Project Summary: ML Ops

Training and Deployment

The sagemaker notebook instance created used the ml.t3.medium as the instance type because that had the lowest price per hour. I felt that was a good starting point being that I did not know how much computing resources I would need for the project. If it seemed like it was taking too long to process the notebook then I would move up to the faster instances.

EC2 Training

For the instance type, t3.large was chosen because that was similar to what used for the sagemaker notebook and it seemed to be sufficient to run the notebook in a reasonable amount of time. The image chosen was the Deep Learning OSS Nvidia Driver AMI GPU PyTorch 2.5 since this EC2 instance is to train and deploy a machine learning model.

Some of the differences between the code in the sagemaker notebook versus the code used for training in the EC2 instance are that in the sagemaker notebook hyperparameters are optimized so that the best parameters are used for training, whereas the code for the EC2 instance they are hard coded or chosen. In the sagemaker notebook extra packages are used from sagemaker such as tuner, pytorch and debugger on top of the packages already used in hpo.py. The code for the EC2 instance is only using the packages that are found in hpo.py.

Security and Testing

In general the workspace is secure because the IAM role created for the lambda function that I created is assigned to or for that particular lambda function. However, because the role has the sagemaker full access policy, if that role were to be assigned to another user then that user would have full access to sagemaker. It may be better to specify access to the sagemaker endpoint for the role just in case. This would be the only vulnerability in the workspace, even though it would require root access or administrator full access to assign the lambda role to a user or for some other purpose.

Concurrency and Auto-Scaling

5 reserved concurrency instances were configured initially to handle a 5x spike in increased traffic. Because I do not know what kind of traffic the lambda function would encounter I believe 5 is a good starting point being that when the function is fed a picture of a dog the time to provide a response is fairly quick like 2 seconds. Also, I went reserved instead of provisioned due to the higher cost of provisioned instances, again because I do not know what kind of traffic that will be encountered the lower cost is a better starting point.

5 maximum instance counts was chosen for the autoscaling property of the endpoint to match the amount of concurrency instances set for the lambda function. 30 seconds was chosen for the scale in cool down to have quick response when traffic picks up and 30 seconds was chosen for the scale out cool down to close out any idle instances quickly to save on costs.