Write Up of the Project: Operationalizing an AWS Machine Learning

Samin Poudel*

This project is started with an objective of learning the process of adjusting, improving, configuring, and preparing a trained model for production graded deployment using the different tool of AWS Sagemaker.

Notebook Instance, Training and Deployment

We started the project by creating a notebook instance named AWS-Chapter5-project with Standard Type of Notebook **ml.t2.medium** as shown in Fig 1**.** This instance was chosen as it is the most optimal choice in terms of optimizing cost. An image classification task might need instances with higher RAM and capacity, but the idea was to check if the image classification task can be performed with optimal cost in a reasonable time.

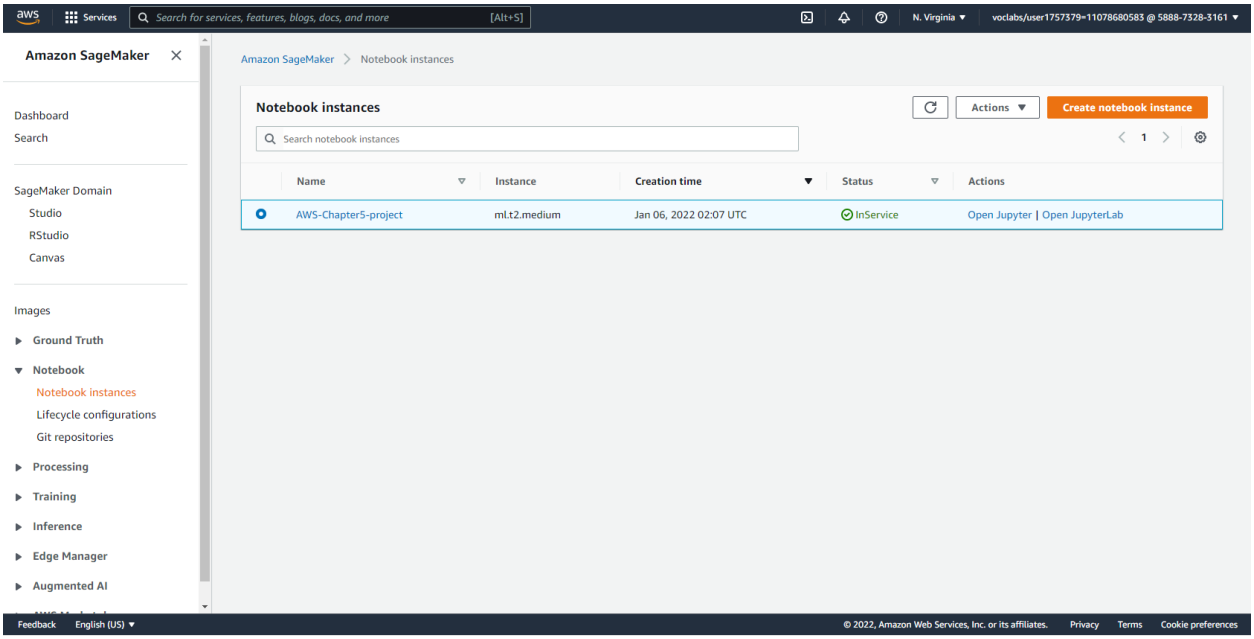

Figure 1. notebook Instance

After that using the created instance, the dog breed data was downloaded and then uploaded to a S3 bucket named chapter5project by running code in the jupyter notebook named train_and_deploy_solution.ipynb. Thd S3 bucked with data is as shown in Fig. 2.

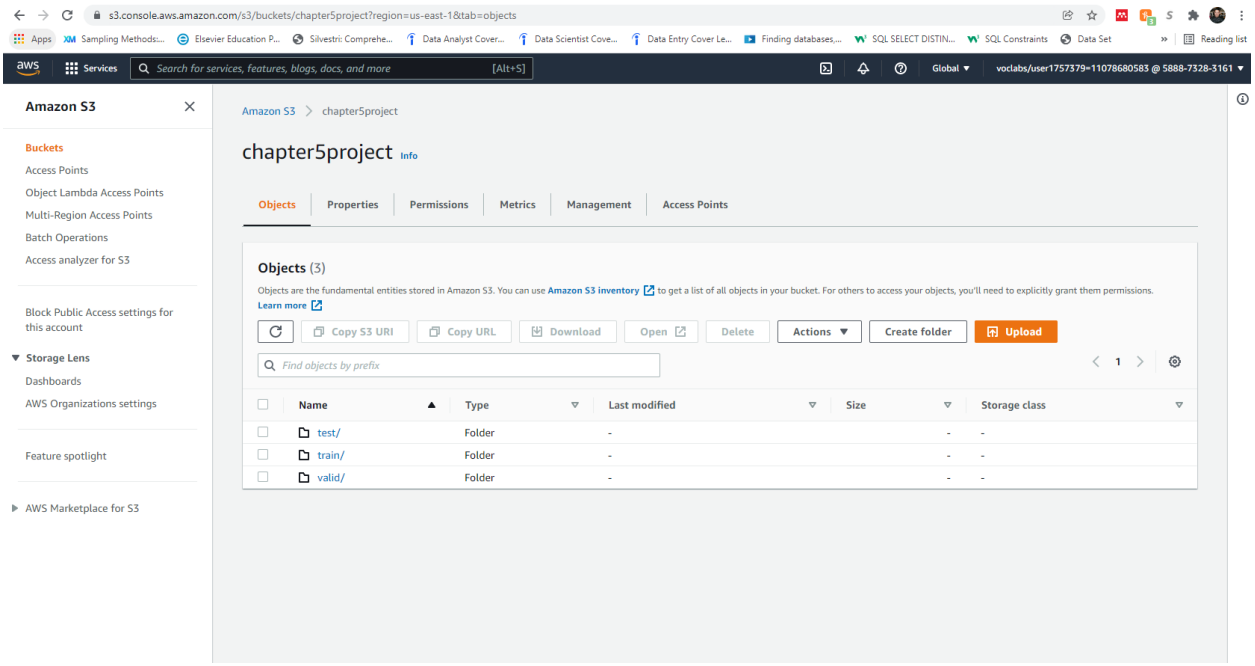

Figure 2. S3 Bucket with Data Uploaded

After that we performed hyperparameter tuning, single instance training of model, multi-instance training of model. Next, we deployed the endpoints for both single instance and multi-instance models. The screenshots of tuning, training and deployment are attached and the process of performing these tasks is as followed in the train_and_deploy_solution.ipynb. file

EC2 Training

Next step was to perform the training in the EC2. The EC2 instance was created, and the training job was performed in it. The AMI assigned to the EC2 instance was Deep Learning AMI (Amazon linux 2) Version 56.0 to make sure the required modules where available for the python code we were executing.

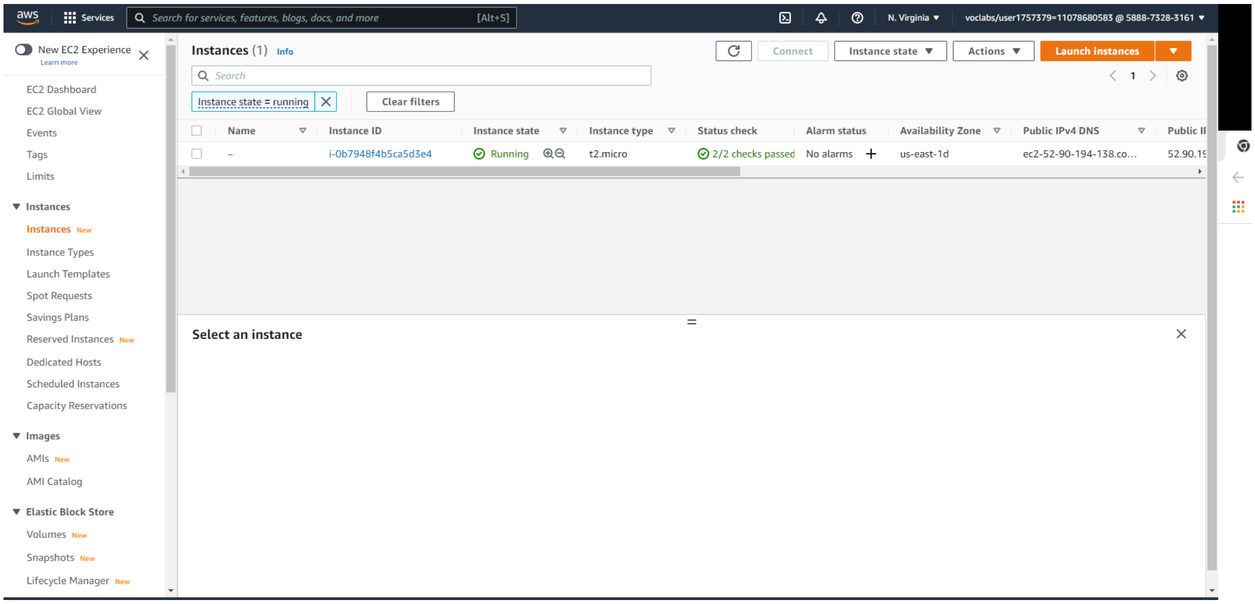

Figure 3. Running EC2 instance

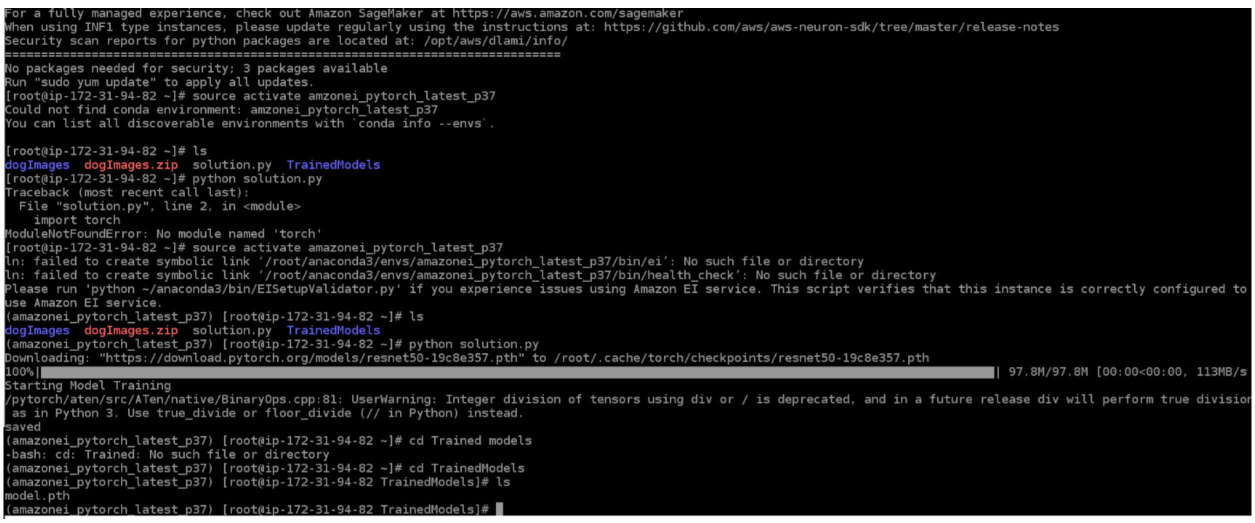

Figure 4. Execution in EC2

We had to perform the execution of the model in EC2 like one time execution script, while the one in Sagemaker notebook had the flexibility of step by step. Sagemaker gives us the ability to tune the best hyperparameters and use them, but in EC2 script hyper parameters are to be predefined. EC2 are cost effective than Sagemaker, while Sagemaker seemed to be more user friendly than the EC2 code.

Lambda Function Set Up, Security and Testing

A Lambda function was set up to invoke the endpoint we created in step 1 of this project as in Fig. 5.

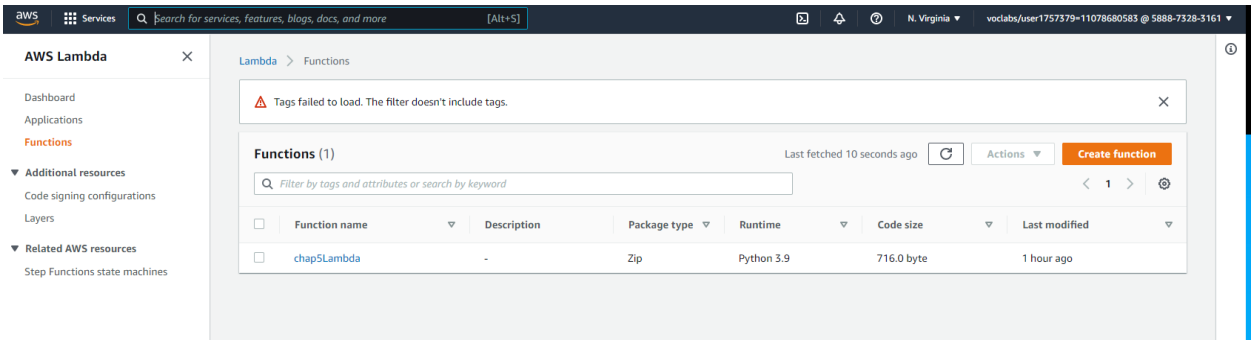

Figure 5. Lambda Function

Then the given test case was applied to lambda but the execution failed as the lamda Function did not have the policy attached to access sagemaker and invoke the endpoint. Therefore, we added the Sagemaker Full Access policy to the lamda Fucntion as shown in Fig6. and the execution with the test case was possible as shown in Fig. 7. The testing of Lambda Function gave list of 33 numbers which the prediction of the class membership of the picture submitted from the test case.

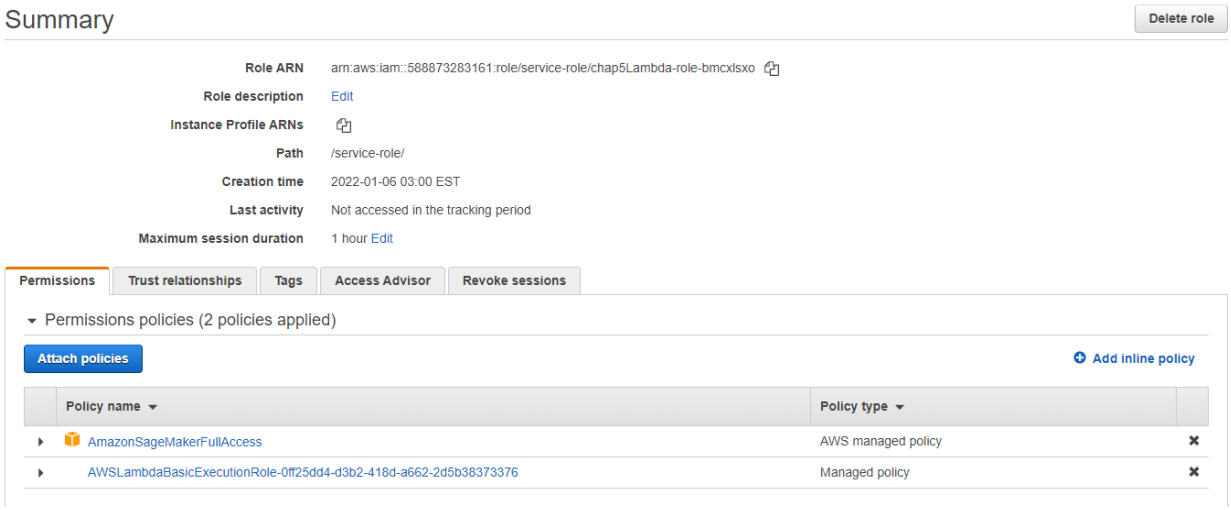

Figure 6 Lamda Function: Policy Added

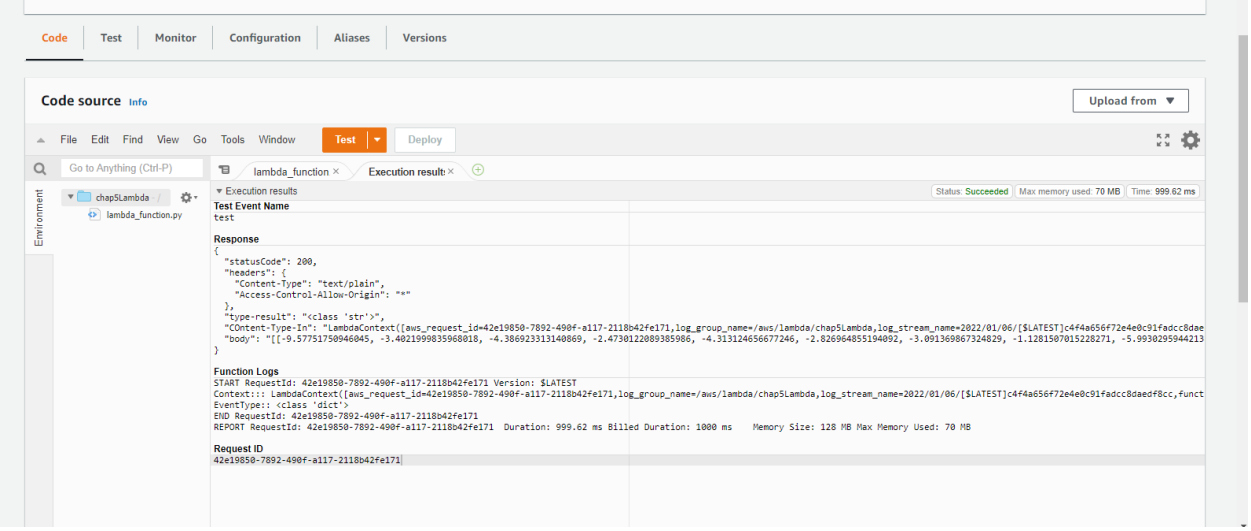

Figure 7. Lamda Function Successful Execution with the Test Case

A list of IAM roles and the is as shown in Fig. 8 and the two policies for lambda Function is already shown in the Fig. 6.

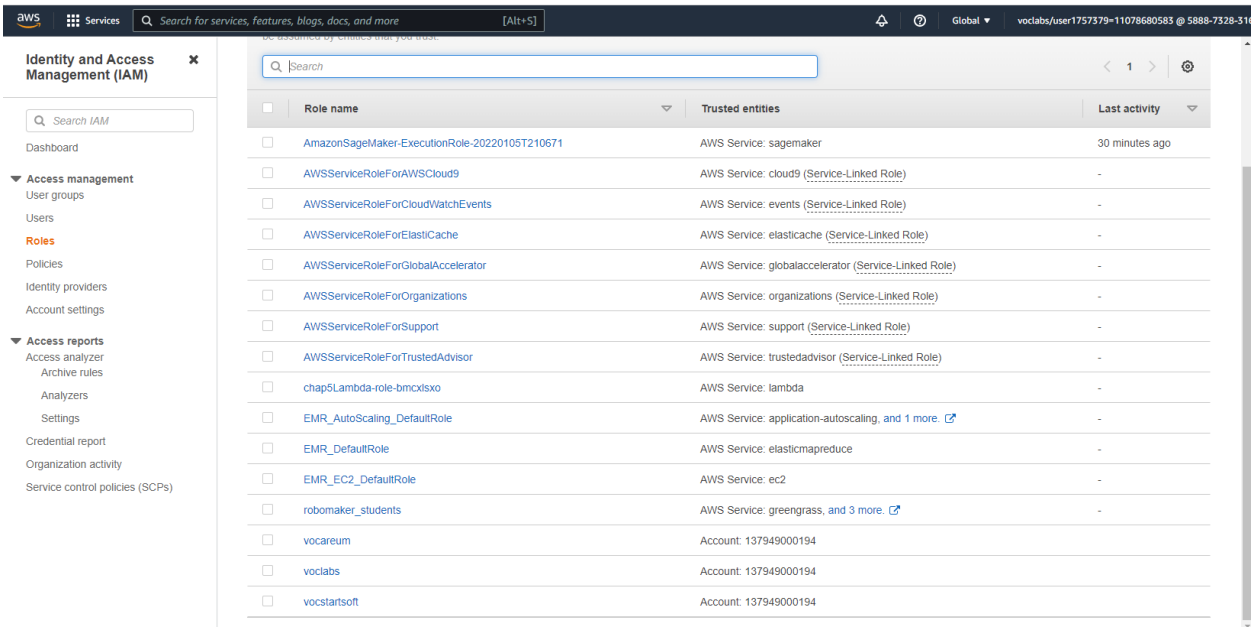

Figure 8. A List of IAM Roles

Addition of AmazonSageMakerFullAcces policy may add some sort of insecurity, so for more secure set up can be based on the location and the role of the endpoint that the lambda function is trying to invoke.

Concurrency and Auto-Scaling

The concurrency for lamda function was created by setting up a version for it. The task of the concurrency is to help during the high traffic input requests, but at it may add some cost, care must be taken to add the amount of concurrency. Similar idea goes for autoscaling of the Endpoints. The screenshot of concurrency of lambda function is as shown in Fig. 9. And, the autoscaling of an endpoint is as shown in Fig. 10.

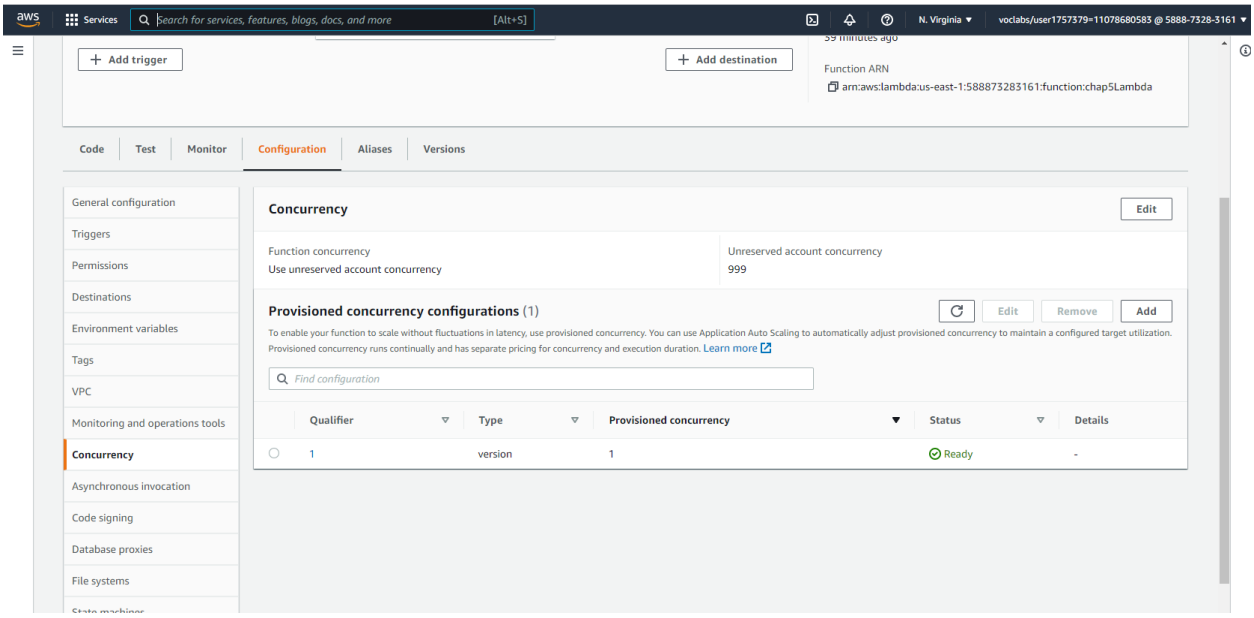

Figure 9 Concurrency of Lambda Function

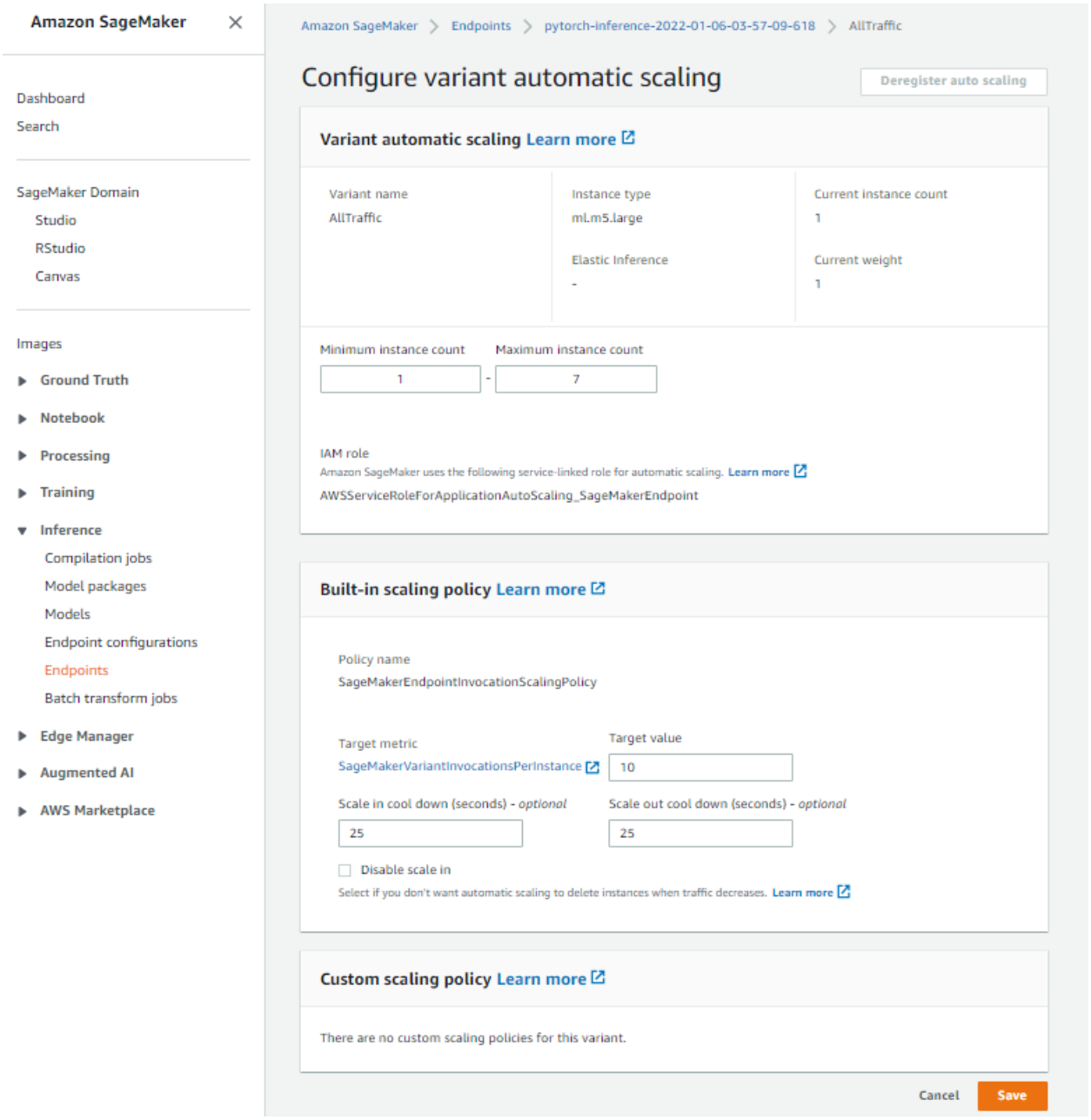

Figure 10 Auto-scaling of an Endpoint