**Operationalizing an AWS ML Project**

**Training and deployment**

The following image displays the SageMaker notebook instance I created for this project. Since we are using SageMaker SDK and will run all the training jobs and processes using ephemeral instances, I've selected an ml.t2.large with 2 CPUs and 8GB of RAM for this work with 15GB of the hard drive so that we have enough storage for the dataset.

Graphical user interface, application

Description automatically generated

The following image S3 bucket which contains the materials from this course.

Graphical user interface, application

Description automatically generated

The following image shows the deployed endpoint.

Graphical user interface, text, application, email, Teams

Description automatically generated

## EC2 Training

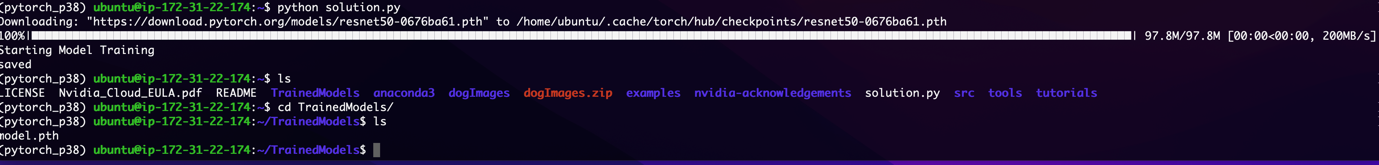
The following image shows running the "python solution.py" scrip using the "pytorch\_p38" environment. And in the last line, you can see the model artifact that is saved on the EC2 instance.

I used an "m5.4xlarge" instance with 16 CPUs and 64 GB of memory. I selected this instance so that the training can be done faster.

Some of the differences between the code ran on the EC2 (ec2train.py) and SageMaker notebook instance(hpo.py):

1. "hpo.py" accepts variables via "parser", while the variables are hardcoded in the "ec2.train1.py"

2. the data is in the root folder on the ec2



## Lambda function setup

The following image shows that the Lambda function that contains the "lambdafunction.py" was tested successfully and received 200 status and the expected prediction values. For this function to run correctly, I did:

1. updated the endpoint\_name to reflect the endpoint name that was deployed during the provided section

2. Add additional policies to the role that was generated to run the Lambda function “AmazonSageMakerFullAccess".

Graphical user interface, text, application, email

Description automatically generated

## Security and testing

## The following image shows the Lambda function policy and permission. As you can see, this role has two policies attached to it:

1. AWSLambdaBasicExecutionRole-45b61ad6-9381-4bbf-a48d-8c1c80e11b0e
2. AmazonSageMakerFullAccess

## The first policy is the basic lambda execution role with limited access. However, the 2nd policy gives this role full access to SageMaker which might not be a great idea since we only need evocation

## While checking roles and polices, I’ve found some old policies related to some old work that is not relevant anymore, so I’ve deleted:

## AnonCoreStack-EcsServiceRole-12M75AG4T5W95

## AnonCoreStack-ECSTaskRole-XJ5AQM9LGJQ

## AnonServiceCodeBuildServiceRole

## AnonServiceCodePipelineServiceRole

Graphical user interface, application

Description automatically generated

The following is a copy of what the Lambda function sent back after testing:

‘’’

Test Event Name

test1

Response

{

"statusCode": 200,

"headers": {

"Content-Type": "text/plain",

"Access-Control-Allow-Origin": "\*"

},

"type-result": "<class 'str'>",

"COntent-Type-In": "<\_\_main\_\_.LambdaContext object at 0x7fc78868fb50>",

"body": "[[-6.760165214538574, -4.508280277252197, -3.5048820972442627, -0.5850778818130493, -6.151078224182129, -5.1742682456970215, -3.093289613723755, -1.1336557865142822, -4.650386810302734, -0.6872179508209229, -2.3171727657318115, -4.739749908447266, -1.8672469854354858, 0.0889175534248352, -4.71138858795166, -2.1049914360046387, -6.296383380889893, -1.2045451402664185, -4.2254719734191895, 0.12927702069282532, -3.1316728591918945, -1.689181923866272, -5.3878302574157715, -4.033997058868408, -3.315932512283325, -4.975756645202637, -3.5796420574188232, -4.829275131225586, -6.147250652313232, -2.787091016769409, -4.579968452453613, -3.4315850734710693, -4.661843776702881, -0.8993250131607056, -6.761765480041504, -5.047989845275879, -8.072355270385742, -3.5069527626037598, -0.44169339537620544, -2.629704475402832, -2.822812557220459, -3.252828598022461, 0.6176385879516602, -3.7262532711029053, -1.6350525617599487, -8.310590744018555, -2.419365644454956, -1.8651039600372314, -3.347033739089966, -3.0976877212524414, -1.6311222314834595, -8.176608085632324, -5.15451717376709, -1.6498225927352905, -5.5006794929504395, -1.416898488998413, -5.129649639129639, -5.343907356262207, -2.730036497116089, -3.1172945499420166, -4.895002841949463, -8.71912956237793, -6.720644474029541, -8.380936622619629, -5.247697353363037, -7.060248851776123, -1.8281346559524536, -6.067103385925293, -3.5480382442474365, 0.2638809382915497, 0.8103920817375183, -6.537317276000977, -6.376285552978516, -4.664766788482666, -3.4472544193267822, -2.895609140396118, -8.044271469116211, -4.309315204620361, -5.380341053009033, -4.506032943725586, -0.7803418040275574, -6.961867332458496, 0.017107024788856506, -0.6097363233566284, -6.366474628448486, -5.132674217224121, -2.099656105041504, -5.492822647094727, -4.579404830932617, -1.0611975193023682, -9.18541145324707, -6.2298054695129395, -4.872824668884277, -7.2843098640441895, -4.915785312652588, -1.0162471532821655, -3.916842222213745, -2.7316884994506836, -6.535037994384766, -7.540546417236328, -8.384956359863281, -2.4749417304992676, -3.589874505996704, -2.7801408767700195, -5.8396806716918945, -5.553654670715332, -5.0006632804870605, -1.787074089050293, -3.163969039916992, -2.678823947906494, -3.297591209411621, -1.9122471809387207, -5.110894203186035, -8.221559524536133, -5.5576653480529785, -0.8410820364952087, -5.753725528717041, -1.3733538389205933, -7.605933666229248, 0.47207266092300415, -1.7703142166137695, -3.3525679111480713, -4.840109348297119, -4.068804740905762, -5.442234992980957, -4.170291423797607, -6.009312629699707, -0.241316556930542, -3.311040163040161, -6.291505336761475, -6.01483678817749, -0.5403186678886414, -4.455959796905518]]"

}

Function Logs

START RequestId: ca34ee07-09dc-4ad4-bce0-d942699c665b Version: $LATEST

Loading Lambda function

Context::: <\_\_main\_\_.LambdaContext object at 0x7fc78868fb50>

EventType:: <class 'dict'>

END RequestId: ca34ee07-09dc-4ad4-bce0-d942699c665b

REPORT RequestId: ca34ee07-09dc-4ad4-bce0-d942699c665b Duration: 1221.51 ms Billed Duration: 1222 ms Memory Size: 128 MB Max Memory Used: 64 MB Init Duration: 326.71 ms

Request ID

ca34ee07-09dc-4ad4-bce0-d942699c665b

‘’’

## Concurrency and auto-scaling

By default, the endpoint can accept more than one concurrent connection. At the same time, we needed to enable the concurrency on the Lambda function so that when/if we set up an API gateway, we would be able to connect to the endpoint with low latency.

We should be conscious that the cost must be monitored as we have Lambda function concurrency and endpoint autoscaling enabled.

The following image shows the auto-scaling setting on the SageMaker end-pint that was created for this project. In this example, I've set a max number of instance to 3 so that independent of how much traffic goes to the server the number of deployed instances will be not be increased to more than 3.

This endpoint is reasonably responsive as it only needs 50sec of both increased/ decreased invocation to spin up/down instances.

I've sat 30 for the "Target value", which means if the instances receive 30 simultaneous connections, it will increase the number of instances up to the max instance count, which is 3 in this example.

Graphical user interface, application

Description automatically generated

The following image depicts the concurrency setting for the lambda function I made for this project. This function has 50 reserved concurrencies with two provisioned concurrencies. The fixed cost for the two provisioned concurrencies is ~$2/month.

Graphical user interface, application

Description automatically generated