

Operationalising Machine Learning

Author: Mitchell Gray

Date: 2022-08-14

1. [Operationalising Machine Learning](#)
 1. [Training](#)
 1. [Notebook Size Selection](#)
 2. [Creation of S3 Bucket](#)
 3. [Single Instance Training](#)
 4. [Multi-Instance Training](#)
 2. [EC2 Training](#)
 1. [Instance Selection](#)
 2. [Code Differences](#)
 1. [Argument Parsing](#)
 2. [Sagemaker Use of Environment Variables](#)
 3. [Lambda Function Setup](#)
 1. [Code Modification](#)
 2. [Lambda Function Code](#)
 1. [General Structure](#)
 2. [Sagemaker Endpoint Invocation](#)
 3. [IAM Role configuration](#)
 4. [Lambda Function Test](#)
 4. [Concurrency & Autoscaling](#)
 1. [Concurrency](#)
 2. [Autoscaling](#)

Training

Notebook Size Selection

The project we are seeking to operationalise is a RESNET50 fine tuning for the purposes of predicting the breed of a dog depicted by an image.

However, close reading of the notebook describing the processes to be follows indicates that the processing that this notebook will need to do

will be limited to downloading the images at first instance and then triggering training jobs and deployments on other infrastructure.

As a result, we should select the smallest instance type possible.

Consulting the AWS prices for notebook instances we find that the cheapest is ml.t2.medium.

Standard Instances	vCPU	Memory	Price per Hour
ml.t3.medium	2	4 GiB	\$0.05
ml.t3.large	2	8 GiB	\$0.10
ml.t3.xlarge	4	16 GiB	\$0.20
ml.t3.2xlarge	8	32 GiB	\$0.399
ml.m5.large	2	8 GiB	\$0.115
ml.m5.xlarge	4	16 GiB	\$0.23
ml.m5.2xlarge	8	32 GiB	\$0.461
ml.m5.4xlarge	16	64 GiB	\$0.922
ml.m5.8xlarge	32	128 GiB	\$1.843
ml.m5.12xlarge	48	192 GiB	\$2.765
ml.m5.16xlarge	64	256 GiB	\$3.686
ml.m5.24xlarge	96	384 GiB	\$5.53
ml.m5d.large	2	8 GiB	\$0.136
ml.m5d.xlarge	4	16 GiB	\$0.271

AWS Notebook Prices

I selected this instance size for the working notebook for this project.

Notebook instances

Search notebook instances

Actions ▾

Create notebook instance

Name ▾	Instance	Creation time ▾	Status ▾	Actions
<div><div></div><div>operationalizing-notebook</div></div>	ml.t3.medium	Aug 13, 2022 08:56 UTC	<div><div></div><div>InService</div></div>	Open Jupyter Open JupyterLab

Creation of S3 Bucket

In order to hold the data for this project, I created an s3 bucket 'mg-operationalise-udacity-solution'

Buckets (1) Info		Refresh	Copy ARN	Empty	Delete	Create bucket
Buckets are containers for data stored in S3. Learn more						
<input type="text" value="Find buckets by name"/> < 1 >						
Name	AWS Region	Access	Creation date			
mg-operationalise-udacity-solution	US East (N. Virginia) us-east-1	Bucket and objects not public	August 13, 2022, 18:42:48 (UTC+10:00)			

Single Instance Training

The single instance training was completed on the notebook and the resulting endpoint was:

Endpoints		Refresh	Update endpoint	Actions	Create endpoint
<input type="text" value="Search endpoints"/> < 1 >					
Name	ARN	Creation time	Status	Last updated	
pytorch-inference-2022-08-13-11-37-34-082	arn:aws:sagemaker:us-east-1:566372991438:endpoint/pytorch-inference-2022-08-13-11-37-34-082	Aug 13, 2022 11:37 UTC	InService	Aug 13, 2022 11:39 UTC	

Output of the Single Training Endpoint

Multi-Instance Training

After completing the single instance training exercise, I altered the code block to conduct multi-instance training. The code used to do this is in the .ipynb file in this repo. The endpoint produced was:

Endpoints		Refresh	Update endpoint	Actions	Create endpoint
<input type="text" value="Search endpoints"/> < 1 >					
Name	ARN	Creation time	Status	Last updated	
pytorch-inference-2022-08-13-12-29-03-528	arn:aws:sagemaker:us-east-1:566372991438:endpoint/pytorch-inference-2022-08-13-12-29-03-528	Aug 13, 2022 12:29 UTC	InService	Aug 13, 2022 12:31 UTC	

EC2 Training

Instance Selection

The code used for training on the EC2 image is not designed to run on a GPU. While it would be possible to alter the code to do this, it didn't make sense to do so if acceptable performance could be achieved without doing so. However, I elected to look at compute optimised instances that had enough memory for my application but which were not too expensive. This workflow will fit relatively comfortably within 8GB of memory. Therefore the `c4.xlarge` instance seemed appropriate.

Instance	vCPU*	Mem (GiB)	Storage	Dedicated EBS Bandwidth (Mbps)	Network Performance
c4.large	2	3.75	EBS-Only	500	Moderate
c4.xlarge	4	7.5	EBS-Only	750	High
c4.2xlarge	8	15	EBS-Only	1,000	High
c4.4xlarge	16	30	EBS-Only	2,000	High
c4.8xlarge	36	60	EBS-Only	4,000	10 Gigabit

Upon looking up the prices for this instance, I found that the price was relatively cheap at \$0.17 per hour in the US-East1 SageMaker region:

<input type="text" value="c5.xlarge"/>					
Instance name ▲	On-Demand hourly rate ▼	vCPU ▼	Memory ▼	Storage ▼	Network performance ▼
c5.xlarge	\$0.17	4	8 GiB	EBS Only	Up to 10 Gigabit

I elected to start up this instance and configure it using an image which provided access to Pytorch - in this case Deep Learning AMI GPU Pytorch 1.12.0.

Instances (1)

Info

Q

Search

Connect

Instance state

Actions

Launch instances

Name

Instance ID

Instance state

Instance type

Status check

Alarm status

Availability Zone

Public IPv4 DNS

Public IPv4 ...

Elastic IP

udacity-operat...

i-Ofa2322a6a5681eb6

Running

c4.xlarge

2/2 checks passed

No alarms

us-east-1b

ec2-34-229-153-223.co...

34.229.153.223

-

1

I configured this instance to only accept ssh connections from my home computer which I then used to remote into the machine and complete the training job:

```
[ec2-user@ip-172-31-29-57 ~]$ exit
logout
Connection to ec2-34-229-153-223.compute-1.amazonaws.com closed.
mitchell@beverley-ubuntu:~$ ssh -i ./Documents/udacity-operationalise/operationalise-key-pair.pem ec2-u
ser@ec2-34-229-153-223.compute-1.amazonaws.com
=====
      _|_ |_) 
      _| ( /   Deep Learning AMI GPU PyTorch 1.12.0 (Amazon Linux 2)
      ---|\---|---|
=====

* To activate pre-built pytorch environment, run: 'source activate pytorch'
* To activate base conda environment upon login, run: 'conda config --set auto_activate_base true'
* NVIDIA driver version: 510.73.08
* CUDA version: 11.6

AWS Deep Learning AMI Homepage: https://aws.amazon.com/machine-learning/amis/
Release Notes: https://docs.aws.amazon.com/dlami/latest/devguide/appendix-ami-release-notes.html
Support: https://forums.aws.amazon.com/forum.jspa?forumID=263
For a fully managed experience, check out Amazon SageMaker at https://aws.amazon.com/sagemaker
Security scan reports for python packages are located at: /opt/aws/dlami/info/
=====
(base) [ec2-user@ip-172-31-29-57 ~]$ conda activate pytorch_p39
Could not find conda environment: pytorch_p39
You can list all discoverable environments with `conda info --envs`.

(base) [ec2-user@ip-172-31-29-57 ~]$ conda info --envs
# conda environments:
#
base                * /opt/conda
pytorch             /opt/conda/envs/pytorch

(base) [ec2-user@ip-172-31-29-57 ~]$ conda activate pytorch
(pytorch) [ec2-user@ip-172-31-29-57 ~]$ 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' inste
ad.
  warnings.warn(
/opt/conda/envs/pytorch/lib/python3.9/site-packages/torchvision/models/_utils.py:223: UserWarning: Argu
ments other than a weight enum or `None` for 'weights' 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)
Downloading: "https://download.pytorch.org/models/resnet50-0676ba61.pth" to /home/ec2-user/.cache/torch
/hub/checkpoints/resnet50-0676ba61.pth
100%|██████████████████████████████████████████████████████████████████████████████| 97.8M/97.8M [00:01<00:00, 97.0MB/s]
Starting Model Training
saved
(pytorch) [ec2-user@ip-172-31-29-57 ~]$ █
```

In order to get the training working, I needed to activate the conda environment which contained the pytorch library. As seen in the screenshot above, this was completed using:

```
conda info --envs
```

to identify the environments available on the EC2 instance and then:

```
conda activate pytorch
```

to activate the relevant environment.

Prior to this, I had copied the contents of the starter file into solution.py and was able to run it. The screenshot of the saved model weights is below:

```
(pytorch) [ec2-user@ip-172-31-29-57 ~]$ cd TrainedModels/  
(pytorch) [ec2-user@ip-172-31-29-57 TrainedModels]$ ls  
model.pth
```

Code Differences

Argument Parsing

Training on EC2 compared with training on Sagemaker requires slight differences in the configuration of the script. In the case of Sagemaker, it is possible to create model code as an entry point and use Sagemaker functions to pass desirable hyperparameters to this code when invoking the training job. As a result, the script expects to be called by the system and to have these values passed as arguments using the `argparse` library.

The code from the `hpo.py` code.

```
if __name__ == '__main__':  
    parser=argparse.ArgumentParser()  
    parser.add_argument('--learning_rate', type=float)  
    parser.add_argument('--batch_size', type=int)  
    parser.add_argument('--data', type=str,  
default=os.environ['SM_CHANNEL_TRAINING'])  
    parser.add_argument('--model_dir', type=str,  
default=os.environ['SM_MODEL_DIR'])  
    parser.add_argument('--output_dir', type=str,  
default=os.environ['SM_OUTPUT_DATA_DIR'])  
  
    args=parser.parse_args()
```

By comparison, the EC2 environment is significantly more closed. While it would be possible to configure a script to accept arguments when invoked from the EC2 SSH, it would only be possible to pass these

arguments from the terminal shell instance connected to the SSH session.

The use of these arguments in the `hpo.py` script is to enable repeated invocation of this script programmatically. When training on the EC2 environment these hyperparameters are invoked inline and without the use of a `main` function or block.

Sagemaker Use of Environment Variables

This difference is visible in the code snippet above from the Sagemaker. For sagemaker, the environment variables are set by the code which spins up the training instance(s). The information is placed into environment variables to enable necessary parameters for training (hyperparameter values, data paths, model paths, etc) to be accessible by the training job at runtime.

Lambda Function Setup

In order to deploy this model in production, it is necessary to create AWS Lambda functions to invoke the endpoint when a request is received and receive the output.

The starter code provided a `lambdafunction.py` starter file which is to be used to accomplish this.

Code Modification

The parallel training job created above and deployed to an endpoint is to be made the target of the Lambda function. I modified the following line from:

```
endpoint_Name='BradTestEndpoint'
```

to

```
endpoint_Name='pytorch-inference-2022-08-13-12-29-03-528'
```

However, this Lambda function will still not work until the appropriate IAM policy is attached to the Lambda execution role.

Lambda Function Code

General Structure

The lambda function script consists mainly in a single function `lambda_handler` which takes two arguments;

1. Event
 1. An event is an JSON object which is used as the request format for the endpoint
2. Context
 1. Provides information about the thing which invokes the endpoint, how it is invoking the endpoint and other important information which may be used to alter how the Lambda function processes the request. See [here](#)

The invocation of the endpoint will be discussed later, but the code then invokes the endpoint, obtains the response, decodes it and creates a JSON object which is to be returned to the requester which provides the model's predictions for the provided input.

In general, it is a requirement that the Lambda function include a `statusCode` key to allow the recipient to understand that the content received is a valid response which is a result of successful invocation of the function. The body is used to provide the endpoint prediction in the form of a json dump which can be interpreted by the recipient.

Sagemaker Endpoint Invocation

When created a lambda function, it is important to note that it is not possible to import non-standard third party libraries and it is not possible to invoke installation through either `conda` or `pip`. As such, when creating a lambda function, if it is necessary to use these libraries, one would need to install versions of these libraries on a machine of identical configuration to the Lambda function executors and zip these scripts,

along with the lambda function itself into a folder which can be uploaded.

This is practically very difficult, so it is desirable to use as few third party libraries as possible. As a result, invocation of endpoints is generally not done using the AWS `sagemaker` library, but rather using the `boto3` sagemaker runtime.

This allows endpoint invocation using only `boto3`, `json` and encoding libraries required for your application.

IAM Role configuration

Owing to an AWS reset problem with my Udacity account, I created a Lambda execution role from scratch for this Lambda function. In order to have access to the Sagemaker parts of the AWS environment, it is necessary to attach a policy which allows access.

Initially, I provided the Lambda execution role with a Sagemaker Full Access policy. However, it is a principle of good security design that the lowest privilege possible which does not obstruct the business process should be granted. In this case the policy I chose to attach was [AmazonSageMakerServiceCatalogProductsLambdaServiceRolePolicy](#). This policy grants Lambda access to a limited range of Sagemaker services. AWS' description of this service is:

Service role policy used by the AWS Lambda within the AWS ServiceCatalog provisioned products from Amazon SageMaker portfolio of products.

I believe that this represents a relatively low security privilege which trades off convenience of not writing my own json policy definition with the need to not provide full access to the lambda function IAM role.

The screenshot of my execution role with the relevant policy attached is below:

IAM > Roles > operationalise-aws-endpoint-invocation-role-xknkapq1

operationalise-aws-endpoint-invocation-role-xknkapq1 Delete

Summary Edit

Creation date
August 13, 2022, 22:37 (UTC+10:00)

Last activity
None

ARN
arn:aws:iam::566372991438:role/service-role/operationalise-aws-endpoint-invocation-role-xknkapq1

Maximum session duration
1 hour

[Permissions](#) | [Trust relationships](#) | [Tags](#) | [Access Advisor](#) | [Revoke sessions](#)

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

Simulate Remove Add permissions

<input type="checkbox"/>	Policy name	Type	Description
<input type="checkbox"/>	AWSLambdaBasicExecutionRole-01ae3094-5354-464f-8103-7e4b492f27ed	Customer managed	
<input type="checkbox"/>	AmazonSageMakerServiceCatalogProductsLambdaServiceRolePolicy	AWS managed	Service role policy used by the AWS Lambda within the AWS ServiceCa...

In general, AWS IAM roles should be reviewed regularly to ensure that services still need access to the policies which are attached to their execution roles. Additionally, regular review of IAM roles and user accounts allows for regular removal of outdated policies which may provide potential vectors for attack. AWS IAM allows convenient creation of security policies which meet most needs and the JSON definition for policies means that if there is a specific use case, you can create a policy which meets those precise needs. The provision of execution roles to services ensures that a service cannot exceed its allotted authority, accruing extra charges or potentially leading to the exposure of sensitive data held within the environment.

I believe as a result of my security policies that this endpoint and the AWS environment are secure.

Lambda Function Test

After attaching this group policy, the suggested test object in the course documentation was created for the Lambda function and a test was run. Screenshots of this test are included below:

operationalise-aws-endpoint-invocation

Throttle Copy ARN Actions

Function overview Info

operationalise-aws-endpoint-invocation

Layers (0)

+ Add trigger

+ Add destination

Description

-

Last modified

5 minutes ago

Function ARN

arn:aws:lambda:us-east-1:566372991438:function:operationalise-aws-endpoint-invocation

Function URL Info

-

Code Test Monitor Configuration Aliases Versions

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": "<__main__.LambdaContext object at 0x7f06305d4b20>",
  "body": "[[-9.87108039855957, -3.274627685546875, -12.171101570129395, -2.219409227371216, -4.293405532836914, -3.1757571697235107, -5.912885665893555, -2.5159709453582764, -6.063247203826904, -5.644251346588135, -2.487307071685791, -2.2699341773986816, -8.111492156982422, -5.746113300323486, -4.665231704711914, -6.925404071807861,
```

This shows the body of 33 numbers included in the prediction which is the response from the endpoint.

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": "<__main__.LambdaContext object at 0x7f06305d4b20>",
  "body": "[[-9.87108039855957, -3.274627685546875, -12.171101570129395, -2.219409227371216, -4.293405532836914, -3.1757571697235107, -5.912885665893555, -2.5159709453582764, -6.063247203826904, -5.644251346588135, -2.487307071685791, -2.2699341773986816, -8.111492156982422, -5.746113300323486, -4.665231704711914, -6.925404071807861,
```

Summary

Code SHA-256	Request ID
PGYn1HP1MOWZCfKluwvj4U2xvChbe0YkLV61Wwesx4=	5a3e62a8-83bd-4b5b-bd11-111fd6b18713
Init duration	Duration
343.88 ms	1034.92 ms
Billed duration	Resources configured
1035 ms	128 MB
Max memory used	
67 MB	

Concurrency & Autoscaling

In order to make a model commercially viable it is necessary to allow concurrent requests to be received and processed and autoscaling of the model endpoint to occur to service these requests.

Concurrency

In order to allow concurrency for the Lambda function, it is necessary to publish a version of it and have this version used for the concurrency. The published version is below:

Versions (1) Info						Delete	Publish new version
<input type="text" value="Find versions"/>						< 1 >	
	Version	Aliases	Description	Last modified	Architecture		
<input type="radio"/>	1	-	concurrency-version	26 minutes ago	x86_64		

My assumptions for this project is that this dog classification system will be operated by a low budget enterprise and will receive little traffic. My own affection for precision in the discussion of dog breeds notwithstanding, the general public is unlikely to clamour for very high throughput and extremely low latency access to dog breed classification tools. As a result, I have elected to set both the reserved concurrency and the provisioned concurrency to low values: 5 and 3 respectively.

Concurrency

Edit

Function concurrency

Use reserved concurrency

Reserved concurrency

5

Provisioned concurrency configurations (1)

Edit

Remove

Add

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](#)

Find configuration

Qualifier	Type	Provisioned concurrency	Status	Details
1	version	3	Ready	-

These numbers will ensure that there is sufficient capacity for a modest amount of calls to the endpoint but not so much as to significantly tax the financial resources of the enterprise.

After configuring concurrency, another test of the endpoint was performed to ensure that it was still functioning well.

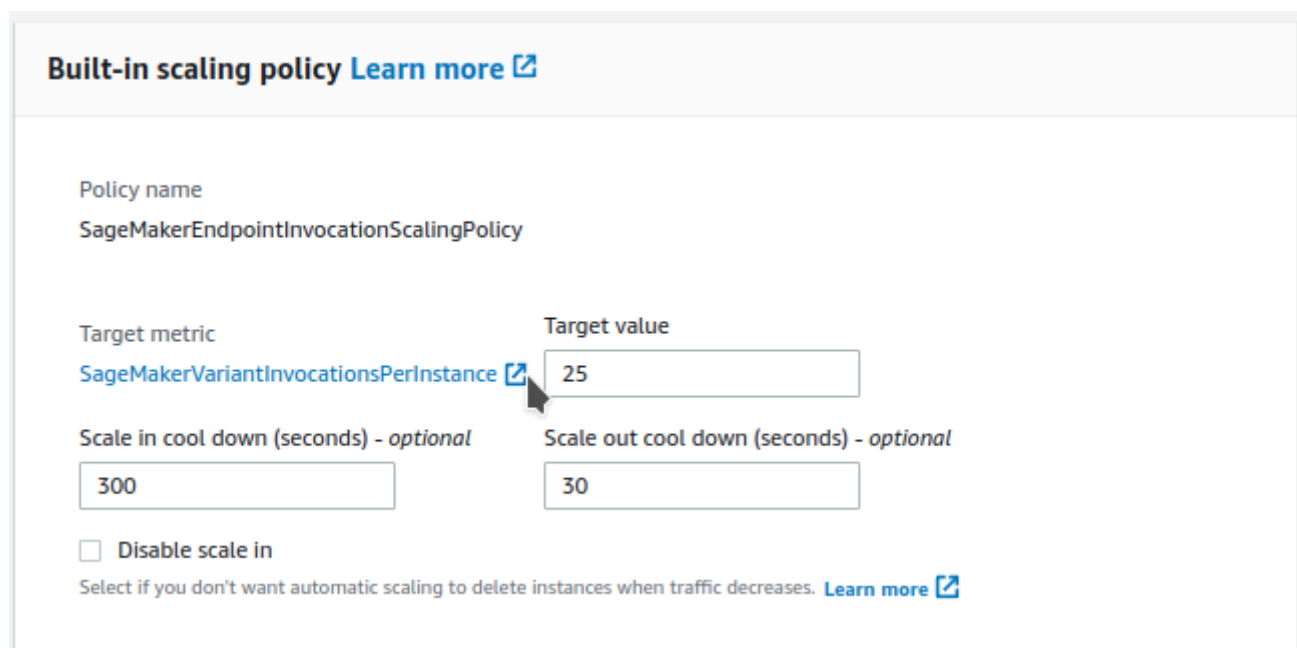
Execution result: succeeded (logs)		Close
<div> <div>▼ Details</div> <div> The area below shows the last 4 KB of the execution log. <pre> "Content-Type": "text/plain", "Access-Control-Allow-Origin": "*" }, "type-result": "<class 'str'>", "Content-Type-In": "<_main_.LambdaContext object at 0x7f1505af5b20>", "body": "[[-9.871088039855957, -3.274627685546875, -12.171101570129395, -2.219409227371216, -4.293405532836914, -3.1757571697235107, -5.912885665893555, -2.5159709453582764, -6.963247203826904, -5.644251346588135, -2.487307071685791, -2.2699341773986816, -8.111492156982422, -5.746113300323486, -4.665231704711914, -6.925404071807861, -5.891304016113281, -1.685286283493042, -7.825011730194092, -0.8985950350761414, -1.3281497955322266, -2.387258529663086, -3.0700345039367676, -5.152482086450195, -8.48027758026123, -11.438613891601562, -1.1647281646728516, -6.438976764678955, -2.1899726390838623, -9.549890518188477, -4.954113483428955, -3.6639914512634277, -10.473698016967773, -1.2244141101837158, -7.646878242492676, -7.085056304931641, -10.012234687805176, -6.803778648376465, -7.189074516296387, -6.535496112670898, 3.6511747064737305, 0.0712704700276105, 2.7007224466706523, 4.220765243711402, 4.0592004375, 6.176706730550527, 4.067067022102920, 3.004212225419701]"] </pre> </div> </div>		
<div> <div>Summary</div> <div> <div> Code SHA-256 PGYn1HP1MOWZCfkluwvj4U2xvChbe0YkLvY61Wwesx4= </div> <div> Request ID e810b390-1cfc-4ccb-ab0f-ce72b6e5d7b6 </div> </div> <div> <div> Init duration 334.15 ms </div> <div> Duration 996.06 ms </div> </div> <div> <div> Billed duration 997 ms </div> <div> Resources configured 128 MB </div> </div> <div> <div> Max memory used 67 MB </div> </div> </div>		
<div> <div>Log output</div> <div> The section below shows the logging calls in your code. Click here to view the corresponding CloudWatch log group. </div> </div>		
<div> <div>START RequestId: e810b390-1cfc-4ccb-ab0f-ce72b6e5d7b6 Version: \$LATEST</div> </div>		

Autoscaling

In the event that there are a large number of requests for the endpoint, it is desirable to configure the endpoint to scale out to additional instances to meet demand. In line with the assumptions of the previous section, I have assumed that a small amount of autoscaling is permissible but it should be for a small amount of instance (I have selected a minimum of 1 and a maximum of 3 instances) and that the scale in and scale out cool down settings should ramp slowly and cut back quickly.

As a result, I have selected a scale in cool down of 300 meaning that the endpoint will wait a significant amount of time at high usage prior to scaling out to an additional instance. Additionally, the scale out cool down has been set to a low value of 30, meaning that the endpoint will wait a shorter amount of time before shedding unutilised instances. While this may lead to bursty performance, it will ensure minimal cost for what is expected to be a fairly low usage endpoint.

Finally, I have selected a target value of 25 which is the number of concurrent or nearly concurrent endpoint activations per instance prior to the threshold for high usage being reached.



The screenshot shows the 'Built-in scaling policy' configuration page in the AWS SageMaker console. The policy name is 'SageMakerEndpointInvocationScalingPolicy'. The target metric is 'SageMakerVariantInvocationsPerInstance' with a target value of 25. The scale in cool down is set to 300 seconds, and the scale out cool down is set to 30 seconds. There is an unchecked checkbox for 'Disable scale in'.

Field	Value
Policy name	SageMakerEndpointInvocationScalingPolicy
Target metric	SageMakerVariantInvocationsPerInstance
Target value	25
Scale in cool down (seconds) - optional	300
Scale out cool down (seconds) - optional	30
Disable scale in	<input type="checkbox"/>

The number of instances permissible for auto scaling is present in the image below along with evidence that the settings were deployed correctly.

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			
<input type="radio"/>	AllTraffic	1	1	m5.large	-	1	1	1 - 3	Yes			