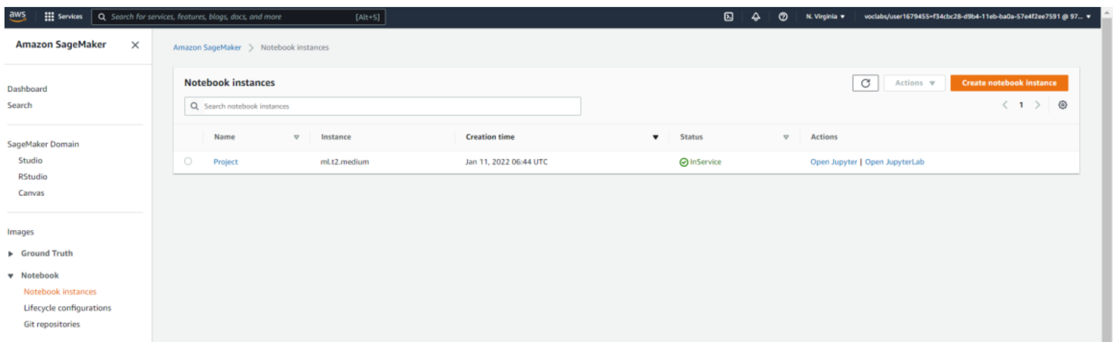
Operationalizing Machine Learning on SageMaker

-Mansi Chilwant

Initial Setup

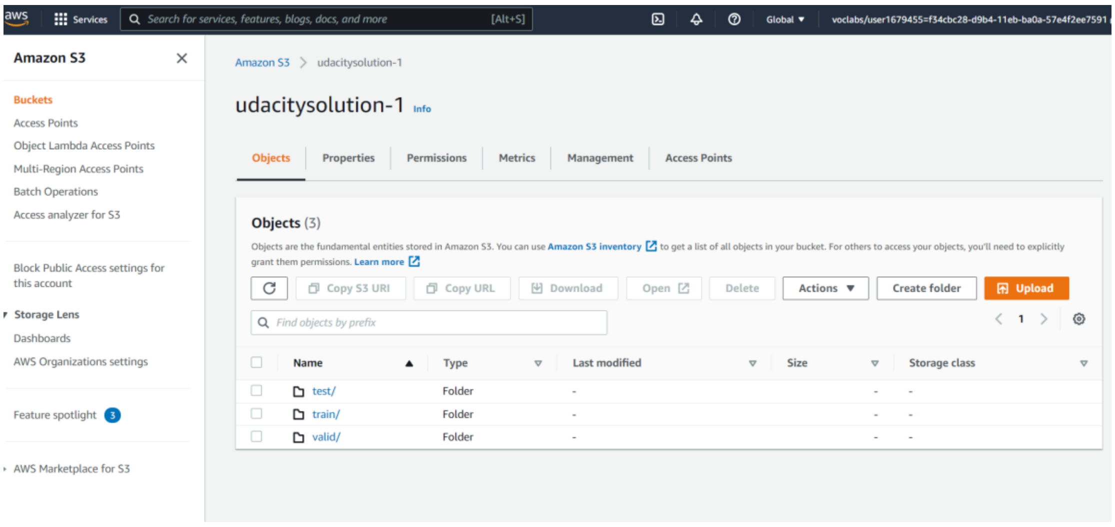
I opted ml.t2.medium (Figure: Sage Maker Instance) as the smallest SageMaker instance accessible for my notebook because I'll be working on the project for more hours and don't require a particularly strong instance in terms of CPU or RAM.



Notebook instances

	Name	Instance	Creation time	Status	Actions
○	Project	ml.t2.medium	Jan 11, 2022 06:44 UTC	InService	Open Jupyter | Open JupyterLab

Figure 1: SageMaker Instance



udacitysolution-1 Info

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	-	-	-

Figure 2: S3 Bucket Setup

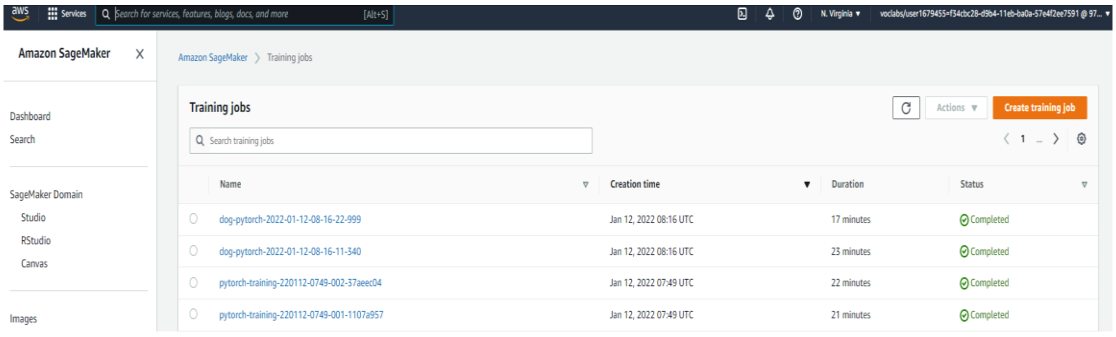
Furthermore, I chose ml.m5.2xlarge for both tuning and training since it has more processing power, allowing the tuning and training operations to be completed faster and avoiding memory issues that I had previously seen with this dataset.

I increased the number of tuning tasks to 12, the number of parallel jobs to 3, and the early stopping type to "Auto" to speed up tuning and guarantee that better hyperparameters were picked.

To allow dispersed training, I made the following changes to the hpo.py file, including changes to the parameters necessary to start the training.

Single instance jobs: 2 pytorch trainings

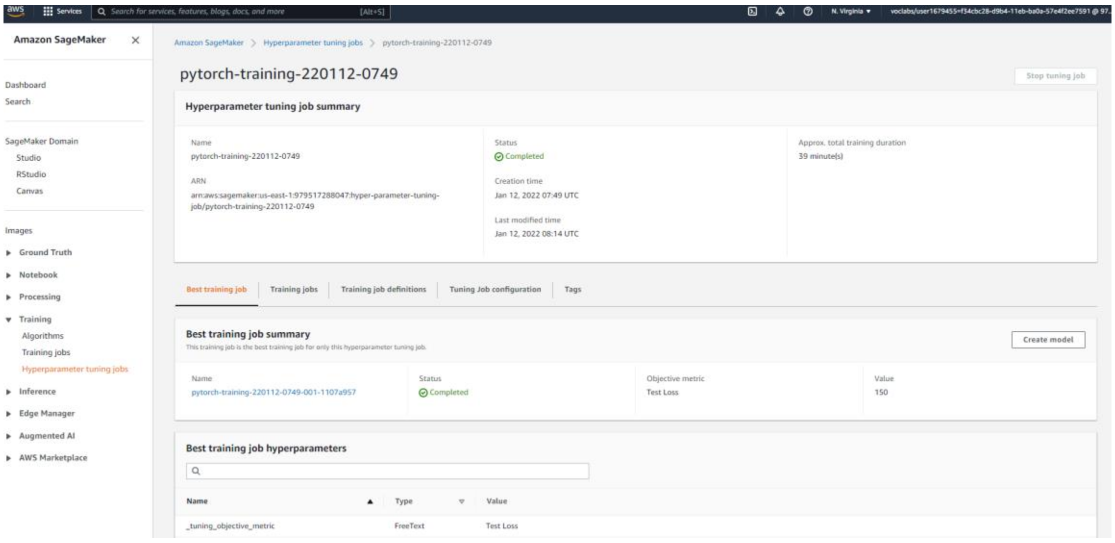
Multi instance jobs : 2 dog-pytorch trainings


Services
Search for services, features, blogs, docs, and more [Alt+S]
N. Virginia
voclabs/user1679455=f34cbc28-d9b4-11eb-ba0a-57e4f2ee7591 @ 97...
Amazon SageMaker
Dashboard
Search
SageMaker Domain
Studio
RStudio
Canvas
Images
Amazon SageMaker > Training jobs
Actions
Create training job
Search training jobs

Training jobs

Name	Creation time	Duration	Status
dog-pytorch-2022-01-12-08-16-22-999	Jan 12, 2022 08:16 UTC	17 minutes	Completed
dog-pytorch-2022-01-12-08-16-11-340	Jan 12, 2022 08:16 UTC	23 minutes	Completed
pytorch-training-220112-0749-002-37aeec04	Jan 12, 2022 07:49 UTC	22 minutes	Completed
pytorch-training-220112-0749-001-1107a957	Jan 12, 2022 07:49 UTC	21 minutes	Completed

Figure 3: Training jobs both single and multi



pytorch-training-220112-0749

Hyperparameter tuning job summary

Name: pytorch-training-220112-0749

ARN: arn:aws:sagemaker:us-east-1:979517288047:hyper-parameter-tuning-job/pytorch-training-220112-0749

Status: Completed

Creation time: Jan 12, 2022 07:49 UTC

Last modified time: Jan 12, 2022 08:14 UTC

Approx. total training duration: 39 minute(s)

Best training job summary

This training job is the best training job for only this hyperparameter tuning job.

Name	Status	Objective metric	Value
pytorch-training-220112-0749-001-1107a957	Completed	Test Loss	150

Best training job hyperparameters

Name	Type	Value
_tuning_objective_metric	FreeText	Test Loss

Figure 3: Best training job (single instance)

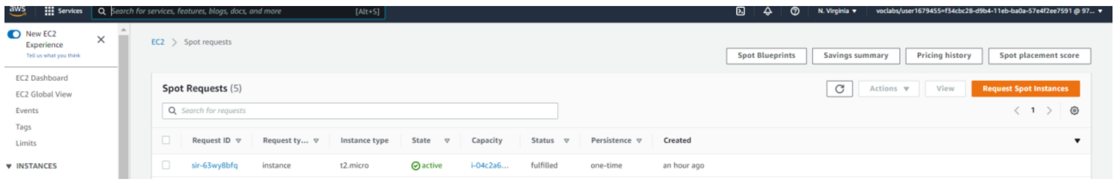
EC2 Training

To save money, I used the t2.x micro instance and the Deep Learning AMI (Amazon Linux 2) Version 57.0. This struck me as a good compromise between performance and price. Because two end points are active while executing EC2 training, it's only natural to employ a more powerful instance type than is necessary.

Similarly, because we don't know how long it will take to set up and debug this EC2 training component, we should pick a smaller instance so we don't have to pay for a huge instance while we're performing setup, debugging, and so on.

As you can see in my screenshot, I've restricted access to my EC2 instance to only my IP address in order to avoid the possibility of a security breach.

First the spot instance is created

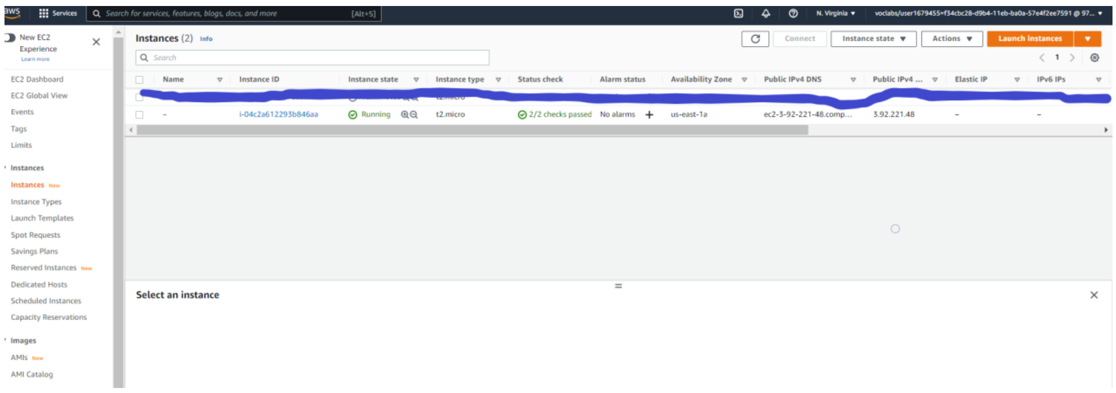


EC2 > Spot requests

Spot Requests (5)

Request ID	Request ty...	Instance type	State	Capacity	Status	Persistence	Created
sir-63wy8bfq	instance	t2.micro	active	i-04c2a6...	fulfilled	one-time	an hour ago

Figure 4: EC2 spot instance



Instances (2)

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP	IPv6 IPs
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
–	i-04c2a612293b846aa	Running	t2.micro	2/2 checks passed	No alarms	us-east-1a	ec2-3-92-221-48.comp...	3.92.221.48	–	–

Select an instance

Figure 5: EC2 Instance

Difference between ec2train1.py (EC2 script) and train_and_deploy

solution.ipynb+hpo.py (SageMaker scripts)

The biggest difference is that ec2train1.py doesn't have a main function, nor does it have the capabilities to handle argument parsing or optional main running. Similarly, ec2train1.py does not support multi-instance learning, like I implemented in my modified hpo.py. The most notable difference is that the ec2train1.py script uses test data to train, which means it uses a much smaller dataset and the same data for both training and testing.

This has been modified.

train_data =

torchvision.datasets.ImageFolder(root=test_data_path,

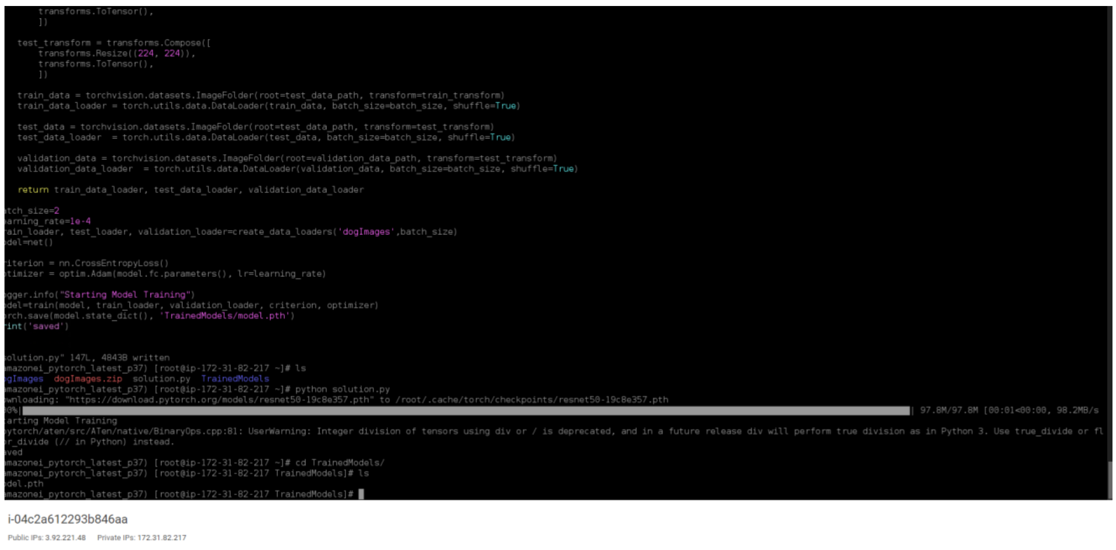
transform=train_transform)

To

train_data =

torchvision.datasets.ImageFolder(root=train_data_path,

transform=train_transform)



```
        transforms.ToTensor(),
        ])

    test_transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        ])

    train_data = torchvision.datasets.ImageFolder(root=test_data_path, transform=train_transform)
    train_data_loader = torch.utils.data.DataLoader(train_data, batch_size=batch_size, shuffle=True)

    test_data = torchvision.datasets.ImageFolder(root=test_data_path, transform=test_transform)
    test_data_loader  = torch.utils.data.DataLoader(test_data, batch_size=batch_size, shuffle=True)

    validation_data = torchvision.datasets.ImageFolder(root=validation_data_path, transform=test_transform)
    validation_data_loader  = torch.utils.data.DataLoader(validation_data, batch_size=batch_size, shuffle=True)

    return train_data_loader, test_data_loader, validation_data_loader

tch_size=2
arning_rate=1e-4
ain_loader, test_loader, validation_loader=create_data_loaders('dogImages',batch_size)
del=net()

iterion = nn.CrossEntropyLoss()
timizer = optim.Adam(model.fc.parameters(), lr=learning_rate)

gger.info("Starting Model Training")
del=train(model, train_loader, validation_loader, criterion, optimizer)
rch.save(model.state_dict(), 'TrainedModels/model.pth')
int('saved')


olution.py" 147L, 4843B written
mazonei_pytorch_latest_p37) [root@ip-172-31-82-217 ~]# ls
gImages  dogImages.zip  solution.py  TrainedModels
mazonei_pytorch_latest_p37) [root@ip-172-31-82-217 ~]# python solution.py
wnloading: "https://download.pytorch.org/models/resnet50-19c8e357.pth" to /root/.cache/torch/checkpoints/resnet50-19c8e357.pth
0%|████████████████████████████████████████████████████████████████████████████████| 97.8M/97.8M [00:01<00:00, 98.2MB/s
arting Model Training
ytorch/aten/src/ATen/native/BinaryOps.cpp:81: UserWarning: Integer division of tensors using div or / is deprecated, and in a future release div will perform true division as in Python 3. Use true_divide or fl
r_divide (// in Python) instead.
ved
mazonei_pytorch_latest_p37) [root@ip-172-31-82-217 ~]# cd TrainedModels/
mazonei_pytorch_latest_p37) [root@ip-172-31-82-217 TrainedModels]# ls
del.pth
mazonei_pytorch_latest_p37) [root@ip-172-31-82-217 TrainedModels]# 
```

i-04c2a612293b846aa

Public IPs: 3.92.221.48 Private IPs: 172.31.82.217

Figure 6: EC2 Training

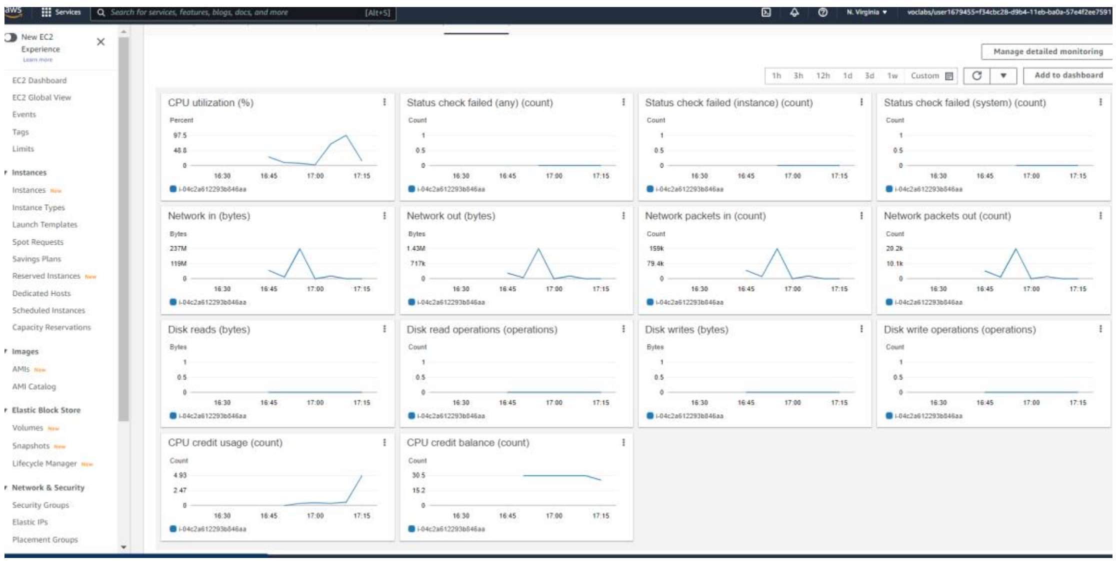


Figure 7: EC2 Training Resource Use

Lambda functions

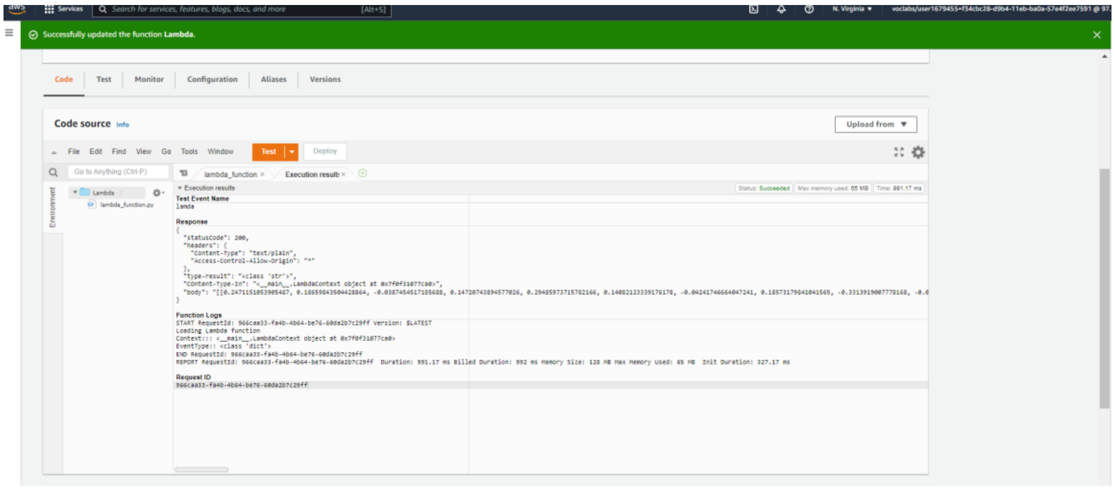
I created a Lambda function named dog-breed-prediction using the lambda function.py starter.

A Boto3 client is created using this function. This enables the function to communicate with other AWS services, such as SageMaker. The function utilises the client to contact the pytorch-inference-2022-01-12-08-25-47-451 endpoint from the multi-instance trained model. It expects a JSON-formatted input/image and returns a JSON. The prediction from the endpoint is in the key 'body' of this JSON, which has a status code of 200.

```
{

"errorMessage": "An error occurred (AccessDeniedException)

when calling the InvokeEndpoint operation: User:

arn:aws:sts::[redacted]

is not authorized to perform: sagemaker:InvokeEndpoint on

resource:

arn:aws:[redacted]

because no identity-based policy allows the

}
```

Permissions

I'm worried about the permissions we've granted these lambda functions because they don't seem to follow the least privilege principle. In a perfect world, we'd only allow these lambda functions to query the endpoints for which they were created. I'll have to investigate further to see if there's anything I can do about it. These lambda functions might also be used as a second line of defence against DOS attacks on
endpoints.


Successfully updated the function Lambda.

Code source

Test Event Name
lamda

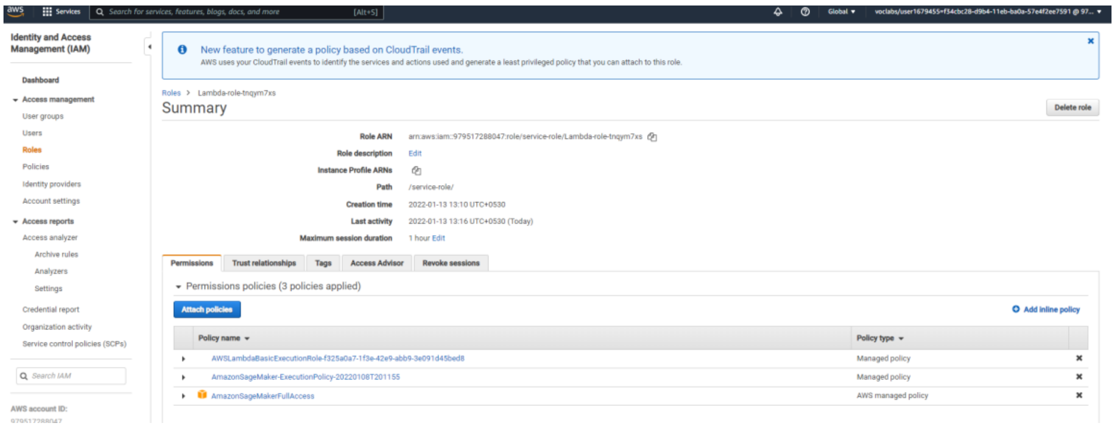
Response
```
{
  "statusCode": 200,
  "headers": {
    "Content-Type": "text/plain",
    "Access-Control-Allow-Origin": "*"
  },
  "type-result": "<class 'str'>",
  "Content-Type-In": "<__main__.LambdaContext object at 0x7f0f31077ca0>",
  "body": "[[0.2471151053905487, 0.18659843504428864, -0.0387454517185688, 0.14720743894577026, 0.29485973715782166, 0.14082123339176178, -0.04241746664047241, 0.18573179841041565, -0.3313919007778168, -0.0
}
```

Function Logs
```
START RequestId: 966caa33-fa4b-4b64-be76-60da2b7c29ff Version: $LATEST
Loading Lambda function
Context::: <__main__.LambdaContext object at 0x7f0f31077ca0>
EventType:: <class 'dict'>
END RequestId: 966caa33-fa4b-4b64-be76-60da2b7c29ff
REPORT RequestId: 966caa33-fa4b-4b64-be76-60da2b7c29ff  Duration: 991.17 ms Billed Duration: 992 ms Memory Size: 128 MB Max Memory Used: 65 MB  Init Duration: 327.17 ms
```

Request ID
966caa33-fa4b-4b64-be76-60da2b7c29ff

Status: Succeeded | Max memory used: 65 MB | Time: 991.17 ms

Figure 8: Lambda Functions success


New feature to generate a policy based on CloudTrail events.
AWS uses your CloudTrail events to identify the services and actions used and generate a least privileged policy that you can attach to this role.

Roles > Lambda-role-tnqym7xs

Summary

Role ARN	arn:aws:iam::979517288047:role/service-role/Lambda-role-tnqym7xs
Role description	Edit
Instance Profile ARNs	
Path	/service-role/
Creation time	2022-01-13 13:10 UTC+0530
Last activity	2022-01-13 13:16 UTC+0530 (Today)
Maximum session duration	1 hour Edit

Permissions policies (3 policies applied)

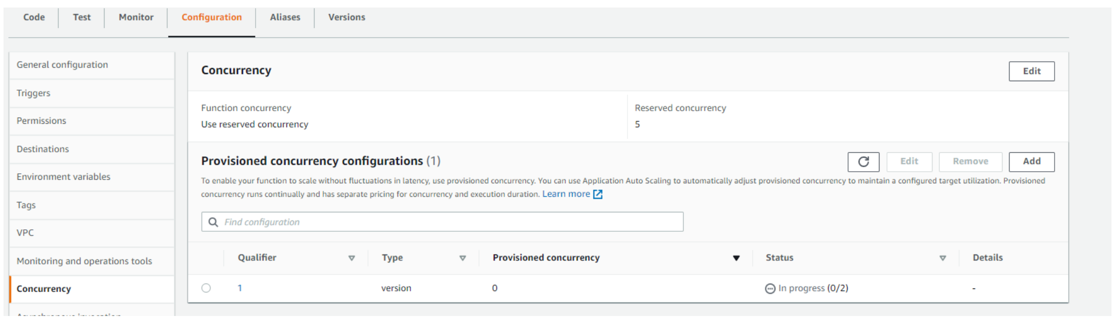
Policy name	Policy type
AWSLambdaBasicExecutionRole-f325a0a7-1f3e-42e9-abb9-3e091d45bed8	Managed policy
AmazonSageMaker-ExecutionPolicy-20220108T201155	Managed policy
AmazonSageMakerFullAccess	AWS managed policy

AWS account ID:
979517288047

Figure 9: Lambda Functions Role Update

Concurrency and auto-scaling

Because reserved concurrency is free and meets our present needs, I chose it over shared concurrency. Furthermore, we are unlikely to handle more than a few requests per endpoint instance at any given time, as this would indicate that the latter is overloaded, so a value that is a multiple of the number of endpoint instances makes sense, so I chose 100, which allows for 20 requests per endpoint instance. I choose to scale the endpoint to between one and five instances. While this is unlikely to ever be an issue, because each forecast takes about 0.25 seconds, this should allow for about 20 requests per second, which, along with appropriate request throttling, should allow for a large number of individuals to utilise the service.



Concurrency

Function concurrency
Use reserved concurrency

Reserved concurrency
5

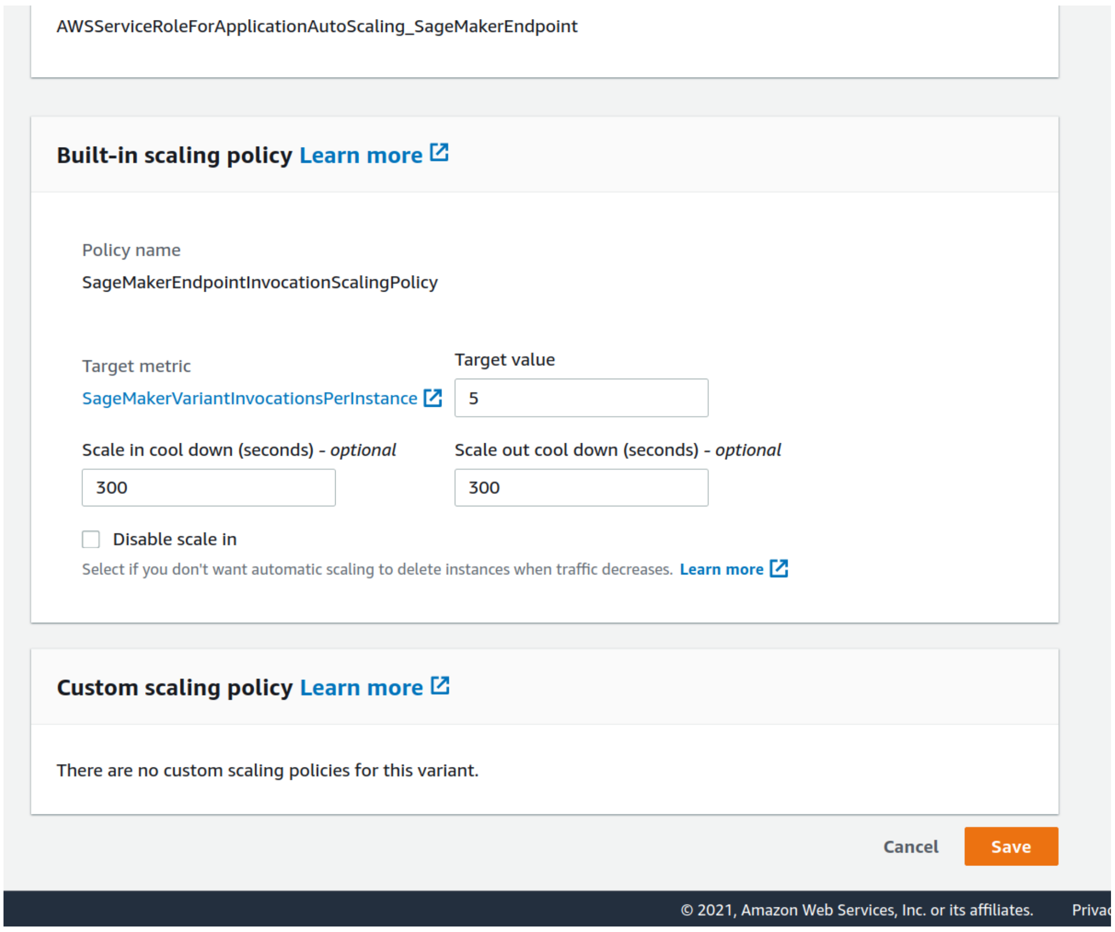
Provisioned concurrency configurations (1)

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.

Find configuration

Qualifier	Type	Provisioned concurrency	Status	Details
1	version	0	In progress (0/2)	-

Figure 10: Configuring Concurrency



AWSServiceRoleForApplicationAutoScaling_SageMakerEndpoint

Built-in scaling policy Learn more

Policy name

SageMakerEndpointInvocationScalingPolicy

Target metric	Target value
SageMakerVariantInvocationsPerInstance	5
Scale in cool down (seconds) - *optional*	Scale out cool down (seconds) - *optional*
300	300

☐ Disable scale in

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

Custom scaling policy Learn more

There are no custom scaling policies for this variant.

Figure 11: Configuring Concurrency