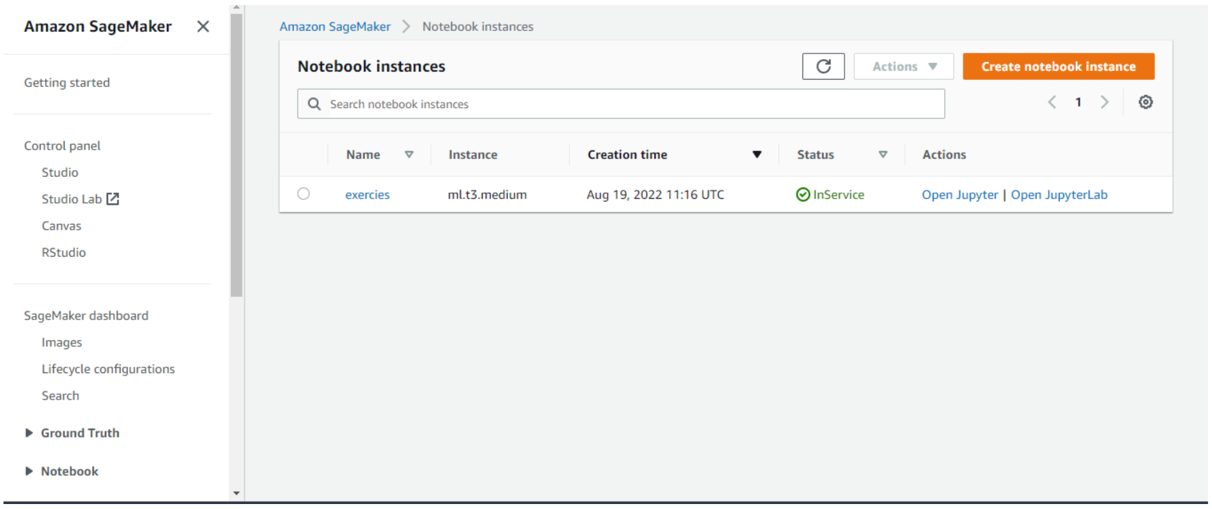
Project Report

Setup, training, and deployment on SageMaker

Notebook Instance

The instance I used for my Jupyter notebook was ml.t3.medium. I selected this instance type since it is affordable and qualifies for the AWS free tier while being powerful enough to run the programs I intended to run on it. I was aware that while model tuning, training, and deployment would take place on other instances, I would be installing packages, downloading, unzipping, and uploading images on this instance. Additionally, I needed ml.t3.medium's quick launch feature because I expected to shut down and restart the notebook several times throughout the project.



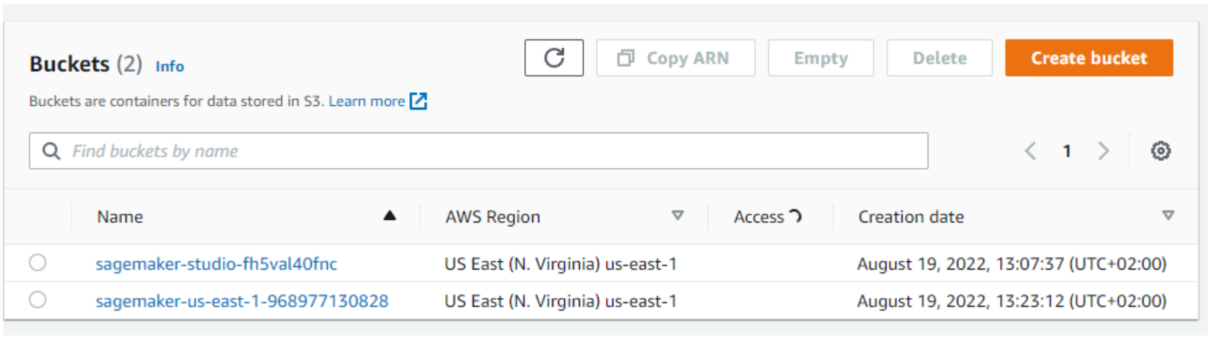
Notebook instances

Name	Instance	Creation time	Status	Actions
exercies	ml.t3.medium	Aug 19, 2022 11:16 UTC	InService	Open Jupyter | Open JupyterLab

fig1 . Notebook instance launch

S3 Bucket

I used the default sagemaker S3 bucket to copy the data into (sagemaker-us-east-1-968977130828) . I changed the references in train_and_deploy-solution.ipynb to point to this bucket, and I uploaded the unzipped dogImages dataset to a file called : s3://sagemaker-us-east-1-968977130828/dogImages/


Buckets (2) Info

Buckets are containers for data stored in S3. Learn more

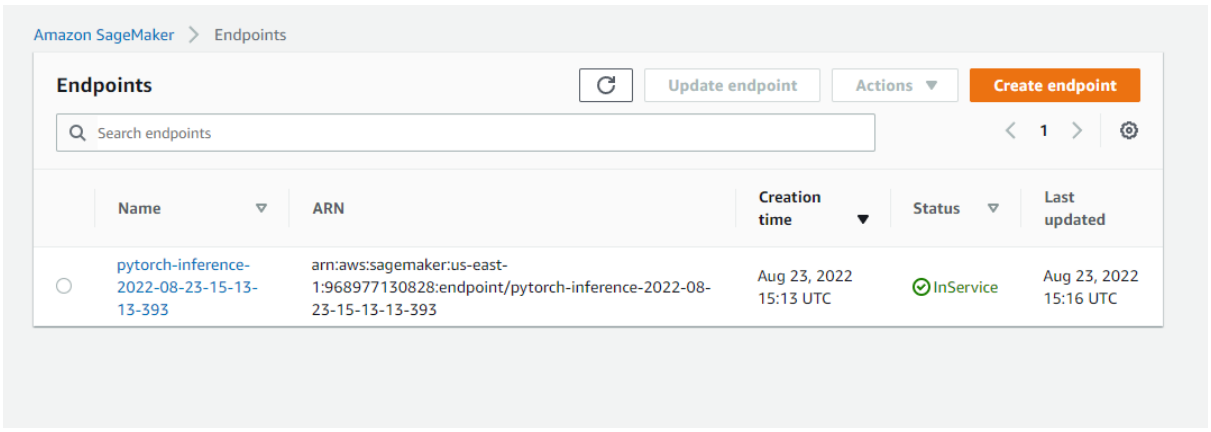
Name	AWS Region	Access	Creation date
sagemaker-studio-fh5val40fnc	US East (N. Virginia) us-east-1		August 19, 2022, 13:07:37 (UTC+02:00)
sagemaker-us-east-1-968977130828	US East (N. Virginia) us-east-1		August 19, 2022, 13:23:12 (UTC+02:00)

fig 2. S3 bucket

Training and Deployment

For hyperparameter tuning I used an ml.m5.xlarge instance with 1 jobs. This instance is part of the free tier for training in SageMaker, and from experience I knew this would take about 25-30 min, which was acceptable. The best hyperparameters were a batch size of 64 and a learning rate of approximately 0.005831981479799012.

I used an ml.m5.xlarge instance to train the estimator, using the best hyperparameters from my tuning job. Once it was trained, I deployed the model to an endpoint on ml.m5.xlarge instance. I chose this type of instance because it wouldn't need to handle very many or very large inference queries. The endpoint's name shown in figure 3 .

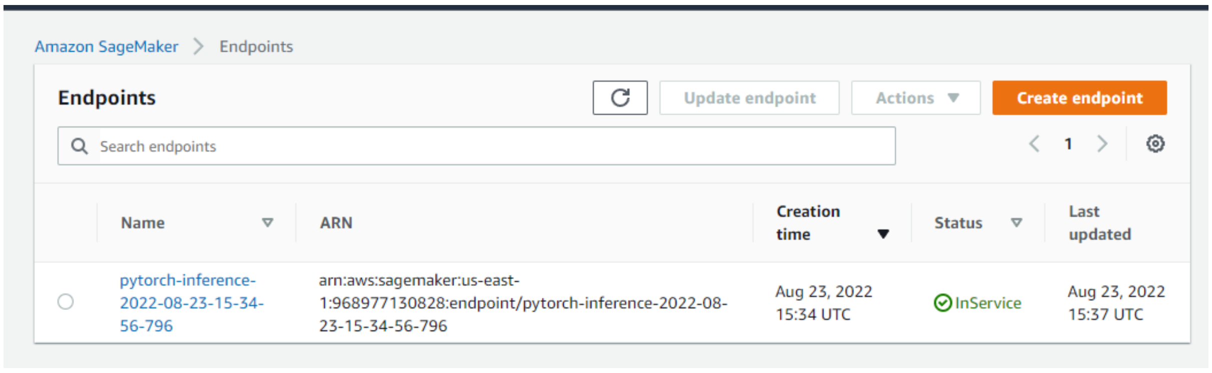

Amazon SageMaker > Endpoints

Endpoints

Name	ARN	Creation time	Status	Last updated
pytorch-inference-2022-08-23-15-13-13-393	arn:aws:sagemaker:us-east-1:968977130828:endpoint/pytorch-inference-2022-08-23-15-13-13-393	Aug 23, 2022 15:13 UTC	InService	Aug 23, 2022 15:16 UTC

fig 3 . single instance endpoint

After running the rest of the notebook, to confirm that the endpoint could perform inference, I went back and used 2 instances of type ml.m5.xlarge to perform multi-instance training. Once it was trained, I deployed to an endpoint on ml.m5.large instance. The endpoint's name shown in figure 4



Amazon SageMaker > Endpoints

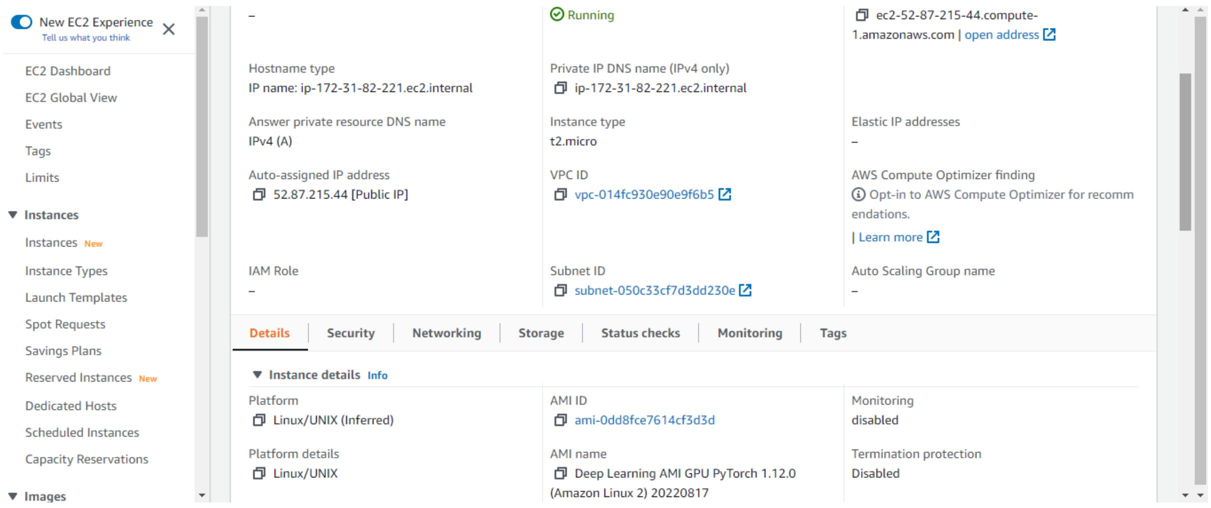
Endpoints

	Name	ARN	Creation time	Status	Last updated
○	pytorch-inference-2022-08-23-15-34-56-796	arn:aws:sagemaker:us-east-1:968977130828:endpoint/pytorch-inference-2022-08-23-15-34-56-796	Aug 23, 2022 15:34 UTC	InService	Aug 23, 2022 15:37 UTC

fig 4 . multi instance endpoint

EC2 Instance

I selected the m5.xlarge instance of the Deep Learning AMI GPU Pytorch. I was confident that this would be potent enough to finish model training quickly and wouldn't deplete my account if I had to keep it running for a few hours to troubleshoot.



–

Running

ec2-52-87-215-44.compute-1.amazonaws.com | open address

Hostname type
IP name: ip-172-31-82-221.ec2.internal

Private IP DNS name (IPv4 only)
ip-172-31-82-221.ec2.internal

Answer private resource DNS name
IPv4 (A)

Instance type
t2.micro

Elastic IP addresses
–

Auto-assigned IP address
52.87.215.44 [Public IP]

VPC ID
vpc-014fc930e90e9f6b5

AWS Compute Optimizer finding
Opt-in to AWS Compute Optimizer for recommendations.
| Learn more

IAM Role
–

Subnet ID
subnet-050c33cf7d3dd230e

Auto Scaling Group name
–

Instance details Info

Platform
Linux/UNIX (Inferred)

AMI ID
ami-0dd8fce7614cf3d3d

Monitoring
disabled

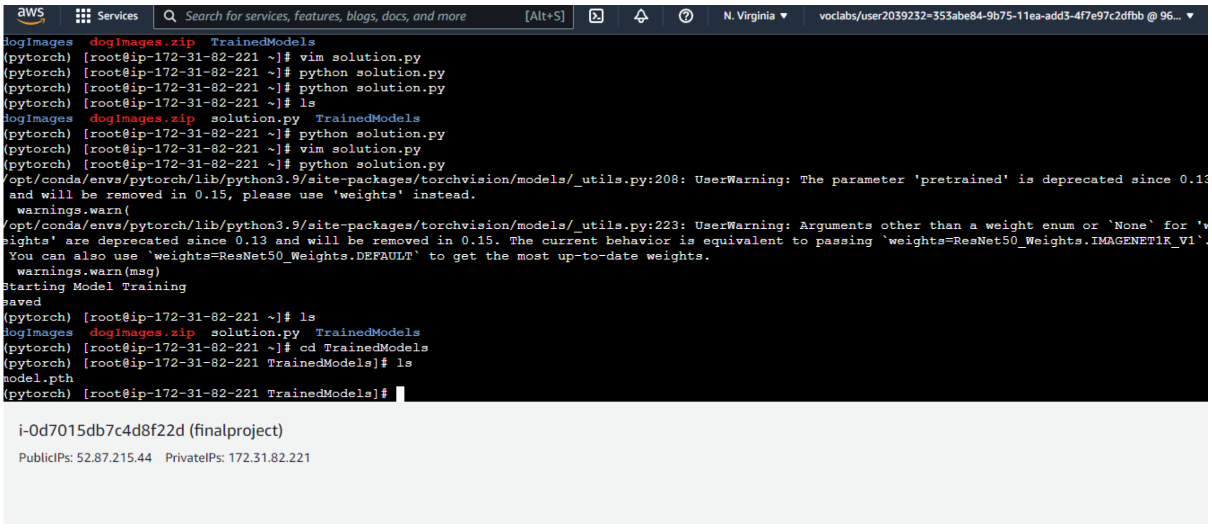
Platform details
Linux/UNIX

AMI name
Deep Learning AMI GPU PyTorch 1.12.0 (Amazon Linux 2) 20220817

Termination protection
Disabled

fig 5 .EC2 instance

I started the instance, downloaded the dataset, trained the model, and then saved it in the directory for trained models.


aws
Services
Search for services, features, blogs, docs, and more
[Alt+S]
N. Virginia
voclabs/user2039232=353abe84-9b75-11ea-add3-4f7e97c2dfbb @ 96...

```
dogImages  dogImages.zip  TrainedModels
(pytorch) [root@ip-172-31-82-221 ~]# vim solution.py
(pytorch) [root@ip-172-31-82-221 ~]# python solution.py
(pytorch) [root@ip-172-31-82-221 ~]# python solution.py
(pytorch) [root@ip-172-31-82-221 ~]# ls
dogImages  dogImages.zip  solution.py  TrainedModels
(pytorch) [root@ip-172-31-82-221 ~]# python solution.py
(pytorch) [root@ip-172-31-82-221 ~]# vim solution.py
(pytorch) [root@ip-172-31-82-221 ~]# python solution.py
/opt/conda/envs/pytorch/lib/python3.9/site-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since 0.13
 and will be removed in 0.15, please use 'weights' instead.
  warnings.warn(
/opt/conda/envs/pytorch/lib/python3.9/site-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `None` for 'w
eights' are deprecated since 0.13 and will be removed in 0.15. The current behavior is equivalent to passing `weights=ResNet50_Weights.IMAGENET1K_V1`.
 You can also use `weights=ResNet50_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
Starting Model Training
saved
(pytorch) [root@ip-172-31-82-221 ~]# ls
dogImages  dogImages.zip  solution.py  TrainedModels
(pytorch) [root@ip-172-31-82-221 ~]# cd TrainedModels
(pytorch) [root@ip-172-31-82-221 TrainedModels]# ls
model.pth
(pytorch) [root@ip-172-31-82-221 TrainedModels]#
```

i-0d7015db7c4d8f22d (finalproject)

PublicIPs: 52.87.215.44   PrivateIPs: 172.31.82.221

fig 6.EC2 instance training

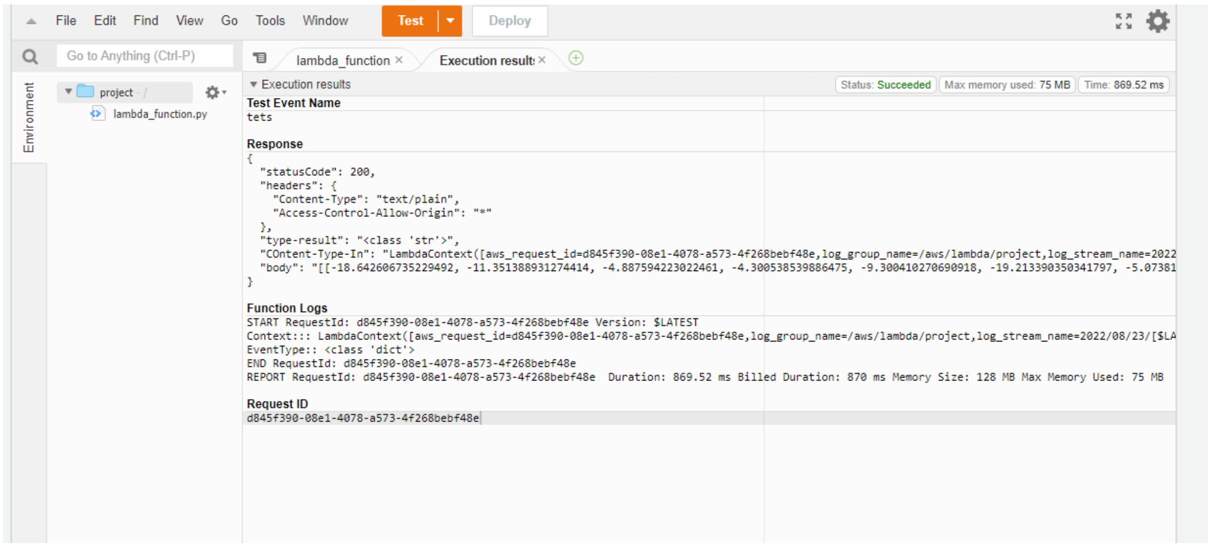
EC2 VS Sagemaker

With a few significant exceptions, the code in ec2train1.py is comparable to train and deploy-solution.ipynb and hpo.py. First off, all training is done locally by the EC2 script. As opposed to SageMaker, where the estimators are fitted on distinct instances from the notebook where the code is performed, and the training data, model, and output are all kept on S3. This contributes to the EC2 script's lack of a parser or main function. The learning rate, batch size, training, model, and output folders are all sent along to the main function via the SageMaker parser.

All of these are merely provided in the script for EC2. Only because the dataset is local, the training is local, and the model can be saved locally, is this possible. Last but not least, from EC2, the trained model cannot be deployed straight to an endpoint. We only know how to deploy endpoints from models within SageMaker, therefore before we could install an endpoint to conduct real-time inference, we would need to import the model from the EC2 instance into SageMaker.

Lambda Function Setup

After attaching the appropriate permissions to the Lambda function role, I ran a successful test of my lambda function as shown in figure 7 .



Execution results

Test Event Name
```
tets
```

Response
```
{
  "statusCode": 200,
  "headers": {
    "Content-Type": "text/plain",
    "Access-Control-Allow-Origin": "*"
  },
  "type-result": "<class 'str'>",
  "COntent-Type-In": "LambdaContext([aws_request_id=d845f390-08e1-4078-a573-4f268bebf48e,log_group_name=/aws/lambda/project,log_stream_name=2022
  "body": "[[-18.642606735229492, -11.351388931274414, -4.887594223022461, -4.300538539886475, -9.300410270690918, -19.213390350341797, -5.07381
}
```

Function Logs
```
START RequestId: d845f390-08e1-4078-a573-4f268bebf48e Version: $LATEST
Context::: LambdaContext([aws_request_id=d845f390-08e1-4078-a573-4f268bebf48e,log_group_name=/aws/lambda/project,log_stream_name=2022/08/23/[$LA
EventType:: <class 'dict'>
END RequestId: d845f390-08e1-4078-a573-4f268bebf48e
REPORT RequestId: d845f390-08e1-4078-a573-4f268bebf48e  Duration: 869.52 ms Billed Duration: 870 ms Memory Size: 128 MB Max Memory Used: 75 MB
```

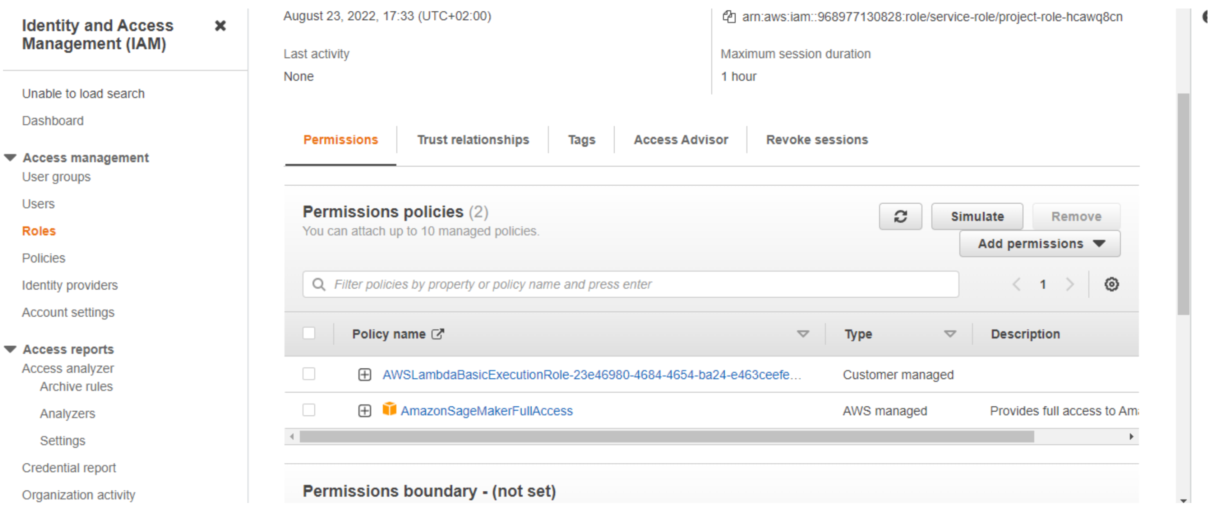
Request ID
```
d845f390-08e1-4078-a573-4f268bebf48e
```

fig 7.Successful Lambda function test

You can check the output also provided on my repo in the text file .

Security

I've given SageMaker's Lambda function complete access. This might be used, among other things, to list the secrets in Secrets Manager, acquire metric data from CloudWatch, establish or delete VPC endpoints in EC2, or get or delete assets from S3. This recommends that we should be cautious about who is first given permission to build Lambda functions, monitor the Access Advisor for roles frequently to see which services are being accessed and when, and withdraw rights if we notice anything suspicious. Around AWS accounts, we should utilize fundamental operational security procedures like multi-factor authentication, privileged account workstations, and routine active-user/role audits.



August 23, 2022, 17:33 (UTC+02:00)

arn:aws:iam::968977130828:role/service-role/project-role-hcawq8cn

Last activity: None

Maximum session duration: 1 hour

Permissions policies (2)
You can attach up to 10 managed policies.

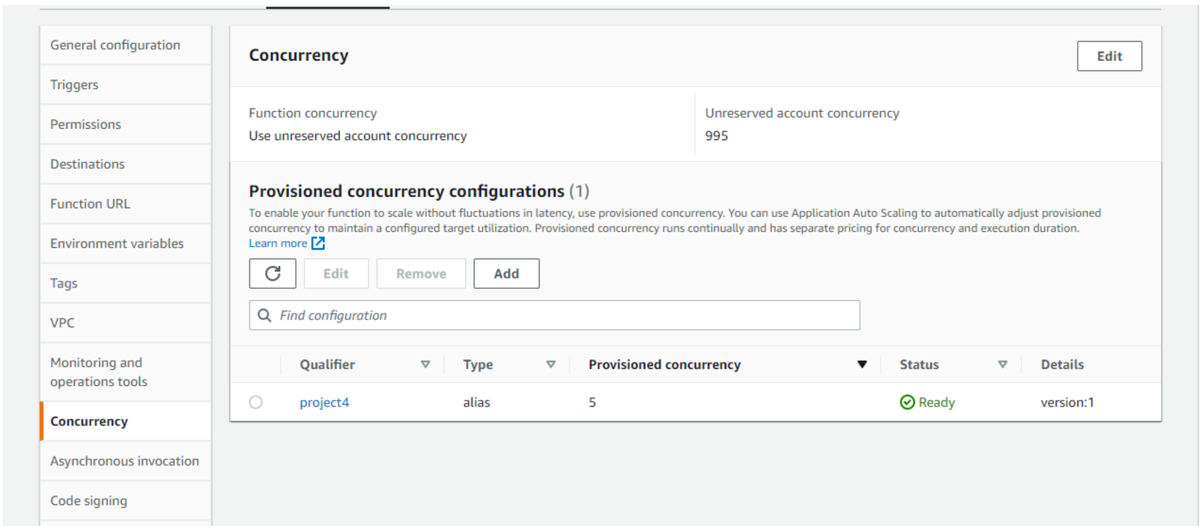
Policy name	Type	Description
AWSLambdaBasicExecutionRole-23e46980-4684-4654-ba24-e463ceefe...	Customer managed	
AmazonSageMakerFullAccess	AWS managed	Provides full access to Am

Permissions boundary - (not set)

fig 8.Permission for lambda function

Concurrency

My Lambda function was configured, and I selected 5 provided concurrency. To show my understanding, I added a small amount of each type of concurrency, and they would be more than enough for me to use the Lambda function as shown in figure 9 .



General configuration
Triggers
Permissions
Destinations
Function URL
Environment variables
Tags
VPC
Monitoring and operations tools
Concurrency
Asynchronous invocation
Code signing

Concurrency Edit

Function concurrency
Use unreserved account concurrency

Unreserved account concurrency
995

Provisioned concurrency configurations (1)

To enable your function to scale without fluctuations in latency, use provisioned concurrency. You can use Application Auto Scaling to automatically adjust provisioned concurrency to maintain a configured target utilization. Provisioned concurrency runs continually and has separate pricing for concurrency and execution duration.
Learn more

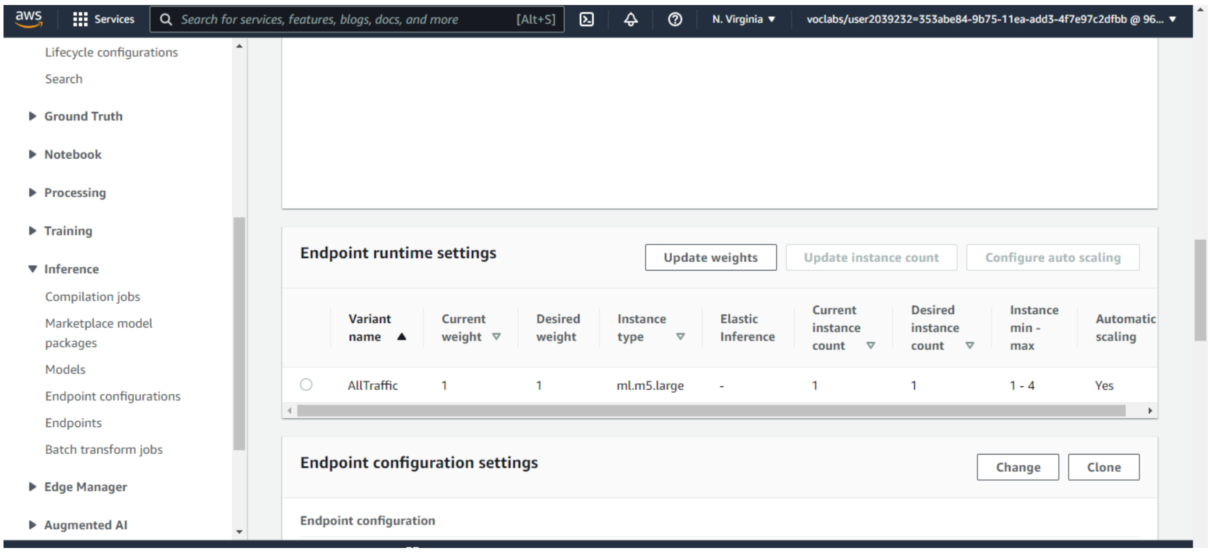
Edit | Remove | Add

Find configuration

	Qualifier	Type	Provisioned concurrency	Status	Details
○	project4	alias	5	✓ Ready	version:1

fig 9 Concurrency of lambda function

For my deployed endpoint, I similarly set up auto-scaling with a minimum instance count of 1, a maximum instance count of 4, and scale-in and scale-out cool down intervals of 30 seconds. Those settings worked perfectly for the activity we completed earlier in the course, so both the cost and performance would be fine as shown in figure 10 .



aws Services | Search for services, features, blogs, docs, and more [Alt+S] | N. Virginia | voclabs/user2039232=353abe84-9b75-11ea-add3-4f7e97c2dfbb @ 96...

Lifecycle configurations
Search
Ground Truth
Notebook
Processing
Training
Inference
Compilation jobs
Marketplace model packages
Models
Endpoint configurations
Endpoints
Batch transform jobs
Edge Manager
Augmented AI

Endpoint runtime settings Update weights | Update instance count | Configure auto scaling

	Variant name	Current weight	Desired weight	Instance type	Elastic Inference	Current instance count	Desired instance count	Instance min - max	Automatic scaling
○	AllTraffic	1	1	ml.m5.large	-	1	1	1 - 4	Yes

Endpoint configuration settings Change | Clone

Endpoint configuration

fig 10.Autoscalling for my endpoint