Operationalizing an AWS ML Project

Dog Image Classification

The objective of the project is to finish the following steps:

1. Train and deploy a model on Sagemaker, using the most appropriate instances. Set up multi-instance training in your Sagemaker notebook.

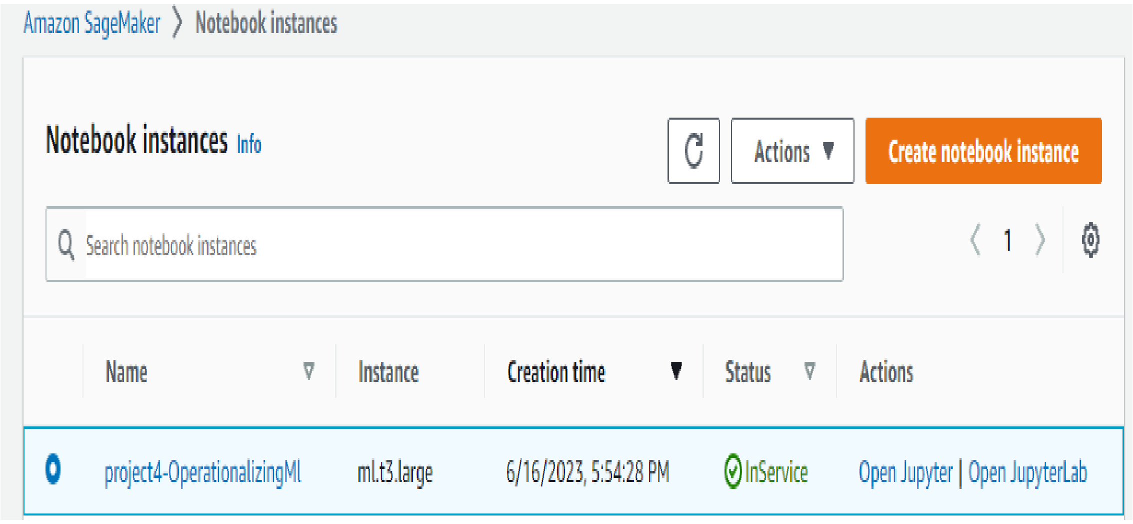
2. Adjust your Sagemaker notebooks to perform training and deployment on EC2.

3. Set up a Lambda function for your deployed model. Set up auto-scaling .\ for your deployed endpoint as well as concurrency for your Lambda function.

4. Ensure that the security on your ML pipeline is set up properl.

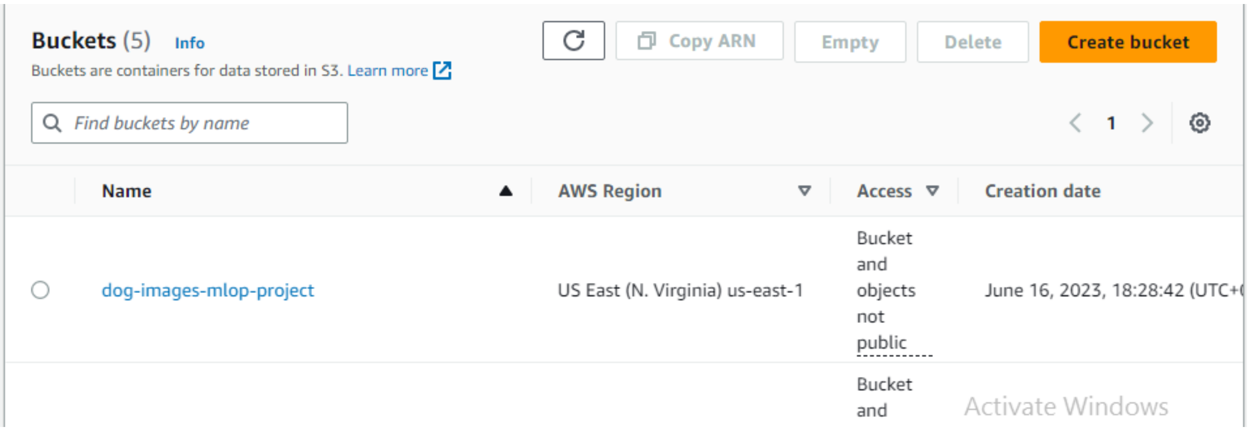
Step 1: Training and deployment on Sagemaker:

Created Sagemaker notebook instance I have used ml.t3.large as this is sufficient and has a low cost to run my notebook.


Amazon SageMaker 〉 Notebook instances
Notebook instances Info
Actions ▼
Create notebook instance
Search notebook instances
〈 1 〉

	Name	Instance	Creation time	Status	Actions
○	project4-OperationalizingMl	ml.t3.large	6/16/2023, 5:54:28 PM	InService	Open Jupyter | Open JupyterLab

S3 bucket for the job (dog–images–mlop–project)

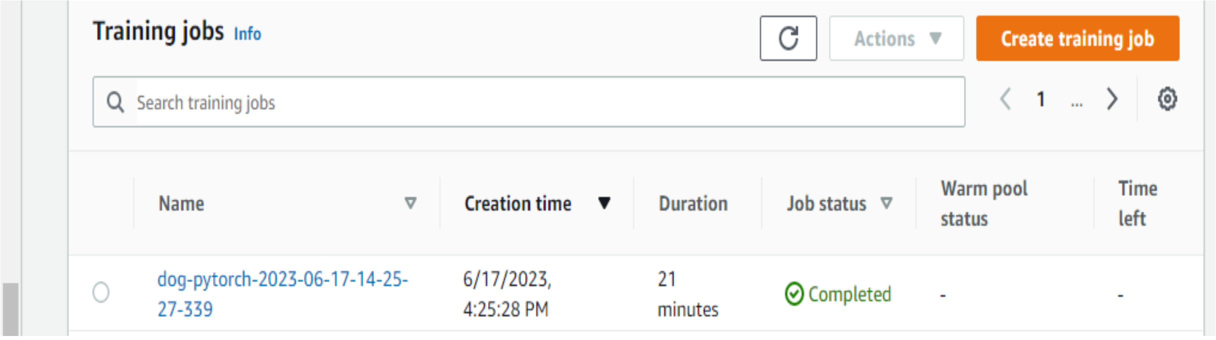


Buckets (5) Info

Buckets are containers for data stored in S3. Learn more

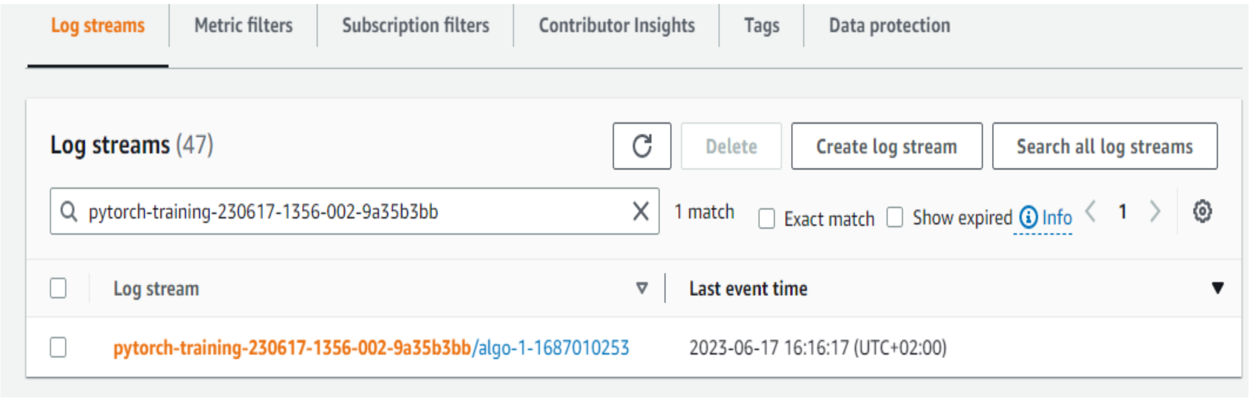
	Name ▲	AWS Region ▽	Access ▽	Creation date
○	dog-images-mlop-project	US East (N. Virginia) us-east-1	Bucket and objects not public	June 16, 2023, 18:28:42 (UTC+
			Bucket and	

Single instance training



Training jobs Info

	Name ▽	Creation time ▼	Duration	Job status ▽	Warm pool status	Time left
○	dog-pytorch-2023-06-17-14-25-27-339	6/17/2023, 4:25:28 PM	21 minutes	✓ Completed	-	-

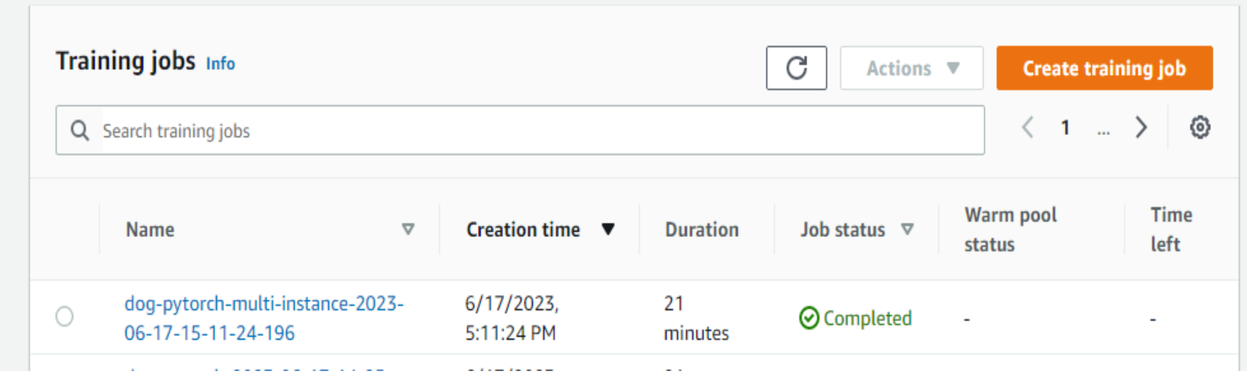


Log streams (47)

pytorch-training-230617-1356-002-9a35b3bb

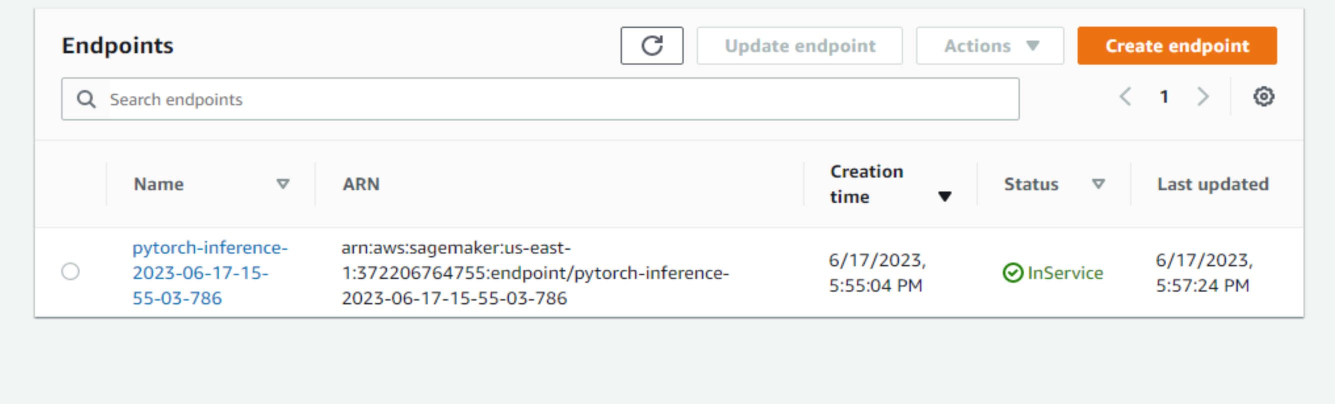
□	Log stream ▽	Last event time ▼
□	pytorch-training-230617-1356-002-9a35b3bb/algo-1-1687010253	2023-06-17 16:16:17 (UTC+02:00)

Multi–Instance training



	Name	Creation time	Duration	Job status	Warm pool status	Time left
○	dog-pytorch-multi-instance-2023-06-17-15-11-24-196	6/17/2023, 5:11:24 PM	21 minutes	Completed	-	-

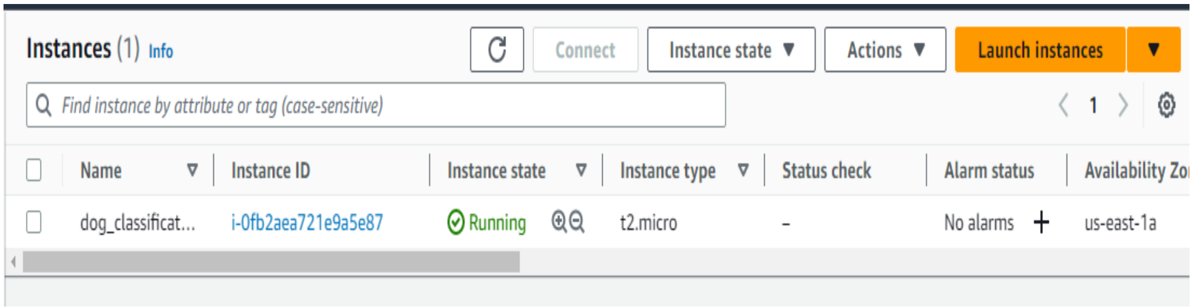
Deployment



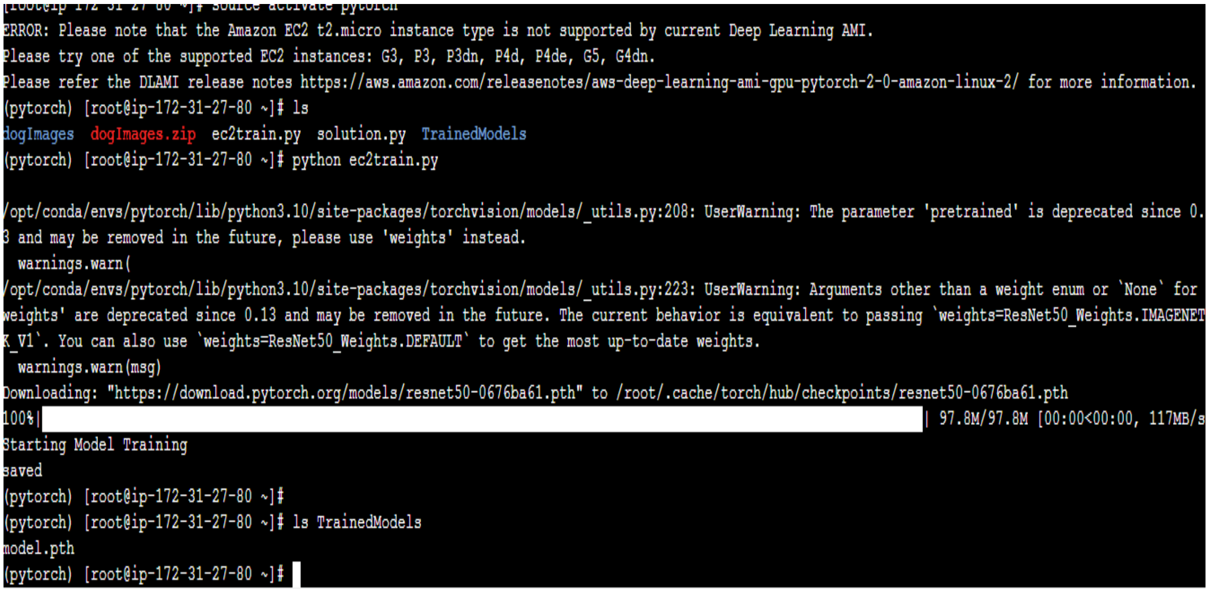
	Name	ARN	Creation time	Status	Last updated
○	pytorch-inference-2023-06-17-15-55-03-786	arn:aws:sagemaker:us-east-1:372206764755:endpoint/pytorch-inference-2023-06-17-15-55-03-786	6/17/2023, 5:55:04 PM	InService	6/17/2023, 5:57:24 PM

Step 2: EC2 Training

I have used t2.micro as it is a free tier



□	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zo
□	dog_classificat...	i-0fb2aea721e9a5e87	✓ Running	t2.micro	-	No alarms +	us-east-1a



```
[root@ip-172-31-27-80 ~]# source activate pytorch
ERROR: Please note that the Amazon EC2 t2.micro instance type is not supported by current Deep Learning AMI.
Please try one of the supported EC2 instances: G3, P3, P3dn, P4d, P4de, G5, G4dn.
Please refer the DLAMI release notes https://aws.amazon.com/releasenotes/aws-deep-learning-ami-gpu-pytorch-2-0-amazon-linux-2/ for more information.
(pytorch) [root@ip-172-31-27-80 ~]# ls
dogImages  dogImages.zip  ec2train.py  solution.py  TrainedModels
(pytorch) [root@ip-172-31-27-80 ~]# python ec2train.py

/opt/conda/envs/pytorch/lib/python3.10/site-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since 0.
3 and may be removed in the future, please use 'weights' instead.
  warnings.warn(
/opt/conda/envs/pytorch/lib/python3.10/site-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `None` for
weights' are deprecated since 0.13 and may be removed in the future. The current behavior is equivalent to passing `weights=ResNet50_Weights.IMAGENET
K_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, 117MB/s
Starting Model Training
saved
(pytorch) [root@ip-172-31-27-80 ~]#
(pytorch) [root@ip-172-31-27-80 ~]# ls TrainedModels
model.pth
(pytorch) [root@ip-172-31-27-80 ~]#
```

The adjusted code in ec2train1.py is very similar to the code in train_and_deploy-solution.ipynb. But there are few differences between the modules used - some modules can only be used in SageMaker. Much of the EC2 training code has also been adapted from the functions defined in the hpo.py starter script. Ec2train.py trains model with specific arguments while hpo.py takes argument for model by parsing through command line. The later code can train multiple model with different hyper parameters.

Advantages and Disadvantages of EC2

The advantages of EC2 Instances are less expensive than SageMaker instances, but the disadvantage of them is that they offer fewer managed.

Step 3: Step 3: Lambda function setup

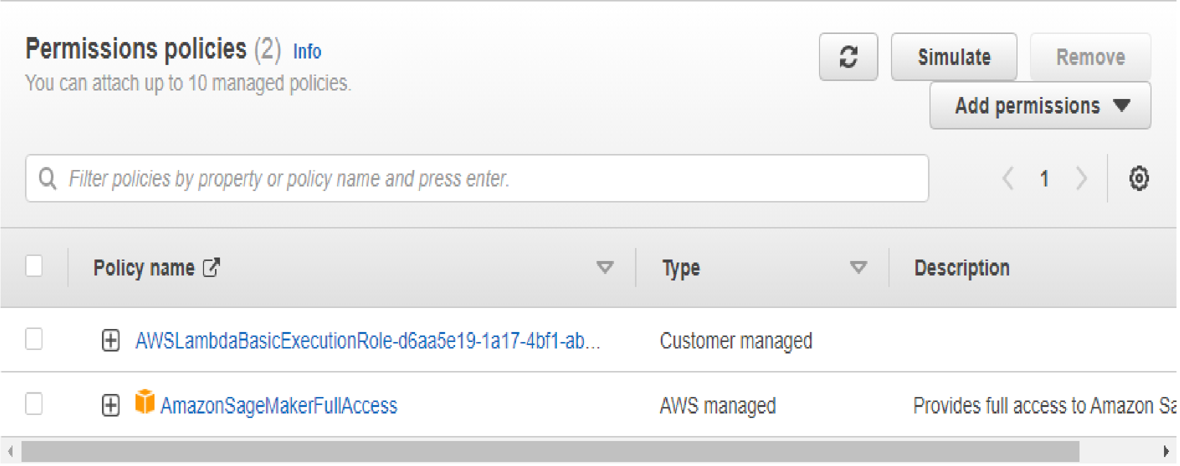
After training and deploying your model, setting up a Lambda function is an important next step. Lambda functions enable your model and its inferences to be accessed by API's and other programs, so it's a crucial part of production deployment.

Thus, I deployed the lambda function with my endpoint name.

Step 4: Lambda security setup and testing

Adding endpoints permission to lambda functions

The lambda function will only be able to invoke endpoint if it has the proper security policies attached to it. Therefore I attached **AmazonSageMakerFullAccess** to the role of the lambda function



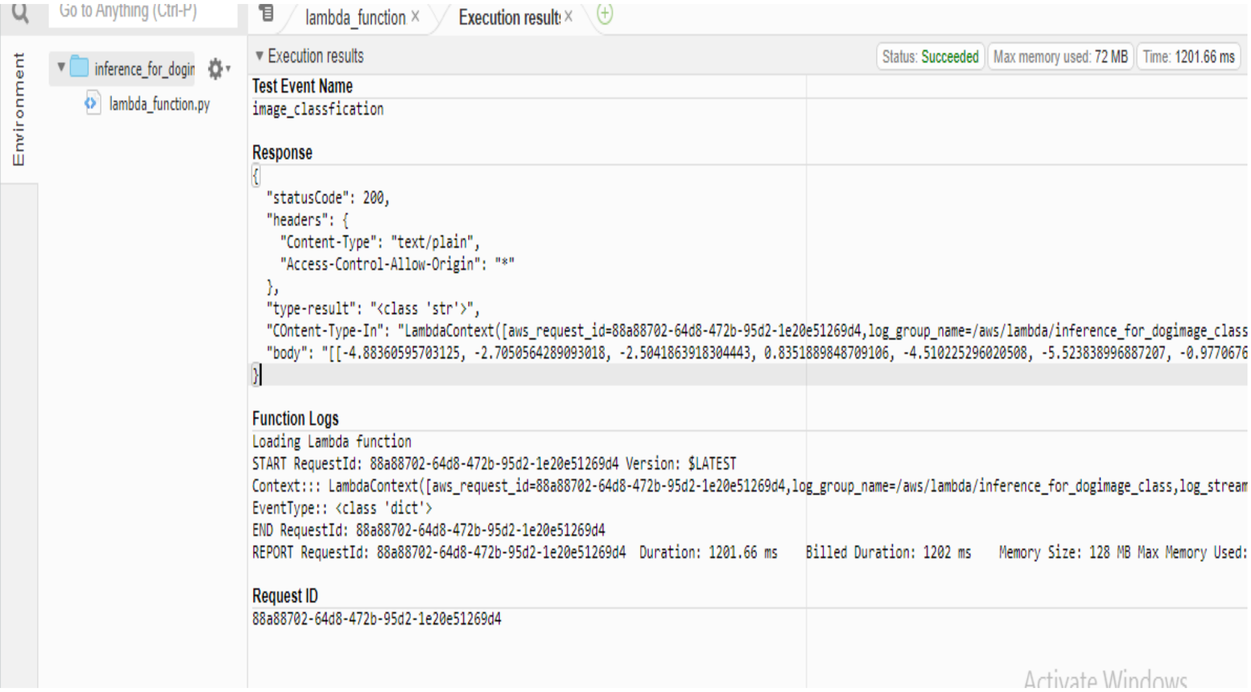
Permissions policies (2)

You can attach up to 10 managed policies.

□	Policy name	Type	Description
□	AWSLambdaBasicExecutionRole-d6aa5e19-1a17-4bf1-ab...	Customer managed	
□	AmazonSageMakerFullAccess	AWS managed	Provides full access to Amazon Sa

Testing lambda function

Now with the right permission we can create a new test to test our Lambda Function.



Test Event Name

```
image_classfication
```

Response

```
{
  "statusCode": 200,
  "headers": {
    "Content-Type": "text/plain",
    "Access-Control-Allow-Origin": "*"
  },
  "type-result": "<class 'str'>",
  "COntent-Type-In": "LambdaContext([aws_request_id=88a88702-64d8-472b-95d2-1e20e51269d4,log_group_name=/aws/lambda/inference_for_dogimage_class
  "body": "[[-4.88360595703125, -2.7050564289093018, -2.5041863918304443, 0.8351889848709106, -4.510225296020508, -5.523838996887207, -0.9770676
}
```

Function Logs

```
Loading Lambda function
START RequestId: 88a88702-64d8-472b-95d2-1e20e51269d4 Version: $LATEST
Context::: LambdaContext([aws_request_id=88a88702-64d8-472b-95d2-1e20e51269d4,log_group_name=/aws/lambda/inference_for_dogimage_class,log_stream
EventType:: <class 'dict'>
END RequestId: 88a88702-64d8-472b-95d2-1e20e51269d4
REPORT RequestId: 88a88702-64d8-472b-95d2-1e20e51269d4  Duration: 1201.66 ms    Billed Duration: 1202 ms    Memory Size: 128 MB Max Memory Used:
```

Request ID

```
88a88702-64d8-472b-95d2-1e20e51269d4
```

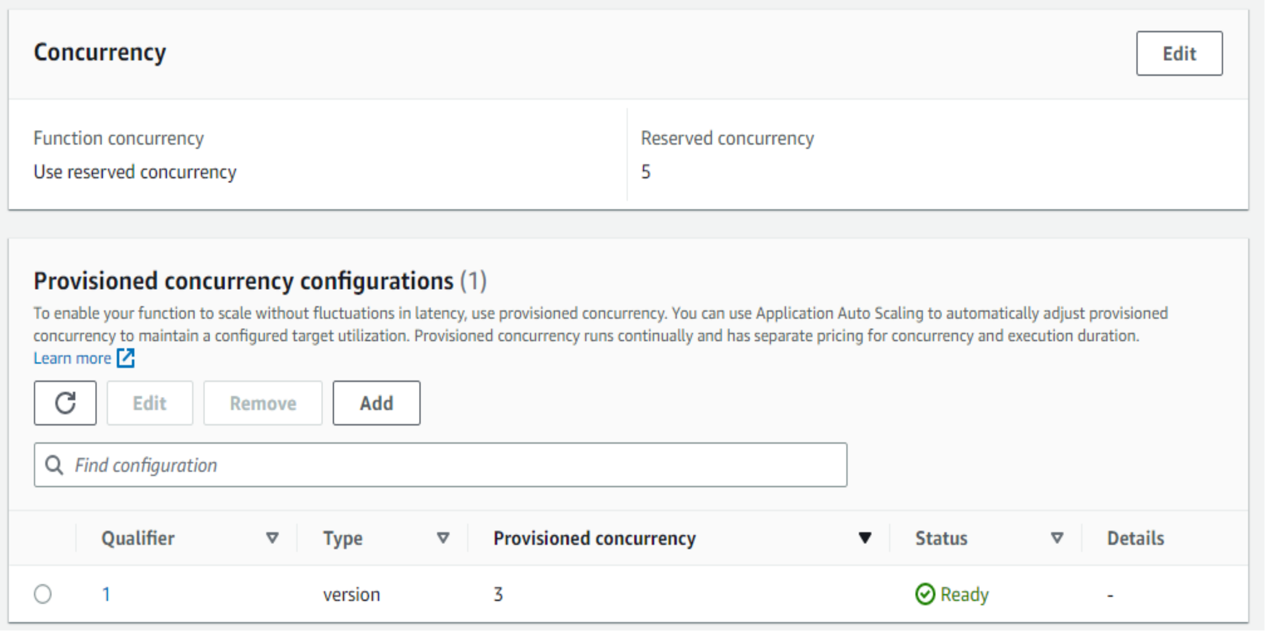
Step 5: Lambda concurrency setup and endpoint auto-scaling

Adding concurrency to Lambda Function

By default, Lambda functions process one request at a time. If they receive multiple requests at the same time, they process one while the other ones wait. After the first request is processed, then the lambda function can move on to process the second request, then the third, and so on.

If you want to decrease the latency of your project in these high-traffic situations, you can implement concurrency, because concurrency so that the Lambda Function can respond to multiple requests at once.

Reserved instances: 5/1000 Provisioned instances: 3/5



Concurrency

Function concurrency
Use reserved concurrency

Reserved concurrency
5

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.

	Qualifier	Type	Provisioned concurrency	Status	Details
○	1	version	3	✓ Ready	-

Auto–scaling

Sagemaker endpoints require automatic scaling to respond to high traffic.

I enabled auto–scaling:

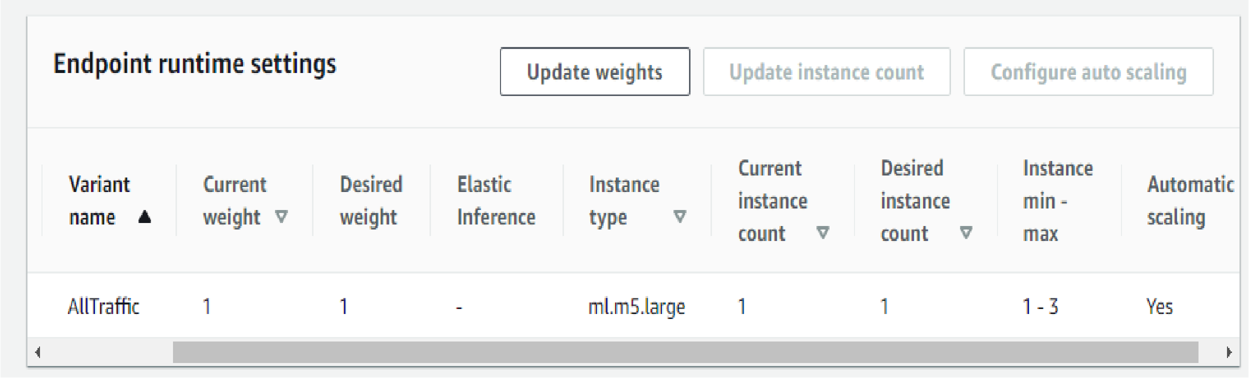
Minimum instances: 1

Maximum instances: 3

Target value: 20 number of simultaneous requests which will trigger scaling

scale–in time: 30 s

scale–out time: 30 s



Endpoint runtime settings

Variant name	Current weight	Desired weight	Elastic Inference	Instance type	Current instance count	Desired instance count	Instance min - max	Automatic scaling
AllTraffic	1	1	-	ml.m5.large	1	1	1 - 3	Yes

scale–in cool down:

The scale-in period is the amount of time AWS will wait before deploying more instances for your endpoint.

- If you choose a high number, then AWS will wait a long time before deploying more instances. This helps you avoid incurring costs for momentary spikes in traffic.
- If you choose a low number, then AWS will deploy instances more quickly, but this responsiveness will be more costly.

scale–out cool down

- The scale-out cool down period is the amount of time AWS will wait before deleting extra deployed instances. If you choose a low number, then AWS will wait only a short time before deleting extra deployed instances. This helps you avoid incurring costs for momentary spikes in traffic. If you choose a high number, then AWS will keep extra instances deployed longer, but this extra capacity will be more costly.

Therefore I choose an optimal value for both of them as well the low number of instances because of the low budget.