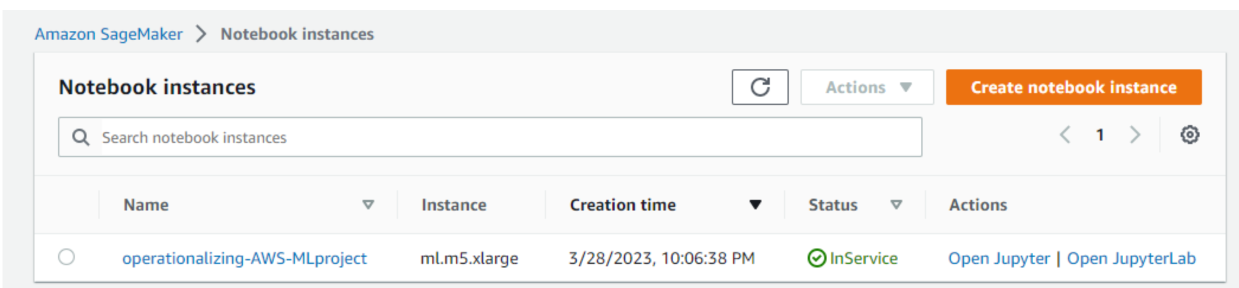
Write-up

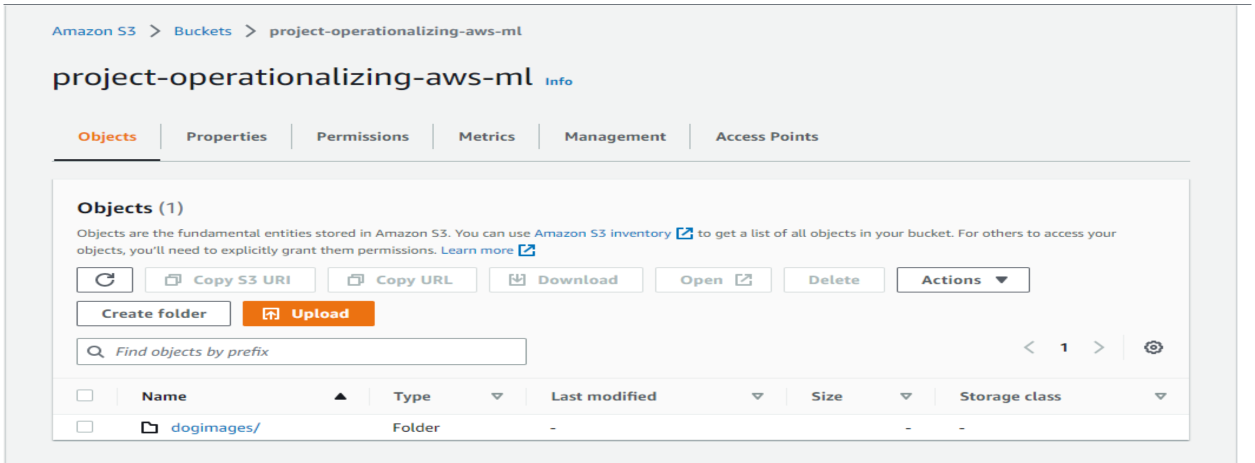
Step 1: Training and deployment on Sagemaker

ml.m5.xlarge is the instance type used to train and deploy the model, choosing this type was after using lower instance type which leads to memory size issue and the recommendations were to increase instance type or memory size so this type was chosen to solve the lower instance type issues.

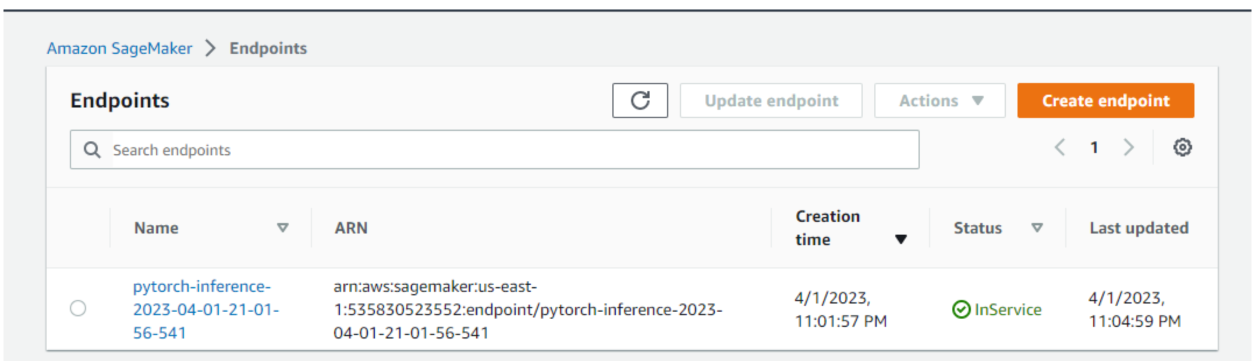
.Instance screenshot:



.Take a screenshot showing that you've set up an S3 bucket.

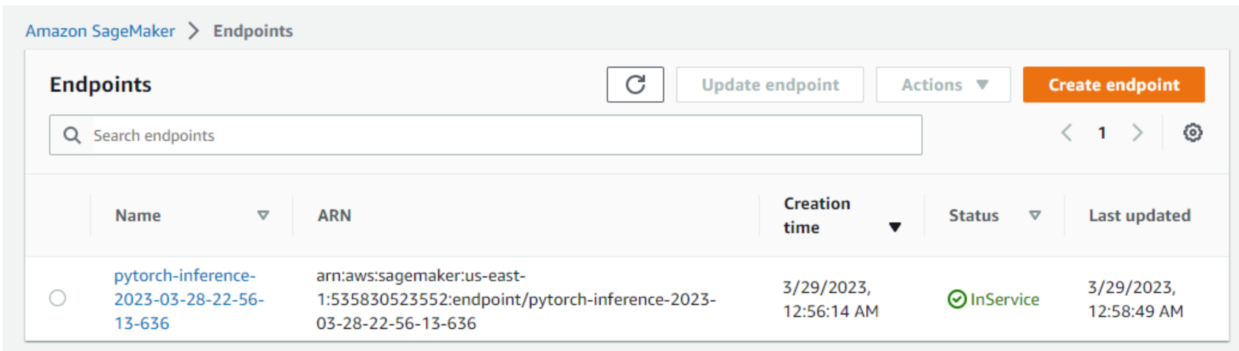


.Endpoint screenshot:



Multi-instance training:

.It's deployed endpoint:



Step 2: EC2 Training

A t2.large instance is used for EC2 instance. I tried to use another instances with lower cost and computing power but they didn't work since numpy and torch packages needed to be installed, higher instance memory is needed so packages are fully installed, to be able to train the model. The used instance price is (0.0928 USD per hour) and computing power is (2vCPU, 8GIB memory), this tradeoff between the cost and computing power and how well it train the model makes it a good choice.

.Screenshot of saved model:

```
-rw-------  1 root root        860 Apr  4 15:38 .viminfo
[root@ip-172-31-95-171 ~]# ls -la /root/TrainedModels/
total 93212
drwxr-xr-x 2 root root       23 Apr  4 15:43 .
dr-xr-x--- 9 root root      254 Apr  4 15:38 ..
-rw-r--r-- 1 root root 95445365 Apr  4 15:43 model.pth
[root@ip-172-31-95-171 ~]#
```

.EC2 code vs in train_and_deploy-solution.ipynb code:

EC2 code:

1- Manually activate particular environment used to run python code:

   Soruce activate pytorch_latest_p37 which contains useful ML modules.

2- It needed to install necessary packages to run the code:
   -pip install numpy –user
   -pip install torchvision –user

3- No prebuilt module to train the model it just the command Python solution.py to run the custom model.
4- The directory to save the output is manually created.

Train_and_deploy-solution.ipynb code:

Sagemaker provides pre-built modules and APIs that set up and configure the environment, install dependencies and output is directly saved on S3.

Step 3: Lambda function setup

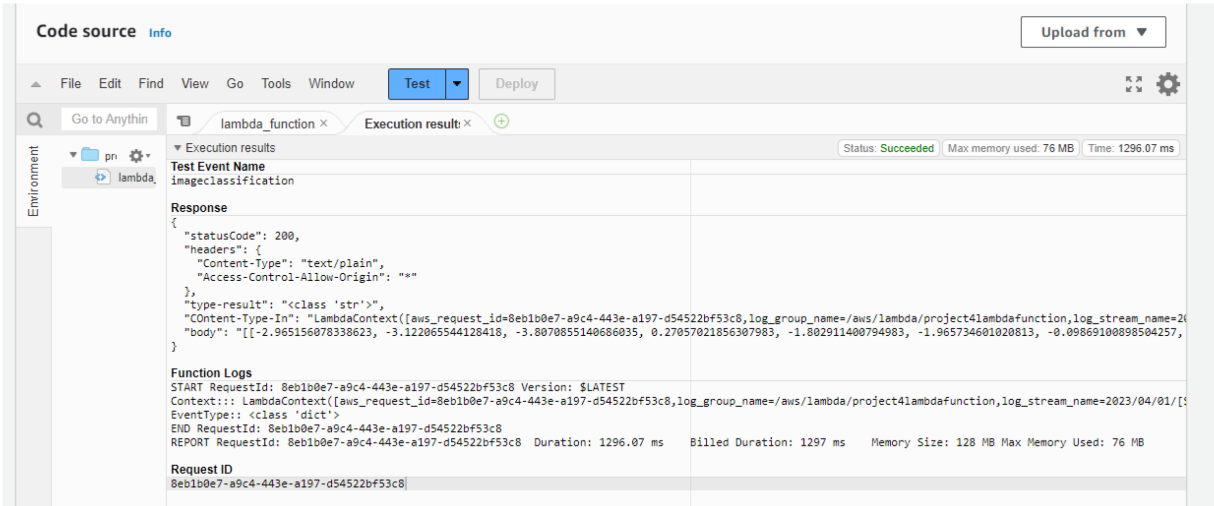
Lambda function act as the intermediary between users and ML models it take the input from the users and pass it to the endpoint and take output and pass it to the users. The function here:

1. Import the necessary modules

2. Lambda_handler function invoke the deployed endpoint 'pytorch-inference-2023-04-01-21-01-56-541' and pass the following event `{ "url": "https://s3.amazonaws.com/cdn-origin-etr.akc.org/wp-content/uploads/2017/11/20113314/Carolina-Dog-standing-outdoors.jpg" }` which act as the input from the users. The invoked endpoint is tested by the event to get a response.

3. If lambda function returns a status code of 200 and give a prediction list of 133 numbers this is a successful run. If function fails to run an error message will be shown to identify the error.

Lambda function screenshot:



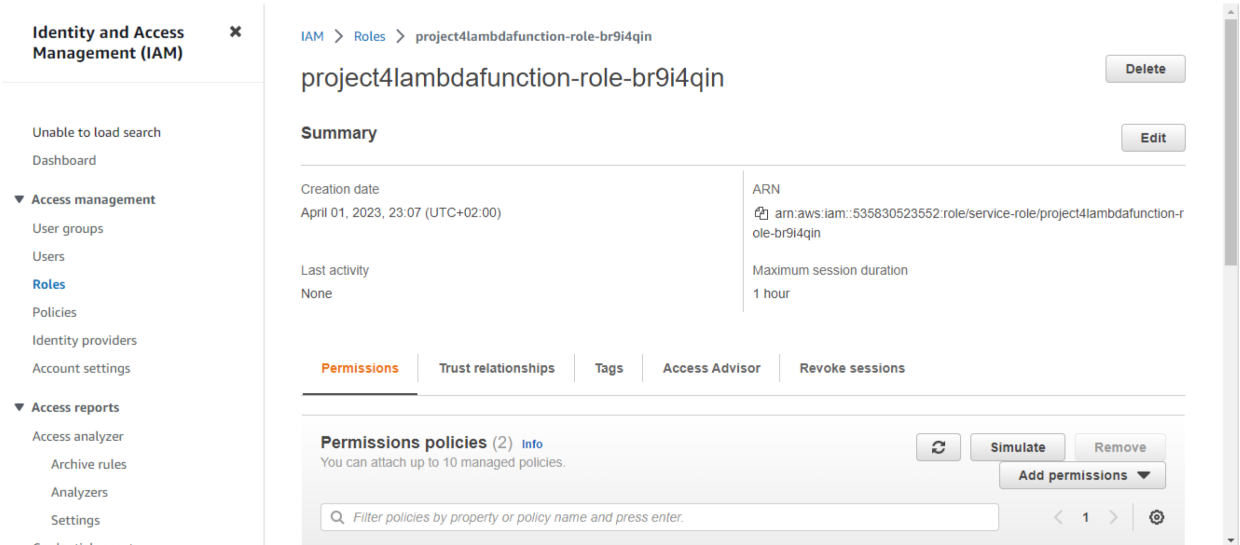
<u>Lambda function returns a list of 133 numbers which represent a prediction about the image (test event) class</u>:

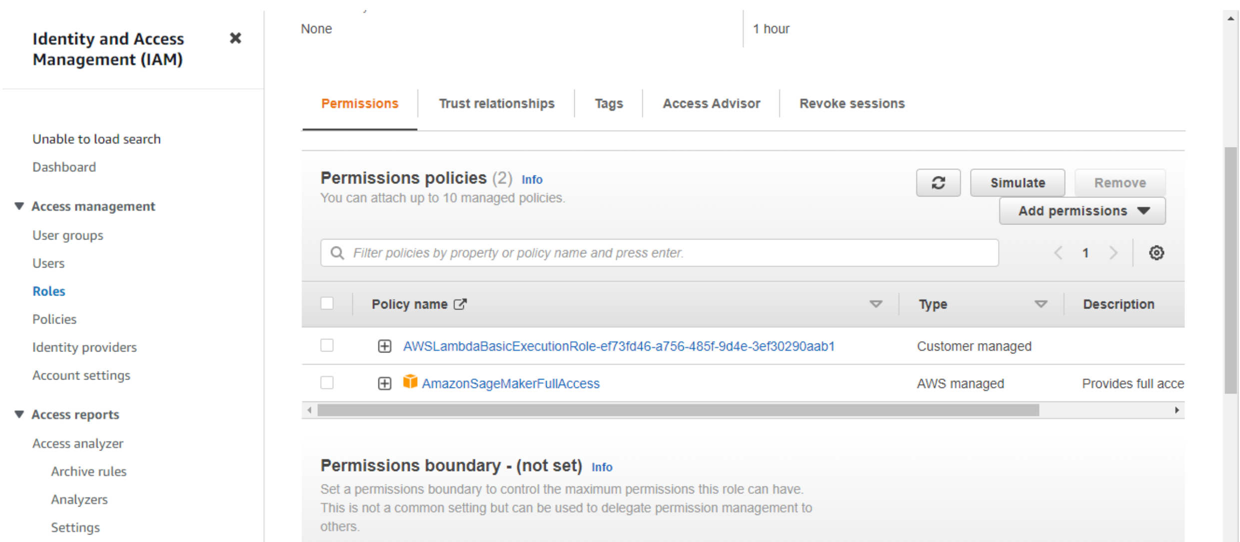
"body": "[[-2.965156078338623, -3.122065544128418, -
3.8070855140686035, 0.27057021856307983, -1.802911400794983, -
1.965734601020813, -0.09869100898504257, 0.7091438174247742, -
4.103690147399902, 0.2727470397949219, -0.07881911098957062, -
2.2885966300964355, -4.393979072570801, 0.6918509602546692, -
1.810187816619873, -1.3562313318252563, -3.525505542755127, -
0.18143923580646515, -3.189654588699341, 0.24673117697238922, -
2.3222339153289795, 0.2515391409397125, -0.9050092697143555, -
4.394808292388916, -2.101229667663574, -2.1128973960876465, -
2.0499517917633057, -2.2694077491760254, -1.796690583229065, -
0.7488433122634888, -2.4382503032684326, 1.6884427070617676, -
3.2836363315582275, 0.45845329761505127, -3.2213449478149414, -
4.033785820007324, -0.879679262638092, -3.3337807655334473,
0.3462081849575043, 0.21124325692653656, -0.11325793713331223, -
3.4006128311157227, 0.8115973472595215, 1.050553321838379,
0.35316774249076843, -2.8111398220062256, -0.1125725582242012,
0.7920854091644287, -1.467564344406128, 0.5609480738639832, -
4.045478343963623, -3.1722970008850098, -4.690579414367676, -

0.7290812730789185, -0.7654348611831665, -0.7097199559211731, -
0.09369193762540817, -4.568695068359375, 0.9161924719810486,
0.6928541660308838, -3.248945474624634, -1.8893002271652222, -
1.2768036127090454, -2.252840042114258, -0.4058700203895569, -
2.813898801803589, -2.8878121376037598, -0.32679909467697144, -
0.009458445012569427, 0.5241784453392029, -1.5524672269821167, -
0.39960819482803345, -1.3223421573638916, -1.0459154844284058, -
1.8224228620529175, -0.8091207146644592, -3.013796091079712,
0.39483505487442017, -1.6841427087783813, -0.2769816219806671,
0.7374738454818726, -4.9048333168029785, 0.9215420484542847,
0.24549220502376556, -2.218259572982788, -3.7947678565979004, -
1.7082021236419678, -4.689435958862305, -2.1829721927642822,
0.9188798069953918, 0.469594806432724, -2.881321430206299, -
1.8373267650604248, -2.799572229385376, -0.9004884362220764,
0.5844532251358032, -4.536390781402588, -1.6237584352493286, -
5.4355549812316895, -2.965827465057373, -5.383357048034668,
0.6111593842506409, 0.23205848038196564, -1.915524959564209, -
0.08068397641181946, -2.2419188022613525, -4.39054536819458, -
1.0663799047470093, 0.6035788059234619, 0.11075326800346375, -
2.788925886154175, -2.110285520553589, -4.460376739501953, -
5.093564510345459, -2.285125255584717, -0.29433637857437134, -
3.183483362197876, 1.6422218084335327, -3.8360977172851562, -
0.39338573813438416, -0.8161062598228455, 0.31698963046073914, -
3.5432841777801514, -4.131503582000732, -4.237150192260742, -
1.156965970993042, -5.0008134841918945, -0.03702395781874657, -
2.4407293796539307, -1.8092091083526611, -2.629108190536499,
0.8512980937957764, -3.956049680709839]]"

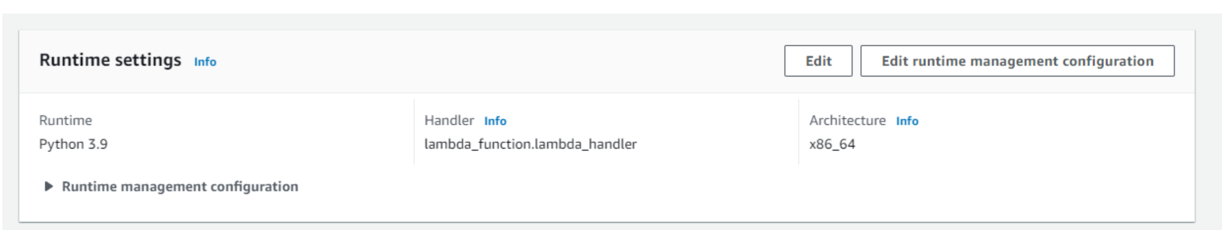
Security

Screenshot of the IAM dashboard:



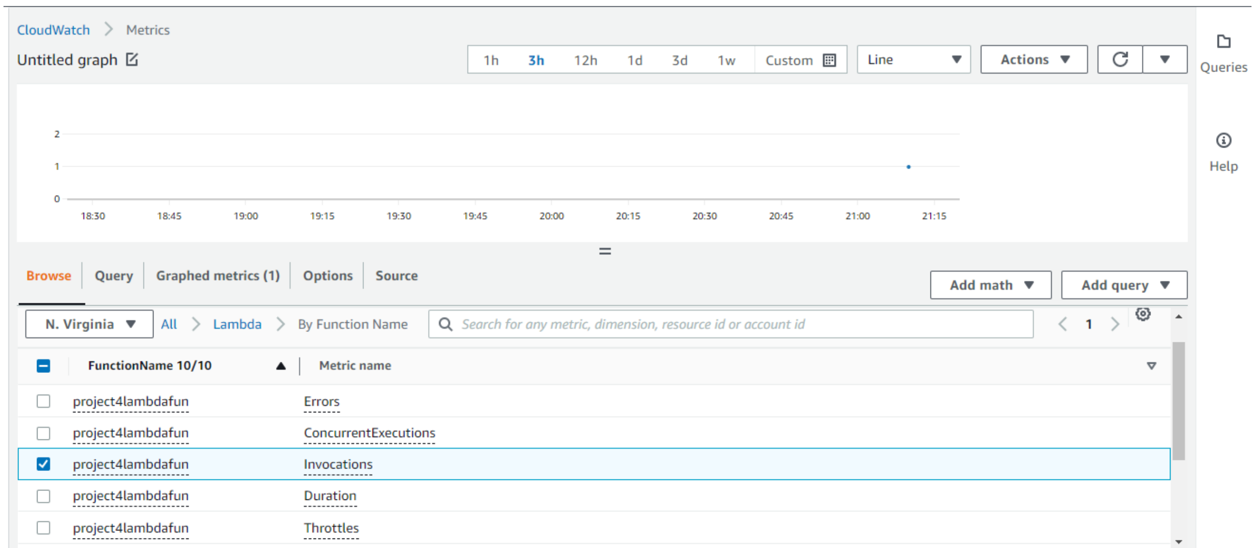


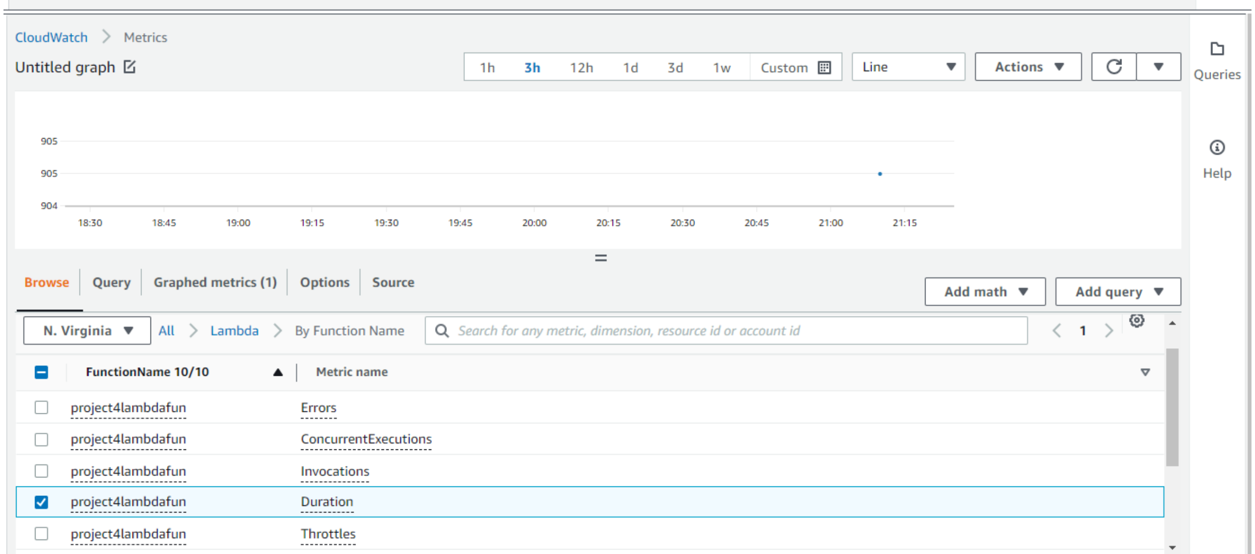
.screenshot of lambda setup:



<u>AWS workspace security:</u>

1- Checking metrics>all metrics and choose to check lambda function invocation and duration

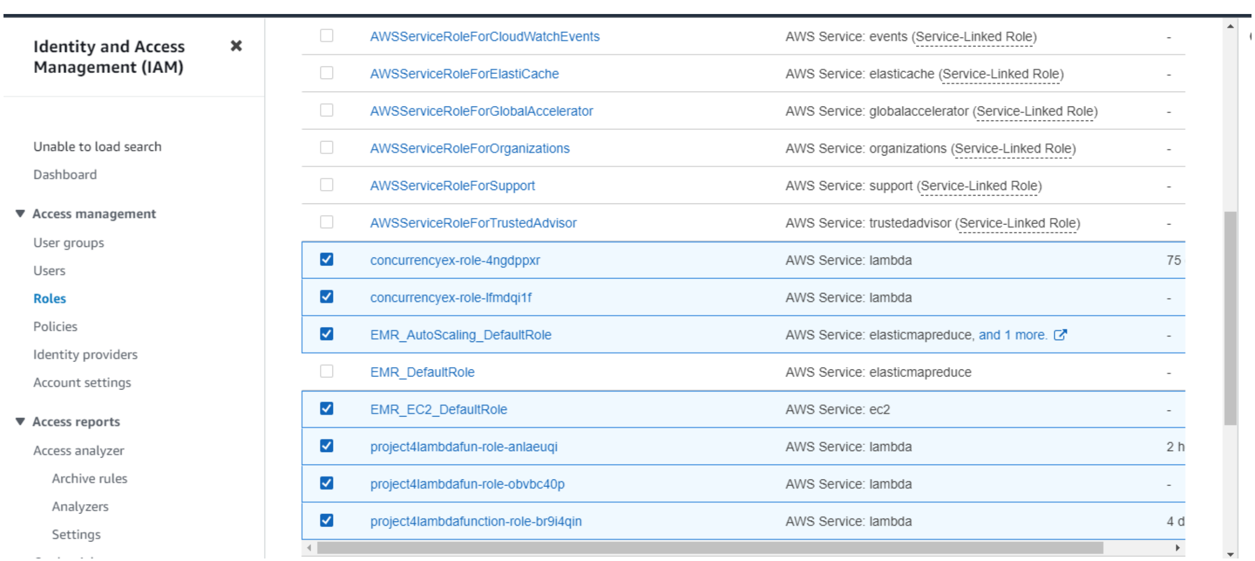




By checking lambda function invocations and duration from the previous figures it's clear that lambda function was invoked only once and having relatively short duration of invocation. So there is no security breach or unauthorized activity.

2- Checking code run on sagemaker instances:
From cloudwatch choose logs>/aws/sagemaker/NotebookInstances and checking the instance **operationalizing-AWS- MLproject/jupyter.log** for any unauthorized code or files uploaded there is no any unauthorized usage records or uploaded files to my instance.

3- Old or inactive Roles where found, those old Roles may lead to vulnerabilities, so they were deleted to insure security.



Step 5: Concurrency and auto-scaling

1-concurrency:

. Concurrency set up:

1- Reserved concurrency:
   The amount of reserve concurrency to be used = 5 instances
2- Provision concurrency:
   Always needed to be less than or equal to reserved concurrency so I choose it to equal to 3.

I choose to setup concurrency for both types to assure my ability to set up lambda function for both concurrency types, but since I don’t know exactly how much traffic lambda function expect to get, provisioned concurrency is recommended so resources are automatically provisioned based on whatever traffic comes.

Lambda function was tested after setting up concurrency:

The results below confirm that: lambda function succeeded, duration processing gets lower (from 1296.07 ms before concurrency to be 976.36 ms after concurrency) and memory size also gets lower (from 76 MB to be 68 MB)

Created provisioned concurrency configuration. Allocating provisioned concurrency can take a few minutes.

Code | Test | Monitor | Configuration | Aliases | Versions

Execution result: succeeded (logs)

▼ Details

The area below shows the last 4 KB of the execution log.


```
{
  "statusCode": 200,
  "headers": {
    "Content-Type": "text/plain",
    "Access-Control-Allow-Origin": "*"
  },
  "type-result": "<class 'str'>",
  "COntent-Type-In": "LambdaContext([aws_request_id=d26b4011-cee5-43ad-b4d2-
f0846c025ddc,log_group_name=/aws/lambda/project4lambdafunction,log_stream_name=2023/04/01/[$LATEST]eafdca36b4f0434d94533d7be14faa38,function_n
ame=project4lambdafunction,memory_limit_in_mb=128,function_version=$LATEST,invoked_function_arn=arn:aws:lambda:us-east-
```


Summary

Code SHA-256
t/3bKrHcohfDXyIQs7AQo8c3bBjvJzZVHxyyNyup4ZI=

Request ID
d26b4011-cee5-43ad-b4d2-f0846c025ddc

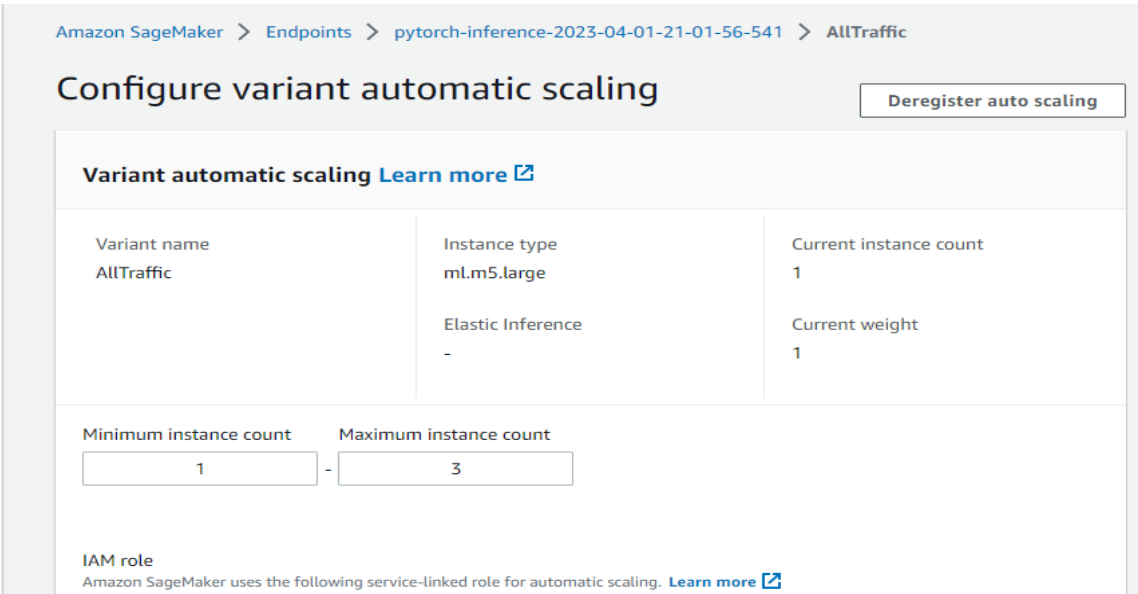
Log output

The section below shows the logging calls in your code. Click here to view the corresponding CloudWatch log group.

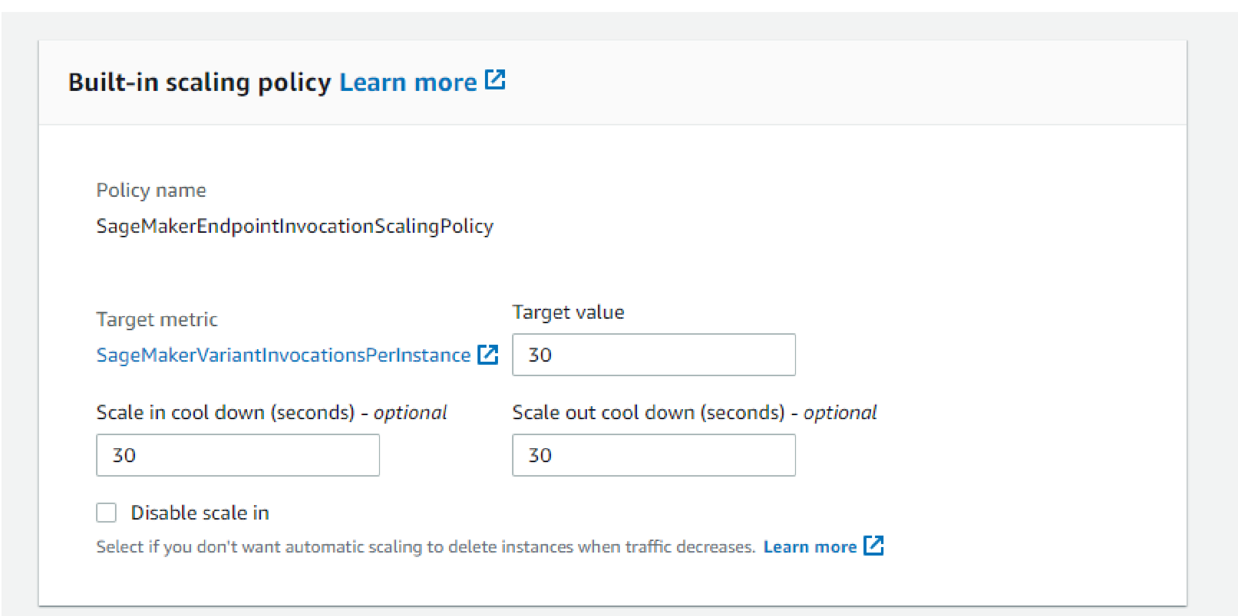
```
START RequestId: d26b4011-cee5-43ad-b4d2-f0846c025ddc Version: $LATEST
Context::: LambdaContext([aws_request_id=d26b4011-cee5-43ad-b4d2-
f0846c025ddc,log_group_name=/aws/lambda/project4lambdafunction,log_stream_name=2023/04/01/[$LATEST]eafdca36b4f0434d94533d7be14faa38,function_n
ame=project4lambdafunction,memory_limit_in_mb=128,function_version=$LATEST,invoked_function_arn=arn:aws:lambda:us-east-
1:535830523552:function:project4lambdafunction,client_context=None,identity=CognitoIdentity([cognito_identity_id=None,cognito_identity_pool_id=None])])
EventType:: <class 'dict'>
END RequestId: d26b4011-cee5-43ad-b4d2-f0846c025ddc
REPORT RequestId: d26b4011-cee5-43ad-b4d2-f0846c025ddc  Duration: 976.36 ms  Billed Duration: 977 ms  Memory Size: 128 MB  Max Memory Used: 68 MB
Init Duration: 330.55 ms
```

2-Auto-scalling:

Set up:



In normal time where requests are low endpoint will run with 1 instance, while with higher requests the endpoint will run will automatically scale to have 2 or 3 instances.



Scale in cool down and Scale out cool down = 30seconds this is going to be relatively responsive endpoint, 30s to deploy more instances for elevated traffic and 30s to delete extra instances for decreased traffic.

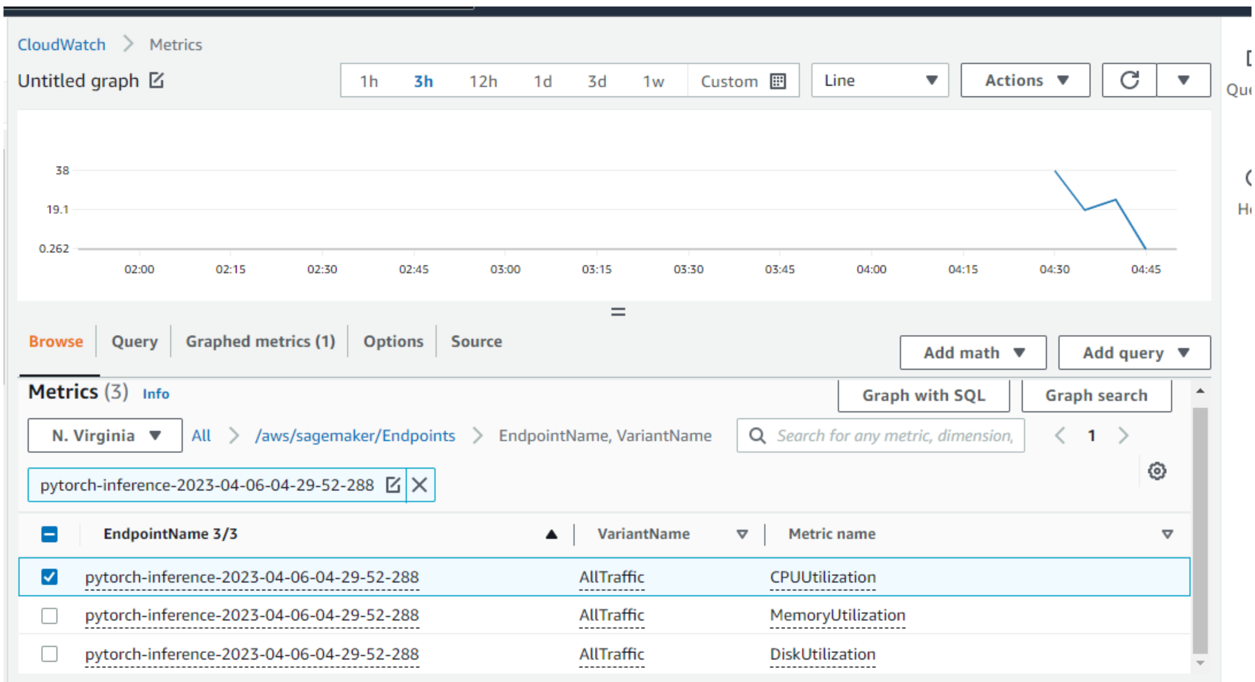
And Target value = 30 this value is the endpoint decide when to initiate auto-scaling, this value is neither very low so endpoint is able to create new instances and deal with all types of traffic but the cost will be high, nor very high so endpoint will have reliability issue.

<u>Testing auto-scaled endpoint:</u>

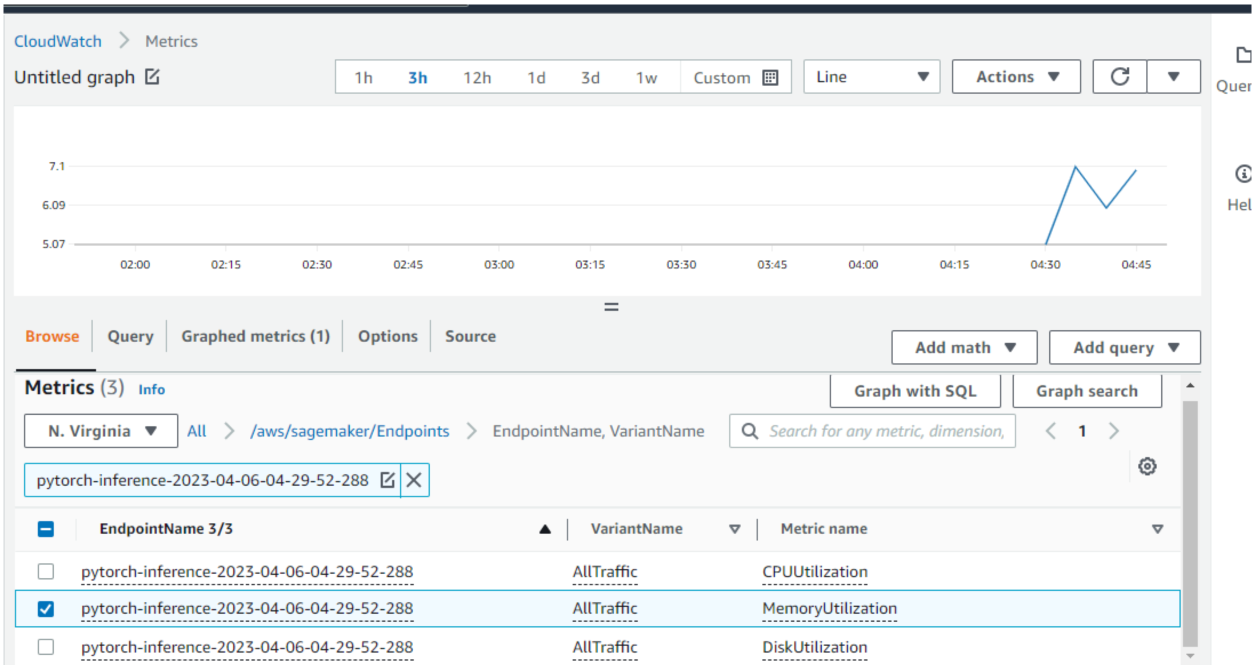
Add the following code to Train_and_deploy-solution.ipynb and run it :

```
#testing endpoint Auto-scaling
import requests
import json
n=0
while n<400:
    request_dict={ "url": "https://s3.amazonaws.com/cdn-origin-etr.akc.org/wp-content/uploads/2017/11/20113314/Carolina-Dog-s
    img_bytes = requests.get(request_dict['url']).content
    print(predictor.predict(json.dumps(request_dict), initial_args={"ContentType": "application/json"}))
    n=n+1
```

This code just run in few seconds. Checking metrics related to invoked endpoint:



It's common in case of repeated invocations of endpoint that it lead to 100% of CPU utilization for some length of time as shown at the beginning of invocations in the above figure and how well the endpoint handle these invocations by automatically scale to have 2 or 3 instances which leads to reduce CPU utilization.



Memory utilization also reach to its limits but again endpoint handle this very well and we notice how it's lower memory usage.