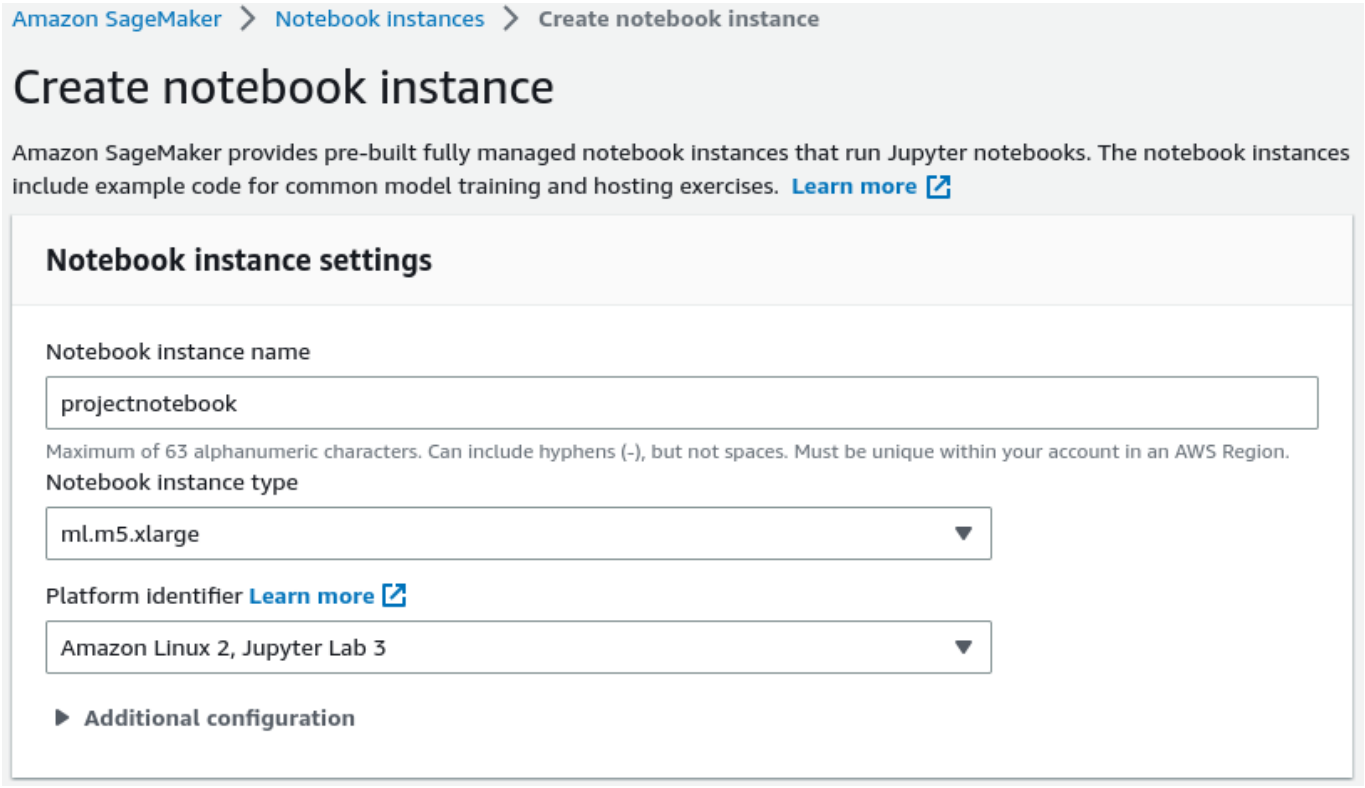
Operationalizing an AWS ML Project

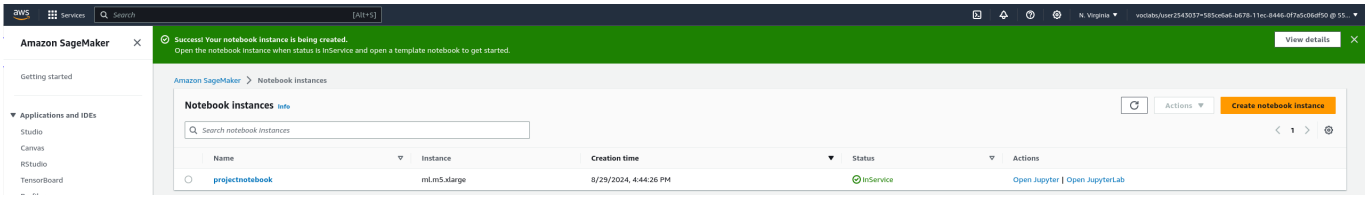
Step 1 - Training and Deplyment on SageMaker

Initial Setup

In SageMaker, I go to `Notebooks -> Create notebook instance`. I choose `ml.m5.xlarge` as the instance type considering its configuration of `2` vCPU, `8GB` Memory, and `$0.115` per hour. This setting is sufficient for this project, and the price is reasonable.



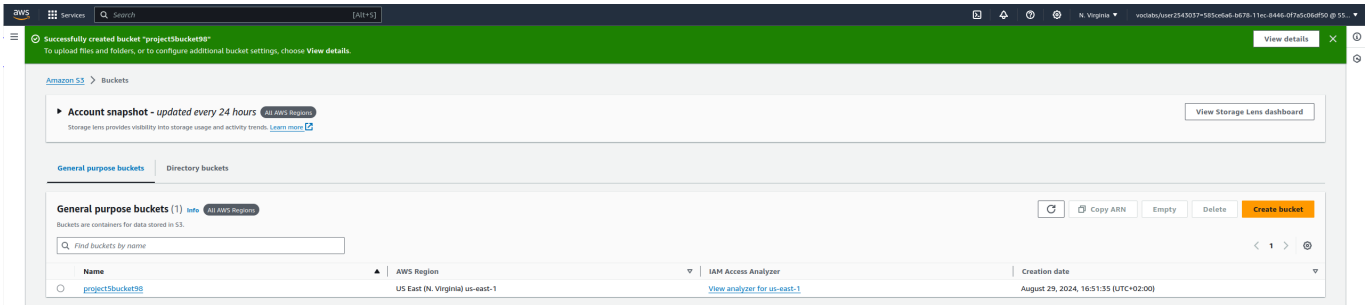
Bellow is the created notebook instance called `projectnotebook`.



In the notebook instance, I upload three files `train_and_deploy-solution.ipynb`, `hpo.py`, and `inference2`, then open the notebook `train_and_deploy-solution.ipynb`.

Download data to an S3 bucket

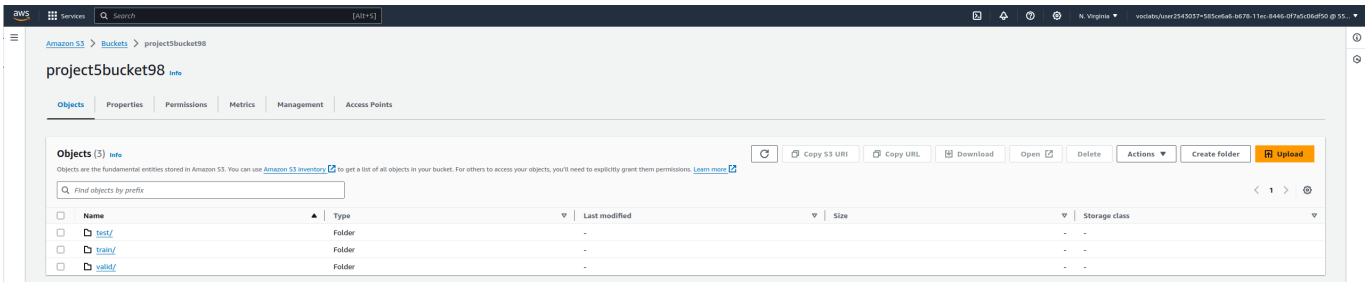
Next, I go to S3 and create a new bucket under the name `project5bucket98`.



In the notebook, I run the following cell to upload the dataset to S3 bucket.

```
!wget https://s3-us-west-1.amazonaws.com/udacity-aind/dog-
project/dogImages.zip
!unzip dogImages.zip
!aws s3 cp dogImages s3://project5bucket98/ --recursive
```

I check the S3 bucket after the dataset has been uploaded.



Train and Deployment```

I continue running cells to perform hyperparameter tuning. I use the file `hpo.py` as entry point to the estimator, which contains the code to train model with different hyperparameters values.

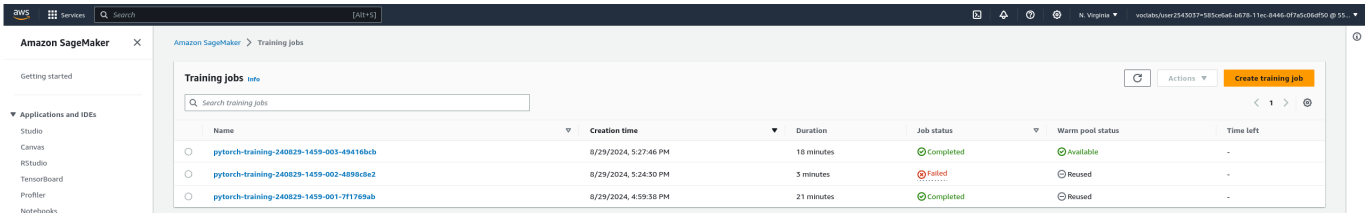
```
estimator = PyTorch(
    entry_point="hpo.py",
    base_job_name='pytorch_dog_hpo',
    role=role,
    framework_version="1.4.0",
    instance_count=1,
    instance_type="ml.g4dn.xlarge",
    py_version='py3'
)

tuner = HyperparameterTuner(
    estimator,
    objective_metric_name,
    hyperparameter_ranges,
    metric_definitions,
    max_jobs=2,
    max_parallel_jobs=1,  # you once have one ml.g4dn.xlarge instance
available
    objective_type=objective_type
```

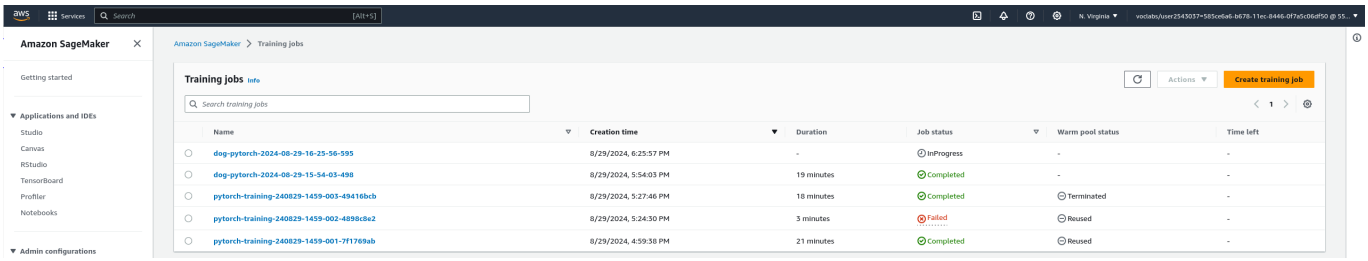
```
)

tuner.fit({"training": "s3://project5bucket98/"})
```

Two training jobs are created for this hyperparameter tuner. I go to the page `Training jobs` to check the status of those jobs. They are completed successfully.

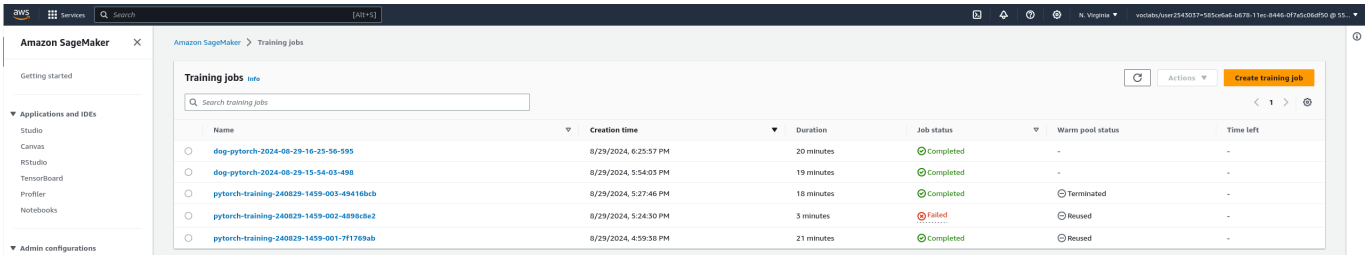


Afterwards, I take the best hyperparameters to train the model (single-instance training).

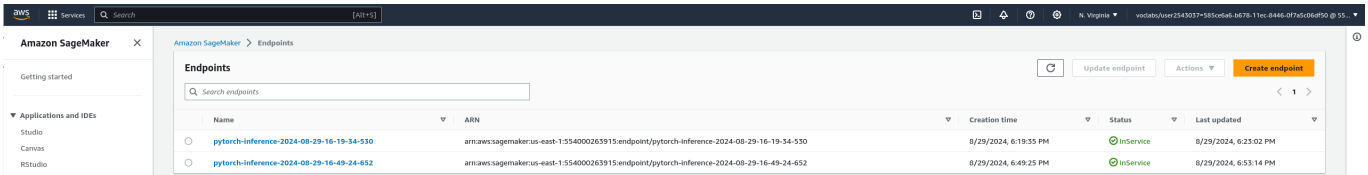


Then the model is deployed with one initial instance and a instance type of `ml.m5.large`. I check the endpoint at `Inference -> Endpoint`.

Next, I perform the multi-instance training by setting `instance_count=5` in the estimator.



I deploy the model again, creating another endpoint.

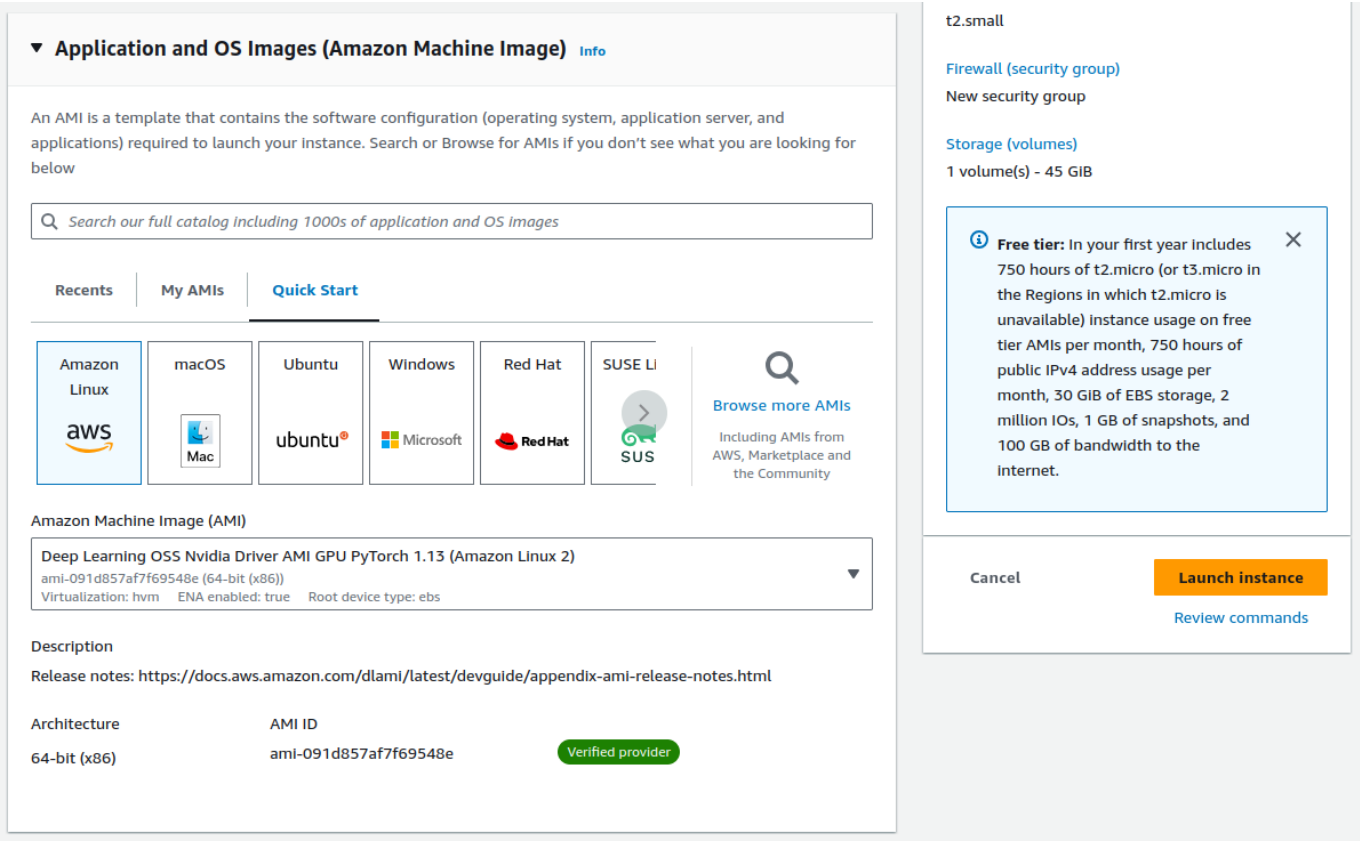


Step 2 - EC2 Training

EC2 Setup

I go to EC 2 and launch a new instance under the name `project5instance98`. I choose the Amazon Machine Image of `Deep Learning OSS Nvidia Driver AMI GPU PyTorch 1.13 (Amazon Linux 2)` because we need the PyTorch library to train the model; and the the instance of `t2.small`, which is

sufficient for this project.



Preparing for EC2 model training

Now I connect to the created instance, get the dataset and extract it. I also create a folder `TrainedModels` where the trained model will be saved.

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

Then I created a Python file, paste the code from `ec2train1.py` into it and save it.

```
vim solution.py
# Ctrl+Shift+V to paste the code
:set paste
:wq!
```

I activate the PyTorch environment and train the model

```
source activate PyTorch
python3 solution.py
```

I check to see if the trained model has been saved. I see that there is a model file `model.pth` in the folder `SavedModels`.

Comparison between EC2 and SageMaker Notebook for training models

EC2 instances can be easily scaled up or down based on computing needs, or customized to meet specific requirements such as the Deep Learning framework (PyTorch or TensorFlow), number of CPUs, memory size. Additionally, the GPU and EC2 instances can be optimized for high-performance computing, reducing the training time on large models.

Notebook instances can be quickly setup as they have pre-configuration with common Machine Learning frameworks and libraries. it can also be easily connected and integrated with others AWS services such as SageMaker and S3, providing lots of tools for Machine Learning engineering.

Step 3 - Lambda Functions Setup

I go to Lambda and create a function with the code taken from the file `lambdafunction.py`. This function operates as follows:

1. Set up the runtime from `boto3`, and the endpoint name created in Step 1.

```
runtime=boto3.Session().client('sagemaker-runtime')
endpoint_Name='pytorch-inference-2024-08-29-16-19-34-530'
```

2. Get the input event as a `json` script containing the image on which we will make prediction. The context parameter is the environment configuration.

```
def lambda_handler(event, context):
```

3. Invoke the endpoint by progating the endpoint name and the event.

```
bs=event
response=runtime.invoke_endpoint(EndpointName=endpoint_Name,
                                 ContentType="application/json",
                                 Accept='application/json',
                                 #Body=bytearray(x)
                                 Body=json.dumps(bs))
```

4. Get the result from the response returned by the endpoint

```
result=response['Body'].read().decode('utf-8')
 sss=json.loads(result)
```

5. Return the json script containing the prediction output values

```
return {
     'statusCode': 200,
     'headers' : { 'Content-Type' : 'text/plain', 'Access-Control-
Allow-Origin' : '*' },
     'type-result':str(type(result)),
     'COntent-Type-In':str(context),
     'body' : json.dumps(sss)
     #'updated_result':str(updated_result)

     }
```

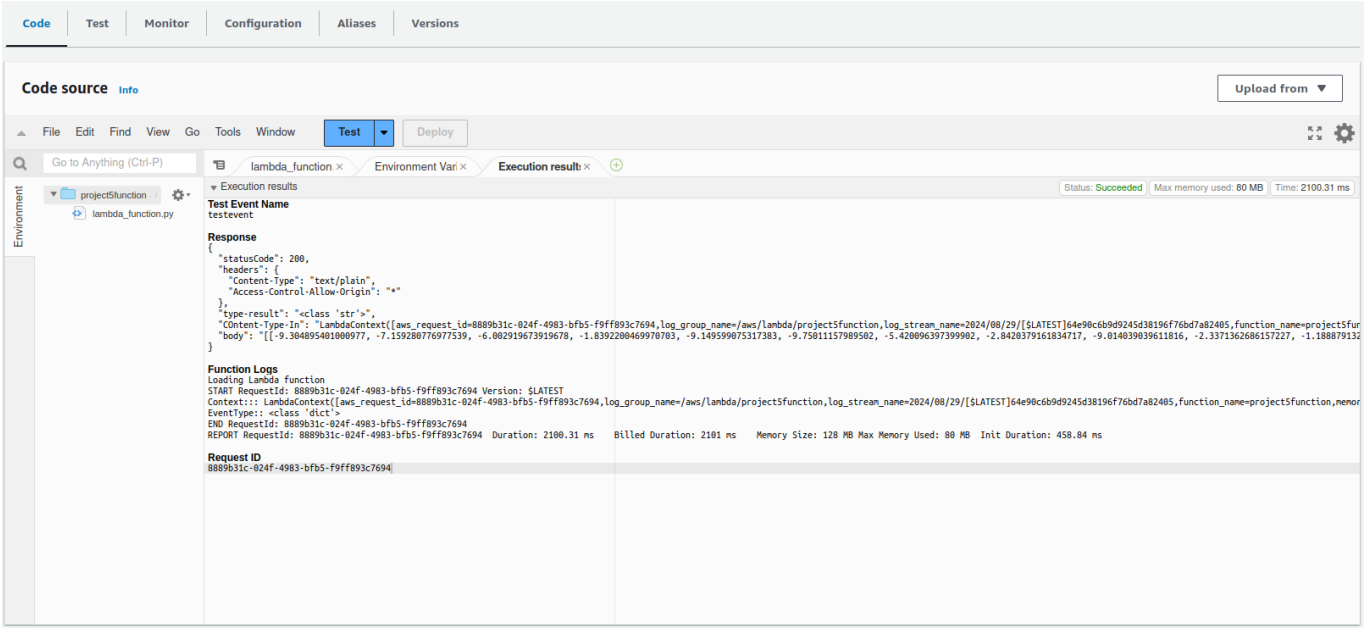
Step 4 - Security and Testing

Lambda function security

I go to the `Configuration` tab `--> Permission`, then open the role name to go to its IAM setting. I add the `SageMakerFullAccess` policy to this role. Now with the right permission I create a test event as the json below, and test the Lambda function.

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

I get the test result showing `133` numbers representing class predictions of the image.

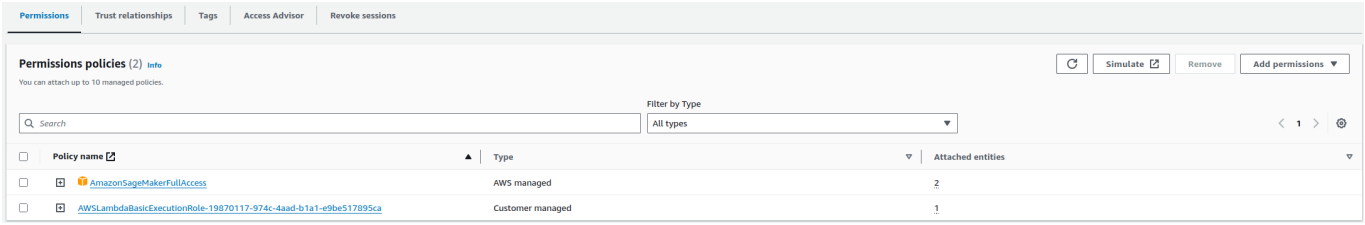


The full list of predictiona values is:

```
[-9.304895401000977, -7.159280776977539, -6.002919673919678,
-1.8392200469970703, -9.149599075317383, -9.75011157989502,
-5.420096397399902, -2.8420379161834717, -9.014039039611816,
-2.3371362686157227, -1.188879132270813, -7.441051959991455,
-2.9033398628234863, -0.8761371374130249, -8.392709732055664,
-7.464737415313721, -12.484731674194336, -2.468195915222168,
-8.797133445739746, -0.0031809359788894653, -5.83886194229126,
-2.689816951751709, -11.852859497070312, -8.563738822937012,
-5.85135555267334, -12.539071083068848, -2.5635366439819336,
-6.393071174621582, -7.177112102508545, -3.3391294479370117,
-5.723604202270508, -3.902200222015381, -8.811140060424805,
-6.649082660675049, -8.800968170166016, -9.328686714172363,
-8.032840728759766, -4.764562129974365, -2.9750850200653076,
-4.016922473907471, -4.654715061187744, -5.58418607711792,
-1.0925631523132324, -5.706500053405762, -1.8298832178115845,
-12.044698715209961, -2.522261381149292, -0.7717123031616211,
-4.545398712158203, -2.799130916595459, -4.876896858215332,
-11.03920841217041, -10.338143348693848, -5.638136386871338,
-6.844571590423584, -2.63582444190979, -6.770293235778809,
-10.125781059265137, -4.720673561096191, -2.937652587890625,
-10.192708015441895, -13.118915557861328, -13.984131813049316,
-11.3939208984375, -6.663053035736084, -9.275955200195312,
-1.295379877090454, -9.144013404846191, -2.4341511726379395,
-2.939272880554199, -0.8635281324386597, -6.562072277069092,
-9.162252426147461, -9.462575912475586, -6.846124172210693,
-3.727454662322998, -13.146017074584961, -4.088266372680664,
-12.302526473999023, -8.321849822998047, -1.2684651613235474,
-10.615811347961426, -3.191828727722168, -3.720151901245117,
-13.102355003356934, -6.731252670288086, -3.3874571323394775,
-7.175037860870361, -7.370479583740234, -3.1150946617126465,
-9.259984970092773, -13.379631996154785, -8.279791831970215,
-10.015170097351074, -10.45757007598877, -2.994457244873047,
-6.950647830963135, -7.235556125640869, -11.54786491394043,
-10.156283378601074, -12.082859992980957, -1.599365472793579,
-5.060778617858887, -8.156784057617188, -7.831529140472412,
-12.820998191833496, -4.349366664886475, -3.0208377838134766,
-6.098126411437988, -3.4641664028167725, -4.033401012420654,
-2.0112905502319336, -12.100760459899902, -8.072874069213867,
-6.127520561218262, -3.024576425552368, -11.435108184814453,
-3.0748751163482666, -11.886699676513672, -1.410701870918274,
-3.4078354835510254, -4.448908805847168, -6.899403095245361,
-5.110894203186035, -10.908452987670898, -9.657973289489746,
-3.7598214149475098, -1.2964024543762207, -11.023666381835938,
-9.663758277893066, -10.409066200256348, -2.5322132110595703,
-6.654378414154053]
```


Other security considerations

Below is the acctached policies to my Lambda function's role. It only has two policies: `AmazonSageMakerFullAccess`, which is needed to access the endpoint in SageMaker, and `AWSLambdaBasicExecutionRole`, which is the basic role for every Lambda function.

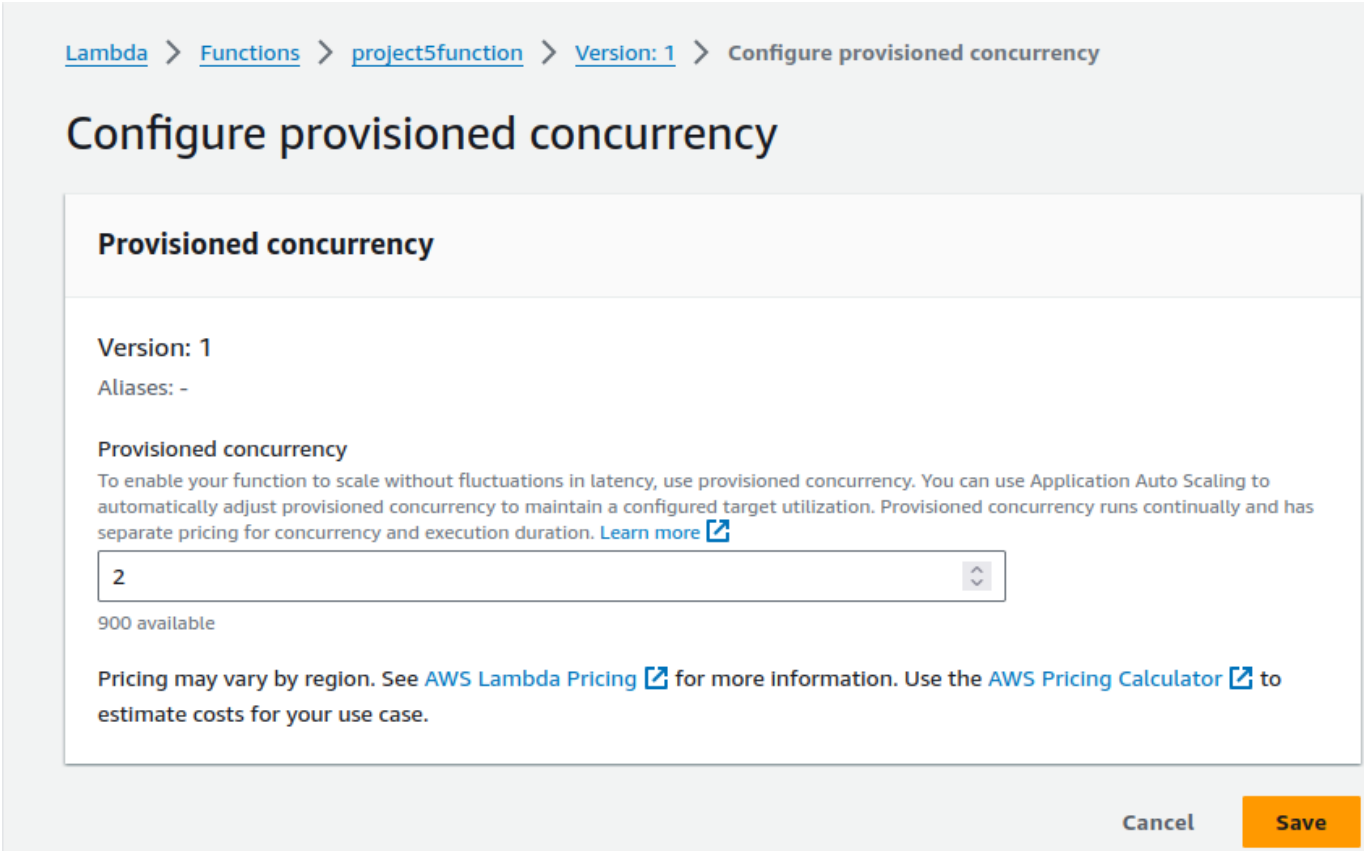


One potential security vulnerability is the "FullAccess" to SageMaker from Lambda Fucntion. It could expose the environment to unnecessary risk. It is essential to ensure each role has only the minimum permissions necessary for its tasks.

Step 5 - Concurrency and Auto-scaling

Concurrency

By default a Lambda Function can only respond one request at once. Concurrency makes it possible to respond to multiple requests at once. To add concurrency, I go to `Configuration --> Provisioned concurrency --> Edit`. I set the concurrency to `2` so that this Lambda Functions can handle two requests at once.



Auto-scaling

With auto scaling, SageMaker automatically increases or decreases the number of instances, ensuring that we only pay for the instances that are actively running. To enable auto-scaling in SageMaker, I go to `Inference -> Endpoints` and click on the endpoint name. I go to `Setting -> Endpoint runtime settings -> Configure auto scaling`. The minimum and maximum number of instances are set to 1 and 3, respectively. I set the `Target value` wto 20, meaning that when our endpoint receives 20 requests simultaneously, auto-scaling will be triggered, and the number of instances will be increased. The `Scale In` and `Scale Out` parameters are both set to 30 seconds, which controls the amount of time auto-scaling

should wait before increasing or decreasing the number of instances.