1. SageMaker instance

Due to [this link](https://github.com/awsdocs/amazon-sagemaker-developer-guide/blob/master/doc_source/notebooks-available-instance-types.md), the “ml.t3.medium” is a suitable option to choose the notebook instance type: fast launch, CPU-intensive, and offering a balance of compute, memory, and network resources.

For training, I chose the “ml.m5.xlarge” instance type due to its acceptable time processing and resource consumption.

1. EC2 instance type

We have a typical task which needs a balance of time, computation and networking resources. As described in [this link](https://aws.amazon.com/ec2/instance-types/) and [this link](https://github.com/awsdocs/amazon-sagemaker-developer-guide/blob/master/doc_source/notebooks-available-instance-types.md), then we can choose a general purpose instance. Among these instances, I chose “ml.m5.xlarge” instance type due to its acceptable time processing and resource consumption and shining former experience I had when I was working with.

1. EC2 code

The script codes for Sage Maker are designed to be run from a notebook. But EC2 codes are like Python scripts that can be run through a command line interface.

1. Lambda function structure

As described in [this link](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/sagemaker-runtime.html#SageMakerRuntime.Client.invoke_endpoint), at the input and the output of the “invoke\_endpoint” method, we deal with “dict” or in other words “JSON” objects to get/give the data to the user. So the method returns the response as a JSON object which is a package of data and metadata of the response.

1. Lambda function test result

[0.17870040237903595, 0.13383135199546814, -0.09375334531068802, 0.2592350244522095, 0.47878167033195496, 0.24253161251544952, -0.012547305785119534, 0.025524454191327095, -0.3065299987792969, -0.05006171390414238, 0.4960617423057556, 0.2937588095664978, -0.10482137650251389, 0.20727938413619995, 0.5738649964332581, 0.11281649023294449, 0.3919632136821747, 0.006053171586245298, 0.10420925915241241, 0.14569886028766632, 0.38699135184288025, 0.03818243369460106, 0.010774191468954086, 0.24934278428554535, -0.14387431740760803, -0.16589699685573578, 0.17532989382743835, -0.3856702148914337, 0.4190075695514679, -0.016984570771455765, 0.2693136930465698, 0.21761853992938995, -0.0067712995223701, 0.12360124289989471, 0.008986278437077999, 0.34518298506736755, 0.10455890744924545, 0.19960807263851166, 0.3352145552635193, 0.1075577437877655, 0.31086888909339905, 0.11815930902957916, 0.0688147023320198, 0.15545524656772614, 0.21596749126911163, 0.23912835121154785, -0.033849433064460754, 0.021845242008566856, 0.0632118433713913, 0.07483804225921631, 0.291199654340744, -0.09055902063846588, -0.24620088934898376, 0.14463458955287933, -0.047534193843603134, 0.20987142622470856, 0.21283788979053497, 0.1323159784078598, -0.27583733201026917, 0.34829121828079224, 0.10004189610481262, -0.01648811064660549, 0.03226951137185097, -0.2865918278694153, -0.31871363520622253, -0.5094499588012695, -0.3448479175567627, 0.29629412293434143, 0.21455055475234985, -0.09459517151117325, 0.20433378219604492, -0.07251743972301483, -0.18220555782318115, -0.2413235902786255, -0.11346618831157684, 0.058772191405296326, -0.22938238084316254, -0.12100984156131744, 0.17572811245918274, -0.10344395041465759, 0.2045227736234665, 0.07121706753969193, -0.15153735876083374, -0.05212985724210739, -0.3077249228954315, 0.14454737305641174, 0.18672487139701843, -0.07933492958545685, 0.2024795114994049, 0.10165112465620041, 0.13367505371570587, -0.1221875324845314, -0.32800599932670593, -0.20697812736034393, -0.029508886858820915, -0.16093388199806213, -0.0038375090807676315, -0.14817175269126892, -0.10099371522665024, -0.2760518193244934, -0.013176222331821918, -0.4690702259540558, 0.24589566886425018, -0.13703452050685883, -0.2985295057296753, 0.1966293603181839, -0.29853489995002747, -0.6490413546562195, -0.03280797600746155, -0.5440608859062195, -0.12705713510513306, 0.01589452475309372, -0.4295775294303894, -0.18115298449993134, 0.3090229332447052, -0.5633568167686462, 0.023684121668338776, 0.14266140758991241, -0.5151317715644836, -0.350257933139801, -0.5966724157333374, -0.2933812141418457, -0.10493782162666321, -0.06118430569767952, -0.4685129225254059, -0.3438847064971924, -0.10647889971733093, -0.46159282326698303, -0.184151291847229, -0.17733435332775116, -0.3439982235431671, -0.6250125169754028, -0.45641621947288513]

1. Secutiry

I examined all the roles that existed in the project and my result is that all the roles are either used by my services or provided by aws infrastructure itself and the workspace is secure in both cases. Generally, any “full access” to resources in software development can lead to a danger zone which may have some problems such as race conditions and data conflicts. However, since AWS provides a FullAccess role to SageMaker itself, then it’s reliable and trustworthy.

1. Concurrency & Auto-scaling

For concurrency, I used provisioned concurrency since we have a requirement receiving a lot of traffic, so we need an automatic way for handling that. So I set provisioned concurrency to 3. For auto-scaling, due to the strength of the instance type I chose(ml.m5.xlarge), I set maximum\_instance\_count to 3. I also set the target value at the configuration of auto-scaling to 25 because I think it’s a suitable threshold to trigger the auto-scaling mechanism. If we choose a small number, then the auto-scaling is over triggered even if it’s not needed. If we choose a high number, then the auto-scaling may never be called. Scale-in and scale-out cool down values were both set to 30 seconds due to a balance of resource management, cost management and time acceptability.