# **AWS**

## **Project 4: Operationalizing an AWS ML Project.**

### **Student: Lucia del Valle, Ledezma.**

**Step 1:** For he Jupyter Notebook instance I have selected the ml.t3.medium instance, because it is a cheaper instance with 0.05 USD per hour, and its performance is enough for this exercise having 2 CPU and 4 GiB of memory. For he hyperparameter tunning job ml.g4dn.xlarge has been selected it has 4 CPU and 16 GiB of memory, since I needed more computing power and speed in spite of being more expensive with a cost of 0.7364 USD hour. Finally for the training Jobs I selected the ml.m5.xlarge.

**Step 2.1:** For EC2 instance I have choosed the AWS Deep Learning AMI in order to use the necessary Python libraries as such Pytorch, numpy, and so. I built this machine by using the m5.2xlarge instance type, because it has a good relation cost – performace (0.384 USD hour for Linux), it has a very large memory (32 GiB) wich is going to be useful for training an image classification model by knowing how large dataset is, and it uses 8 CPU wich allows the execution time of the processes to be less.

**Step 2.2:** training a model by using EC2 instance , related to the training algorithm the ec2train1.py and hpo.py from the step one are very similar, but these have very important differences, ec2train1.py does not use the sagemaker library, and its way of execution is different too, in this case does not set enviroment variables and arguments by command line, and the model only uses the state\_dict() method to be saved and not model.cpu() as in hpo.py script.

**Step 3:** The lambda\_function.py allows us to invoke the endpoint and get an inference, this receive an url to an image as parameter, this function has the endpoint name setted in “endpoint\_Name” variable wich is invoked by the sagemaker-runtime client, several parameters are sent into invokation too, as ContentType, accept, and Body (it contains the url). Once the invokation is done, the response which has the inference is loaded and decoded. Finally the result is returned into the response body.

**Step 4.1:** Result of lambda function test:

Test Event Name  
testeventproject4

Response{ "statusCode": 200,  "headers": { "Content-Type": "text/plain",  
    "Access-Control-Allow-Origin": "\*"  },  
  "type-result": "<class 'str'>",  
  "COntent-Type-In": "LambdaContext([aws\_request\_id=ab056b97-c797-415f-968e-950fd506ee93,log\_group\_name=/aws/lambda/lambda-project4,log\_stream\_name=2023/03/31/[$LATEST]dac571d176b847ae913e05f05fb3bda2,function\_name=lambda-project4,memory\_limit\_in\_mb=128,function\_version=$LATEST,invoked\_function\_arn=arn:aws:lambda:us-east-1:359422136675:function:lambda-project4,client\_context=None,identity=CognitoIdentity([cognito\_identity\_id=None,cognito\_identity\_pool\_id=None])])",  
  "body": "[[0.882513165473938, 0.5167074203491211, 0.426849901676178, 0.5620660781860352, 0.6208546757698059, 0.8086071610450745, 0.4141761362552643, 0.6626482009887695, 0.13570047914981842, 0.76571124792099, 0.787838339805603, 0.07131706178188324, 0.4469018578529358, 0.11340431123971939, 0.8523727655410767, 0.6618640422821045, 0.6453067660331726, 0.4899321496486664, 0.5414660573005676, 0.05213979259133339, 0.7148356437683105, 0.5427155494689941, 0.6774144172668457, 0.7857545018196106, 0.2623352110385895, 0.04548359662294388, 0.881848156452179, 0.26364150643348694, 1.107208251953125, 0.5125839114189148, 0.08467399328947067, 0.1456524133682251, 0.3760055899620056, 0.7331759929656982, 0.6642904281616211, 0.9787653088569641, 0.43836459517478943, 0.46724846959114075, 0.858930766582489, 0.645396888256073, 0.971462607383728, 0.28441834449768066, 0.48656705021858215, 0.6726579666137695, 0.5861241817474365, 0.7359197735786438, 0.768077552318573, 0.4032851457595825, 0.6051168441772461, 0.3973546326160431, 0.859266996383667, 0.46674492955207825, 0.15698368847370148, 0.8693864345550537, 0.5840587615966797, 0.7362004518508911, 0.8679995536804199, 0.5159177780151367, 0.4057011008262634, 0.6450192332267761, 0.8971908688545227, 0.814225435256958, -0.05028507858514786, 0.20211076736450195, 0.35047656297683716, 0.16786134243011475, 0.15656089782714844, -0.22473837435245514, 0.41489464044570923, 0.3936682641506195, 0.5143868923187256, 0.5444900989532471, 0.4462874233722687, 0.3810587227344513, 0.4398932456970215, 0.403121680021286, 0.4278503954410553, 0.3757137954235077, 0.7093659043312073, 0.42822927236557007, 0.6912877559661865, 0.6368561387062073, 0.37409061193466187, 0.4320651888847351, 0.20254983007907867, 0.5922621488571167, 1.1343439817428589, 0.484945684671402, 0.42538169026374817, 0.7372361421585083, 0.5086299180984497, 0.39423930644989014, 0.11816835403442383, 0.3781088888645172, 0.4846985340118408, 0.4966146945953369, 0.6035683155059814, 0.4931066930294037, 0.2234441637992859, 0.2517850995063782, -0.1401878297328949, -0.23961390554904938, 0.5755254030227661, 0.4187173545360565, 0.10440994799137115, 0.612420380115509, 0.4535435736179352, -0.1549675464630127, 0.15625017881393433, -0.023106427863240242, 0.16944992542266846, 0.37478160858154297, 0.42415401339530945, 0.10444873571395874, 0.8762215971946716, -0.12775292992591858, 0.42574650049209595, 0.6854622960090637, 0.06403432041406631, 0.25307053327560425, -0.008782122284173965, 0.10054205358028412, 0.0749494656920433, 0.43878740072250366, 0.013737346976995468, -0.10068760067224503, 0.1838422417640686, -0.01405341923236847, 0.4534834325313568, 0.24830004572868347, 0.022848669439554214, -0.1988856941461563, 0.1215156689286232]]"  
}  
Function Logs  
Loading Lambda function  
START RequestId: ab056b97-c797-415f-968e-950fd506ee93 Version: $LATEST  
Context::: LambdaContext([aws\_request\_id=ab056b97-c797-415f-968e-950fd506ee93,log\_group\_name=/aws/lambda/lambda-project4,log\_stream\_name=2023/03/31/[$LATEST]dac571d176b847ae913e05f05fb3bda2,function\_name=lambda-project4,memory\_limit\_in\_mb=128,function\_version=$LATEST,invoked\_function\_arn=arn:aws:lambda:us-east-1:359422136675:function:lambda-project4,client\_context=None,identity=CognitoIdentity([cognito\_identity\_id=None,cognito\_identity\_pool\_id=None])])  
EventType:: <class 'dict'>  
END RequestId: ab056b97-c797-415f-968e-950fd506ee93  
REPORT RequestId: ab056b97-c797-415f-968e-950fd506ee93 Duration: 970.72 ms Billed Duration: 971 ms Memory Size: 128 MB Max Memory Used: 68 MB Init Duration: 332.47 ms  
Request ID  
ab056b97-c797-415f-968e-950fd506ee93

**Step 4.2:** Related to AWS security, I think AWS allows us the control of the Access to every service, thanks the IAM roles we can control giving permitions only when is needed. About vulnerabities, I think is very important to check the role when we are creating a new object, like a notebook or a S3 bucket and do not allow a fullaccess to every role. In the case of my lambda function, it uses a rol with sagemaker fullaccess, this can be dangereous because can delete or damage some important object into sagemaker.

**Step 5:** Related to lambda concurrency, I have setted the reserved concurrency, allowing until 5, so 5 lambda functions is going to be available if the traffic increase. I choose this option because is the cheaper one and I do not spect too many requests. For the endpoint I selected a range of 1- 5 instances, with scale-out cool down of 60 seconds and scale-in cool down of 180, so aws will wait only one minute before deleting the extra endpoints, and will wait 3 minutes before deploying the new ones; every choice has been done in order to avoid incurring extra costs.