CHOICE OF EC2 INSTANCE

I used a c5.xlarge (computing) with spot instance (not obtained from the first time, I had to try 2 or 3 times)

p3.xlarge -> No spot instance (error : "Max spot instance”)

I tried to get instance names which are similar to Sagemaker training instances (ml.c5.xlarge, ml.p3.xlarge...)

I was not interested in instances with memory (r5.large), so I choose a compute one.

DIFFERENCES BETWEEN THE IPYNB FILE AND THE EC2 PY FILE

1. All the dependencies (pytorch, pytorchvision) already exist in the DL image (notebook), called indirectly with an instantiation of the PyTorch class (imported in the module sagemaker.pytorch).

Whereas for the EC2 instance, I had to install with pip the modules torch and torchvision, even if I choose an EC2 Deep Learning AMI and then activated the virtual environment amazon\_pytorch\_37.

1. For the notebook, we recover environment variables ('SM\_CHANNEL\_TRAINING', 'SM\_MODEL\_DIR', 'SM\_OUTPUT\_DATA\_DIR') which indicates us where to get the data and where to store artifacts. There is no need to do that for the ec2 instance as all is in the virtual machine locally (/dogImages to get data, and /TrainingModels for artifacts).
2. The notebook calls the hpo.py to get the model (entry\_point argument of the PyTorch class). It passes indirectly arguments to the python program which gets them with an ArgumentParser. This working is more or less hidden to the user who just have to recover the arguments.

Whereas for the EC2 program, all is in the same file, and parameters are in our case hard-coded.

LAMBDA FUNCTION:

* For debugging and logging: imports of logging module, creation of a logger, with level DEBUG
* Creation of a runtime client (the 'sagemaker-runtime' of boto3) which invoke the endpoint created in step 1 by its name (The name is recovered in Sagemaker>Inference>Endpoints).

This client gets a response in json format, after the content (path file of the image dog) has been sent in json format too (json.dumps(bs))

* Then, we get the field 'Body' of the response and transform Bytes in 'utf-8'
* If no errors is raised by the function, all is fine and the response of the invocation is put in the HTML response, with a status code of 200.

SECURITY RISKS

1. Multiplication of roles without a clear policy of management is risky, as we can have a lack of control: are we sure this role is still useful?

- If we delete it, will it be OK? Everything is still going to work?

- If we keep it, it is potentially an entry point for hackers. And how to manage thousands of roles which have been accumulated over time?

Resource policy maybe a complementary solution.

2) Protection of data at rest (S3) and transit (TLS certificates, end users) maybe another solution.

3) If we use EC2, we can use Inspector to detect threats in the instance.

4) Mechanisms of authentication (Cognito + API Gateway) in front of lambda can reduce the surface of attacks and select persons able to ask a service.

PROVISIONNED CONCURRENCY

If I would have to put the lambda function in production, and would wait for a panel of 100 persons per day, perhaps I will adapt with a concurrency between 5 and 10.

For this exercise, I just put 2.

AUTOSCALING

For the cooldown periods (scale-in and scale-out), I let 5 mins (300 s): not too brutal behaviors but reactive ones. So, I let the system to adapt without having to pay for too much instances retained (during scale-in) or created nervously (during scale-out).

For the target metric (number of invocations per instance – creating or deleting one), I put 100: it seems realistic, but I have no big experience with putting ML systems in production, and the magnitude of the volumetrics.

I noticed the response of the invocation endpoint was nearly immediate, so we could surely wait before adding instances too fastly.

Things may be different if the latency of the invocation was perhaps 500 ms (which may be important). But perhaps, this would not be the main bottleneck.