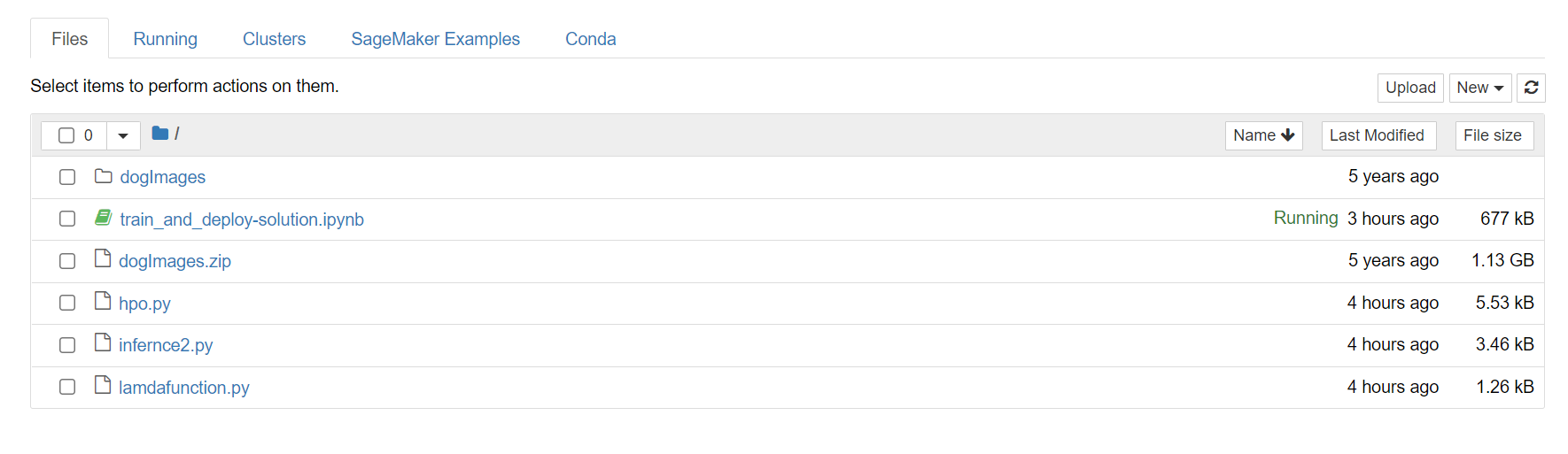
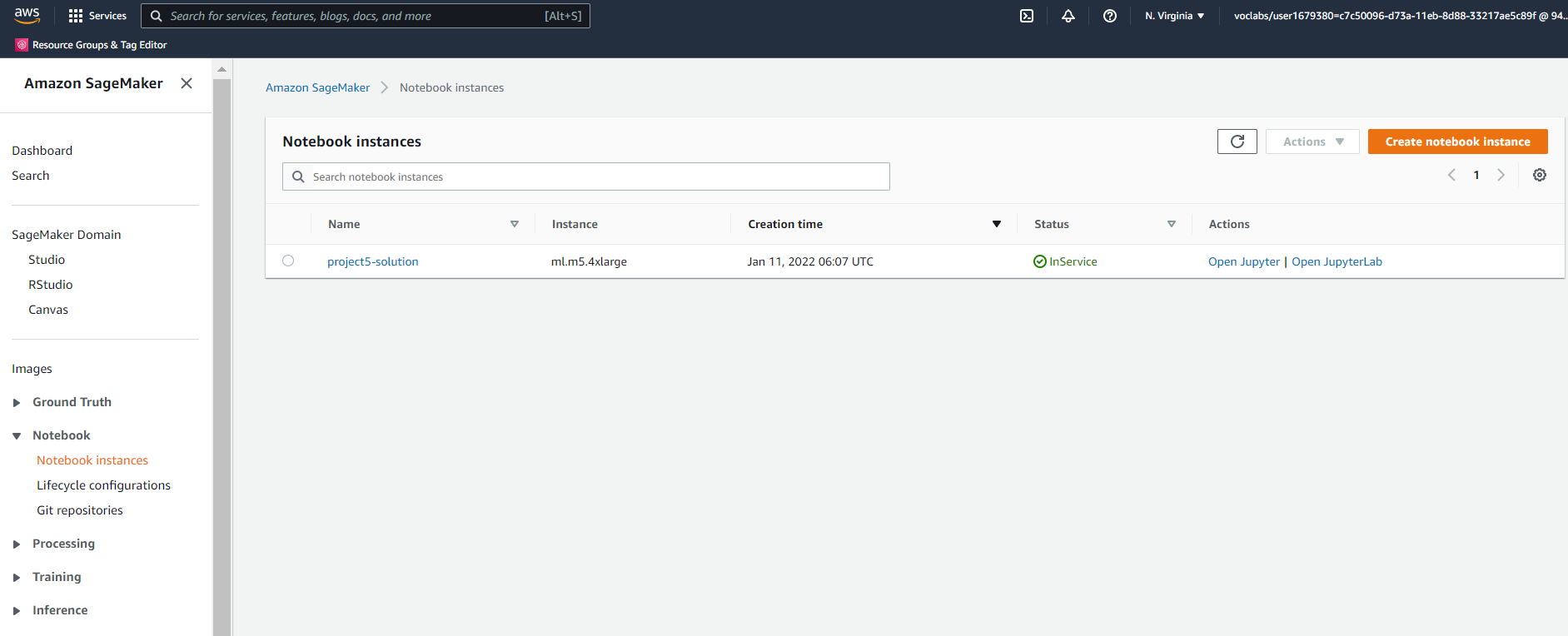
# Step 1: Training and Deployment

## Initial Setup

## Create and open a Sagemaker instance.



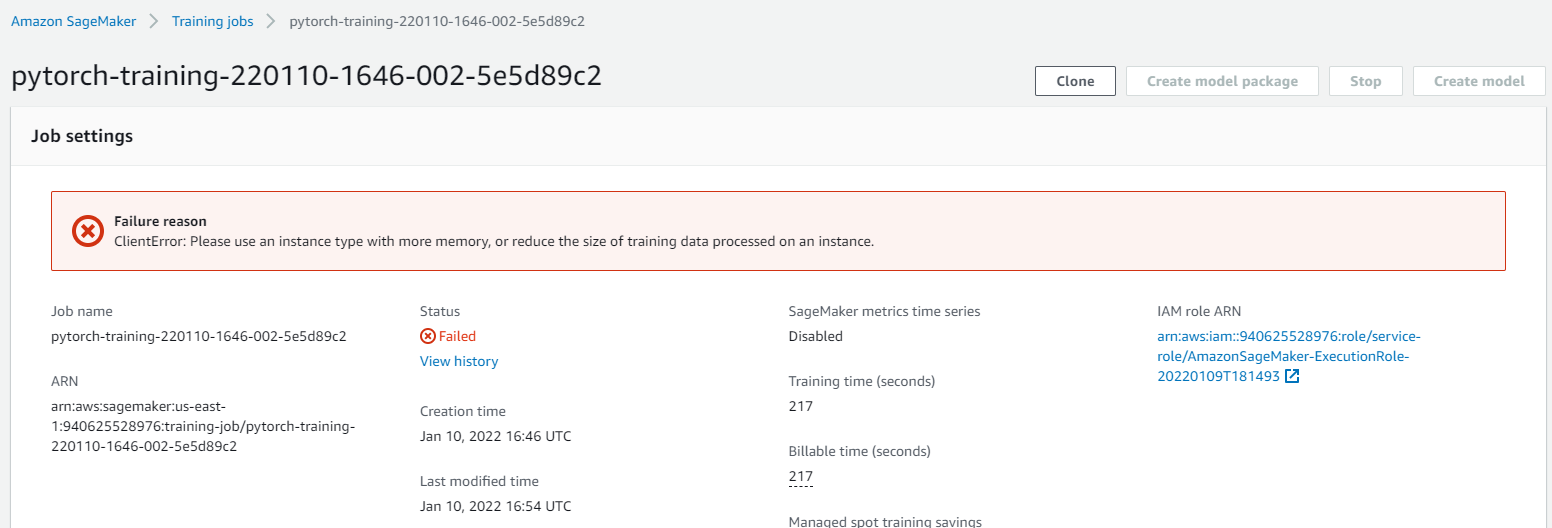
## Take a screenshot of your Sagemaker instance setup



## Write about the Sagemaker instance you created, including a justification of why you chose the instance type you did.

Instance Type: ml.m5.4xlarge

* I started off with default instance type and the execution happened for a long time and didn’t complete (server went down)
* I then updated the instance type to ml.m5.large which in turn failed during deployment due to volume constraint

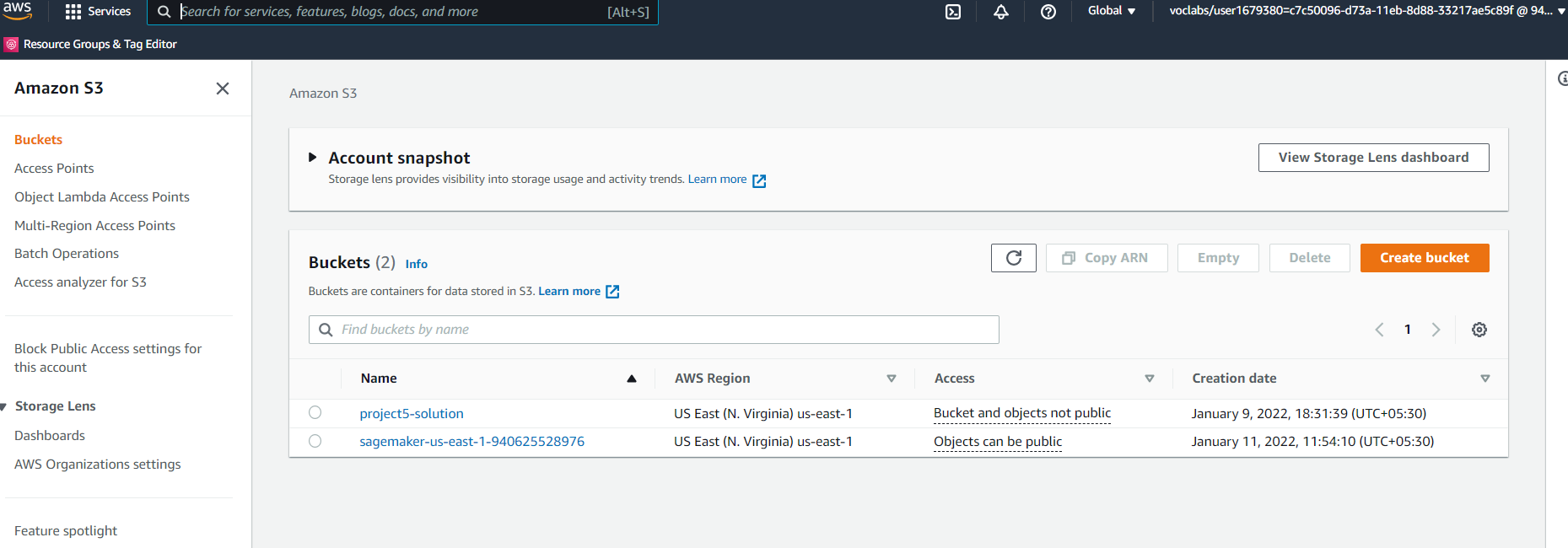


Hence chose the instance type with increased volume (ml.m5.4xlarge). The execution was faster and I was able to get the endpoint. Although, this is not a cost effective solution, the productivity increases with this selection.

## Download data to an S3 bucket

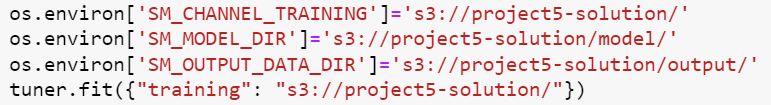
## Create an S3 bucket in your AWS workspace

Bucket name: project5-solution



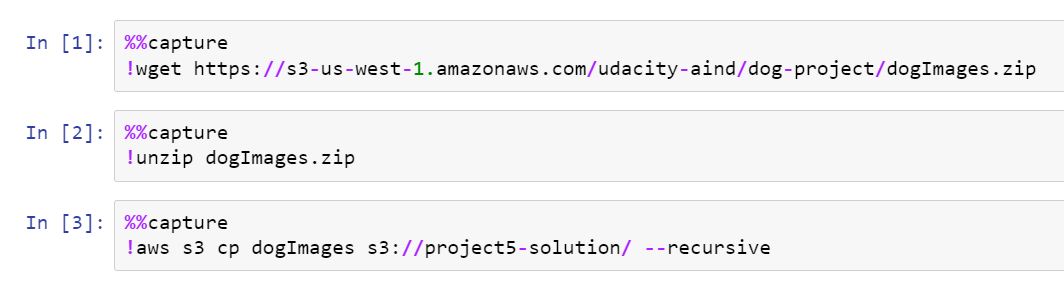
## Add the address of your S3 bucket to all cells of your train\_and\_deploy-solution.ipynb notebook that contain references to "s3://udacitysolution/".







## Run the first three cells of the train\_and\_deploy-solution.ipynb notebook.



## 

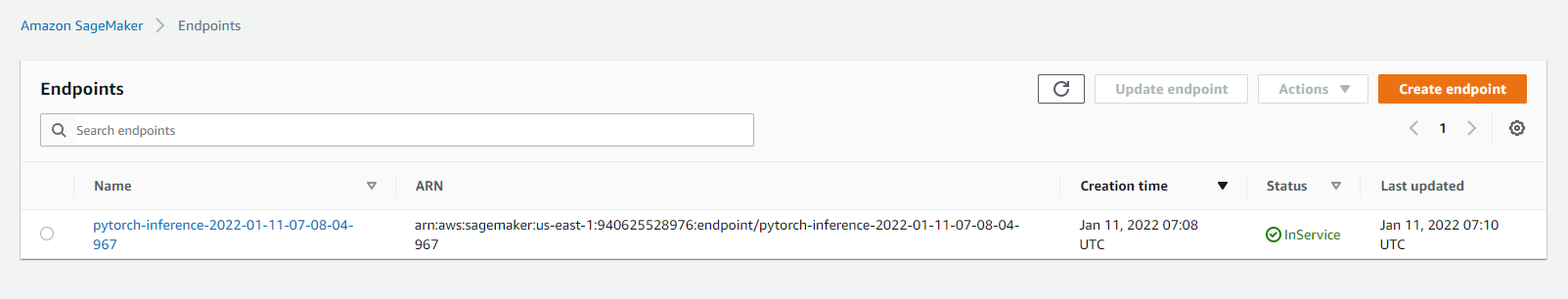
## Training and Deployment

## Run all remaining cells in the train\_and\_deploy-solution.ipynb notebook.

Github url:

## Make a note of the name of the endpoint that your notebook deploys.

Endpoint name: [pytorch-inference-2022-01-11-07-08-04-967](https://console.aws.amazon.com/sagemaker/home?region=us-east-1#/endpoints/pytorch-inference-2022-01-11-07-08-04-967)

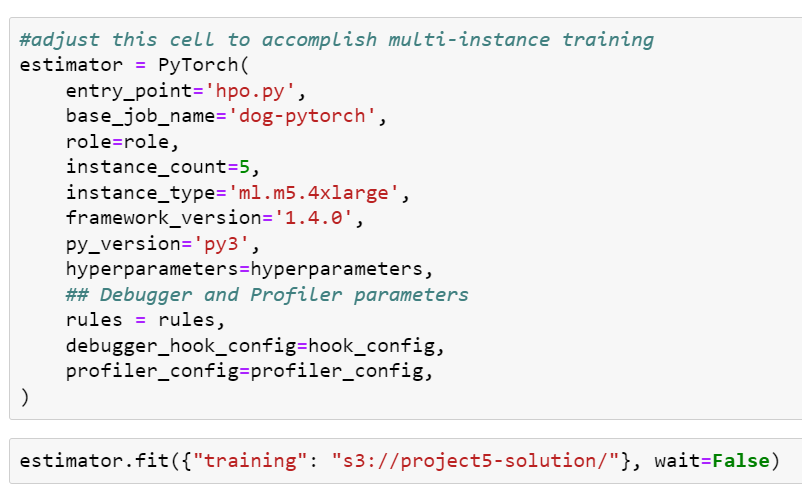


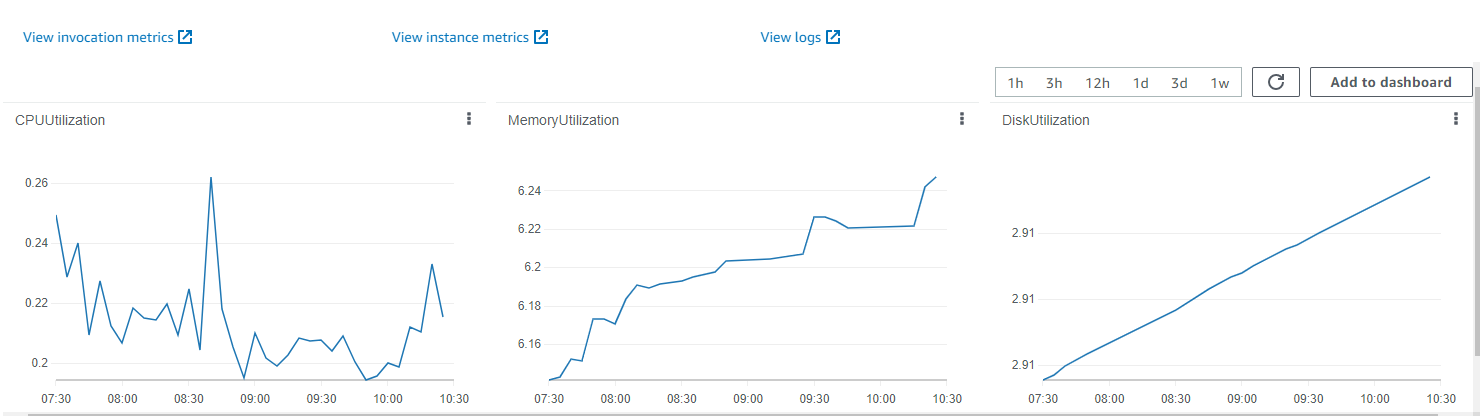
## Multi-instance training

## Alter your train\_and\_deploy\_solution.ipynb notebook so it performs multi-instance training

Changed instance\_count: 4 and instance\_type: ml.m5.4xlarge and executed again

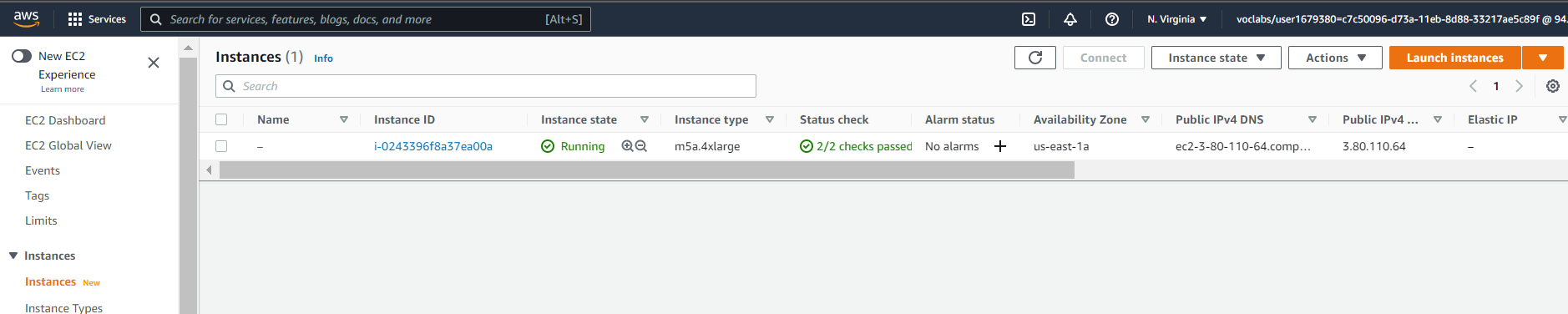
## Run your train\_and\_deploy\_solution.ipynb notebook again so that it deploys a new endpoint that was trained on multiple instances





# Step 2: EC2 Training

## Create and open an EC2 instance



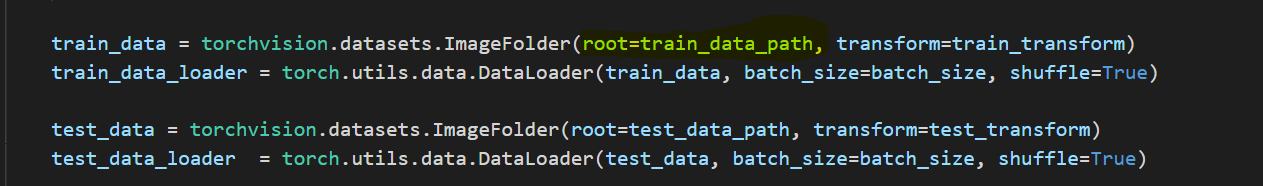
## Write a justification of why you chose your type of EC2 instance

Created an EC2 instance with Deep Learning AMI setting with a basic instance type. The execution happened for a long time (took almost whole day) and did not complete. Hence created an instance with instance type – m5a.4xlarge

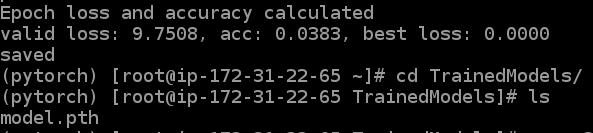
## Prepare and run code to accomplish model training on your EC2 instance

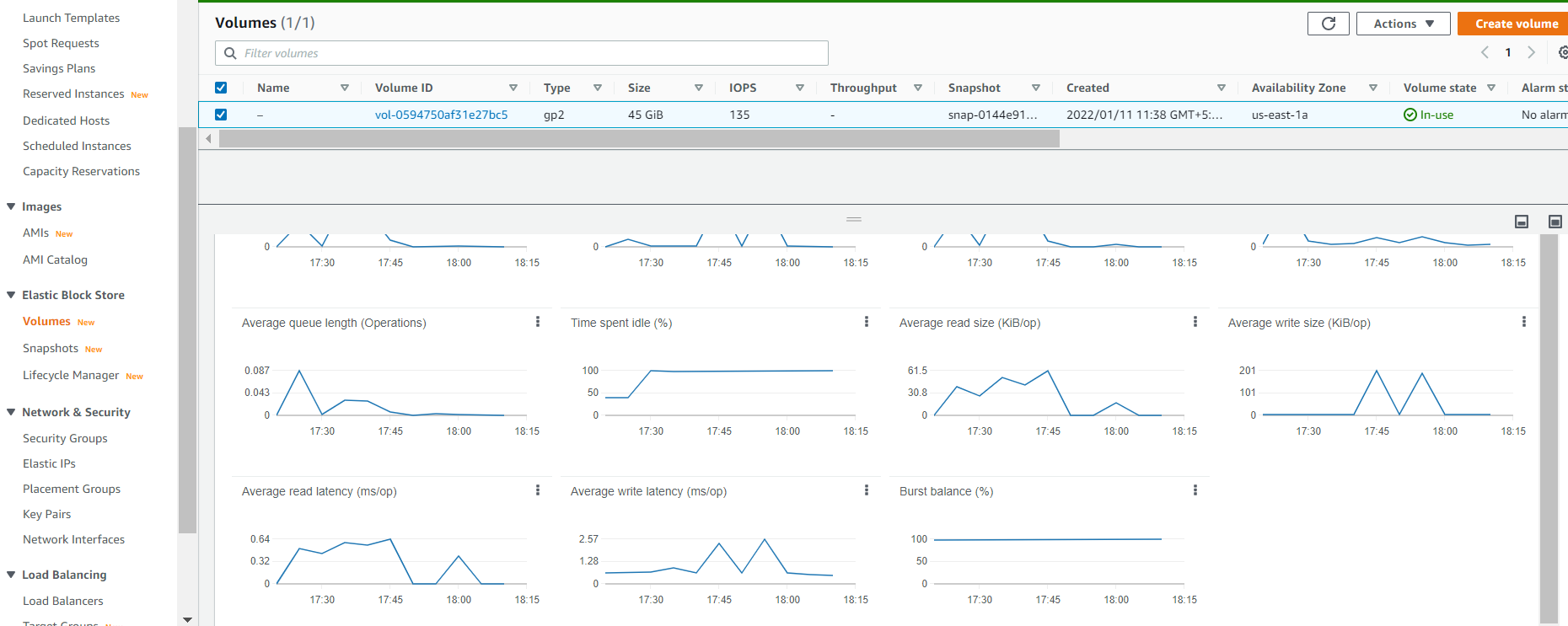
Updated ec2train1.py code with loggers and created solution.py in EC2 instance.

PS: The original ec2train1.py code needs a minor change as in below screenshot. (The root parameter for train\_data is train\_data\_path. But it was given as test\_data\_path). I had corrected this and executed the script.



## Save a trained model to your EC2 instance





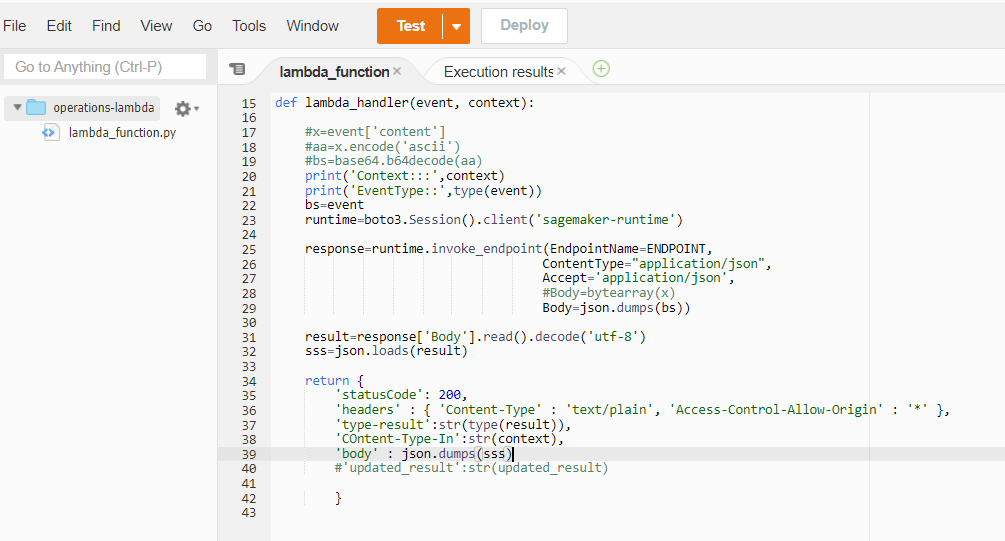
## Write about your EC2 code and how it compares to the code from Step 1

Below comparison is made with this project experience. EC2 may have better features than sagemaker for any other computations / purposes.

|  |  |
| --- | --- |
| EC2 | Sagemaker |
| Not so expensive | Expensive |
| Everything is built from scratch | Built in algorithms, frameworks available |
| Needs external app integration for deployment | Deployment can be done |
| Monitoring script execution progress is not user friendly | Visibility is more on monitoring the executions |

# Step 3: Setting up a Lambda function

## Set up a Lambda function in your AWS workspace, using the lambdafunction.py starter file as its Python code.



## Add the name of your deployed endpoint to the definition of the endpoint\_name variable in your Lambda function code



## Add your lambdafunction.py script to your final submission

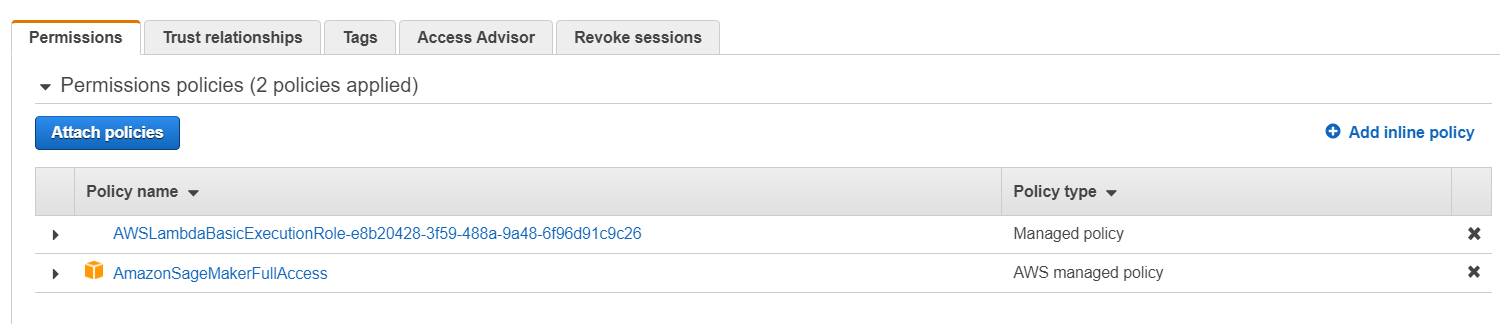
Git url:

## Write about the Lambda function in your final writeup

This serverless computing service is always handy for quicker validation of endpoint APIs.

# Step 4: Lambda Security and Testing

## Attach a security policy to your Lambda function so it can access your Sagemaker endpoint



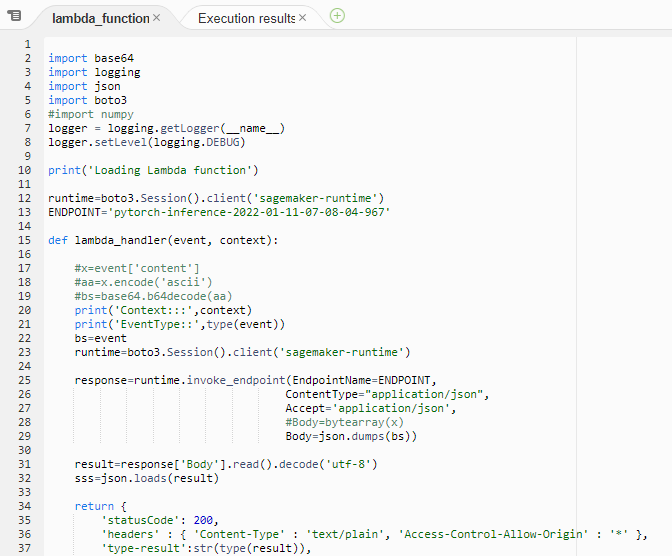
## Test your Lambda function, and add the result of your test to your writeup

Test event:

{ "url": "https://s3.amazonaws.com/cdn-origin-etr.akc.org/wp-content/uploads/2017/11/20113314/Carolina-Dog-standing-outdoors.jpg" }



## Take a screenshot of your Lambda setup and add it to your solution archive

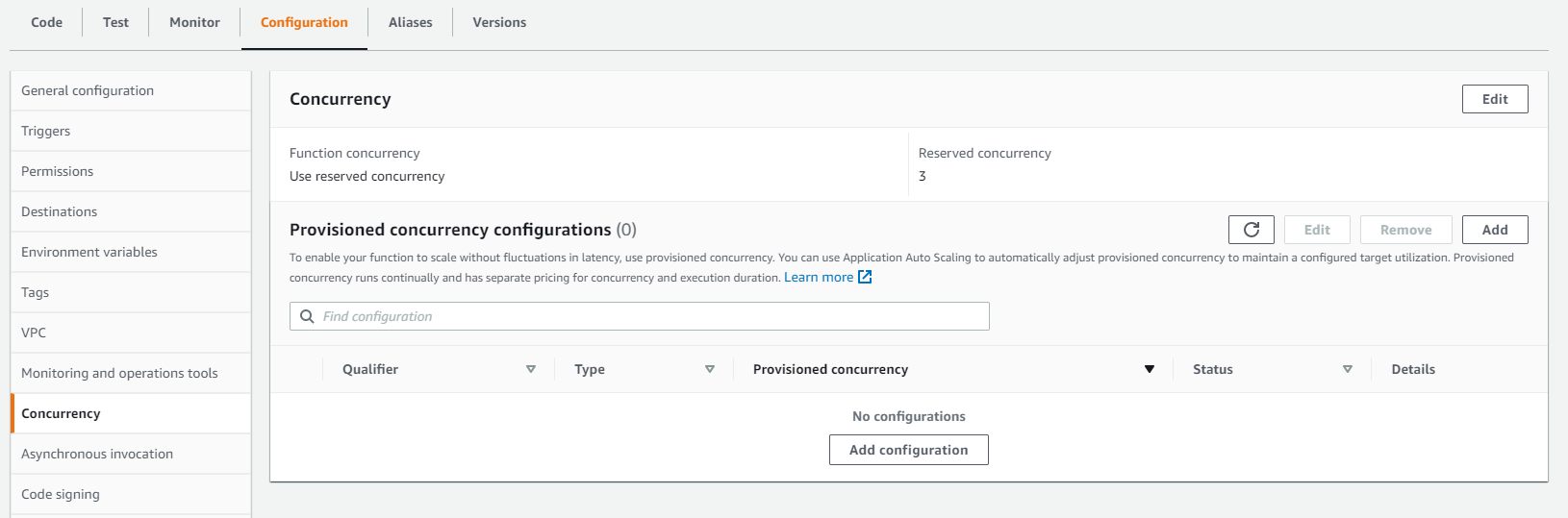


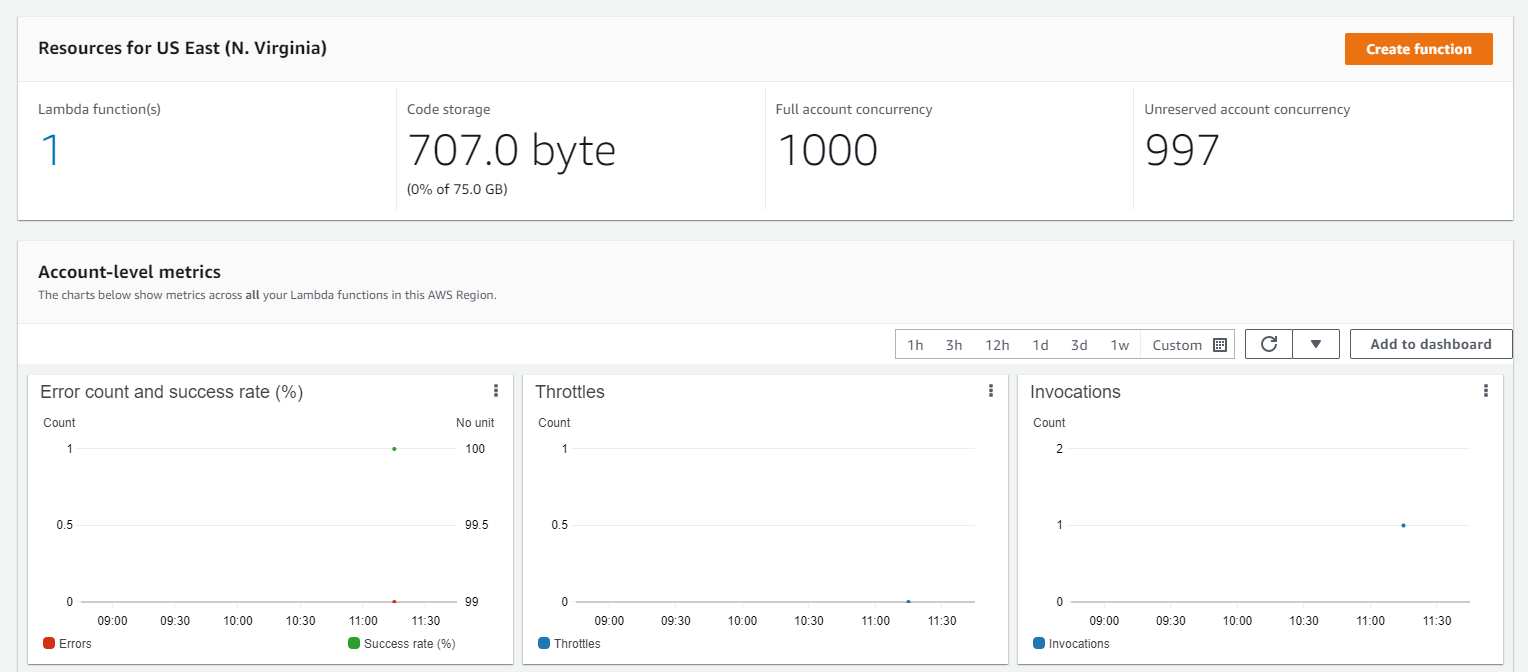
### Write about whether you think your AWS workspace is secure, and whether you think there are any vulnerabilities

* The lambda function is attached with “AmazonSagemakerFullAccess” and hence we can ensure it can access those functions in Sagemaker alone and not any other applications (like EC2, S3 etc).
* The access to other applications can then be attached only on demand.
* This ensures the security and privacy of the execution and so there is no identified vulnerabilities except for the **endpoint being hardcoded**

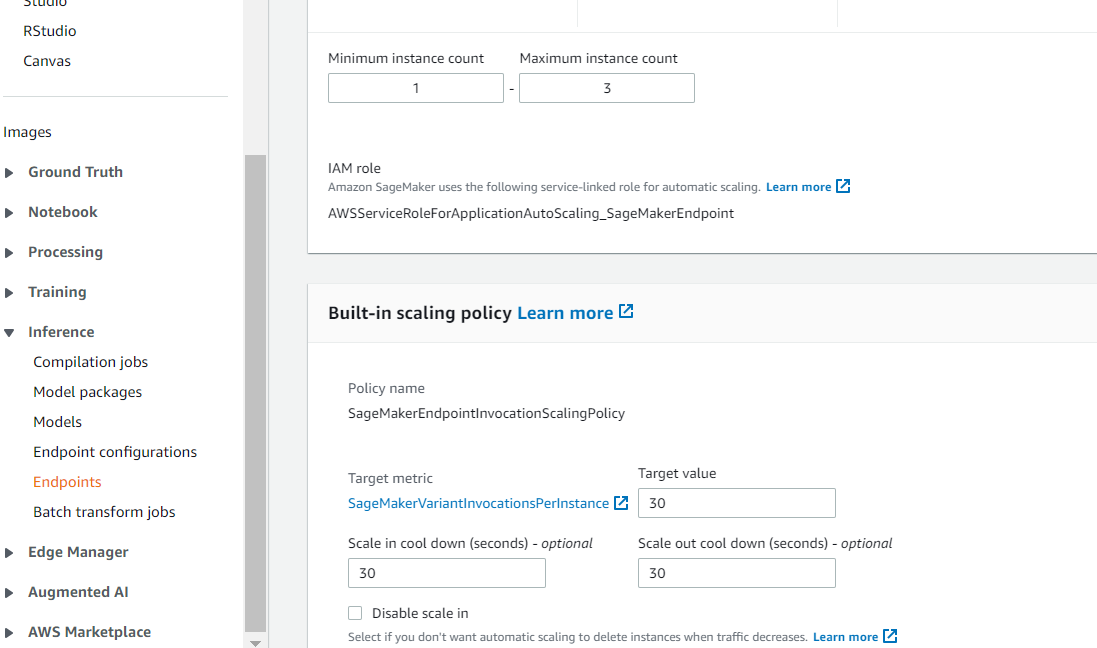
# Step 5: Concurrency and auto-scaling

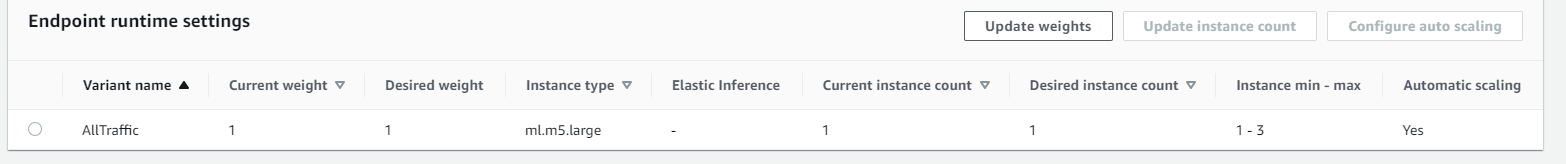
## Set up concurrency for your Lambda function





## Set up auto-scaling for your deployed endpoint





## Write about your configuration of concurrency and auto-scaling. Make sure to mention what kind of concurrency you set up, how you set up auto-scaling, and why you made your decisions

**Concurrency**: Added concurrency to decrease the latency in high traffic situations

Selected Concurrency: Reserved (Max 3 requests) as this is for a study project, the expected requests may not go more than 3 requests and so it is cost effective.

Justification: Provisioned Concurrency incurs higher cost and is charged every month. Hence selected Reserved Concurrency.

**Auto-scaling:** Configured for the endpoints to respond multiple requests at the same time.

Justification: To be reasonable for the project nature and to be cost-effective, selected the specified configuration