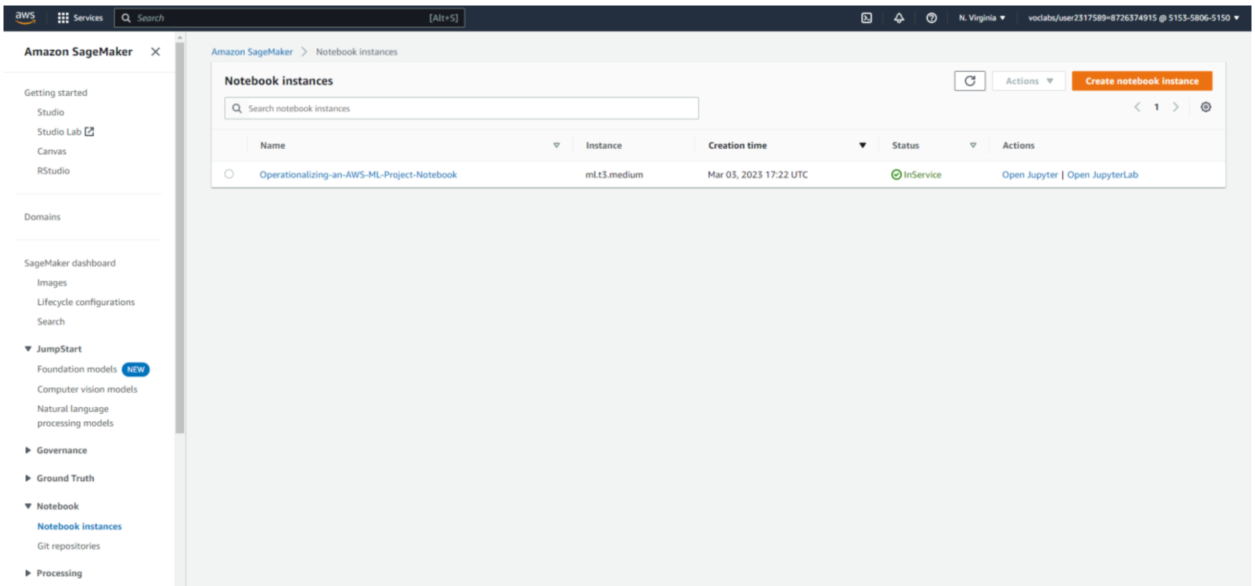
Initial setup, training and deployment

1- Initial Setup

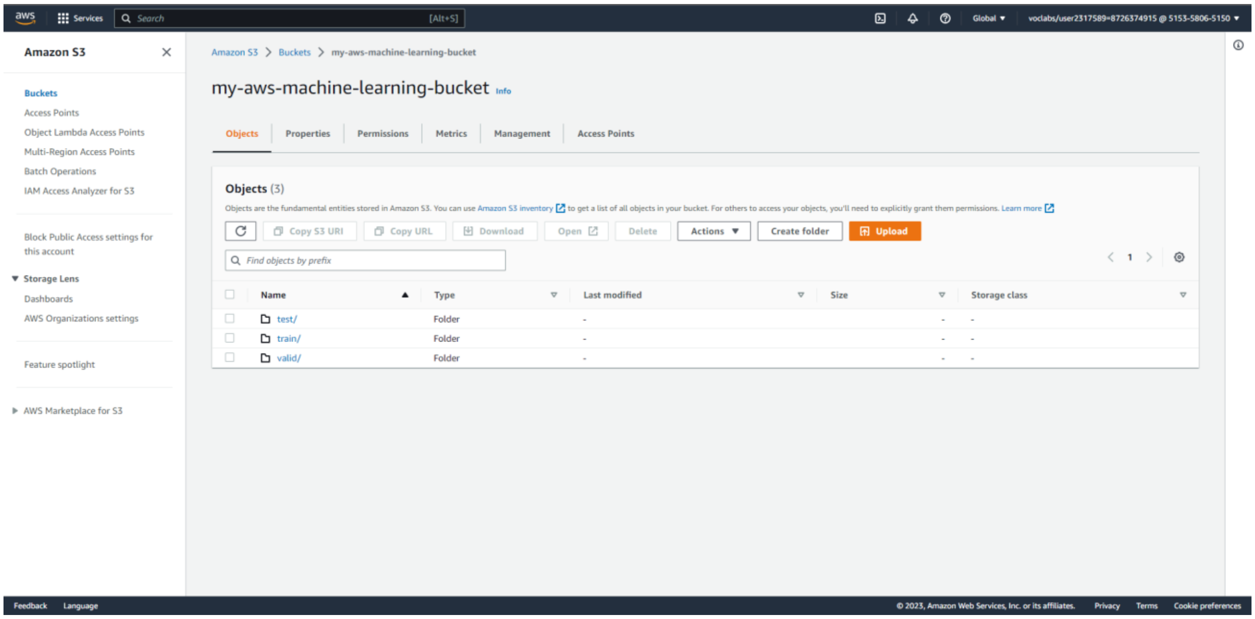
I chose the instance type "ml.t2. medium" for the Notebook instance because:

- I don't need a very powerful CPU in terms of computation and high RAM to run the notebook cells.
- In order to avoid high costs, we should select a laptop with a low cost per hour and reasonably good CPU and RAM, because we need to keep this laptop running for a long time while we work on the project.
- Looking at the type of instance and its price, "ml.t2. medium" has the lowest possible cost for our work.



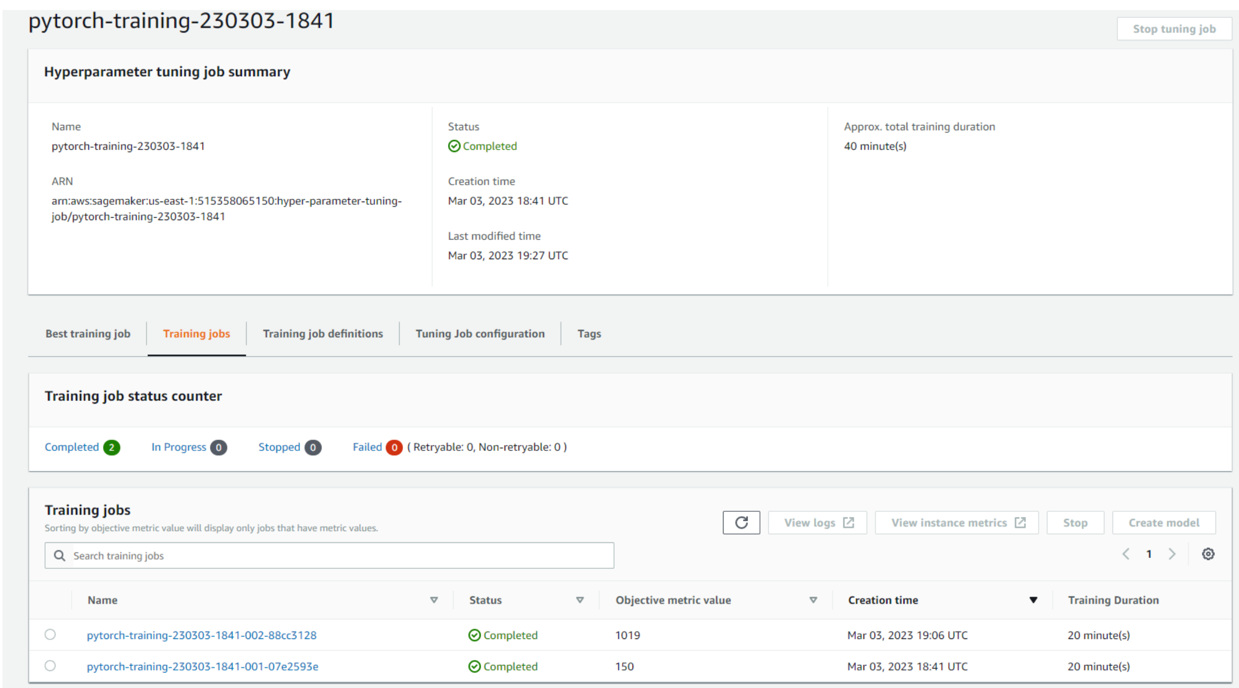
2- Download data to an S3 bucket

Here is the Cloud bucket where our dataset resides so all our work will be saved and our model will train from.



3- Training and Deployment

Here we launched a hyperparameter tuning job to try to look for the best parameters that our model can train with, and we are using an "ml. g4dn.xlarge" for faster training.



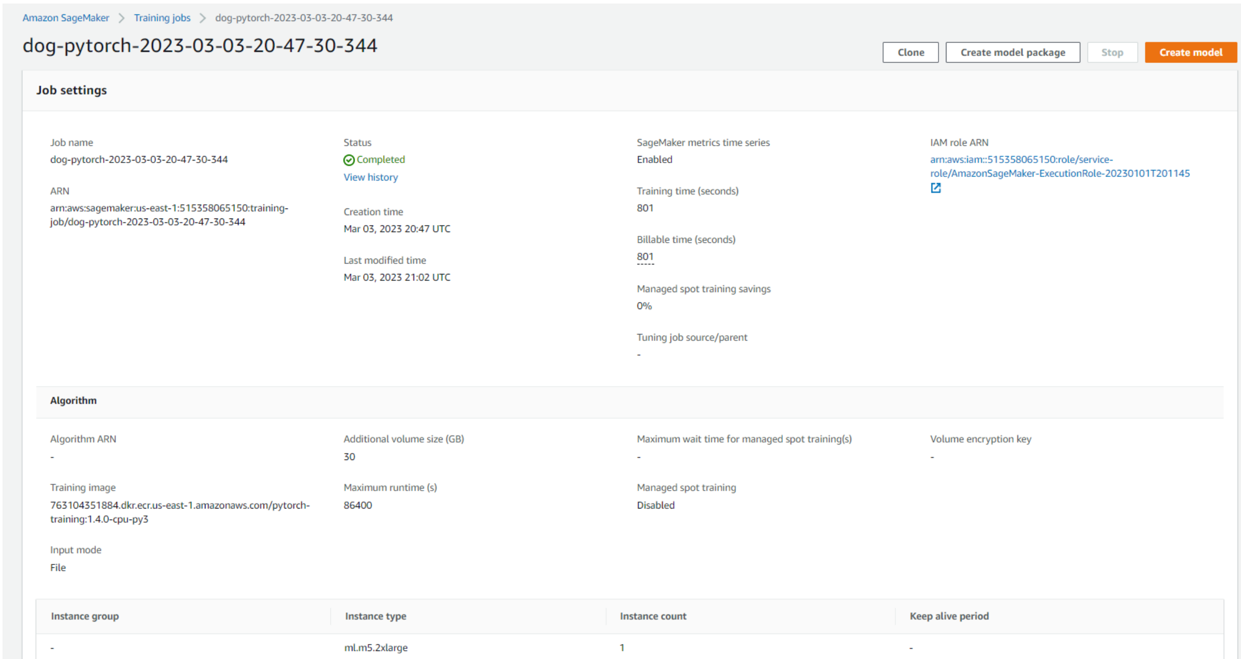
Describe the tuning results

[7]:
```
exp = HyperparameterTuningJobAnalytics(
    hyperparameter_tuning_job_name='pytorch-training-230303-1841')

jobs = exp.dataframe()

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

[7]:

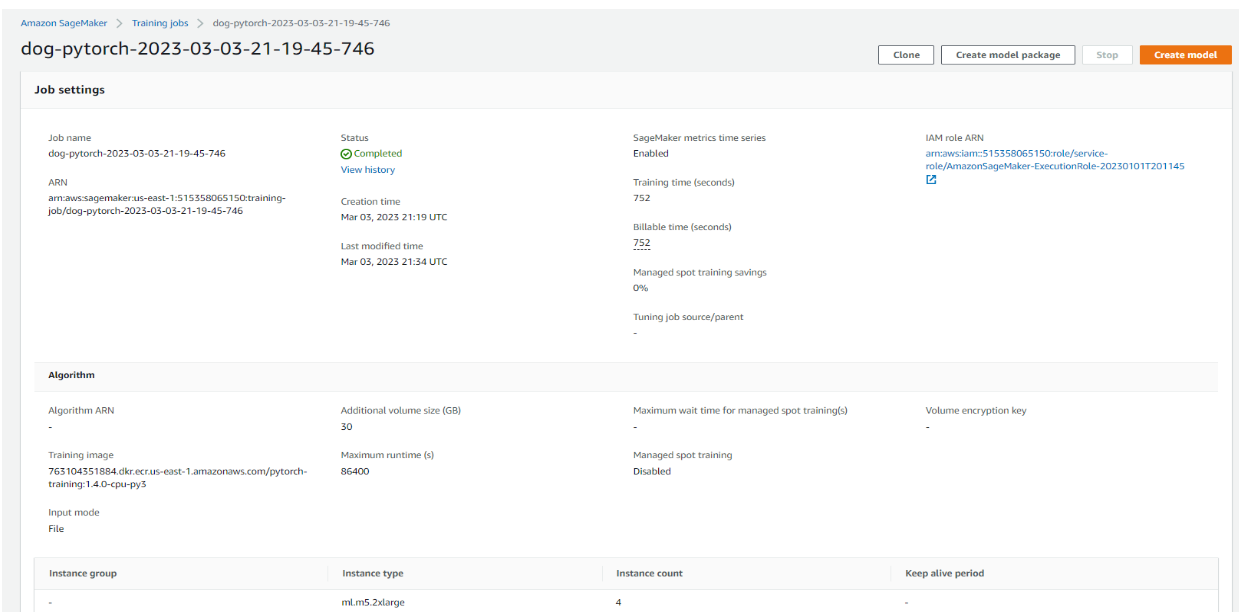
	batch_size	learning_rate	TrainingJobName	TrainingJobStatus	FinalObjectiveValue	TrainingStartTime	TrainingEndTime	TrainingElapsedTimeSeconds
0	"256"	0.022315	pytorch-training-230303-1841-002-88cc3128	Completed	1019.0	2023-03-03 19:06:31+00:00	2023-03-03 19:26:33+00:00	1202.0
1	"32"	0.017952	pytorch-training-230303-1841-001-07e2593e	Completed	150.0	2023-03-03 18:43:16+00:00	2023-03-03 19:03:27+00:00	1211.0

After training the model with the best parameters and also changed the instance type to "ml.m5.2xlarge". here we have the result.



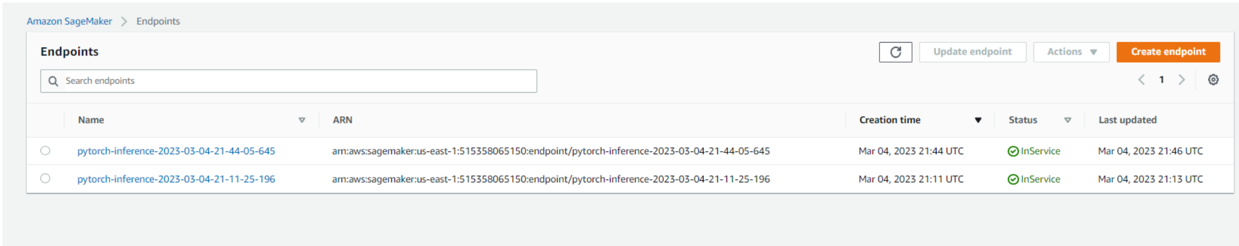
4- **Multi-instance training**

Here we altered the code for multi-instance training by increasing the instance count to 4.



5- **Deploying endpoints for single and multi-instance training:**

single instance endpoint: “pytorch-inference-2023-03-04-21-11-25-196” and multi-instance endpoint: “pytorch-inference-2023-03-04-21-44-05-645”



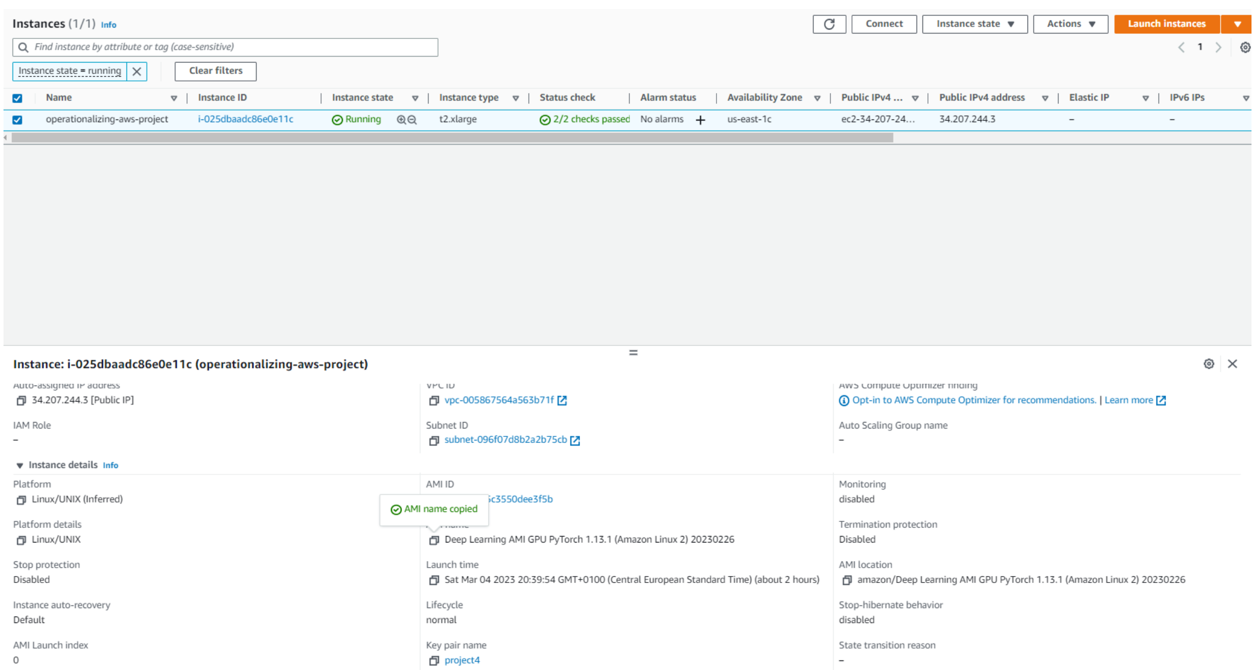
EC2 Setup

1- Preparing for EC2 model training

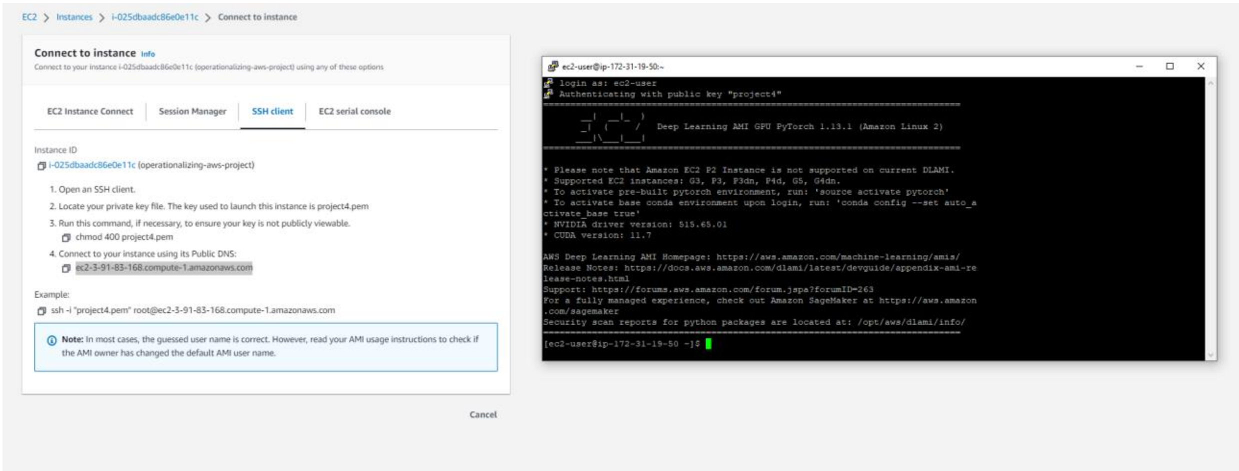
I used a t2. xlarge instance with the Deep Learning AMI GPU Pytorch 1.13.1 (Amazon Linux 2) Image. This setup has good performance and cost. After taking all the parameters into accounts for the training and duration, I've concluded that the T2 instance is the way to go by referring to AWS documentation:

"Amazon EC2 T2 instances are Burstable Performance Instances that provide a baseline level of CPU performance with the ability to burst above the baseline. T2 Unlimited instances can sustain high CPU performance for as long as a workload needs it."

And the best to go for a medium-sized instance so that we don't have to pay more while we do any other work then training

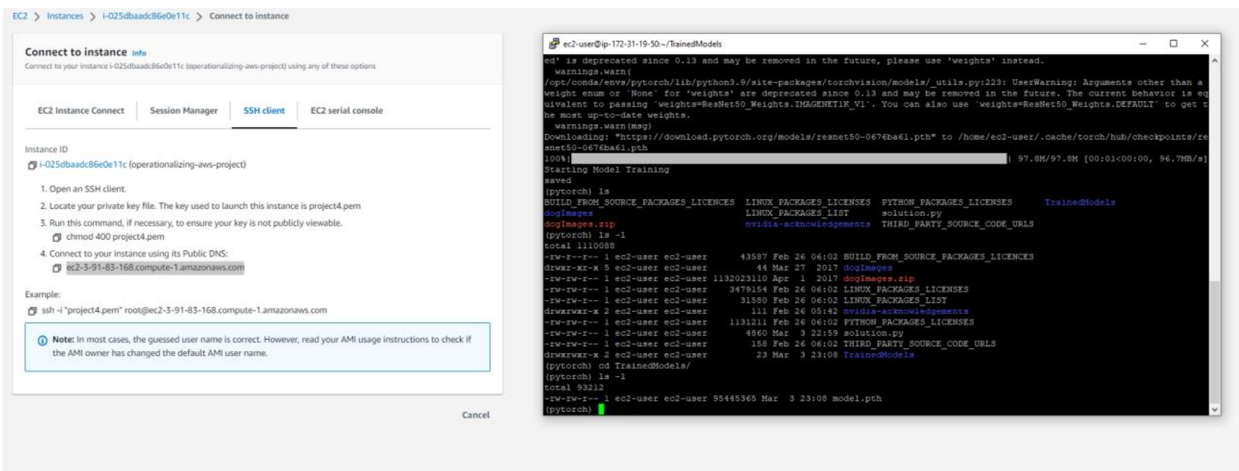


After launching the instance with Putty (an SSH Terminal), I've activated the "Pytorch" environment so we can run our code without any dependencies error.



2- Training and saving on EC2

After training the model was saved in the specified folder.



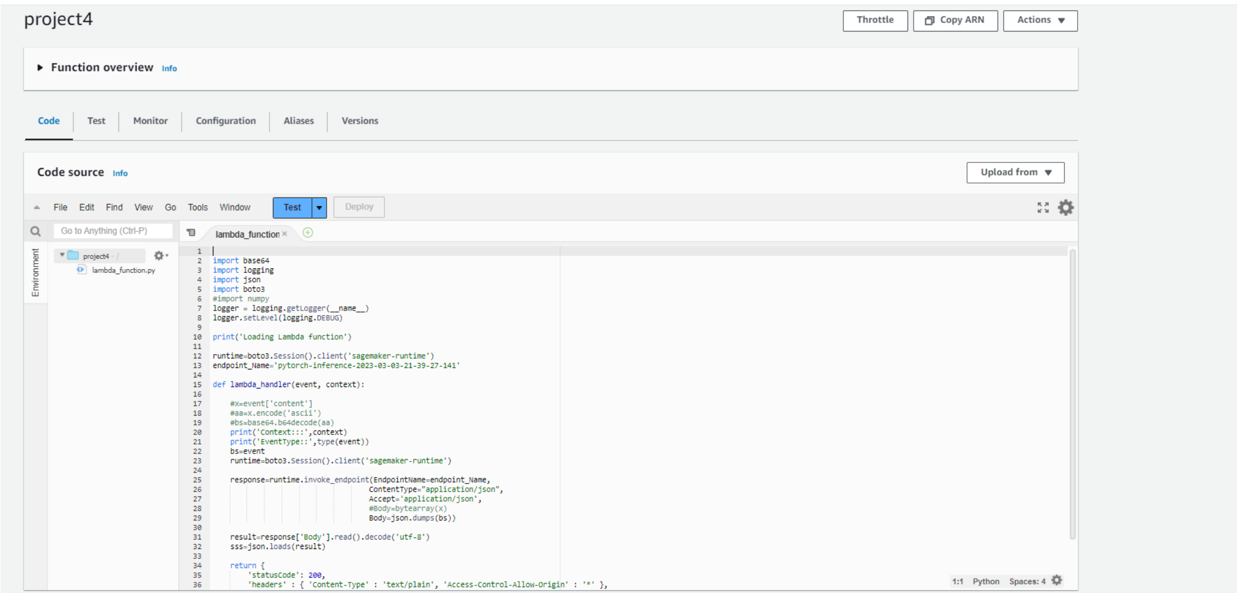
In the python code, we didn't need an Estimator or Tuner, all functions were handled by the code. and all variables like hyperparameters and output locations were hard-coded, so no arguments had to be passed to the script.

One thing is that deploying the trained model will take some work since the model was served locally.

Setting up a Lambda function

Here, we will use this lambda function to invoke our multi-instance training endpoint. As with the endpoint invocation from our Notebook, we need to serialize our input to JSON Format and then pass it to the endpoint, after which we will dump the result.

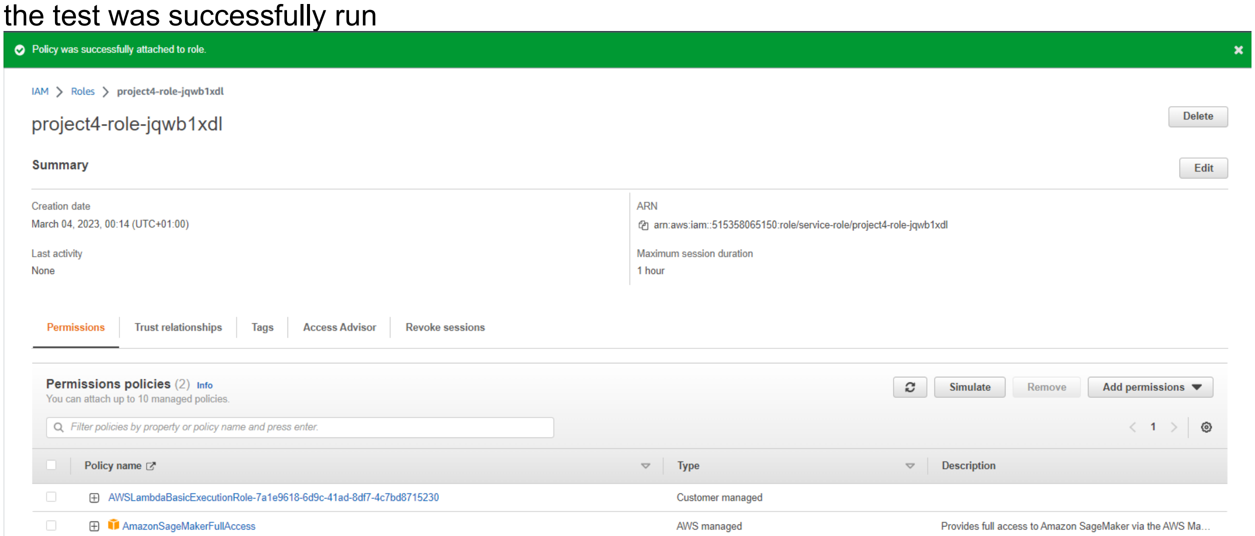
Replacing the endpoint name with our multi-instance endpoint

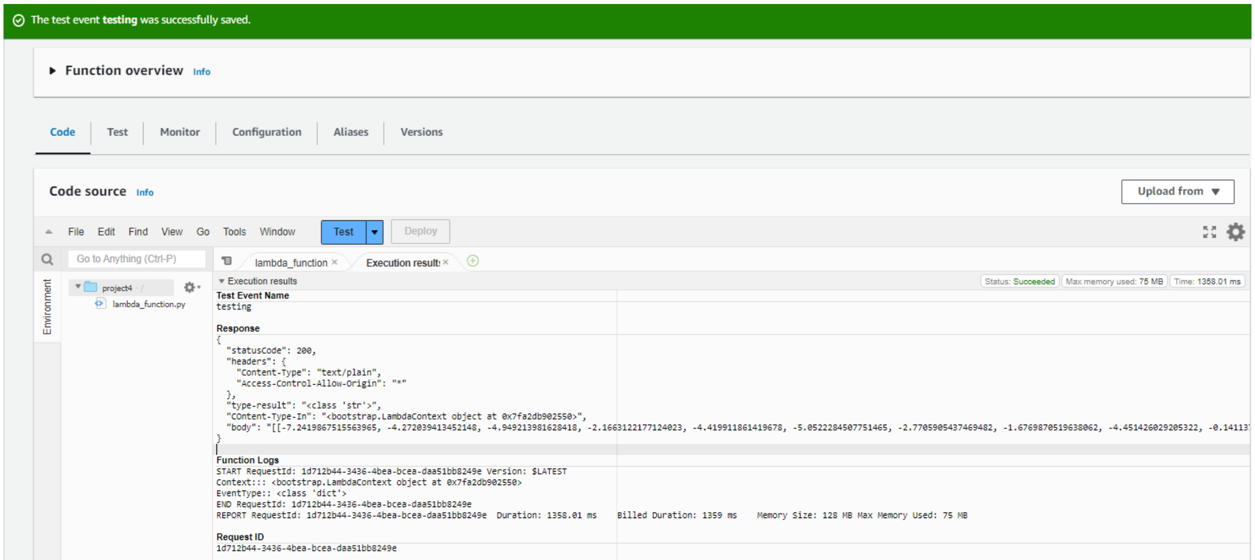


Lambda function security

1- Lambda function testing

after executing the lambda function event I got an error message, tracing the log I found it was an AccessDeniedException error. which means lambda didn't have access to Sagemaker resources, after modifying the lambda function role to have the “SageMakerFullAccess” policy. the test was successfully run



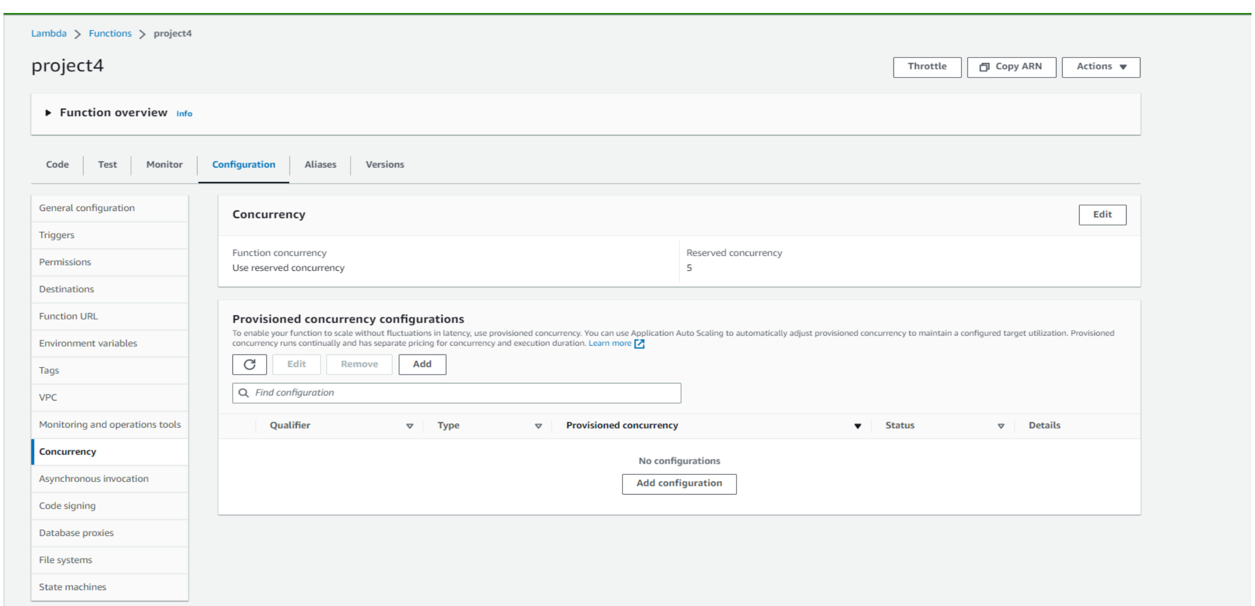


a lambda function having “Full Access” to the SageMaker resources is not an ideal type of security I want to deploy in the future. the policies should be stricter and carefully considered. and each lambda function is allowed to query only one endpoint.

Concurrency and Auto-scaling

1- Concurrency

First, I created a Version Config for the lambda function. I chose reserved concurrency because it would decrease latency issues in high-traffic situations above normal. The lambda function would be able to manage up to 5 simultaneous requests.



2- Auto-scaling

When configuring auto-scaling, I added these parameters to the endpoint runtime settings:

- A maximum number of instances of 3, a minimum of 1, and the target metric "SageMakerVariantInvocationsPerInstance" of 10 because I see it will fulfill the present requirement of our project also I added an input and output cooling time of 30 seconds.