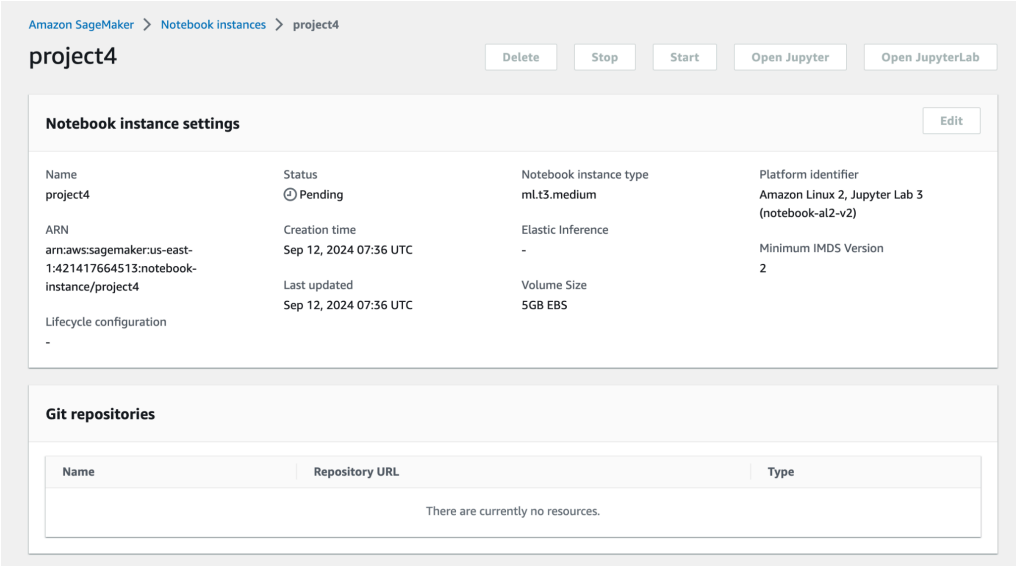
Operationalizing ML on Sagemaker

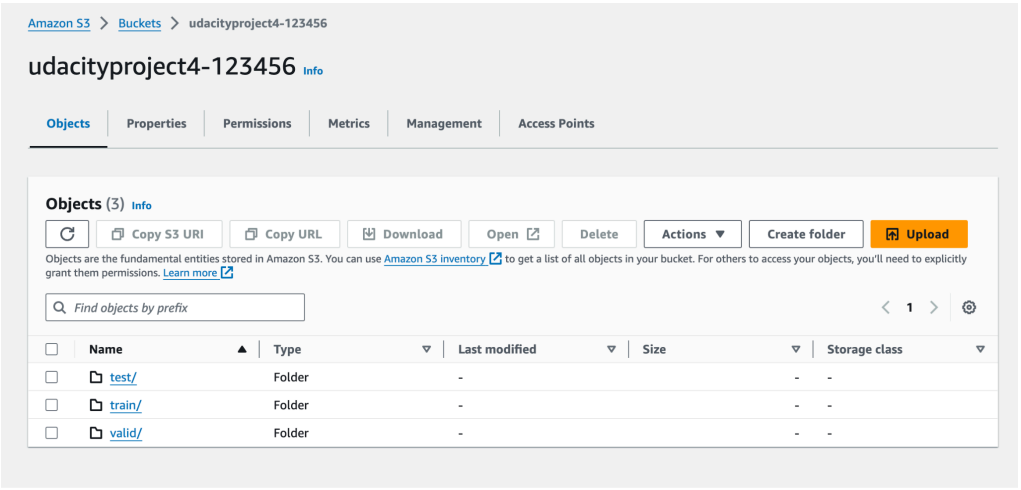
Training and Deployment on Sagemaker

Initial Setup

I chose the `ml.t3.medium` instance because it offers a good balance between cost and computing power for small to medium-sized training tasks. It provides sufficient CPU capacity, memory, and network performance to efficiently handle the training and deployment of a computer vision model without incurring excessive costs. The instance also launches quickly, making it an ideal choice for tasks that don't require intensive GPU resources but still need reasonable computational speed.

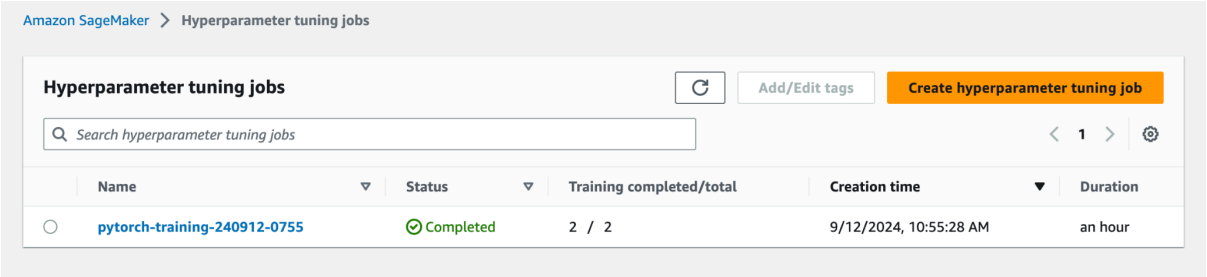


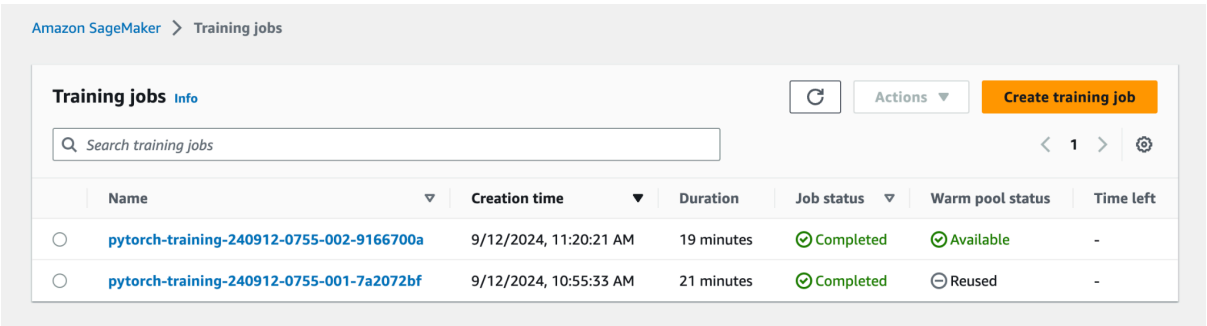
S3 Setup



Training and Deployment

Hyperparameter Tuning





Describe the tuning results

[7]:
```
exp = HyperparameterTuningJobAnalytics(
  hyperparameter_tuning_job_name='pytorch-training-240912-0755')

jobs = exp.dataframe()

jobs.sort_values('FinalObjectiveValue', ascending=0)
```

[7]:

	batch_size	learning_rate	TrainingJobName	TrainingJobStatus	FinalObjectiveValue	TrainingStartTime	TrainingEndTime	TrainingElapsedTimeSe
0	"256"	0.013478	pytorch-training-240912-0755-002-9166700a	Completed	726.0	2024-09-12 08:20:26+00:00	2024-09-12 08:39:49+00:00	
1	"64"	0.047914	pytorch-training-240912-0755-001-7a2072bf	Completed	290.0	2024-09-12 07:56:13+00:00	2024-09-12 08:16:32+00:00	

Prepare to perform Training on Best Estimator

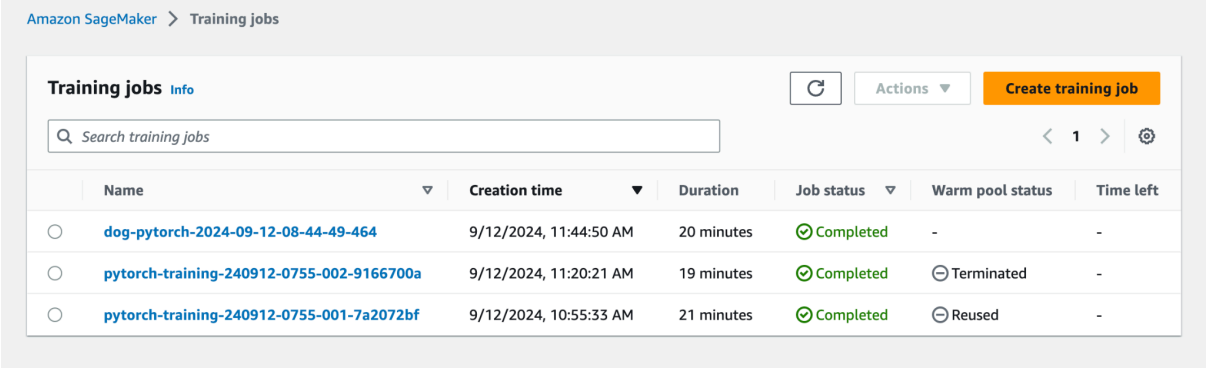
[8]:
```
best_estimator=tuner.best_estimator()
```

```
2024-09-12 08:20:24 Starting - Preparing the instances for training
2024-09-12 08:20:24 Downloading - Downloading the training image
2024-09-12 08:20:24 Training - Training image download completed. Training in progress.
2024-09-12 08:20:24 Uploading - Uploading generated training model
2024-09-12 08:20:24 Completed - Resource reused by training job: pytorch-training-240912-0755-002-9166700a
```

[9]:
```
best_estimator.hyperparameters()
```

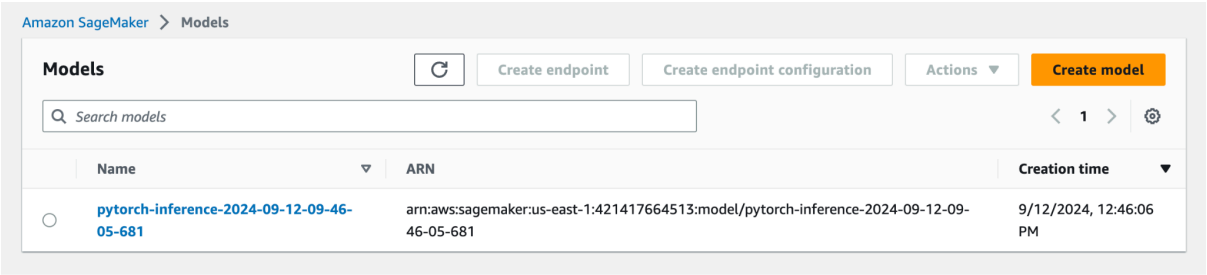
[9]:
```
{'_tuning_objective_metric': '"Test Loss"',
 'batch_size': '"64"',
 'learning_rate': '0.047914138854644156',
 'sagemaker_container_log_level': '20',
 'sagemaker_estimator_class_name': '"PyTorch"',
 'sagemaker_estimator_module': '"sagemaker.pytorch.estimator"',
 'sagemaker_job_name': '"pytorch_dog_hpo-2024-09-12-07-55-27-690"',
 'sagemaker_program': '"hpo.py"',
 'sagemaker_region': '"us-east-1"',
 'sagemaker_submit_directory': '"s3://sagemaker-us-east-1-421417664513/pytorch_dog_hpo-2024-09-12-07-55-27-690/source/sourcedir.
tar.gz"'}
```

Training

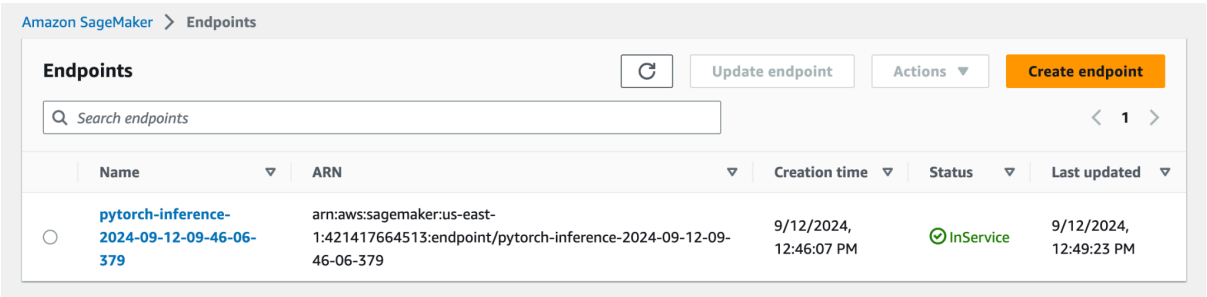


Deployed Endpoint

First the model is created:



Then the endpoint is deployed:



Multi Instance Training

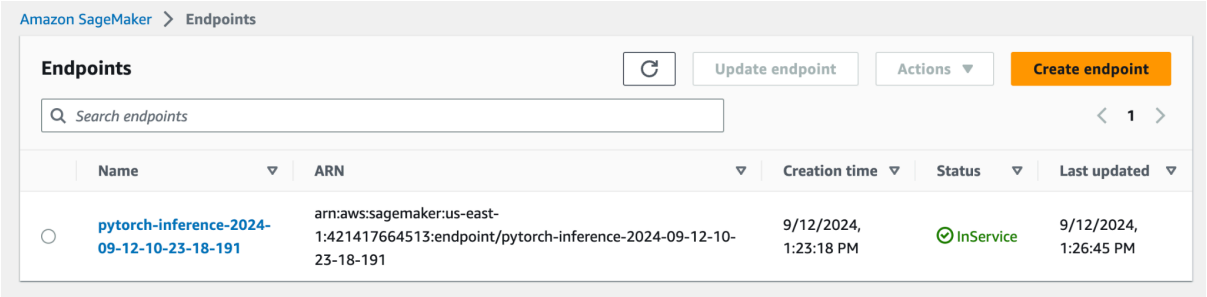
Creating an Estimator - Multi-Instance Training,

```
###in this cell, create and fit an estimator using multi-instance training
#adjust this cell to accomplish multi-instance training
estimator = PyTorch(
    entry_point='hpo.py',
    base_job_name='dog-pytorch',
    role=role,
    instance_count=2,  # Change this to 2 or more for multi-instance training
    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,
)
```

```
estimator.fit({"training": "s3://udacityproject4-123456/"}, wait=False)
```

Changed the instance_count = 2 to do multi-instance training.

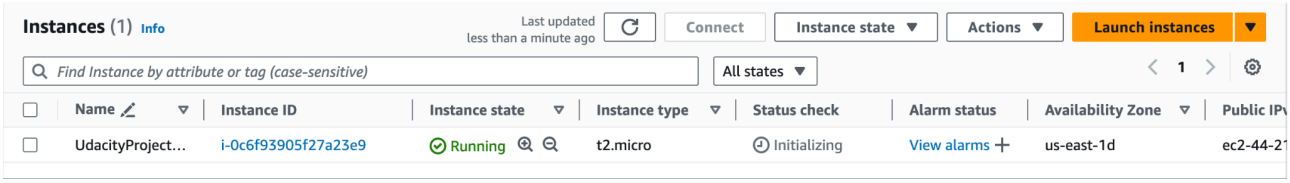
New Endpoint



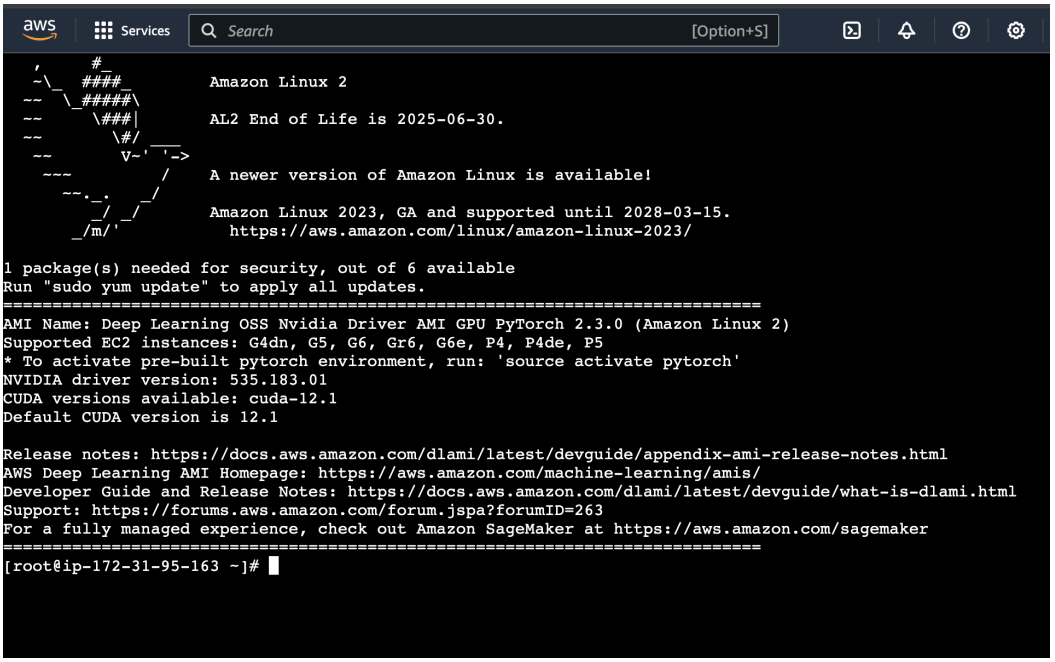
EC2 Training

EC2 Setup

I selected “Deep Learning OSS Nvidia Driver AMI GPU PyTorch 2.3 (Amazon Linux 2)” on the AMI list, so I can proceed with the AMI compatible with the Pytorch environment, which we are using. `t2.micro` is selected as the instance type since we are training a small model and it is eligible for free tier.

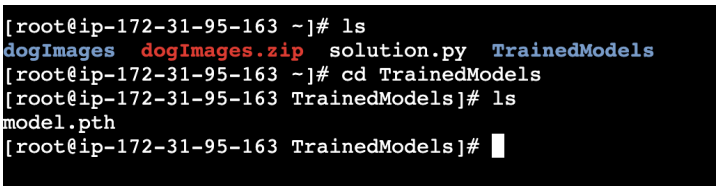


Connected to my newly created instance.



Training and Saving on EC2

Created the file and pasted the code, and the training is done.

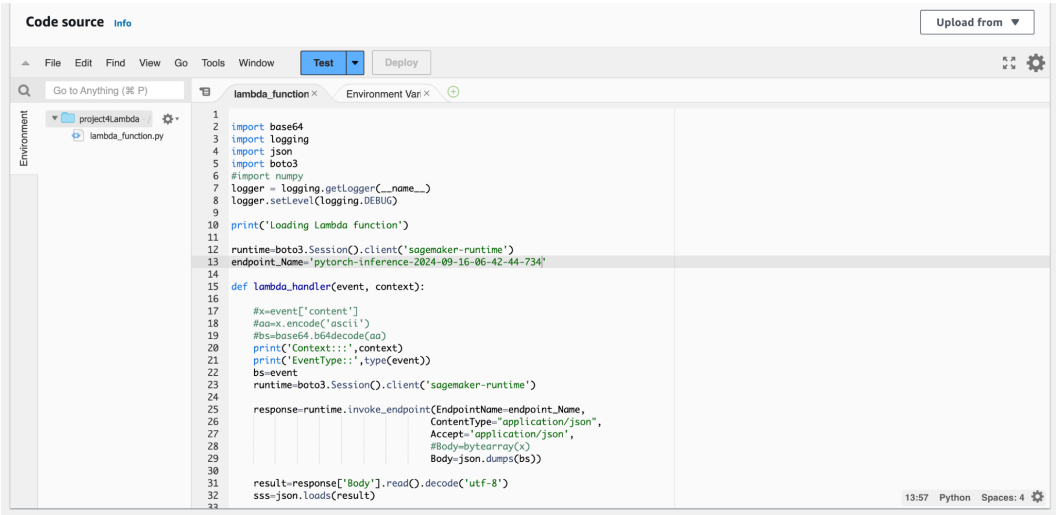


Write about the EC2 Code

Firstly, we used a Python file on EC2 instead of a Jupyter Notebook, which needs a different environment to run. We also did not use the capabilities of Sagemaker such as hyperparameter tuning and instead used hard coded values. Other than these, the code is very similar to hpo.py, which we used as an entry point for our estimator when we were using the notebook.

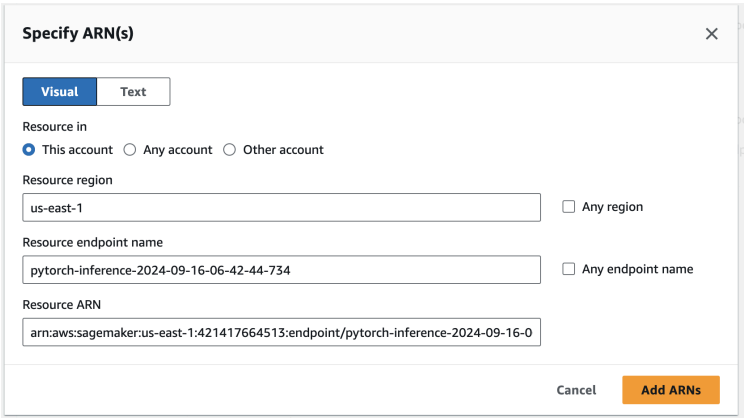
Lambda Function Setup

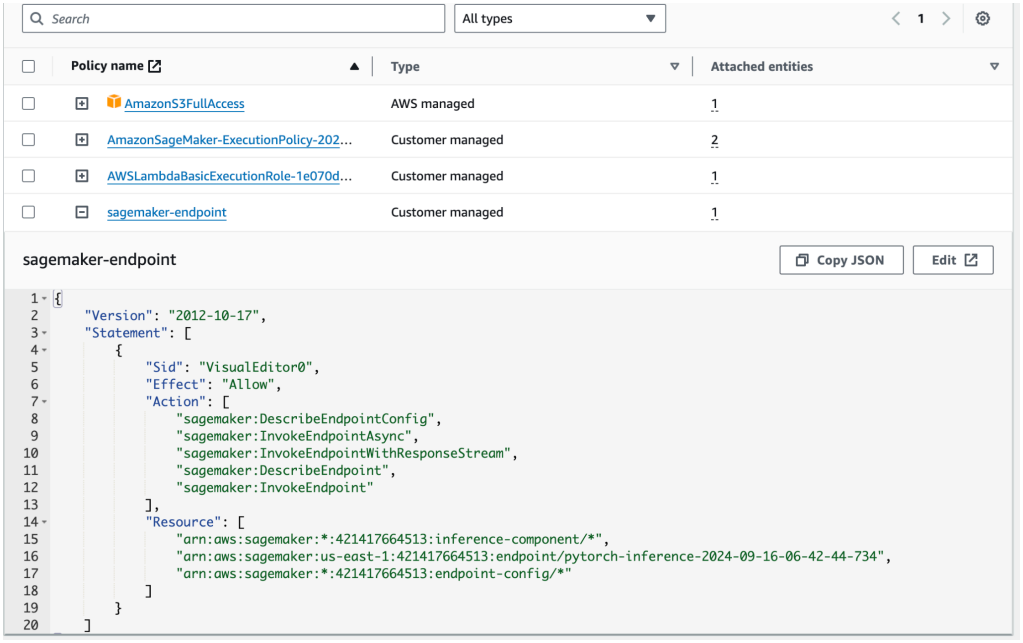
The `lambda_function.py` takes in an event and context, decodes the event using base64, and sends the resulting data as a request to a SageMaker endpoint. The function then handles the response from the endpoint, formatting it into a JSON object that includes HTTP status codes and headers. Additionally, it uses the Python logging module to record debug messages throughout the process.



Lambda Function Security

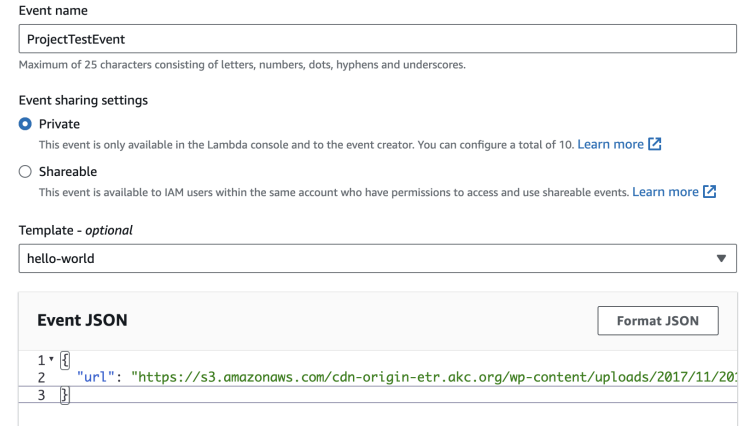
We only added the policies we needed, Sagemaker execution, S3 access and Lambda execution as our permissions to ensure security and also to ensure that our user will not access other functions that we do not need at the moment. We need to be able to invoke the endpoint, so we need to attach the rule to ensure that. I created a rule to invoke the endpoint and attached this policy as well. Otherwise our test function does not work:

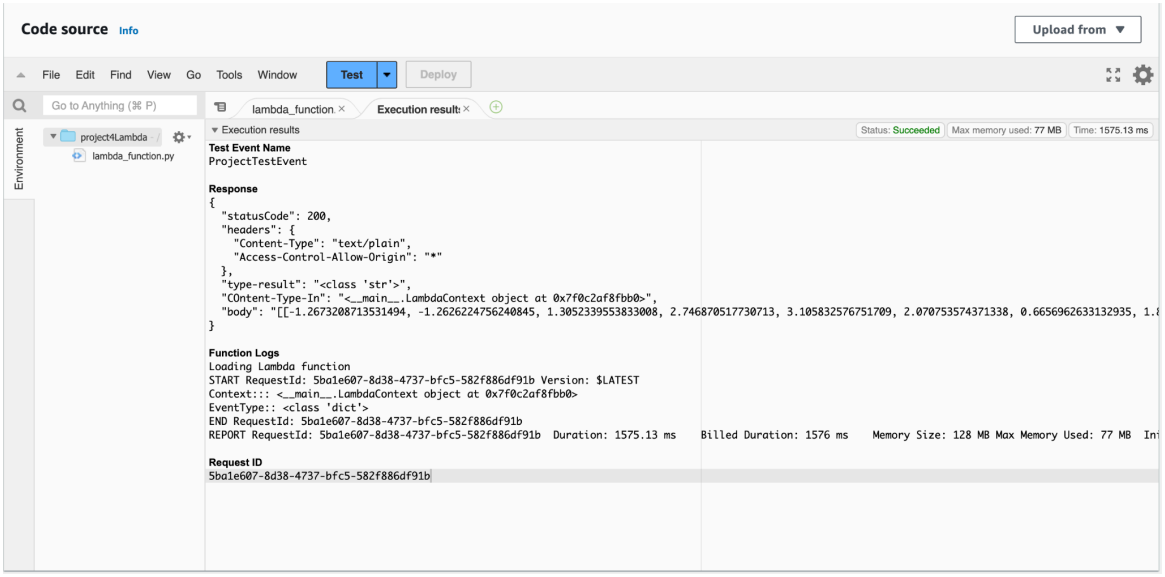




Lambda Function Testing

Used the json provided in the project page to test the lambda function:



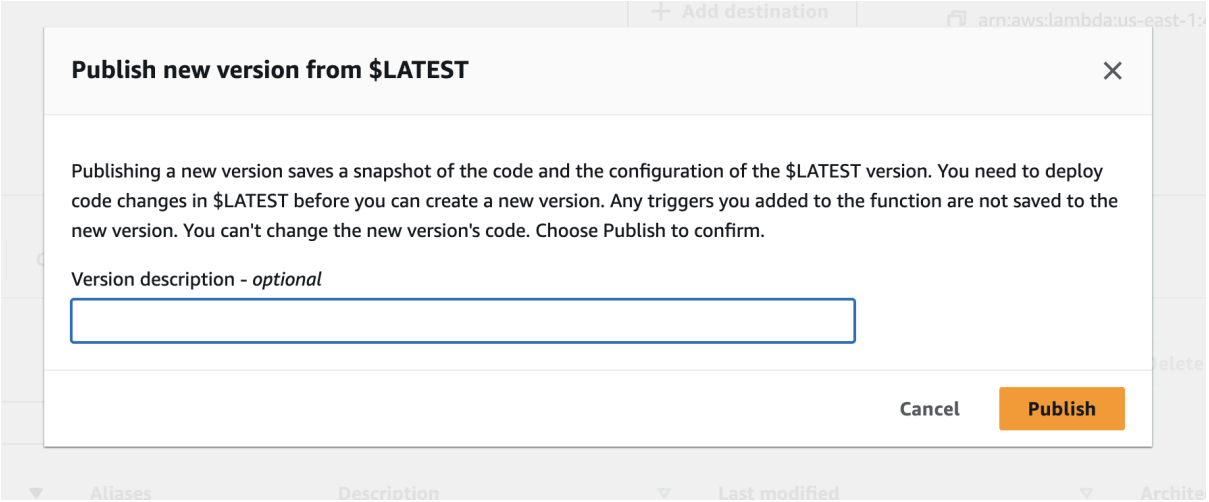


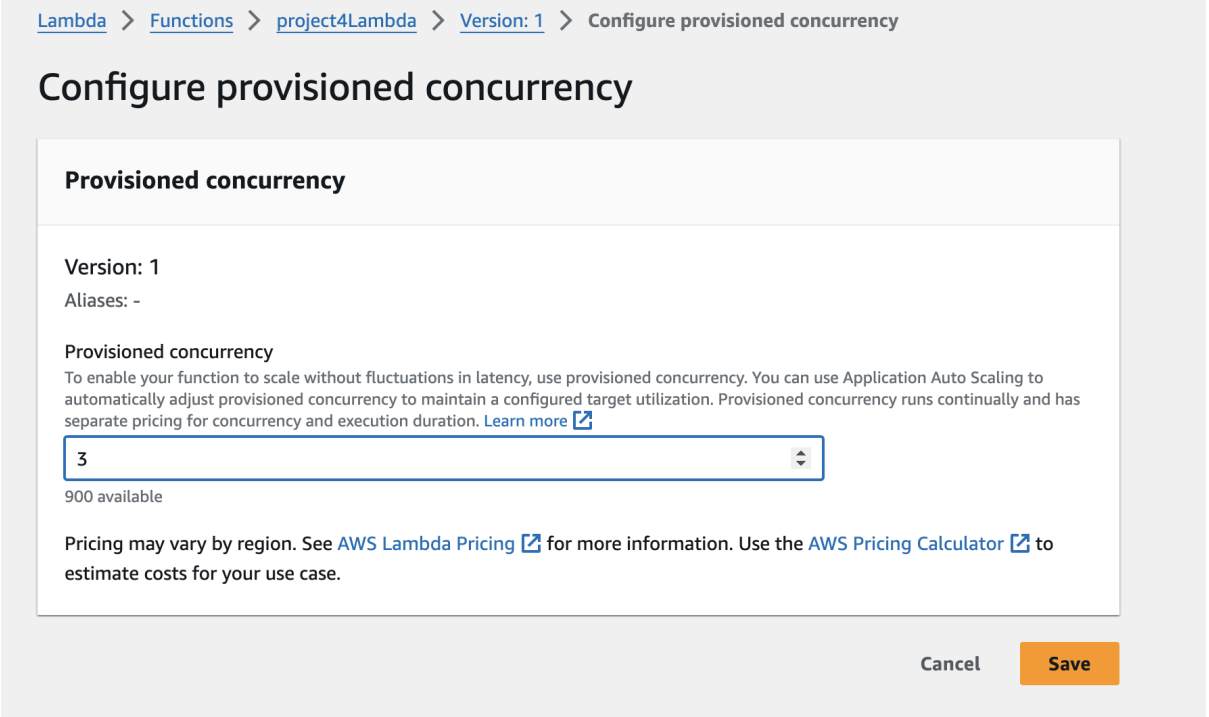
The account is safe since we only added certain policies, we delete the resources (the endpoint) after usage and keep monitoring the cost and other access.

Concurrency and Auto-Scaling

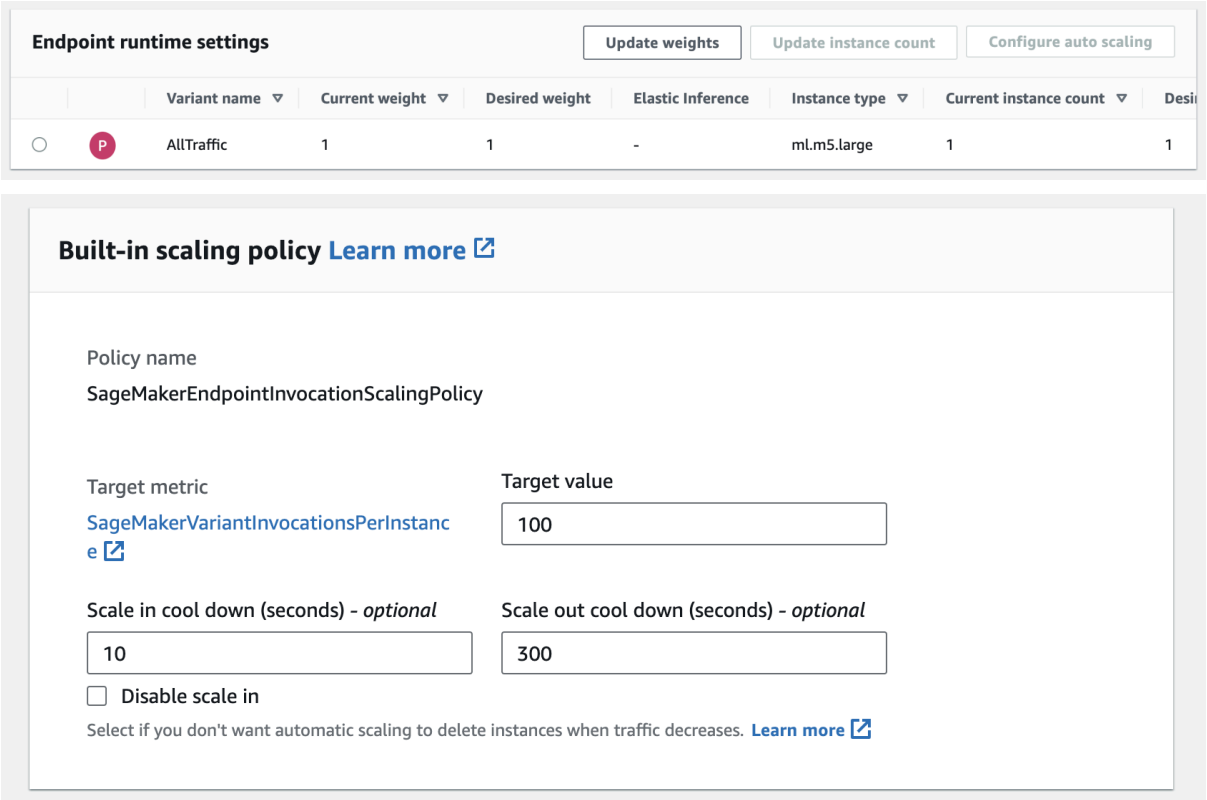
Concurrency

I allocated three instances for the concurrency of my Lambda function to further reduce expenses.





Auto Scaling



I set up auto scaling for my endpoint to activate when the number of invocations hits 100 per second, with a scale-in period of 10 seconds and a scale-out period of 300 seconds. This configuration was aimed at optimizing costs while managing high traffic.