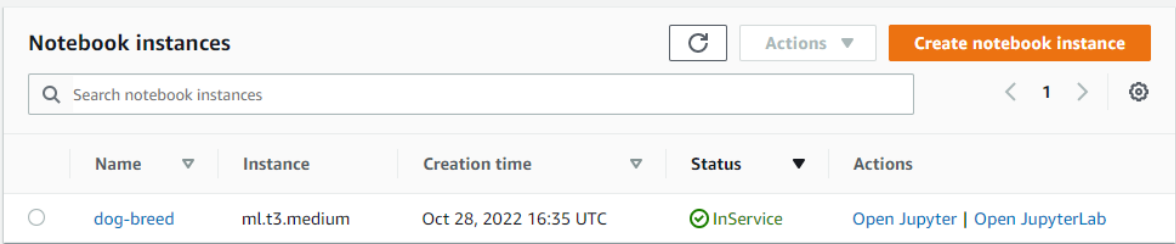


Operationalizing Machine Learning on SageMaker

Training & Deployment:

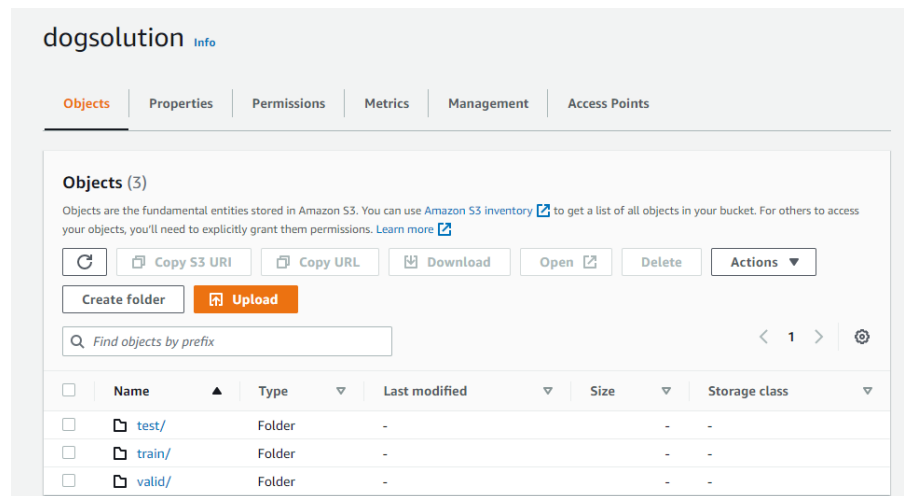
- Every notebook in SageMaker needs a computing instance in order to run and we use the notebook instance to create and manage Jupyter notebooks for preprocessing data and to train and deploy machine learning models.
- There are many types of instances, and every time we run an instance, we will need to make a good choice for the type of instance to run.
- The instance type we choose needs to be sufficient for your computing needs, but it should also minimize costs.
- But We will need to keep this notebook instance in “in Service” status for a long time while we are working on the project So, avoid high costs we should select a notebook that is low in per hour cost and also offers reasonably good CPU and RAM for our training process.
- So, I chose the “ml.t3.medium” instance type for Notebook instance which fulfills the previous two requirements



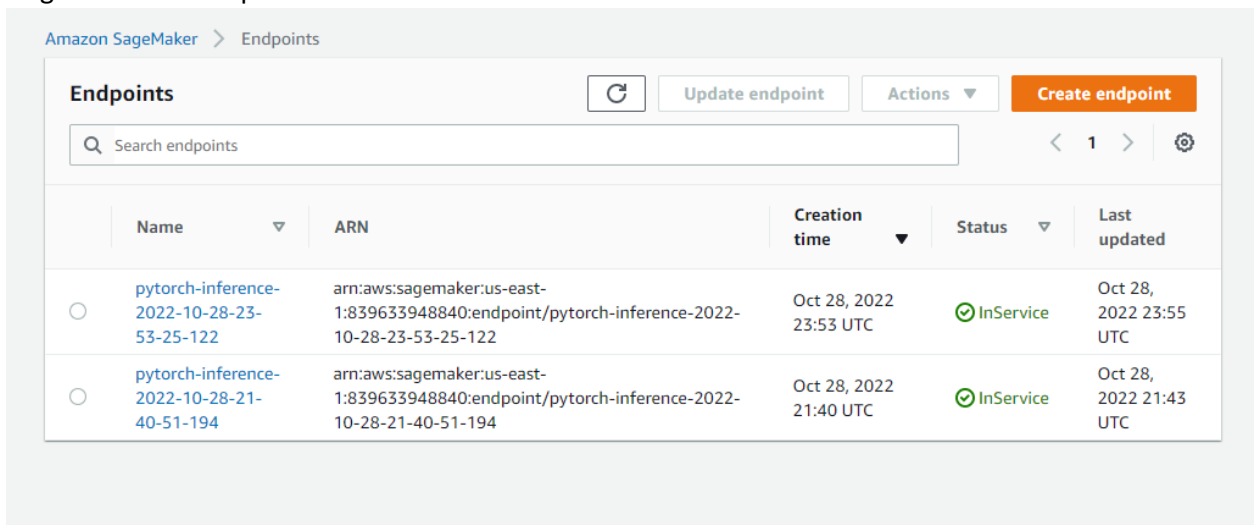
The screenshot displays the Amazon SageMaker Notebook instances management console. At the top, there is a breadcrumb trail 'Amazon SageMaker > Notebook instances'. Below this, a header bar contains the title 'Notebook instances', a refresh button, an 'Actions' dropdown menu, and a prominent orange button labeled 'Create notebook instance'. A search bar with the placeholder text 'Search notebook instances' is positioned below the header. The main content area features a table with the following columns: Name, Instance, Creation time, Status, and Actions. A single instance is listed with the name 'dog-breed', instance type 'ml.t3.medium', creation time 'Oct 28, 2022 16:35 UTC', and status 'InService' (indicated by a green checkmark icon). The Actions column for this instance provides links to 'Open Jupyter' and 'Open JupyterLab'.

Name	Instance	Creation time	Status	Actions
dog-breed	ml.t3.medium	Oct 28, 2022 16:35 UTC	InService	Open Jupyter Open JupyterLab

- I successfully uploaded the dog breed dataset to a dedicated S3 to be used in our training job



- At hyperparameter Tuning I used
 - Instance type= ml.m5.2xlarge
 - Max jobs = 3
 - Max parallel jobs = 3
- Multi-instance Training
- training data on multiple machines is very hard to implement and requires a lot of intricate configurations to get different machines to work together.
- One of the great things about AWS and SageMaker is that they make multi-instance training much easier.
- Multi-instance training in a term for fitting is machine learning models using multiple separate computers or servers
- You can see the two deployed endpoints from the notebook either multi-instance endpoint or single-instance endpoint



EC2:

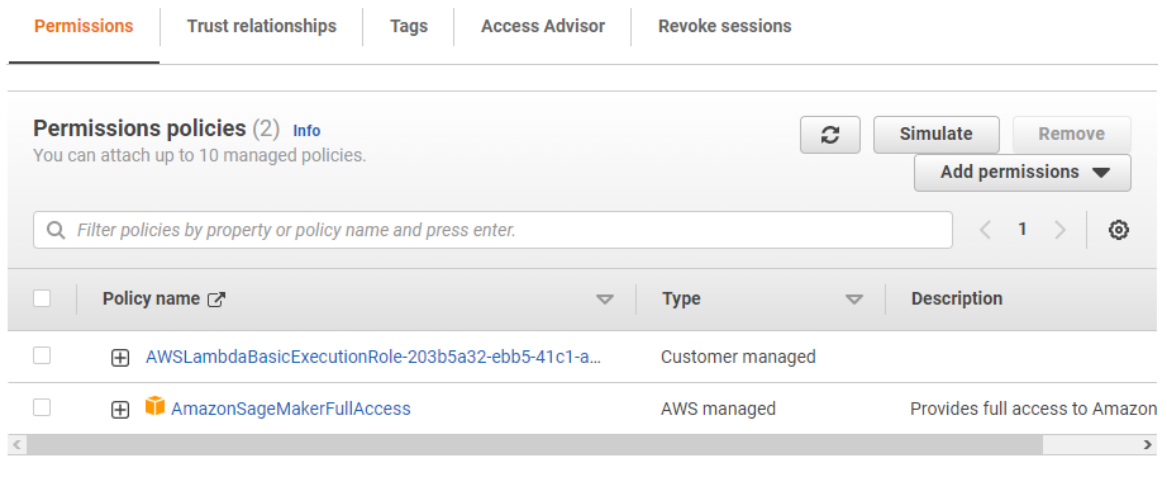
- I used a type of EC2 instance called a Spot Instance to minimize costs.

- ```
[root@ip-172-31-17-78 ~]# pip install contextlib2 typing scandir pathlib2 importlib-resources tqdm
Downloading pathlib2-2.3.7.post1-py2.py3-none-any.whl (18 kB)
Collecting contextlib2; python_version < "3"
 Downloading contextlib2-0.6.0.post1-py2.py3-none-any.whl (9.8 kB)
Collecting typing; python_version < "3.5"
 Downloading typing-3.10.0.0-py2-none-any.whl (26 kB)
Requirement already satisfied: six in /usr/lib/python2.7/site-packages (from singledispatch; python_version < "3.4"-->importlib-resources; python_version < "3.7"-->tqdm) (1.9.0)
Collecting scandir; python_version < "3.5"
 Downloading scandir-1.10.0.tar.gz (33 kB)
Using legacy 'setup.py install' for scandir, since package 'wheel' is not installed.
Installing collected packages: contextlib2, zipp, singledispatch, typing, scandir, pathlib2, importlib-resources, tqdm
Running setup.py install for scandir ... done
Successfully installed contextlib2-0.6.0 post1 importlib-resources-3.3.1 pathlib2-2.3.7.post1 scandir-1.10.0 singledispatch-3.7.0 tqdm-4.64.1 typing-3.10.0.0 zipp-1.2.0
[root@ip-172-31-17-78 ~]# python solution.py
Downloading: "https://download.pytorch.org/models/resnet50-19c8e357.pth" to /root/.cache/torch/checkpoints/resnet50-19c8e357.pth
100% |██| 97.8M/97.8M [00:00<00:00, 121MB/s]
Starting Model Training
saved
[root@ip-172-31-17-78 ~]# cd TrainedModels
[root@ip-172-31-17-78 TrainedModels]# ls
model.pth
[root@ip-172-31-17-78 TrainedModels]#
```

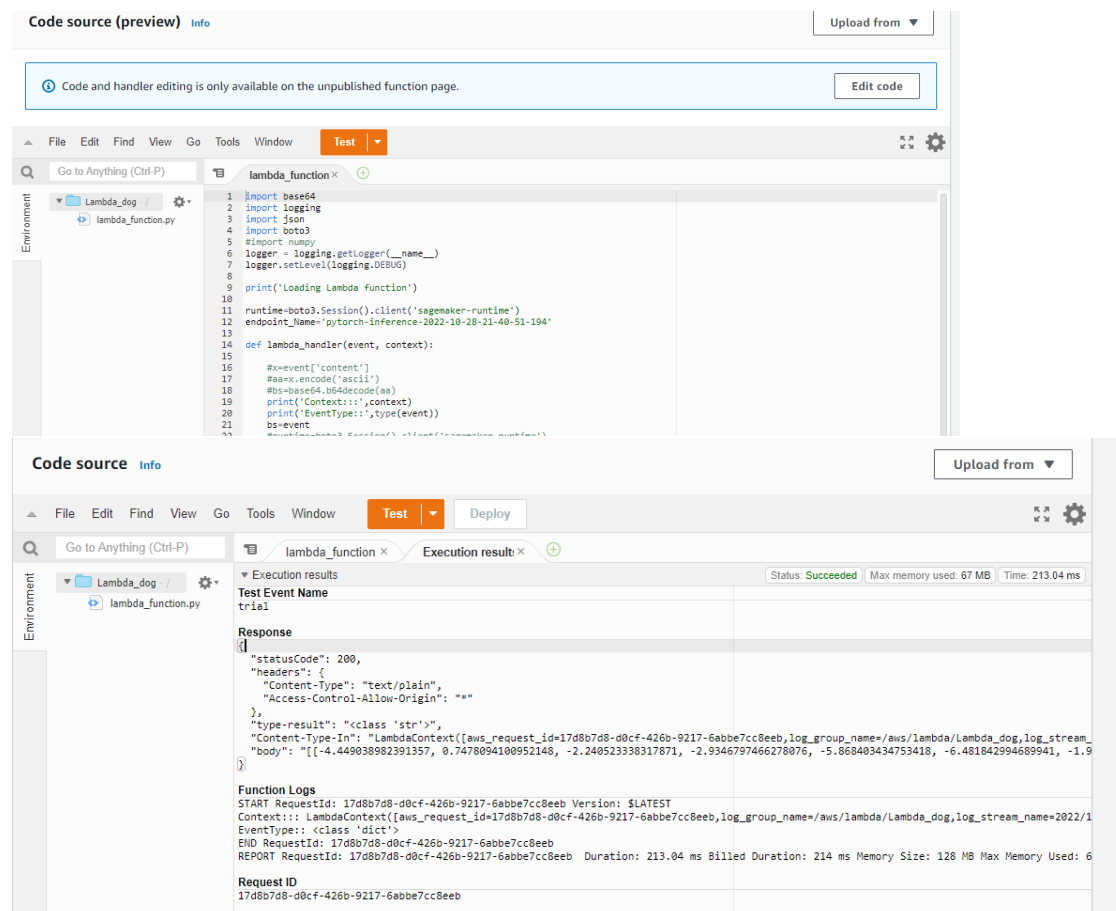
- ## Lambda Function:

- By default, Lambda functions are not given permission to invoke your SageMaker endpoints, And In order to allow the Lambda function to invoke my endpoint, I need to make adjustments to the IAM settings.
- When I test my Lambda function it showed an error that this lambda role is not authorized to invoke the endpoint so I attached new policy to the role which is called AmazonSageMakerFullAccess which

allows access to all sagemaker resources/endpoints which is what we're trying to invoke:



After attaching the policies to the Lambda function, and tested it successfully it resulted in a response list of 33 numbers, each of which represents a prediction about the class membership of the picture submitted:



- In result, Lambda function is delivering model predictions without any security. But that isn't considered dangerous because AWS workspace is secure and by using Amazon Workspaces, we have no data leakage and no copying or downloading of files and data.
- Also, AWS has improved flexibility, scalability, and reliability on the Workspaces as well as we can securely access various data resources and we can control who we give access to and what tools and level of access each user gets.
- We have multiple proven benefits that the AWS Workspaces technology provides such as flexibility, scalability and secure access.

### Concurrency & Auto Scaling:

- Concurrency refers to the ability of a Lambda function to serve multiple requests simultaneously.
- I set up concurrency for my Lambda functions to allow our Lambda function to be able to access minimum of three instances to reply to multiple requests simultaneously.
- I could only access settings to setup concurrency is called provisioned concurrency.
- provisioned concurrency is expensive and it creates instances that are always on and can reply to all traffic without requiring a wait for startup times.
- Choosing a high number for provisioned concurrency is suitable for very high-traffic projects, because it will turn on instances that can be used by your Lambda function any time.

Autoscaling for endpoints allows my deployed endpoints to respond to multiple requests at the same time.

- So, I Choose my maximum instance count to be = 3 My endpoint will be able to automatically scale from the minimum instance count I set up to the maximum instance count I select.
- Then I Choose what's called a scale-in cool down time period to be: 30
  - The scale-in period is the amount of time AWS will wait before deploying more instances for your endpoint.
  - If I choose a high number, then AWS will wait a longer time before deploying more instances.
  - And, this helps me avoid incurring more costs for momentary spikes in traffic.
  - If I choose a low number, then AWS will deploy instances more quickly, but this responsiveness will be more costly.
- Choose a scale-out cool down time period also to be: 30
  - The scale-out cool down period is the amount of time AWS will wait before deleting extra deployed instances.
  - If I choose a low number, then AWS will wait only a short time before deleting extra deployed instances.
  - And, this helps you avoid incurring costs for momentary spikes in traffic.
  - However, If I choose a high number, then AWS will keep extra instances deployed longer, but this extra capacity will be more costly.