The completed ML project

Project requirements:

- In Step 1, you'll need to download data to an S3 bucket and run your notebooks in a Sagemaker instance.
- In Step 2, you'll need to run your notebooks using EC2 in your workspace.
- In Step 3, you'll need to use the Lambda section of your workspace.
- In Step 4, you will use the IAM functions in your workspace to set up security.
- Finally, in Step 5, you'll set up concurrency for your Lambda function and auto-scaling for your deployed endpoint in Sagemaker.

What I submit

- A writeup document named ***wrireup.pdf*** that describes the decisions I make during the project and justifications for them.
- A file ***train_and_deploy_solution.html*** is easy to view the running results from the notebook train_and_deploy_solution.ipynb
- A directory of ***code files*** for my solutions (notebooks and Python files):
  1. Directory ***code*** contains:
     a) train_and_deploy_solution.ipynb
     b) hpo.py
     c) interfernce2.py
     d) lambdafunction.py
     e) solution.py
- A directory of screenshots contains:
  s3_bucket.png, notebook_instance.png, jupyter_lab_nb_instance.png, tunning_job.png, training_job.png, endpoint.png, train_multi_inst.png, endpoint_multi_inst.png, endpoint_multi_inst.png, g4dn_not_eligible.png, EC2_instance, 33_hours_ec2.png, training_ec2.png, IAM_lambda_role.png

- A compressed tar file ***project4.tgz*** contains the working stuffs in EC2:
  1. ***solution.py,***
  2. ***dogImages.zip*** the directory ***dogImages*** of 4 classes data.
  3. The directory ***TrainedModels*** contains the file ***model.pth.***

- All submitted documents are compressed into one file ***project4_thiendoan.zip.***

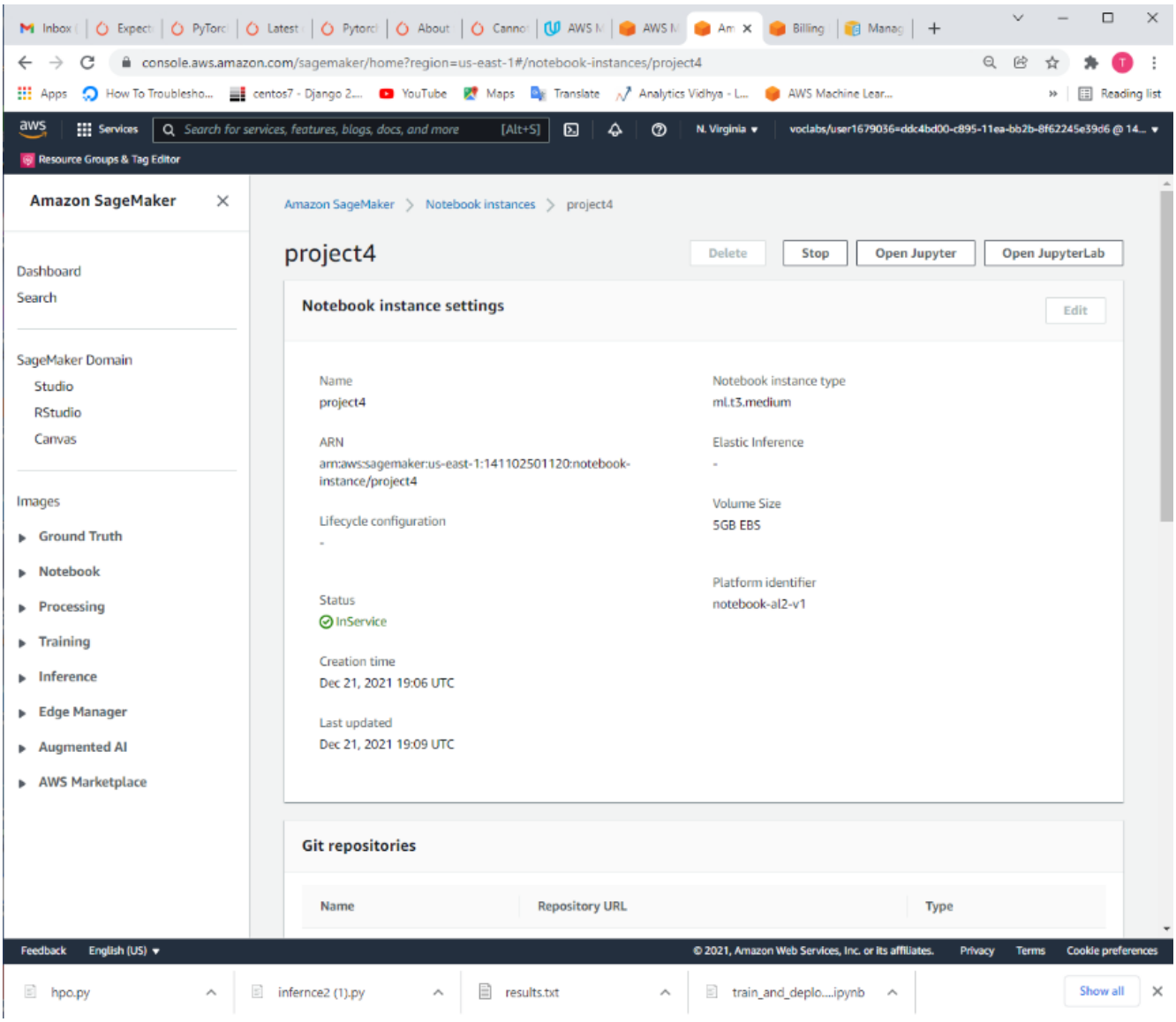
- I used many documents from Learning PyTorch with Examples — PyTorch Tutorials 1.10.1+cu102 documentation. to create my code.

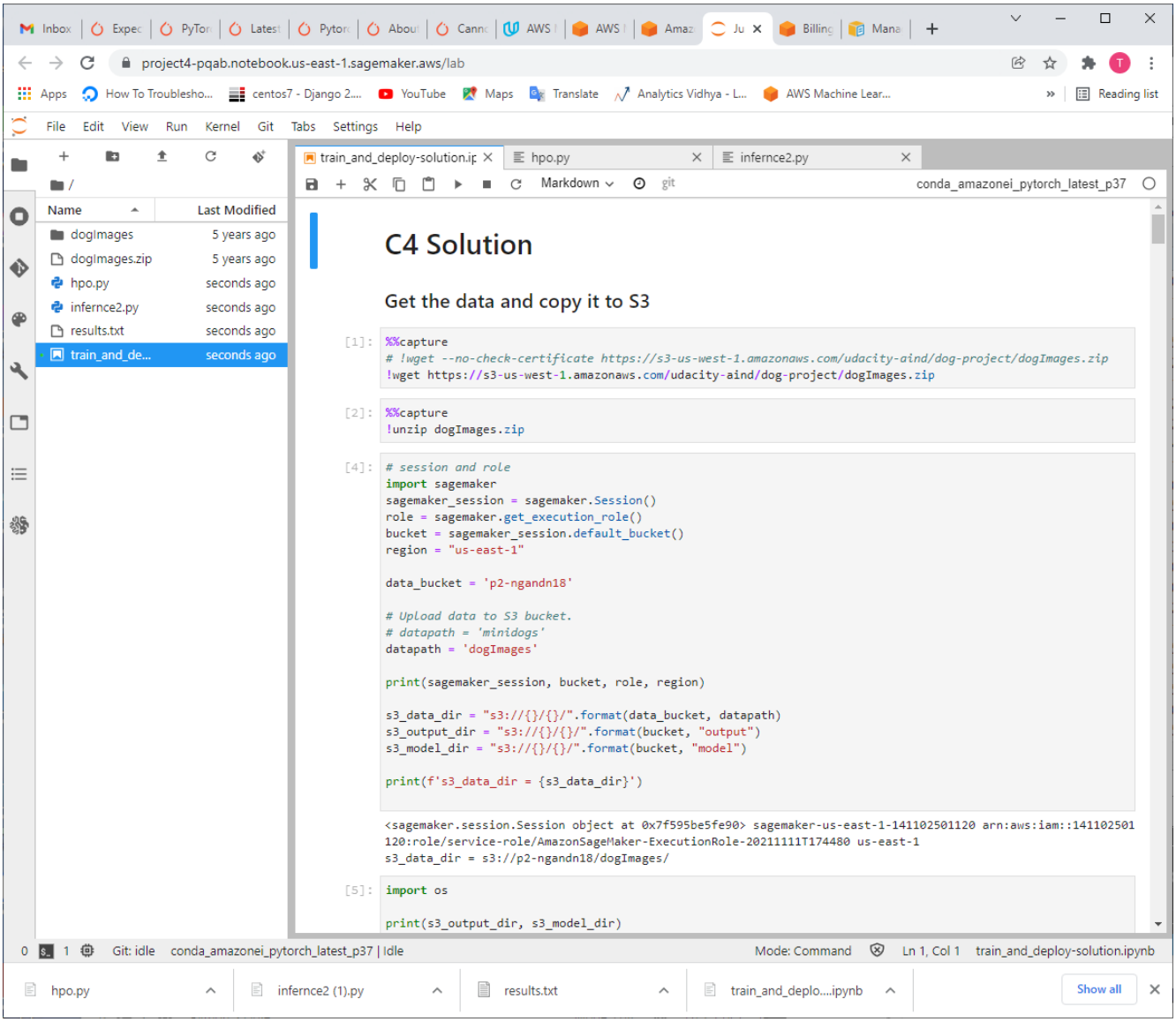
I. Step 1: Initial setup, training and deployment

Initial Setup

I open a notebook instance ***ml.t3.medium***. I choose this instance because the cost is suitable for my project. The first time I choose ***ml.t2.medium*** for low cost, but frequently I get trouble with it (kernel dies unexpected) then I change to ml.t3.medium and everything is stable for my project. Because this project using Py Torch, I think the more powerful instance is better. The type of instance is ***conda_amazonei_pytorch_latest_p37***.

I use Jupyter Lab for working space, because it is easy to see the multi-tab interface than Jupyter, and it is familiar to me. The notebook instance name is ***project4***.





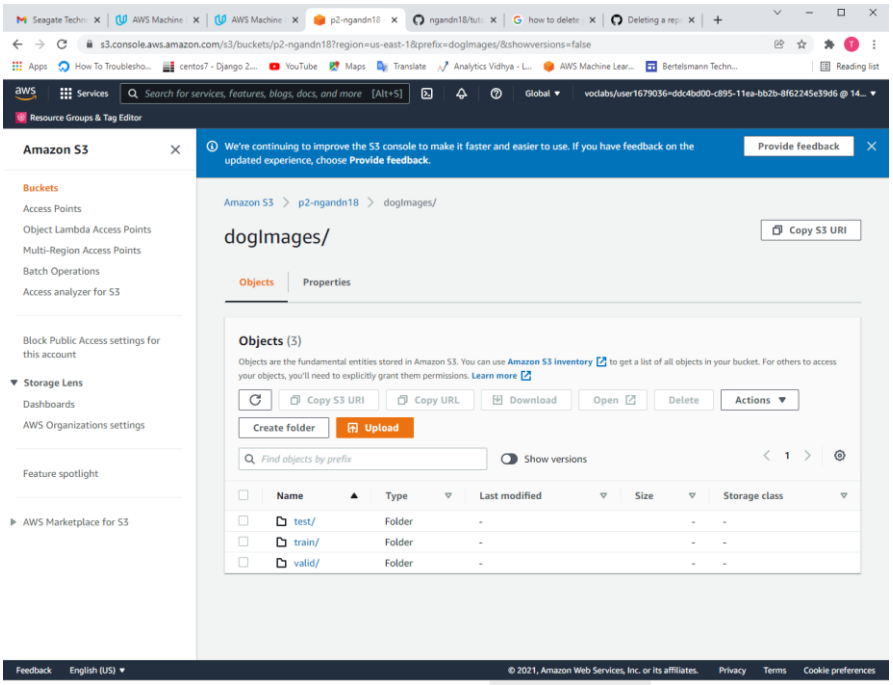
A screenshot showing a created notebook instance on Sagemaker.

Download data to an S3 bucket

Finish downloading data to my S3 bucket:

s3://p2-ngandn18/dogImages contains three directories:

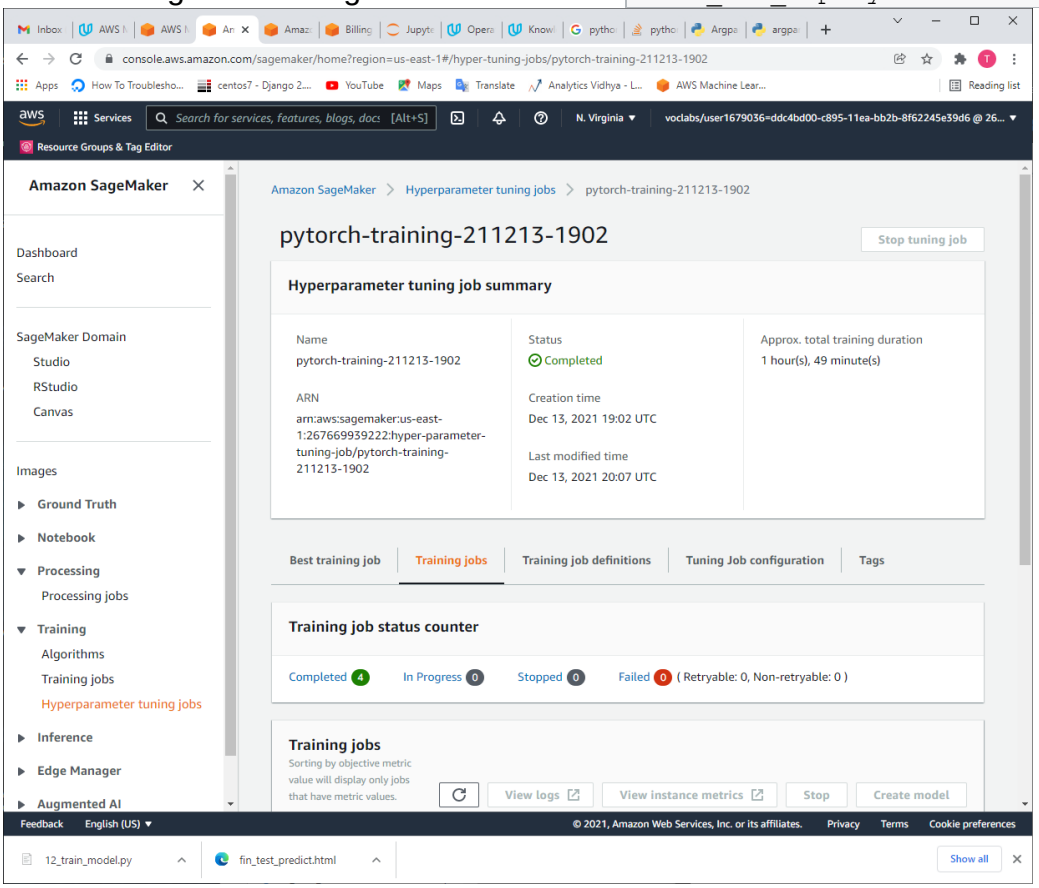
- train
- test
- valid



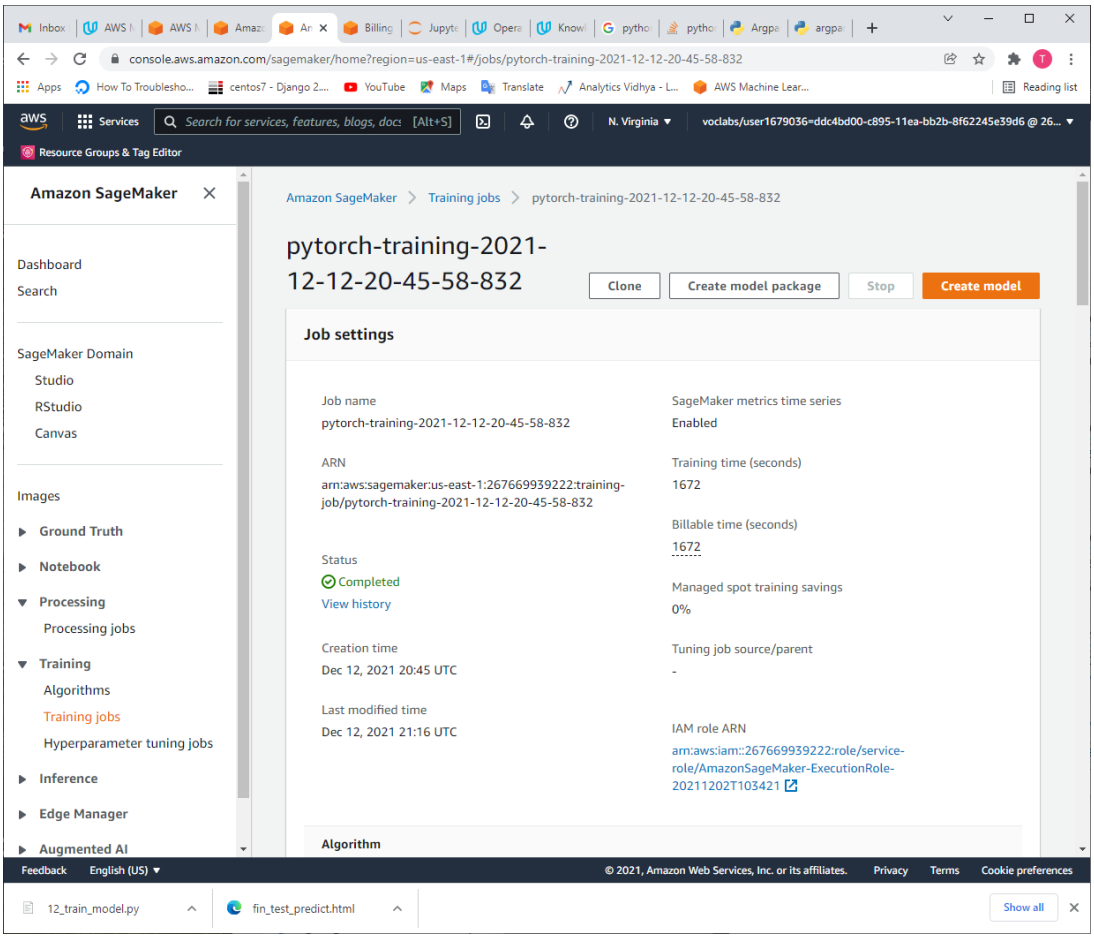
s3_bucket.png

Training and Deployment

Finish tuning and training with 1 instance in `train_and_deploy-solution.ipynb` notebook.

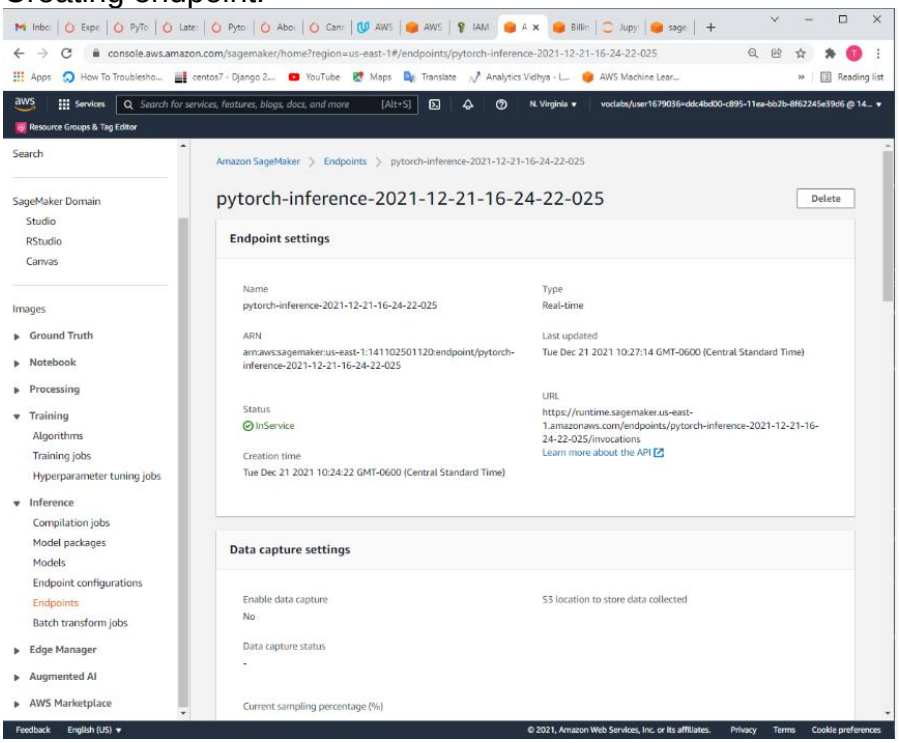


tunning_job.png



training_job.png

Creating endpoint:

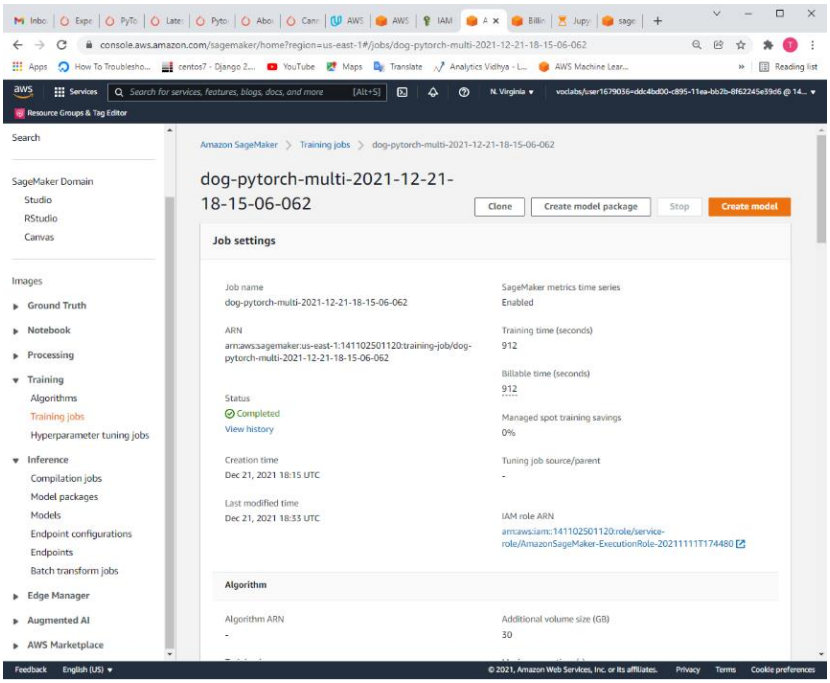


endpoint.png

endpoint_name = pytorch-inference-2021-12-21-16-24-22-025

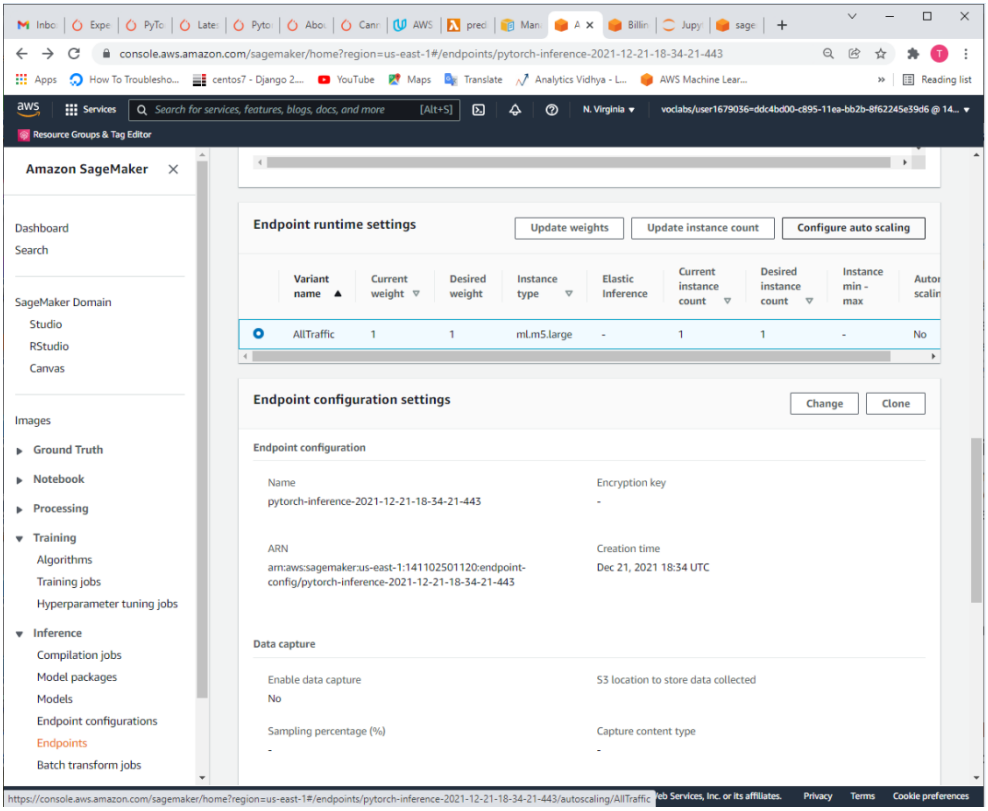
Multi-instance training

Finish performing multi-instance training.



train_multi_inst.png

Creating endpoint from multi-instance training job

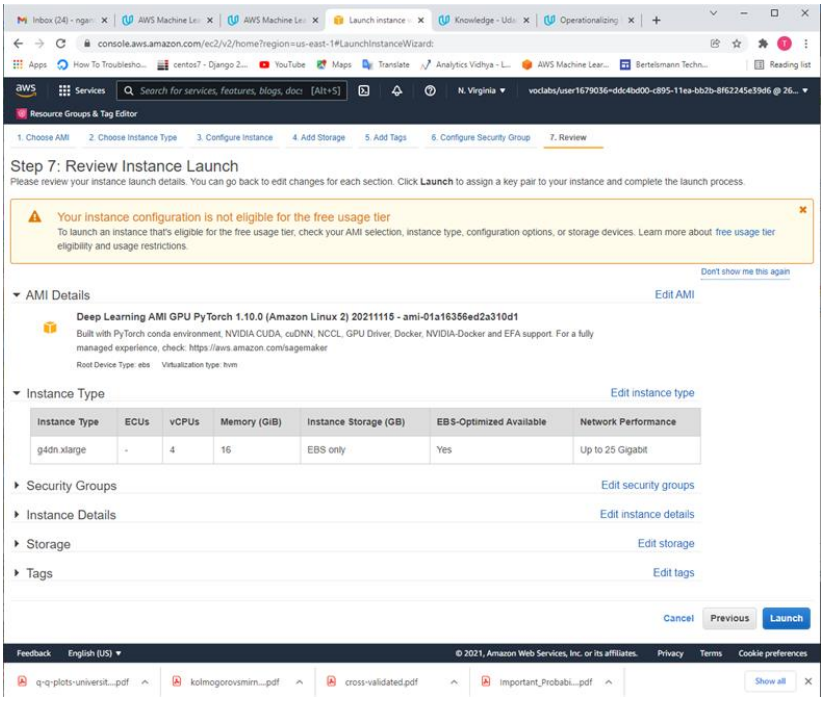


endpoint_multi_inst.png

`endpoint_name = pytorch-inference-2021-12-21-18-34-21-443`

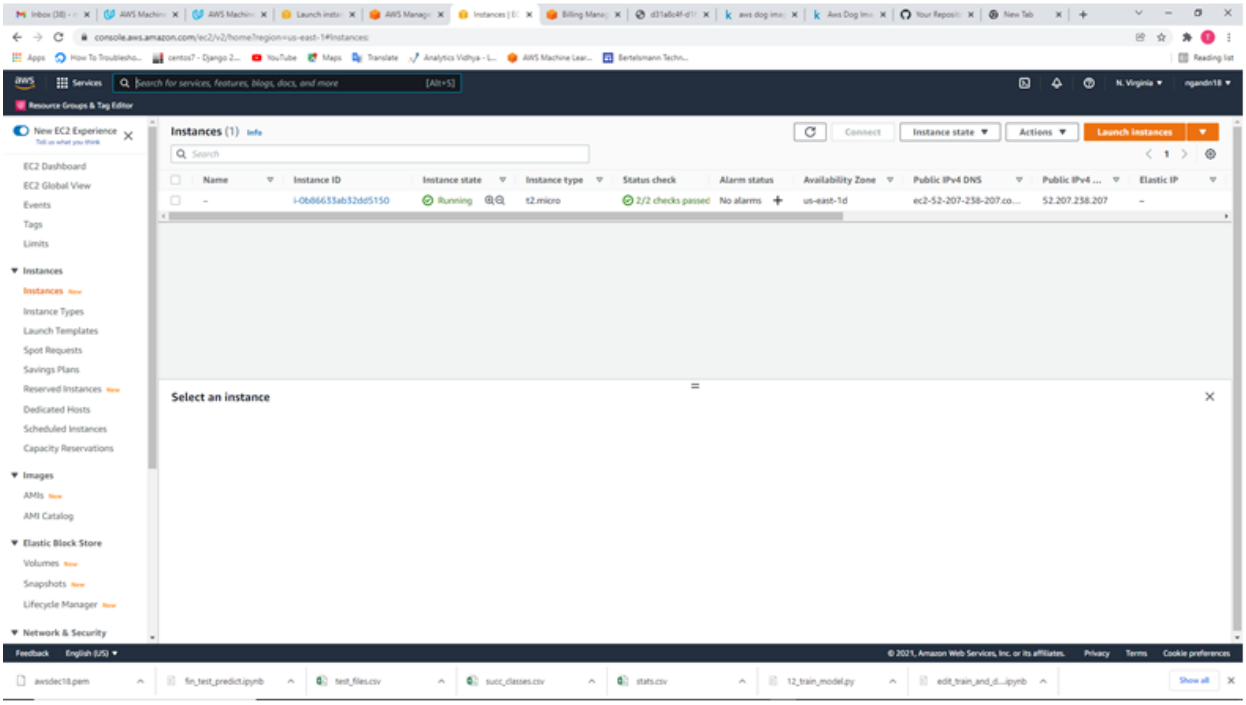
II. Step 2: EC2 Setup

Our project is a Py Torch one, so we need a powerful EC2 instance. But according to our policy, we cannot choose ***ml.g4dn.xlarge*** as we want.



g4dn_not_eligible.png

We have only one eligible selection is ***t2.micro***, then we select an AMI for your EC2 instance: "***Amazon Deep Learning AMI GPU Py Torch***" to use its libraries.



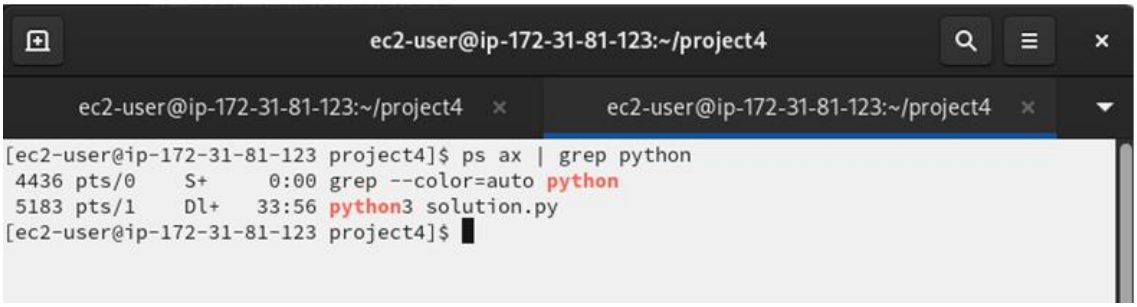
EC2_instance.png

Preparing for EC2 model training

From our Linux machine, we connect to our EC2 instance by ***ssh*** protocol to prepare training:

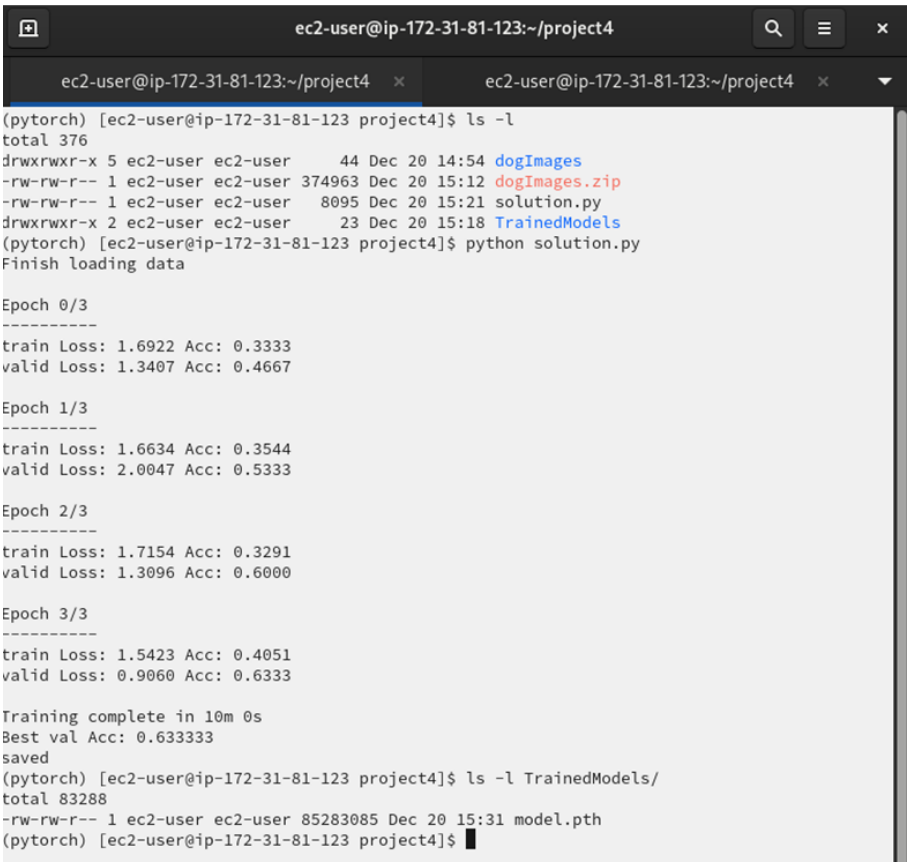
1. The first time, after a few seconds, the system reports "full of memory" and it rejects the program. Our EC2 instance has 1 GB RAM, so its memory cannot work with our model.

2. I try to make swap space (8GB) and it can work, but I wait for a terribly long time (over 33 hours), the training is still in epoch 1.



33_hours_ec2.png

3. After referring the mentor guide, I decide to decrease the number of classes to 4 and I test successfully. Because the EC2 instance power is too weak, it cannot run with a large data.



training.ec2.png

Write about the EC2 code

In solution.py,

1. we don't need to use args as the code in hpo.py

```
    parser=argparse.ArgumentParser()
    parser.add_argument("--batch_size", type=int, default=64, metavar="N",
        help="batch_size for training (default: 64)")
    parser.add_argument("--lr", type=float, default=0.001, metavar="LR",
        help="learning rate (default: 0.001)")
    parser.add_argument('--learning_rate', type=float, default=0.001, metavar="LR",
        help="learning rate (default: 0.001)")
    # parser.add_argument('--batch_size', type=int)
    parser.add_argument('--data_dir', type=str,
default=os.environ['SM_CHANNEL_TRAINING'])
    parser.add_argument('--model_dir', type=str, default=os.environ['SM_MODEL_DIR'])
    parser.add_argument('--output_dir', type=str,
default=os.environ['SM_OUTPUT_DATA_DIR'])
    args=parser.parse_args()
```

We use the variables such as:

```
    data_dir = 'dogImages'
    batch_size = 2
```

2. We don't need to use os.path

```
    torch.save(model.cpu().state_dict(), os.path.join(args.model_dir, "model.pt"))
```

We can use

```
    torch.save(model.state_dict(), "TrainedModels/model.pth")
```

III. Step 3: Setting up a Lambda function

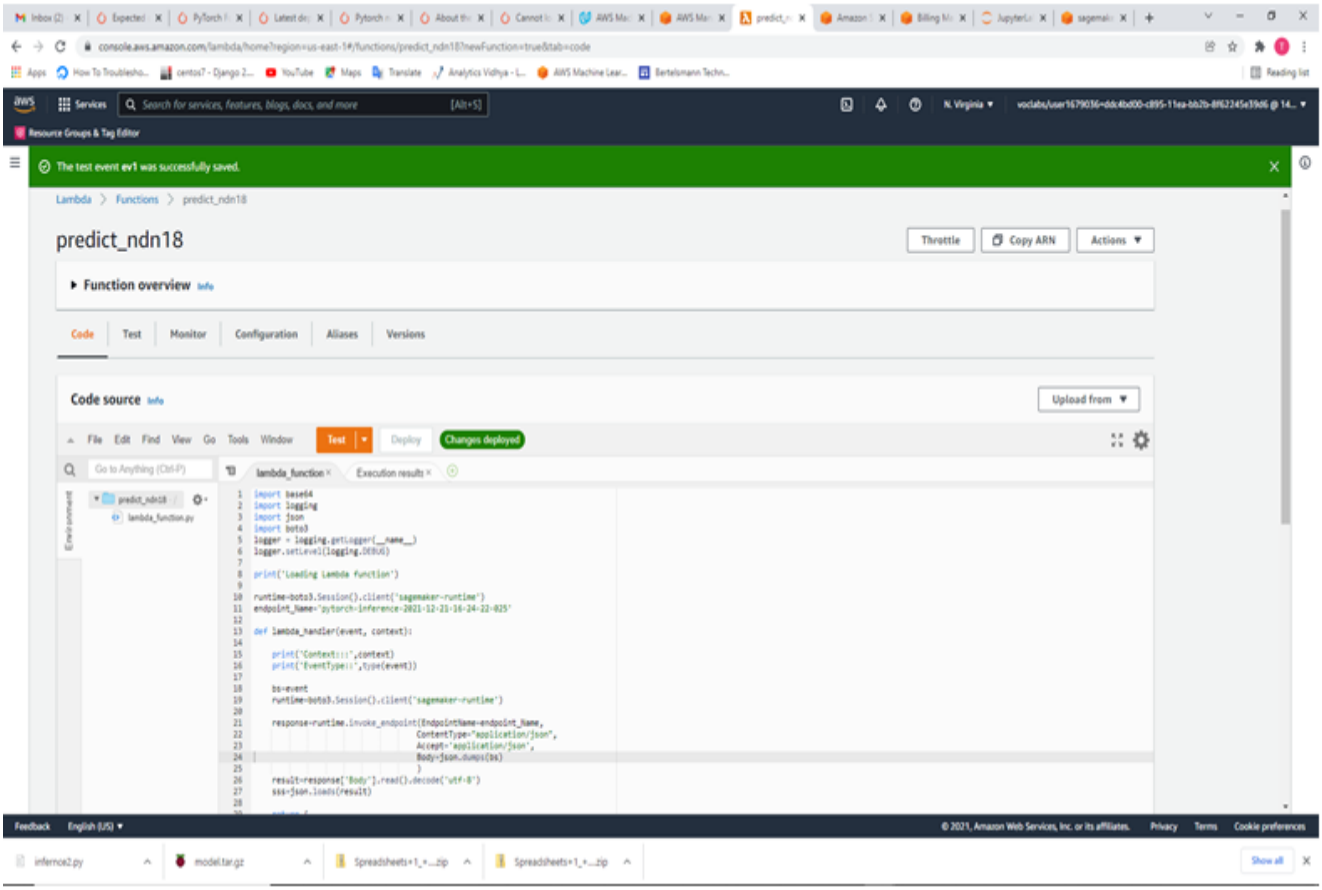
My lambda_function.py name is ***predict_ndn18***

```
import base64
import logging
import json
import boto3
#import numpy
logger = logging.getLogger(__name__)
logger.setLevel(logging.DEBUG)

print('Loading Lambda function')
runtime=boto3.Session().client('sagemaker-runtime')
endpoint_Name='pytorch-inference-2021-12-21-16-24-22-025'

def lambda_handler(event, context):
    print('Context:::',context)
    print('EventType::',type(event))
    bs=event
```

```
    runtime=boto3.Session().client('sagemaker-runtime')

    response=runtime.invoke_endpoint(EndpointName=endpoint_Name,
                                     ContentType="application/json",
                                     Accept='application/json',
                                     #Body=bytearray(x)
                                     Body=json.dumps(bs))

    result=response['Body'].read().decode('utf-8')
    sss=json.loads(result)

    return {
        'statusCode': 200,
        'headers' : { 'Content-Type' : 'text/plain',
                      'Access-Control-Allow-Origin' : '*' },
        'type-result':str(type(result)),
        'COntent-Type-In':str(context),
        'body' : json.dumps(sss)
        }
```

We need the name of our working endpoint:

```
endpoint_Name='pytorch-inference-2021-12-21-16-24-22-025'
```



lambda_func.png

IV. Step 4: Security and testing

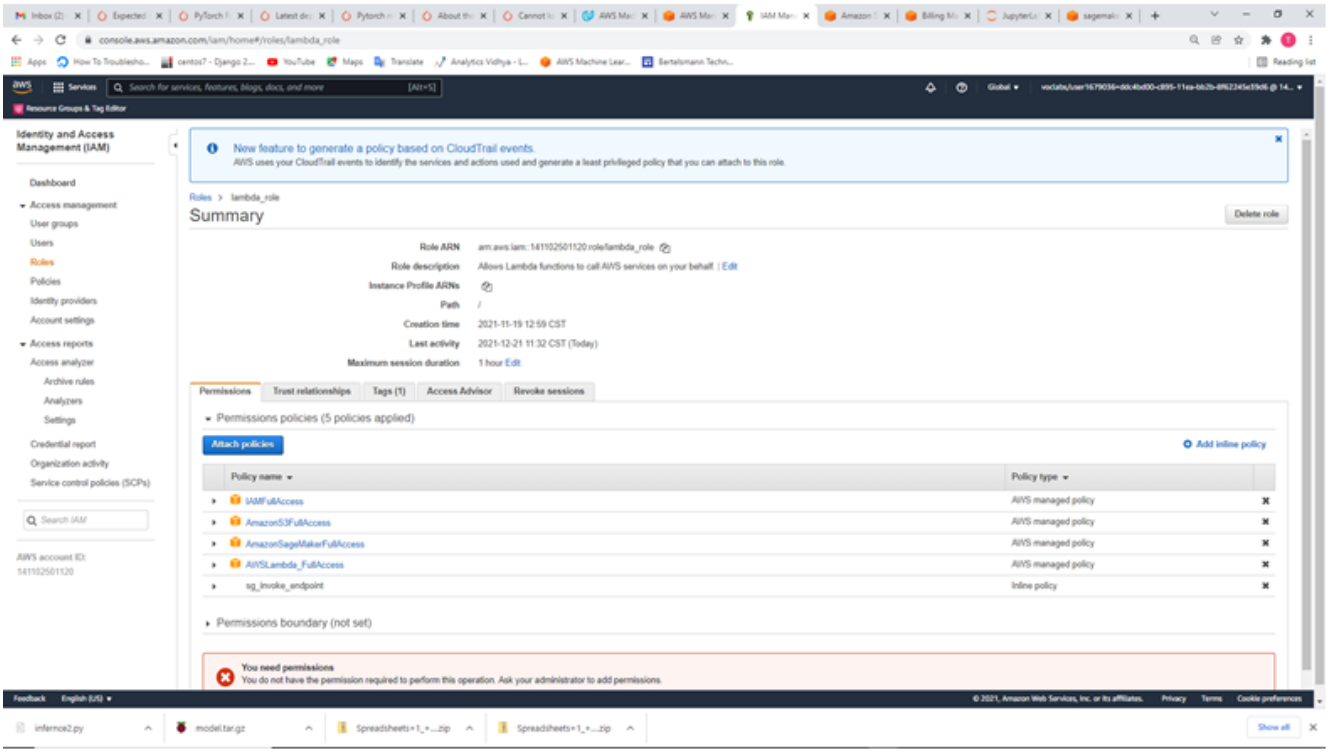
By this way, we can confirm that our ***AWS workspace is very secure*** and ***hardly to have vulnerabilities***.

Lambda function security

We need the IAM role to run the lambda function that revoke runtime endpoint. I have the role ***lambda_role*** with some modification for invoke runtime endpoint:

{"Version": "2012-10-17", "Statement": [{"Sid": "VisualEditor0", "Effect": "Allow", "Action": "sagemaker:InvokeEndpoint", "Resource": "" }] }*

Ref: [Call an Amazon SageMaker model endpoint using Amazon API Gateway and AWS Lambda | AWS Machine Learning Blog]



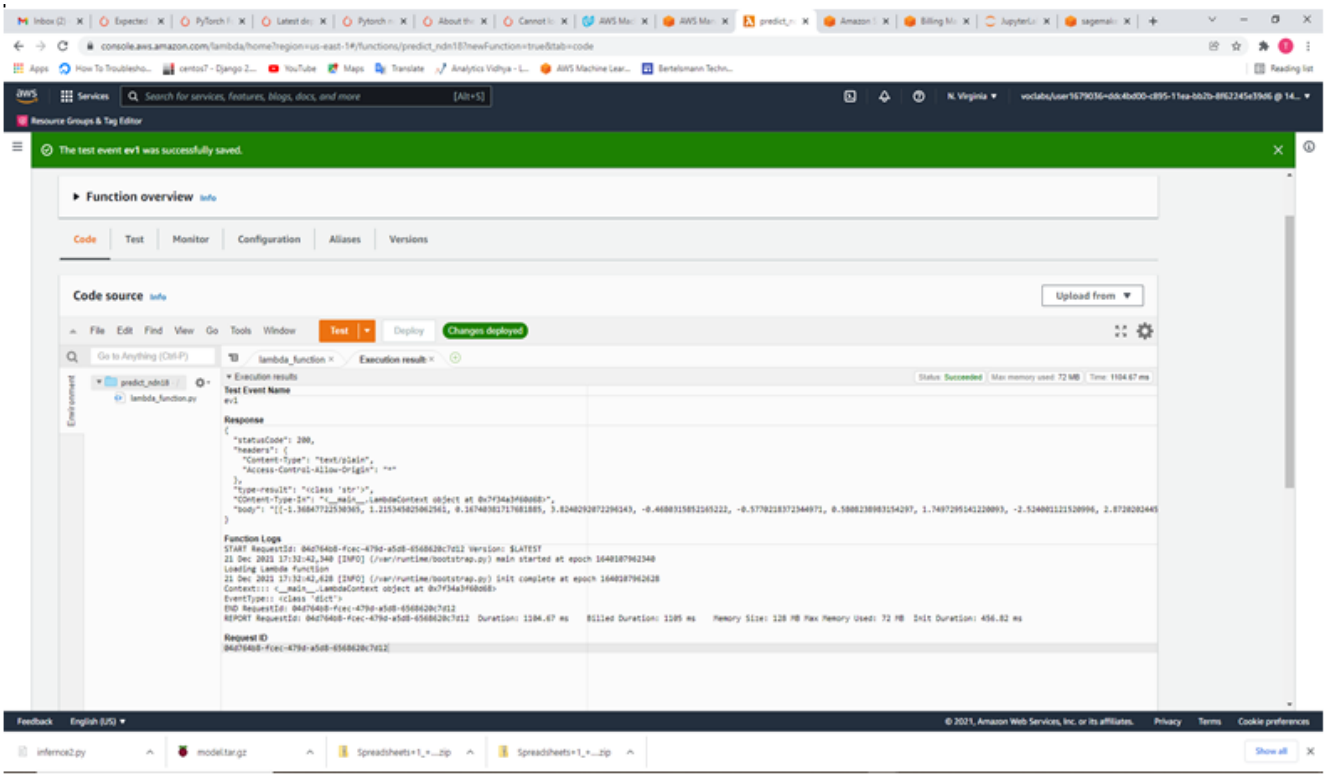
IAM_lambda_role.png

Lambda function testing

With the test event:

```
{"url": "https://s3.amazonaws.com/cdn-origin-etr.akc.org/wp-content/uploads/2017/11/20113314/Carolina-Dog-standing-outdoors.jpg"}
```

We have the successful test screenshot



lambda_test.png

Required result:

"body" : " [[-1.2382080554962158, 1.3273835182189941, 0.43715518712997437, 3.3721883296966553, -0.3995043039 3218994, -0.6558273434638977, 0.6161381006240845, 1.4932968616485596, -2.2320802211761475, 2.65943241119384 77, 1.0774480104446411, -2.785491704940796, 0.8365713357925415, 3.0817410945892334, -2.1395881175994873, -1.0 758476257324219, -1.8001441955566406, 0.2904803454875946, -0.3480547070503235, 5.295973777770996, 3.1650018 6920166, 2.738400459289551, -3.5832791328430176, -2.0125010013580322, -0.9806011915206909, -1.88955581188201 9, -0.29492682218551636, -1.4020986557006836, -1.505773663520813, -0.5953605771064758, 0.6105459332466125, -1. 222814679145813, -1.540320873260498, -0.15392744541168213, -2.2085208892822266, -0.5182346701622009, -0.2015 720158815384, -0.07690005004405975, 3.015324115753174, -0.8311461210250854, 1.3749828338623047, 2.354694604 8736572, 6.490350723266602, 1.2858328819274902, 0.5774887204170227, -3.322958469390869, 1.1982107162475586, 0.55669766664505, 0.7860574126243591, 1.3508168458938599, -0.015121916308999062, -1.5248268842697144, -2.418 215751647949, 0.1560056209564209, -1.308781385421753, -0.721088707447052, -1.0353213548660278, -2.3086767196 655273, 1.7806625366210938, 1.1612213850021362, -1.8093127012252808, -2.7293269634246826, -2.05367279052734 38, -3.31512188911438, -1.6630479097366333, -0.5676873326301575, 3.5655899047851562, -1.9211697578430176, 0.6 224690675735474, 2.3312771320343018, 3.6460840702056885, -0.009003717452287674, -1.3386286497116089, 1.6006 838083267212, -0.7080104351043701, -0.8765071630477905, -2.586827278137207, 1.3131886720657349, -0.683409690 8569336, -0.7863197326660156, 4.080479621887207, -2.018925666809082, 4.188110828399658, 2.891594648361206, - 2.3618345260620117, -1.7194092273712158, 2.2415754795074463, -0.8918696045875549, 1.4400774240493774, 1.6143 591403961182, -2.842221736907959, -0.9285373091697693, 0.1497386246919632, -1.6729791164398193, -0.732226848 6022949, 0.2776722013950348, -1.0475971698760986, -0.44095420837402344, -1.8494633436203003, -2.173632383346 5576, -1.8202781677246094, 0.674027144908905, 0.7670314311981201, -1.2528178691864014, -0.16699440777301788, -2.2693119049072266, -1.765668272972107, 3.662839412689209, 1.032089114189148, 2.2445805072784424, 1.1335695 98197937, -0.078820139169693, -1.9275776147842407, -0.4728870391845703, -0.9335609078407288, 0.2269243150949 4781, -0.9603256583213806, 1.0221415758132935, -1.2410348653793335, 3.8171565532684326, -0.0057304389774799 35, -1.1045581102371216, 0.7753564119338989, 0.39711835980415344, -2.5719871520996094, -2.1391401290893555, - 0.16758006811141968, 1.7172927856445312, -2.193082332611084, -1.5543423891067505, -1.261643886566162, 1.9406 766891479492, 0.04177605360746384]]"

V. Step 5: Concurrency – Auto Scaling

Set up concurrency for our Lambda function.

Concurrency will ***make our Lambda function better able to accommodate high traffic*** because it will enable our function to respond to multiple invocations at once.

Concurrency will require our project to use more computing resources, so it will ***increase our costs***.

We need to ***check if our lambda function is really in high traffic situations***. If it's relatively low, no concurrency or minimal concurrency may suffice, if it is really in high traffic situation we need to setup concurrency for our Lambda function.

To setup concurrency, we open our Lambda function in the Lambda section of AWS.

1. Open the ***Configuration*** tab.
2. Configure a ***Version*** for our function, in the ***Version*** section of the ***Configuration*** tab.
3. After configuring a Version for our Lambda function, navigate to the ***Concurrency*** section of the ***Configuration*** tab of our function. Use this section to configure concurrency for our Lambda function.
4. We must notice to the cost of our concurrency. We'll have to consider the traffic that our project is expected to get.
5. If we are not sure how high the traffic is, we can set up minimal concurrency and adjust suitable for it.
6. For example, because provisioned concurrency has a higher cost than reserved concurrency:

    a) Set up the maximum reserved instances is **4**.
    b) Provisioned concurrency is **2**.
    After a short time to determine the real accurate traffic, we adjust these number to be suitable.

7. After setting up concurrency on our Lambda function, we should test it again to make sure it's still working.

Auto-scaling

In addition to setting up concurrency for our Lambda function, we should also set up ***auto-scaling for our deployed endpoint***.

1. We can accomplish this by navigating to the ***Inference*** > ***Endpoints*** section of Sagemaker. This will open a list of our deployed endpoints.
2. Click on the endpoint that we deployed in Step 1 of the project. This will open a configuration dashboard for our deployed endpoint.
3. This dashboard has a table called ***Endpoint runtime settings*** that has a row for every ***variant*** of our endpoint. There should be one row in this table.
4. Select the only row in this table, and click ***Configure auto scaling*** to start the process of configuring auto-scaling for our endpoint.

    a. Click on the variant of our endpoint, this is the default variant of our endpoint - ***AllTraffic***.
    b. We click on "Configure auto-scaling" and specify a minimum and maximum instance count.
    c. For testing and follow the true traffic, we can set "maximum instance" is 3 and "minimum instance" is 1.
    d. We need to track the true traffic to avoid incurring costs.
5. Specify our scale in cool down and our scale out cool down:
    a. When I test the executive time of our endpoint, I record that each request needs 0,19s to complete, so the number 20 for our two parameters is acceptable.
    b. We need to track the real traffic to adjust these suitable numbers.

When we set up concurrency and auto-scaling, we have several choices about configuration. We must consider the efficiency and the cost that we must paid, so we always examine carefully the traffic, the execute time of the lambda function, the response time of our endpoint and the necessary requirements to decide which configuration is compatible.