Project Report

Setup, training, and deployment on SageMaker

Notebook Instance

The instance I used for my Jupyter notebook was ml.t3.medium. I selected this instance type since it is affordable and qualifies for the AWS free tier while being powerful enough to run the programs I intended to run on it. I was aware that while model tuning, training, and deployment would take place on other instances, I would be installing packages, downloading, unzipping, and uploading images on this instance. Additionally, I needed ml.t3.medium's quick launch feature because I expected to shut down and restart the notebook several times throughout the project.

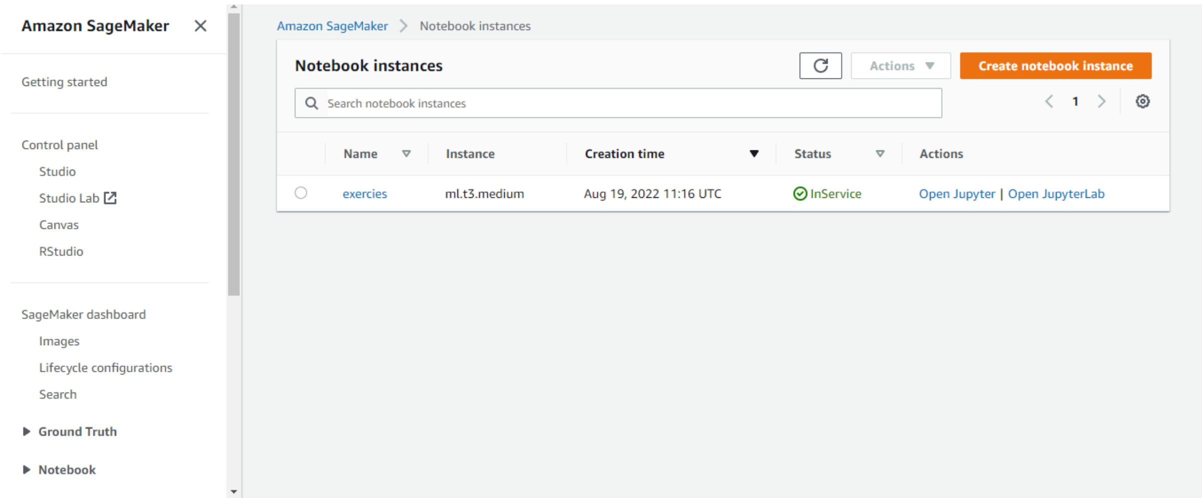

fig1 . Notebook instance launch

S3 Bucket

I used the default sagemaker S3 bucket to copy the data into (sagemaker-us-east-1-968977130828) . I changed the references in train_and_deploy-solution.ipynb to point to this bucket, and I uploaded the unzipped dogImages dataset to a file called :
s3://sagemaker-us-east-1-968977130828/dogImages/

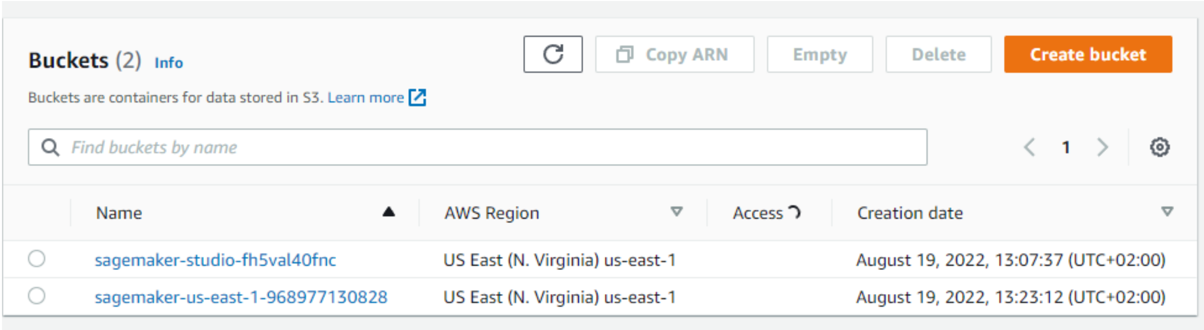

fig 2. S3 bucket

Training and Deployment

For hyperparameter tuning I used an ml.m5.xlarge instance with 1 jobs. This instance is part of the free tier for training in SageMaker, and from experience I knew this would take about 25-30 min, which was acceptable. The best hyperparameters were a batch size of 64 and a learning rate of approximately 0.005831981479799012.

I used an ml.m5.xlarge instance to train the estimator, using the best hyperparameters from my tuning job. Once it was trained, I deployed the model to an endpoint on ml.m5.xlarge instance. I chose this type of instance because it wouldn't need to handle very many or very large inference queries. The endpoint's name shown in figure 3 .

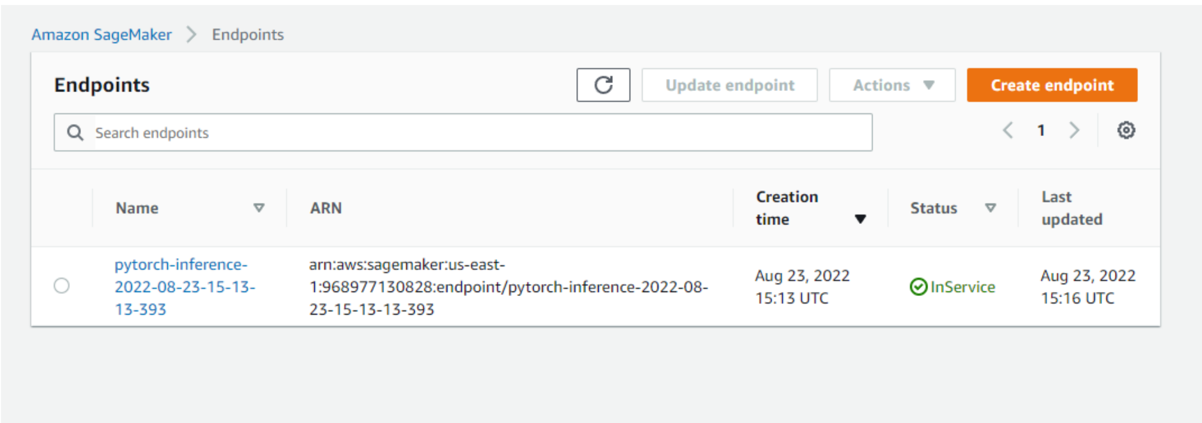

fig 3 . single instance endpoint

After running the rest of the notebook, to confirm that the endpoint could perform inference, I went back and used 2 instances of type ml.m5.xlarge to perform multi-instance training. Once it was trained, I deployed to an endpoint on ml.m5.large instance. The endpoint's name shown in figure 4

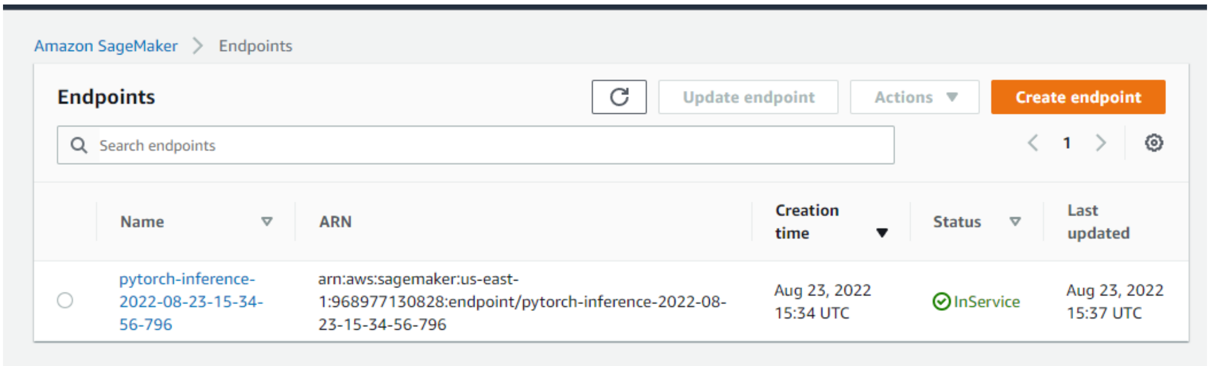

fig 4 . multi instance endpoint

EC2 Instance

I selected the m5.xlarge instance of the Deep Learning AMI GPU Pytorch. I was confident that this would be potent enough to finish model training quickly and wouldn't deplete my account if I had to keep it running for a few hours to troubleshoot.

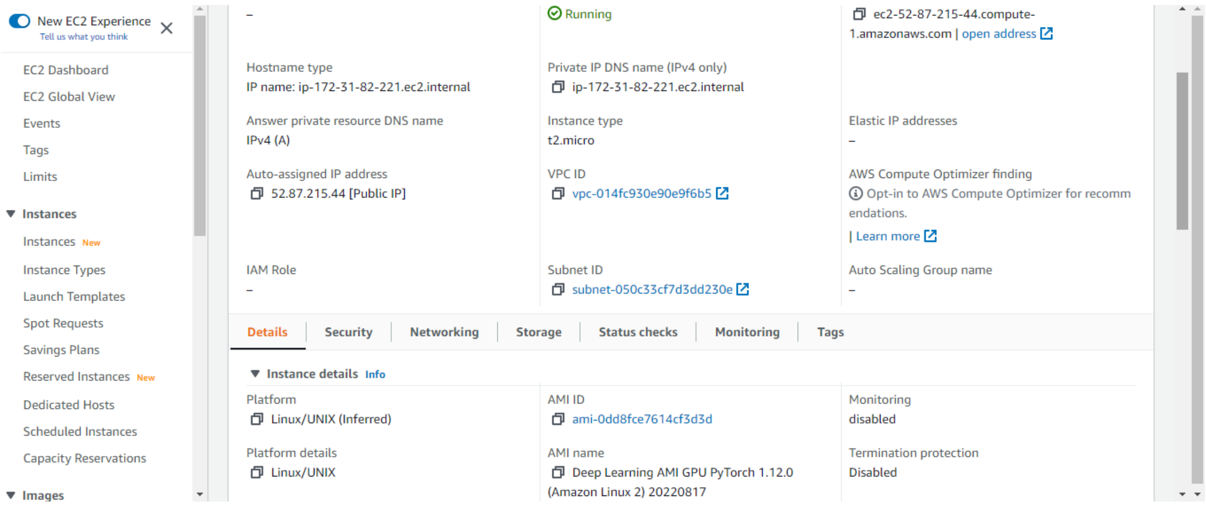

fig 5 .EC2 instance

I started the instance, downloaded the dataset, trained the model, and then saved it in the directory for trained models.

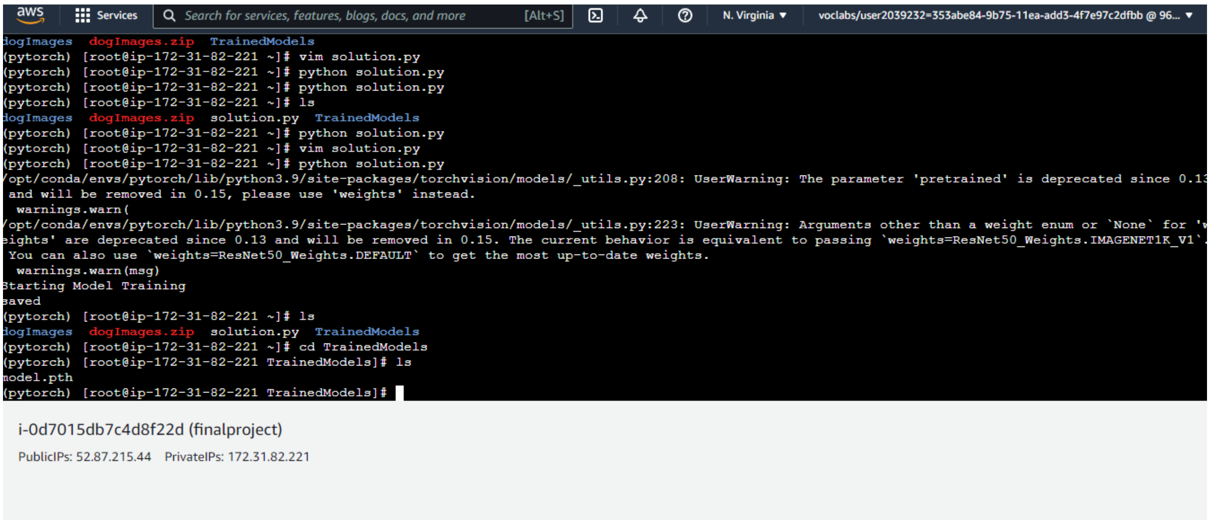

fig 6.EC2 instance training

EC2 VS Sagemaker

With a few significant exceptions, the code in ec2train1.py is comparable to train and deploy-solution.ipynb and hpo.py. First off, all training is done locally by the EC2 script. As opposed to SageMaker, where the estimators are fitted on distinct instances from the notebook where the code is performed, and the training data, model, and output are all kept on S3. This contributes to the EC2 script's lack of a parser or main function. The learning rate, batch size, training, model, and output folders are all sent along to the main function via the SageMaker parser.

All of these are merely provided in the script for EC2. Only because the dataset is local, the training is local, and the model can be saved locally, is this possible. Last but not least, from EC2, the trained model cannot be deployed straight to an endpoint. We only know how to deploy endpoints from models within SageMaker, therefore before we could install an endpoint to conduct real-time inference, we would need to import the model from the EC2 instance into SageMaker.

Lambda Function Setup

After attaching the appropriate permissions to the Lambda function role, I ran a successful test of my lambda function as shown in figure 7 .

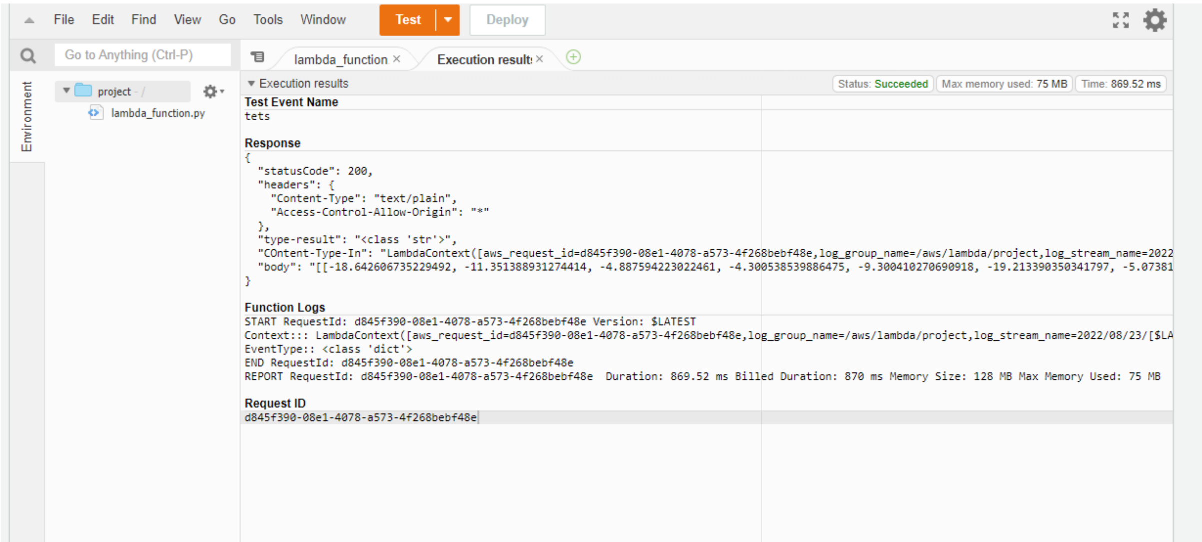

fig 7.Successful Lambda function test

You can check the output also provided on my repo in the text file .

Security

I've given SageMaker's Lambda function complete access. This might be used, among other things, to list the secrets in Secrets Manager, acquire metric data from CloudWatch, establish or delete VPC endpoints in EC2, or get or delete assets from S3. This recommends that we should be cautious about who is first given permission to build Lambda functions, monitor the Access Advisor for roles frequently to see which services are being accessed and when, and withdraw rights if we notice anything suspicious. Around AWS accounts, we should utilize fundamental operational security procedures like multi-factor authentication, privileged account workstations, and routine active-user/role audits.

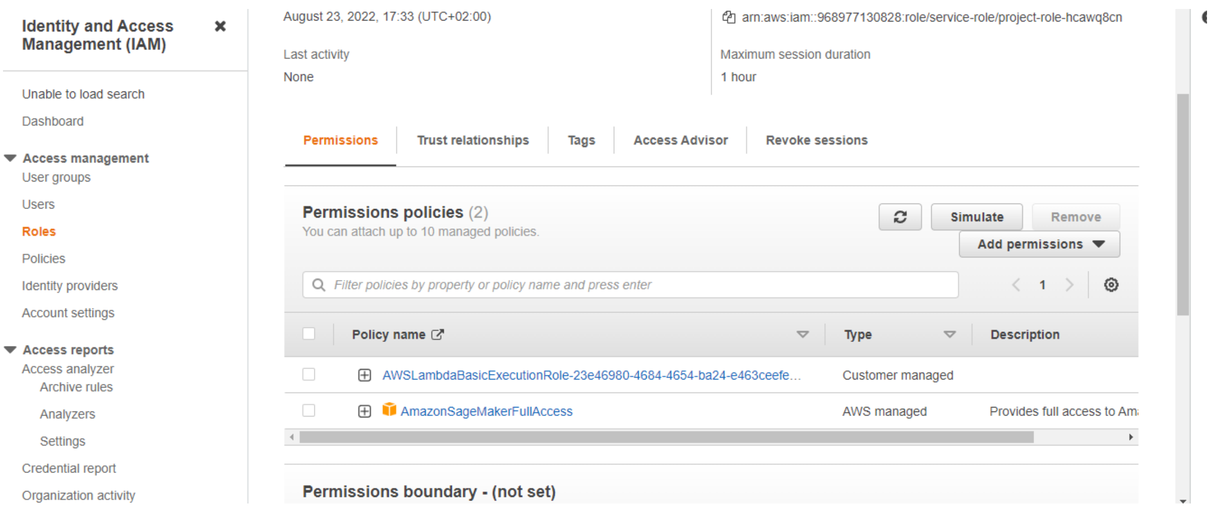

fig 8.Permission for lambda function

Concurrency

My Lambda function was configured, and I selected 5 provided concurrency. To show my understanding, I added a small amount of each type of concurrency, and they would be more than enough for me to use the Lambda function as shown in figure 9 .

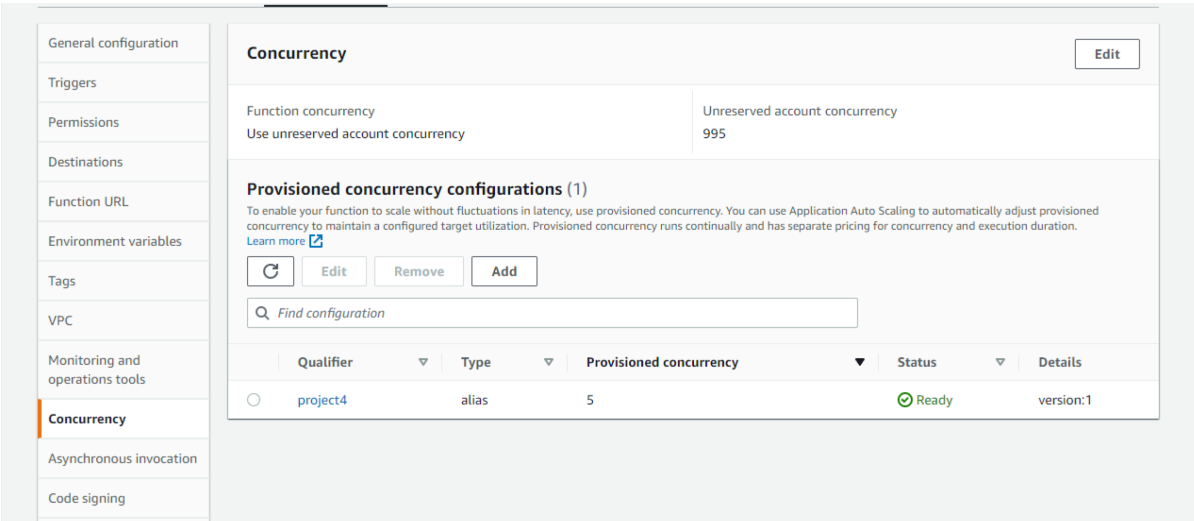

fig 9 Concurrency of lambda function

Auto scaling

For my deployed endpoint, I similarly set up auto-scaling with a minimum instance count of 1, a maximum instance count of 4, and scale-in and scale-out cool down intervals of 30 seconds. Those settings worked perfectly for the activity we completed earlier in the course, so both the cost and performance would be fine as shown in figure 10 .

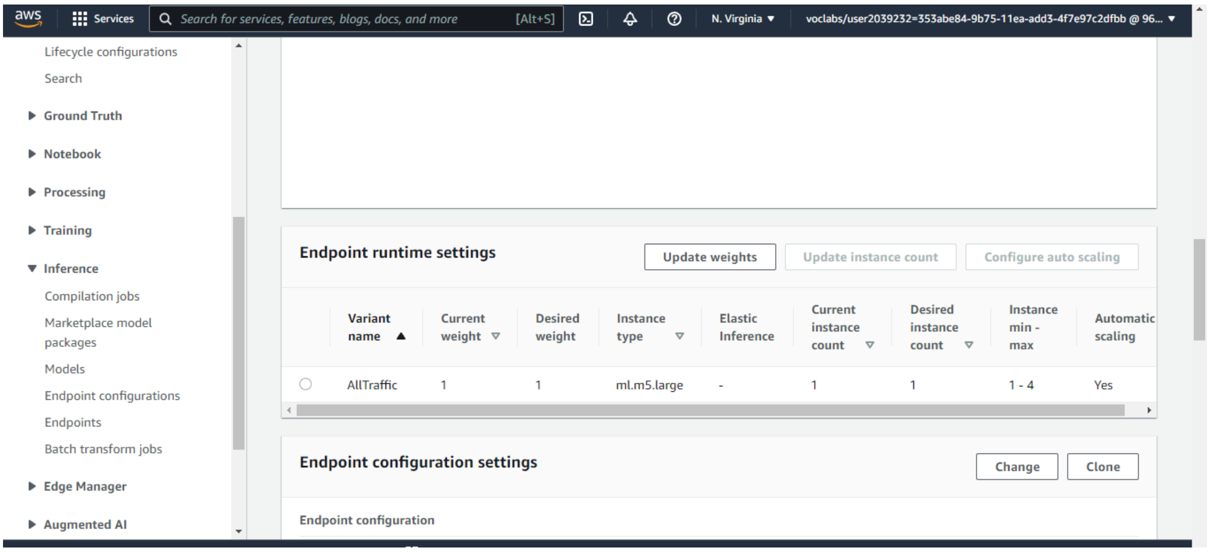

fig 10.Autoscalling for my endpoint