writeup

The aim of this project is to make the Dogs breed classifier model production ready. With this intention several industrial practices have been applied to reduce training and compute time, to ensure availability and to enhance security while keeping cost considerations in constant view.

The details of the adopted practices are summarized in this report.

Step 1: Training and deployment on Sagemaker

Model training is primarily the most time intensive process in complex machine learning projects which needs to process a large dataset.

Several mechanisms can be adopted to deal with this challenge. For instance, if the number of features are huge then, in the data preprocessing phase, dimensionality reduction can be considered which reduces the number of input features while retaining the variation in the dataset. Another approach could be to subsample the dataset which will naturally reduce the training time.

If budget permits then multi-instance training can be explored. In multi-instance training, several instanses can be introduced to reduce the training time. To further reduce the training time data can be sharded by key and distributed across the multiple instances.

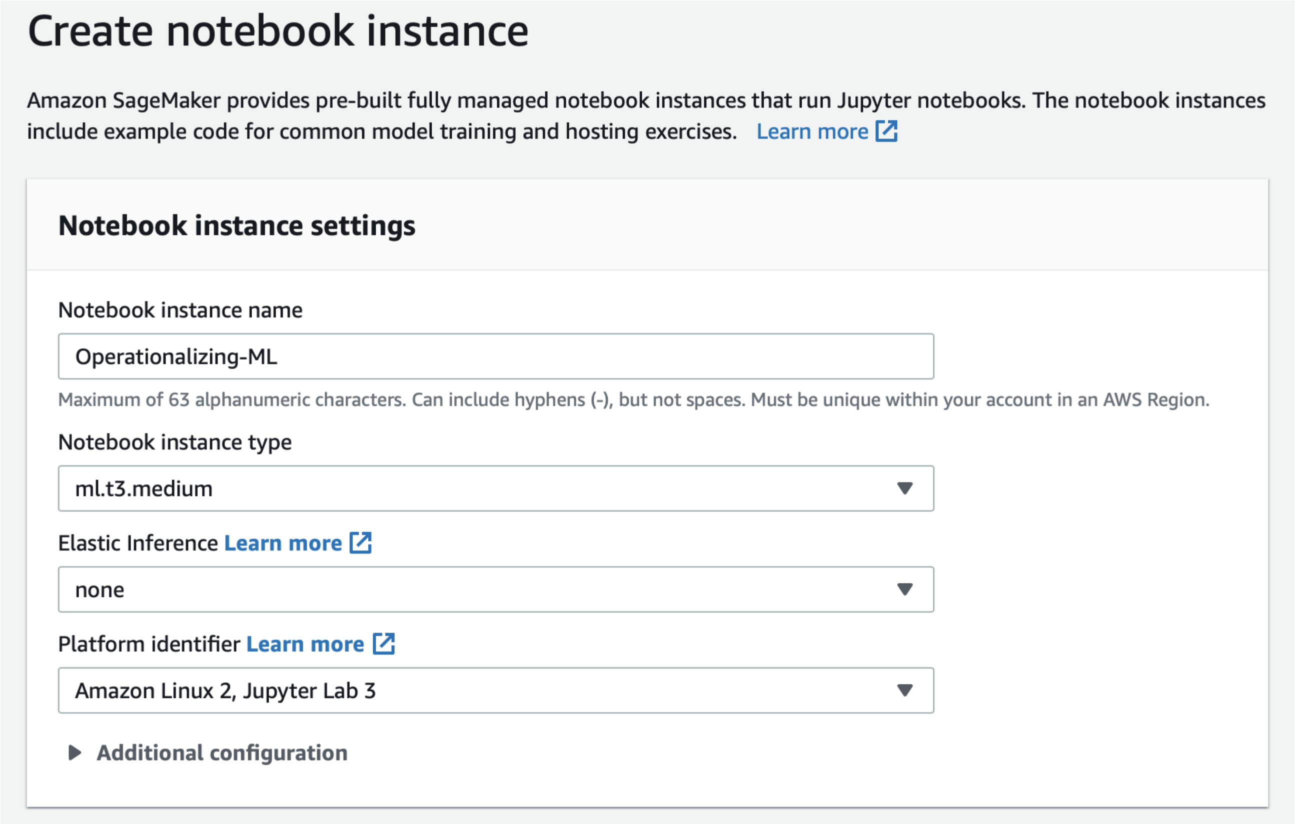
Model training

For reducing the model training time of the Dogs breed classifier, multi-instance training is setup in 'train_and_deploy-solution.ipynb'. The following steps were performed:

1. Created AWS Sagemaker notebook instance
   To execute the 'train_and_deploy-solution.ipynb' containing the model training and deployment cofiguration code, Operationalizing-ML sagemaker notebook instance is created with 'ml.t3.medium' instance type.

   The 'ml.t3.medium' was deemed appropriate for the task as it is the least expensive option in US East (N. Virginia) and provides sufficient computing power (2 CPUs and 4 GiB) to run the generic tasks in the notebook like downloading data, uploading to S3 and other config steps.

The other time intensive steps like hyperparameter tuning, training have a separate, more powerful instance type defined in the tuner and estimator objects respectively.



Create notebook instance

Amazon SageMaker provides pre-built fully managed notebook instances that run Jupyter notebooks. The notebook instances include example code for common model training and hosting exercises. Learn more

Notebook instance settings

Notebook instance name

Operationalizing-ML

Maximum of 63 alphanumeric characters. Can include hyphens (-), but not spaces. Must be unique within your account in an AWS Region.

Notebook instance type

ml.t3.medium

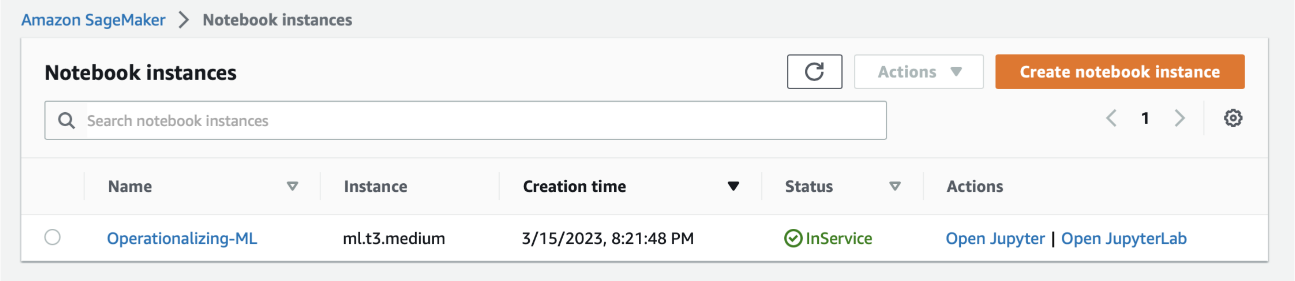
Elastic Inference Learn more

none

Platform identifier Learn more

Amazon Linux 2, Jupyter Lab 3

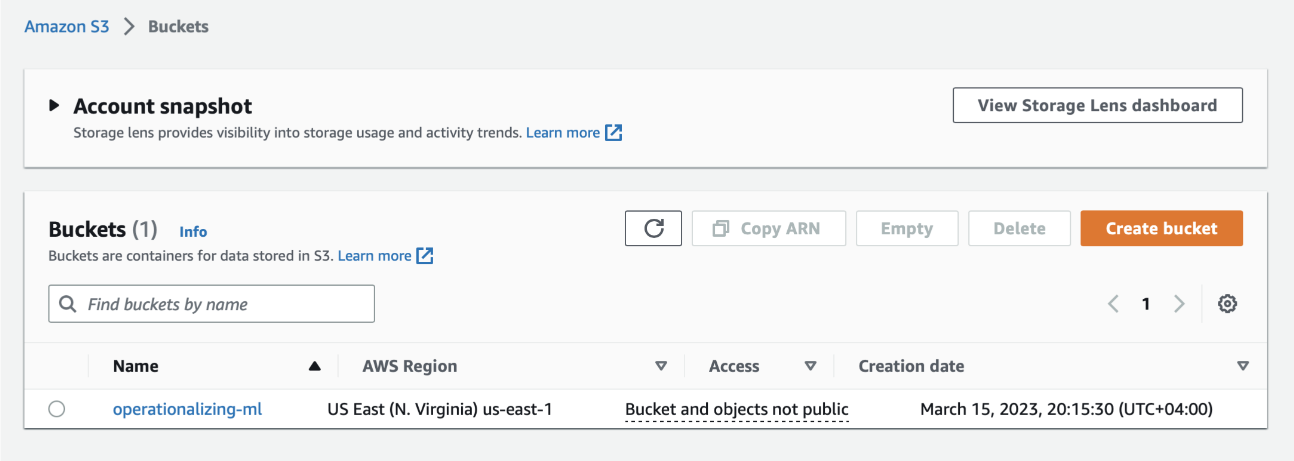
▶ Additional configuration



Amazon SageMaker > Notebook instances
Notebook instances
Actions ▼
Create notebook instance
Search notebook instances
1

	Name	Instance	Creation time	Status	Actions
○	Operationalizing-ML	ml.t3.medium	3/15/2023, 8:21:48 PM	InService	Open Jupyter | Open JupyterLab

2. Setup S3 bucket
   The unstructured image data from dogs breed dataset is fetched and uploaded to S3 bucket named 'operationalizing-ml' in preparation for training.



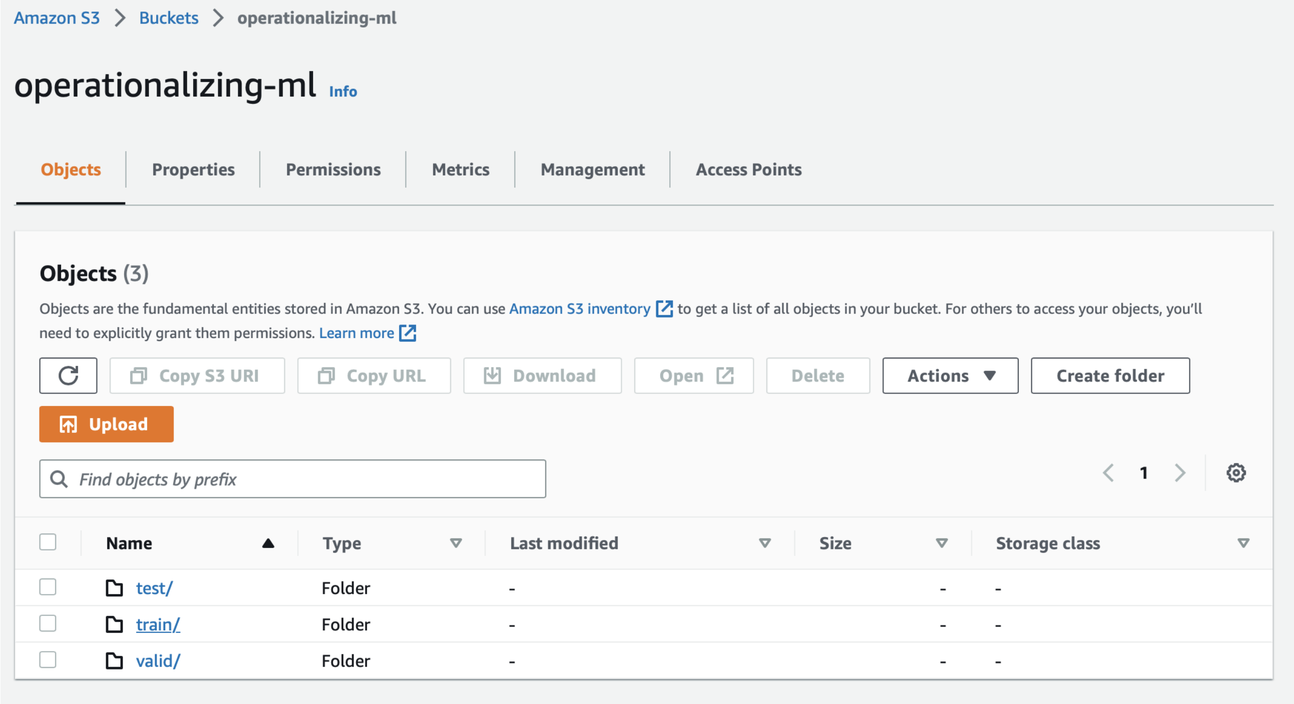
Account snapshot

Storage lens provides visibility into storage usage and activity trends. Learn more

Buckets (1)

Buckets are containers for data stored in S3. Learn more

	Name	AWS Region	Access	Creation date
○	operationalizing-ml	US East (N. Virginia) us-east-1	Bucket and objects not public	March 15, 2023, 20:15:30 (UTC+04:00)



operationalizing-ml

Objects (3)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

□	Name	Type	Last modified	Size	Storage class
□	test/	Folder	-	-	-
□	train/	Folder	-	-	-
□	valid/	Folder	-	-	-

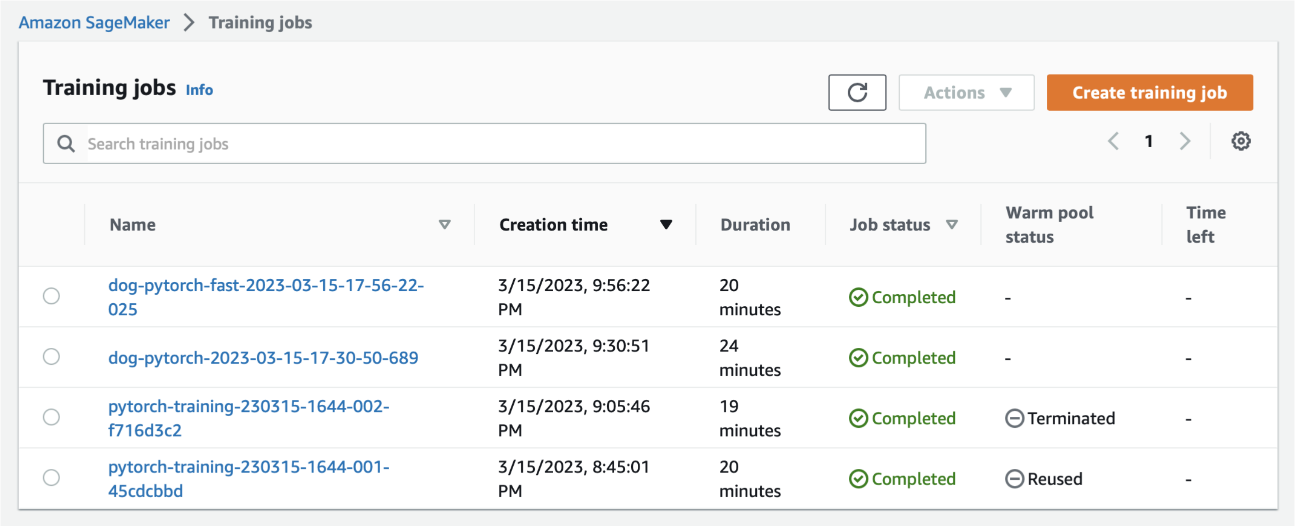
3. Adjusted the parameter values in the training estimator

   As can be seen in the code instance below the instance type was set to 'ml.m5.xlarge' and the instance_count was increased to 4 to enable the training job to be executed across 4 instances and gain reduction in training time.

```
fastestimator = PyTorch(entry_point='hpo.py',base_job_name='dog-pytorch-fast', role=role, instance_count=4, instance_type='ml.m5.xlarge', framework_version='1.4.0', py_version='py3', hyperparameters=hyperparameters, rules = rules, debugger_hook_config=hook_config, profiler_config=profiler_config, )
```

4. Compared training outcome

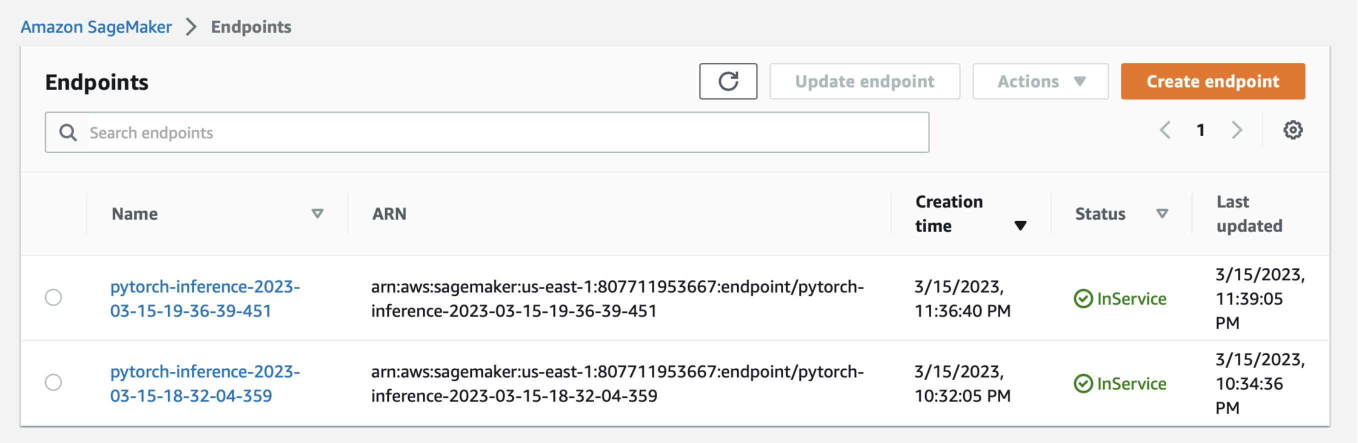
As can be observed from the screenshot, with multi-training instance the training time was reduced from 24 mins to 20 mins.



	Name	Creation time	Duration	Job status	Warm pool status	Time left
○	dog-pytorch-fast-2023-03-15-17-56-22-025	3/15/2023, 9:56:22 PM	20 minutes	✓ Completed	-	-
○	dog-pytorch-2023-03-15-17-30-50-689	3/15/2023, 9:30:51 PM	24 minutes	✓ Completed	-	-
○	pytorch-training-230315-1644-002-f716d3c2	3/15/2023, 9:05:46 PM	19 minutes	✓ Completed	⊖ Terminated	-
○	pytorch-training-230315-1644-001-45cdcbbd	3/15/2023, 8:45:01 PM	20 minutes	✓ Completed	⊖ Reused	-

Model Deployment

After training was completed, both models i.e. one trained with a single instance and the one trained with multiple instances were deployed as an endpoint.



	Name	ARN	Creation time	Status	Last updated
○	pytorch-inference-2023-03-15-19-36-39-451	arn:aws:sagemaker:us-east-1:807711953667:endpoint/pytorch-inference-2023-03-15-19-36-39-451	3/15/2023, 11:36:40 PM	✓ InService	3/15/2023, 11:39:05 PM
○	pytorch-inference-2023-03-15-18-32-04-359	arn:aws:sagemaker:us-east-1:807711953667:endpoint/pytorch-inference-2023-03-15-18-32-04-359	3/15/2023, 10:32:05 PM	✓ InService	3/15/2023, 10:34:36 PM

Step 2: EC2 Training

The above model training can be carried out at a lower cost using EC2 instances.

To achieve this some code changes were made to the training script used in 'train_and_deploy-solution.ipynb' notebook to make it EC2 instance compliant.
As can be seen the EC2 instance does not support code from sagemaker modules which are inherently supported by a sagemaker instance.

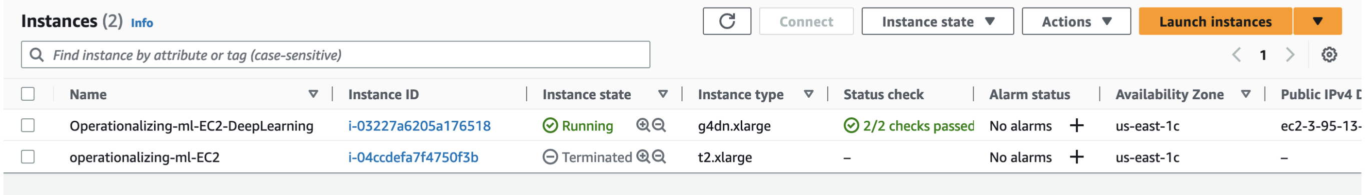
Thus, the sagemaker like modules boto3, sagemaker, sagemaker.tuner, sagemaker.pytorch, sagemaker.debbuger etc. are not used in solution.py model training script.

Thus, the sagemaker objects like 'HyperparameterTuner' for convenient and efficient hyperparameter tunning, 'PyTorch' for creating a model training estimator object, and PyTorchModel for deploying the model built by sagemaker aren't available to be used in the EC2 instance.

As a trade-off for low cost, EC2 instance codes are thereby less managed and involve more work.

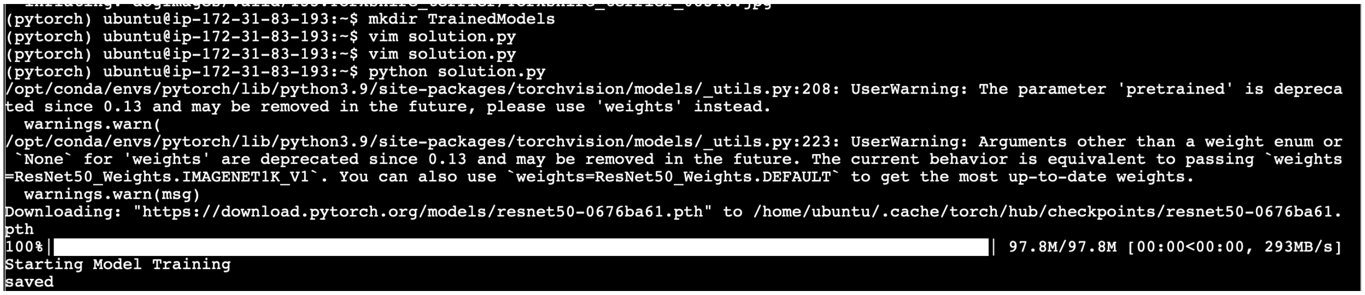
Besides that the training script used in both hpo.py (executed on sagemaker instance) and solution.py (executed on EC2 instance) are mostly same. Both perform transfer learning by fine tuning a pre-trained ResNet50 CNN model.

To train the model on EC2, an instance called 'Operationalizing-ml-EC2-DeepLearning' was created with 'Deep Learning AMI GPU PyTorch 1.13.1 (Ubuntu 20.04) 20230309' image and with instance type 'g4dn.xlarge'. This instance was chosen because the select AMI image only supported G3, P3, P3dn, P4d, G5, G4dn instance family type. Also, this instance has the same configuration (4 CPUs and 16 GiB Memory) like the 'ml.m5.xlarge' instance used in sagemaker to train the model in Step 1.



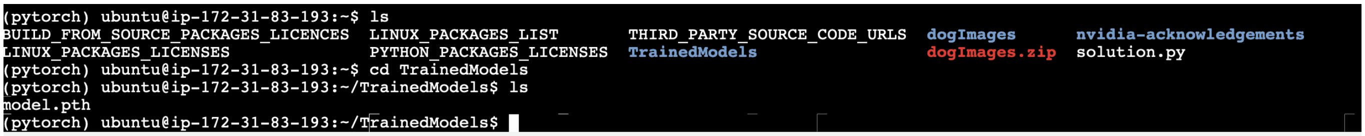
□	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 D
□	Operationalizing-ml-EC2-DeepLearning	i-03227a6205a176518	✓ Running	g4dn.xlarge	✓ 2/2 checks passed	No alarms +	us-east-1c	ec2-3-95-13-
□	operationalizing-ml-EC2	i-04ccdefa7f4750f3b	⊖ Terminated	t2.xlarge	–	No alarms +	us-east-1c	–

The following screenshot depicts the model training on EC2:



```
(pytorch) ubuntu@ip-172-31-83-193:~$ mkdir TrainedModels
(pytorch) ubuntu@ip-172-31-83-193:~$ vim solution.py
(pytorch) ubuntu@ip-172-31-83-193:~$ vim solution.py
(pytorch) ubuntu@ip-172-31-83-193:~$ python solution.py
/opt/conda/envs/pytorch/lib/python3.9/site-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is depreca
ted 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 'weights' 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 /home/ubuntu/.cache/torch/hub/checkpoints/resnet50-0676ba61.
pth
100%|██████████████████████████████████████████████████████████████| 97.8M/97.8M [00:00<00:00, 293MB/s]
Starting Model Training
saved
```

The final model 'model.pth' was saved by Pytorch at TrainedModel path.



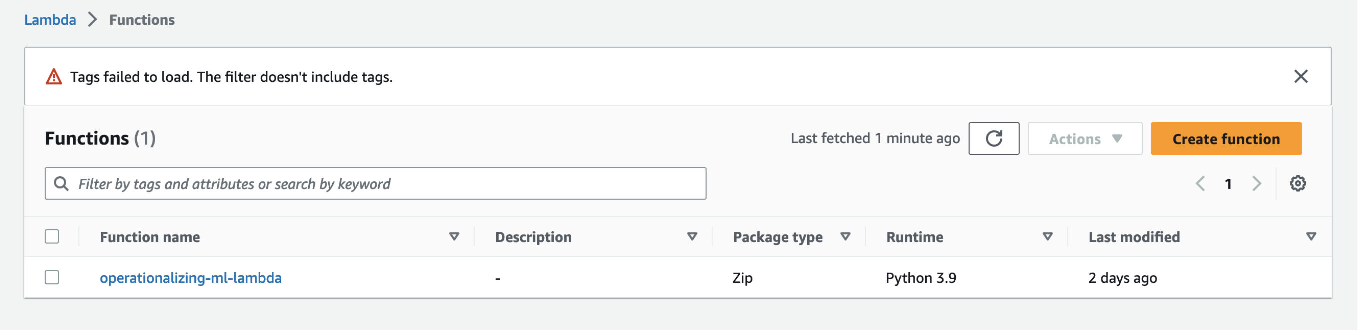
```
(pytorch) ubuntu@ip-172-31-83-193:~$ ls
BUILD_FROM_SOURCE_PACKAGES_LICENCES  LINUX_PACKAGES_LIST       THIRD_PARTY_SOURCE_CODE_URLS  dogImages      nvidia-acknowledgements
LINUX_PACKAGES_LICENSES              PYTHON_PACKAGES_LICENSES  TrainedModels                 dogImages.zip  solution.py
(pytorch) ubuntu@ip-172-31-83-193:~$ cd TrainedModels
(pytorch) ubuntu@ip-172-31-83-193:~/TrainedModels$ ls
model.pth
(pytorch) ubuntu@ip-172-31-83-193:~/TrainedModels$
```

Step 3: Lambda function setup

To make the endpoint 'pytorch-inference-2023-03-15-18-32-04-359' accessible to the end users for running inferences a Lambda function with name 'operationalizing-ml-lambda' and python 3.9 is setup.

The lambda function is setup to receive an image url in json format. It sends this information to the endpoint by calling the invoke_endpoint() method on sagemaker runtime client object. On succeful invocation the enpoint returns a list of 133 predictions indicating the likelihood of the dog in the image belonging to one of the 133 dog breed classes. The lambda function returns this information from the endpoint in the response body with a 200 response status code.

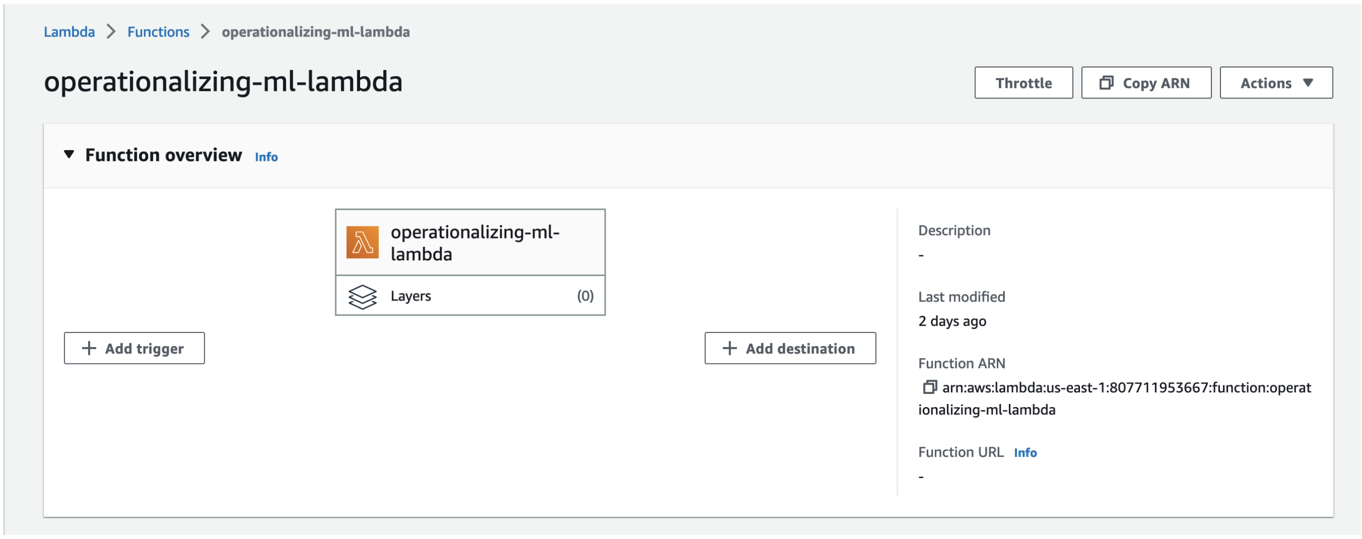
Lambdafunction.py contains the entire lambda function code.


Lambda > Functions
Last fetched 1 minute ago
Actions ▼
Create function
Filter by tags and attributes or search by keyword
< 1 >

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

Functions (1)

□	Function name	Description	Package type	Runtime	Last modified
□	operationalizing-ml-lambda	-	Zip	Python 3.9	2 days ago


Lambda > Functions > operationalizing-ml-lambda
Throttle
Copy ARN
Actions ▼
Info
operationalizing-ml-lambda
Layers
(0)
+ Add trigger
+ Add destination
Info

operationalizing-ml-lambda

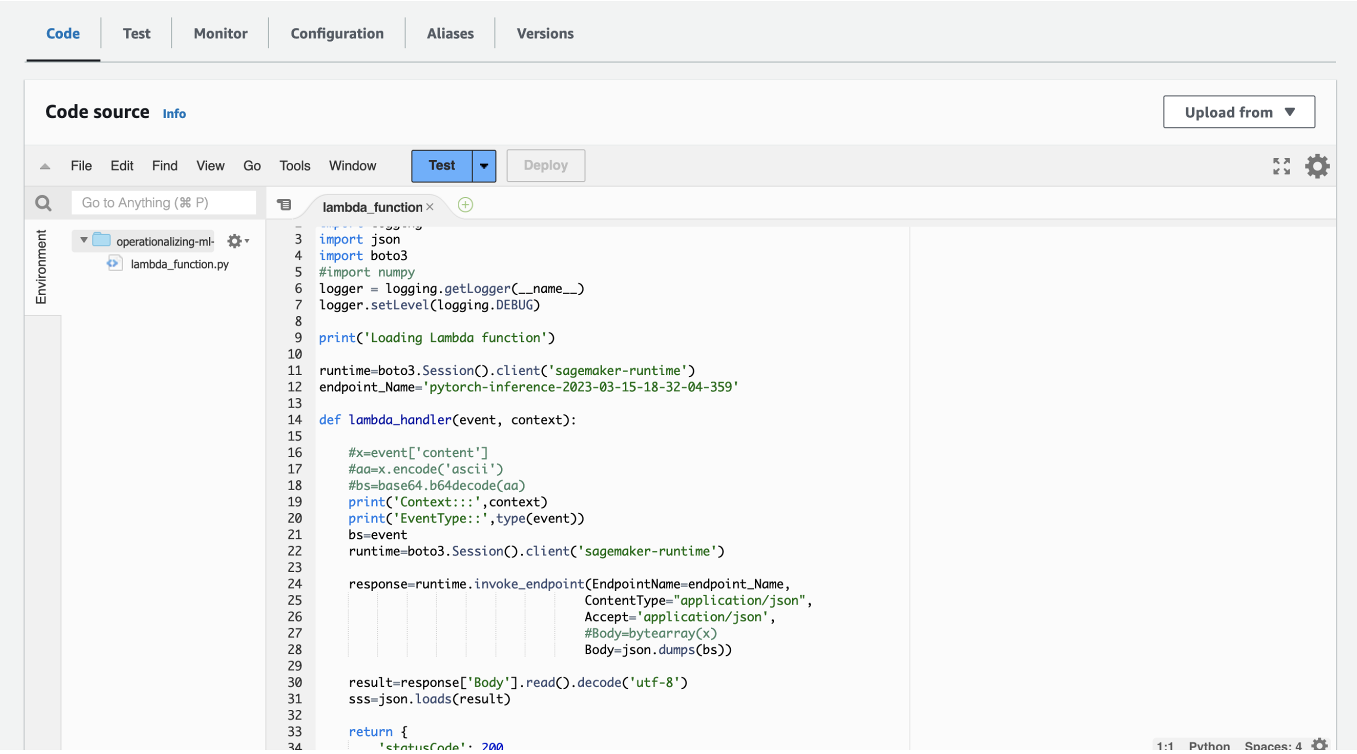
Function overview

Description
-

Last modified
2 days ago

Function ARN
arn:aws:lambda:us-east-1:807711953667:function:operationalizing-ml-lambda

Function URL
-



```python
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='pytorch-inference-2023-03-15-18-32-04-359'

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
```

Step 4: Security and Testing

'Operationalizing-ml-test' test event is configured as follows to test the functioning of Lambda function:

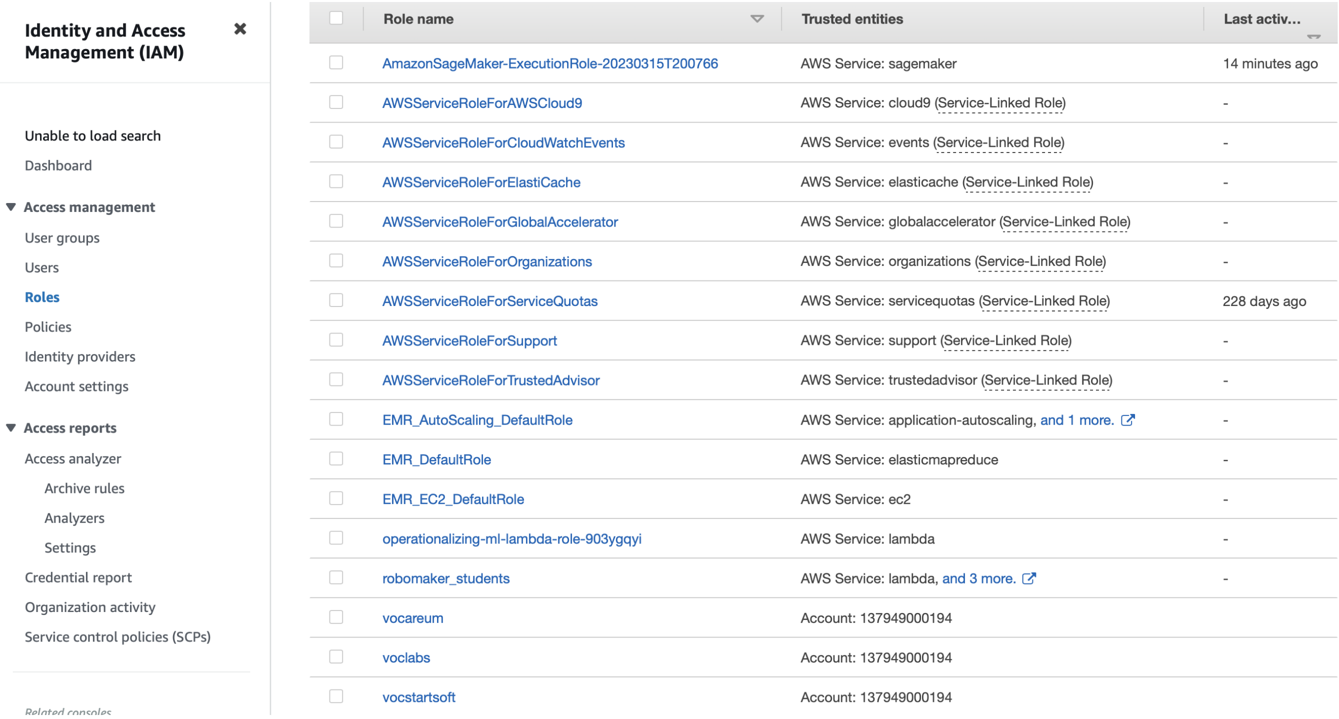
```
{ "url": "https://s3.amazonaws.com/cdn-origin-etr.akc.org/wp-content/uploads/2017/11/20113314/Carolina-Dog-standing-outdoors.jpg" }
```

On execution it gives the following error message as expected:

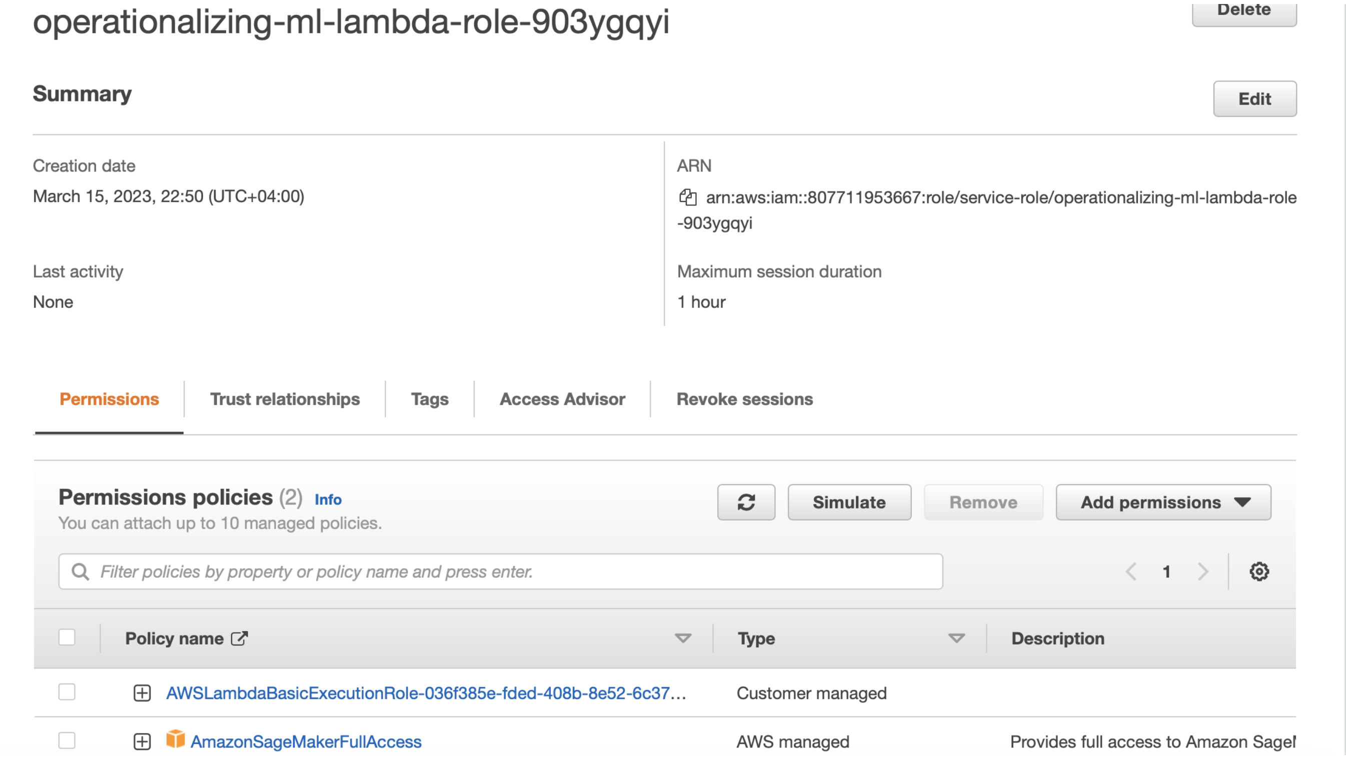
```
"errorMessage": "An error occurred (AccessDeniedException) when calling the InvokeEndpoint operation: User: arn:aws:sts::807711953667:assumed-role/operationalizing-ml-lambda-role-903ygqyi/operationalizing-ml-lambda is not authorized to perform: sagemaker:InvokeEndpoint on resource: arn:aws:sagemaker:us-east-1:807711953667:endpoint/pytorch-inference-2023-03-15-18-32-04-359 because no identity-based policy allows the sagemaker:InvokeEndpoint action"
```

This message ensures that the lambda function does not have access to S3 objects unless explicitly granted with the necessary access.

To grant the same, AmazonSagemakerFullAccess policy is attached to the 'operationalizing-ml-lambda-role-903ygqyi' as seen below:



□	Role name	Trusted entities	Last activ...
□	AmazonSageMaker-ExecutionRole-20230315T200766	AWS Service: sagemaker	14 minutes ago
□	AWSServiceRoleForAWSCloud9	AWS Service: cloud9 (Service-Linked Role)	-
□	AWSServiceRoleForCloudWatchEvents	AWS Service: events (Service-Linked Role)	-
□	AWSServiceRoleForElastiCache	AWS Service: elasticache (Service-Linked Role)	-
□	AWSServiceRoleForGlobalAccelerator	AWS Service: globalaccelerator (Service-Linked Role)	-
□	AWSServiceRoleForOrganizations	AWS Service: organizations (Service-Linked Role)	-
□	AWSServiceRoleForServiceQuotas	AWS Service: servicequotas (Service-Linked Role)	228 days ago
□	AWSServiceRoleForSupport	AWS Service: support (Service-Linked Role)	-
□	AWSServiceRoleForTrustedAdvisor	AWS Service: trustedadvisor (Service-Linked Role)	-
□	EMR_AutoScaling_DefaultRole	AWS Service: application-autoscaling, and 1 more.	-
□	EMR_DefaultRole	AWS Service: elasticmapreduce	-
□	EMR_EC2_DefaultRole	AWS Service: ec2	-
□	operationalizing-ml-lambda-role-903ygqyi	AWS Service: lambda	-
□	robomaker_students	AWS Service: lambda, and 3 more.	-
□	vocareum	Account: 137949000194	
□	voclabs	Account: 137949000194	
□	vocstartsoft	Account: 137949000194	



operationalizing-ml-lambda-role-903ygqyi

Creation date
March 15, 2023, 22:50 (UTC+04:00)

ARN
arn:aws:iam::807711953667:role/service-role/operationalizing-ml-lambda-role-903ygqyi

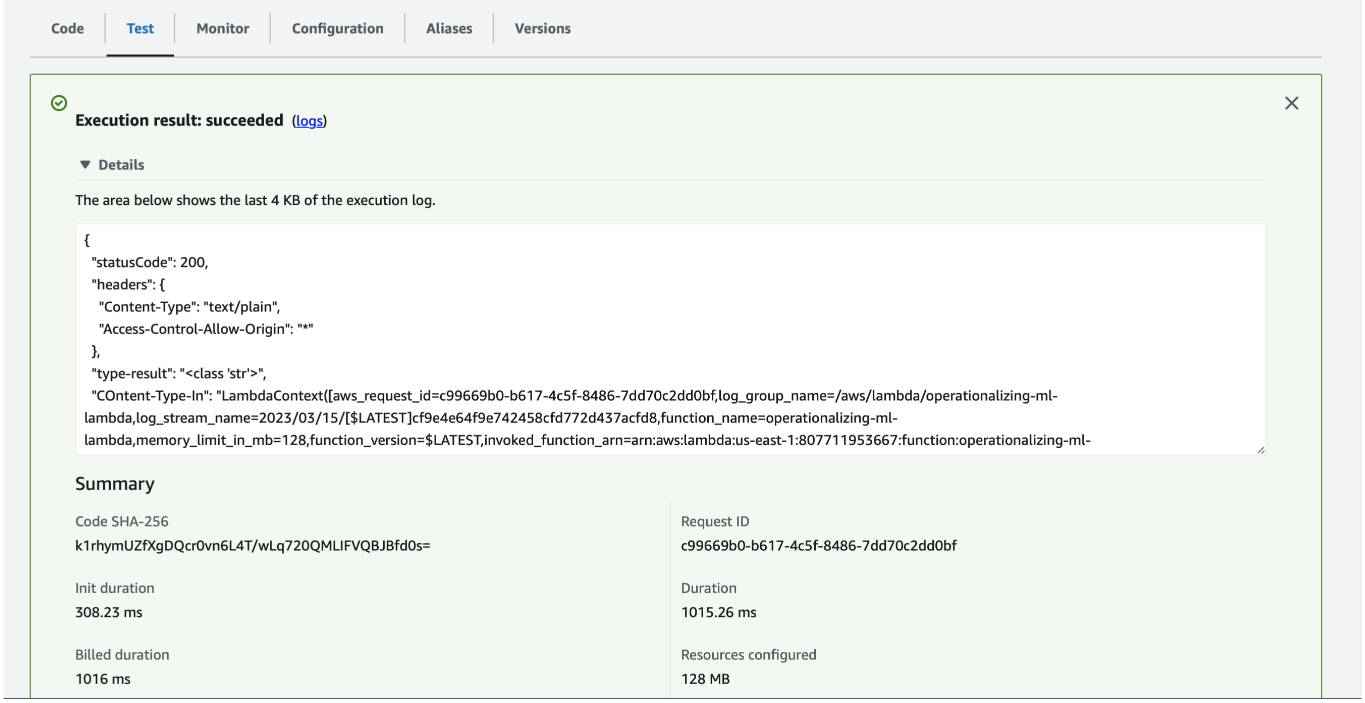
Last activity
None

Maximum session duration
1 hour

Permissions
Trust relationships
Tags
Access Advisor
Revoke sessions
Permissions policies (2) Info
You can attach up to 10 managed policies.
Simulate
Remove
Add permissions
Filter policies by property or policy name and press enter.
1

□	Policy name	Type	Description
□	AWSLambdaBasicExecutionRole-036f385e-fded-408b-8e52-6c37…	Customer managed	
□	AmazonSageMakerFullAccess	AWS managed	Provides full access to Amazon SageI

On execution, after extending the permission policies of the role, the test event executes with a successful response as expected:



Execution result: succeeded (logs)

▼ Details

The area below shows the last 4 KB of the execution log.

```
{
  "statusCode": 200,
  "headers": {
    "Content-Type": "text/plain",
    "Access-Control-Allow-Origin": "*"
  },
  "type-result": "<class 'str'>",
  "COntent-Type-In": "LambdaContext([aws_request_id=c99669b0-b617-4c5f-8486-7dd70c2dd0bf,log_group_name=/aws/lambda/operationalizing-ml-
lambda,log_stream_name=2023/03/15/[$LATEST]cf9e4e64f9e742458cfd772d437acfd8,function_name=operationalizing-ml-
lambda,memory_limit_in_mb=128,function_version=$LATEST,invoked_function_arn=arn:aws:lambda:us-east-1:807711953667:function:operationalizing-ml-
```

Summary

Code SHA-256 k1rhymUZfXgDQcr0vn6L4T/wLq720QMLIFVQBJBfd0s=	Request ID c99669b0-b617-4c5f-8486-7dd70c2dd0bf
Init duration 308.23 ms	Duration 1015.26 ms
Billed duration 1016 ms	Resources configured 128 MB

```
"body": "[[-8.549363136291504, -5.614318370819092, -5.197885513305664,
-1.690418004989624, -6.251997470855713, -6.444180011749268, -3.172017812728882,
-2.0982232093811035, -7.4178643226623535, -2.4237399101257324,
-2.4068186283111572, -6.362928867340088, -4.442510604858398,
-1.8487855195999146, -5.968094825744629, -6.309549331665039, -9.503728866577148,
-4.92889928817749, -5.832282543182373, 1.0415486097335815, -6.517759799957275,
-1.4663174152374268, -10.380915641784668, -7.659205913543701,
-7.613673686981201, -7.488474369049072, -3.9692068099975586, -6.155353546142578,
-10.148274421691895, -3.1330666542053223, -4.8581013679504395,
-4.434658050537109, -8.248491287231445, -2.431837558746338, -8.916293144226074,
-7.780113697052002, -5.135960102081299, -6.0632195472717285, -2.544854164123535,
-4.793948650360107, -5.059201240539551, -5.231955528259277, -1.7597802877426147,
-3.5927388668060303, -3.5185933113098145, -10.696097373962402,
-2.6491823196411133, -1.2767730951309204, -2.8555235862731934,
-3.1387343406677246, -5.158836364746094, -6.059427261352539, -8.342958450317383,
-4.9864702224731445, -8.660323143005371, 0.28339412808418274, -5.25429630279541,
-8.431004524230957, -3.9614527225494385, -3.484525680541992, -8.365265846252441,
-5.747337341308594, -9.701278686523438, -7.524048805236816, -6.1441450119018555,
-10.135348320007324, -2.351712942123413, -9.876770973205566,
-1.5957671403884888, -2.385610580444336, -0.18273085355758667,
-4.415204048156738, -6.40791654586792, -4.878027439117432, -6.22317361831665,
-6.677184104919434, -10.009824752807617, -3.4748613834381104,
-7.388906002044678, -4.416584014892578, -1.0663141012191772, -7.193986415863037,
0.19556310772895813, -3.7093749046325684, -8.09249210357666, -5.605825901031494,
-2.567988157272339, -8.55727481842041, -7.2055792808532715, -2.828704833984375,
-11.929056167602539, -6.271511554718018, -9.331279754638672, -6.886322975158691,
-6.251905918121338, -4.8519392013549805, -4.266046047210693, -7.650031566619873,
-9.228922843933105, -5.981687545776367, -6.750699520111084, -4.380775451660156,
-4.505864143371582, -5.021382808685303, -4.954136848449707, -10.334794998168945,
-5.173328876495361, 0.09145405888557434, -2.4960107803344727,
-1.0954514741897583, -4.555173397064209, -3.3644869327545166,
-7.519066333770752, -7.942852973937988, -5.404903411865234, -3.3457705974578857,
-9.471820831298828, -1.4199553728103638, -5.766317844390869,
-0.20906755328178406, -3.5378124713897705, -4.121579170227051,
-5.855740547180176, -6.884402275085449, -8.63366985321045, -4.11212682723999,
-4.414002418518066, -2.652315855026245, -7.142421245574951, -7.874303340911865,
-8.272895812988281, -1.9525110721588135, -4.833139896392822]]"
```

As seen from the above values, '1.0415486097335815' is the highest likelihood value from the array at index 19. The dog is thereby incorrectly predicted to belong to category 20 (index +1 as the indexes start from 0) in the dataset i.e. a 'Belgian malinois'

Thus, the model needs further training and fine tuning to improve the prediction accuracy.

There are no deprecated roles seen in the IAM dashboard. The only two roles using FullAccess policy and which can pose a threat due to their permisive nature are **AmazonSageMaker-ExecutionRole-20230315T200766** and **operationalizing-ml-lambda-role-903ygqyi**. Both roles have AmazonSagemakerFullAccess policy attached to it.

Extra care needs to be taken with the role **operationalizing-ml-lambda-role-903ygqyi** attached to lambda function accessible by an end user. The **AmazonSagemakerFullAccess** policy was attached to provide access to the deployed endpoint. The other sagemaker policies need to be explored to see if the endpoint can be invoked without full access to reduce security challenges.

Step 5: Concurrency and Auto-Scaling

At times of peak traffic the model might deal with high latency issues (i.e. response time) for providing the inference. This can discourage users from using the model for receiving predictions and can lead to loss of customers and profit for the business owner.

Thus, it is essential to monitor for latency issues in the pipeline and address the bottlenecks in the ML pipeline (if any) proactively.

Concurrency

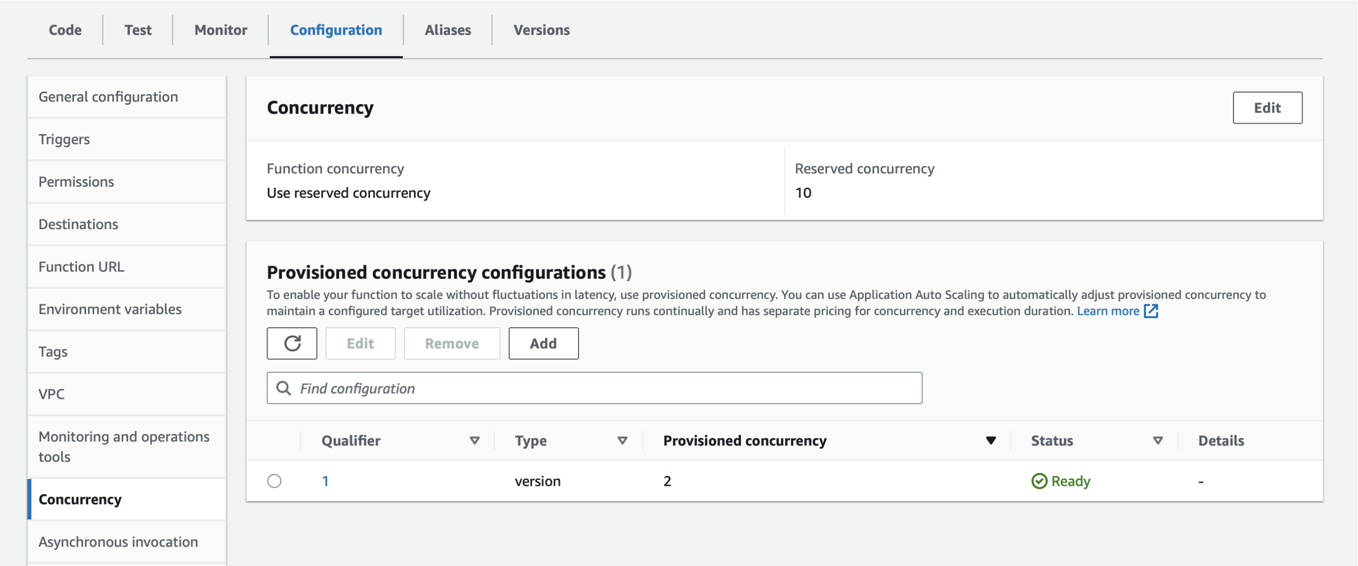
If the bottleneck is noticed in the lambda function i.e. if the lambda function has some complex computations and causing the responses to slowdown then the lambda function can be configured to concurrently execute in multiple instances. This is referred to as enabling concurrency for lambda functions.

AWS offers two approaches to configure for concurrency as follows:

1. Reserved concurrency - This option guarantees the requested amount of instances to the lambda function. This will be reserved from the available unreserved concurrency i.e. the maximum concurrency allowed per region.

   This option incurs no charge unless the instances are utilized.

2. Provisioned concurrency - On the other hand, provisioned concurrency initializes the number of instances as soon as configured and ensures it is available to respond immediately. So, it is important to configure this option wisely taking anticipated traffic and budget constraints into consideration. Also, the number of instances chosen for provisioned concurrency cannot exceed the number of reserved instances for this function.



Concurrency

Function concurrency
Use reserved concurrency

Reserved concurrency
10

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

	Qualifier	Type	Provisioned concurrency	Status	Details
○	1	version	2	✓ Ready	-

As seen in the screenshot, 10 instances are reserved for the dogs breed classification model. This means the lambda function can execute concurrently, without any interruptions in 10 instances. Since, no charges are incurred at setup this number was chosen.

Also, only 2 of those instances were provisioned because this incurs cost right from setup. A $2.47 will be charged monthly on the AWS account, since this falls within the project budget.

This number was chosen randomly for demonstration purposes. In real-world, the throughput needs to be monitored. If the volume of requests received is low then this configuration might not be required at all and we can stick to a single instance. If the number of requests doubles at some point in future and the response time exceeds the agreed upon response time in the SLA for the project then the number of instances can be doubled as shown.

Also, it is important to re-test and ensure successful execution of the lambda function after configuring concurrency.

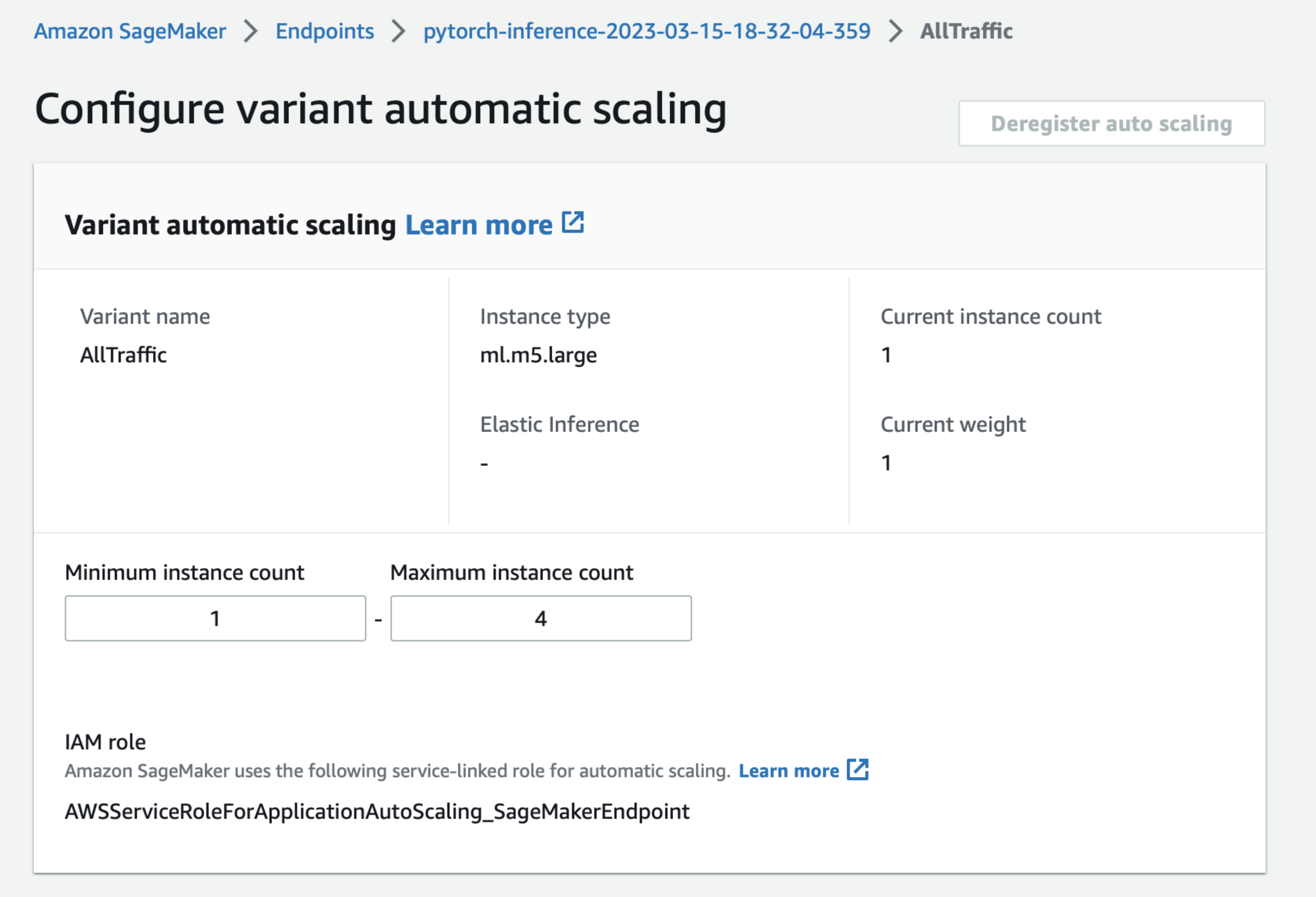
Note: Before setting up provisioned concurrency it is required to setup the version number for the lambda function.

Auto-Scaling

If the bottleneck is noticed in the endpoint i.e. the response time is large after the request is made to the endpoint then it might be better to perform auto-scaling of the endpoints.

The four main configurations for auto-scaling are explained below:

1. Maximum instance count - This specifies the maximum number of instances AWS can create to handle traffic at peak hours with low latency.
2. Target value - AWS will create new instances when the endpoint receives this many simultaneous requests.
3. Scale in cool down - Refers to the amount of seconds AWS will wait to create a new instance.
4. Scale out cool down - Refers to the amount of seconds AWS will wait to terminate an instance when the simultaneous traffic reduces below the target value.



Amazon SageMaker > Endpoints > pytorch-inference-2023-03-15-18-32-04-359 > AllTraffic

Configure variant automatic scaling

Variant automatic scaling Learn more

Variant name
AllTraffic

Instance type
ml.m5.large

Elastic Inference
-

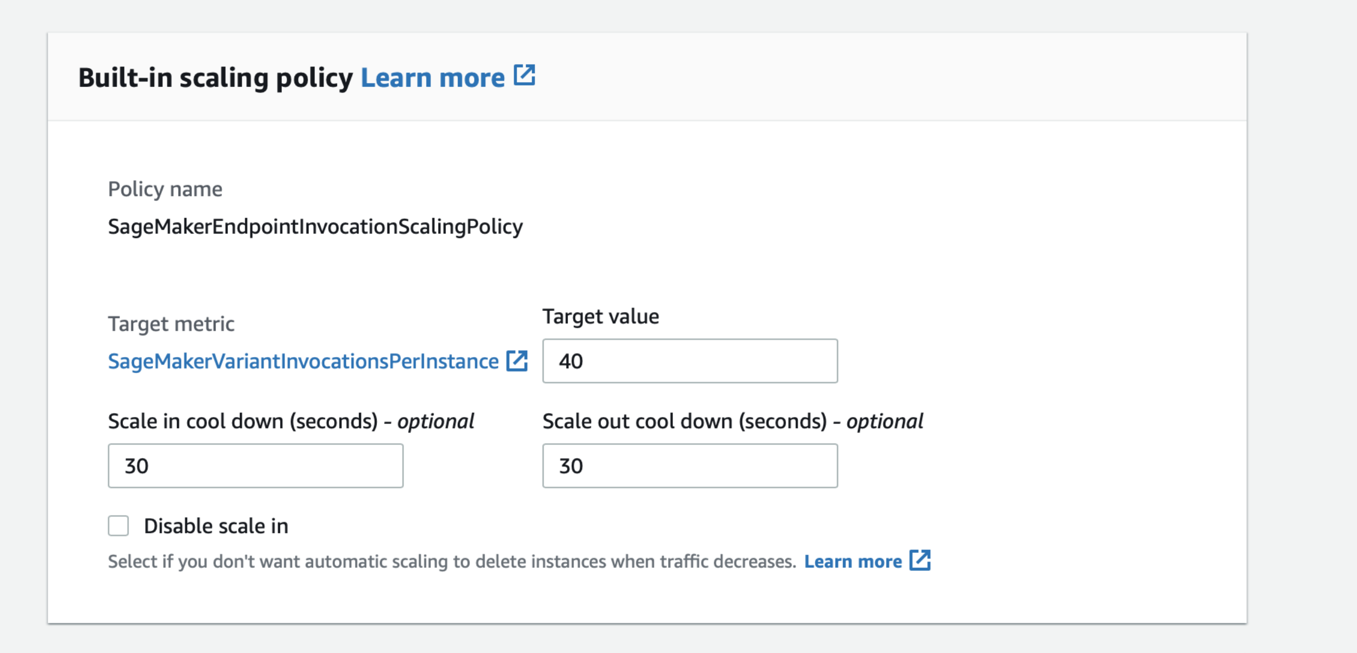
Current instance count
1

Current weight
1

Minimum instance count: 1 - Maximum instance count: 4

IAM role
Amazon SageMaker uses the following service-linked role for automatic scaling. **Learn more**

AWSServiceRoleForApplicationAutoScaling_SageMakerEndpoint



Built-in scaling policy Learn more

Policy name
SageMakerEndpointInvocationScalingPolicy

Target metric
SageMakerVariantInvocationsPerInstance

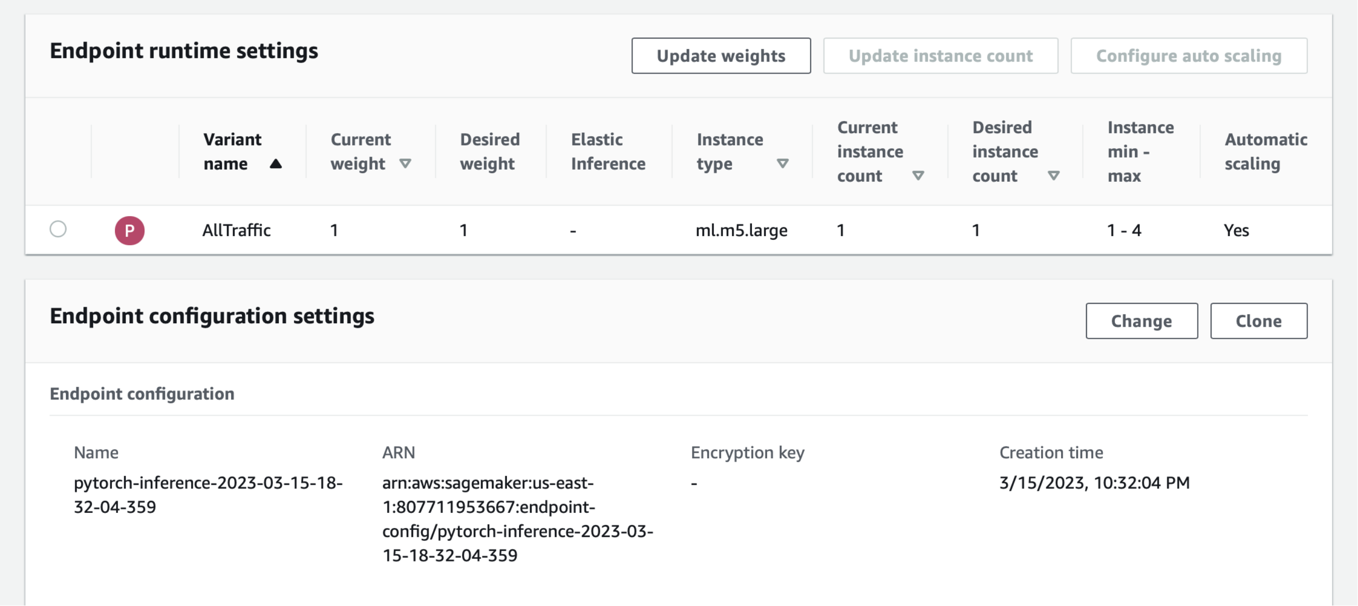
Target value
40

Scale in cool down (seconds) - *optional*
30

Scale out cool down (seconds) - *optional*
30

□ Disable scale in

Select if you don't want automatic scaling to delete instances when traffic decreases. **Learn more**



Endpoint runtime settings

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

Endpoint configuration settings

Endpoint configuration

Name	ARN	Encryption key	Creation time
pytorch-inference-2023-03-15-18-32-04-359	arn:aws:sagemaker:us-east-1:807711953667:endpoint-config/pytorch-inference-2023-03-15-18-32-04-359	-	3/15/2023, 10:32:04 PM

The above auto-scaling configurations were made for the deployed endpoint trained and deployed in Step 1.

The following configurations were made:

1. Maximum instance count - 4
2. Target value - 40
3. Scale in cool down - 30 seconds
4. Scale out cool down - 30 seconds

The endpoint can scale upto 4 instances to handle peak trafiic hours and is configured to be very responsive (only 30 secs wait time) to increased request volume as well as very responsive to decreased traffic inorder to save on AWS charges.

Conclusion

As detailed in this report, many complex challenges faced in production like managing availability, security, efficiency etc. can be easily overcome using the AWS services with minimal and intuitive configurations.

However, operationalizing industrial ML projects require a lot of thought and careful considerations before implementing the configurations.