Operationalizing AWS ML Project

Technical Write Up

Alejandro Santillan Iturres

Operationalizing AWS ML Project

As part of requirements for the AWS MLEND

Udacity Project 4: AWS Machine Learning Engineer Nanodegree Program (2021-2022)

In this proyect we are requested to accomplish five steps towards applying accuired skills about most importan services that AWS offer to the operative part of training and deploying Machine Learning models.

The project consists in 5 Steps
Step 1: Train and deploy a model on a Sagemaker notebook (one of the modalities that Sagemaker offer amongst several more).
Step 2: Perform a similar task on an EC2 instance.
Step 3: Create a Lambda function that will consume your model inference capabilites via endpoints.
Step 4: Security and testing.
Step 5: Set up concurrence for your lambda function and auto-scaling for your deployed endpoint.

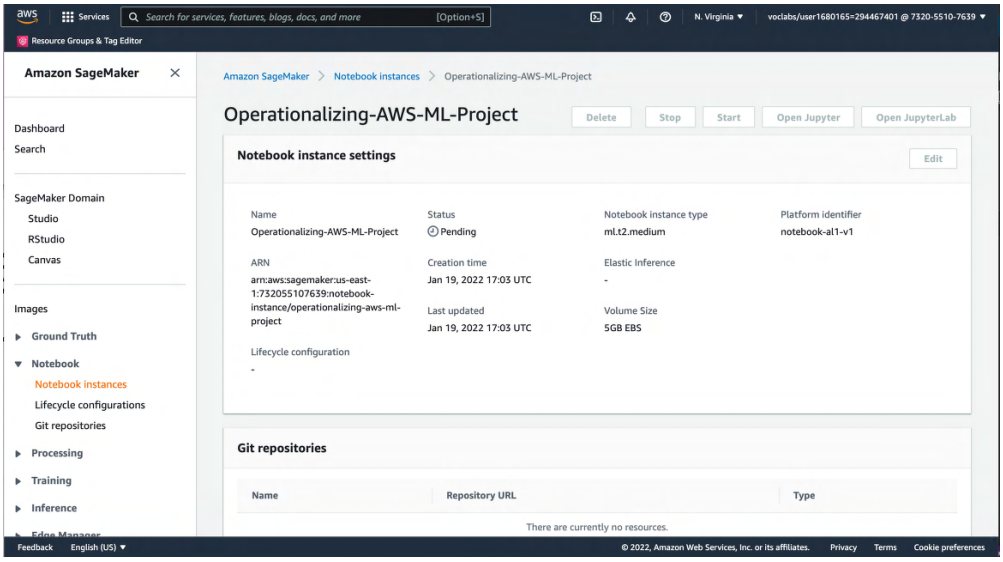
Step 0: Project Set Up and Installation

You will need to have a AWS account, which can be opened for free at https://aws.amazon.com You will be able to follow all the steps in this project in a day of work and with a cost of around $4.00 Follow the instructions in the write up file located here: https://github.com/alelasantillan/Operationalizing_AWS_ML_Project/blob/main/writeup.pdf

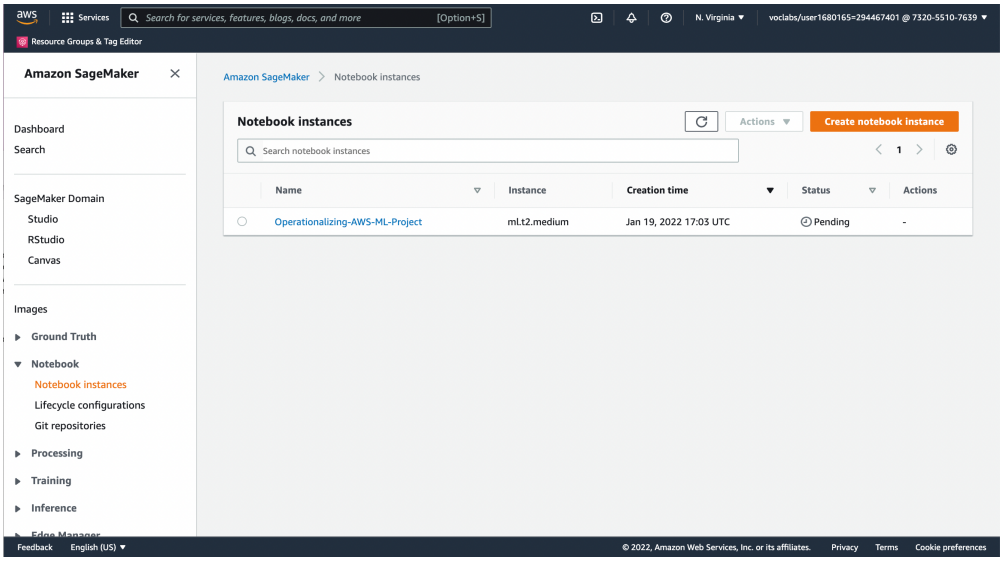
Step 1: Train and deploy a model on a Sagemaker notebook

Notebook Instance setup

1.1 Created a *ml.t2.medium* instance named Operationalizing-AWS-ML-Project. The cost of this instance is not published, but certainly is less than $0.05/hour, which is the price for *ml.t3.medium* according AWS: https://aws.amazon.com/sagemaker/pricing/. This instance will allow me to perform code debugging without incurring in great costs. In general the computing resources are consumed by the processes launched by the notebook rather than the notebook itself, so no much is required in this instance. Once the project is running, I will consider increasing capacity along with the stress testing I will eventually perform. In previous experiences I've used also ml.m5.large ($0.115) with sucess, but this time I will take more care of resources used since in the former project I ended too close to the limit.

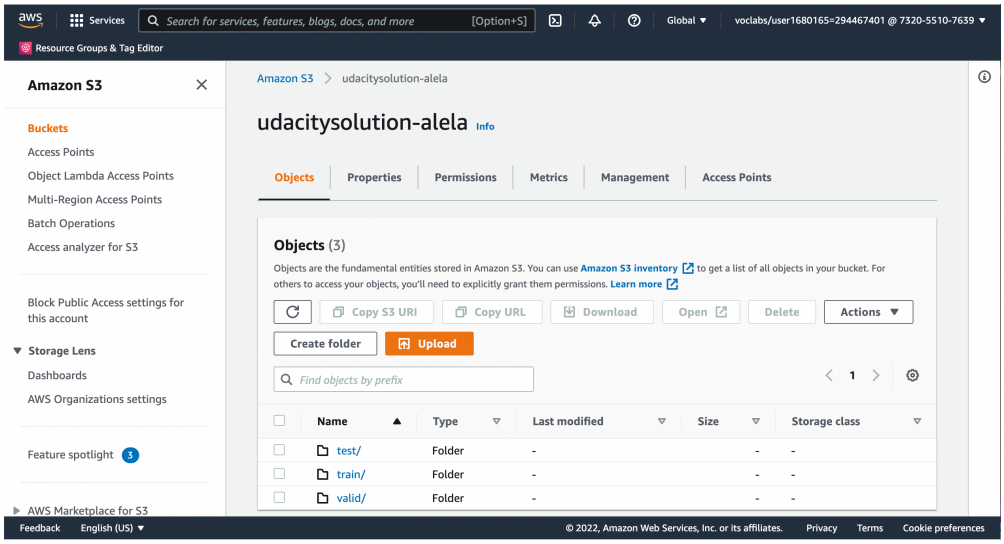


I've launched the Notebook Instance, but it took a long time to be ready. It happens from time to time, but is not usual. You just have to wait:



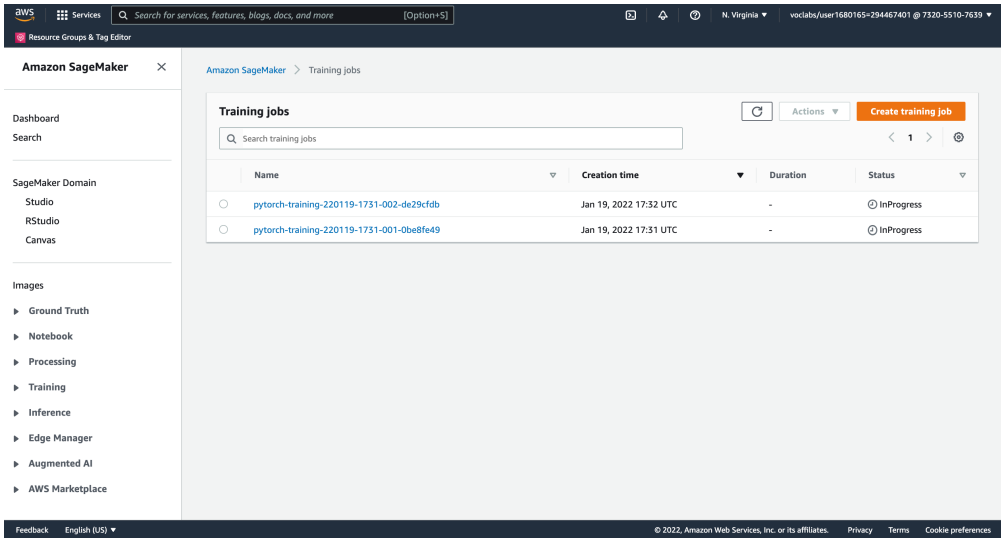
1.2 I uploaded the train_and_deploy-solution.ipynb into the SageMaker notebook instance, as well as the files hpo.py and infernce2.py to run the Hyperparameter Optimization part, the training-debugging part and the endpoint deploy part. I adjusted the bucket name in all ocurrences and changed the instance types for running the three different process: two *ml.m5.xlarge* for the hyperparameter optimization and training-debugging and *ml.m5.large* for deploy of endpoint for inferences.

1.3 Created a bucket named "udacitysolution-alela" and changed the notebook to use that bucket. Run the train_and_deploy-solution.ipynb first cells refered about data collection, unzipping and syncronization with s3 and the cells created the images folders, and images into the bucket.

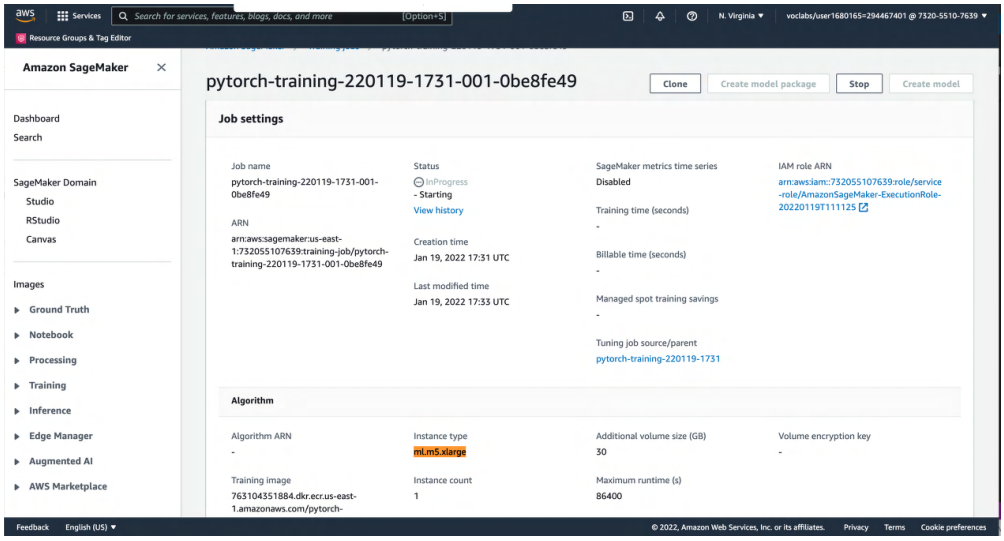


1.4 Run the following cells of the notebook to peform Hyperparameter optimization. This computation takes some time, depending on the instance type you choose to run computation.

I reserved the values of the optimization to perform the training of the model. At this point using the smdebug module, web perform debugging of the model to avoid the following problems that can show up in any training: overfitting, vanishin gradients, poor weight initialization or overtraining. Once the model is trained this way, we create another identical model but with multi instance. The multi instance training resulted in: Training seconds: 4221 whereas the single instance just: Training seconds: 1339



This are the details of each training instance:

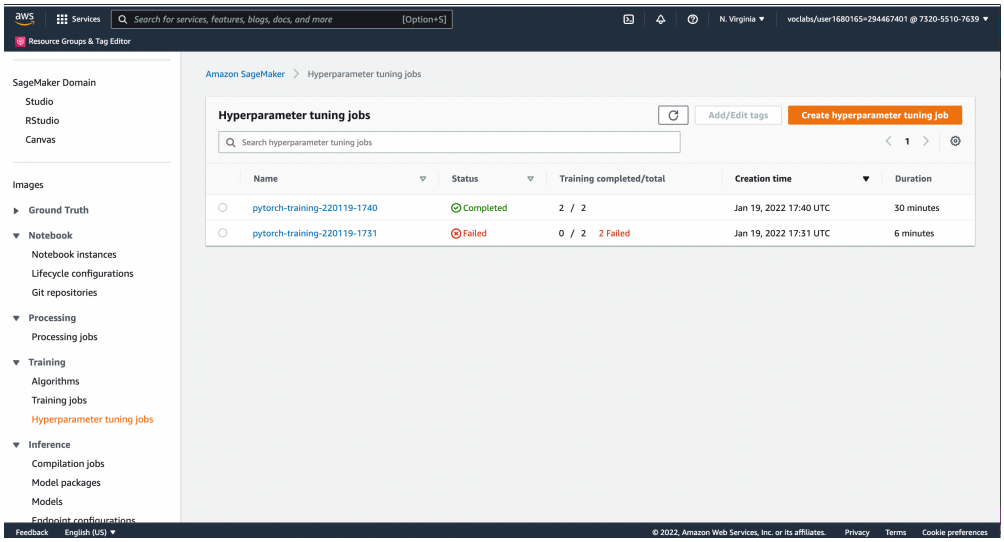


And this is the cell code of the notebook that determined the two training instances to accelerate computation:

```
In [7]: estimator = PyTorch(
            entry_point="hpo.py",
            base_job_name='pytorch_dog_hpo',
            role=role,
            framework_version="1.4.0",
            instance_count=1,
            instance_type="ml.m5.xlarge",
            py_version='py3'
        )

        tuner = HyperparameterTuner(
            estimator,
            objective_metric_name,
            hyperparameter_ranges,
            metric_definitions,
            max_jobs=2,
            max_parallel_jobs=2,
            objective_type=objective_type
        )
```

And the tuning job can also be seen from here:



Finally, when those jobs completed execution, we have the following result on the cell:

```
In [23]: hyperparameters = {"batch_size": int(best_estimator.hyperparameters()['batch_size'].replace('"', '')), \
                            "learning_rate": best_estimator.hyperparameters()['learning_rate']}
         hyperparameters

Out[23]: {'batch_size': 128, 'learning_rate': '0.012257979272044682'}
```

We can keep this values to use them later for the training-debugging of the model with optimal parameters.

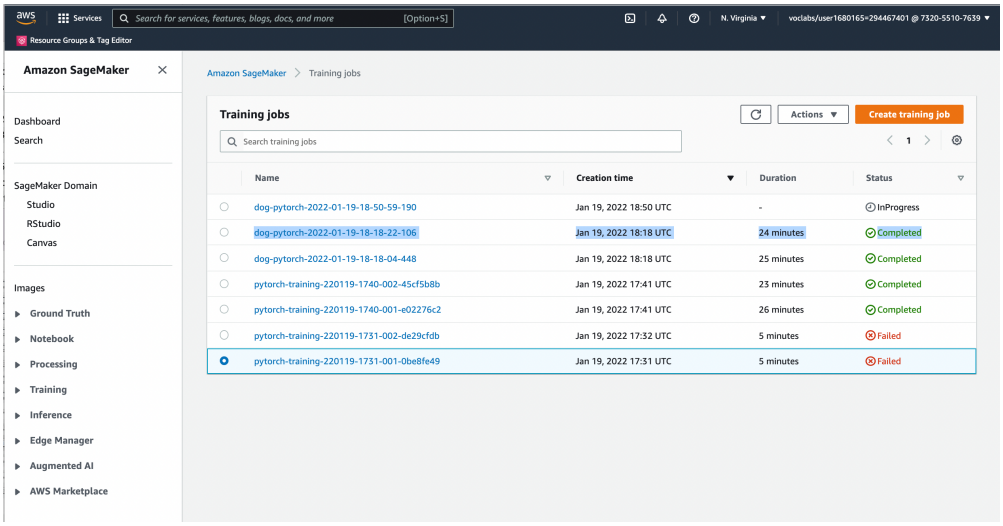
1.5 We deployed two endpoints for inference in both single instance and multi instance and peformed the prediction for the same data and we obtained different results as well ad different inference times. We kept the logs of both invocations to see if there is some sensitive difference but inference times were similar. We should instead perform a lot of requests to see how the endpoints latency behaves in case of higher throughput. The code for the single instance:

Creating an Estimator

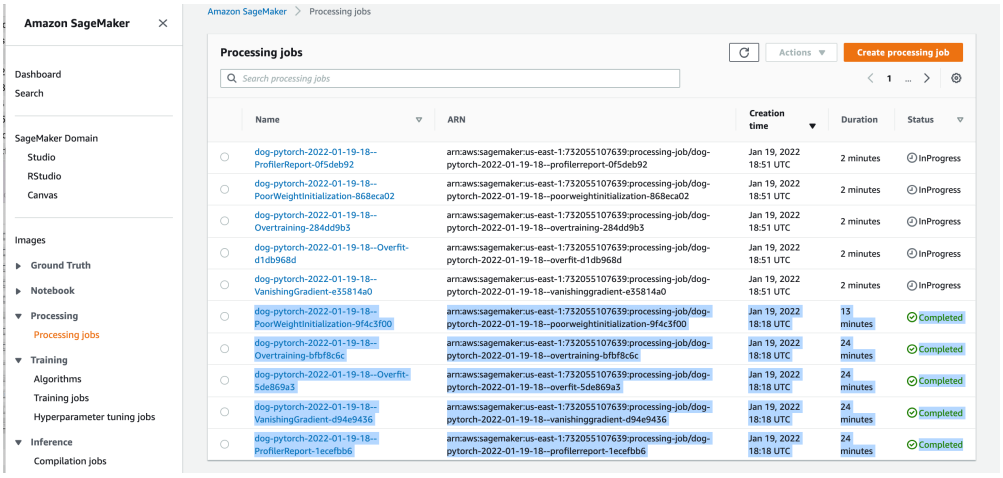
In [29]:
```
#adjust this cell to accomplish multi-instance training
estimator = PyTorch(
    entry_point='hpo.py',
    base_job_name='dog-pytorch',
    role=role,
    instance_count=1,
    instance_type='ml.m5.xlarge',
    framework_version='1.4.0',
    py_version='py3',
    hyperparameters=hyperparameters,
    ## Debugger and Profiler parameters
    rules = rules,
    debugger_hook_config=hook_config,
    profiler_config=profiler_config,
)
```

In [30]:
```
estimator.fit({"training": "s3://udacitysolution-alela/"}, wait=False)
```

The training for the single instance estimator produced by that code:



Process jobs completed to avoid overfitting, poor weight initilizacion, overtraining and vanishing gradients:



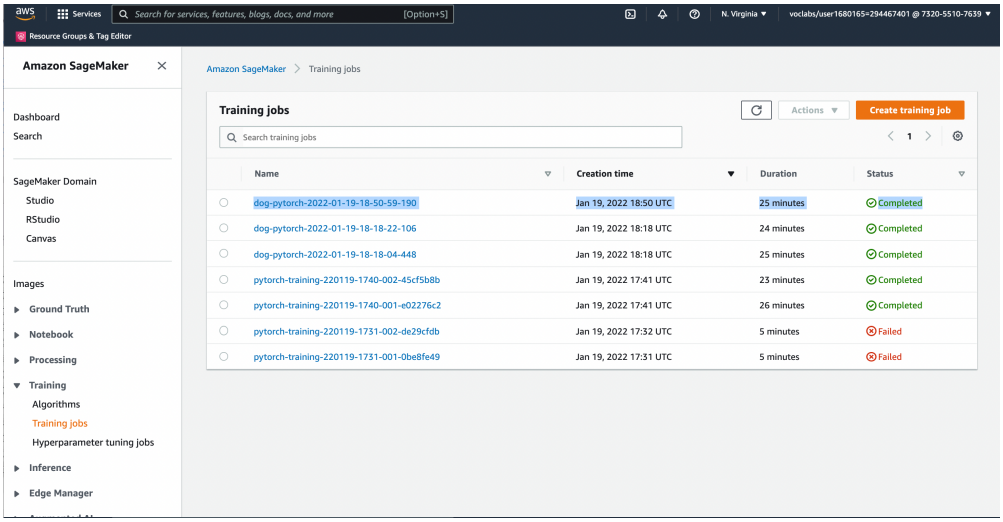
1.6 Analogously, we performed same computation for the multi-instance model:
The code for the multi instance:

Creating an Estimator - Multi-Instance Training,

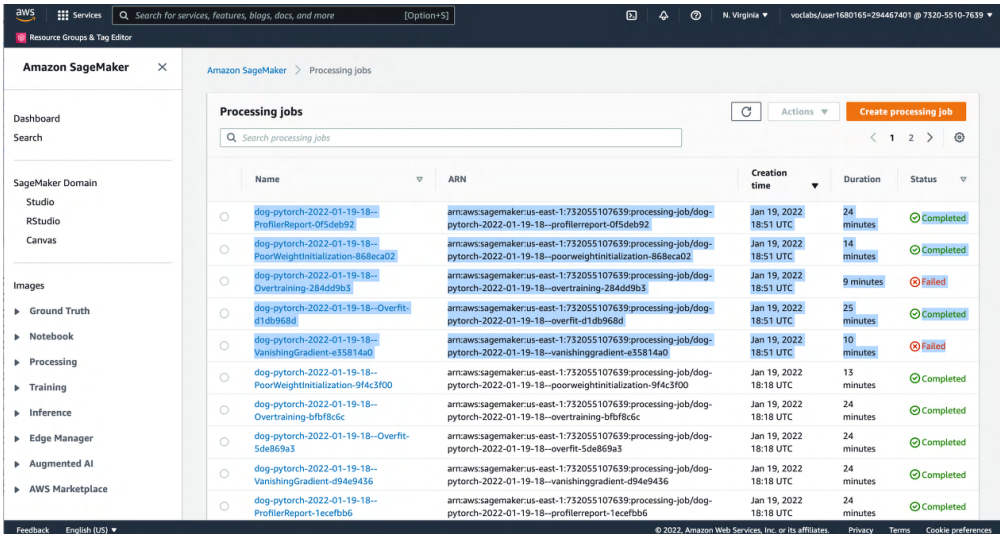
```
In [37]: ###in this cell, create and fit an estimator using multi-instance training
estimator_multi_instance = PyTorch(
    entry_point='hpo.py',
    base_job_name='dog-pytorch',
    role=role,
    instance_count=3,
    instance_type='ml.m5.xlarge',
    framework_version='1.4.0',
    py_version='py3',
    hyperparameters=hyperparameters,
    ## Debugger and Profiler parameters
    rules = rules,
    debugger_hook_config=hook_config,
    profiler_config=profiler_config,
)

In [38]: estimator_multi_instance.fit({"training": "s3://udacitysolution-alela/"}, wait=False)
```

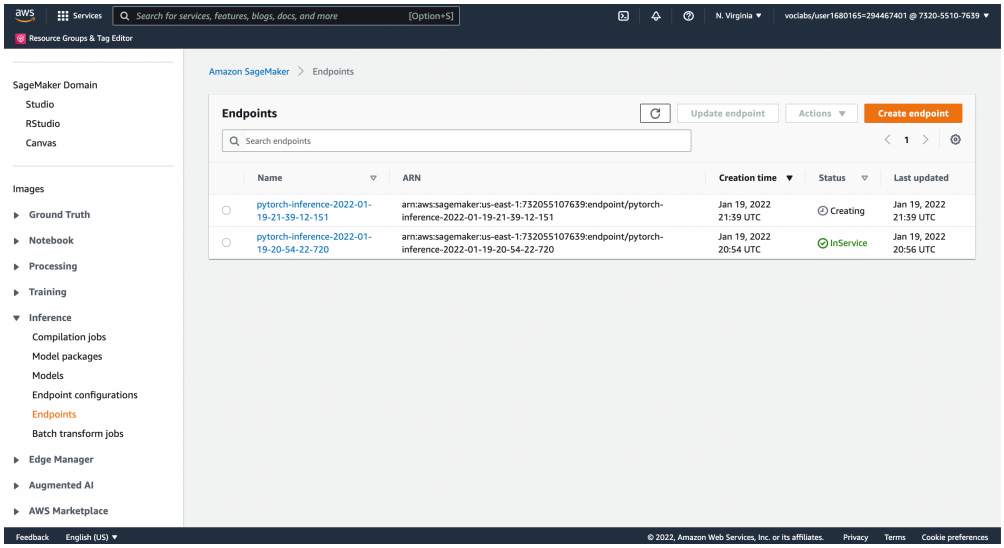
The training for the multi instance estimator produced by that code:



Process jobs completed to avoid overfitting, poor weight initilizacion, overtraining and vanishing gradients:



1.7 After creating this two endpoints, the final version of the notebook is the one in this repo and we deleted the endpoints and stop the notebook instance to avoid charges.

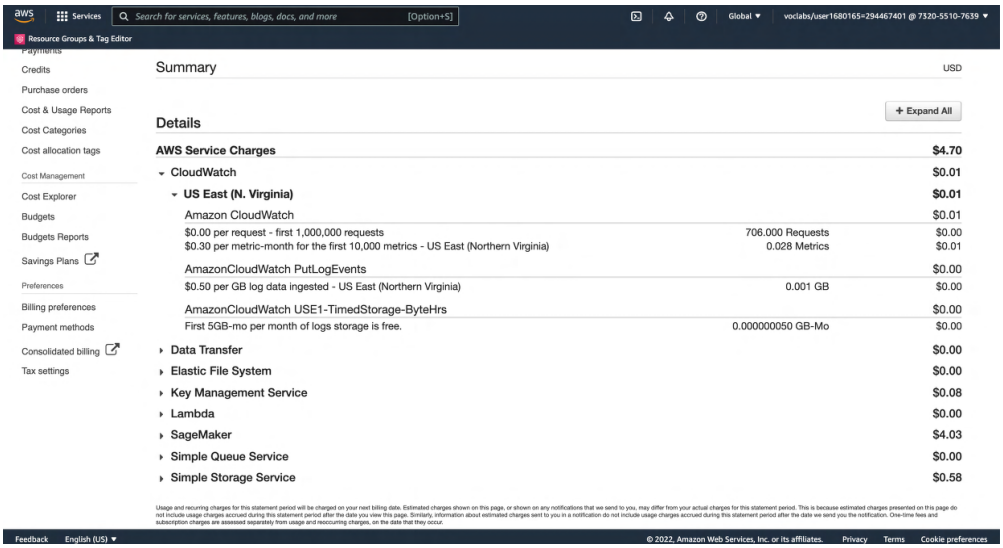


Step 2: Perform a similar task on an EC2 instance.

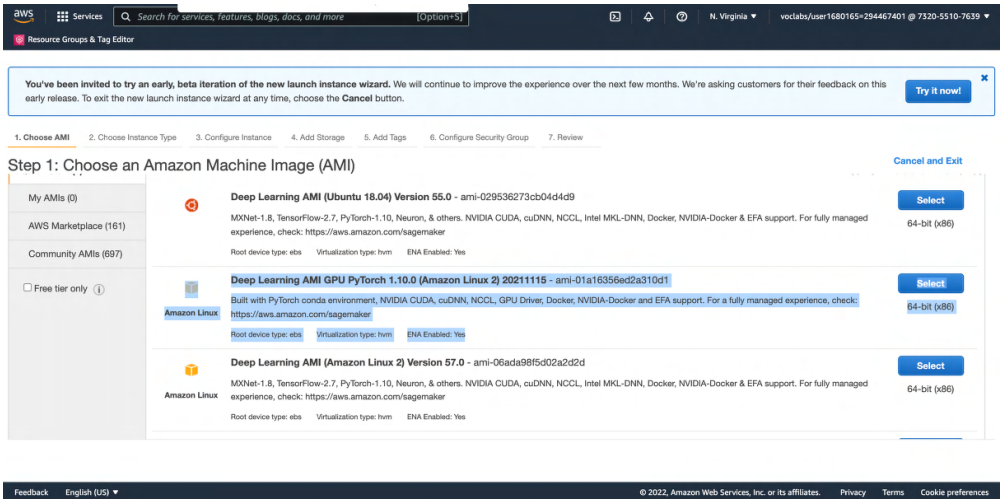
EC2 Instance setup

2.1 We choose first to launch a t2.micro since it's free tier to avoid costs, if I consider it insufficient, I will retry with a larger instance. Anyway the load is not in the EC2, as it was not on the notebook in sagemaker, but in the jobs launched for hpo and training.

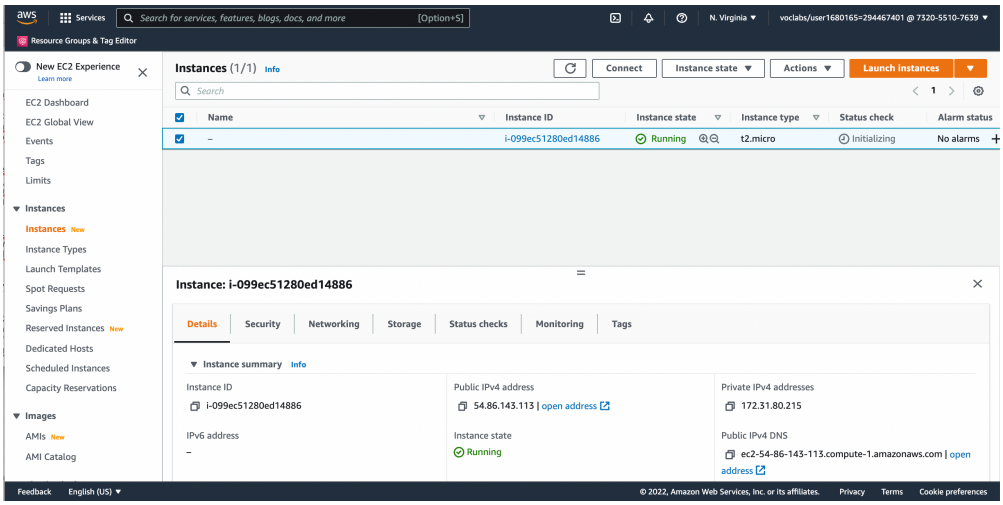
To compare: In the sagemaker task we used ml.t2.medium for the notebook (very light work) and two ml.m5.xlarge for the trainings and ml.m5.large for inferences. The total costs of performing the tasks with sagemaker were $4.03 The total costs of EC2 using same combination of resources were much less than that, but the jobs were different too.



I choose the AMI Amazon Deep Learning because it comes with ML learning environment integrated already.

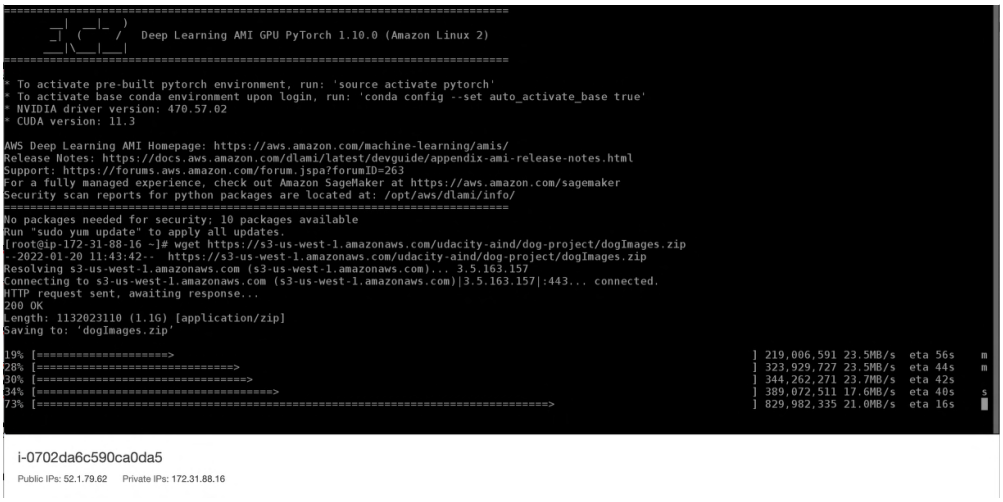


And finally launched the instance:



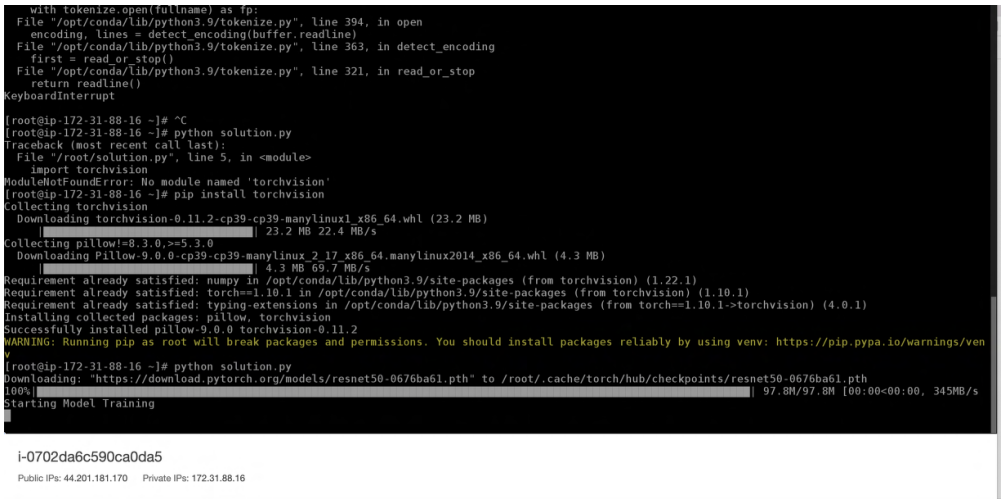
Later on, it turned out that for amazon deep learning free tier is not available for the Amazon Deep Learning AMI and when installing torch by doing: pip install torch there was a memory problem. For that reason I stop the instance and I re launched a ml.t3.medium and connected to this new instance with the same information I had in the t2.micro in a matter of seconds.

2.2 I created the dir TrainedModels and downloaded and unzipped there the file: https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/dogImages.zip using wget and unzip commands



2.3 Created the file solution.py and I pasted the contents of the scrip ec2train1.py

2.4 Run the solution.py and took a screenshot of the model into the TrainedModels directory After inspecting the code in solution.py I can see that it performs the same tasks that were performed in the notebook of step 1 (train_and_deploy-solution.ipynb. It was adapted to work in a typical linux distro but with some changes as follows:



Considerations about the code into the solution.py file:

The code resembles the one used in hpo.py, but it has no smdebug module to perform the final debugging, so the result will be less effective. As well, hyperparameters are fixed, so there is no hyperparameter optimization. Also, this code does not perform the deploy of the endpoint. All that will have to be worked later.

<img src="screenshots/Step1/ width="80%">

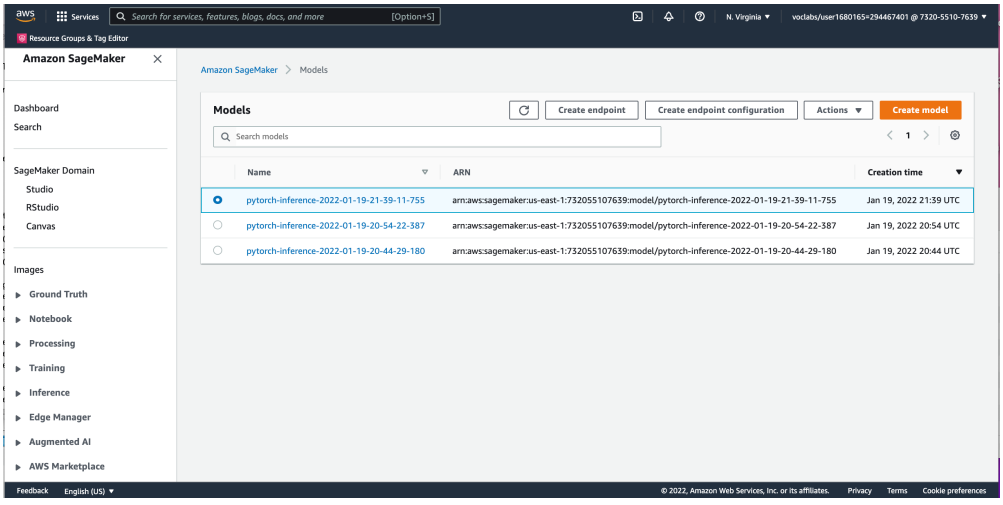
2.5 Finally the job ended as follows: Execution time start 6:57 7:24 ended. And the proof of the job run well is the model saved into the directory:

i-0702da6c590ca0da5

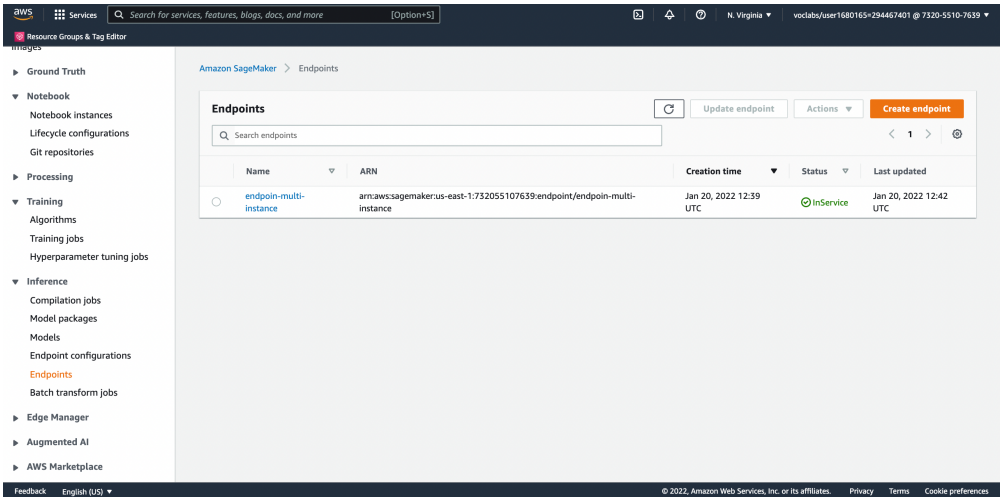
Public IPs: 44.201.181.170 Private IPs: 172.31.88.16

Step 3: Create a Lambda function that will consume your model inference capabilites via endpoints.

3.1 For this task I had to re create the endpoint I deleted yeasterday. I have the models for the endpoint configuration created for both multi-instance and single-instance I went to models in SageMaker and all the models created were there. I decided to use the multi-instance one:

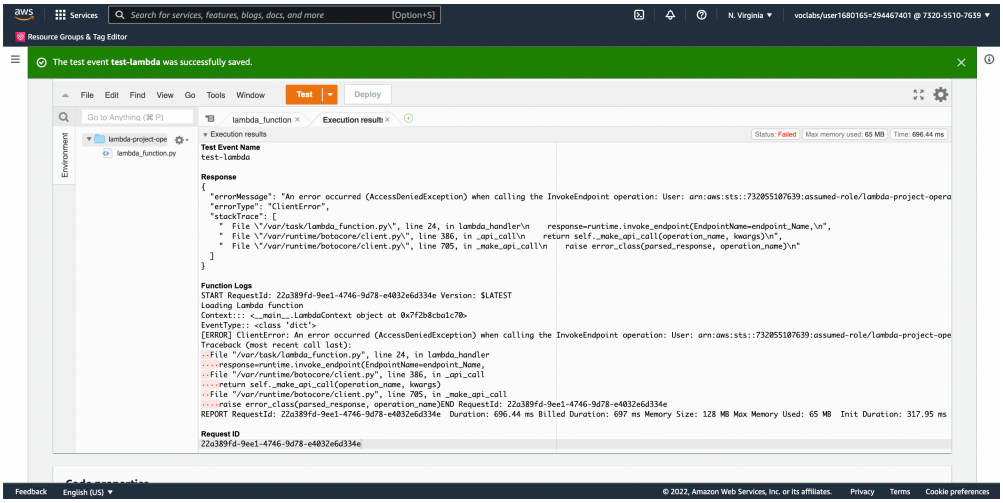


3.2 Then I went to endpoints on SageMaker and created the enpoint using the multi-instance model shown above and I choose a new name for the endpoint.

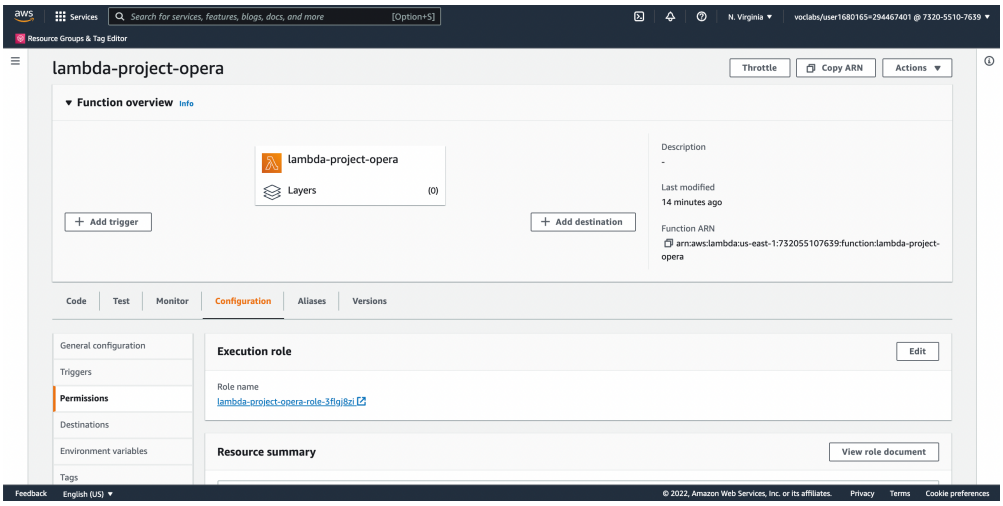


Step 4: Security and testing.

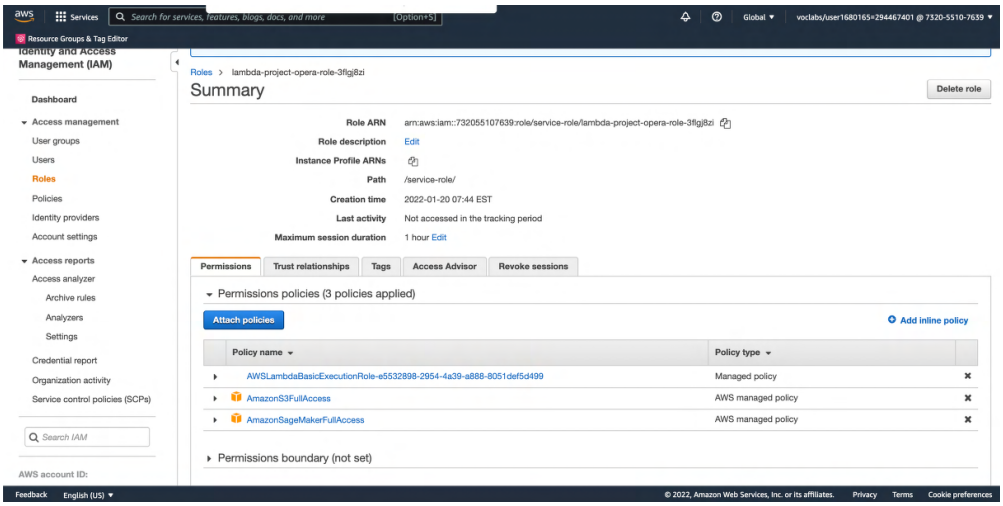
4.1 I tested the lambda funcion using the following test: {"url": "https://s3.amazonaws.com/cdn-origin-etr.akc.org/wp-content/uploads/2017/11/20113314/Carolina-Dog-standing-outdoors.jpg" } But I run into an error when testing.



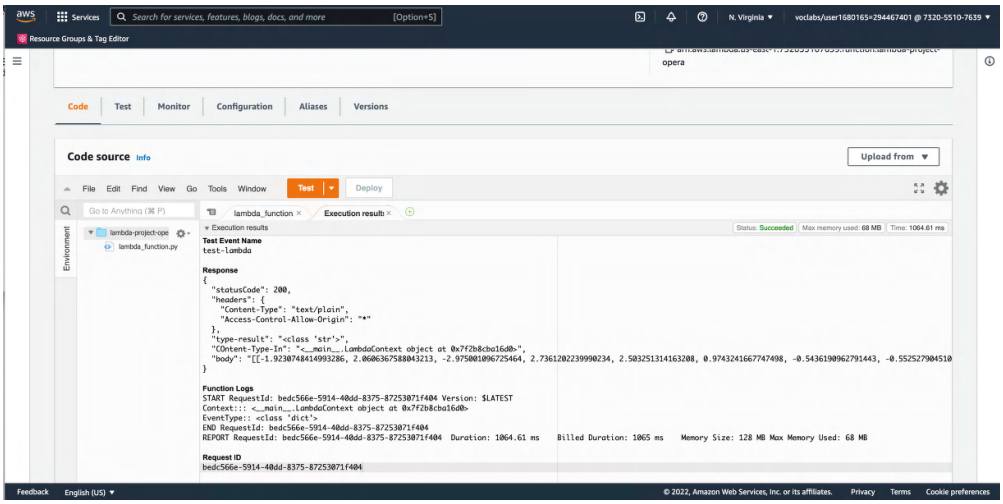
This error was caused because of Lambda running on a rol that has no permissions over SageMaker nor Over s3.



To solve that we added the policies for sagemakerfullaccess and s3fullaccess to the execution role.



4.2 I rerun the test and now it worked!



Here is the complete response from the endpoint:

Test Event Name test-lambda

Response { "statusCode": 200, "headers": { "Content-Type": "text/plain", "Access-Control-Allow-Origin": "*" }, "type-result": "<class 'str'>", "COntent-Type-In": "<**main**.LambdaContext object at 0x7f2b8cba16d0>", "body": "[[-1.9230748414993286, 2.0606367588043213, -2.975001096725464, 2.7361202239990234, 2.503251314163208, 0.9743241667747498, -0.5436190962791443, -0.552527904510498, -6.05665397644043, 1.721237063407898, 2.425633668899536, 1.1596161127090454, 0.23340359330177307, 1.1787495613098145, -1.682868480682373, 0.04093865305185318, -3.0608670711517334, -1.0202009677886963, 0.5772457122802734, 2.3727235794067383, 1.725218415260315, -0.061895087361335754, -0.507676362991333, -3.4325389862060547, -2.3014237880706787, -6.103825092315674, 2.229447841644287, -1.3559074401855469, 0.214759960770607, -0.4876475930213928, 2.2932322025299072, -1.8778115510940552, -2.646745443344116, -0.4442533254623413, -1.4839411973953247, 0.018974244594573975, 0.5177682042121887, -0.4669727385044098, -0.7576069235801697, -1.7498409748077393, 1.264542818069458, 1.4518052339553833, 2.3175251483917236, 0.6133086681365967, 2.315450429916382, -2.0466291904449463, 1.5150392055511475, -0.7387037873268127, 0.36982569098472595, -0.008169978857040405, 1.6424503326416016, -2.1007399559020996, -2.434668779373169, -0.008525431156158447, -4.083022117614746, -0.00435736775398254, -2.7408454418182373, -0.48699548840522766, -3.6401796340942383, -1.6554871797561646, -1.546553373336792, -1.7185020446777344, -2.040588855743408, -4.549625396728516, -2.8638806343078613, -2.800689697265625, 1.9408971071243286, -0.13130821287631989, -1.1525102853775024, -1.2089629173278809, 3.5017807483673096, -2.380239725112915, -1.1759413480758667, -1.4099303483963013, -2.453126907348633, -2.063462018966675, -3.7739031314849854, -1.0499930381774902, 1.1904747486114502, -1.380522608757019, 1.0462281703948975, -5.719635009765625, 0.3957575261592865, 1.9719444513320923, -4.87777853012085, -5.302970886230469, -0.5300191640853882, -3.840769052505493, -2.534928798675537, 1.4356107711791992, -3.2125887870788574, 0.5481349229812622, -3.9147355556488037, -1.0964760780334473, 1.0998213291168213, 0.7646914124488831, -1.695220708847046, -1.466501235961914, -3.896580696105957, -4.843873023986816, -9.731215476989746, -2.6416592597961426, 1.6428296566009521, -2.0402002334594727, -0.4763360619544983, -0.48405128717422485, -2.3959767818450928, 1.7910501956939697, 3.1037676334381104, 1.1668872833251953, -0.15623413026332855,

0.057766884565353394, -3.2649059295654297, -1.6079130172729492, -3.212416410446167, 0.8164639472961426, -0.5811924934387207, 2.154860258102417, -2.4100425243377686, 0.4797557294368744, -0.26428431272506714, -1.8739787340164185, -0.9146941304206848, 0.05285979062318802, -6.775721073150635, 0.4937722682952881, -3.6679766178131104, 1.0705149173736572, -0.2790021300315857, -3.6403069496154785, -7.05629825592041, -2.225168466567993, -5.770076274871826]]" }

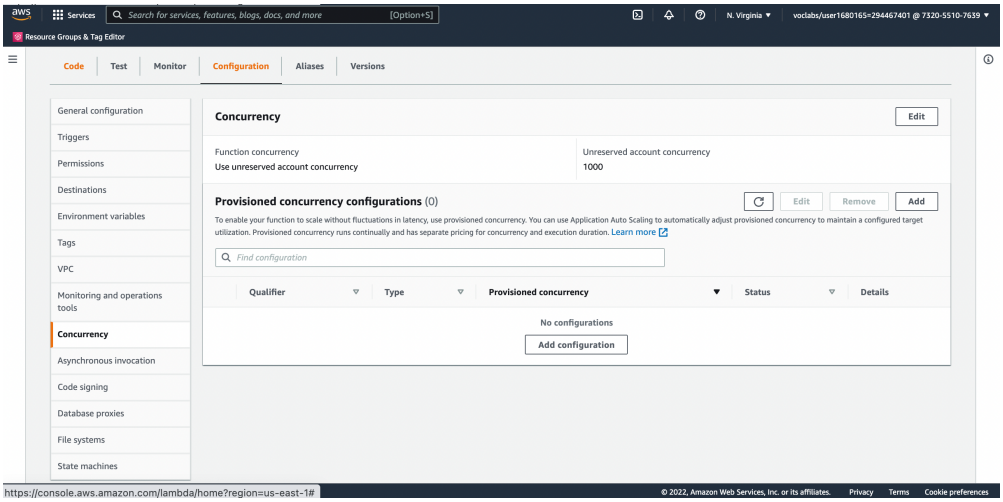
Function Logs START RequestId: bedc566e-5914-40dd-8375-87253071f404 Version: $LATEST Context::: <**main**.LambdaContext object at 0x7f2b8cba16d0> EventType:: <class 'dict'> END RequestId: bedc566e-5914-40dd-8375-87253071f404 REPORT RequestId: bedc566e-5914-40dd-8375-87253071f404 Duration: 1064.61 ms Billed Duration: 1065 ms Memory Size: 128 MB Max Memory Used: 68 MB

Request ID bedc566e-5914-40dd-8375-87253071f404

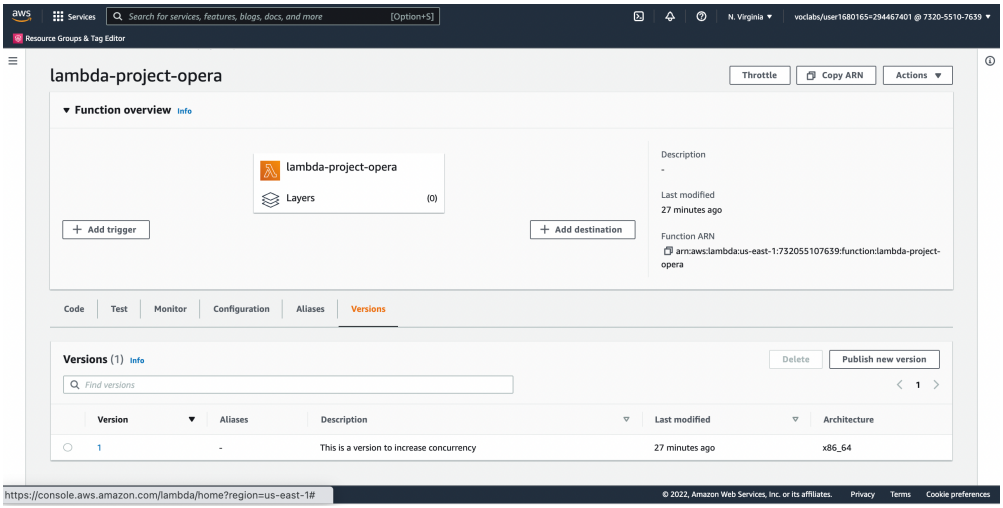
Step 5: Set up concurrence for your lambda function and auto-scaling for your deployed endpoint.

Concurrecy

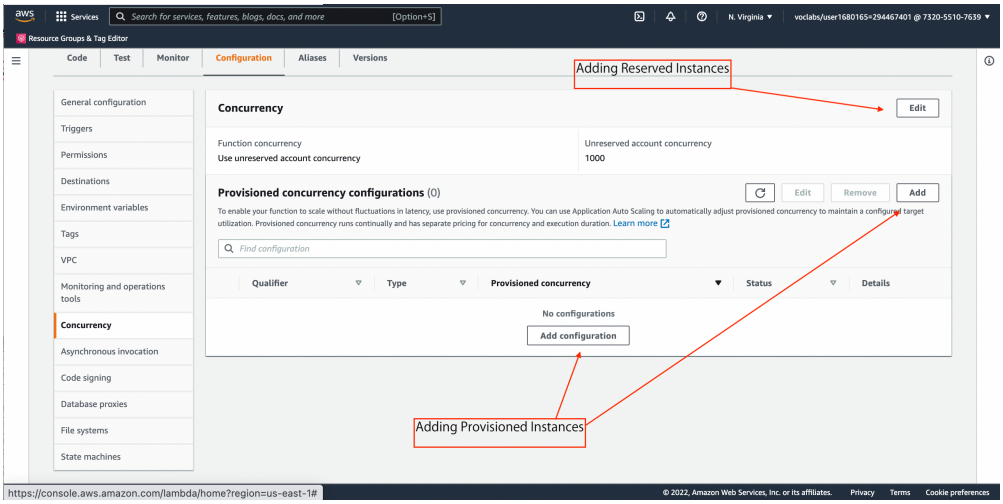
5.1 To let the lamdba answer requests in a parallel fashion we added concurrency from the configuration of the lambda function. The concurrency setup can be located in left menu of the lambda configuration pane.



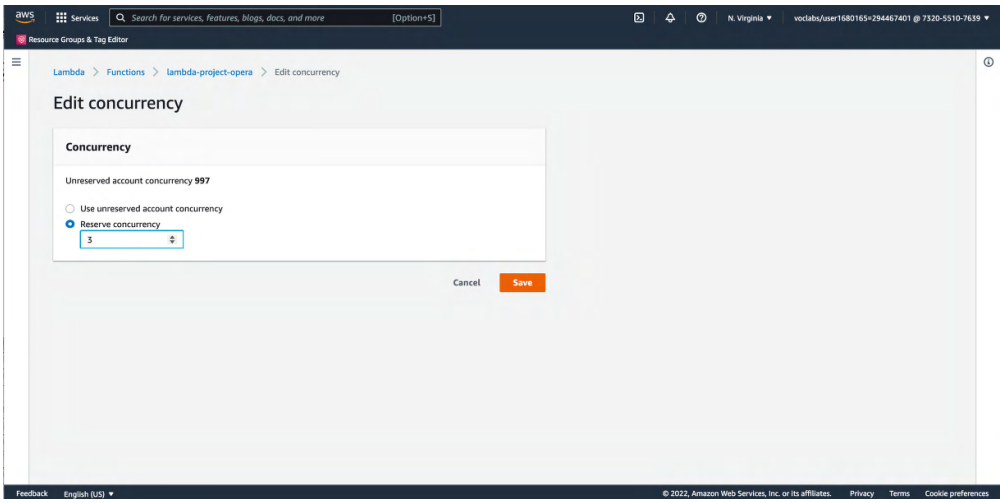
5.2 First we must set up the version1 of the function and using the edit button in Concurrency pane we selected reserved concurrency of 3 (to avoid costs while the enpoint is deployed).



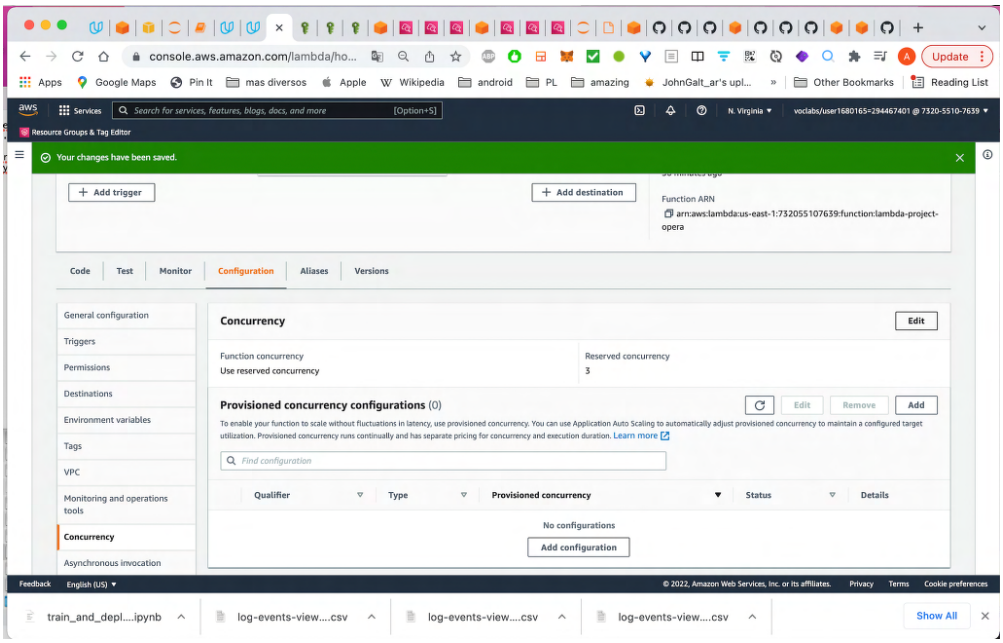
5.3 There are two tipes of concurrency settings: Reserved concurrency price is low, but latency could be high. Provisioned concurrency is always available and so, more costly.



5.4 I will configure concurrency only for reserved instances:

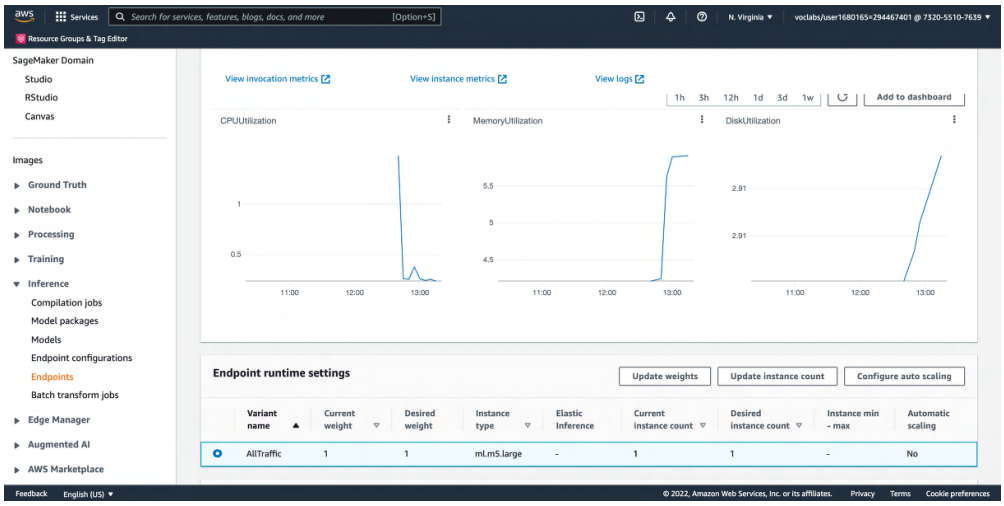


5.5 Finally the reserved concurrency setting is ready.

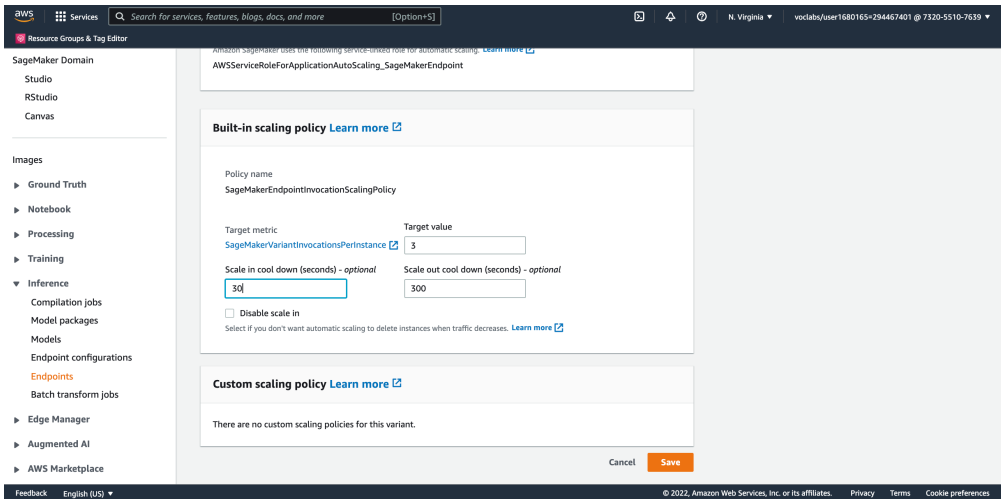


Auto scaling

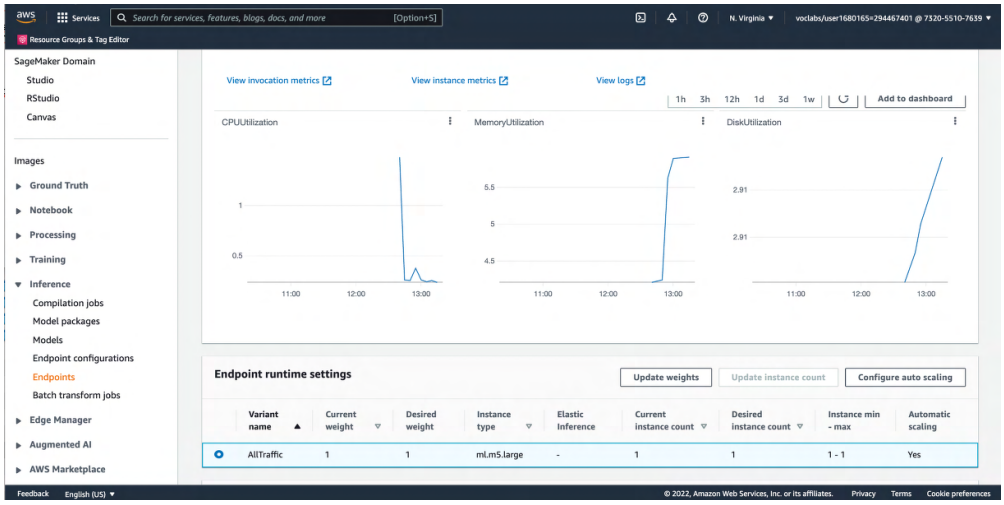
5.6 I will scale our endpoint to scale to more instances and with some short scale in cool down time, and some longer scale out cool down time.



5.7 I choose 3 instances of auto scaling as well as 3 reserved concurrency to be able to deal with triple increase in demand.



5.8 Finally auto-scaling is set!



Final words:

AWS was able to let us quickly create this model and deploy it in a way that is easily scalable and secure as shown in Step 3 and Step 4. This is one of the strongest features of the AWS Sagemaker for quick and professional Machine Learning solutions.