# Project for AWS MLE Course 4: Operationalizing an AWS ML Project

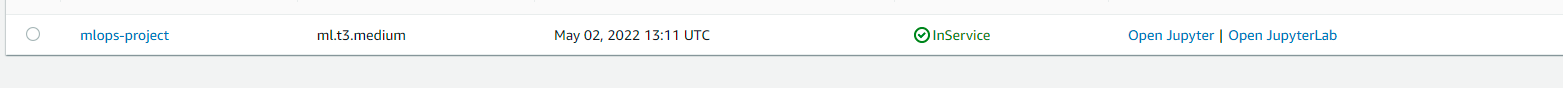
## Initial Setup

### Creating an AWS SageMaker Instance

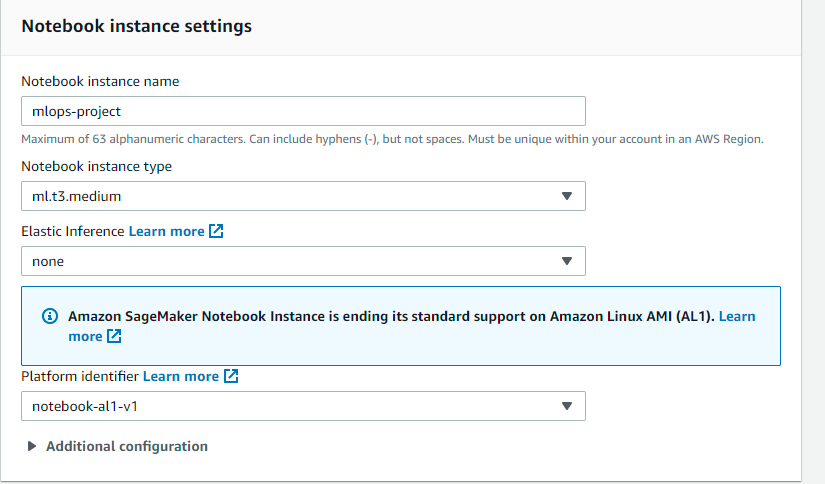
I selected the ml.t3.medium instance for my notebook instance because (1) it is AWS free-tier eligible; (2) I do not need incredible power in the notebook since all of my training will be completed on more powerful instances; and (3) it is the recommended default by AWS SageMaker team.

#### Screenshots

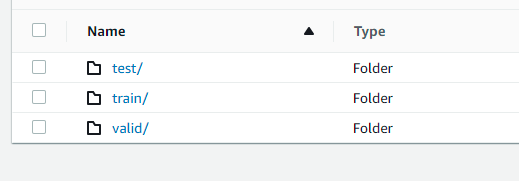
#. 1



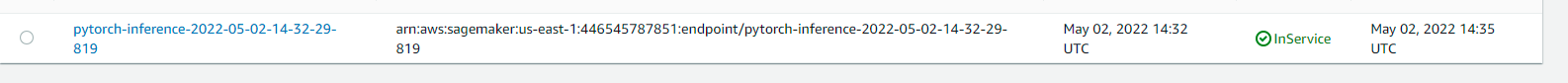
#. 2



### Download data to an S3 bucket screenshot



### Inference endpoint screenshot



# EC2 Setup

For this section, I chose a spot instance because we can acquire EC2 computing instances at an incredible discount (up to 90% lower than On-Demand pricing). I also chose the c5.xlarge because of its balance between being high performance and cost-effective. Amazon recommends it for high-performance computing. I want to optimize for speed and cost.

# EC2 code

The most glaring differences are number of epochs we used in the hpo.py script vs. the ec2train1.py which went from 20 to 5 epochs, respectively. We are also not passing arguments with argument parser (‘argparse’) since all our training code is contained in the ec2train1.py script.

### Screenshots

#1. 

#2.

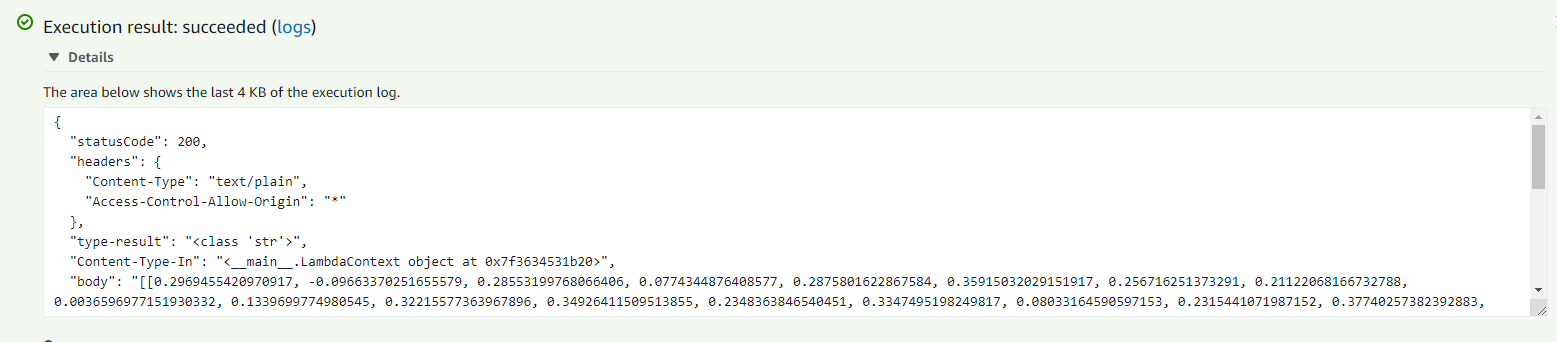


# Setting up a Lambda function

This lambda function is expected to be invoked by an event which has a header type of ‘application/json’. We use *json.dumps* to properly read and format the json object / event we receive. We invoke the endpoint with the *invoke\_endpoint* object of the *runtime* class. It gives us a response object which we return as plain text re – *text/plain* content-type if the return object header.

### Screenshots

#1. Successful lambda invocation

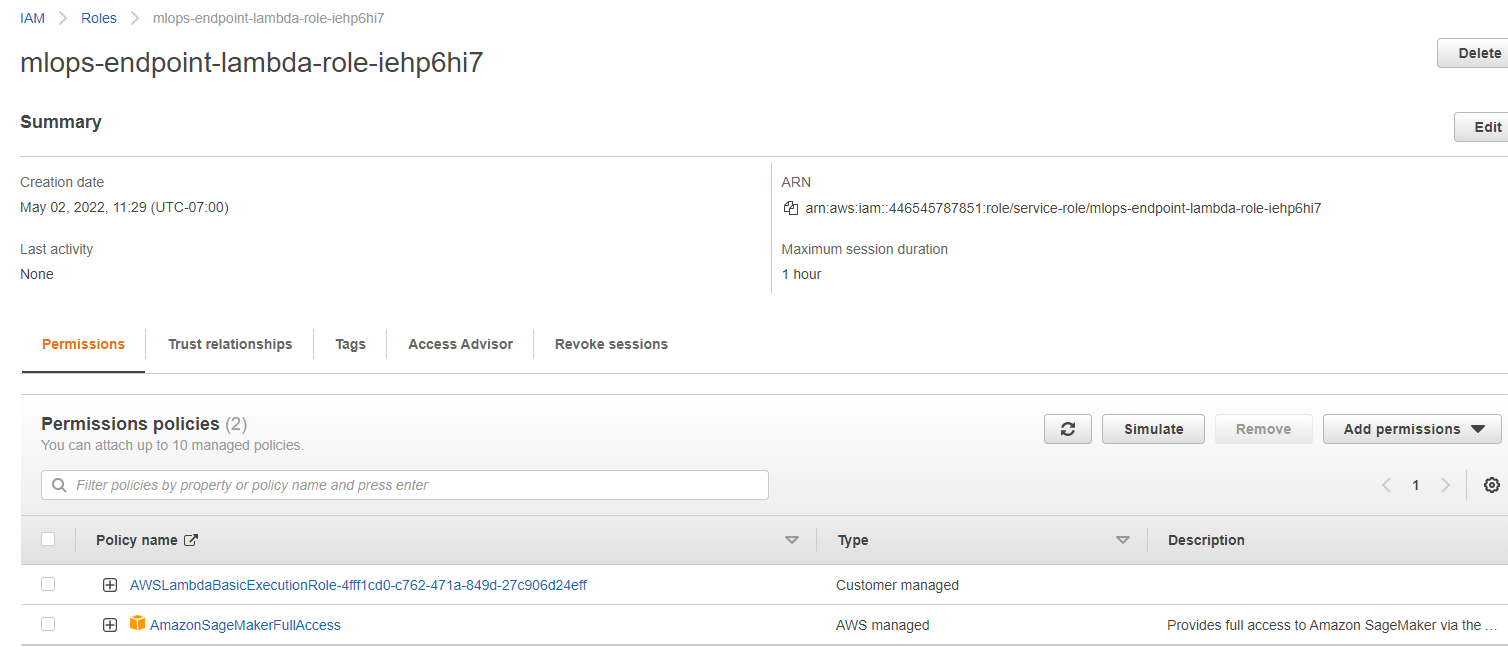


#2. Results list (33)

[[0.2969455420970917, -0.09663370251655579, 0.28553199768066406, 0.0774344876408577, 0.2875801622867584, 0.35915032029151917, 0.256716251373291, 0.21122068166732788, 0.0036596977151930332, 0.1339699774980545, 0.32215577363967896, 0.34926411509513855, 0.2348363846540451, 0.3347495198249817, 0.08033164590597153, 0.2315441071987152, 0.37740257382392883, 0.18352261185646057, 0.24217186868190765, -0.024665623903274536, 0.2604244649410248, 0.20952969789505005, 0.15247435867786407, 0.3404410481452942, -0.15007925033569336, 0.05873933434486389, 0.25048530101776123, 0.03797757625579834, 0.4434950649738312, 0.21291139721870422, 0.1752752959728241, -0.14106231927871704, 0.19625896215438843, 0.08831295371055603, 0.32017338275909424, 0.1898910105228424, 0.15351179242134094, 0.19981223344802856, 0.2972318232059479, -0.1182529479265213, -0.11551813036203384, -0.13372060656547546, -0.08215172588825226, 0.08100209385156631, -0.0071273744106292725, -0.0981057807803154, 0.21658554673194885, 0.2930316627025604, 0.1411442905664444, -0.014004018157720566, 0.31354162096977234, 0.23541590571403503, 0.19761553406715393, 0.36366933584213257, 0.03993688151240349, 0.34535807371139526, -0.22024092078208923, 0.25603044033050537, 0.09564554691314697, 0.0981941819190979, 0.45516660809516907, 0.11499208956956863, 0.19279129803180695, 0.048115137964487076, -0.031462252140045166, -0.013365799561142921, -0.04248575121164322, 0.22878527641296387, -0.06505841761827469, 0.03742501512169838, 0.38738706707954407, 0.05157548561692238, 0.11655306816101074, 0.0061577074229717255, 0.13972605764865875, 0.34583643078804016, 0.008916547521948814, 0.02967807464301586, 0.3188445270061493, 0.1369655430316925, 0.15091632306575775, 0.017677905037999153, 0.10748575627803802, 0.1792883425951004, 0.006831857841461897, 0.22043177485466003, 0.42183080315589905, 0.14110437035560608, 0.37211495637893677, 0.21601265668869019, 0.2541100084781647, 0.17072606086730957, -0.10661734640598297, 0.18127396702766418, 0.1243753656744957, -0.03751647472381592, 0.2633017897605896, 0.24332009255886078, 0.0729166716337204, 0.027394723147153854, 0.14930188655853271, -0.1300300657749176, 0.14610883593559265, 0.11689348518848419, -0.21259404718875885, 0.16726016998291016, 0.03384844958782196, -0.22306692600250244, -0.08091544359922409, -0.059152886271476746, 0.12819312512874603, 0.2590980529785156, 0.11229720711708069, -0.07477729767560959, 0.0903053879737854, -0.26825496554374695, 0.2247321456670761, -0.05164617300033569, -0.04535885155200958, 0.0911916121840477, -0.06009773910045624, -0.1294434666633606, 0.047251369804143906, 0.15292459726333618, 0.08458971977233887, -0.05868825688958168, -0.025660550221800804, -0.15540486574172974, 0.19160999357700348, 0.09076547622680664, -0.1156853586435318, -0.2228838950395584, -0.09707610309123993]]

#3. Other security considerations

This is too permissive for the lambda role with respect to its access to SageMaker. For better security, I would only allow it read access to invoke SageMaker endpoints.



# Concurrency and Autoscaling

I selected 5 reserved concurrency lambda functions and a provisioned concurrency of 2. This allows for instances needed to serve multiple concurrent requests to be ‘warmed up’, reducing cold starts, and reduces latency. Autoscaling will allow us scale the endpoint compute instances as well to serve multiple requests, in response to those concurrent lambda requests, and also reduce latency. For autoscaling, I set a minimum instance count of 1 and maximum count of 4. I also set the *SageMakerVariantInvocationsPerInstance* to 500 (meaning once an instance hits maximum invocations of 500 per minute, a scale out event should be triggered).

### Screenshots

