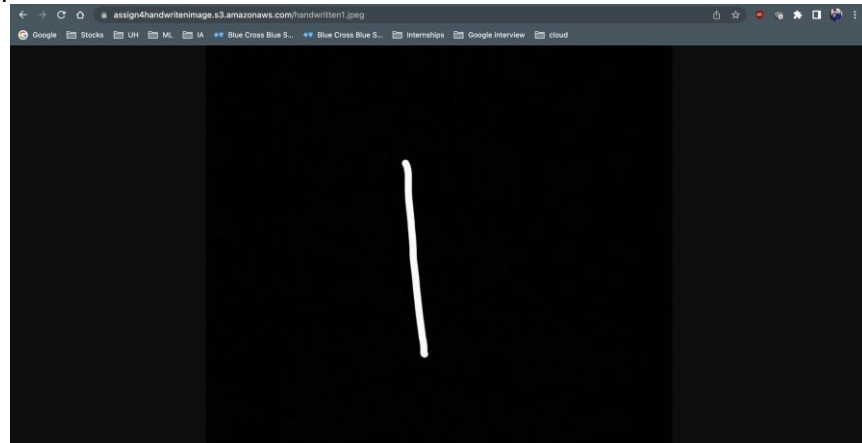
{"Predicted Number = 6"}
jagadisharepalli@Jagadishs-MacBook-Pro ~ %
```

### Output from Postman

### Limitations



- 1) As we know that ML models are not always 100% accurate, the below test resulted in a wrong prediction.



- 2) Due to complexity of working with the databases I wasn't able to figure how to host the dataset in a database and access it in Sagemaker. All the testing for database connection was done in AWS academy and I lost the code when the session expired. SO couldn't provide you with screenshots of the work that I did with the databases.
- 3) Due to access issue with AWS academy I used my own AWS account to continue with my assignment which allowed me to create my own role with all the required accesses.

## **References**

### **MNIST Model Code**

- [https://github.com/aws/amazon-sagemaker-examples/blob/main/sagemaker-python-sdk/pytorch\\_mnist/pytorch\\_mnist.ipynb](https://github.com/aws/amazon-sagemaker-examples/blob/main/sagemaker-python-sdk/pytorch_mnist/pytorch_mnist.ipynb)
- [https://github.com/aws/amazon-sagemaker-examples/blob/main/sagemaker-python-sdk/pytorch\\_mnist/mnist.py](https://github.com/aws/amazon-sagemaker-examples/blob/main/sagemaker-python-sdk/pytorch_mnist/mnist.py)
- <https://github.com/aws/amazon-sagemaker-examples>

### **Lambda**

Layers configuration

- <https://medium.com/@shimo164/lambda-layer-to-use-numpy-and-pandas-in-aws-lambda-function-8a0e040faa18>
- <https://www.youtube.com/watch?v=lrEAu75zhNI>

Lambda code

- <https://predictivehacks.com/how-to-build-a-rest-api-calling-the-sagemaker-endpoint/>
- <https://www.youtube.com/watch?v=xe9-GZ1tX28>

### **API Gateway**

- <https://www.youtube.com/watch?v=uFsaiEhr1zs>