**Sagemaker instance selection choice**

I selected Sagemaker instance ml.t3.large for the following reasons:

* Since this instance would only be running jupyter notebook code and some of the basic processing I decided to go with fairly descent CPU and Memory (vCPU=2 and Memory=8GiB).
* Model training is running on separate instance type so I don’t need to worry about making Sagemaker instance very compute intensive.
* S3 bucket is used for data storage so that alleviates large memory requirements.
* Model endpoints are running under different instance type.

**Justification for choosing type of EC2 instance**

I went with general purpose t2.xlarge EC2 instance type as it provides moderate CPU and Memory requirements sufficient to run data processing, basic operations and model training. With this configuration model training took close to 20mins which is fairly short duration given the amount of processing involved. Price per hour is less than $0.20 which is good cost tradeoff.

**Write about EC2 code and how it compares to the code from Step 1**

Listed below are key difference between EC2 code and Sagemaker instance code

* Several sagemaker modules and libraries are available in sagemaker notebook environment which are not available in EC2 instance.
* Importing sagemaker module provides sagemaker runtime environment under which notebook code is executed. It provides context and runtime environment for code execution which is not supported in EC2 code.
* With sagemaker libraries various object can be instantiated like Hyperparameter tuner, estimator, setting environment variables, debugger, profiler etc which facilities configuration with low code and enhanced capabilities. Same is not feasible in EC2 as it would entail writing lot of code and perform extensive environment setup.
* EC2 code is simple python environment with pytorch support. We can write pytorch script for model training and testing.
* Entire ML pipeline can be executed on Sagemaker which is not feasible in EC2 code.

**Write about Lambda function in final writeup.**

In Lambda function we instantiate runtime object and endpoint\_name. Runtime object is used to invoke the endpoint and accepts the format of event message such as ‘application/json’ referring to json format for request body content. Result returned from the endpoint is converted into JSON format before being send back in response body with status code of ‘200’ enabling cross domain calls with Access-Control-Allow-Origin set to ‘\*’. Also its important to note that modules are imported to support functions and methods used in lambda function.

**Results:**  "body": "[[0.5860685706138611, 0.581749677658081, 0.30365851521492004, 0.6547749638557434, 0.578667938709259, -0.06040169671177864, -0.1682649701833725, 0.5599482655525208, -0.004726633429527283, 0.26198041439056396, 0.44466689229011536, 0.6322798728942871, 0.23320285975933075, 0.6302286982536316, 0.6538736820220947, 0.503573477268219, 0.5196398496627808, 0.2090550810098648, 0.282077431678772, 0.5264465808868408, 0.5401625633239746, 0.24061383306980133, 0.5089805126190186, 0.6206284761428833, -0.016930846497416496, 0.008964188396930695, 0.6029733419418335, -0.25446122884750366, 0.7258913516998291, 0.2825552821159363, 0.504551887512207, 0.5072543025016785, 0.041132666170597076, 0.5149539709091187, 0.337560772895813, 0.6005417108535767, 0.3293423652648926, 0.39723503589630127, 0.6054755449295044, 0.3309067189693451, -0.06646368652582169, 0.6616803407669067, 0.26549050211906433, 0.6130195260047913, 0.24571208655834198, 0.6165071129798889, 0.31599143147468567, 0.4608200490474701, 0.2141287475824356, 0.20662344992160797, 0.38995176553726196, 0.27706003189086914, 0.2728506028652191, 0.438828706741333, 0.3287105858325958, 0.591151237487793, 0.6151048541069031, 0.3438055217266083, 0.24811337888240814, 0.47794631123542786, 0.11838075518608093, 0.3076281249523163, 0.34239906072616577, -0.02150164544582367, 0.09787628799676895, -0.02746940776705742, -0.05814646556973457, 0.44791263341903687, 0.2661702334880829, 0.28706124424934387, 0.4709480404853821, 0.2293863445520401, 0.0467015765607357, 0.3039526641368866, 0.17233839631080627, 0.561221718788147, 0.2366209775209427, 0.19137583673000336, 0.5240403413772583, 0.255370557308197, 0.49720728397369385, 0.4307907819747925, 0.12711751461029053, 0.2995118498802185, 0.02879352681338787, 0.30821502208709717, 0.4563232958316803, -0.045396991074085236, 0.3409106731414795, 0.6171738505363464, -0.14309659600257874, 0.1574305146932602, -0.07284145057201385, 0.14746177196502686, 0.29568302631378174, 0.12199152261018753, 0.4861227571964264, 0.24723970890045166, 0.2210666537284851, -0.09887337684631348, 0.36195996403694153, -0.21740089356899261, 0.29109564423561096, 0.04220682382583618, -0.10023950040340424, 0.24861504137516022, 0.24743123352527618, -0.29791533946990967, 0.19722631573677063, -0.05723617225885391, 0.23458585143089294, 0.34250569343566895, -0.03882203251123428, -0.004864443093538284, 0.5508145689964294, -0.1445448249578476, 0.257218599319458, 0.345347136259079, -0.1522929072380066, 0.017558464780449867, -0.18726523220539093, -0.08899061381816864, 0.1663469523191452, 0.3544248342514038, -0.018308008089661598, -0.15787075459957123, 0.19376394152641296, -0.1657421886920929, 0.167301744222641, 0.3652785122394562, -0.235820472240448, -0.24553430080413818, -0.20475737750530243]]"

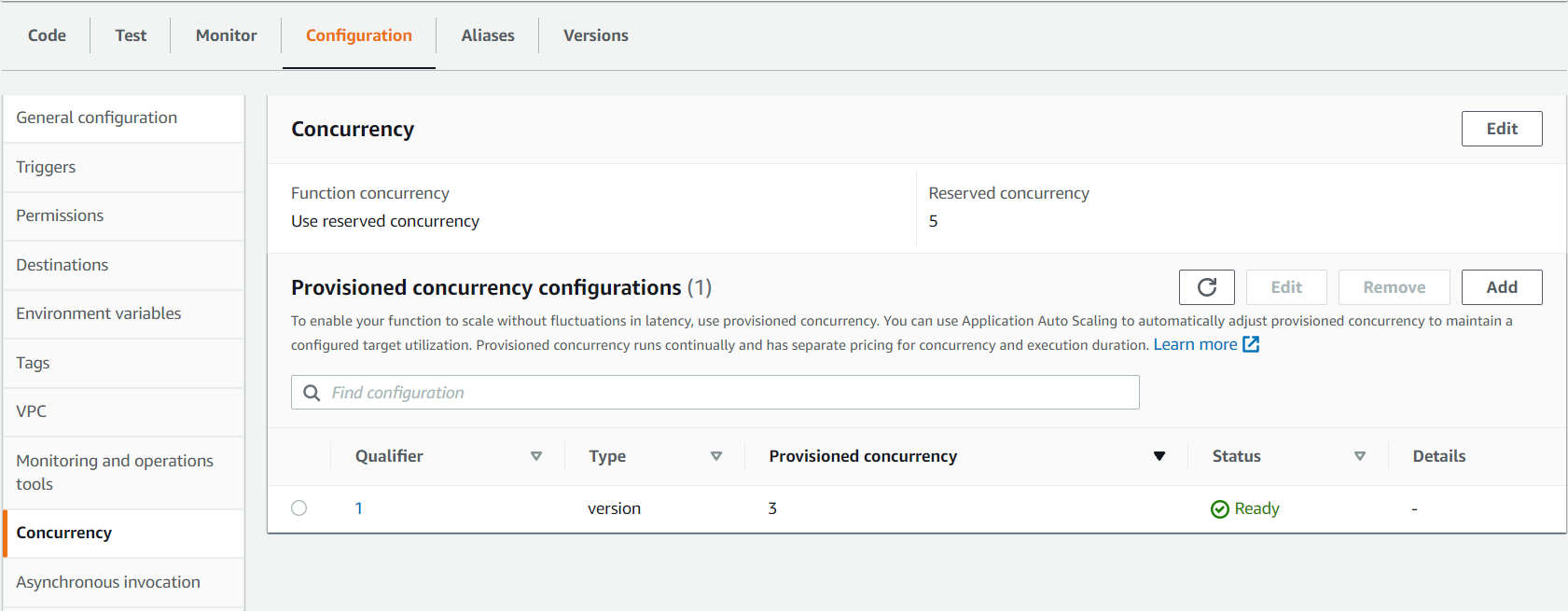
**Writeup about whether you think your AWS workspace is secure, and whether you think there are any vulnerabilities**

Observation from Security perspective:

* As we can see there are several AWS Service roles which are not used when running only Sagemaker service. Following the principal of least privilege access these roles should be removed if not used.
* We have enabled Sagemaker full access for Lambda function which can be further restricted to limited Sagemaker resources such as endpoint so that other sagemaker resources can be secured from lambda function execution.
* We have launched EC2 in default VPC and made it accessible to public. This is not good security practice. EC2 instance can be created inside private VPC and subnets and access to instance can be whitelisted from specific IP address.

**Configuration of Concurrency and Auto-Scaling**

In order to setup concurrency I had to first version the lambda function I created and then publish the version. Once the version is available, under concurrency section of configuration tab Reserved Concurrency is set. I decided to go with 5 Reserved concurrency as I do not expect heavy traffic or workload in my experimentation which necessitates more than 5 reserved lambda instance. At max I expect 3 lambda function to run concurrently so I set provisioned concurrency as 3. This allows at most 3 lambda function to run concurrently.



To setup Auto-Scaling I had to configure endpoint runtime settings for AllTraffic variant with min instance count of 1 and max instance of 3. I expect max 3 instance of endpoint to spin beyond which it would incur additional cost which I need to be sensitive about. At least I would be running 1 instance of endpoint. I kept Scale In and Scale out cool down at 40 sec meaning it will take 40 sec of elevated traffic to spin up more instances and 40 sec of low traffic to spin down endpoint instances. Its highly responsive configuration given the experimentation I ran in this project. Variant Invocation per instance I set to 40 meaning if 40 simultaneous requests are coming per instance then to respond to elevated traffic more instance will be created.

