# Operationalizing Machine Learning on SageMaker by Neil Simon

## Initial Setup

I decided to use the smallest SageMaker instance `ml.t2.medium` (Figure: SageMaker Instance) available for my notebook as I will be keeping it open for hours as I run through the project, and I do not need a particularly capable instance from a CPU or RAM point of view. Note that I failed to take an appropriate screenshot when I started the project and re-did that portion at the end.

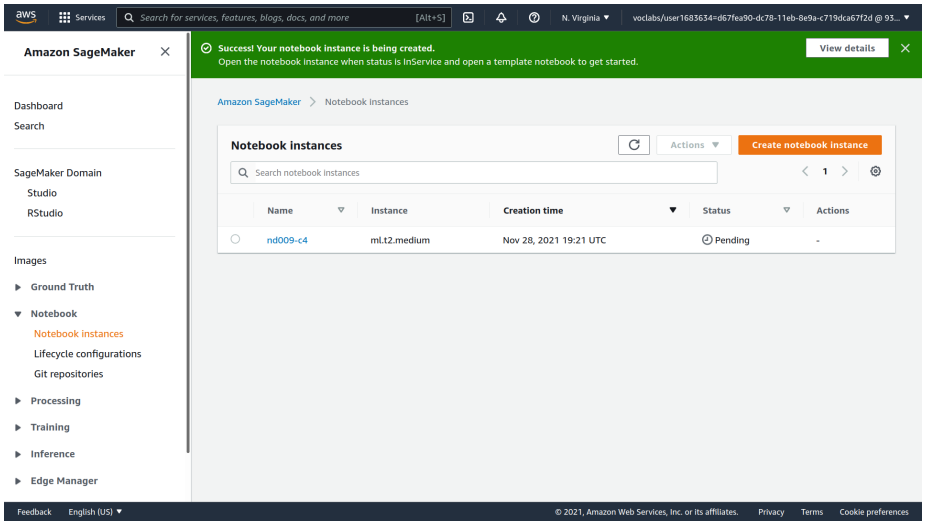


Figure 1: SageMaker Instance

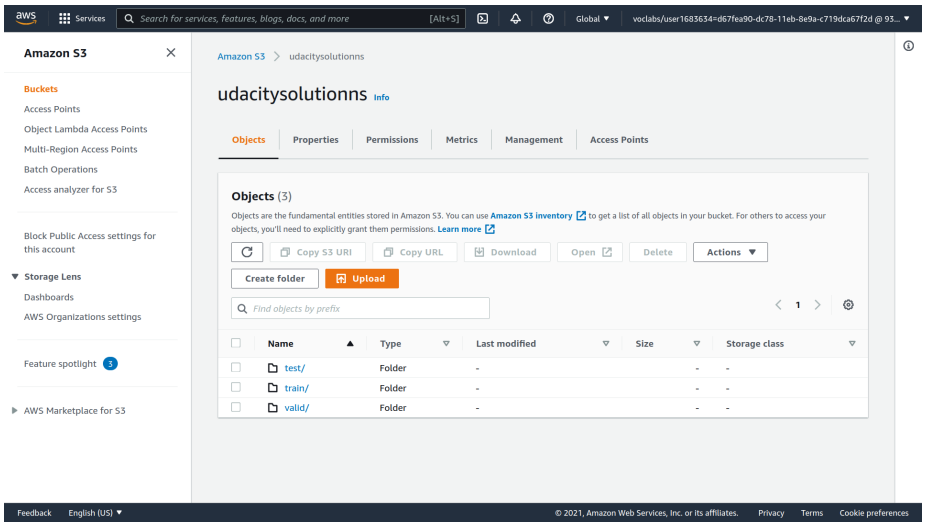


Figure 2: S3 Bucket Setup

Additionally, for both tuning and training I chose to use `ml.m5.2xlarge` for the greater processing power so that the tuning job and training jobs could be completed more quickly and to avoid memory issues I had experienced when working with this dataset in a previous project. I also increased the `max_jobs` to 12 for tuning, `max_parallel_jobs` to 3 and enabled `early_stopping_type` as "Auto" to speed up tuning and make sure that better hyperparameters were picked.

To modify the `hpo.py` file to perform distributed training I derived changes based on, including changes to the parameters used to start the training (setting the number of instances to 4 and using the gloo backend): https://github.com/aws/amazon-sagemaker-examples/blob/master/sagemaker-python-sdk/pytorch_mnist/mnist.py

I found that I had to upgrade pandas as I got errors otherwise.

- Single instance trained endpoint: pytorch-inference-2021-11-28-16-08-01-790
- Multi instance trained endpoint: pytorch-inference-2021-11-28-17-26-20-083

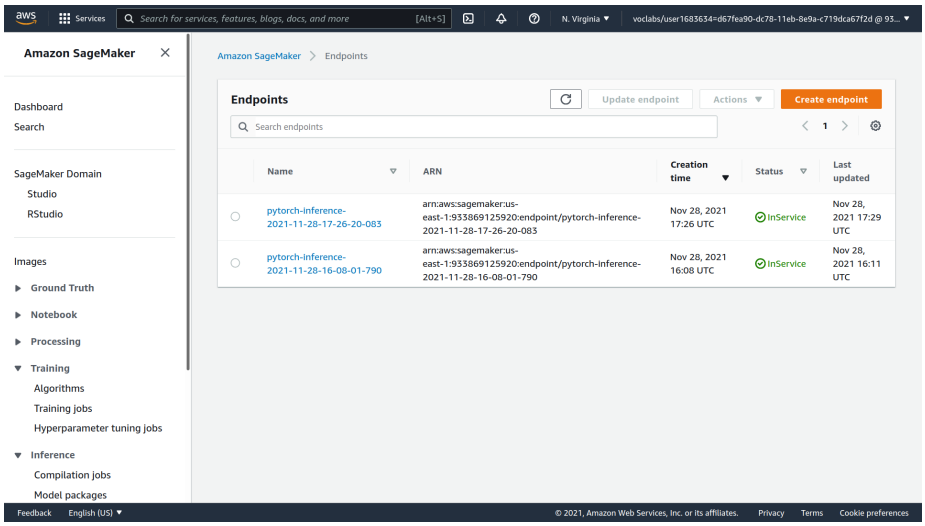


Figure 3: Trained Endpoints

## EC2 Training

I used the `Deep Learning AMI (Amazon Linux 2) Version 54.0` ami. and `t3.xlarge` instance. I believed this to be a good compromise between performance and cost. As while running EC2 training there are 2 end points running it makes sense to pick a more powerful instance type than would be necessary. Similarly, since we do not know how long it will take to complete the setup and debug of this EC2 training section, it makes sense to go with a modest sized instance and not incur the outsized cost while we are running setup, debug, etc. As you will note from my screen shot, I have restricted access to my EC2 instance to only my IP address to reduce the chance that it would be inappropriately accessed.

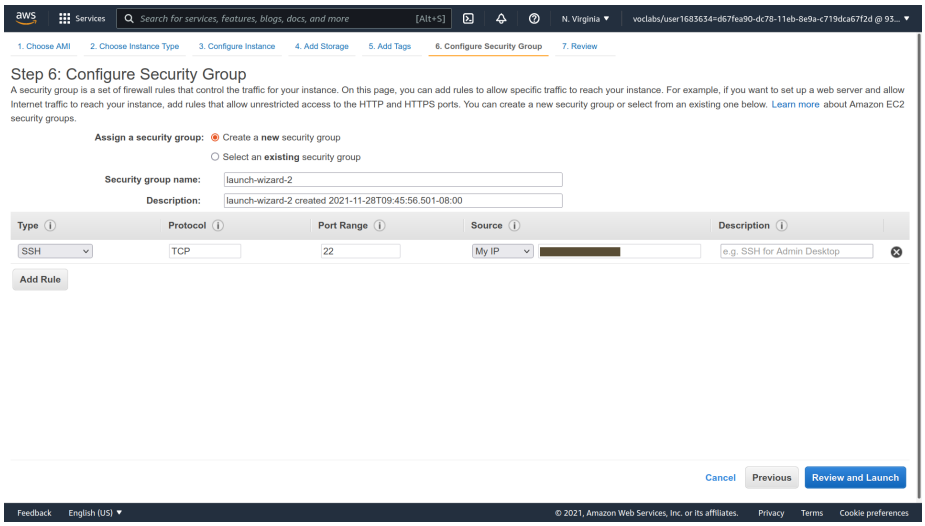


Figure 4: EC2 Secure Access

### Differences between ec2train1.py (EC2 script) and train_and_deploy-solution.ipynb+hpo.py (SageMaker scripts)

The most obvious difference is that `ec2train1.py` does not have the main function, nor the code for handling argument parsing and optional running of the main. Similarly, `ec2train1.py` does not have support for multi-instance learning as I implemented in my updated `hpo.py`. Perhaps the most consequential difference is that the `ec2train1.py` script trains using the test data, meaning that it uses a much smaller dataset and uses the same data for training as testing. I have changed:

```
train_data =
    torchvision.datasets.ImageFolder(root=test_data_path,
    transform=train_transform)
```

to

```
train_data =
    torchvision.datasets.ImageFolder(root=train_data_path,
    transform=train_transform)
```

to get a more complete training on the EC2 instance. This obviously took longer to run as you can see in the two spikes in CPU use on the `EC2 Training Resouce Use` image.

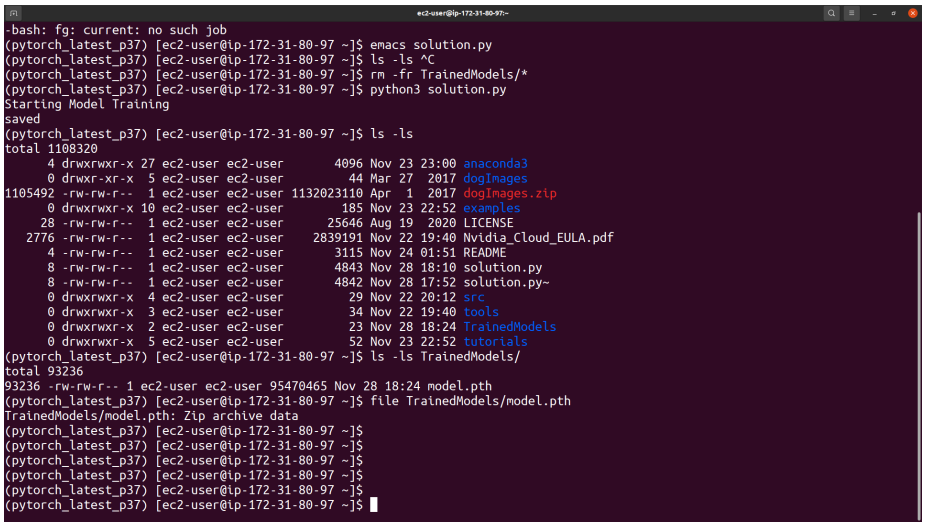


Figure 5: EC2 Training

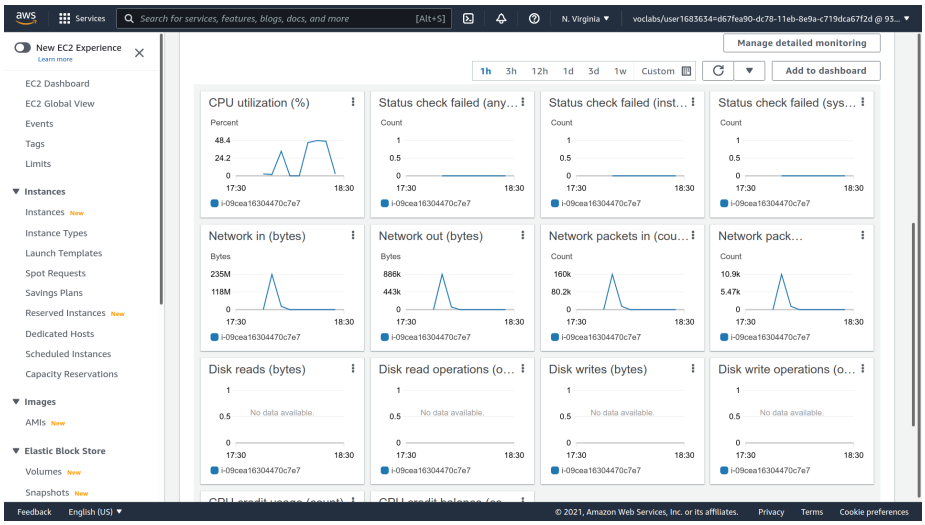


Figure 6: EC2 Training Resource Use

## Lambda functions

I created 2 lambda functions as I had 2 endpoints as above. The first is `testEndpoint1Trainer` to test the single instance trained endpoint, and the

second is `testEndpointMTrainer` to test the multiple instance trained endpoint. The handler code seems to require access to the SageMaker runtime, and is somewhat oddly layed out, with code running outside of the lambda_handler function. There is clearly some legacy code commented out. Otherwise, the code is written to invoke the endpoint with the argument provided. It seems like this would have been a good time to perform verification on the input to reduce the load on the endpoint and possibly prevent DOS attacks.

Before adding full access to SageMaker to the role attached to the lambda functions I got the following output from a test:

```
{
  "errorMessage": "An error occurred (AccessDeniedException)
      when calling the InvokeEndpoint operation: User:
      arn:aws:sts::933869125920:assumed-role/testEndpointMTrainer-role-pr3dpwbk/testEndpoint
      is not authorized to perform: sagemaker:InvokeEndpoint on
      resource:
      arn:aws:sagemaker:us-east-1:933869125920:endpoint/pytorch-inference-2021-11-28-17-26-2
      because no identity-based policy allows the
      sagemaker:InvokeEndpoint action",
  "errorType": "ClientError",
  "stackTrace": [
    "  File \"/var/task/lambda_function.py\", line 24, in
        lambda_handler\n
        response=runtime.invoke_endpoint(EndpointName=endpoint_Name,\n",
    "  File \"/var/runtime/botocore/client.py\", line 386, in
        _api_call\n    return
        self._make_api_call(operation_name, kwargs)\n",
    "  File \"/var/runtime/botocore/client.py\", line 705, in
        _make_api_call\n    raise error_class(parsed_response,
        operation_name)\n"
  ]
}
```

After adding full access to SageMaker to the role attached to the lambda functions I got the following output from a test (note that there are 133 numbers in the body, not 33 as in the instructions):

```
{
  "statusCode": 200,
  "headers": {
    "Content-Type": "text/plain",
    "Access-Control-Allow-Origin": "*"
  },
  "type-result": "<class 'str'>",
  "COntent-Type-In": "<__main__.LambdaContext object at
      0x7f7f23782e80>",
```

```
 "body": "[[-9.833578109741211, -2.3149001598358154,
    -4.496285915374756, -2.459660768508911,
    -3.2292635440826416, -7.208012104034424,
    -2.019676923751831, -4.80747652053833, -7.580871105194092,
    0.1604955494403839, -0.9565751552581787,
    -3.9131810665130615, -3.6020150184631348,
    0.05120508745312691, -4.187273025512695,
    -2.8582608699798584, -5.9626336097717285,
    -4.536521911621094, -5.346053123474121,
    0.8844369649887085, -4.171061038970947,
    -1.9665920734405518, -8.535749435424805,
    -7.713301181793213, -6.517572402954102,
    -11.525176048278809, -1.6054246425628662,
    -2.6769134998321533, -6.043071269989014,
    -4.014992713928223, -1.5675303936004639,
    -5.836017608642578, -9.598838806152344,
    -3.9479377269744873, -7.269865989685059,
    -8.273256301879883, -7.2418389320373535,
    -6.530744552612305, -2.6606929302215576,
    -4.69330358505249, -4.120052337646484, -5.510358810424805,
    -0.34800487756729126, -4.418677806854248,
    -1.6251585483551025, -8.261250495910645,
    -3.4547064304351807, -3.625526189804077,
    -2.2067673206329346, -5.123303413391113,
    -4.32491397857666, -9.562714576721191, -10.10139274597168,
    -2.621943235397339, -7.87992525100708,
    -2.6131465435028076, -2.555243968963623,
    -8.795278549194336, -2.1555397510528564,
    -5.046584606170654, -8.843852996826172,
    -4.459414958953857, -7.6189351081848145,
    -8.326446533203125, -4.4117021560668945,
    -8.239740371704102, -2.5183329582214355,
    -5.56660270690918, -5.840574741363525,
    -0.9575398564338684, -0.9576200246810913,
    -5.738442897796631, -7.9645609855651855,
    -7.754147529602051, -7.372220039367676,
    -2.7810957431793213, -7.490314483642578,
    -3.23928165435791, -5.329830169677734,
    -4.7164130210876465, -1.013601541519165,
    -8.067651748657227, -0.8396211266517639,
    -2.3040759563446045, -6.821334362030029,
    -4.568439483642578, -1.9165948629379272,
    -6.500880241394043, -2.1579692363739014,
    -2.284560441970825, -8.265277862548828, -5.20717191696167,
    -8.007201194763184, -6.459601402282715,
    -5.552811622619629, -4.417727470397949,
```

```
    -4.309690952301025, -4.329131126403809,
    -5.873167514801025, -5.903502941131592,
    -9.697685241699219, -2.39001202583313,
    -3.9008214473724365, -7.240103244781494,
    -7.310409069061279, -8.4434175491333, -5.835858345031738,
    -1.4929323196411133, -5.384642601013184,
    -2.533282995223999, -3.6708028316497803,
    -2.3309476375579834, -11.313907623291016,
    -7.884079456329346, -7.088305473327637,
    -2.394468307495117, -4.874009132385254,
    -4.4106550216674805, -6.073166370391846,
    -1.498389482498169, -3.4654202461242676,
    -4.046548366546631, -6.391988754272461,
    -4.826951026916504, -9.816082954406738,
    -7.140218734741211, -4.48201847076416, -3.733743190765381,
    -4.512071132659912, -7.819015026092529,
    -6.860950946807861, -1.9007254838943481,
    -5.910882949829102]]"
}
```

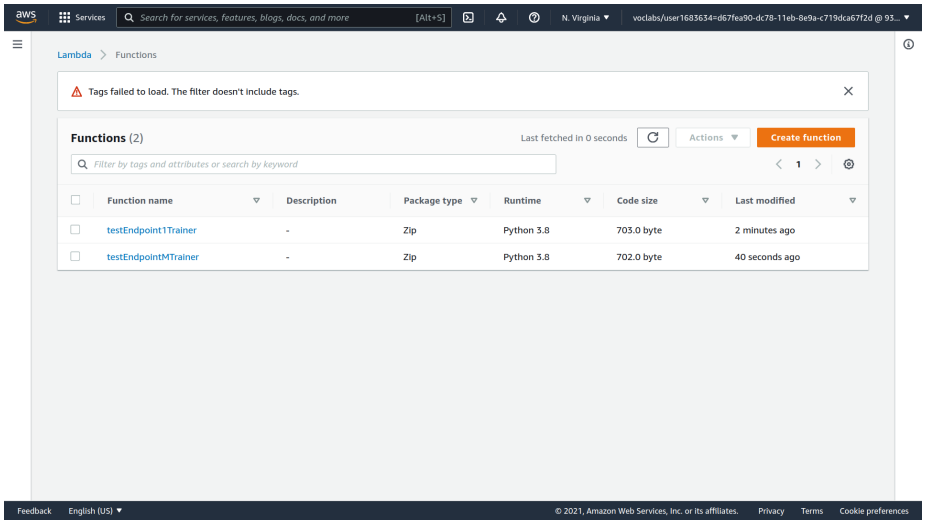


Figure 7: Lambda Functions

### Permissions

I have concerns with the permissions we have given these lambda functions as it clearly does not meet the principle of least privilege. Ideally we would give these lambda functions permission to query the endpoint which they are supposed to

be able to query, and only that. I need to perform further research to determine if there is any way to limit such. Additionally, these lambda functions could be used as a layer of protection from DOS types of attacks against the endpoints.

Additionally, I am concerned that the root user for this account does not use MFA.

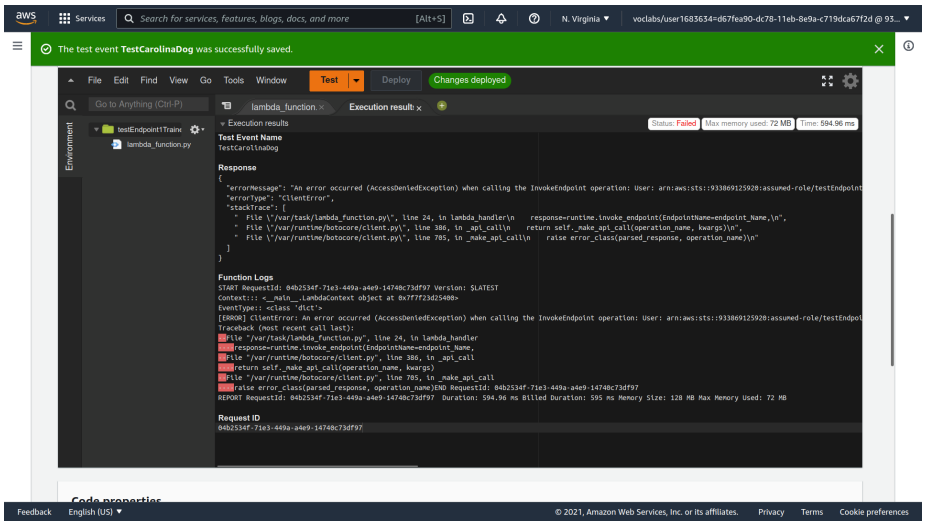


Figure 8: Lambda Functions Failed Test

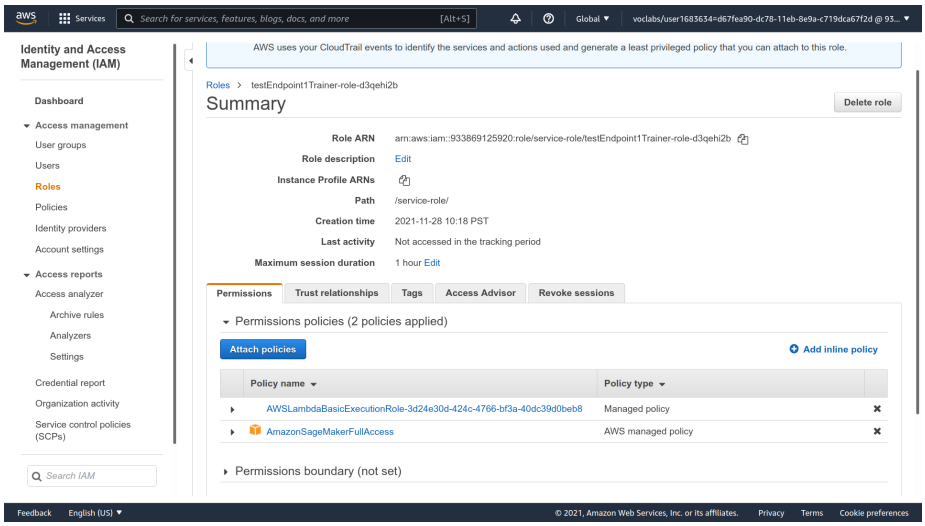


Figure 9: Lambda Functions Role Update

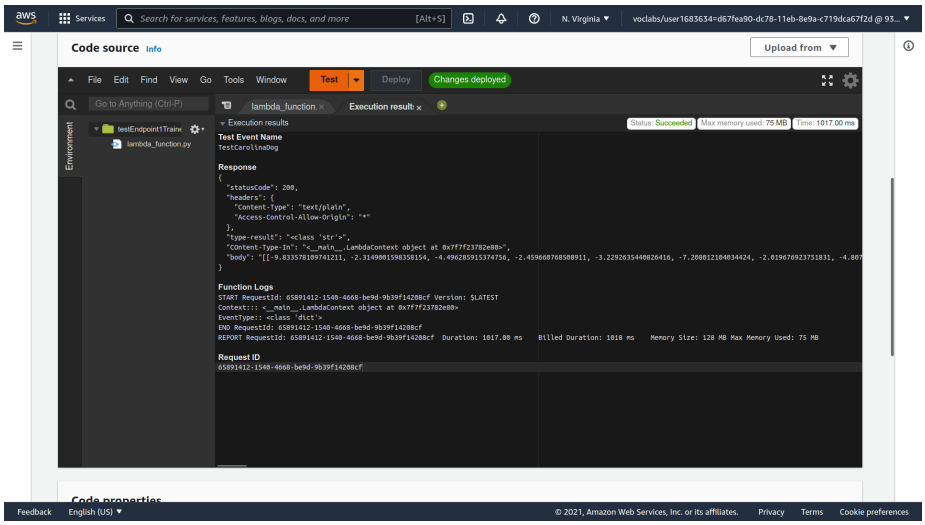


Figure 10: Lambda Functions Passed Test

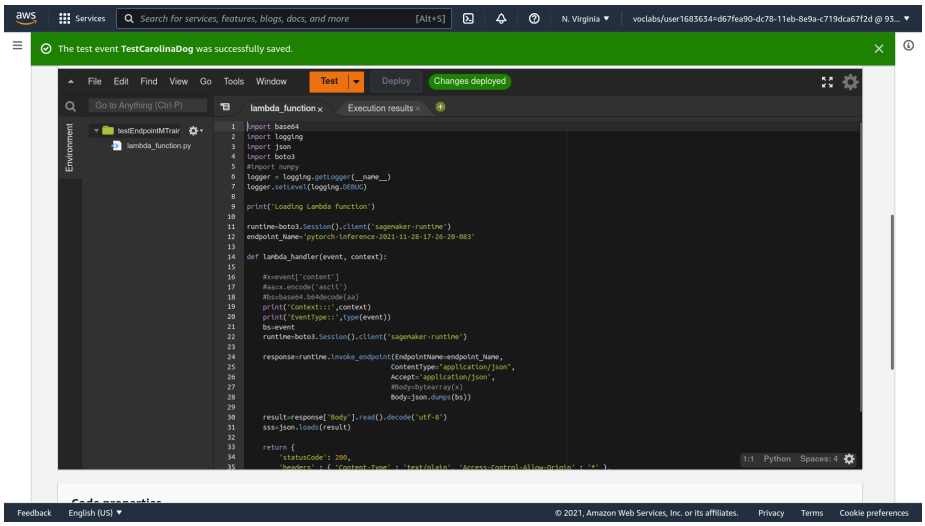


Figure 11: Lambda Function

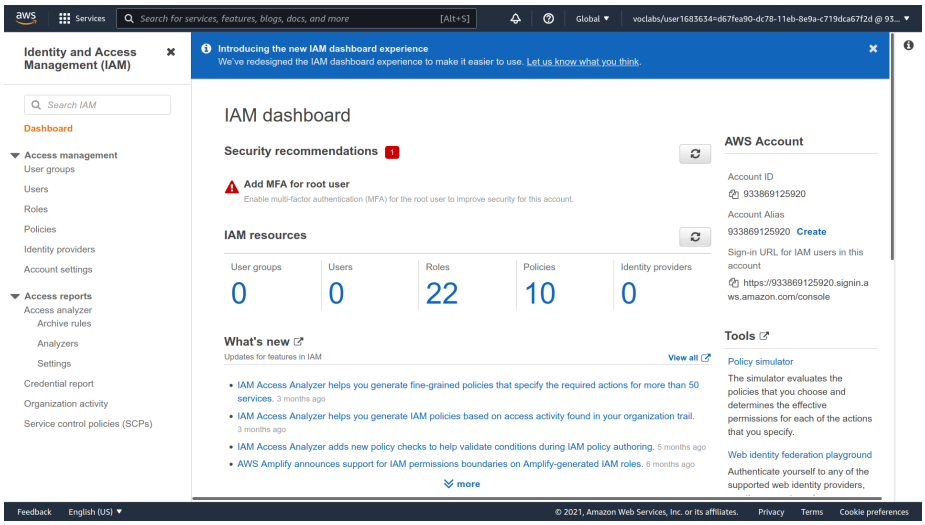


Figure 12: IAM Dashboard

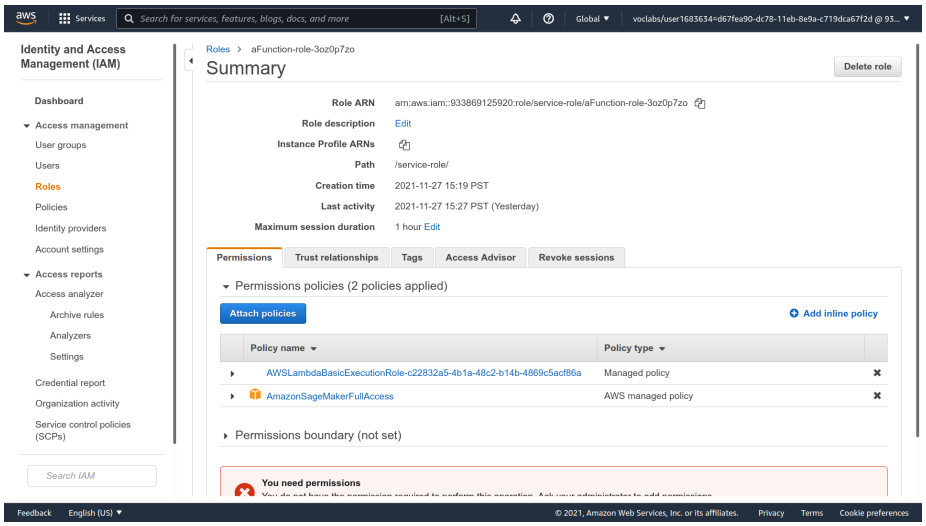


Figure 13: IAM Dashboard Roles

## Concurrency and auto-scaling

I chose to use reserved concurrency as it comes without cost and meets our rather small needs at this time. Additionally, it is unlikely that at any time we will be handling more than a few requests per endpoint instance, since such would indicate that the latter is overloaded, so it makes sense to only have this value be a multiple of the number of endpoint instances, therefore I chose 100, allowing for 20 requests per endpoint instance. I chose to allow the endpoint scale out and in between 1 and 5 instances. While it is unlikely that this will ever be a problem, since it takes about 0.25s per prediction, such should allow for around 20 queries per second, which would with proper request throttling, allow for use by a very large number of users.

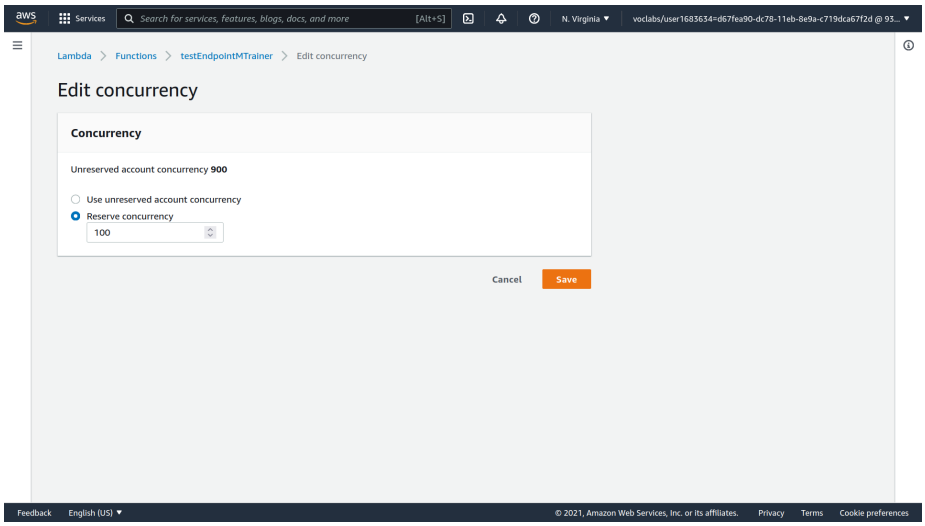


Figure 14: Configuring Concurrency

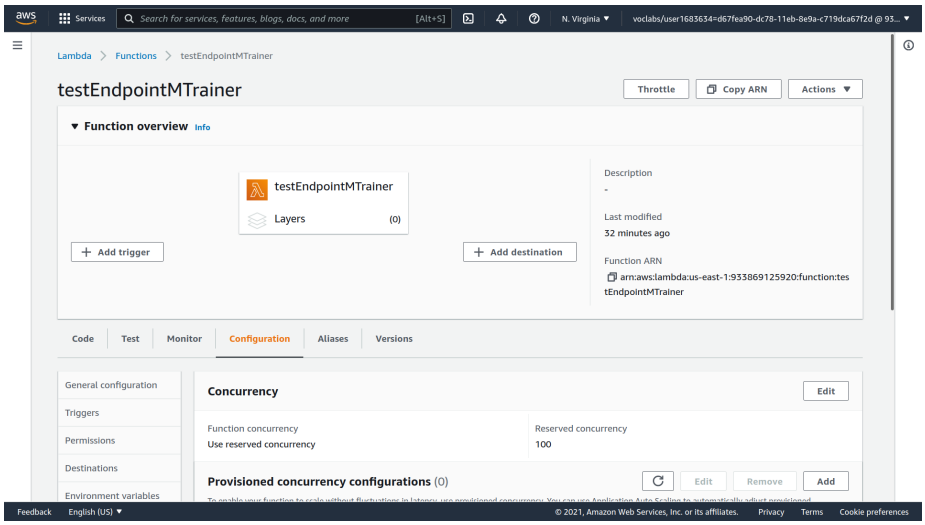


Figure 15: Configured Concurrency

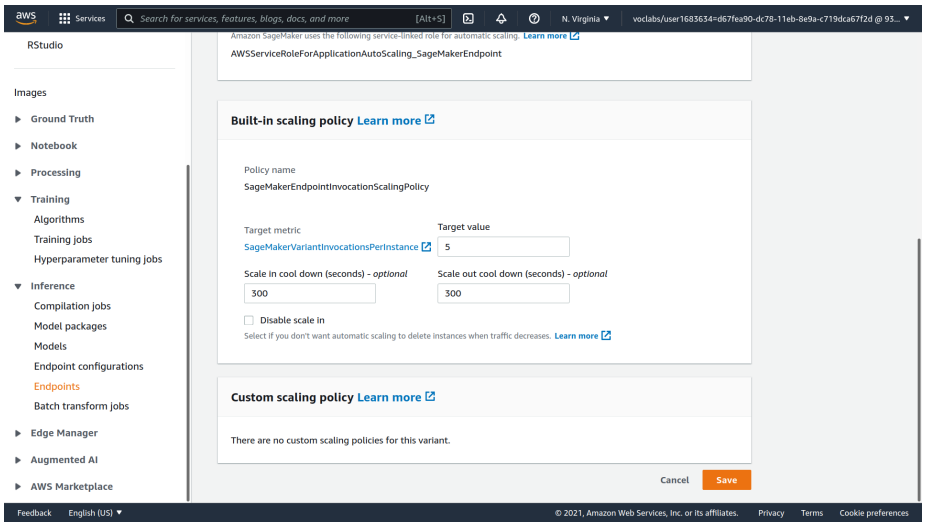


Figure 16: Configuring Concurrency

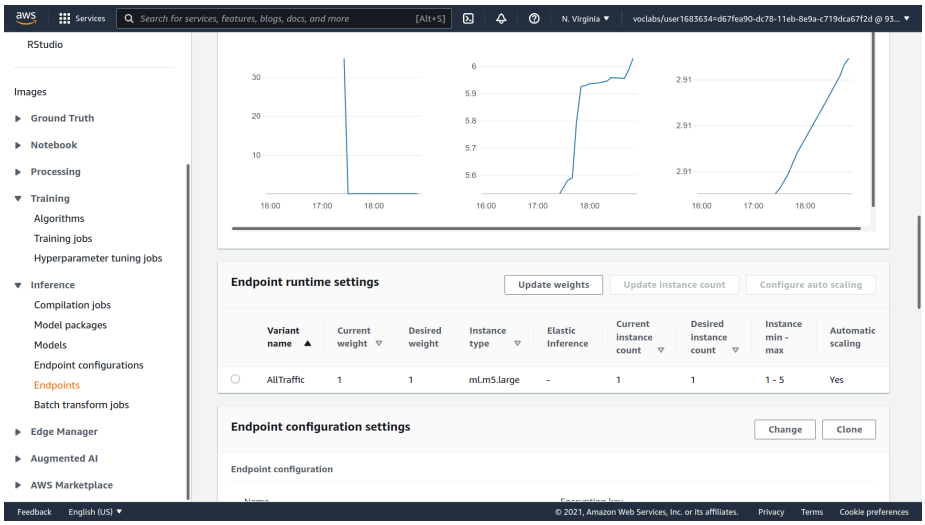


Figure 17: Configured Concurrency