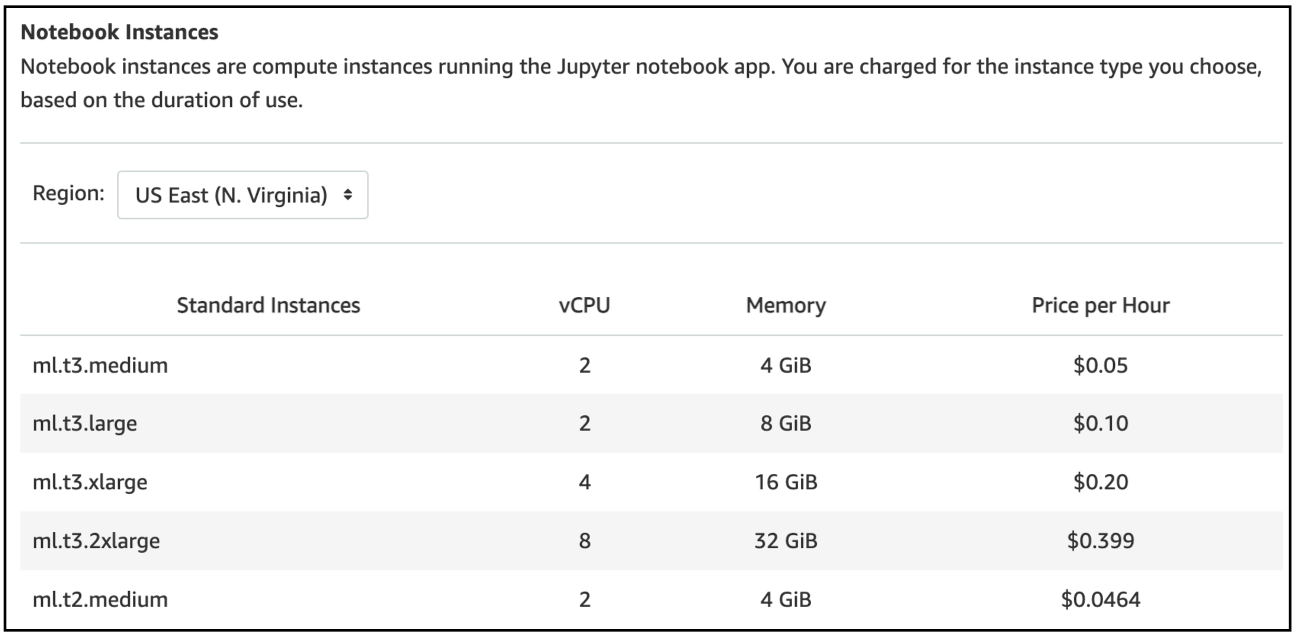
Project 4: Operationalizing an AWS ML Project

Sagemaker instance selection

For the sagemaker instance, we use the ml.t2.medium. The AWS EC2 instances with the ml prefix are specific EC2 instance classes for use in AWS Sagemaker. At the moment of writing this is the cheapest available instance in the N. Virginia region, and it suffices for this exercise as from the notebook we are just launching training jobs, not doing training itself. It has a cost of $0.0464/hour, and comes with 2 vCPU cores, 4GB of memory, and no GPU. This is enough for basic data processing, feature engineering, and EDA that uses small datasets held within 4 GB memory.



Notebook Instances

Notebook instances are compute instances running the Jupyter notebook app. You are charged for the instance type you choose, based on the duration of use.

Region: US East (N. Virginia)

Standard Instances	vCPU	Memory	Price per Hour
ml.t3.medium	2	4 GiB	$0.05
ml.t3.large	2	8 GiB	$0.10
ml.t3.xlarge	4	16 GiB	$0.20
ml.t3.2xlarge	8	32 GiB	$0.399
ml.t2.medium	2	4 GiB	$0.0464

Image taken from [1]

Additionally, the T family of instances are burstable [2], which means that it provides baseline CPU performance and the ability to burst CPU usage when needed. This makes it a good choice for workloads that have variable CPU requirements, such as notebooks, which may require more CPU power when running certain code cells.

EC2 instance selection

For the EC2 instance we select the t3.medium instance. T3 instances are a type of general-purpose instance that is designed to be low cost and burstable. They provide a baseline level of CPU performance, but can also burst CPU usage when needed for as long as required. These instances are particularly useful for applications with moderate CPU usage that experience temporary spikes in use. In addition, T3 instances offer a good balance of compute, memory, and network resources, making them a cost-effective way to run a wide range of general-purpose workloads [3]. The t3.medium instance has a cost of $0.0416, with 2 vCPU cores, 4GB of Memory, and up to 5 Gigabit of network performance [4]. This should be more than enough for the training script provided as it does not require GPU, and it uses iterates over batches of size 2.

For this instance, we use the Amazon Machine Iimage Deep Learning AMI (Amazon Linux 2) Version 68

Training a Pytorch model inside the EC2 instance

Inside the EC2 instance we run the code contained in ec2train.py. To do so we connect to the instance and run:

```
wget https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/dogImages.zip
unzip dogImages.zip

mkdir TrainedModels

PATH=/home/ec2-user/anaconda3/bin:$PATH
source activate pytorch_p39
python ec2train.py
```

The script ec2train.py does not receive any arguments, nor reads any environment variable. Instead, all parameters are specified inside the code. The script reads directly the dog images from the directory dogImages and saves the images in the directory TrainedModels.

Using a Lambda Function to invoke a Sagemaker endpoint

The code contained in lambdafunction.py is used to invoke a Sagemaker endpoint though a Lambda function. This code, uses a Boto3 session with a SagemakerRuntimeClient, and uses the invoque_endpoint method to call the deployed endpoint, passing the image information received in the request body. Sequentially, the response from the Sagemaker endpoint is processed and an array with the scores for each class is returned in the body of the response.
To call the Lambda function we can use:

```
{
  "url": “https://s3.amazonaws.com/<image-location>/<image-name>”
}
```

Security vulnerabilities in the AWS workspace

One of the security vulnerabilities that we have is the lack of a VPC for the project. Another, is the fact that we are granting the Lambda function a role that a "FullAccess" policies to Sagemaker. Also, outdated or inactive roles within a project can create security risks as they may be associated with individuals who are no longer actively involved in the project

Concurrency in the Lambda Function and Auto-Scaling in the Sagemaker endpoint

Concurrency and Auto-Scaling ensure we can handle high volumes of traffic without affecting performance.

For concurrency, we can use a combination of reserved concurrency and provisioned concurrency. Reserved concurrency allows to reserve a certain number of concurrent executions for a specific function. This ensures that a minimum number of resources are available at all times, even during periods of high traffic. Provisioned concurrency, on the other hand, allows you to pre-warm the function by setting a specific number of concurrent executions to maintain. This ensures that the function is always ready to handle incoming requests without incurring any cold-start penalties.

Additionally, it is also important to consider auto-scaling to ensure that the endpoint can handle sudden spikes in traffic without affecting performance.

In real life situations, if the number of user requests is unknown, concurrency and auto-scaling should be planned for the worst-case scenario. It is also important to constantly monitor the traffic and adjust accordingly the original configurations chosen.

For this project, the expected traffic to the Lambda function is not known in advance. As this a pet-project, in the Lambda function I chose to use 6 instances of reserved concurrency, 3 instanced of provisioned one (these are more expensive; in the Sagemaker endpoint I chose to auto-scale up to 6 instances.

References

[1] Amazon SageMaker Pricing. https://aws.amazon.com/sagemaker/pricing/?nc1=h_ls
[2] Ensure efficient compute resources on Amazon SageMaker. https://aws.amazon.com/blogs/machine-learning/ensure-efficient-compute-resources-on-amazon-sagemaker/
[3] Amazon EC2 T3 Instances. https://aws.amazon.com/ec2/instance-types/t3/
[4] Amazon EC2 On-Demand Pricing. https://aws.amazon.com/ec2/pricing/on-demand/
[5] Reduce ML inference costs on Amazon SageMaker with hardware and software acceleration. https://aws.amazon.com/blogs/machine-learning/reduce-ml-inference-costs-on-amazon-sagemaker-with-hardware-and-software-acceleration/