To start, the process requires creating a Sagemaker notebook instance. Due to its affordability, I chose an ml.t3.medium instance, as high processing power or extensive RAM is unnecessary. This instance is solely for running notebook code and not for model training or inference. It's crucial to have this instance for the smooth functioning of the code.

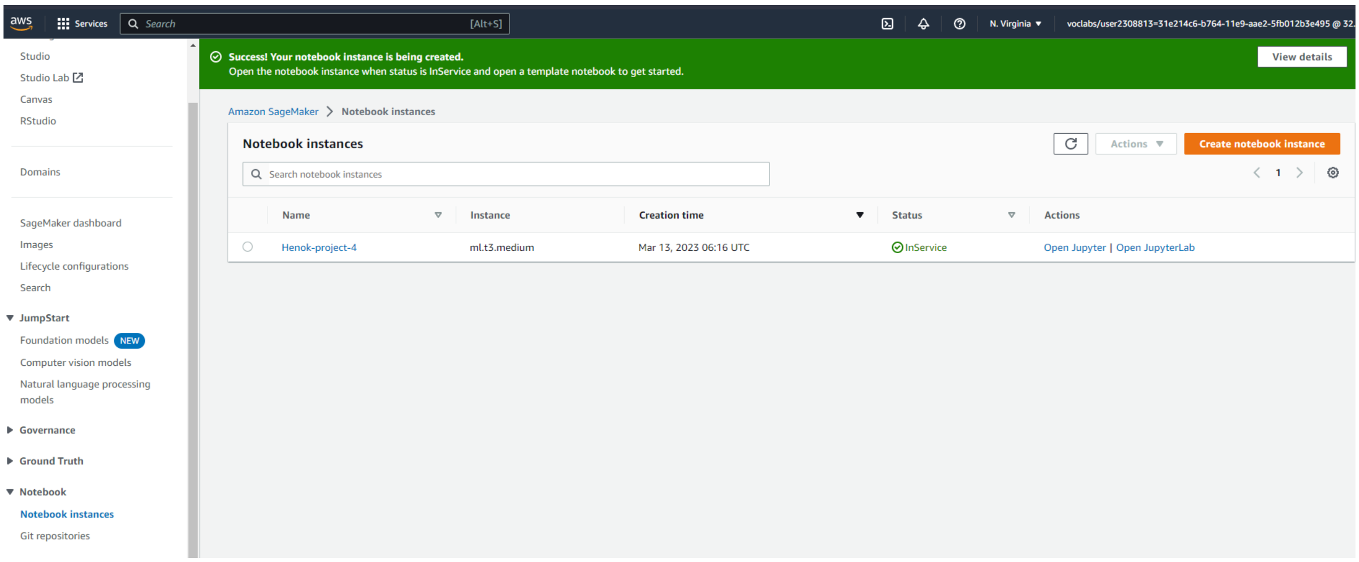


Fig 1. Sagemaker NoteBook Instance.

Then upload code archive starter.zip to the notebook instance to run the experiment.

- `train_and_deploy-solution.ipynb`: a notebook containing code for training and deploying a computer vision model on AWS
- `hpo.py`: this is a Python script that your ***train_and_deploy-solution.ipynb*** notebook can use as its "entry point."
- `lambdafunction.py`: This is a starter Python script that you can use for your Lambda function - remember, it will take a few adjustments before it functions correctly as a Lambda function
- `infernce2.py`: this is a Python file that you can use for deploying your trained model to an endpoint
- `ec2train1.py`: this is a Python file that you can use for training a model on an EC2 instance

To upload and extract the source code files open a Jupiter notebook in the instance, then open the terminal and use the following commands.

```
cd /home/ec2-user/SageMaker
wget https://video.udacity-data.com/topher/2022/June/62a388d0_starter/starter.zip
unzip starter.zip
```

Download data and upload to s3 bucket

The provided dataset is the dog breed classification dataset which can be downloaded from this link.

It contains images of 133 dog breeds. divided into 6680 training images, 835 validation images, and 836 testing images. The first three cells of train_and_deploy-solution.ipynb downloads the dog breed dataset to our AWS workspace. The third cell copies the data to the AWS S3 bucket.

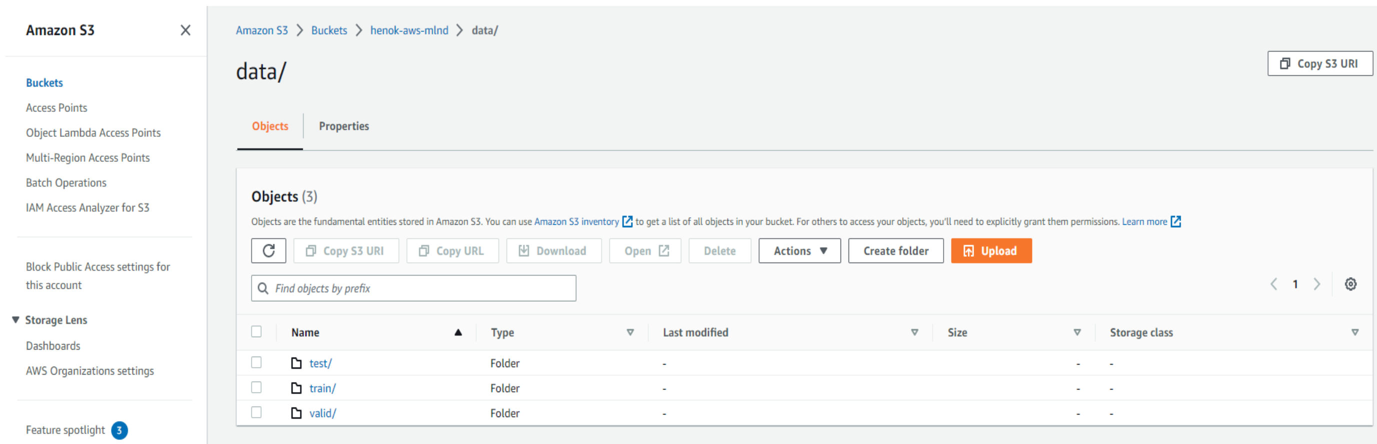


Fig 2. S3 Bucket (File Uploaded)

Training and deployment (Single Instance)

From the fourth to the sixteenth cell of the train_and_deploy-solution.ipynb notebook, I created a tuning job with an instance type ml.g4dn.xlarge, max_jobs=2, and max_parallel_jobs=1; it took approximately an hour to complete.

The best hyperparameters found were `{'batch_size' 64, 'learning_rate' '0.0011225748748143192'}`.

After identifying the best hyperparameters through tuning, I trained the model using an ml.m5.xlarge instance with higher processing power. Next, I executed the cells in the Deployment section of the notebook to establish an endpoint, opting for ml.m5.large as it was suitable for the inference task at hand and allowed me to run it for extended periods without incurring excessive costs, facilitating subsequent project steps and lambda function testing.

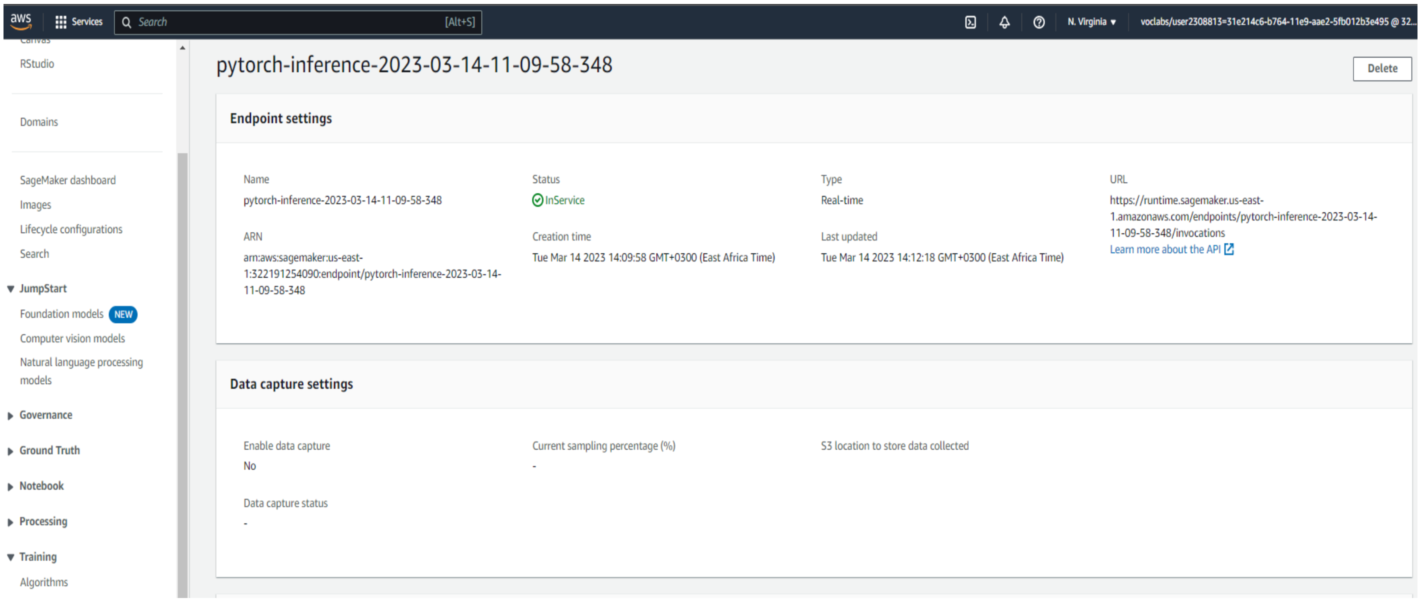


Fig 3. Endpoint

Training and deployment (Multi-Instance)

I created a multi-instance training job by modifying the parameter insance_count=4 to run 4 instances simultaneously for training.

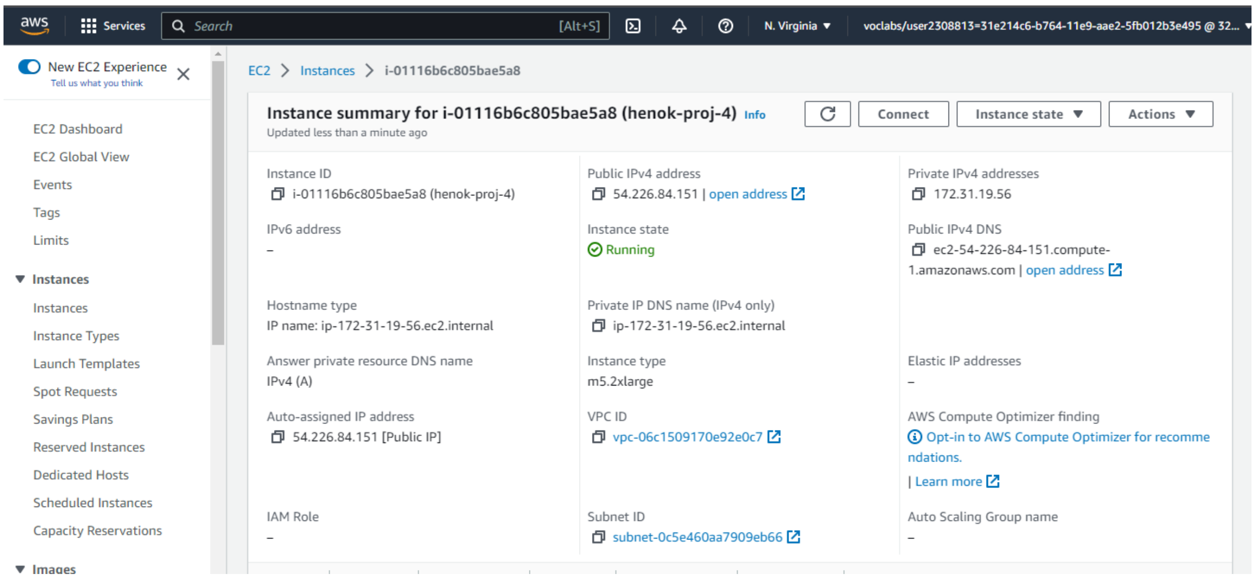
```
estimator_multi_instance = PyTorch(
    entry_point='hpo.py',
    base_job_name='dog-porch,
    role=role,
    instance_count=4,
    instance_type='ml.m5.2xlarge',
    framework_version='1.4.0',
    py_version='py3',
    hyperparameters=hyperparameters,
    ## Debugger and Profiler parameters
    rules = rules,
    debugger_hook_config=hook_config,
    profiler_config=profiler_config,
)
```

Then deployed another endpoint.

EC2 Training

I used t3.2xlarge instances, which are low-cost burstable general-purpose instance types that provide a baseline level of CPU performance with the ability to burst CPU usage anytime for as long as required.

As for the training image, I have used Deep Learning AMI GPU PyTorch 1.13.1 (Ubuntu 20.04) 20230309 to train the model.



After connecting to the EC2 instance, I downloaded the data. I created the model output directory:

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

Created a blank Python file by running the following command in your EC2 Terminal:

```
vim solution.py
```

Paste all of the code from the starter file called `ec2train1.py` into Solution.py

Activated the PyTorch environment that we will use to train our model:

```
Source activate pytorch
```

Run the code

```
python solution.py
```

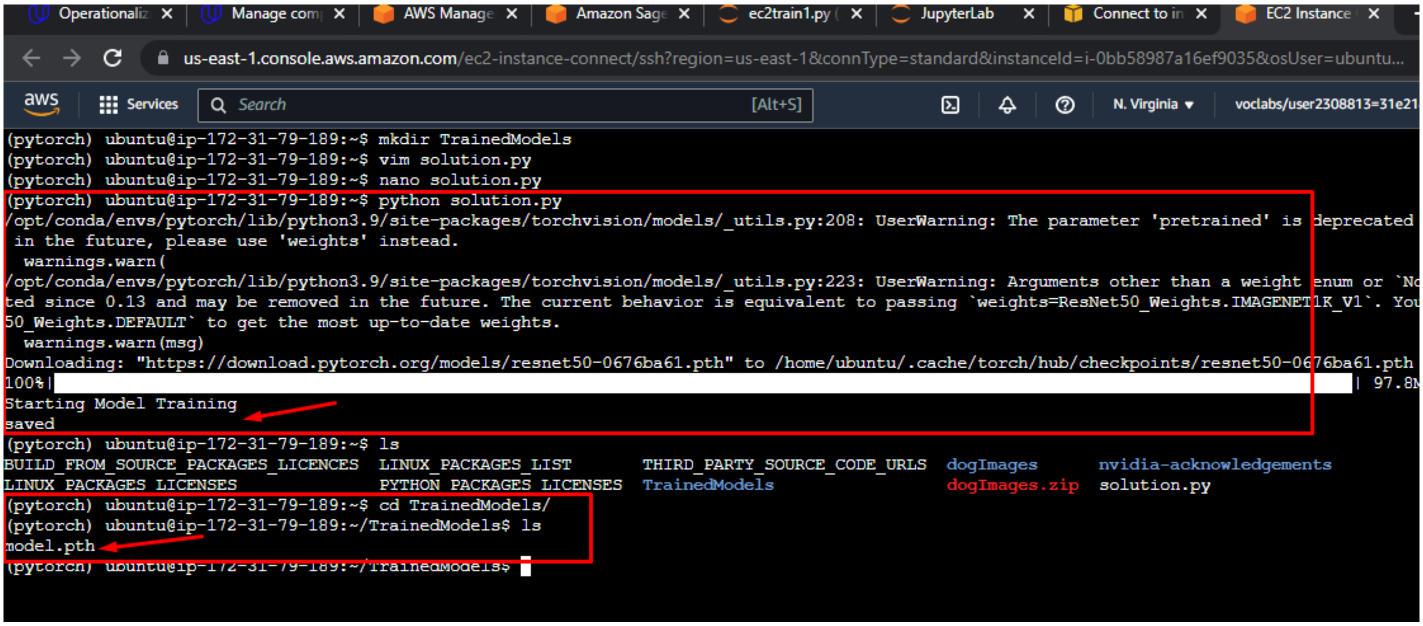


Fig5. EC2 Model training.

Difference Between EC2 Training Code and the Code used in Sagemaker

There are significant differences between the EC2 training and Sagemaker approaches. When executing the ec2train1.py file directly, the training occurs locally on the same computing machine. On the other hand, in the Sagemaker notebook train_and_deploy-solution.ipynb, a new training job instance is created, and all the training parameters are passed to it. The hyperparameters in the Sagemaker starter script, hpo.py, are explicitly passed to the script and recorded in the instance environment variables.

In Sagemaker, the output trained model.pth is saved to the training job compute instance in SM_MODEL_DIR=/opt/ml/model, compressed with the source code hpo.py, and automatically transferred to an output directory in the S3 bucket. In contrast, during EC2 training, the model is saved locally inside the ./TrainedModels directory, and the user is responsible for transferring the model to S3 using the aws s3 cp command.

Another difference is that Sagemaker can use the training script to spawn multiple instances and perform distributed training. In contrast, an EC2 instance is limited to running the script on a single instance.

Setting up a Lambda function

The lambda function lamdafunction.py that was provided to us has been updated by integrating it with the endpoint we previously created. To initiate the function, image URLs must be passed as a request dictionary with a content type of application/json. The request dictionary follows the format {'url': 'http://website.com/image-url.ext'}.

Subsequently, the function invokes the endpoint by sending the request dictionary in the request's body and setting the content type to application/json. The endpoint generates predictions, which are then returned to the lambda function. The lambda function, in turn, returns the predictions to the user in the response's body with a status code of 200.
and I got the following ineference vector:


```
[
  [
    -4.641569137573242, -2.4902334213256836, -3.0957112312316895,
    0.15276429057121277, -1.6126495599746704, -4.362102031707764,
    -0.4059160351753235, -1.9189156293869019, -2.861233711242676,
    -0.1609220951795578, 0.06969735771417618, -2.2069153785705566,
    -1.4752274751663208, 1.2776659727096558, -3.8395941257476807,
    -2.589606761932373, -5.573470592498779, -2.4858646392822266,
    1.0921307802200317, -2.9852418899536133, -0.20568329095840454,
    -2.924389123916626, -2.5274670124053955, -2.7377119064331055,
    -4.125853538513184, -1.6769689321517944, -2.4335925579071045,
    -2.139456033706665, -1.8561334609985352, -1.8811671733856201,
    -2.7960755825042725, -3.0512707233428955, -3.2729029655456543,
    -4.121776580810547, -4.067134380340576, -3.778663158416748,
    -1.411120891571045, -0.12106069922447205, -1.6981004476547241,
    -2.4159348011016846, -2.6264500617980957, -0.3506350815296173,
    -1.5564167499542236, -0.8254825472831726, -5.626651287078857,
    -0.981521487236023, -1.3561633825302124, -2.987349033355713,
    -1.6731294393539429, -2.5423879623413086, -5.724238872528076,
    -4.708513259887695, -3.597390651702881, -2.9752042293548584,
    -2.1273915767669678, -4.594807147979736, -2.596667766571045,
    -2.103217363357544, -0.2720622420310974, -6.684284210205078,
    -2.8606960773468018, -3.337257146835327, -3.5181264877319336,
    -2.7811193466186523, -3.9340620040893555, -2.3415918350219727,
    -4.728598594665527, -1.4737393856048584, -0.7698911428451538,
    -0.21613816916942596, -3.020470142364502, -4.157853603363037,
    -3.1226062774658203, -2.6615681648254395, -2.5284364223480225,
    -3.9090964794158936, -2.730088710784912, -4.386427879333496,
    -3.318309783935547, 0.43747007846832275, -5.103751182556152,
    -1.3342610597610474, -1.4758692979812622, -5.300479412078857,
    -4.564700603485107, -0.502282977104187, -5.332031726837158,
    -2.708049774169922, -1.5609245300292969, -4.028351783752441,
    -2.710907220840454, -4.13096284866333, -3.427432060241699,
    -1.419296383857727, -1.4140992164611816, -2.670917510986328,
    -2.362429618835449, -5.258486270904541, -3.4513394832611084,
    -5.059468746185303, -0.6168566942214966, -2.3428192138671875,
    -3.495523691177368, -2.038451910018921, -6.103508472442627,
    -2.5095627307891846, -1.134243130683899, 0.24228109419345856,
    -2.0794692039489746, -1.2858896255493164, -0.9553165435791016,
    -4.32726526260376, -3.874574661254883, -2.470676898956299,
    -1.2499037981033325, -6.718826770782471, -1.0573168992996216,
    -5.314194202423096, -0.15379099547863007, -1.3503224849700928,
    -4.789350509643555, -3.0358684062957764, -4.994699478149414,
    -3.0797832012176514, -4.172922611236572, -2.089721202850342,
    0.36488601565361023, -3.428293228149414, -4.512879371643066,
    -4.373014450073242, -1.7803796529769897, -5.944170951843262,
  ]
]
```

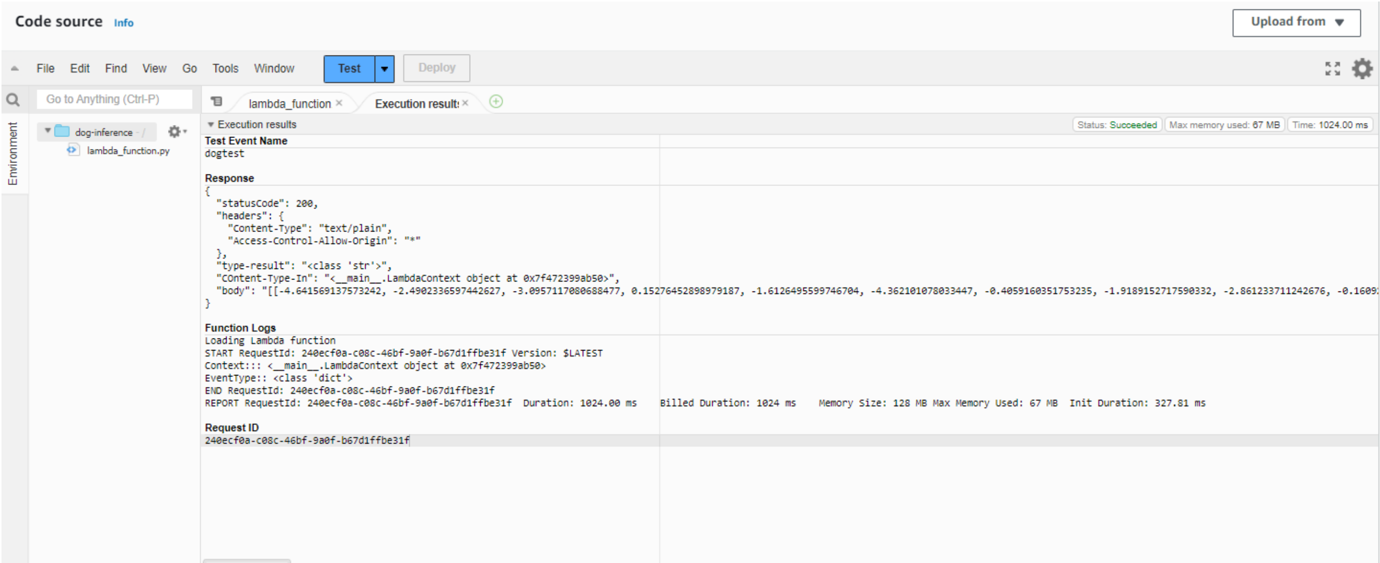


Fig 6. Lambda test output

After configuring our lambda function to invoke the endpoint, we attached the AmazonSageMakerFullAccess security policy that will allow us to access the endpoint.

The AmazonSageMakerFullAccess policy may grant more permissions than necessary for our lambda function, which only executes endpoints in Sagemaker. It might be a better practice to restrict its permissions to endpoints only. Additionally, it is crucial to be mindful of deleting unused lambdas and roles and assigning the least privileges necessary to resources in use to prevent potential vulnerabilities.

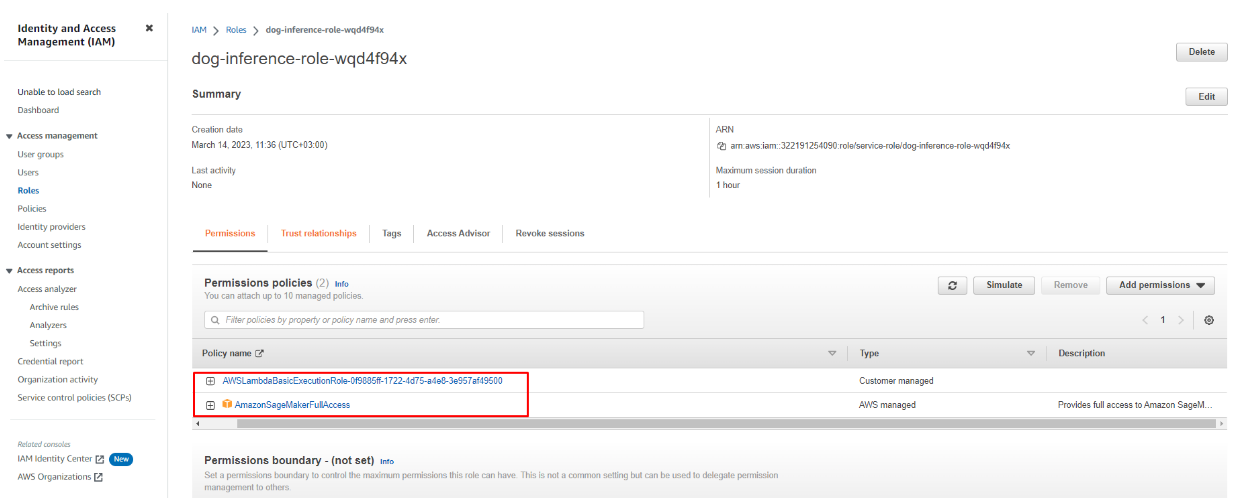


Fig 7. IAM Role

Concurrency & Autoscaling

Concurrency refers to the ability of Lambda functions to handle multiple requests simultaneously. We can use either reserved or provisioned concurrency for our function. Provisioned concurrency is more responsive but also leads to higher costs. We don't anticipate high volumes on these functions, so selecting very high concurrency is unnecessary. Therefore, I set the provisioned concurrency to 4, sufficient for our expected load.

Auto-scaling, on the other hand, refers to the ability of endpoints to handle multiple requests from lambda functions simultaneously. I decided to configure the endpoints to auto-scale up to a maximum of 4 instances, with a scale-in cool-down time of 30 seconds and a scale-out cool-down time of 300 seconds. These settings meet our project's needs and workload requirements.

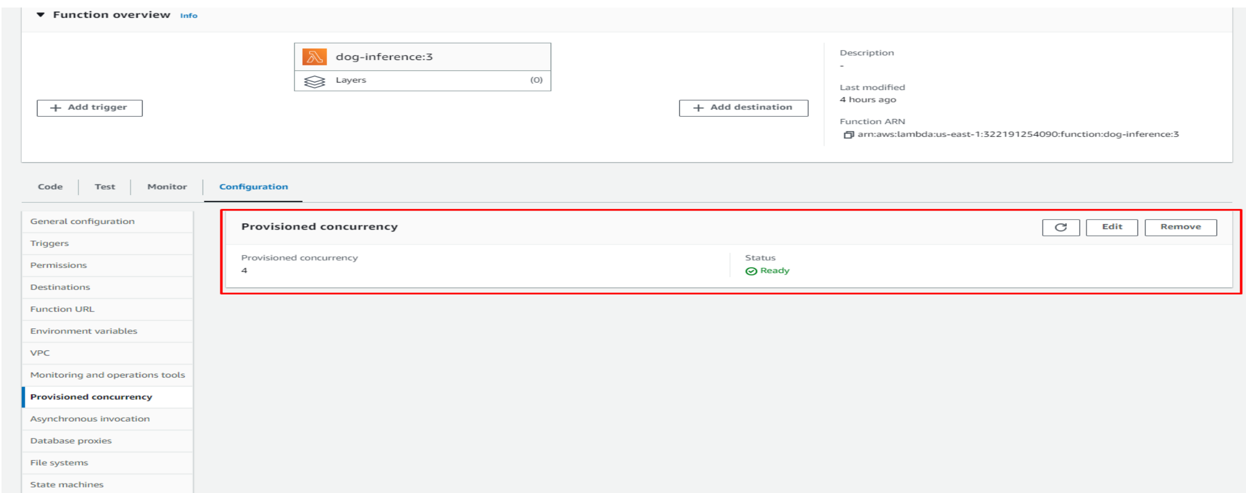
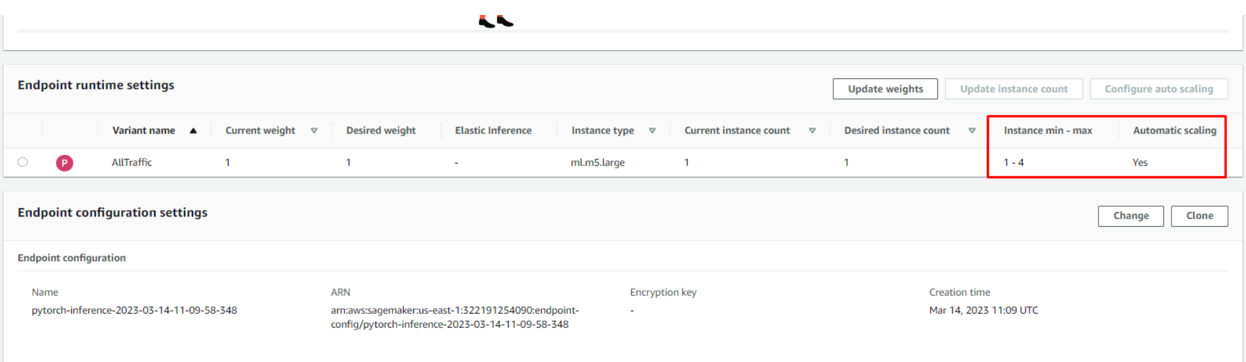


Fig 8. Provisioned Concurrency (AWS Lambda)



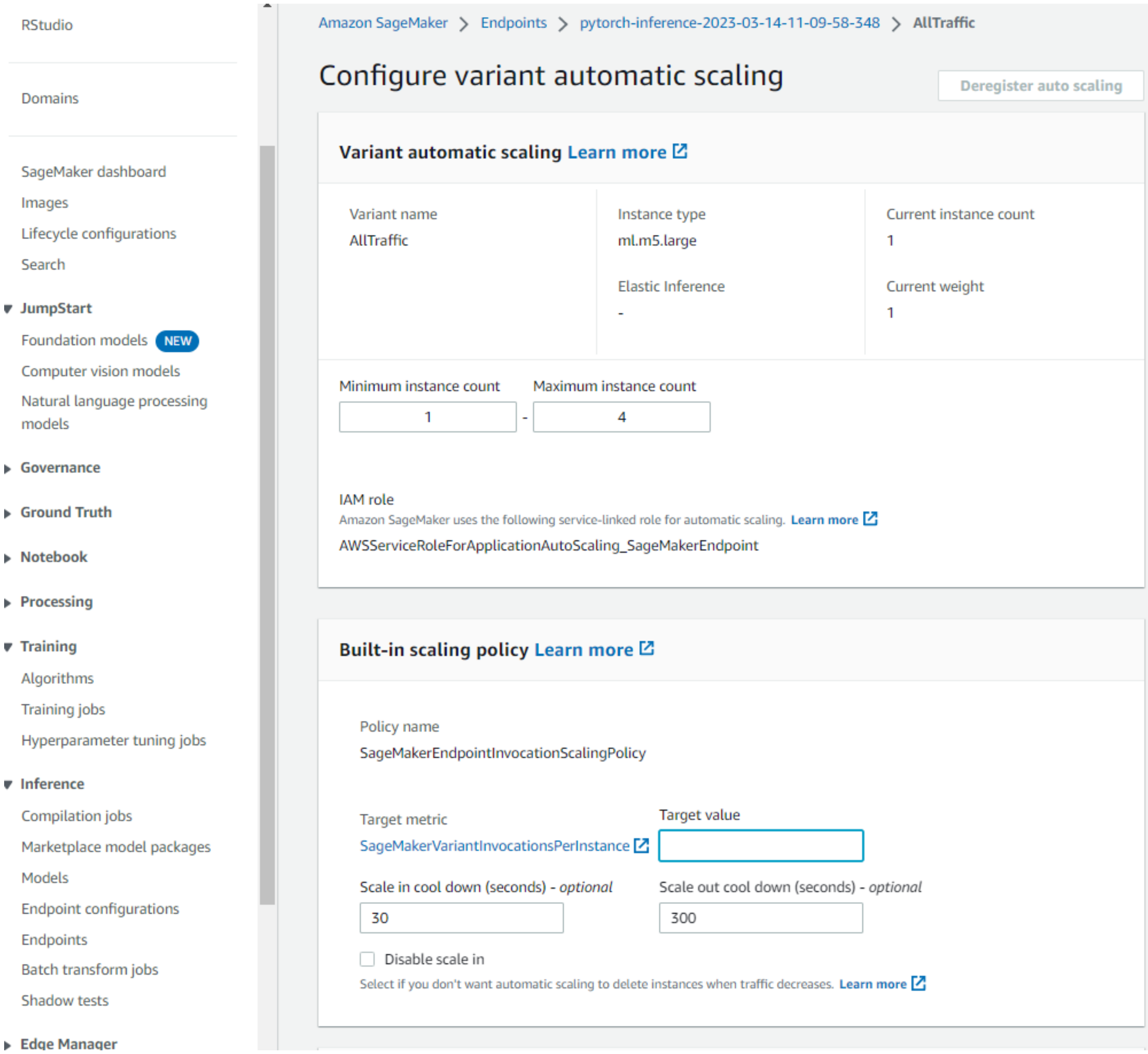


Fig 10. Autoscaling (Sagemaker Endpoint)