

# Project 4 Submission - Operationalizing an AWS ML Project

In this file I will describe my approach on the 4th project of the nanodegree

## Sagemaker Instance

The first step of the project is to create a new Notebook Instance in order to serve the submission scripts that were provided by the instructor. Since this instance needs only to handle the running of the jupyter notebook itself (importing libraries, running the basic code of the notebook) and will not perform any training, deployments or hyper parameter optimizations, since they will be handled by Sagemaker containers, so the most basic solution will be sufficient. Thus I chose to use the `ml.t3.medium` instance type, which not only costs half the price of the next option [0.05\$ per hour], it is also eligible for the free tier. It offers 2vCpus and 4 Gb of RAM, which I consider more than enough to handle its load. More info can be obtained here, from where the following screenshots were taken.

### *Free tier eligibility*

Amazon SageMaker capability	Free Tier usage per month for the first 2 months
Studio notebooks, and notebook Instances	250 hours of ml.t3.medium Instance on Studio notebooks OR 250 hours of ml.t2 medium Instance or ml.t3.medium Instance on notebook Instances

### *Pricing options*

Region:

US East (N. Virginia)

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

### *Notebook Instance creation*

Amazon SageMaker > Notebook Instances > Create notebook instance

## 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**

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

**Notebook instance type**

**Elastic Inference** [Learn more](#)

**Platform Identifier** [Learn more](#)

► Additional configuration

## S3 Bucket

The next step is to create an S3 buck to upload the dataset. In my workspace, I created a new S3 bucket, keeping the default options, and made the required changes to the code. In general, the storing costs in AWS are low, and we only get charged when download and/or uploading data to it. In case of upload failure, don't forget to attach the `AmazonS3FullAccess` policy to the notebook IAM role. *The S3 Bucket*

Amazon S3 > Buckets > project-4-data

project-4-data [info](#)

Objects Properties Permissions Metrics Management Access Points

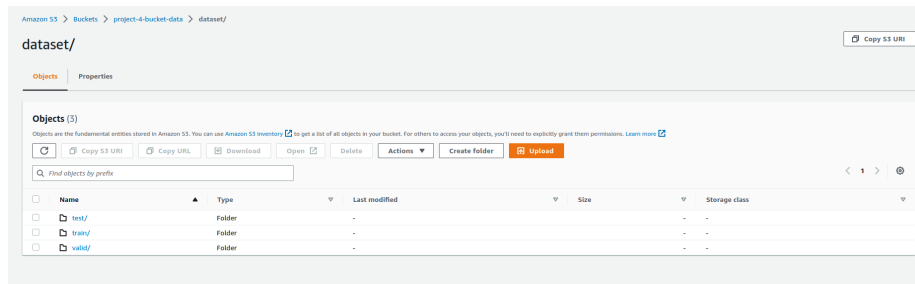
**Objects (0)**

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](#)

Find objects by prefix

Name	Type	Last modified	Size	Storage class
No objects				
You don't have any objects in this bucket.				

And we can check how the uploaded data are shown inside the bucket



## Hyperparameter tuning

Based on the instructions of the project, we created the training job in order to perform the hyperparameter optimization. Below is a screenshot of the training jobs in Sagemaker

Name	Creation time	Duration	Job status	Warm pool status	Time left
pytorch-training-230404-1710-002-46f92c74	4/4/2023, 8:31:41 PM	-	InProgress	InUse	-
pytorch-training-230404-1710-001-0a748aaf	4/4/2023, 8:11:00 PM	20 minutes	Completed	Reused	-

Below, we can see a screenshot of the best hyperparameters, as calculated by the model

```
[9]: {'tuning objective metric': 'Test Loss',
      'batch_size': '32',
      'learning_rate': '0.0015758985540195415',
      'sagemaker_container_log_level': '20',
      'sagemaker_estimator_class_name': 'PyTorch',
      'sagemaker_estimator_module': 'sagemaker.pytorch.estimator',
      'sagemaker_job_name': 'pytorch_dog_hpo-2023-04-04-17-10-54-889',
      'sagemaker_program': 'hpo.py',
      'sagemaker_region': 'us-east-1',
      'sagemaker_submit_directory': 's3://sagemaker-us-east-1-557664247624/pytorch_dog_hpo-2023-04-04-17-10-54-889/source/sourcedir.tar.gz'})

[10]: hyperparameters = {'batch_size': int(best_estimator.hyperparameters()['batch_size'].replace(' ', '')), \
                        'learning_rate': best_estimator.hyperparameters()['learning_rate']}
      hyperparameters

[10]: {'batch_size': 32, 'learning_rate': '0.0015758985540195415'}
```

## Estimator

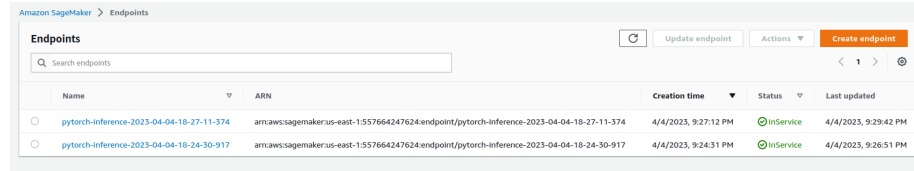
The next step was to create two estimators, a single instance and a multiinstance one. Here is the screenshot of the corresponding training jobs.

Name	Creation time	Duration	Job status	Warm pool status	Time left
dog-pytorch-2023-04-04-17-59-36-862	4/4/2023, 8:59:37 PM	-	InProgress	-	-
dog-pytorch-2023-04-04-17-58-17-177	4/4/2023, 8:58:18 PM	-	InProgress	-	-
dog-pytorch-2023-04-04-17-52-27-201	4/4/2023, 8:52:28 PM	-	InProgress	-	-
dog-pytorch-2023-04-04-17-52-25-572	4/4/2023, 8:52:26 PM	-	InProgress	-	-

## Model Deployment

After finishing the training process, we were able to deploy those models to endpoints. I made some minor changes to the code provided, in order to create the two endpoints rather simultaneously. Below is a screenshot of the deployed

endpoints.



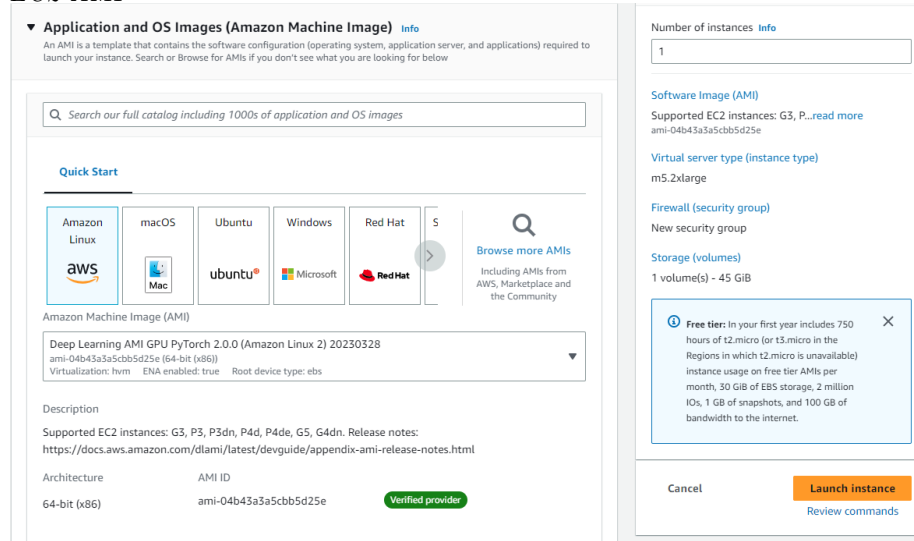
The screenshot shows the Amazon SageMaker Endpoints console. At the top, there's a search bar and buttons for 'Update endpoint', 'Actions', and 'Create endpoint'. Below this is a table with columns: Name, ARN, Creation time, Status, and Last updated. Two endpoints are listed, both with a status of 'InService'.

Name	ARN	Creation time	Status	Last updated
pytorch-inference-2023-04-04-18-27-11-374	arn:aws:sagemaker:us-east-1:557664247624:endpoint/pytorch-inference-2023-04-04-18-27-11-374	4/4/2023, 9:27:12 PM	InService	4/4/2023, 9:29:42 PM
pytorch-inference-2023-04-04-18-24-30-917	arn:aws:sagemaker:us-east-1:557664247624:endpoint/pytorch-inference-2023-04-04-18-24-30-917	4/4/2023, 9:24:31 PM	InService	4/4/2023, 9:26:51 PM

## EC2 Training

The first step is to create an EC2 instance to train the model. According to the instructor's guidelines, for the AMI the deep learning was selected.

*EC2 AMI*



The screenshot shows the AWS Management Console page for 'Application and OS Images (Amazon Machine Image)'. It includes a search bar, a 'Quick Start' section with various OS options (Amazon Linux, macOS, Ubuntu, Windows, Red Hat, S), and a list of AMIs. The selected AMI is 'Deep Learning AMI GPU PyTorch 2.0.0 (Amazon Linux 2) 20230328'. The right sidebar shows configuration options: 'Number of instances' (1), 'Software Image (AMI)' (ami-04b43a3a5cbb5d25e), 'Virtual server type (instance type)' (m5.2xlarge), 'Firewall (security group)' (New security group), and 'Storage (volumes)' (1 volume(s) - 45 GiB). A 'Free tier' warning is also present.

▼ Application and OS Images (Amazon Machine Image) [Info](#)

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below

Search your full catalog including 1000s of application and OS images

**Quick Start**

Amazon Linux macOS Ubuntu Windows Red Hat S [Browse more AMIs](#)

Amazon Machine Image (AMI)

Deep Learning AMI GPU PyTorch 2.0.0 (Amazon Linux 2) 20230328  
ami-04b43a3a5cbb5d25e (64-bit (x86))  
Virtualization: hvm ENA enabled: true Root device type: ebs

Description

Supported EC2 instances: G3, P3, P3dn, P4d, P4de, G5, G4dn. Release notes:  
<https://docs.aws.amazon.com/dlami/latest/devguide/appendix-ami-release-notes.html>

Architecture AMI ID  
64-bit (x86) ami-04b43a3a5cbb5d25e [Verified provider](#)

Number of instances [Info](#)  
1

Software Image (AMI)  
Supported EC2 instances: G3, P...[read more](#)  
ami-04b43a3a5cbb5d25e

Virtual server type (instance type)  
m5.2xlarge

Firewall (security group)  
New security group

Storage (volumes)  
1 volume(s) - 45 GiB

**Free tier:** In your first year includes 750 hours of t2.micro (or t3.micro in the Regions in which t2.micro is unavailable) instance usage on free tier AMIs per month, 30 GiB of EBS storage, 2 million IOs, 1 GB of snapshots, and 100 GB of bandwidth to the internet.

[Cancel](#) [Launch instance](#) [Review commands](#)

The next thing to set up was the instance type to be used. As a general guide, the more computing power an instance has, the more it costs to run it. For the training I used them **g5.xlarge** instance, which although seems to be equivalent to the **g4dn.xlarge** that was used in the previous case, according to Amazon the G5 instances can deliver up to 3x higher performance and up to 40% better price performance for machine learning inference compared to G4dn instances, and it's cost is not that significantly higher, as we can see from the following pics and is in the supported types for the AMI we chose.

*g3.4xlarge*

▼ Instance type [Info](#)

Instance type

g5.xlarge

Family: g5 4 vCPU 16 GiB Memory

On-Demand Windows pricing: 1.19 USD per Hour

On-Demand RHEL pricing: 1.066 USD per Hour

On-Demand SUSE pricing: 1.0623 USD per Hour

On-Demand Linux pricing: 1.006 USD per Hour

All generations

Compare instance types

*g4dn.xlarge*

Amazon macOS Ubuntu Windows Red Hat S

Q g4dn

g4dn.12xlarge

Family: **g4dn** 48 vCPU 192 GiB Memory

On-Demand Linux pricing: 3.912 USD per Hour

On-Demand SUSE pricing: 4.037 USD per Hour

On-Demand Windows pricing: 6.12 USD per Hour

On-Demand RHEL pricing: 4.042 USD per Hour

g4dn.xlarge

Family: **g4dn** 4 vCPU 16 GiB Memory

On-Demand Linux pricing: 0.526 USD per Hour

On-Demand SUSE pricing: 0.582 USD per Hour

On-Demand Windows pricing: 0.71 USD per Hour

On-Demand RHEL pricing: 0.586 USD per Hour

g4dn.16xlarge

Family: **g4dn** 64 vCPU 256 GiB Memory

On-Demand Windows pricing: 7.296 USD per Hour

On-Demand SUSE pricing: 4.477 USD per Hour

On-Demand RHEL pricing: 4.482 USD per Hour

On-Demand Linux pricing: 4.352 USD per Hour

g4dn.8xlarge

Family: **g4dn** 32 vCPU 128 GiB Memory

On-Demand SUSE pricing: 2.301 USD per Hour

On-Demand Windows pricing: 3.648 USD per Hour

On-Demand Linux pricing: 2.176 USD per Hour

On-Demand RHEL pricing: 2.306 USD per Hour

The next step is to create a key pair, in order to authenticate ourselves and be able to ssh in the EC2 instance. AWS makes it really easy to create it.

*Key Pair Creation*

Create key pair

Key pairs allow you to connect to your instance securely.

Enter the name of the key pair below. When prompted, store the private key in a secure and accessible location on your computer. **You will need it later to connect to your instance.**

[Learn more](#)

Key pair name

project-4-key

The name can include upto 255 ASCII characters. It can't include leading or trailing spaces.

Key pair type

☒ RSA
 

RSA encrypted private and public key pair

☐ ED25519
 

ED25519 encrypted private and public key pair (Not supported for Windows instances)

Private key file format

☒ .pem
 

For use with OpenSSH

☐ .ppk
 

For use with PuTTY

Cancel

Create key pair

Next step, after creating the instance, is to connect to it. This can be performed as shown in the following images

### EC2 Instance selection

Instances (3) info

Find instance by attribute or tag (case-sensitive)

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
<input type="checkbox"/> project4-ec2	i-054c80fa23edca6	Terminated	m5.2xlarge	-	No alarms	us-east-1b	-	-	-
<input type="checkbox"/> project4-ec2	i-06847f6ccb4f5db3	Terminated	m5.2xlarge	-	No alarms	us-east-1b	-	54.196.137.114	-
<input type="checkbox"/> project4-insta...	i-0b076555ed273108	Running	g5.xlarge	initializing	No alarms	us-east-1c	ec2-3-88-24-64.comput...	3.88.24.64	-

### Connect to EC2 Instance

6

EC2 > Instances > i-068471f6ccbf5db3 > Connect to instance

### Connect to instance [Info](#)

Connect to your instance i-068471f6ccbf5db3 (project4-ec2) using any of these options

[EC2 Instance Connect](#) | [Session Manager](#) | [SSH client](#) | [EC2 serial console](#)

---

Instance ID  
 i-068471f6ccbf5db3 (project4-ec2)

Public IP address  
 54.196.137.114

User name  
 Enter the user name defined in the AMI used to launch the instance. If you didn't define a custom user name, use the default user name, root.

**Note:** In most cases, the default user name, root, is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI user name.

Cancel **Connect**

If everything goes well, we will be connected to the EC2 instance and will be able to activate the pytorch virtual environment.

### EC2 Instance Running

```

  _ _ _ _ _
 _ _ _ _ _ ( _ _ _ )
 _ _ _ _ _ \ _ _ _ _ _
 _ _ _ _ _ \ _ _ _ _ _

Deep Learning AMI GPU PyTorch 2.0.0 (Amazon Linux 2)

=====
* Please note that Amazon EC2 P2 Instance is not supported on current DLAMI.
* Supported EC2 instances: G3, P3, P3dn, P4d, P4de, G5, G4dn.
* 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: 525.85.12
* CUDA version: 11.8

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/

=====
[root@ip-172-31-44-246 ~]# source activate pytorch
(pytorch) [root@ip-172-31-44-246 ~]#

```

Now it's time for action. In general, the training process will follow the same guidelines as in the SageMaker case, although there would be some differences. First of all, in an EC2 instance, we can local filesystem to store the dataset. The commands go as following, after we ssh in the EC2 instance

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

```
mkdir TrainedModels
```

Next we need to create the python script to run the code that was provided by the instructor. First we copy the contents of the python script `ec2train1.py` thas was provided. Then we can use the following commands.

```
vim solution.py
:set paste
:wq! [and Enter]
```

python solution.py

If everything rolls out well, the code should run and the model be saved. And we could see that it did not take that much of time.

EC2 model saved

```
[root@condaenv:/pytorch/lib/python3.6/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('The parameter \'pretrained\' is deprecated since 0.13 and may be removed in the future, please use \'weights\' instead.')
[root@condaenv:/pytorch/lib/python3.6/site-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight sum or 'None' for 'weights' are deprecated since 0.13 and may be removed in the future. The current behavior is equivalent to passing 'weights=nn.Parameter(torch.FloatTensor(1)). You can also use 'weights=nn.init.zeros_([1])' to get the most up-to-date weights.
warnings.warn('Arguments other than a weight sum or \'None\' for \'weights\' are deprecated since 0.13 and may be removed in the future. The current behavior is equivalent to passing \'weights=nn.Parameter(torch.FloatTensor(1))\'. You can also use \'weights=nn.init.zeros_([1])\' to get the most up-to-date weights.')
Downloading: "https://download.pytorch.org/model_zoo/torchvision/segm/pretrained/maskrcnn_resnet50_fpn.pth" to /root/.cache/torch/hub/checkpoints/resnet50-fpn_maskrcnn.pth [97.3M/97.3M] [100.0%0.00, 32.0MB/s]
```

## Comparison between EC2 and SageMaker Notebook Instance

In terms of results, both solutions yield the same. After that, each case has it's own advantages and disadvantages. EC2 is more similar to conventional scripting, we write the scripts we need and then we execute the code via terminal. On the other hand, SageMaker can be integrated better in AWS and use more advanced and specific AWS features.

## Lambda Function Setup

Again, as per instructor's instructions, in the script provided we need to change the `endpoint_name` variable with the one that corresponds to our case. We can find it in the SageMaker part of our session, under Inference -> Endpoints. Then, though the AWS portal, we create a lambda function, using the latest version of Python as runtime.

### Lambda function



Lambda > Functions > Create function

## Create function info

AWS Serverless Application Repository applications have moved to [Create application](#).

☒ **Author from scratch**  
Start with a simple Hello World example.

☐ **Use a blueprint**  
Build a Lambda application from sample code and configuration presets for common use cases.

☐ **Container image**  
Select a container image to deploy for your function.

---

### Basic information

**Function name**  
Enter a name that describes the purpose of your function.  

Use only letters, numbers, hyphens, or underscores with no spaces.

**Runtime** info  
Choose the language to use to write your functions. Note that the console code editor supports only Node.js, Python, and Ruby.

**Architecture** info  
Choose the instruction set architecture you want for your function code.  
☒ x86\_64  
☐ arm64

**Permissions** info  
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

► Change default execution role

► Advanced settings

Cancel **Create function**

One key parameter that we must not forget to set, is to grant access to the IAM role that was automatically created for that lambda function to have access to SageMaker. In this case, for simplicity and since this lambda won't go out to production, we can choose the **AmazonSageMakerFullAccess** policy to attach, but in production environments, when security is an issue, we might want to be more restrictive.

IAM > Roles > project4-lambda-role-6f3xjl99

project4-lambda-role-6f3xjl99 Delete

**Summary** Edit

Creation date April 05, 2023, 19:56 (UTC+03:00)	ARN arn:aws:iam::557664247624:role/service-role/project4-lambda-role-6f3xjl99
Last activity None	Maximum session duration 1 hour

Permissions | Trust relationships | Tags | Access Advisor | Revoke sessions

**Permissions policies (2)** info ↻ Simulate Remove Add permissions ▼

You can attach up to 10 managed policies.

<input type="checkbox"/>	Policy name <small>?</small>	Type	Description
<input type="checkbox"/>	AWSLambdaBasicExecutionRole-d6432a3-9c57-4c18-869d-8d3ab8dbb1d	Customer managed	
<input type="checkbox"/>	AmazonSageMakerFullAccess	AWS managed	Provides full access to Amazon SageMaker via the ...

### Attaching policy in IAM role

The lambda function that is provided for us, invokes the endpoint we created earlier (and pasted it into the body of the lambda function) in order to use it for prediction. Before we can use the lambda function, we must deploy it.

### Deployed Lambda Function

```

1 import boto3
2 import logging
3 import json
4
5 logger = logging.getLogger(__name__)
6 logger.setLevel(logging.DEBUG)
7
8 print('Loading Lambda Function')
9
10 runtime = boto3.Session().client('sagemaker-runtime')
11 endpoint_name = 'gymfch-inference-2022-04-09-05-20-07-487' #not the same name as in train_and_deploy_solution.ipynb file, had to recreate the endpoint
12
13 def lambda_handler(event, context):
14
15     #event['url']
16     #event['url']
17     #event['url']
18     #event['url']
19     #event['url']
20     #event['url']
21     #event['url']
22     #event['url']
23     #event['url']
24     #event['url']
25     #event['url']
26     #event['url']
27     #event['url']
28     #event['url']
29     #event['url']
30     #event['url']
31     #event['url']
32     #event['url']
33     #event['url']
34     #event['url']
35     #event['url']
36     #event['url']
37     #event['url']
38     #event['url']
39     #event['url']
40     #event['url']
41     #event['url']
42     #event['url']
43     #event['url']
44     #event['url']
45     #event['url']
46     #event['url']
47     #event['url']
48     #event['url']
49     #event['url']
50     #event['url']
51     #event['url']
52     #event['url']
53     #event['url']
54     #event['url']
55     #event['url']
56     #event['url']
57     #event['url']
58     #event['url']
59     #event['url']
60     #event['url']
61     #event['url']
62     #event['url']
63     #event['url']
64     #event['url']
65     #event['url']
66     #event['url']
67     #event['url']
68     #event['url']
69     #event['url']
70     #event['url']
71     #event['url']
72     #event['url']
73     #event['url']
74     #event['url']
75     #event['url']
76     #event['url']
77     #event['url']
78     #event['url']
79     #event['url']
80     #event['url']
81     #event['url']
82     #event['url']
83     #event['url']
84     #event['url']
85     #event['url']
86     #event['url']
87     #event['url']
88     #event['url']
89     #event['url']
90     #event['url']
91     #event['url']
92     #event['url']
93     #event['url']
94     #event['url']
95     #event['url']
96     #event['url']
97     #event['url']
98     #event['url']
99     #event['url']
100    #event['url']

```

The lambda functions in general are invoked through events, which in general are JSON objects, and we will create such an event (as a test event) which consists of a single key-value pair

```
{ "url": "https://s3.amazonaws.com/cdn-origin-etr.akc.org/wp-content/uploads/2017/11/201133"
}
```

*Lambda Test Event*

Configure test event

A test event is a JSON object that mocks the structure of requests emitted by AWS services to invoke a Lambda function. Use it to see the function's invocation result.

To invoke your function without saving an event, configure the JSON event, then choose Test.

Test event action

Create new event

Edit saved event

Event name

project4-test1

Maximum of 25 characters consisting of letters, numbers, dots, hyphens and underscores.

Event sharing settings

Private

This event is only available in the Lambda console and to the event creator. You can configure a total of 10. [Learn more](#)

Shareable

This event is available to IAM users within the same account who have permissions to access and use shareable events. [Learn more](#)

Template - optional

hello-world

Event JSON

Format JSON

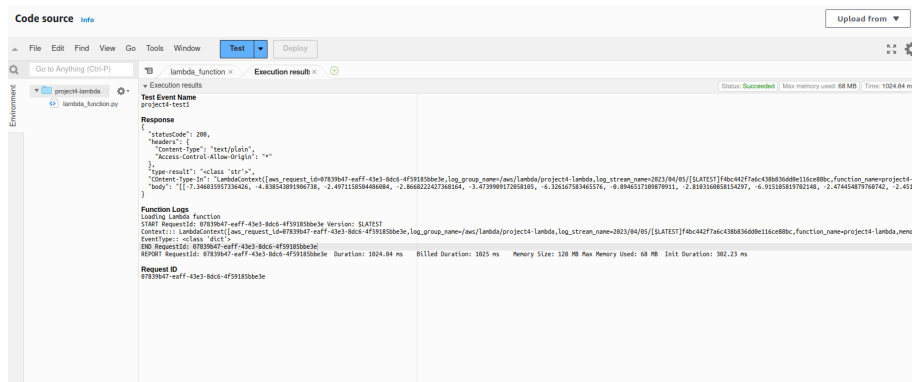
1 {

2 "url": "https://s3.amazonaws.com/cdn-origin-etr.akc.org/wp-content/uploads/2017/11/20113314/Carolina-Do

3 }

Upon successful invocation of the lambda function, it should respond with JSON object, containing a status code of 200 and the response of the given endpoint.

*Successful Lambda Invocation*



## Other Security Considerations

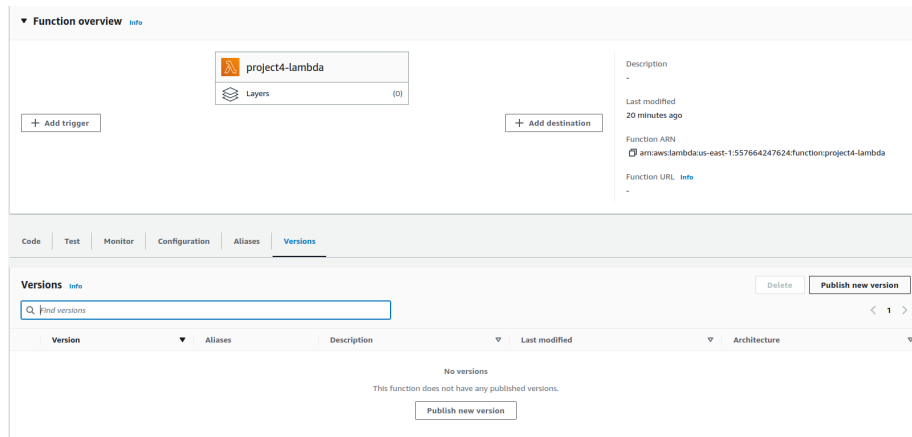
As we stated above, for the sake of simplicity, we granted the IAM the FullAccess role. Although it is a quick way to be sure that, if our code is not working as it should be, there is a bug somewhere and the problem is not something regarding proper permissions, in production environments we should use the more proper, and restrictive, policies. Of course, part from that, we should periodically confirm that all the IAM roles that we have granted permissions are indeed valid, and also use other tools that AWS offers, such as CloudWatch, to make sure that everything is running smoothly and everyone that interacts with our application should interact and in the proper manner.

## Concurrency of Lambda Functions

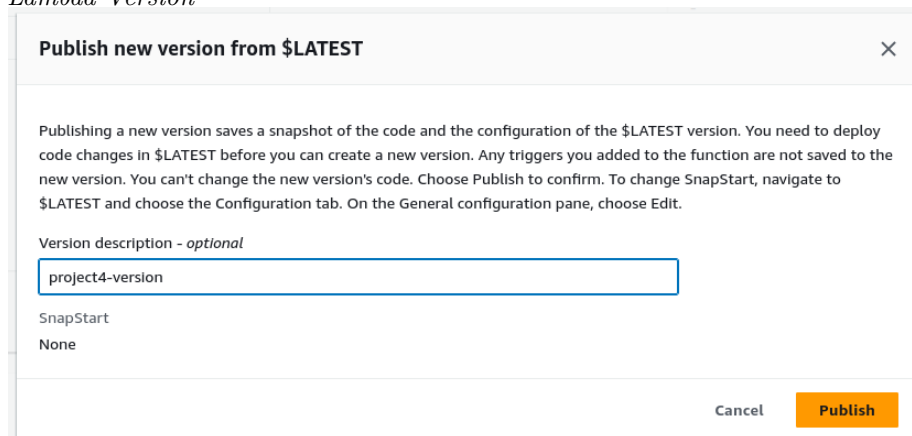
Another characteristic of the lambda functions is that they, by default, do not use concurrency, they only respond to one event at a time. AWS offers the possibility to add concurrency in the lambda functions. Prior adding concurrency, we need to navigate to the **Versions** tab of the lambda functions and configure a version.

Then we add concurrency through **Configuration -> Provisioned concurrency** and click the **Publish\_new** version button.

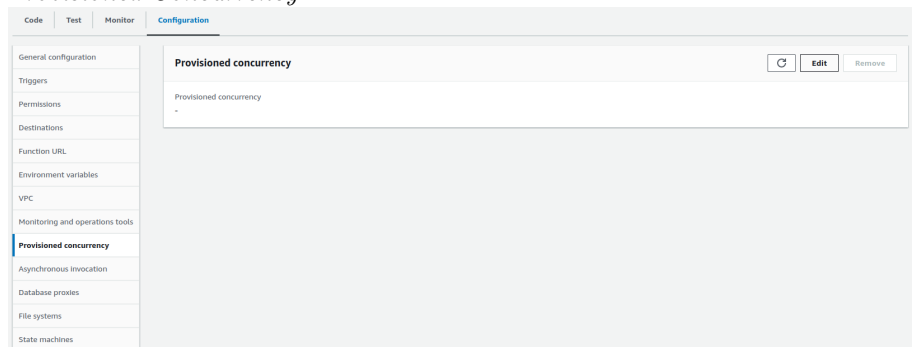
*Lambda Version Configuration*



Then we need to give a reference name  
*Lambda Version*



Then, in order to configure concurrency, we go to **Configuration->Provisioned Concurrency** and hit the **Edit** button  
*Provisioned Concurrency*



Finally, we need to setup how many instances should spawn. We should note that AWS informs us about the costs that will occur.

### *Lambda Set Up Provisioned Concurrency*

**Configure provisioned concurrency**

**Provisioned concurrency**

Version: 1  
Aliases: -

**Provisioned concurrency**  
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](#)

**\$2.79 per month in addition to pricing for duration and requests. [Pricing](#)**

2  
900 available

Cancel Save

The term provisioned concurrency refers to the fact that AWS initializes a specific number of lambda functions (execution environments). This has the effect that our app will be more responsive as requests get higher (has less latency), but as everything, has a prize. In our example, we used 2 as a value.

## Auto-Scaling Endpoints

Part from provisioned concurrency in the lambda function, there is also the option of auto-scaling the endpoint. The difference with the provision concurrency of the lambda function case is that in the endpoint they are dynamically adjusted. Based on the workload, AWS decides to spawn more instances or decrease their number, ensuring that in the end we only pay for what is actively running.

To enable autoscaling we need to navigate to SageMaker -> Endpoints -> Endpoint runtime settings

### *Endpoint Auto Scaling*

Endpoint runtime settings

Update weights

Update instance count

Configure auto scaling

	Variant name ▲	Current weight ▼	Desired weight	Elastic Inference	Instance type ▼	Current instance count ▼	Desired instance count ▼	Instance min - max	Automatic scaling
<div><div></div><div></div></div>	AllTraffic	1	1	-	ml.m5.large	1	1	-	No

There we define the minimum and maximum number of instances of the endpoint to be used. Part from that, we need also to define the Built-in scaling policy. Through this setting we define as the target value the number of (almost) simultaneous requests that need to be made for the auto-scaling to be triggered. Also we can optionally define two extra parameters. The scale-in cooldown refers to the time that should pass while the number of requests are below the target

metric, in order to decrease the number of instances, whereas the scale-out refers to the opposite, the number of time that should pass while the number of the requests are above the target metric, in order for AWS to spawn extra instances. In the project, I used the following configuration

Maximun instance count : 4

Target value : 25

Scale in cool down : 30

Scale out cool down : 5

#### *Endpoint Auto Scaling Count*

**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

#### *Endpoint Built-in Scaling Policy*

**Built-in scaling policy** [Learn more](#)

**Policy name**  
SageMakerEndpointInvocationScalingPolicy

**Target metric**  
[SageMakerVariantInvocationsPerInstance](#)

**Target value**  
25

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

**Scale out cool down (seconds) - optional**  
5

☐ **Disable scale in**  
Select if you don't want automatic scaling to delete instances when traffic decreases. [Learn more](#)

I chose to use a different number if those two values in order for my app to be more responsive. AWS is instructed to spaw relatively fast new instances up to

4, if 25 simultaneous requests occur for more than 5 secs, and also to keep those instances alive, even if the requests are below the threshold for 30 secs. The choice of these parameters is based on the nature of the app and the budget we have for the specific scenario. In general, there is no right-wrong answer to what those numbers should be, it depends whether we want to optimize for availability, cost or something in between.