Operationalizing an AWS ML Project

- Kanchi Tank

Note: *For each of the required steps all the snapshots are also present in the* ***snapshots*** *folder in the repository.*

1. Initial Setup

I have chosen the "**ml.t2.medium**" instance type for Notebook instance (Figure 1 - SageMaker Notebook Instance). There are multiple reasons for selecting this instance type for my notebook.

- Firstly, for completing the execution of this project's jupyter notebooks we **do not need** a very computationally **powerful CPU** and **high RAM.**
- We will need to keep this **notebook** instance in "**inService**" **status** for a **long time** while we are working on the project
- In order to **avoid high costs,** we should **select a notebook** that is **low in per hour cost** and **offers** reasonably **good CPU and RAM**.
- So looking at the instance type and their pricing: https://aws.amazon.com/SageMaker/pricing/ , we have two choices "ml.t2.medium" and "ml.t3.medium". Both have 2 vCPU and 4 GB Memory, and as per doc "ml.t3.medium" has a slightly higher cost as it has a fast boot time.
- Now given that we do have a critical requirement for a fast boot time, so we can go ahead with the "**ml.t2.medium**" as it offers the same computational power and is lower in per hour cost.

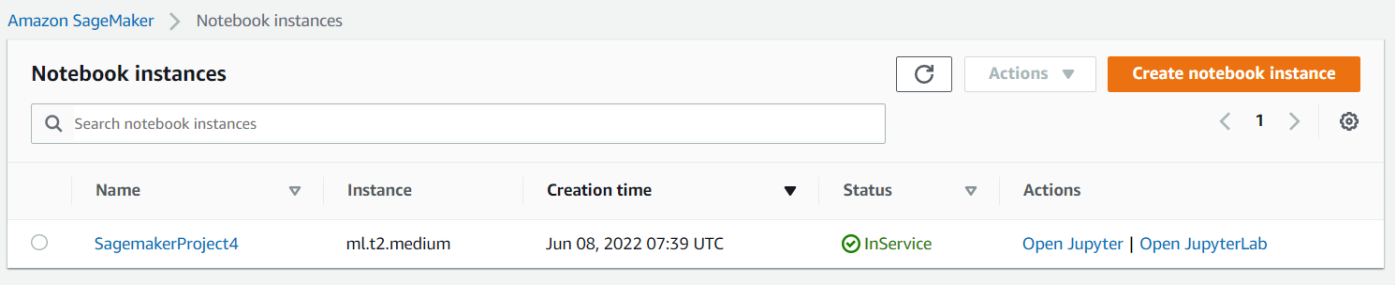

Figure 1. SageMaker Notebook Instance

The dog breed dataset was uploaded to a newly created S3 bucket, successfully.

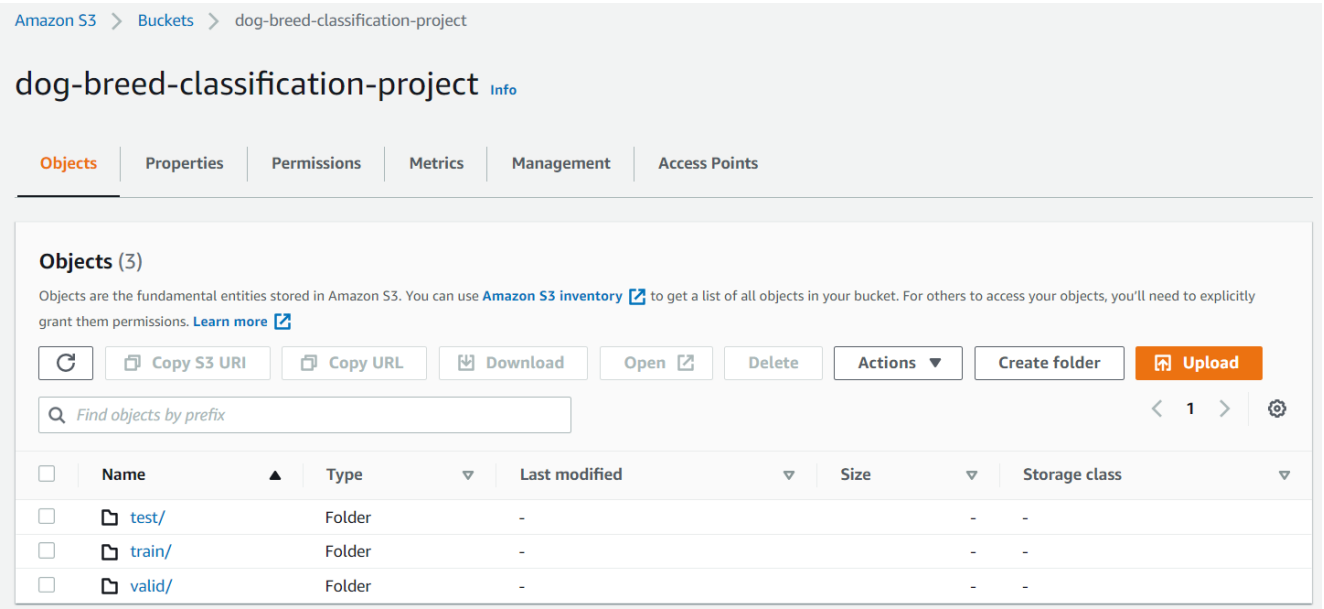

Figure 2. S3 Bucket snapshot

2. SageMaker Training and Deployment

For hyperparameter tuning I used the same “ml.m5.xlarge” instance_type , however since it took too much time for the tuning, while the training process I tried to increase the processing power a bit by using the “ml.m5.2xlarge” for the single instance and multi instance training purposes.

Hyperparameter tuning job (max_jobs = 2, max_parallel_jobs = 2) completed successfully:

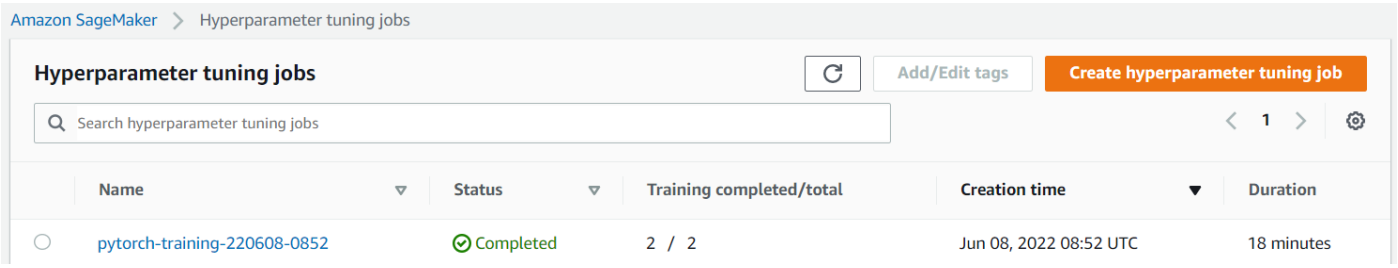

Figure 3. Hyperparameter Tuning Job

However, upon training the model with the best parameters from above tuning, the model gave a 0 test accuracy! So increased the max_jobs = 6, max_parallel_jobs = 3 and also changed its instance_type = “ml.m5.xlarge” to speed up the computations a bit.

Reran the hyperparameter jobs and it executed successfully:

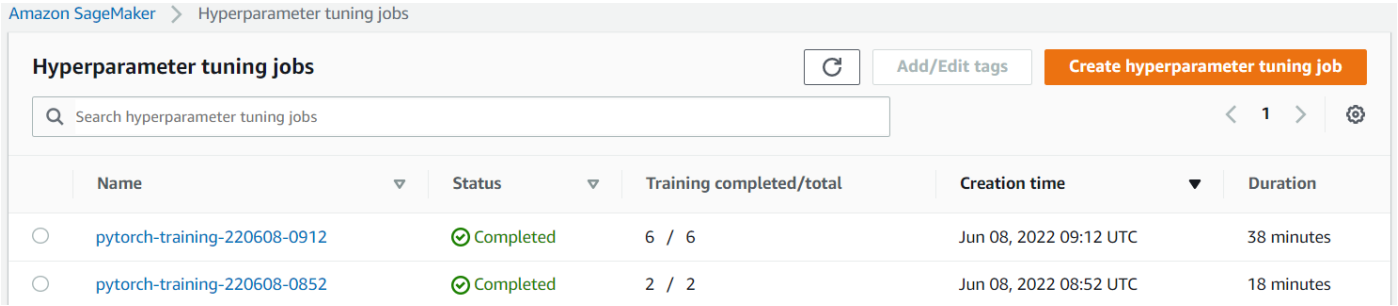

Figure 4. Hyperparameter jobs summary

Post which triggered the **single instance** and **multi-instance training jobs**. Jobs completed successfully. For snapshot of the training jobs refer the images folder in the repository.

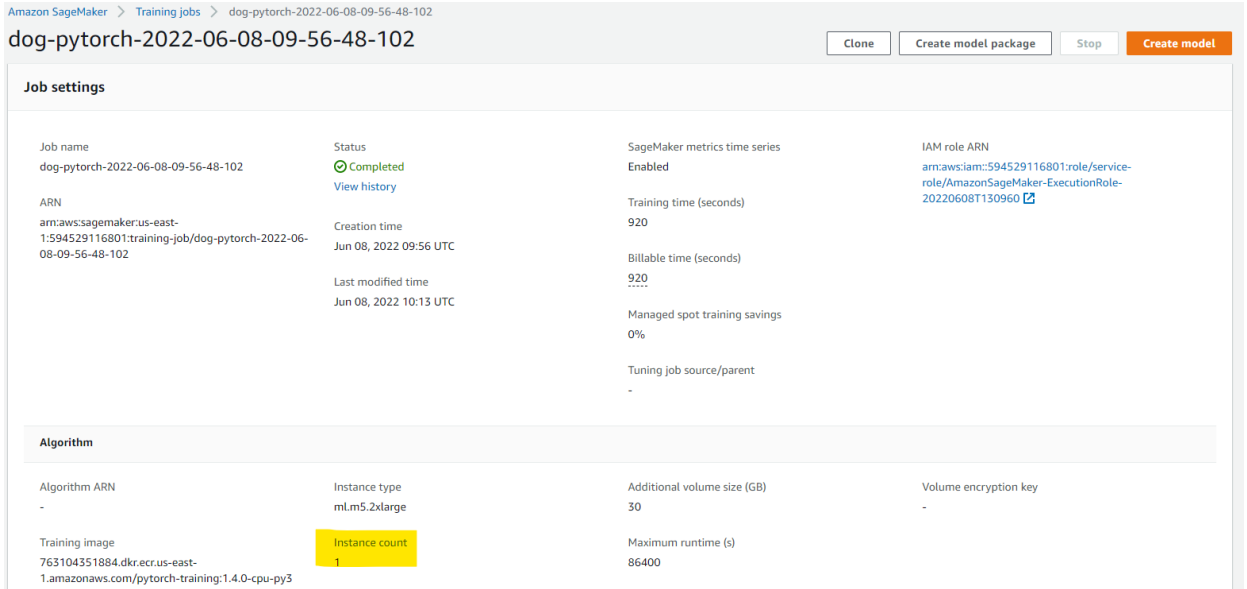

Figure 5. Single Instance Training Job

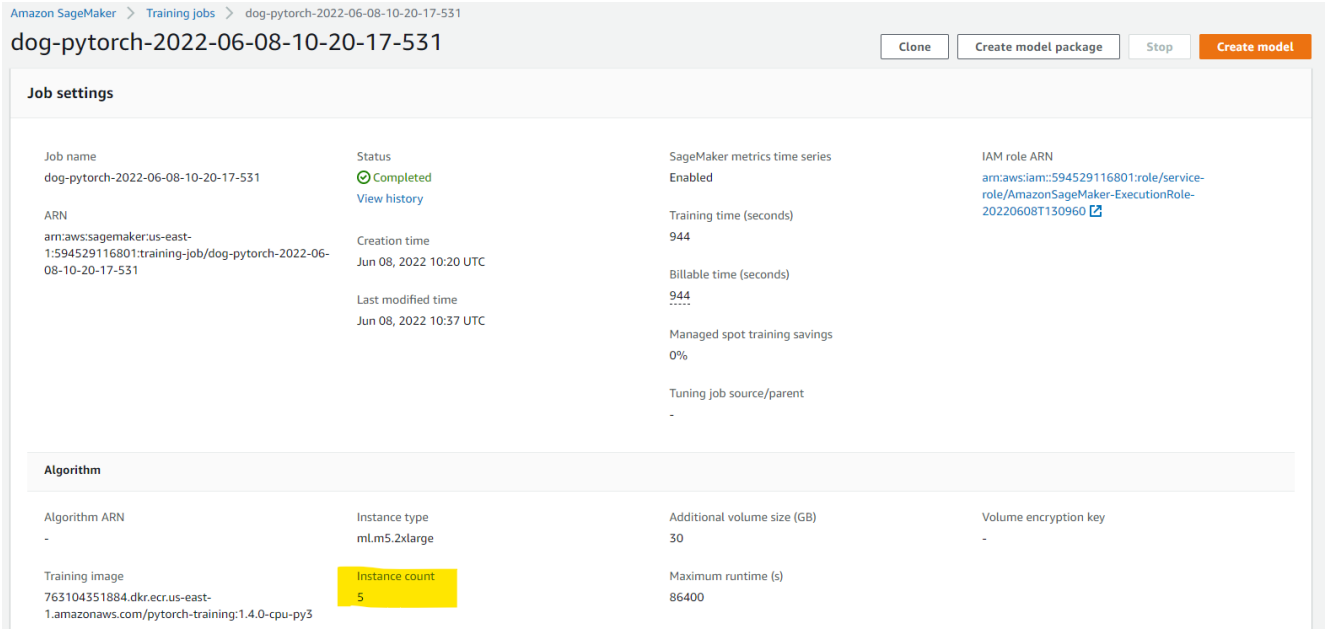

Figure 6. Multi Instance Training Job

Deployed Endpoints:

- Single instance deployed endpoint: “pytorch-inference-2022-06-08-10-16-07-494”
- Multi instance deployed endpoint: “pytorch-inference-2022-06-08-20-39-47-131”

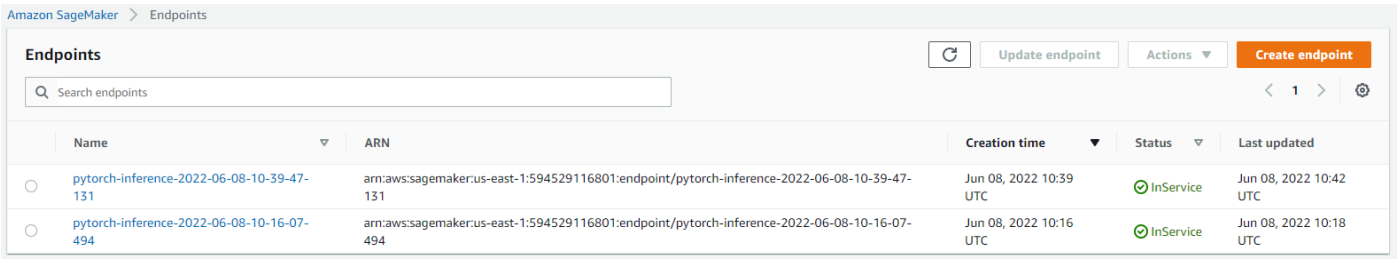

Figure 7. SageMaker Endpoints

3. EC2 Training

- We have utilized the **t2.xlarge** instance and the **Deep Learning AMI (Amazon Linux 2) Version 55.0**. This seems like a reasonable balance of performance and affordability.
- As per the documentation, T2 instances can sustain high CPU performance for as long as a workload needs it.
- For most general-purpose workloads, T2 instances will provide ample performance without any additional charges.
- Similarly, because we don't know the duration for which we might need to keep this EC2 instance running for training, it's better to go with a medium size instance so we don't have to pay for a large instance while we're doing setup, debugging and other tasks.

Difference between ec2train1.py (EC2 script) and train_and_deploy-solution.ipynb + hpo.py (SageMaker scripts)

- There is no logic for calling any Estimator or Tuner functions in the EC2 script. The code in the EC2 script is responsible for saving the model to the local path. While in the SageMaker scripts this was handled internally by SageMaker where the model data was stored to a S3 location.
- In the EC2 training script, all the variables like hyperparameters and output locations, etc are already mentioned in the script itself and so there is no need for **argparse.** Meaning while running the EC2 script we do not need to mention any arguments.

- In the EC2 script the training happens on the same server on which the script is invoked/executed, however in the SageMaker scripts the training job that is invoked, it runs on a separate container than the one on which the SageMaker notebook is running.
- Another difference is that ec2train1.py lacks the main function
- For the EC2 Training, given that the training data and model, all are stored on the EC2 instance host itself it would be difficult to deploy the saved model to an endpoint in SageMaker. If we wish to do that then we might need to manually upload the model first to SageMaker and then use that to deploy an endpoint. This is not the case in models trained via the SageMaker notebook instances, as the model can be easily deployed to an endpoint.

For snapshot refer below:

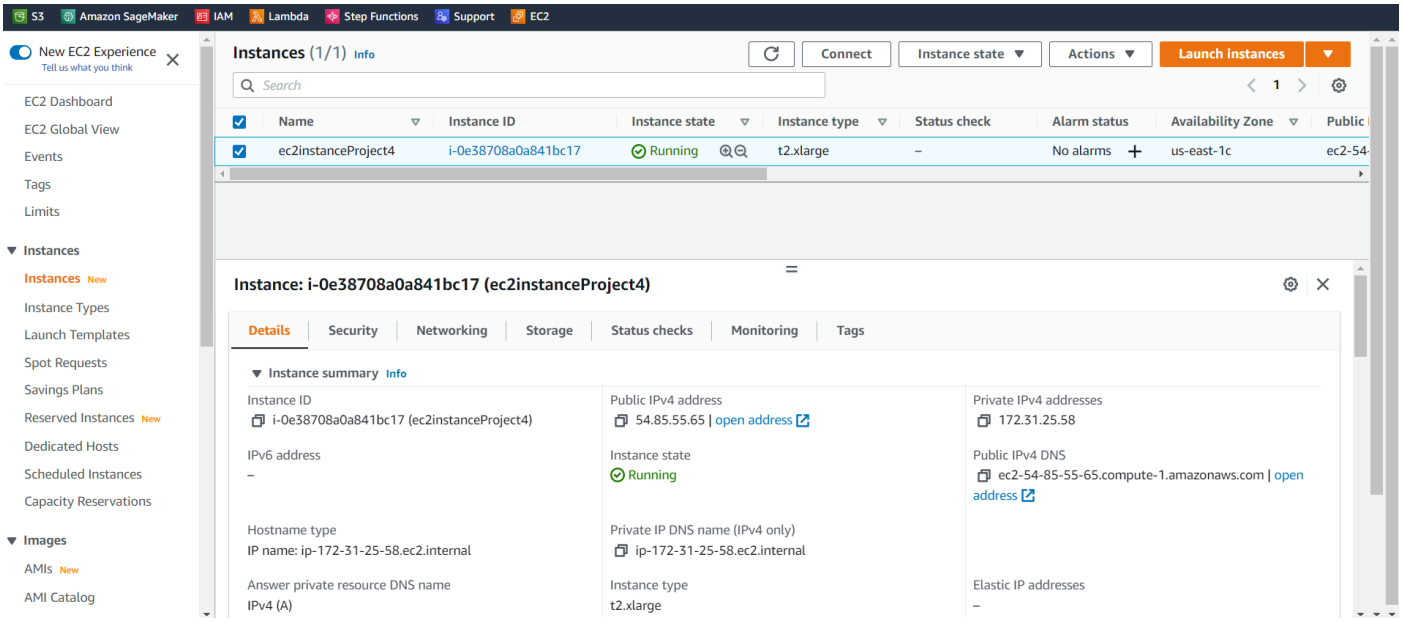

Figure 8. EC2 Instance snapshot

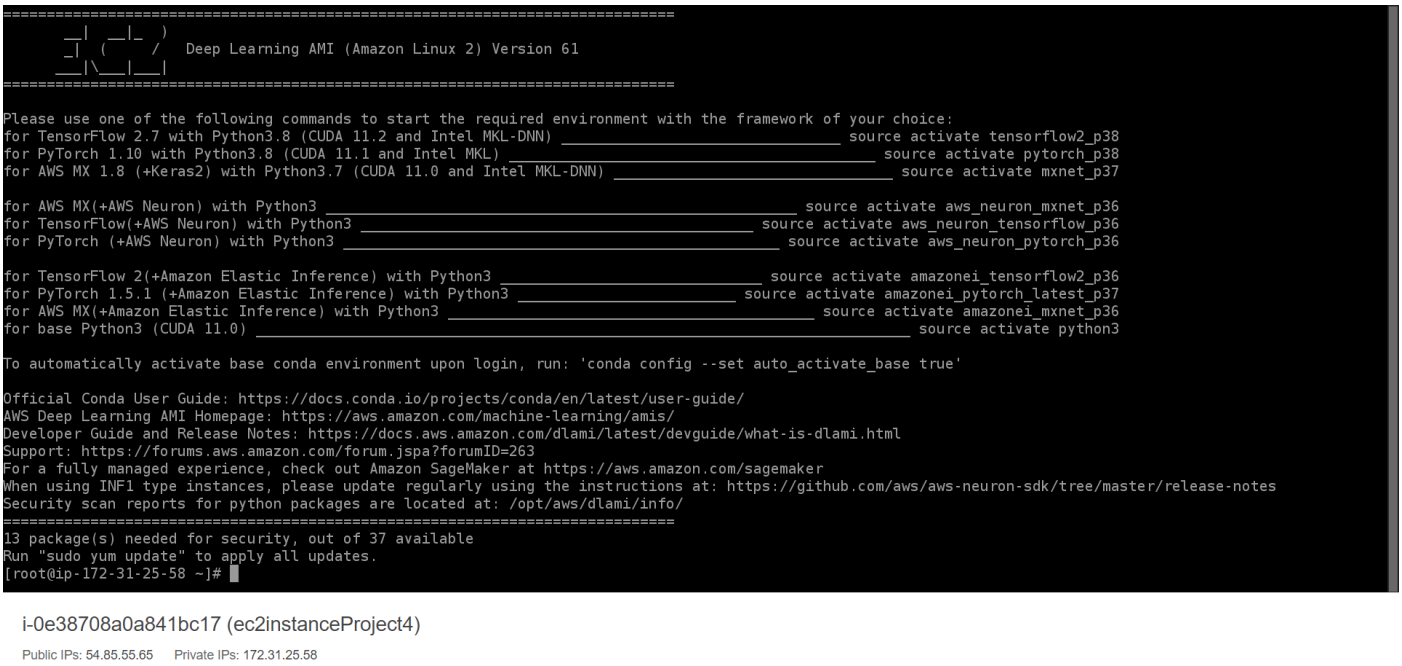

Figure 9. EC2 Terminal

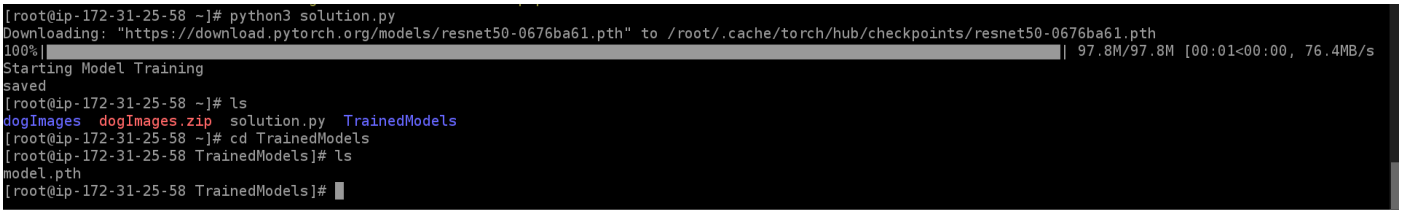

Figure 10. EC2 Training saved - model.pth

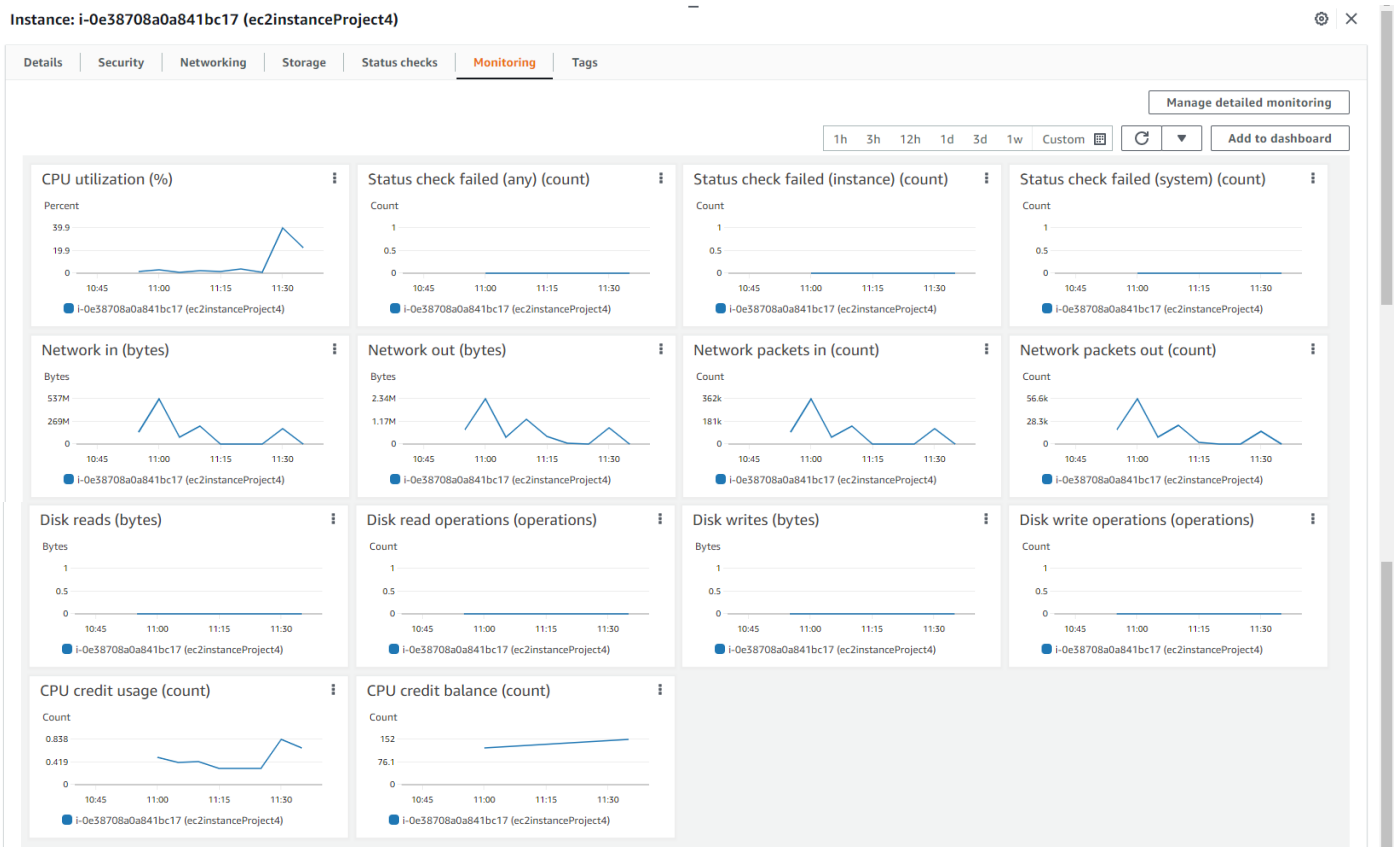

Figure 11. EC2 Instance Resource Metrics

4. Lambda functions

- The lambda functions will be used for invoking our deployed endpoints.
- The lambda function implemented in this project expects the image inputs in json format, which is used to invoke the model's deployed endpoint
- Given we have two endpoints deployed, one for the single instance training and the other for the multi-instance training, we will only use the multi-instance training jobs endpoint and create a lambda function for invoking that endpoint.
- Multi instance trained endpoint that we will be using: "**pytorch-inference-2022-06-08-20-39-47-131**"
- We created the lambda function with the corresponding changes to invoke the endpoint:

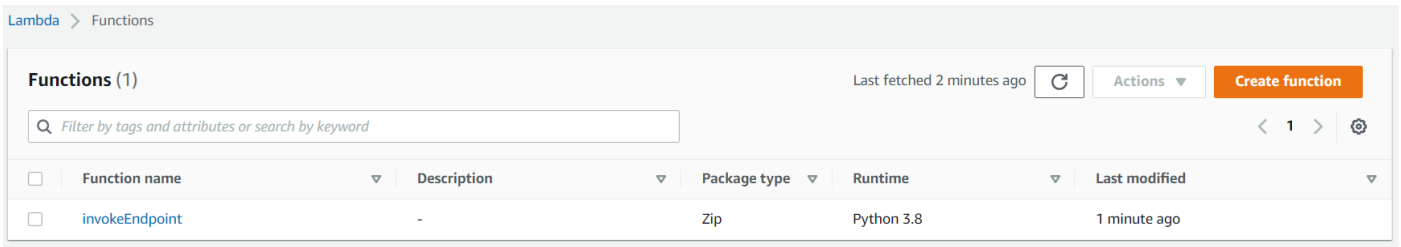

Figure 12. Lambda Function

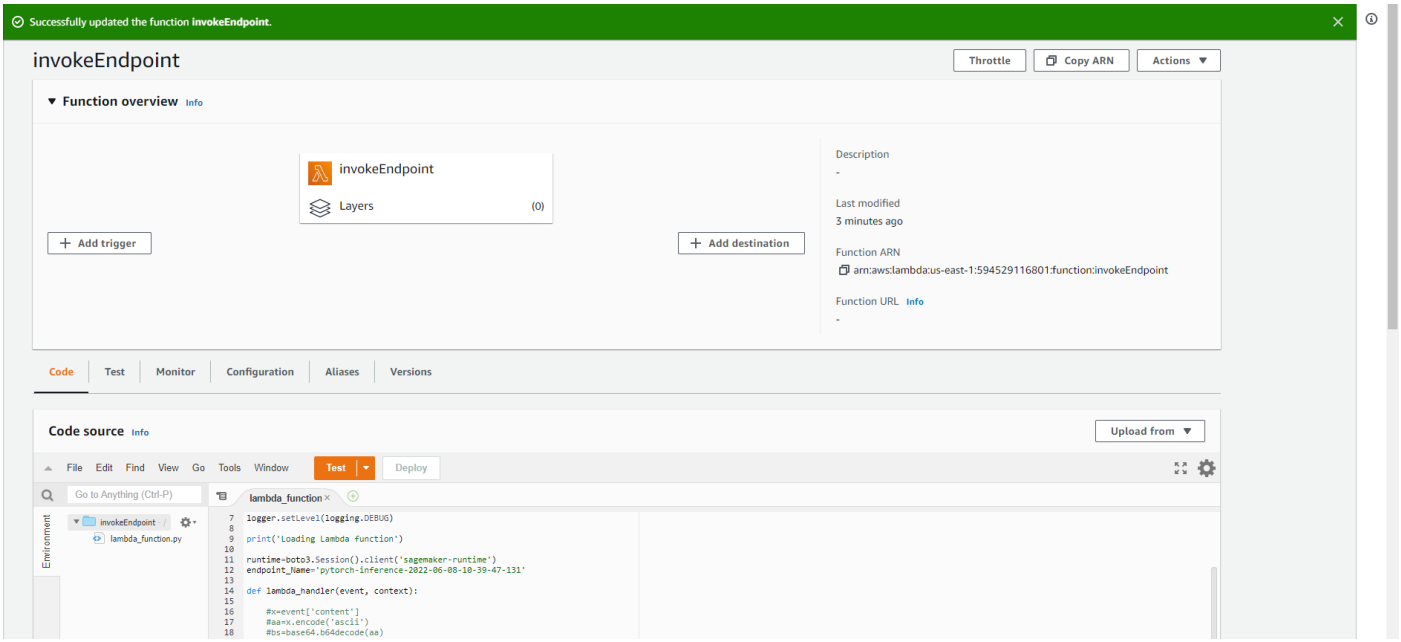

Figure 13. Lambda Function

5. Security and Testing

- We had created a new role for this lambda functions with basic accesses.
- We used the below test event to test our lambda function:

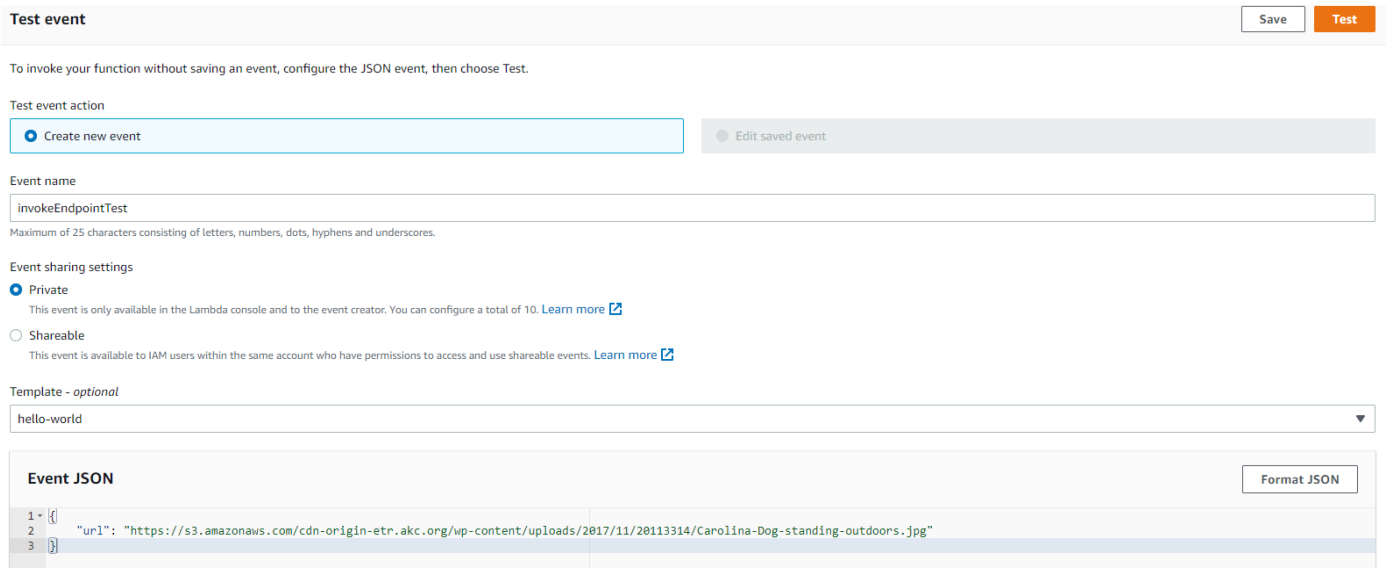

Figure 14. Lambda Function Test Event

- Now when we tried to execute the test event we got and **AccessDeniedException.** This was expected as the lambda function did not have access to invoke the SageMaker endpoint.

- Error Message:


```
{
  "errorMessage": "An error occurred (AccessDeniedException) when calling the
InvokeEndpoint operation: User: arn:aws:sts::594529116801:assumed-role/invokeEndpoint-role-
4qryi0br/invokeEndpoint is not authorized to perform: sagemaker:InvokeEndpoint on resource:
arn:aws:sagemaker:us-east-1:594529116801:endpoint/pytorch-inference-2022-06-08-10-39-47-131
because no identity-based policy allows the sagemaker:InvokeEndpoint action",
  "errorType": "ClientError",
  "stackTrace": [
```

```
    "  File \"/var/task/lambda_function.py\", line 24, in lambda_handler\n
response=runtime.invoke_endpoint(EndpointName=endpoint_Name,\n",
    "  File \"/var/runtime/botocore/client.py\", line 391, in _api_call\n    return
self._make_api_call(operation_name, kwargs)\n",
    "  File \"/var/runtime/botocore/client.py\", line 719, in _make_api_call\n    raise
error_class(parsed_response, operation_name)\n"
  ]
}
```

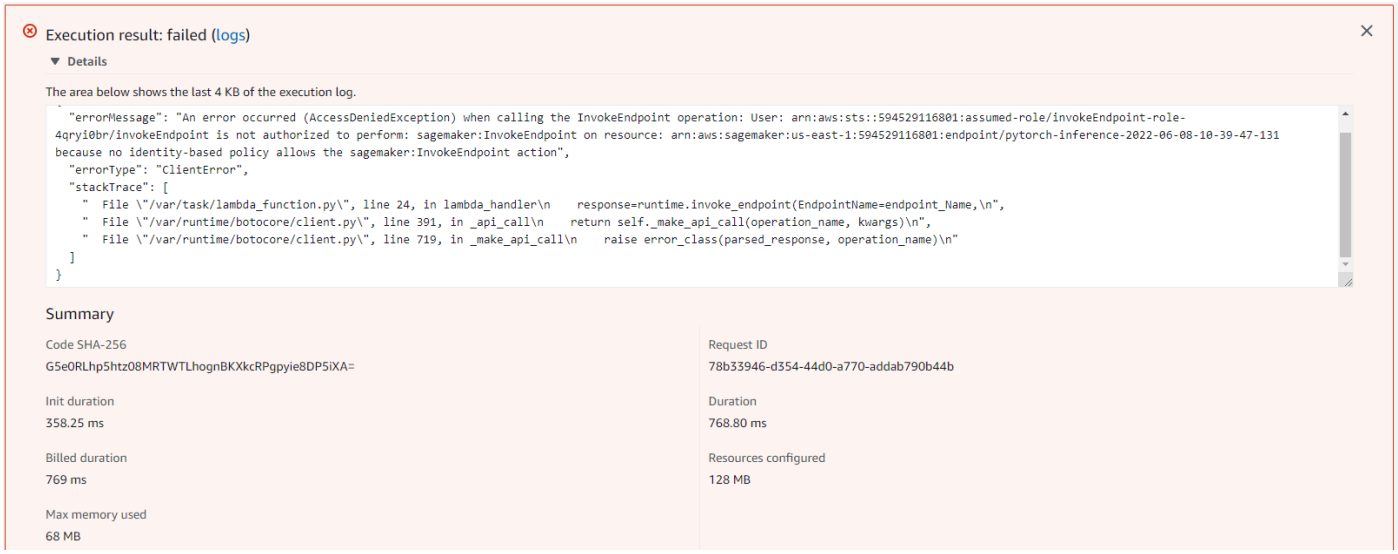

Figure 15. Lambda function test event failure response

- So went ahead and added the "**SageMakerFullAccess**" policy role to the lambda function's role.

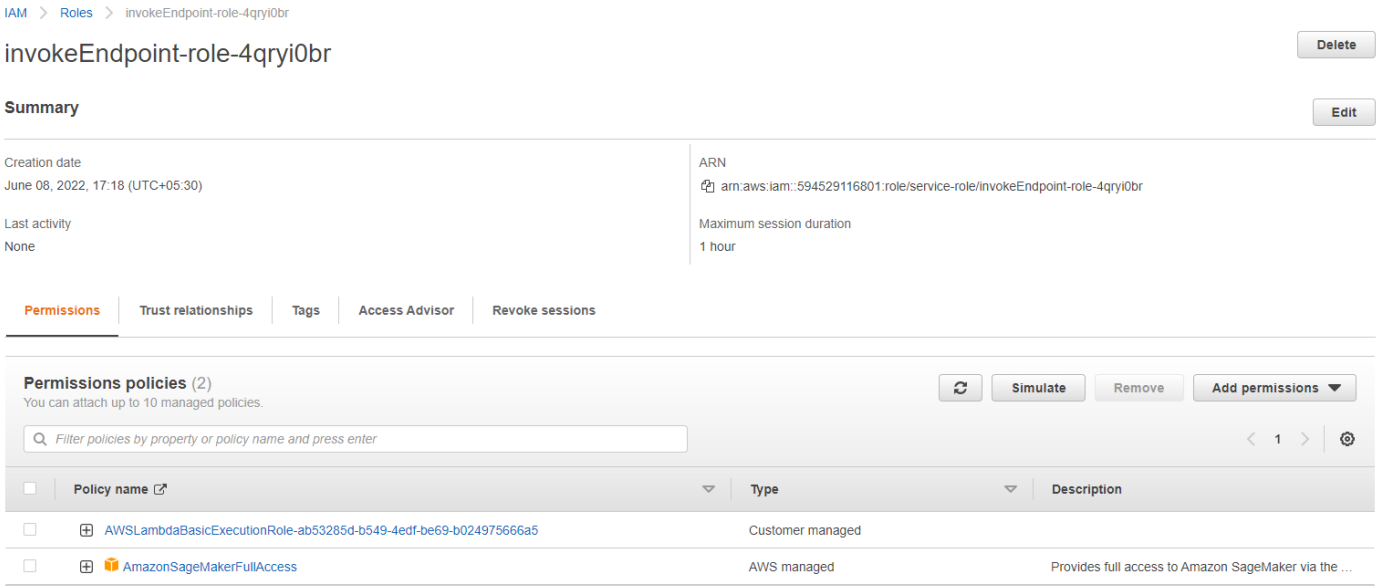

Figure 16. Lambda function role IAM permissions

- Post which we got the following output from the test event:
- (Please note that there are 133 dog breed and not 33 as mentioned in the project instructions. So, we do expect there to be around 133 elements in the response object.)
- Response:

```
{
  "statusCode": 200,
  "headers": {
    "Content-Type": "text/plain",
    "Access-Control-Allow-Origin": "*"
  },
  "type-result": "<class 'str'>",
```

```
  "COntent-Type-In": "<__main__.LambdaContext object at 0x7f3802e37b20>",
  "body":      "[[-7.732433795928955,      -7.176344394683838,      -6.378891468048096,      -
1.5176353454589844,    -3.4210686683654785,    -9.79492473602295,    -6.717096328735352,    -
1.7850936651229858,    -8.523921012878418,    -1.404198169708252,    -0.7228893637657166,    -
5.840402603149414,    -5.7331461906433105,    -2.0368082523345947,    -9.905051231384277,    -
6.480345726013184,     -10.379494667053223,     -2.0244674682617188,     -10.030835151672363,
0.24102921783924103,    -7.826473712921143,    -2.3736279010772705,    -7.411016464233398,    -
9.585257530212402,    -6.935138702392578,    -10.370619773864746,    -6.9618120193481445,    -
5.8000054359436035,    -7.852663516998291,    -4.261888027191162,    -8.121268272399902,    -
2.3057727813720703,    -9.783154487609863,    -4.807986736297607,    -8.92819595336914,    -
8.51605224609375,    -8.291374206542969,    -5.918830394744873,    -3.329312324523926,    -
7.71879768371582,    -4.240849018096924,    -8.23236083984375,    -0.6575157642364502,    -
3.2674717903137207,    -1.3542909622192383,    -12.321500778198242,    -5.892731666564941,    -
0.7067917585372925,    -4.4090352058410645,    -3.0732972621917725,    -3.965545892715454,    -
12.868936538696289,    -10.151001930236816,    -5.127017974853516,    -9.019100189208984,    -
2.250539779663086,    -6.182336807250977,    -10.275800704956055,    -1.599454641342163,    -
6.101325988769531,    -13.15511417388916,    -13.388274192810059,    -12.73229694366455,    -
6.6955180168151855,    -3.183396577835083,    -13.372259140014648,    -2.6620848178863525,    -
5.736507892608643,    -3.5820565223693848,    -0.9081988334655762,    1.058609127998352,    -
7.375503063201904,    -7.269904136657715,    -5.875174045562744,    -8.469480514526367,    -
4.669737815856934,    -8.637510299682617,    -2.2320778369903564,    -6.134050369262695,    -
4.5598530769348145,    -2.207737922668457,    -10.790899276733398,    -2.3271493911743164,    -
3.509336471557617,    -10.742084503173828,    -11.33560848236084,    -7.614548683166504,    -
12.094978332519531,    -5.790018558502197,    -2.1963579654693604,    -9.642144203186035,    -
7.176695346832275,    -6.40428352355957,    -11.0121488571167,    -6.617325305938721,    -
3.0795605182647705,    -7.987880706787109,    -3.7228918075561523,    -9.426275253295898,    -
7.498056888580322,    -11.697783470153809,    -1.4410004615783691,    -2.5681569576263428,    -
4.899290084838867,    -3.1299190521240234,    -5.716061115264893,    -8.185333251953125,    -
3.402395248413086,    -2.385983467102051,    -2.442678928375244,    -8.68392562866211,    -
2.8638479709625244,    -11.222406387329102,    -10.302404403686523,    -7.736847877502441,    -
4.212231636047363,    -7.31355619430542,    -4.08905029296875,    -12.546906471252441,    -
2.2557764053344727,    -2.3533856868743896,    -6.100186347961426,    -7.259090900421143,    -
6.4662346839904785,    -7.800014495849609,    -6.287154197692871,    -6.792305946350098,    -
2.09424090385437,    -4.197015762329102,    -12.041462898254395,    -10.169670104980469,    -
1.5505112409591675,  -6.9473676681518555]]"
}
```


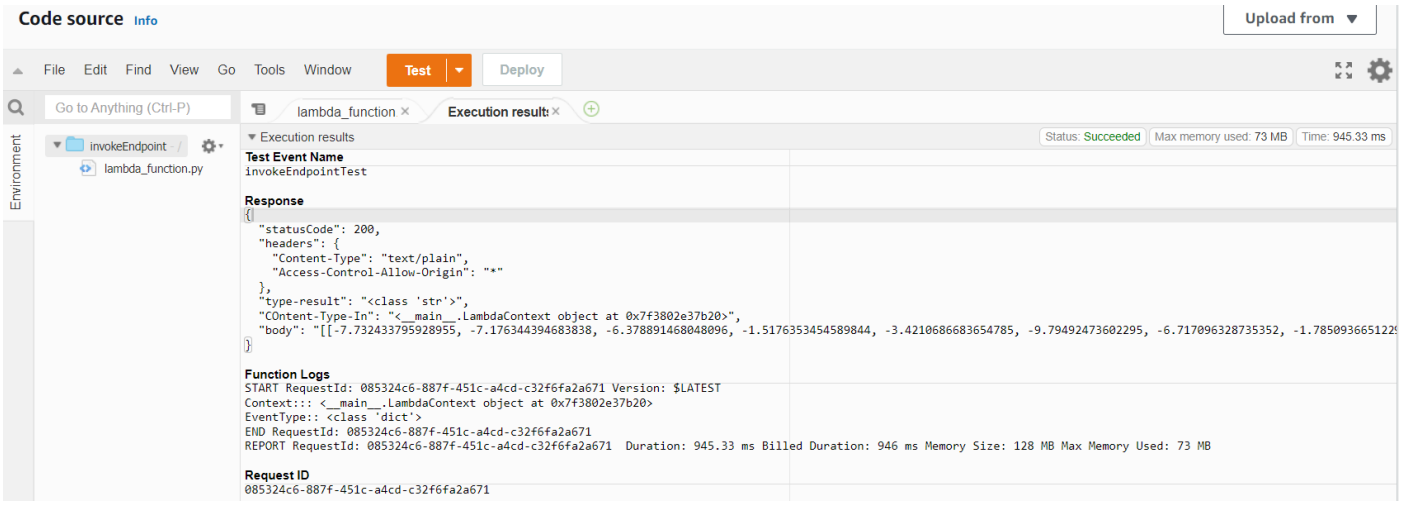

Figure 17. Lambda function test event success response

- I'm concerned about the “Full Access” type permission policies that are available.
- For example, for this lambda functions we have provided the lambda function a Full Access to SageMaker resources, but this does not seem to follow the concept of least privilege access.
- Ideally, one should only allow these lambda functions to query the endpoints that they're intended and allowed to query.
- We will have to do some more analysis to figure out if there's anything we could do about it.
- Furthermore, another concern is that the account's root user does not employ MFA

- Looking at the IAM roles that are currently active, all the roles seem to be necessary and most of the roles have been added on a per need basis.
- However, we need to keep an eye on the roles dashboard to make sure only relevant roles necessary for currently active projects, are the only roles that are in active state to prevent unauthorized accesses.

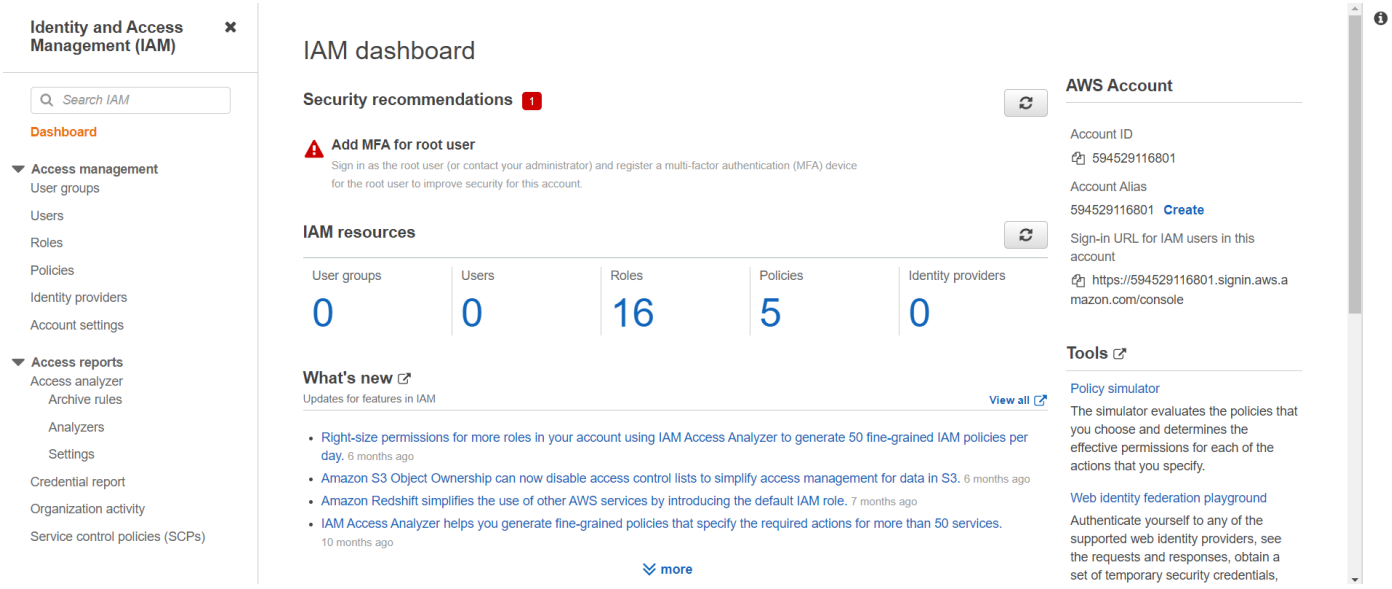

Figure 18. IAM Dashboard

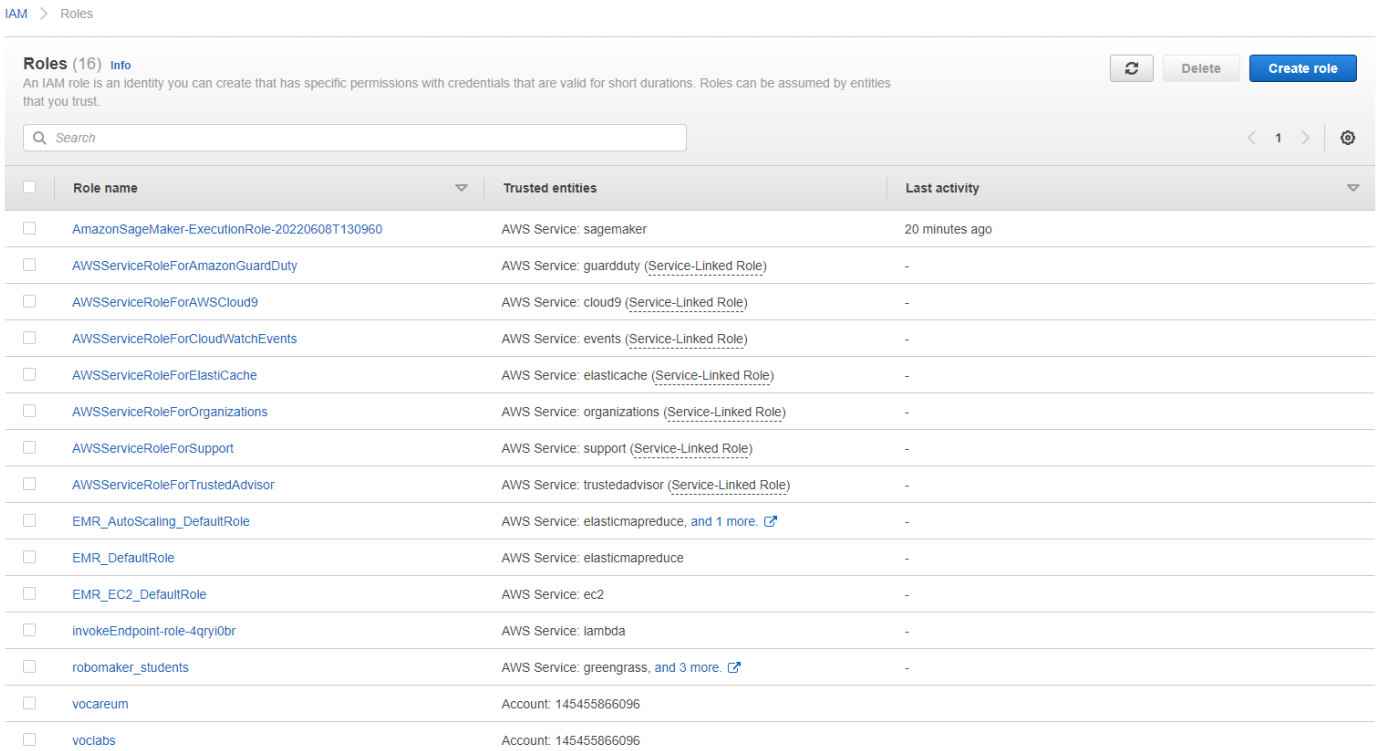

Figure 19. IAM Roles

6. Concurrency and Auto-scaling

- Before adding in configs for Concurrency and Auto-scaling for our lambda functions we will first create a version config for our lambda function.

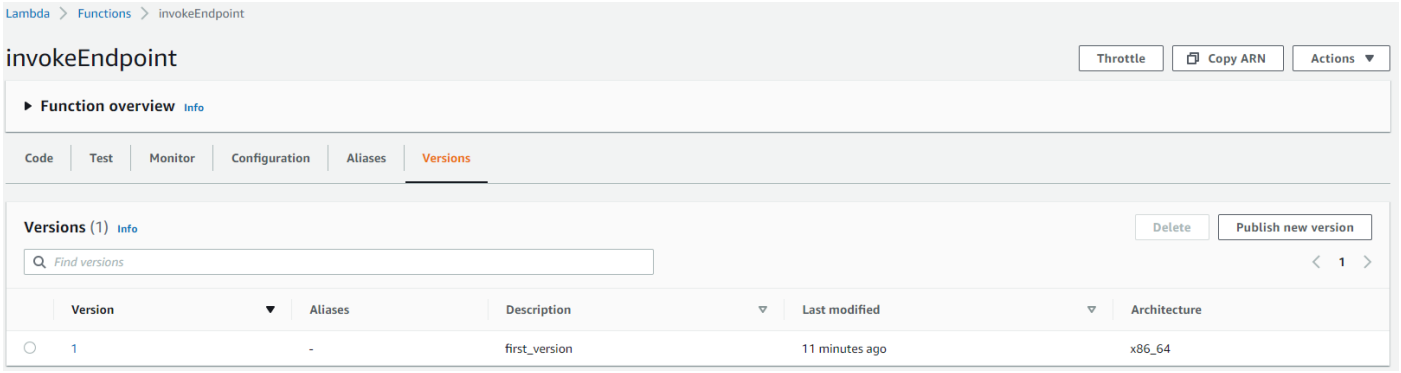

Figure 20. Lambda function version config

- For the lambda function we have set the **reserved concurrency** to be 5. This implies that the lambda function would be able to handle up to 5 requests concurrently at the same time. This would help lower latency issues in situations when there is higher traffic than usual.

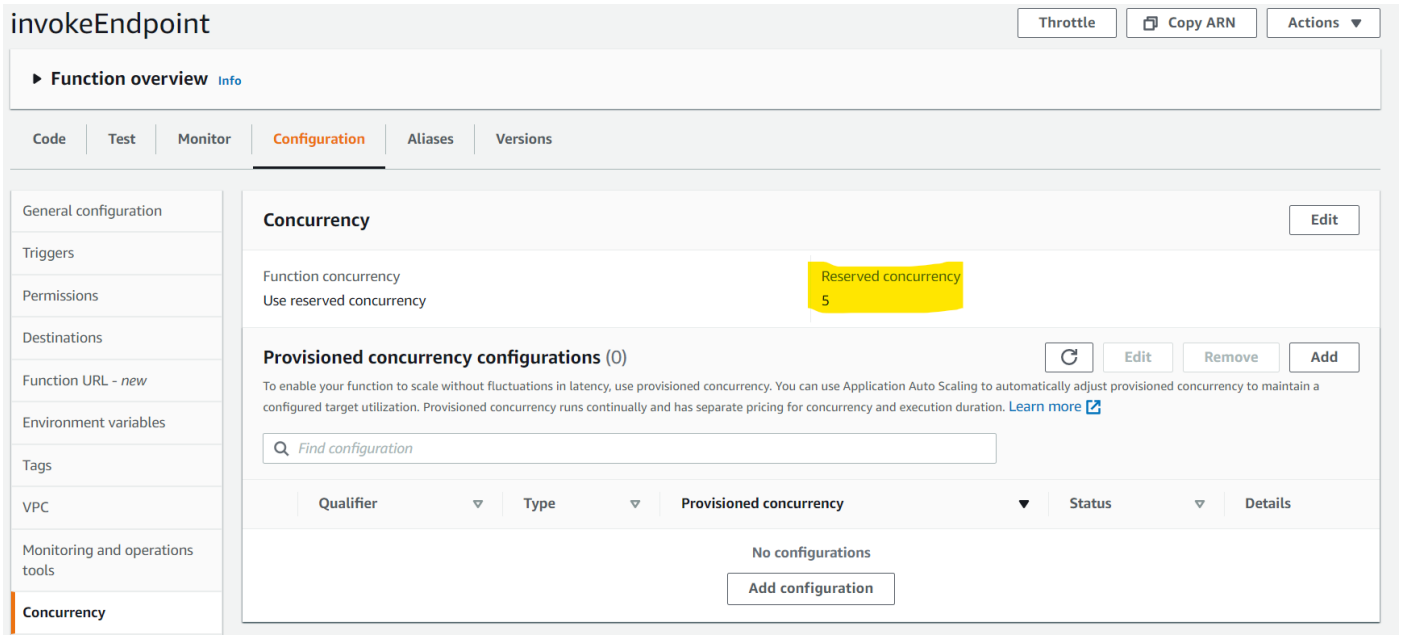

Figure 21. lambda function reversed concurrency

- Given the current use case, ideally using only reserved concurrency should have sufficed for our use case and we might not need to consider using the provisioned concurrency configs. However, for the sake of completion, I tried to add in the config for the provisioned concurrency as well. We set the provision concurrency to be 2.

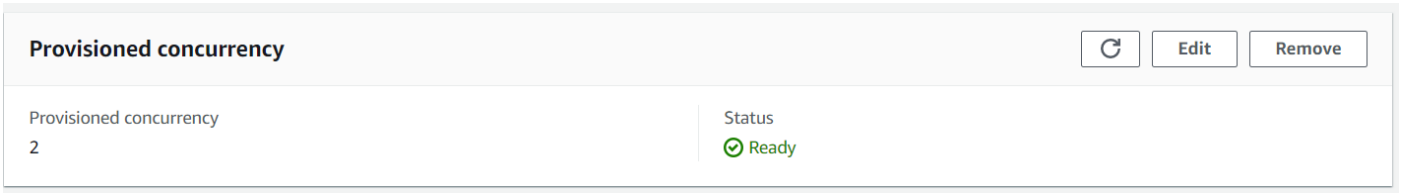

Figure 22. lambda function provisioned concurrency

- For adding config for auto-scaling we have added the below configs:

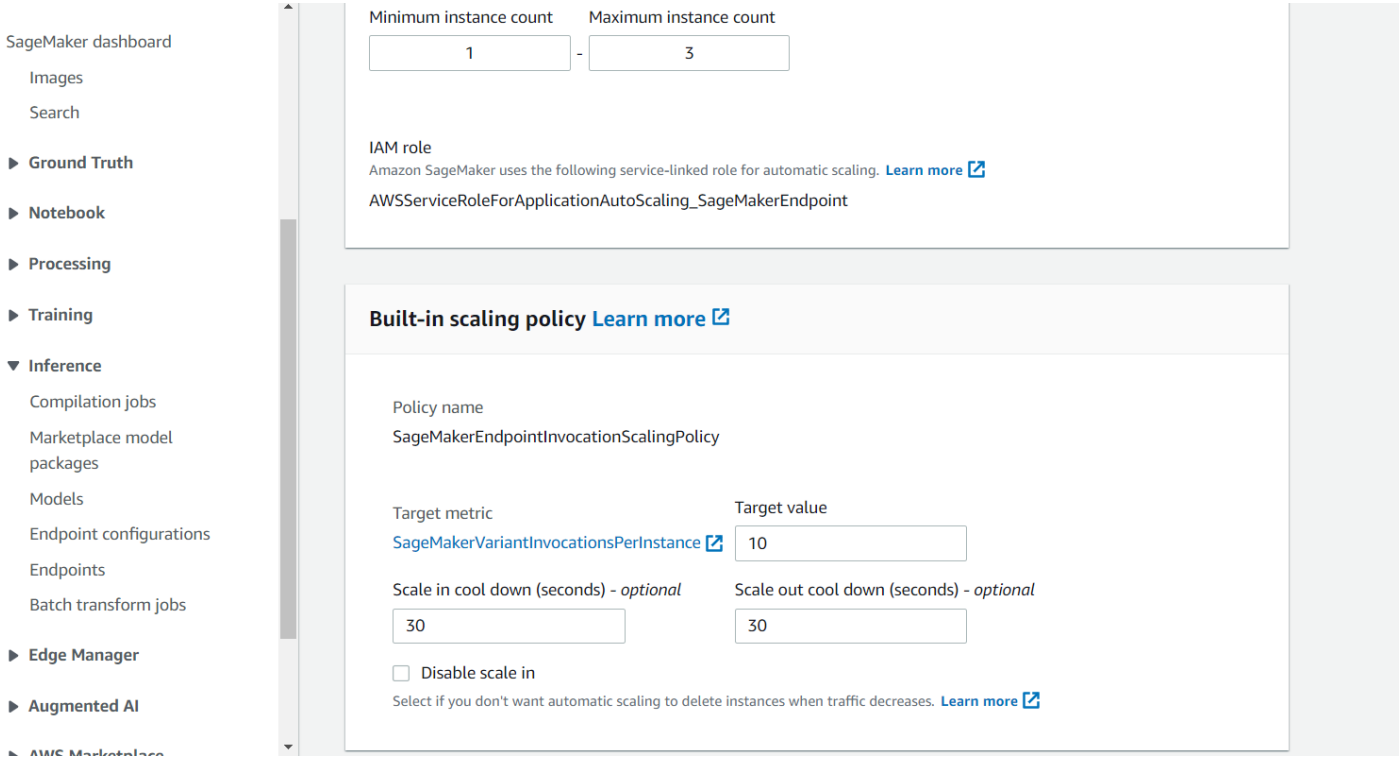

Figure 23. Endpoint Auto-scaling Config

- We have set the max instance count to 3 for Auto-scaling, as considering the current requirement, auto scaling on 3 instances with a scale-in and scale-out cool down time of 30 seconds should be a reasonably good config.

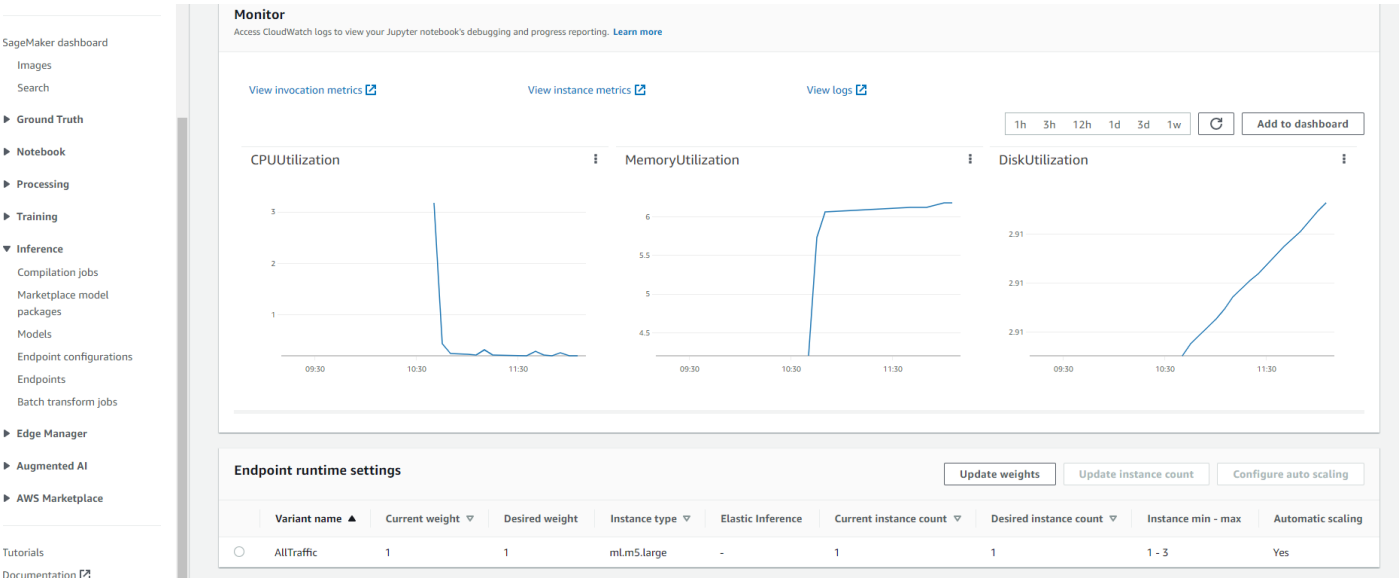

Figure 24. Endpoint Auto-scaling Metrics