A Writeup for Operationalizing an AWS ML Project

The purpose of this writeup is to describe the decision I made during the project and justifications for them. It covers Sagemaker Instance selection, EC2 Instance selection, EC2 Code Difference, Lambda function, Test Result for Lambda Function, AWS Workspace Security and Concurrency and Autoscaling.

SageMaker Instance

For the Sagemaker instance, I selected the most basic `ml.t3.medium` instance type, which costs $0.05/hour and has 2 CPUs and 4GB of memory. This is because the hyperparameter optimization, model training, and model deployment tasks we need to complete are all handled by the Sagemaker container, so our instance only needs to be able to load a few essential libraries to run the submission script.

EC2 Instance

For the EC2 instance, I selected `ml.m5.2xlarge` as the instance type and it costs $0.461/hour. Compared to the ml.m5.xlarge used for training in Sagemaker notebook which takes about 30 minutes, it should take less time with the `ml.m5.2xlarge`, which is what I want, and the price is reasonable.

EC2 Code Difference with Sagemaker

EC2 code only uses the hyperparameters inside the python script. In contrast to SageMaker, where the code is executed through a managed notebook, writing code for EC2 is similar to writing code for Python scripts that will be executed via a command-line or terminal.

Lambda Function

The lambda function is written to accept data in a JSON object and invoke an endpoint that returns a response with a prediction. After that, the function creates a dictionary out of data like the status code and body and returns it as the function's response.

Test Result for Lambda Function

I use the following text for the test event:

```
{ "url": "https://s3.amazonaws.com/cdn-origin-etr.akc.org/wp-content/uploads/2017/11/20113314/Carolina-Dog-standing-outdoors.jpg" }
```

This is the test result:

```
[[-11.5274292, -2.407780408859253, -4.484838962554932, -5.3391008377075195,
-10.926896095275879, -4.425899505615234, -8.4246187210083, -2.857327938079834,
-6.604519844055176, -5.456014156341553, -3.7839150428771973, -3.137073278427124,
-7.370544910430908, -8.03581714630127, -8.529093742370605, -10.43710994720459,
-5.5333638191223145, -9.14289665222168, -0.12866991758346558, -7.953580856323242,
-3.6379873752593994, -2.9352548122406006, -0.253826379776001, -6.19246768951416,
-5.590920448303223, -8.882161140441895, -5.566829681396484, -0.544374406337738,
-12.586575508117676, -2.454815626144409, -7.308746814727783, -6.596512317657471,
-0.5196906328201294, -8.821796417236328, -1.1116344928741455, -2.842531204223633,
-7.648394584655762, -6.9921064376831055, -4.3556413650512695, -8.820635795593262,
-3.2644126415252686, -2.216978073120117, -9.175309181213379, -9.784005165100098,
-12.165433883666992, -8.596067428588867, -5.395246982574463, -2.514040470123291,
-4.982823848724365, -3.446772575378418, -11.017499923706055, -8.454011917114258,
-10.435539245605469, -3.32136869430542, -2.673523426055908, -8.395874977111816,
-9.213384628295898, -9.281765937805176, -4.752413272857666, -1.1308624744415283,
-4.492363452911377, -1.7632801532745361, -3.5538218021392822,
-0.3554377257823944, -7.898016452789307, -5.0349440574646, -9.195297241210938,
-4.65649938583374, -7.463343143463135, -1.0483142137527466, -7.169613838195801,
-0.6898795962333679, -2.6490845680236816, -4.7636799812316895, -7.24975061416626,
-4.917212963104248, -10.391488075256348, -5.4493408203125, -4.253044605255127,
-3.066664218902588, -5.1529645919799805, -7.4855241775512695, -9.868621826171875,
-0.8030180931091309, -9.546015739440918]]
```

AWS Workspace Security

By configuring the IAM roles of the AWS component(Sagemaker, Lambda Function) that the project must interact with, I have so far completed it without incident. The AWS Workspace is secure overall. Regarding the vulnerability, I believe we can regularly remove any roles with full access that are no longer in use. We can also keep an eye on log logs using a CloudWatch pattern to look for any unusual activity.

Concurrency and Autoscaling

I set up 5 Reserved concurrency and 3 Provisioned concurrency for Lambda function concurrency configurations. I actually think this can be used as a test configuration, and I will change it later according to the actual business requirements. For small to medium volume users and request scenarios, I think 5 Reserved concurrency is enough, and 3 Provisioned concurrency is for backup in case the whole pipeline goes down due to the sudden increase of requests.

For autoscaling, below is my configuration:

```
Maximum Instance Count: 3
Target Value:  30
Scale in cool down: 30 seconds
Scale out cool down: 30 seconds
```

One instance of the m5.xlarge endpoint, with its vCPUs and 8GB of memory, is sufficient to make an inference. I increased the Maximum Instance Count to 3 to make sure it could handle the sudden increase in throughput.

In order to prevent the endpoint from being overloaded with traffic while also preventing autoscaling from occurring when it is most likely not necessary, I set the Target Value for the built-in scaling policy to 30, which may be a sufficient number. 30 seconds were chosen for both the Scale-In Cool Down and Scale-Out Cool Down values. This was done to save money and allocate resources and deals efficiently.