

Operationalizing an AWS ML Project

-Pooja Singari

1. Initial Setup:

SageMaker notebook instance type selected was the **ml.t3.medium**. This is because I did not need a lot of resources just to run the notebook. It also has an hourly rate of only **\$0.0416**, and from SageMaker Studio, it was included as one of the instances that had fast launch speeds.

Kernel: Python3

Image: Data Science

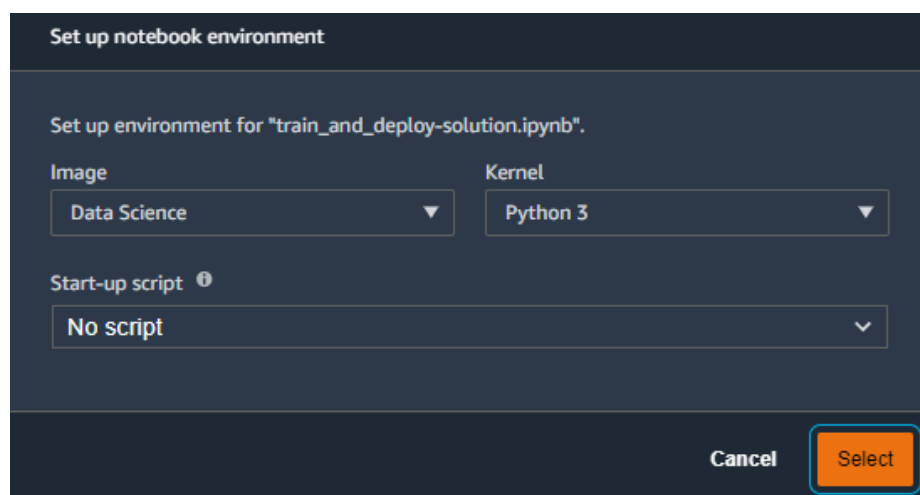


Figure 1 – Setting up Notebook environment in JupyterLab

The dog-breed dataset downloaded and unzipped after which it was successfully uploaded to s3 bucket via `aws s3 cp dogImages s3://my-classification-project/ --recursive` command

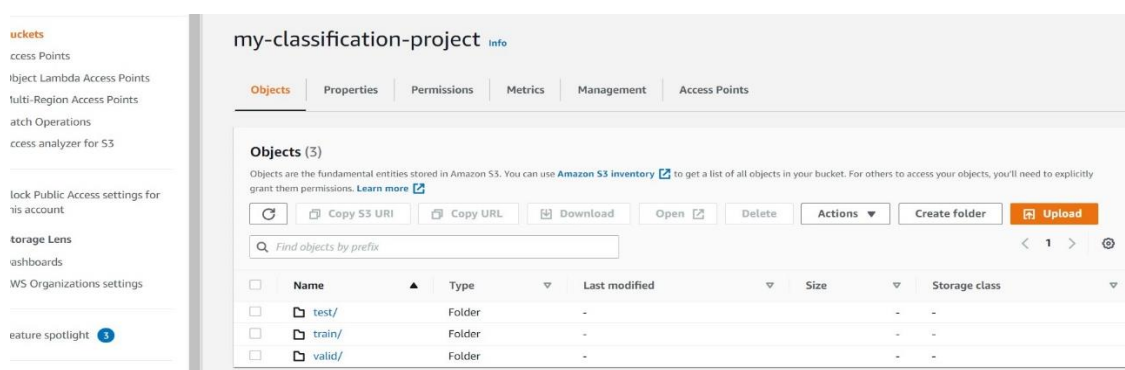
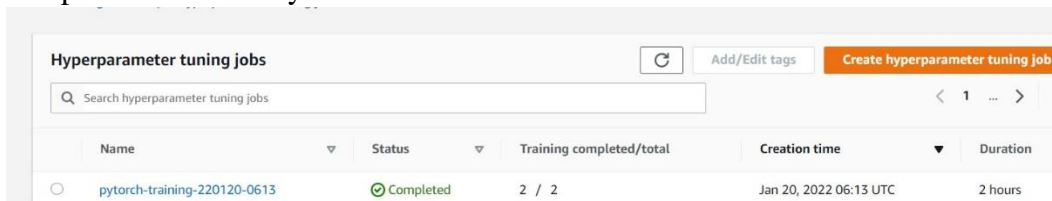


Figure 2 – S3 bucket holding the dog-breed dataset

2. SageMaker Training and Deployment:

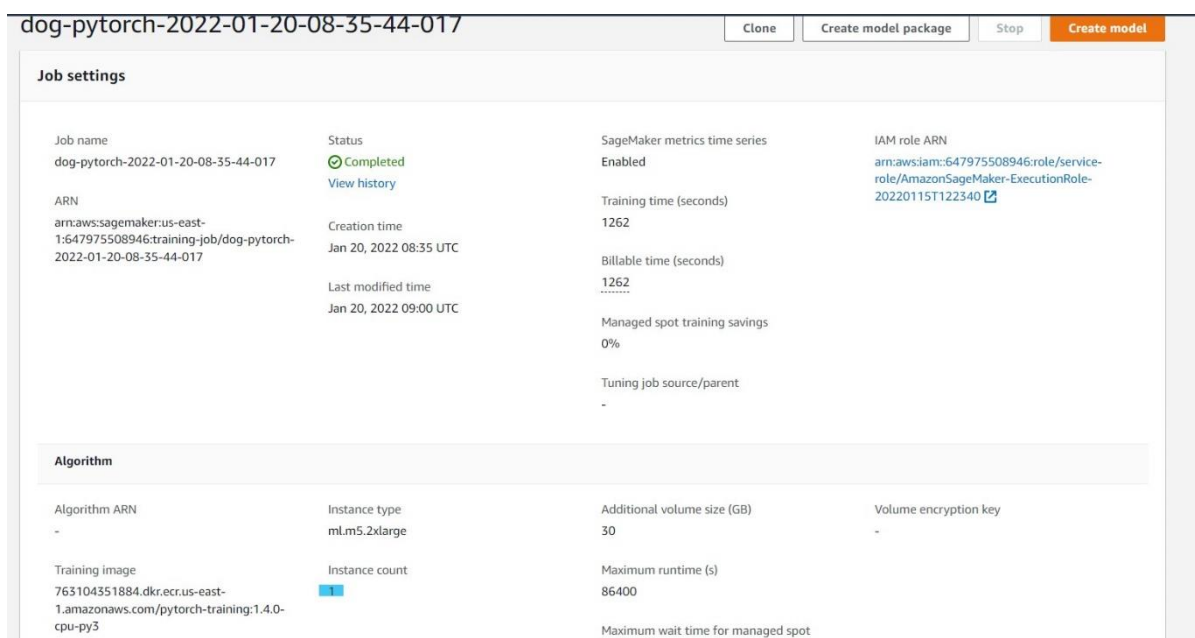
For hyperparameter tuning I used the same “ml.m5.xlarge” instance_type ,as it’s computational performance seemed sufficient for the current Image classification project. Hyperparameter tuning job (**max_jobs = 2, max_parallel_jobs = 2**) completed successfully:



Hyperparameter tuning jobs					
<input type="text" value="Search hyperparameter tuning jobs"/> 1					
Name	Status	Training completed/total	Creation time	Duration	
pytorch-training-220120-0613	Completed	2 / 2	Jan 20, 2022 06:13 UTC	2 hours	

Figure 3 – Hyperparameter Tuning Jobs

After completion of tuning jobs we had to create single and multi-instance training jobs which are demonstrated in the figures below



dog-pytorch-2022-01-20-08-35-44-017			
Clone Create model package Stop Create model			
Job settings			
Job name dog-pytorch-2022-01-20-08-35-44-017	Status Completed View history	SageMaker metrics time series Enabled	IAM role ARN arn:aws:iam::647975508946:role/service-role/AmazonSageMaker-ExecutionRole-20220115T122340
ARN arn:aws:sagemaker:us-east-1:647975508946:training-job/dog-pytorch-2022-01-20-08-35-44-017	Creation time Jan 20, 2022 08:35 UTC	Training time (seconds) 1262	
	Last modified time Jan 20, 2022 09:00 UTC	Billable time (seconds) 1262	
		Managed spot training savings 0%	
		Tuning job source/parent -	
Algorithm			
Algorithm ARN -	Instance type ml.m5.2xlarge	Additional volume size (GB) 30	Volume encryption key -
Training image 763104351884.dkr.ecr.us-east-1.amazonaws.com/pytorch-training:1.4.0-cpu-py3	Instance count 1	Maximum runtime (s) 86400	Maximum wait time for managed spot -

Figure 4 – Single Instance training job

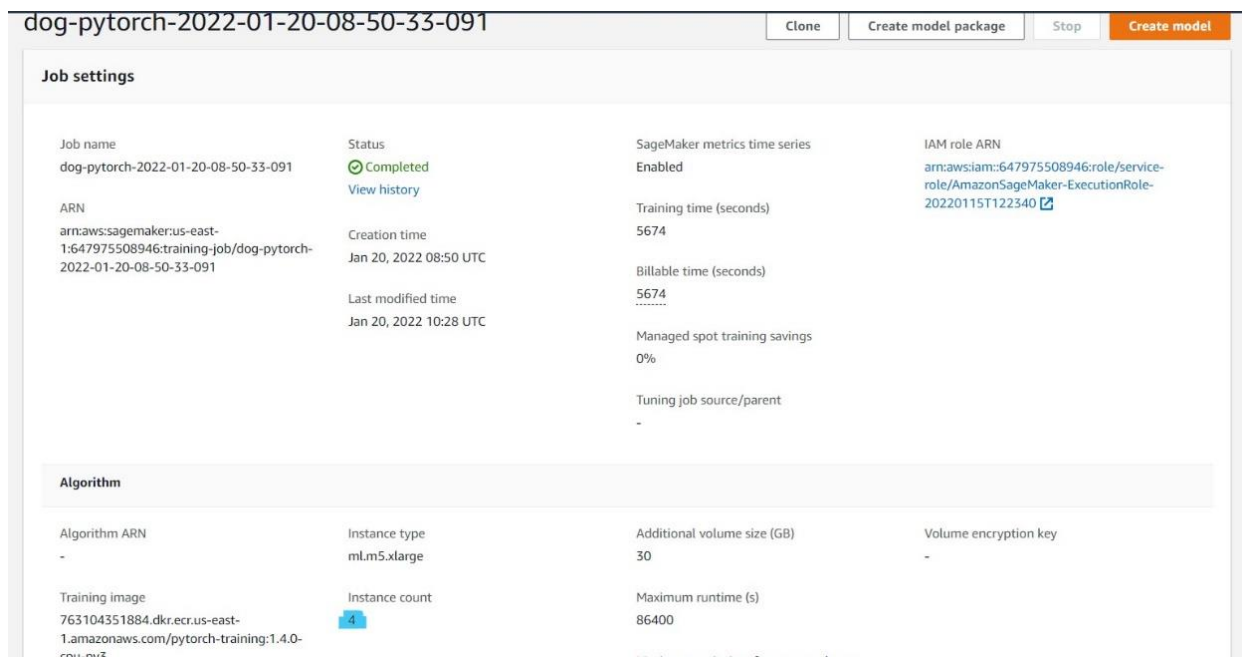


Figure 5– Multi-instance training job

Deployed Endpoints to invoke the model and receive the outputs:

- Single instance deployed endpoint: “pytorch-inference-2022-01-20-09-21-34-210”
- Multi instance deployed endpoint: “dog-pytorch-endpoint-multi-instance”

Name	ARN	Creation time	Status	Last updated
pytorch-inference-2022-01-20-09-21-34-210	arn:aws:sagemaker:us-east-1:647975508946:endpoint/pytorch-inference-2022-01-20-09-21-34-210	Jan 20, 2022 09:21 UTC	InService	Jan 20, 2022 09:24 UTC
dog-pytorch-endpoint-multi-instance	arn:aws:sagemaker:us-east-1:647975508946:endpoint/dog-pytorch-endpoint-multi-instance	Jan 20, 2022 09:00 UTC	InService	Jan 20, 2022 09:03 UTC

Figure 6 – Deployed Endpoints

3. EC2 Instance

- 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)

- 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.
- Moreover, writing code in EC2 is similar to a command-line terminal which does not have a user friendly interface as opposed to the well structured interface provided via SageMaker scripts.

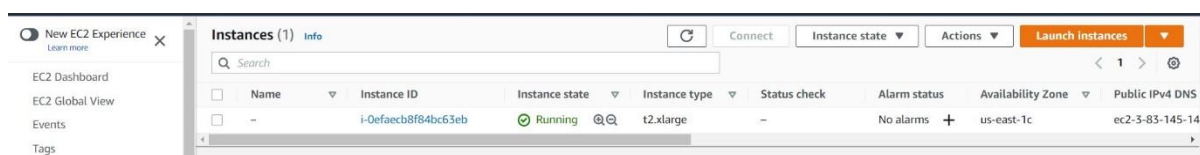


Figure 7 – EC2 Instance snapshot

```

=====
[root@ip-172-31-83-222 ~]# ls -ltr
total 1105568
drwxr-xr-x 5 root root      44 Mar 27  2017 dogImages
-rw-r--r-- 1 root root 1132023110 Apr  1  2017 dogImages.zip
-rw-r--r-- 1 root root    4860 Jan 20 10:33 solution5.py
-rw-r--r-- 1 root root    4846 Jan 20 10:35 solution6.py
-rw-r--r-- 1 root root    4849 Jan 20 10:36 solution7.py
-rw-r--r-- 1 root root    4847 Jan 20 10:37 solution8.py
-rw-r--r-- 1 root root    4846 Jan 20 10:49 sol3.py
-rw-r--r-- 1 root root    4849 Jan 20 10:50 sol4.py
-rw-r--r-- 1 root root    4842 Jan 20 10:52 sol5.py
-rw-r--r-- 1 root root    1920 Jan 20 10:54 sol6.py
-rw-r--r-- 1 root root    4943 Jan 20 11:10 sol8.py
-rw-r--r-- 1 root root    4809 Jan 20 11:16 sol1.py
drwxr-xr-x 2 root root      23 Jan 20 11:32 TrainedModels
[root@ip-172-31-83-222 ~]# cd TrainedModels
[root@ip-172-31-83-222 TrainedModels]# ls
model.pth
[root@ip-172-31-83-222 TrainedModels]#

```

Figure 8– EC2 Training saved model.pth

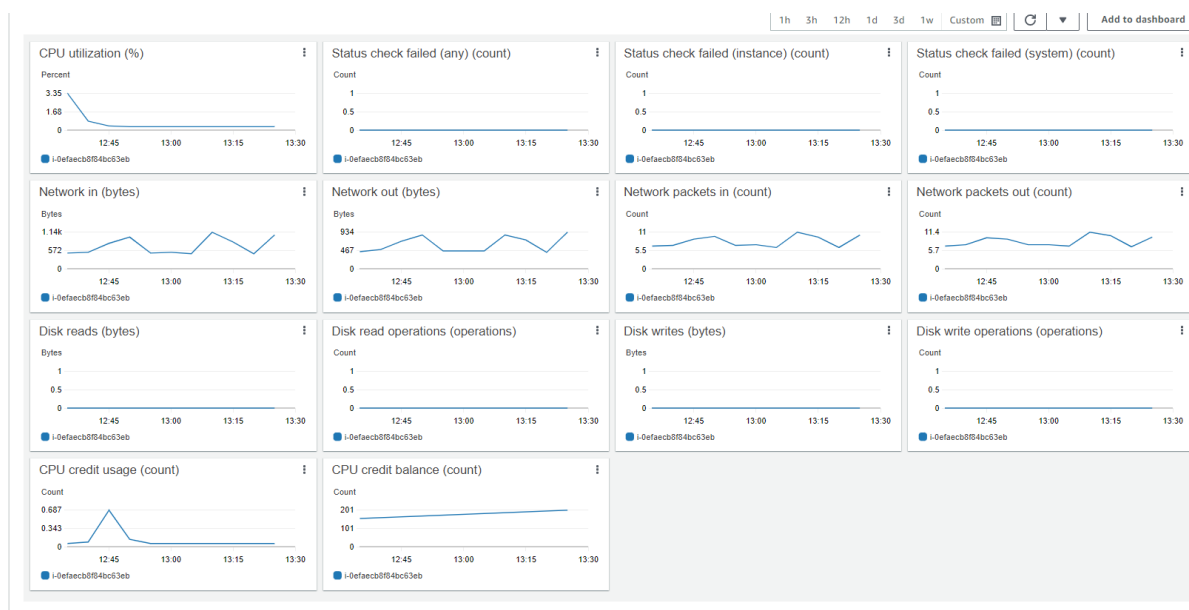


Figure 9 – EC2 Instance Resource Monitoring

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: “dog-pytorch-endpoint-multi-instance”
- We created the lambda function with the corresponding changes to invoke the endpoint:

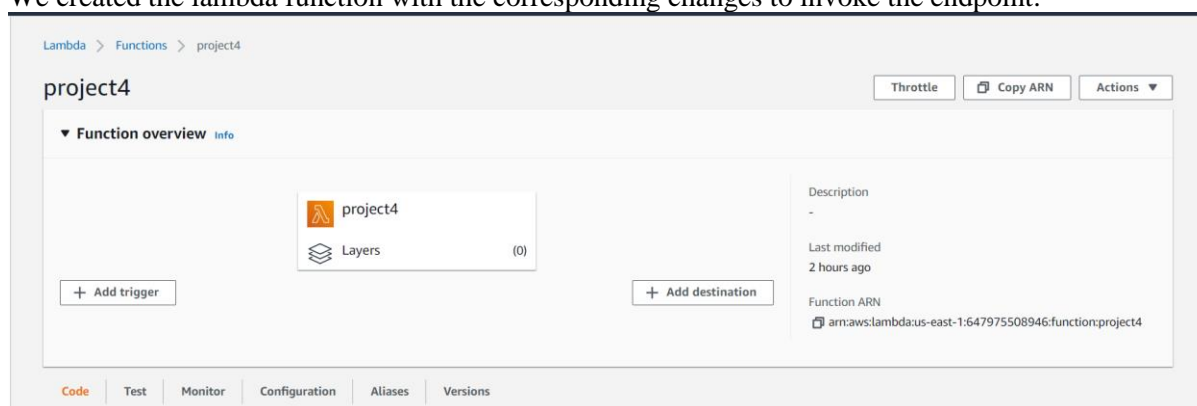


Figure 10 – Lambda Function

5. Security and Testing

The lambda function is written to accept data in a JSON object and invoke an endpoint that returns a response with a prediction. The function then packages information such as status code and body into a dictionary and returns it as the response of the function.

- We had created a new role for this lambda functions with “**SageMakerFullAccess**”(without which it leads to **Invocation Error**)
- We had to use the below test event to test our Lambda function:

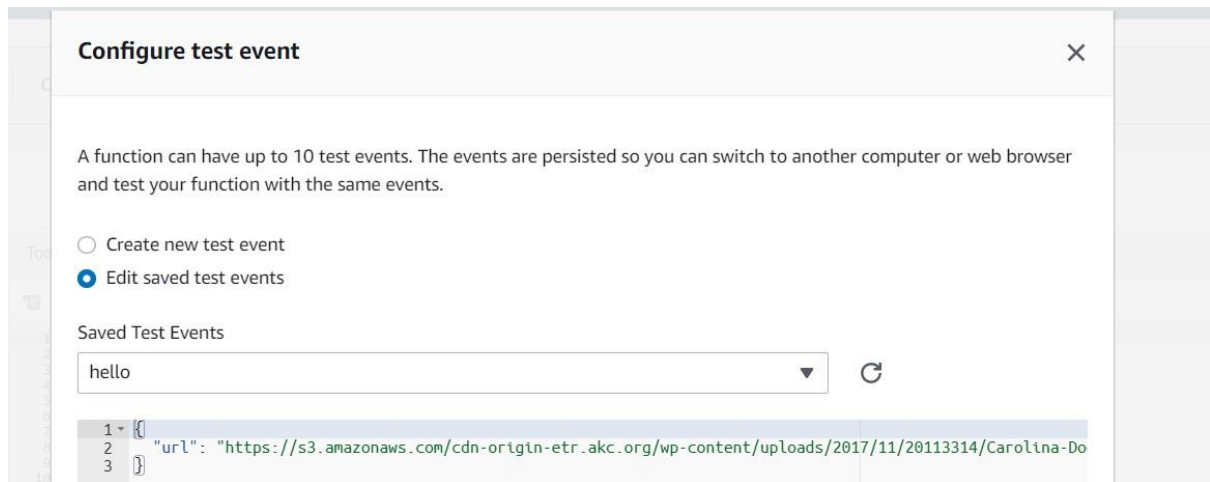


Figure 11 – Lambda Function Test Event

Response from Lambda Function:

```
{
  "statusCode": 200,
  "headers": {
    "Content-Type": "text/plain",
    "Access-Control-Allow-Origin": "*"
  },
  "type-result": "<class 'str'>",
  "Content-Type-In": "LambdaContext([aws_request_id=c6dae78f-40b0-46ca-a44a-58d7b1cdca4b,log_group_name=/aws/lambda/project4,log_stream_name=2022/01/20/[$LATEST]cac131cc588742a0a203734c6298d530,function_name=project4,memory_limit_in_mb=128,function_version=$LATEST,invoked_function_arn=arn:aws:lambda:us-east-1:647975508946:function:project4,client_context=None,identity=CognitoIdentity([cognito_identity_id=None,cognito_identity_pool_id=None]))",
  "body": "[[-8.509687423706055, -7.77571964263916, -2.675638198852539, -2.3557212352752686, -6.151639938354492, -7.035409927368164, -3.780886173248291, -2.774186134338379, -8.81319522857666, -1.3065375089645386, 1.6190638542175293, -5.865751266479492, -3.352370023727417, 0.33569395542144775, -7.82021427154541, -3.9492571353912354, -8.446880340576172, -2.839498996734619, -4.65122229003906, 1.0388201475143433, -7.5899882316589355, -3.5086326599121094, -9.712128639221191, -8.360410690307617, -8.240933418273926, -13.016434669494629, -5.23638391494751, -7.427854537963867, -6.946070194244385, -2.703977108001709, -7.135708332061768, -5.68403434753418, -11.949724197387695, -4.561254024505615, -11.778522491455078, -8.986464500427246, -7.212772369384766, -3.3191754817962646, -1.1046655178070068, -
```

```

6.461777687072754, -5.033671855926514, -2.7278974056243896, -0.9936132431030273, -
6.044423580169678, -1.883324384689331, -9.649934768676758, -3.4763667583465576, -
1.0421169996261597, -2.717921257019043, -3.589296817779541, -5.795350551605225, -
7.381248474121094, -8.431464195251465, -7.530313491821289, -8.174006462097168, -
3.8078136444091797, -6.326535701751709, -8.39573860168457, -2.9995415210723877, -
5.429052829742432, -10.660608291625977, -9.486539840698242, -11.510260581970215, -
6.826721668243408, -6.756300449371338, -14.48521614074707, -1.6789501905441284, -
9.15797233581543, -2.646780490875244, -1.1237974166870117, -0.6023721694946289, -
6.902181148529053, -5.664377689361572, -8.664715766906738, -7.221043586730957, -
3.847078323364258, -13.318488121032715, -4.377753257751465, -4.632699489593506, -
6.257696151733398, -1.2542197704315186, -10.409688949584961, -2.2787129878997803, -
1.0235614776611328, -8.419547080993652, -8.682510375976562, -2.885606527328491, -
10.519001960754395, -4.825105667114258, -2.1089701652526855, -8.725203514099121, -
5.307856559753418, -5.575226783752441, -9.437766075134277, -5.222029685974121, -
4.098916053771973, -6.385500431060791, -7.366503715515137, -7.367201805114746, -
6.432024002075195, -7.426565170288086, -1.6843739748001099, -7.017165184020996, -
5.080165863037109, -8.466792106628418, -14.218036651611328, -4.277678966522217, -
0.39628297090530396, -1.378735899925232, -3.197972536087036, -2.170497179031372, -
3.1416969299316406, -8.856181144714355, -6.239938259124756, -7.6649956703186035, -
1.4355008602142334, -6.978602409362793, -1.5712144374847412, -4.779670715332031, -
1.2244170904159546, -3.855419397354126, -6.736294746398926, -5.722675800323486, -
6.578609943389893, -12.074214935302734, -5.719165802001953, -4.999317169189453, -
2.3081724643707275, -5.571413516998291, -9.936185836791992, -8.904071807861328, -
4.077090263366699, -7.278857231140137]]]"
}

```

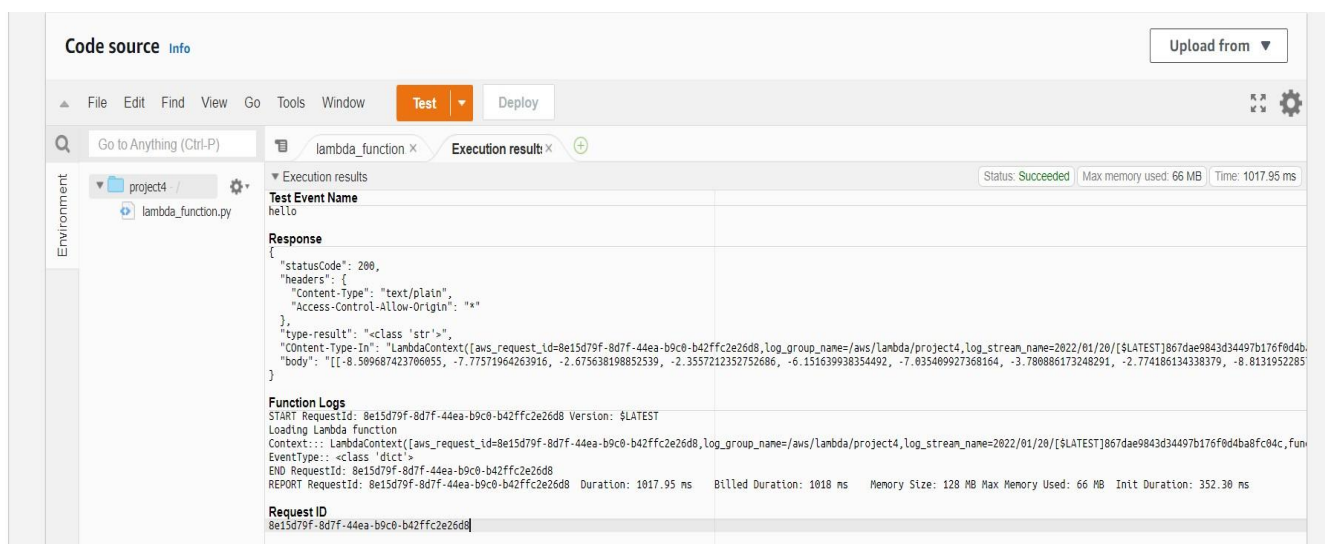


Figure 12 – Lambda Function (200-Status Code) Response

Workspace Security: At moment this project was done, I'd say that my workspace security was good as I didn't have any roles that were not being used by the services I needed, i.e all inactive roles had been deleted. As for vulnerability, it would be notable to mention that having some roles being attached with any **FullAccess** policies may lead to some vulnerability especially when the roles probably needed access to a single component of a service such as Endpoints in SageMaker for a Lambda function.

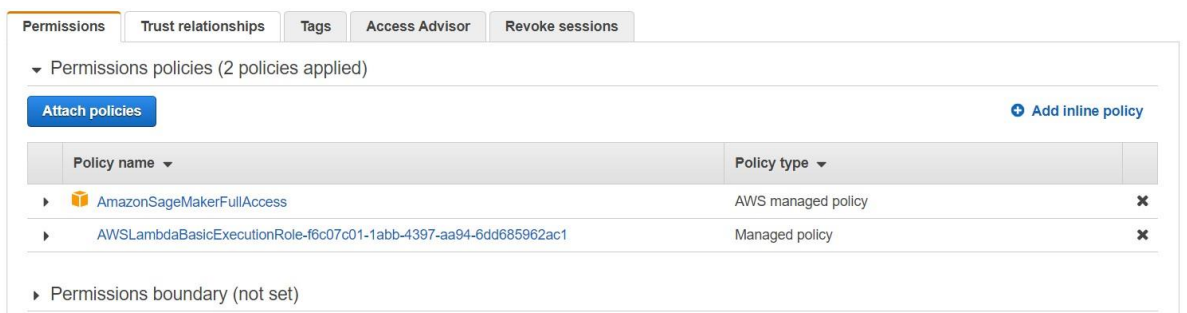


Figure 13- Permission Policies

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 seems to be necessary and also 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 absolutely necessary for currently active projects, are the only roles that are in active state to prevent unauthorized accesses.

6. Concurrency and AutoScaling

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

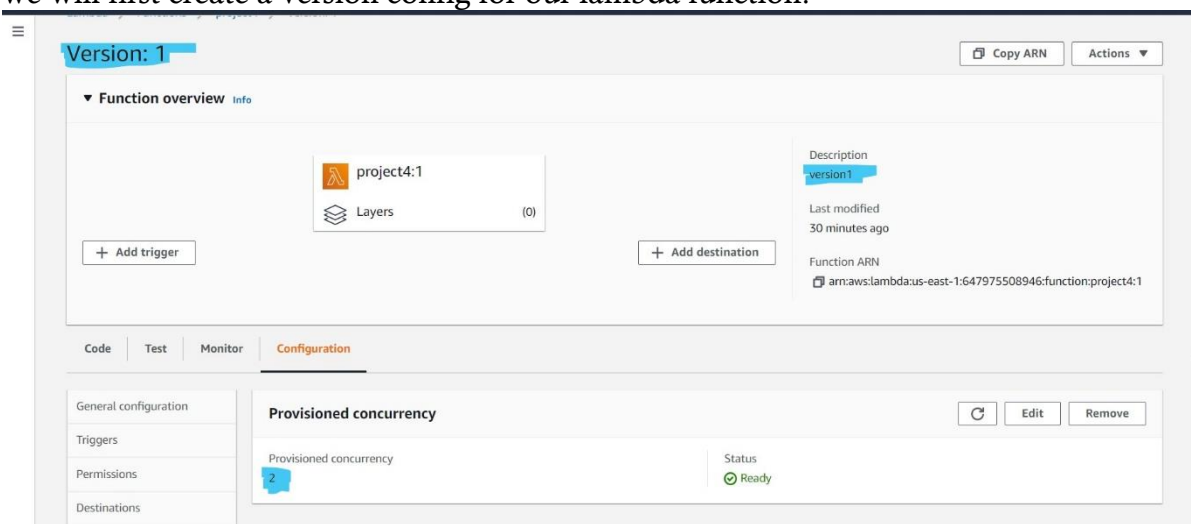


Figure 14- Version(1): Provisioned Concurrency

- I have set up **Provisioned Concurrency of 2** because it creates instances that are always on and can reply to all traffic without requiring a wait for start-up times.

- Provisioned concurrency has a higher cost than reserved concurrency but since it turns on instances that are always ready to respond to traffic, it can achieve low latency even in very high traffic scenarios.

Autoscaling: 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 configuration.

Built-in scaling policy [Learn more](#)

Policy name
SageMakerEndpointInvocationScalingPolicy

Target metric
[SageMakerVariantInvocationsPerInstance](#)

Target value
30

Scale in cool down (seconds) - optional
300

Scale out cool down (seconds) - optional
300

☐ Disable scale in

Select if you don't want automatic scaling to delete instances when traffic decreases. [Learn more](#)

Figure 15- AutoScale Configuration

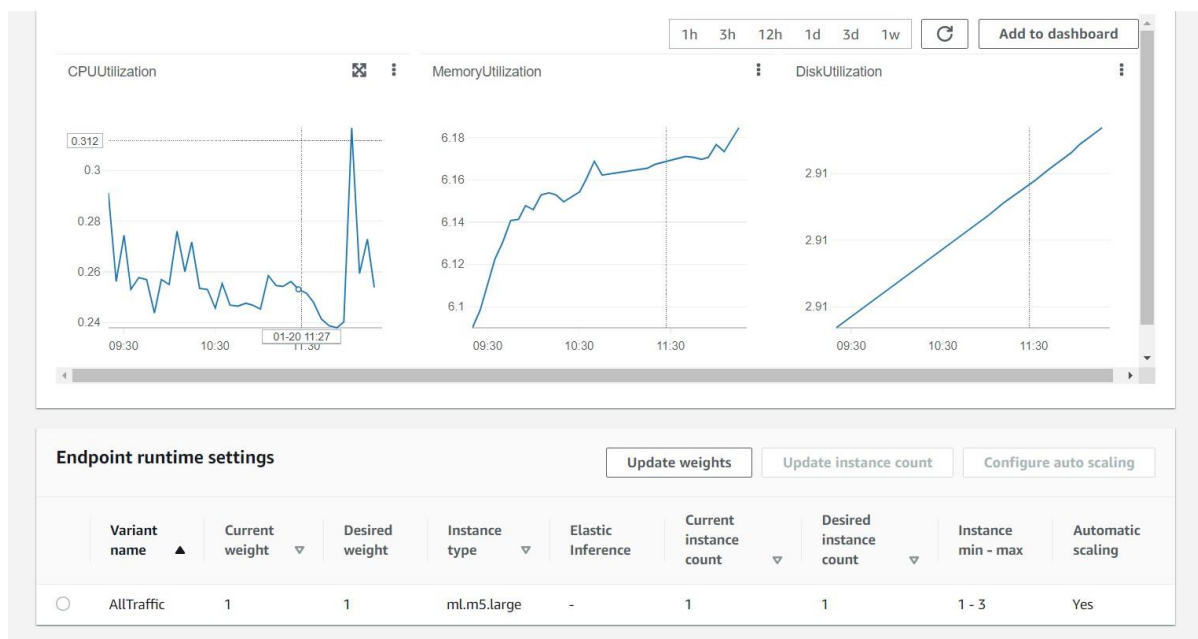


Figure 16- Endpoint AutoScaling Metrics