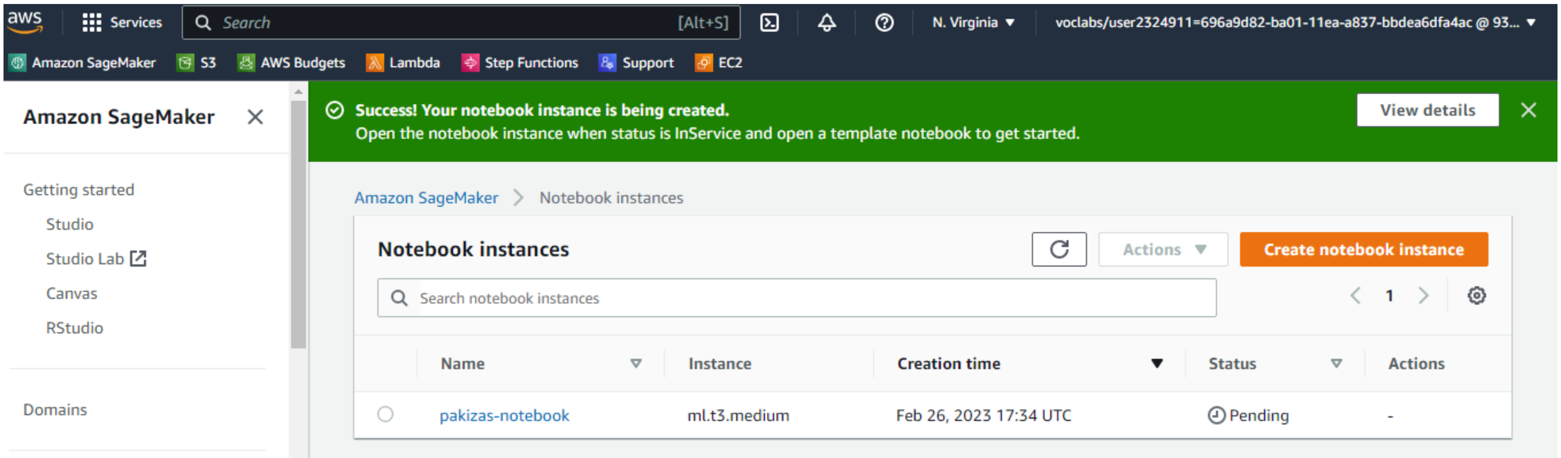
Operationalizing an AWS ML Project

Step 1: Training and Deployment on Sagemaker

1) Created a new notebook instance



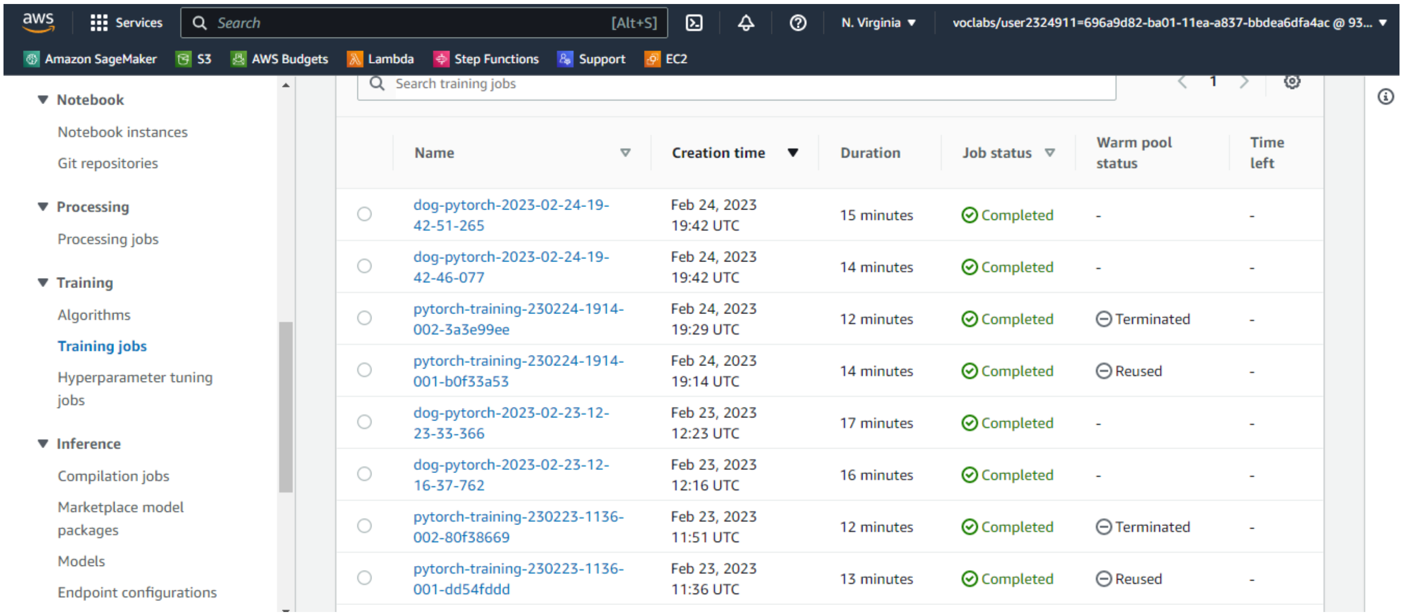
Success! Your notebook instance is being created.
Open the notebook instance when status is InService and open a template notebook to get started.

Amazon SageMaker > Notebook instances

Notebook instances

	Name	Instance	Creation time	Status	Actions
○	pakizas-notebook	ml.t3.medium	Feb 26, 2023 17:34 UTC	Pending	-

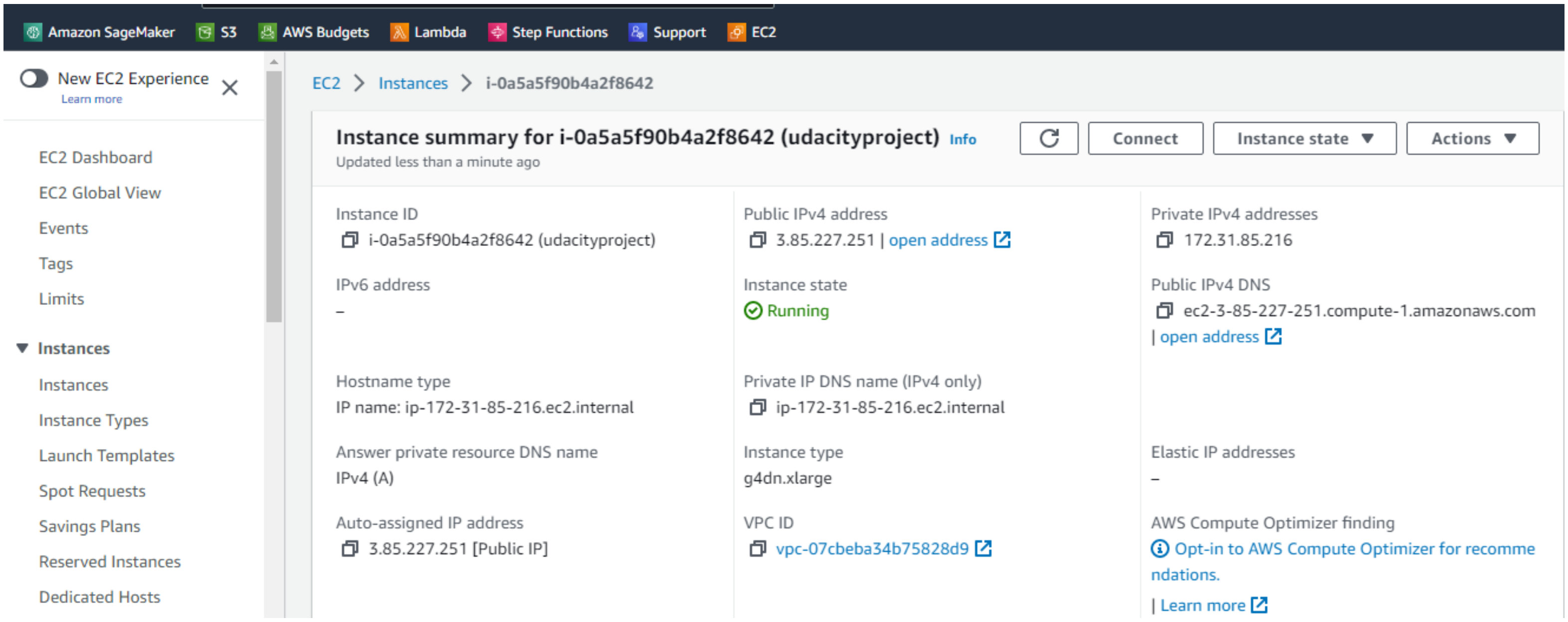
2) Used ‘ml.t3.medium’ instance type for notebook, but later changed the instance type to ‘ml.m5.2xlarge’ in order to train the models without getting resource limit error.
3) Fixed some bugs in the code
4) Trained using multiple instances and prepared results
5) Multi instance training ended after 15 minutes.



	Name	Creation time	Duration	Job status	Warm pool status	Time left
○	dog-pytorch-2023-02-24-19-42-51-265	Feb 24, 2023 19:42 UTC	15 minutes	Completed	-	-
○	dog-pytorch-2023-02-24-19-42-46-077	Feb 24, 2023 19:42 UTC	14 minutes	Completed	-	-
○	pytorch-training-230224-1914-002-3a3e99ee	Feb 24, 2023 19:29 UTC	12 minutes	Completed	Terminated	-
○	pytorch-training-230224-1914-001-b0f33a53	Feb 24, 2023 19:14 UTC	14 minutes	Completed	Reused	-
○	dog-pytorch-2023-02-23-12-23-33-366	Feb 23, 2023 12:23 UTC	17 minutes	Completed	-	-
○	dog-pytorch-2023-02-23-12-16-37-762	Feb 23, 2023 12:16 UTC	16 minutes	Completed	-	-
○	pytorch-training-230223-1136-002-80f38669	Feb 23, 2023 11:51 UTC	12 minutes	Completed	Terminated	-
○	pytorch-training-230223-1136-001-dd54fddd	Feb 23, 2023 11:36 UTC	13 minutes	Completed	Reused	-

Step 2: EC2 Setup

1) Launched an EC2 instance. Used ‘ml.g4dn.xlarge’ instance type in order to train the model without getting memory error. I chose “Deep Learning AMI GPU PyTorch 1.13.1 (Amazon Linux 2) 20230221” as a system type.



Instance summary for i-0a5a5f90b4a2f8642 (udacityproject)
Updated less than a minute ago

Instance ID: i-0a5a5f90b4a2f8642 (udacityproject)

Public IPv4 address: 3.85.227.251 | open address

Private IPv4 addresses: 172.31.85.216

IPv6 address: –

Instance state: Running

Public IPv4 DNS: ec2-3-85-227-251.compute-1.amazonaws.com | open address

Hostname type: IP name: ip-172-31-85-216.ec2.internal

Private IP DNS name (IPv4 only): ip-172-31-85-216.ec2.internal

Answer private resource DNS name: IPv4 (A)

Instance type: g4dn.xlarge

Elastic IP addresses: –

Auto-assigned IP address: 3.85.227.251 [Public IP]

VPC ID: vpc-07cbeba34b75828d9

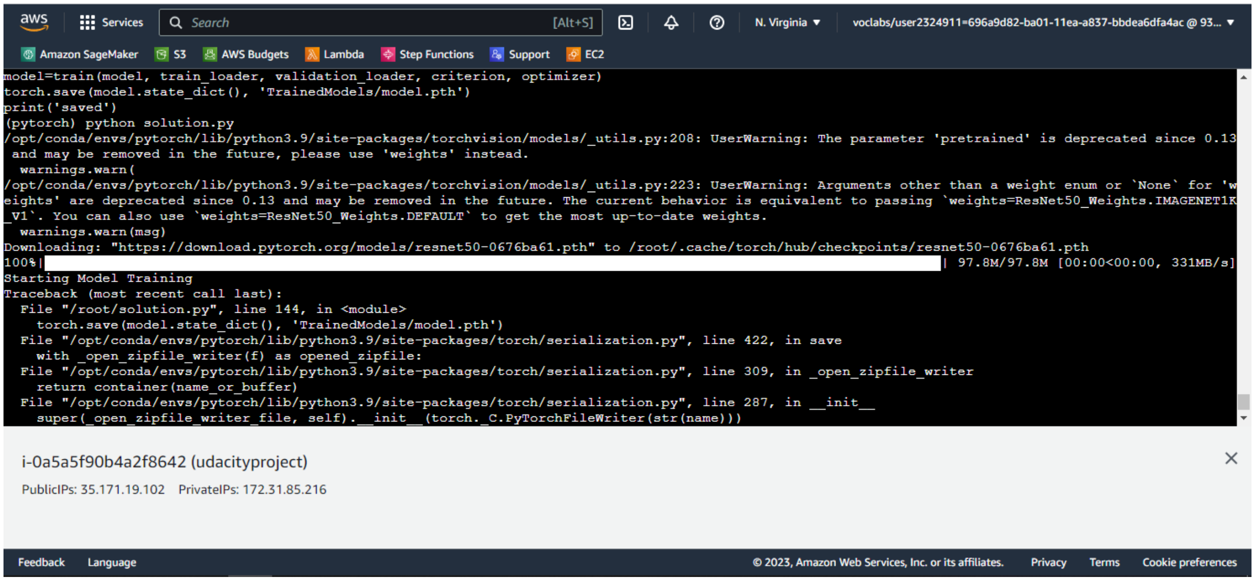
AWS Compute Optimizer finding: Opt-in to AWS Compute Optimizer for recommendations. | Learn more

2) Connected to the instance and trained the model successfully.

Before uploading the data, I used “source activate pytorch” in order not to get any error related to python libraries. After that, I run the following commands in my EC2 terminal:

```
wget https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/dogImages.zip

unzip dogImages.zip

mkdir TrainedModels

cat > solution.py
```

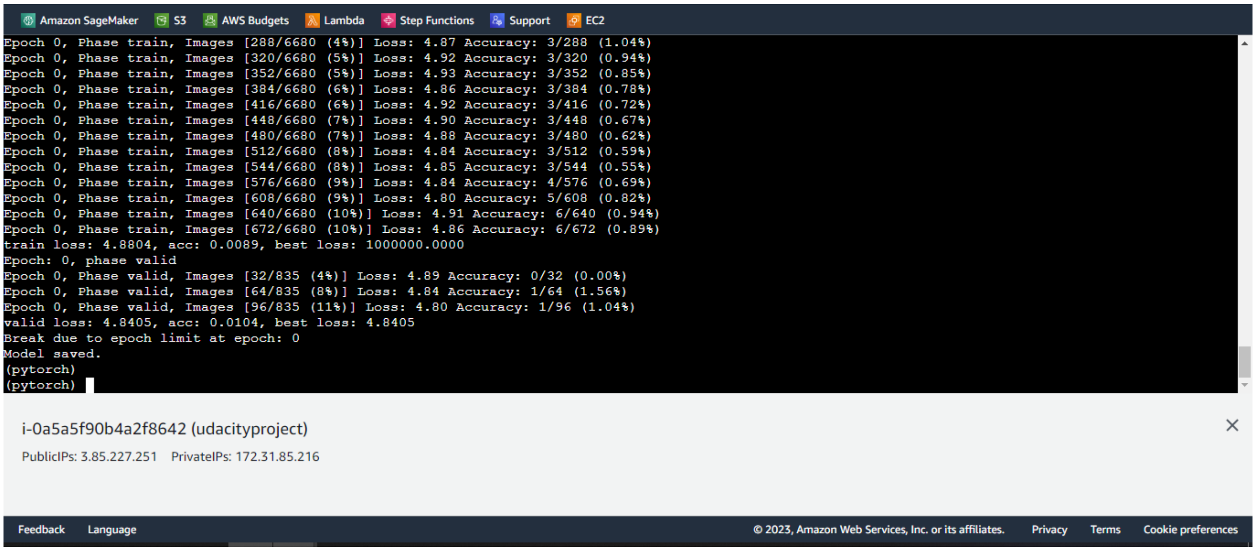
After that, I copied and pasted the file content in “ec2train1.py”. After pasting the code, I used Ctrl+D command to exit.



```
model=train(model, train_loader, validation_loader, criterion, optimizer)
torch.save(model.state_dict(), 'TrainedModels/model.pth')
print('saved')
(pytorch) 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 may be removed in the future, 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 may be removed in the future. 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)
Downloading: "https://download.pytorch.org/models/resnet50-0676ba61.pth" to /root/.cache/torch/hub/checkpoints/resnet50-0676ba61.pth
100%|████████████████████████████████████████████████████████████████████████████████| 97.8M/97.8M [00:00<00:00, 331MB/s]
Starting Model Training
Traceback (most recent call last):
  File "/root/solution.py", line 144, in <module>
    torch.save(model.state_dict(), 'TrainedModels/model.pth')
  File "/opt/conda/envs/pytorch/lib/python3.9/site-packages/torch/serialization.py", line 422, in save
    with _open_zipfile_writer(f) as opened_zipfile:
  File "/opt/conda/envs/pytorch/lib/python3.9/site-packages/torch/serialization.py", line 309, in _open_zipfile_writer
    return container(name_or_buffer)
  File "/opt/conda/envs/pytorch/lib/python3.9/site-packages/torch/serialization.py", line 287, in __init__
    super(_open_zipfile_writer_file, self).__init__(torch._C.PyTorchFileWriter(str(name)))
```

i-0a5a5f90b4a2f8642 (udacityproject)

PublicIPs: 35.171.19.102 PrivateIPs: 172.31.85.216



```
Epoch 0, Phase train, Images [288/6680 (4%)] Loss: 4.87 Accuracy: 3/288 (1.04%)
Epoch 0, Phase train, Images [320/6680 (5%)] Loss: 4.92 Accuracy: 3/320 (0.94%)
Epoch 0, Phase train, Images [352/6680 (5%)] Loss: 4.93 Accuracy: 3/352 (0.85%)
Epoch 0, Phase train, Images [384/6680 (6%)] Loss: 4.86 Accuracy: 3/384 (0.78%)
Epoch 0, Phase train, Images [416/6680 (6%)] Loss: 4.92 Accuracy: 3/416 (0.72%)
Epoch 0, Phase train, Images [448/6680 (7%)] Loss: 4.90 Accuracy: 3/448 (0.67%)
Epoch 0, Phase train, Images [480/6680 (7%)] Loss: 4.88 Accuracy: 3/480 (0.62%)
Epoch 0, Phase train, Images [512/6680 (8%)] Loss: 4.84 Accuracy: 3/512 (0.59%)
Epoch 0, Phase train, Images [544/6680 (8%)] Loss: 4.85 Accuracy: 3/544 (0.55%)
Epoch 0, Phase train, Images [576/6680 (9%)] Loss: 4.84 Accuracy: 4/576 (0.69%)
Epoch 0, Phase train, Images [608/6680 (9%)] Loss: 4.80 Accuracy: 5/608 (0.82%)
Epoch 0, Phase train, Images [640/6680 (10%)] Loss: 4.91 Accuracy: 6/640 (0.94%)
Epoch 0, Phase train, Images [672/6680 (10%)] Loss: 4.86 Accuracy: 6/672 (0.89%)
train loss: 4.8804, acc: 0.0089, best loss: 1000000.0000
Epoch: 0, phase valid
Epoch 0, Phase valid, Images [32/835 (4%)] Loss: 4.89 Accuracy: 0/32 (0.00%)
Epoch 0, Phase valid, Images [64/835 (8%)] Loss: 4.84 Accuracy: 1/64 (1.56%)
Epoch 0, Phase valid, Images [96/835 (11%)] Loss: 4.80 Accuracy: 1/96 (1.04%)
valid loss: 4.8405, acc: 0.0104, best loss: 4.8405
Break due to epoch limit at epoch: 0
Model saved.
(pytorch)
(pytorch)
```

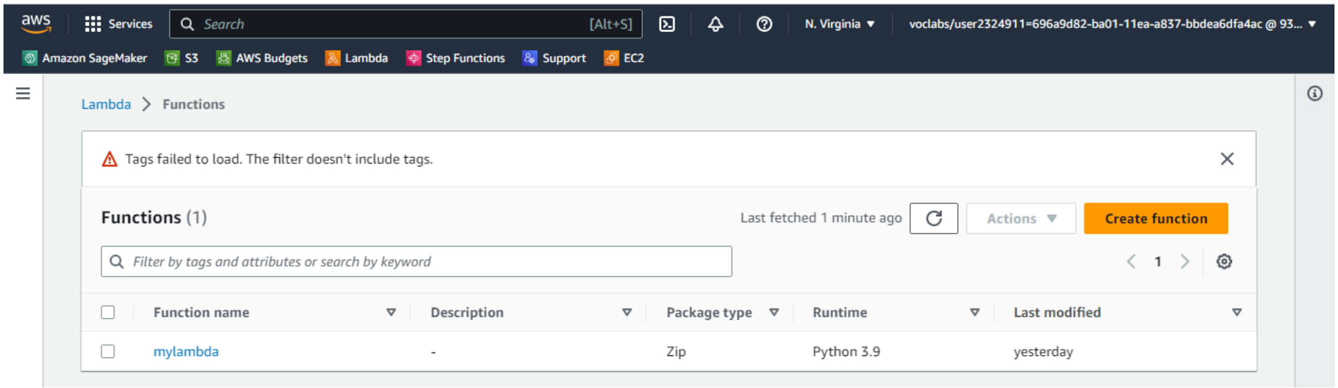
i-0a5a5f90b4a2f8642 (udacityproject)

PublicIPs: 3.85.227.251 PrivateIPs: 172.31.85.216

There are some similarities between the Amazon Sagemaker Studio and EC2 instances. However, the code for Amazon EC2 instance should be a self-contained Python script. By using the EC2 instance, we can train a model by using a local file system to access the data, as opposed to Amazon Sagemaker script that need methods for accessing Amazon S3 datasets.

Step 3,4: Lambda function setup and Security and testing

1) Created a new lambda function by using the “lambdafunction.py”.
2) After creating a new lambda function, additional polices were added in order to run the lambda function successfully.

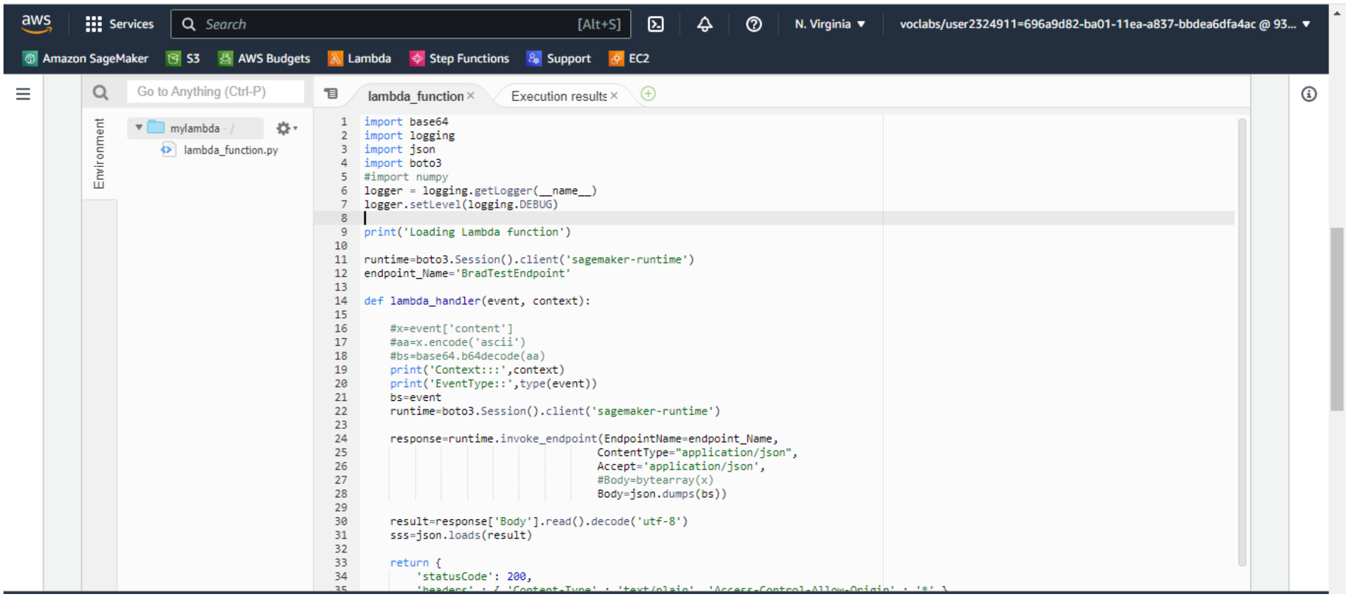


Tags failed to load. The filter doesn't include tags.

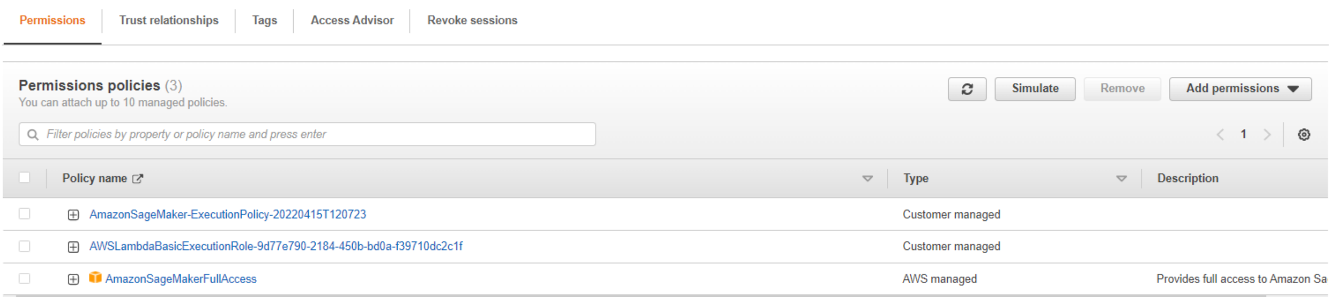
Functions (1)

Last fetched 1 minute ago

□	Function name	Description	Package type	Runtime	Last modified
□	mylambda	-	Zip	Python 3.9	yesterday



```python
import base64
import logging
import json
import boto3
#import numpy
logger = logging.getLogger(__name__)
logger.setLevel(logging.DEBUG)

print('Loading Lambda function')

runtime=boto3.Session().client('sagemaker-runtime')
endpoint_Name='BradTestEndpoint'

def lambda_handler(event, context):

    #x=event['content']
    #aa=x.encode('ascii')
    #bs=base64.b64decode(aa)
    print('Context:::',context)
    print('EventType::',type(event))
    bs=event
    runtime=boto3.Session().client('sagemaker-runtime')

    response=runtime.invoke_endpoint(EndpointName=endpoint_Name,
                                        ContentType="application/json",
                                        Accept='application/json',
                                        #Body=bytearray(x)
                                        Body=json.dumps(bs))

    result=response['Body'].read().decode('utf-8')
    sss=json.loads(result)

    return {
        'statusCode': 200,
        'headers' : { 'Content-Type' : 'text/plain', 'Access-Control-Allow-Origin' : '*' }
```



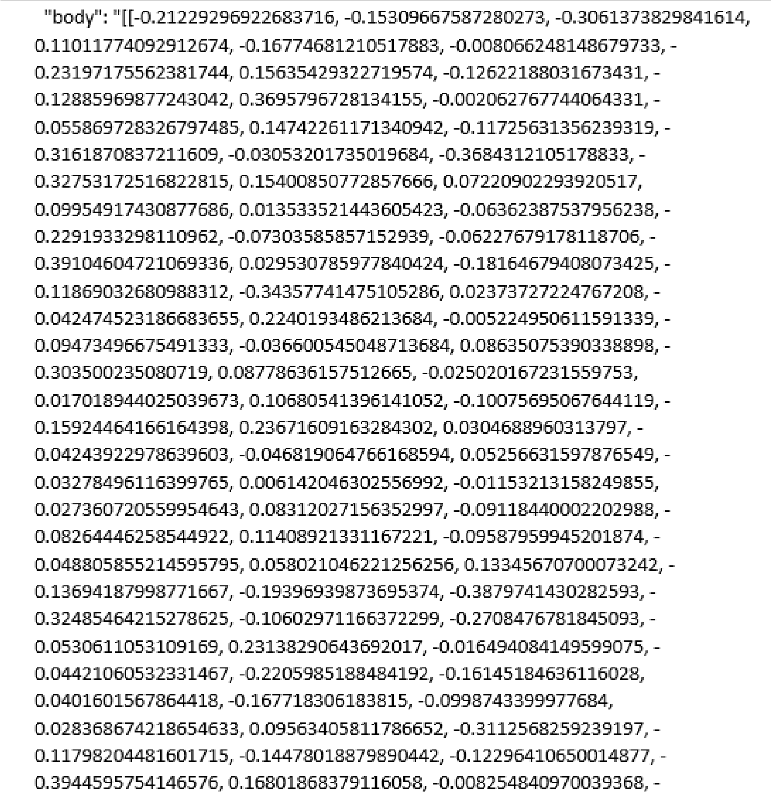
Permissions policies (3)

You can attach up to 10 managed policies.

□	Policy name	Type	Description
□	⊞ AmazonSageMaker-ExecutionPolicy-20220415T120723	Customer managed	
□	⊞ AWSLambdaBasicExecutionRole-9d77e790-2184-450b-bd0a-f39710dc2c1f	Customer managed	
□	⊞ AmazonSageMakerFullAccess	AWS managed	Provides full access to Amazon Sa

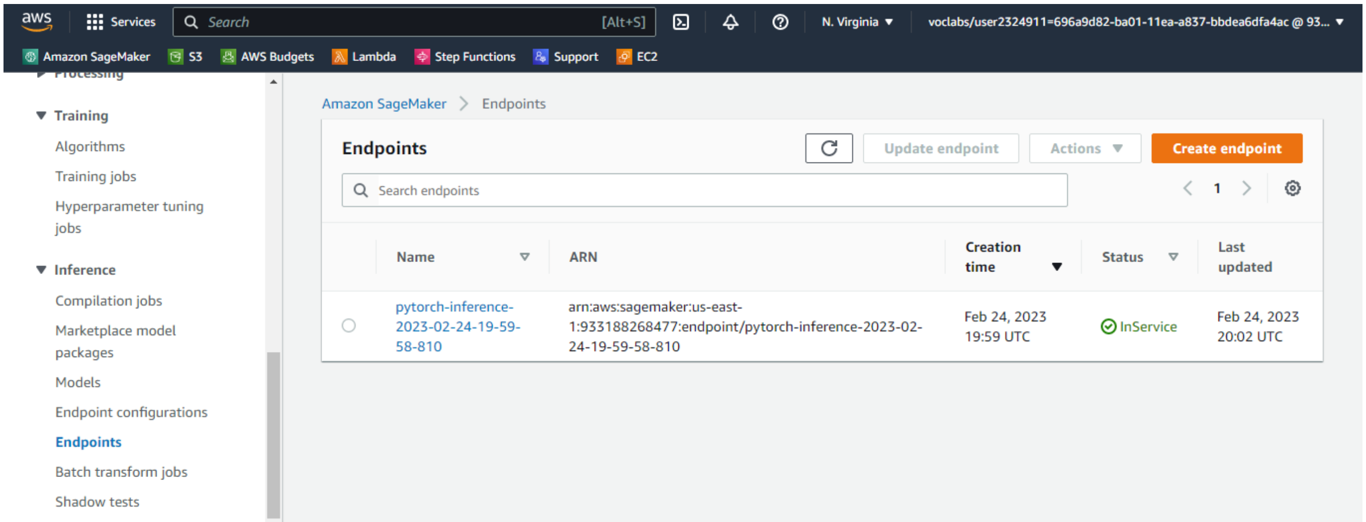
“Invoke Endpoint” works in the same way as a “Predict” in Amazon Sagemaker Studio.

Output of the lambda function:



```
  "body": "[[-0.21229296922683716, -0.15309667587280273, -0.3061373829841614,
0.11011774092912674, -0.16774681210517883, -0.008066248148679733, -
0.23197175562381744, 0.15635429322719574, -0.12622188031673431, -
0.12885969877243042, 0.3695796728134155, -0.002062767744064331, -
0.055869728326797485, 0.14742261171340942, -0.11725631356239319, -
0.3161870837211609, -0.03053201735019684, -0.3684312105178833, -
0.32753172516822815, 0.15400850772857666, 0.07220902293920517,
0.09954917430877686, 0.013533521443605423, -0.06362387537956238, -
0.2291933298110962, -0.07303585857152939, -0.06227679178118706, -
0.39104604721069336, 0.029530785977840424, -0.18164679408073425, -
0.11869032680988312, -0.34357741475105286, 0.02373727224767208, -
0.042474523186683655, 0.2240193486213684, -0.005224950611591339, -
0.09473496675491333, -0.036600545048713684, 0.08635075390338898, -
0.303500235080719, 0.08778636157512665, -0.025020167231559753,
0.017018944025039673, 0.10680541396141052, -0.10075695067644119, -
0.15924464166164398, 0.23671609163284302, 0.0304688960313797, -
0.04243922978639603, -0.046819064766168594, 0.05256631597876549, -
0.03278496116399765, 0.006142046302556992, -0.01153213158249855,
0.027360720559954643, 0.08312027156352997, -0.09118440002202988, -
0.08264446258544922, 0.11408921331167221, -0.09587959945201874, -
0.048805855214595795, 0.058021046221256256, 0.13345670700073242, -
0.13694187998771667, -0.19396939873695374, -0.3879741430282593, -
0.32485464215278625, -0.10602971166372299, -0.2708476781845093, -
0.0530611053109169, 0.23138290643692017, -0.016494084149599075, -
0.044210605323331467, -0.22059851884841 92, -0.16145184636116028,
0.0401601567864418, -0.167718306183815, -0.0998743399977684,
0.028368674218654633, 0.09563405811786652, -0.3112568259239197, -
0.11798204481601715, -0.14478018879890442, -0.12296410650014877, -
0.3944595754146576, 0.16801868379116058, -0.008254840970039368, -
```

Step 5: Concurrency and auto-scaling



	Name	ARN	Creation time ▼	Status	Last updated
○	pytorch-inference-2023-02-24-19-59-58-810	arn:aws:sagemaker:us-east-1:933188268477:endpoint/pytorch-inference-2023-02-24-19-59-58-810	Feb 24, 2023 19:59 UTC	✓ InService	Feb 24, 2023 20:02 UTC

I set a minimum acceptable value of 2 concurrent lambda and endpoint executions for this project, the expected endpoint traffic is zero for this project as it will be removed.