**Operationalizing an AWS ML Project**

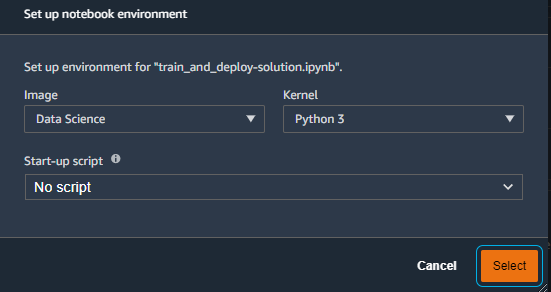
-Pooja Singari

1. Initial Setup:

**ml.t3.medium** instance type was chosen for the notebook because of it’s fast-launch,Moreover they are designed specifically for tasks that result in moderate usage of CPU with temporary spikes.Despite the higher cost as compared to ml.t2.medium it has extra CPU credits which proves to be more beneficial in terms of performance.

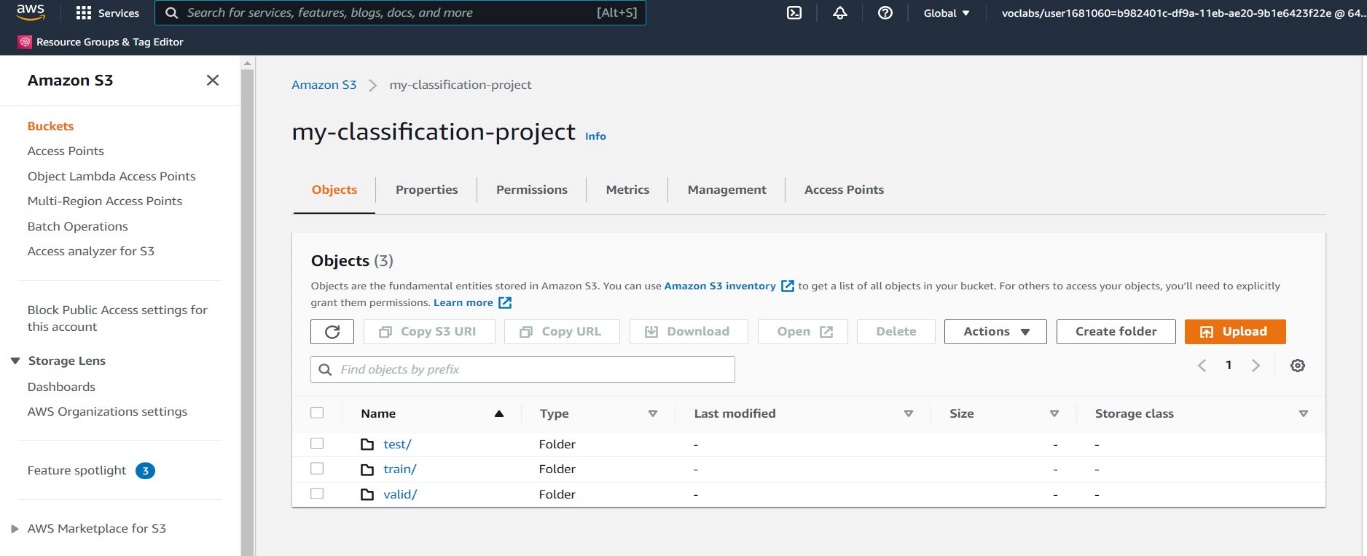
**Kernel:** Python3

**Image**:Data Science



**Figure 1 – Setting up Notebook environment in JupyterLab**

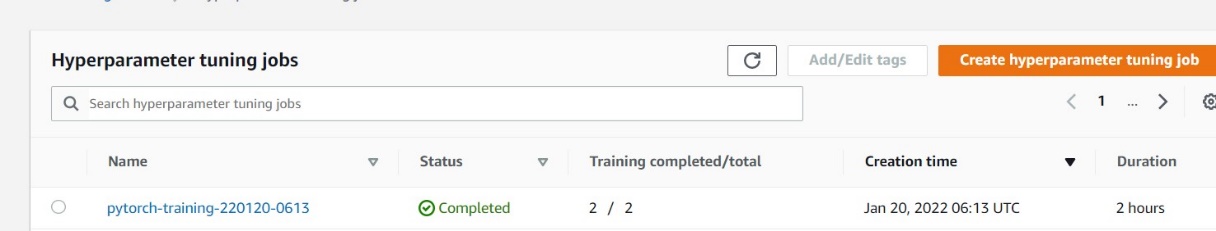
The dog-breed dataset downloaded and unzipped after which it was successfully uploaded to s3 bucket via !aws s3 cp dogImages s3://my-classification-project/ --recursive command



**Figure 2 – S3 bucket holding the dog-breed dataset**

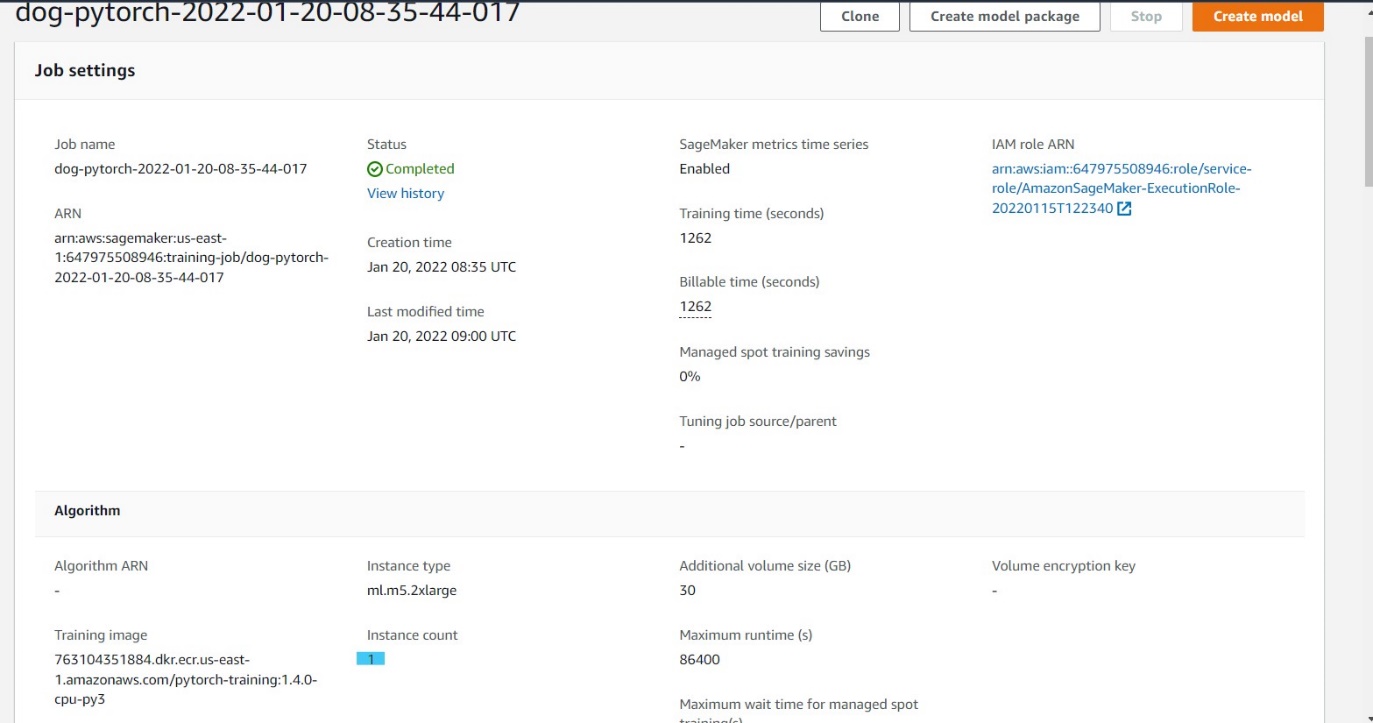
1. SageMaker Training and Deployment:

For hyperparameter tuning I used “ml.m5.xlarge” instance\_type ,as it’s computational performance seemed sufficient for the current Image classification project.Hyperparameter tuning job ( **max\_jobs = 2, max\_parallel\_jobs = 2** ) was completed successfully:

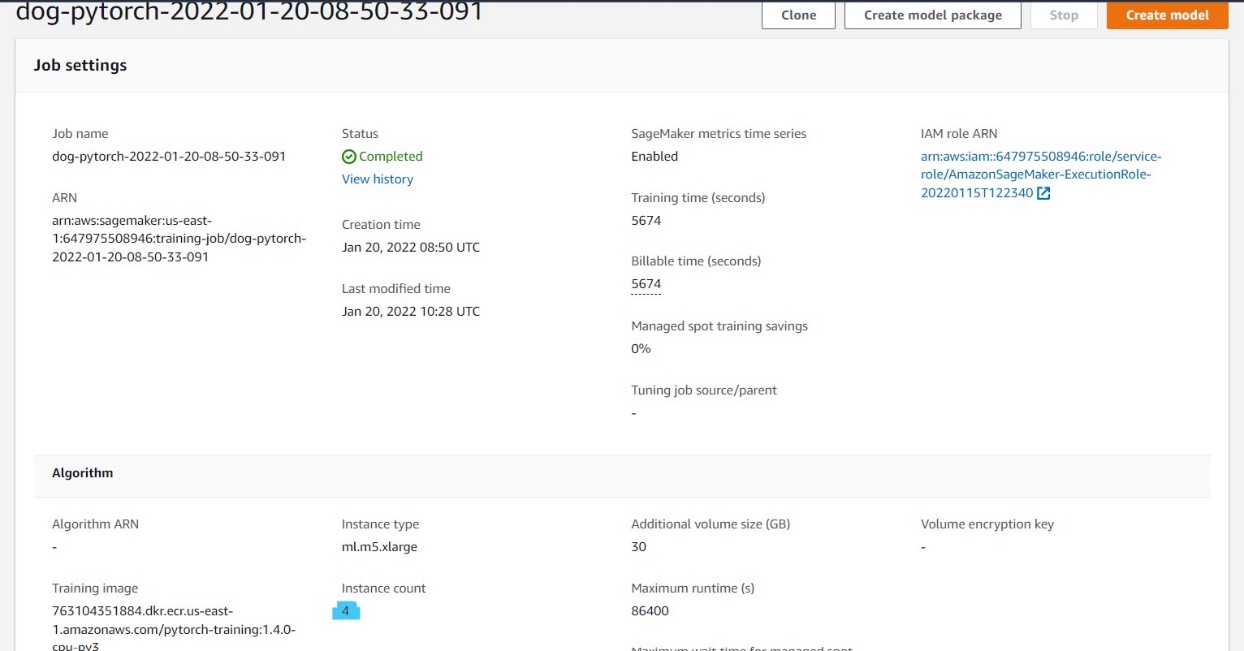


**Figure 3 – Hyperparameter Tuning Jobs**

After completion of tuning jobs we had to create single and multi-instance training jobs as shown in the figures below



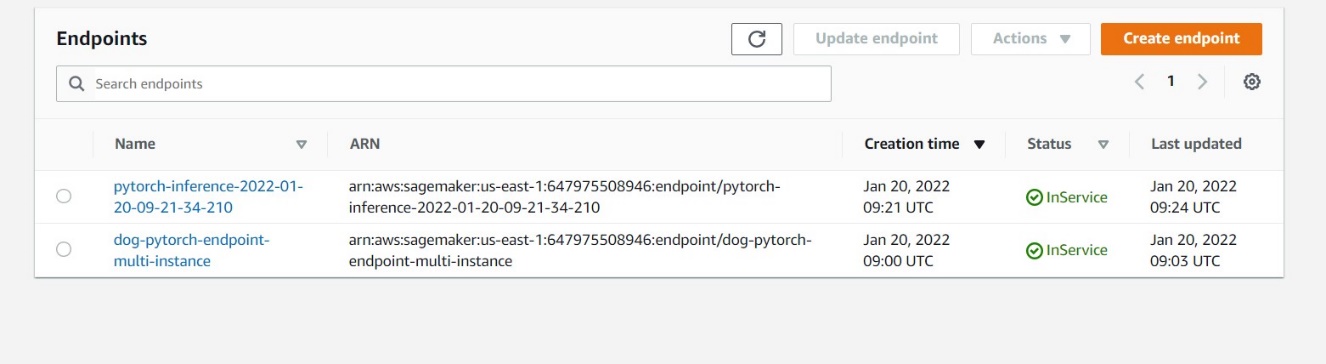
**Figure 4 – Single Instance training job**



**Figure 5– Multi-instance training job**

**Deployed** Endpoints to invoke the model and receive the outputs:

* Single instance deployed endpoint: “pytorch-inference-2022-01-20-09-21-34-210”
* Multi instance deployed endpoint: “dog-pytorch-endpoint-multi-instance”



**Figure 6 – Deployed Endpoints**

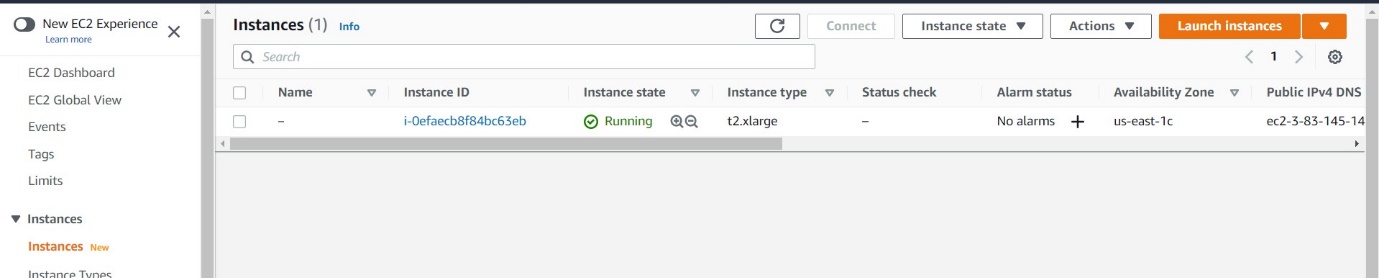
1. EC2 Instance

* I used the **t2.xlarge** instance and the **Deep Learning AMI (Amazon Linux 2) Version 55.0**. as I wanted a moderate performance which was not on the high end in terms of cost for a mere Image classification model
* Moreover t2.xlarge has a baseline performance of **90%** with **4 vCPU’s** along with 16 GiB of memory just at the charge of **$0.188** per hour which is considerably lower than sagemaker notebook instances.

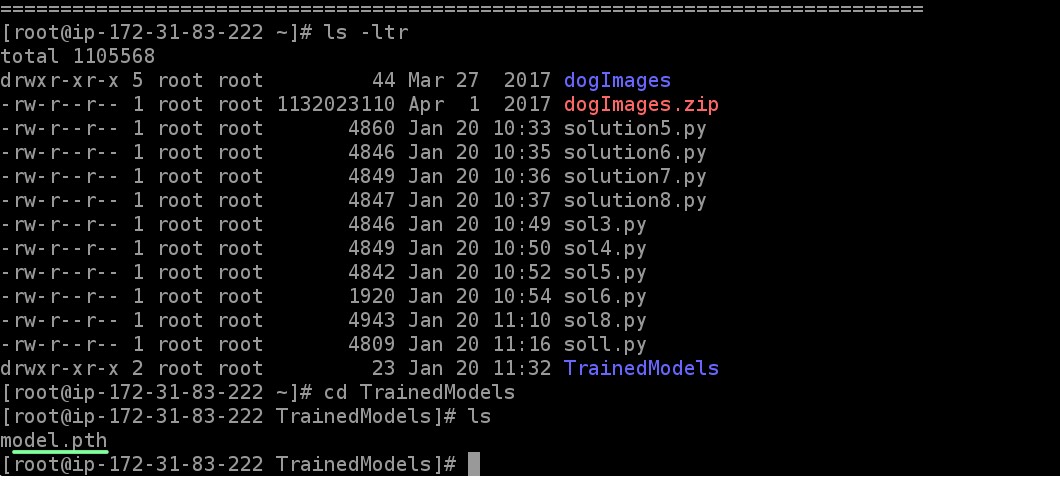
**Difference between ec2train1.py (EC2 script) and train\_and\_deploy**

**solution.ipynb (SageMaker scripts)**

1. In the EC2 training script, all the hyperparameters are mentioned within the script and there is no need for **argparse** unlike SageMaker scripts.
2. In EC2 ,the training happens on the server where you write the code,the screen just hangs in there till your model finishes training ,there is no way to know the progress of your training,but in SageMaker you can employ several training jobs and also keep a track of it’s progress via SageMaker dashboard.
3. You cannot directly deploy an endpoint using EC2 trained model as opposed to the ease of creation of an endpoint using SageMaker Trained models
4. Moreover ,writing code in EC2 is similar to a command-line terminal which does not have a user friendly interface as opposed to the well structured interface provided via SageMaker scripts which is easier to navigate and edit.
5. It is harder to edit code on EC2 instances ,it requires one to remember certain codes such as**:** :wq! **(**for saving) , :qa! (for exiting). Even upwards traversal through your code changes spacing which leads to syntax error.hence it’s not convenient for larger projects unlike sagemaker scripts



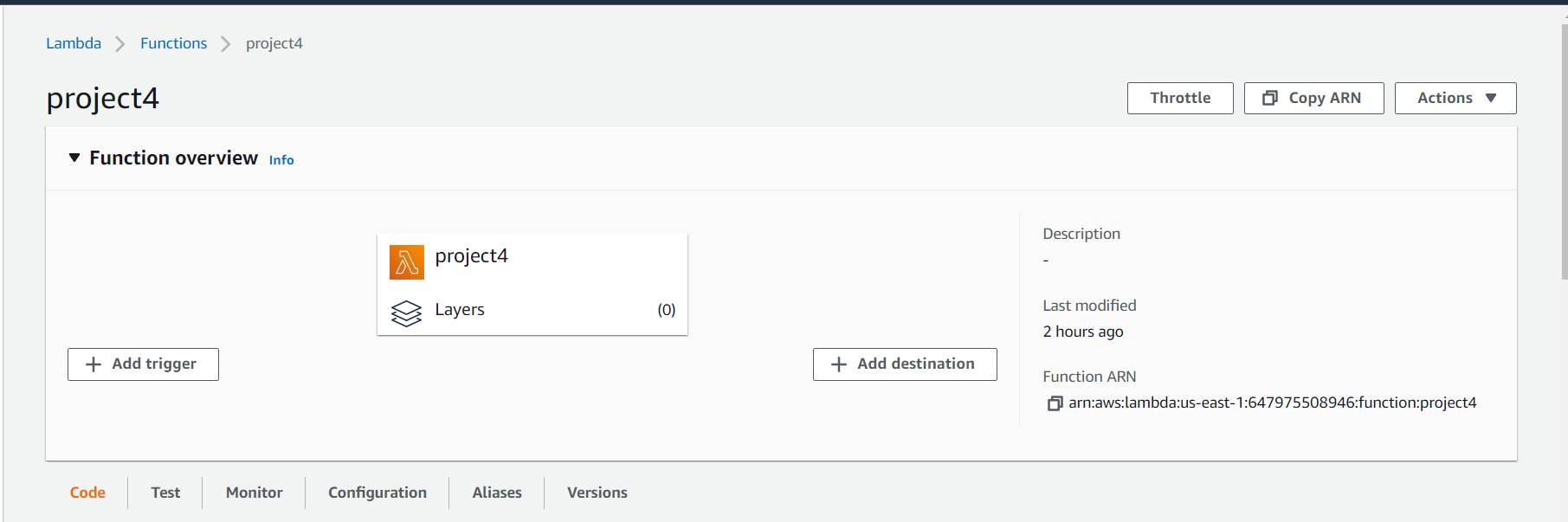
**Figure 7 – EC2 Instance snapshot**



**Figure 8– EC2 Training saved model.pth**

1. Lambda Functions

I have used a previously deployed endpoint named ““dog-pytorch-endpoint-multi-instance” to invoke my endpoint which had been deployed using the model that was created earlier in the project to classify dog breeds.

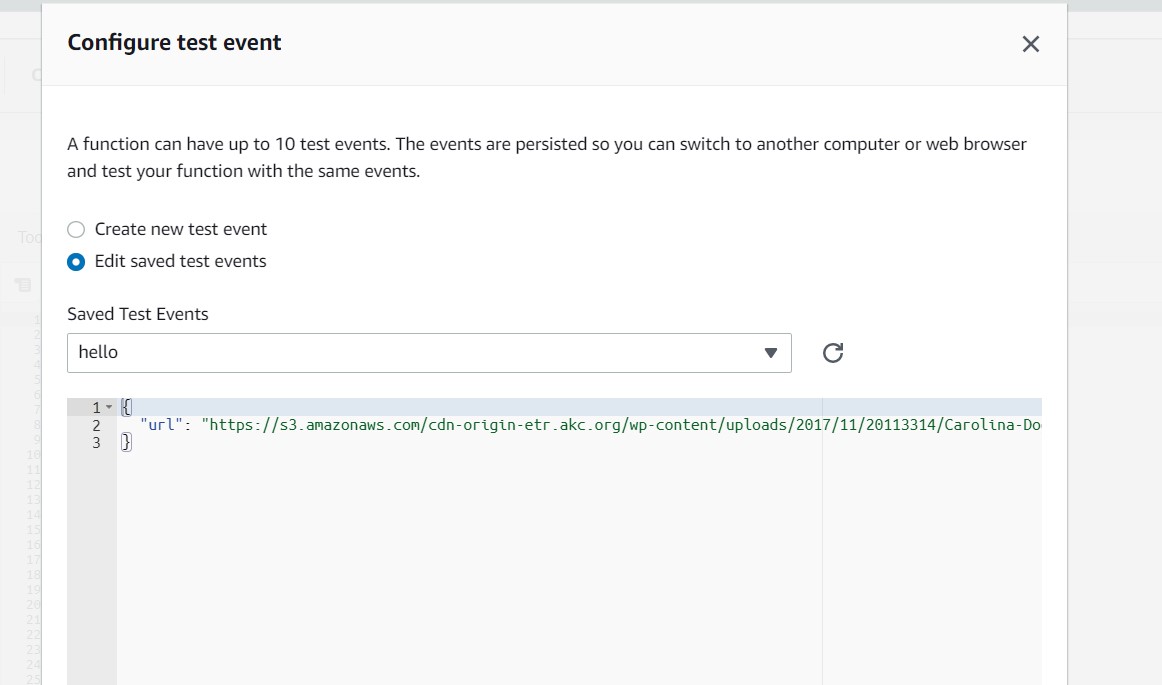


**Figure 9 – Lambda Function**

5.Security and Testing

Lambda functions respond to events that have been passed to it by invoking an earlier deployed endpoint. Hence it allows us to perform serverless computing by accessing S3 buckets and inference endpoints via managed policies.

* We had created a new role for this lambda functions with **“SageMakerFullAccess”**(without which it leads to **Invocation Error**)
* We had to use the below test event to test our Lambda function: It’s a JSON object which on being passed returns a 200 status code along with an output array.



**Figure 10 – Lambda Function Test Event**

**Response** from Lambda Function:

{

"statusCode": 200,

"headers": {

"Content-Type": "text/plain",

"Access-Control-Allow-Origin": "\*"

},

"type-result": "<class 'str'>",

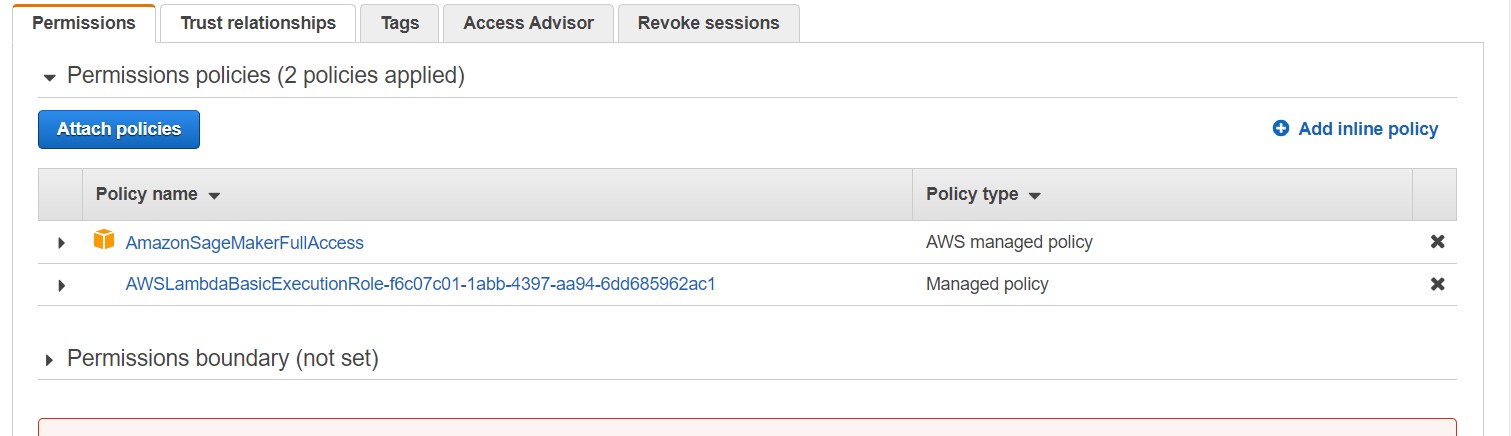
"COntent-Type-In": "LambdaContext([aws\_request\_id=c6dae78f-40b0-46ca-a44a-58d7b1cdca4b,log\_group\_name=/aws/lambda/project4,log\_stream\_name=2022/01/20/[$LATEST]cac131cc588742a0a203734c6298d530,function\_name=project4,memory\_limit\_in\_mb=128,function\_version=$LATEST,invoked\_function\_arn=arn:aws:lambda:us-east-1:647975508946:function:project4,client\_context=None,identity=CognitoIdentity([cognito\_identity\_id=None,cognito\_identity\_pool\_id=None])])",

"body": "[[-8.509687423706055, -7.77571964263916, -2.675638198852539, -2.3557212352752686, -6.151639938354492, -7.035409927368164, -3.780886173248291, -2.774186134338379, -8.81319522857666, -1.3065375089645386, 1.6190638542175293, -5.865751266479492, -3.352370023727417, 0.33569395542144775, -7.82021427154541, -3.9492571353912354, -8.446880340576172, -2.839498996734619, -4.651222229003906, 1.0388201475143433, -7.5899882316589355, -3.5086326599121094, -9.712128639221191, -8.360410690307617, -8.240933418273926, -13.016434669494629, -5.23638391494751, -7.427854537963867, -6.946070194244385, -2.703977108001709, -7.135708332061768, -5.68403434753418, -11.949724197387695, -4.561254024505615, -11.778522491455078, -8.986464500427246, -7.212772369384766, -3.3191754817962646, -1.1046655178070068, -6.461777687072754, -5.033671855926514, -2.7278974056243896, -0.9936132431030273, -6.044423580169678, -1.883324384689331, -9.649934768676758, -3.4763667583465576, -1.0421169996261597, -2.717921257019043, -3.589296817779541, -5.795350551605225, -7.381248474121094, -8.431464195251465, -7.530313491821289, -8.174006462097168, -3.8078136444091797, -6.326535701751709, -8.39573860168457, -2.9995415210723877, -5.429052829742432, -10.660608291625977, -9.486539840698242, -11.510260581970215, -6.826721668243408, -6.756300449371338, -14.48521614074707, -1.6789501905441284, -9.15797233581543, -2.646780490875244, -1.1237974166870117, -0.6023721694946289, -6.902181148529053, -5.664377689361572, -8.664715766906738, -7.221043586730957, -3.847078323364258, -13.318488121032715, -4.377753257751465, -4.632699489593506, -6.257696151733398, -1.2542197704315186, -10.409688949584961, -2.2787129878997803, -1.0235614776611328, -8.419547080993652, -8.682510375976562, -2.885606527328491, -10.519001960754395, -4.825105667114258, -2.1089701652526855, -8.725203514099121, -5.307856559753418, -5.575226783752441, -9.437766075134277, -5.222029685974121, -4.098916053771973, -6.385500431060791, -7.366503715515137, -7.367201805114746, -6.432024002075195, -7.426565170288086, -1.6843739748001099, -7.017165184020996, -5.080165863037109, -8.466792106628418, -14.218036651611328, -4.277678966522217, -0.39628297090530396, -1.378735899925232, -3.197972536087036, -2.170497179031372, -3.1416969299316406, -8.856181144714355, -6.239938259124756, -7.6649956703186035, -1.4355008602142334, -6.978602409362793, -1.5712144374847412, -4.779670715332031, -1.2244170904159546, -3.855419397354126, -6.736294746398926, -5.722675800323486, -6.578609943389893, -12.074214935302734, -5.719165802001953, -4.999317169189453, -2.3081724643707275, -5.571413516998291, -9.936185836791992, -8.904071807861328, -4.077090263366699, -7.278857231140137]]"

}

## **Security:** I believe that in an industrial set up granting **FullAccess** policies may do more harm than good when the task requires just the invocation of a single component as in this project,wherein without SageMakerFullAccess policy,the lambda function kept throwing invocation error.

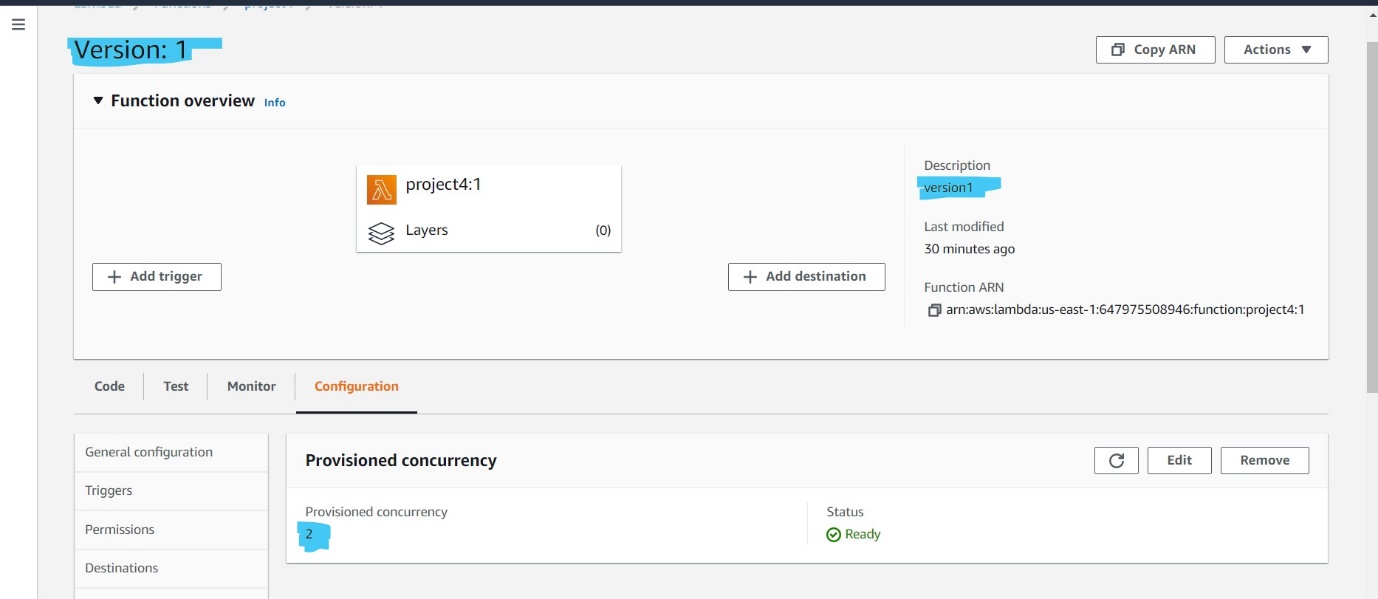
It would be considered a better practice to create **IAM customer managed policies** that provide your team members with permission which is required to perform the task at hand.



**Figure 11- Permission Policies**

6.Concurrency and AutoScaling

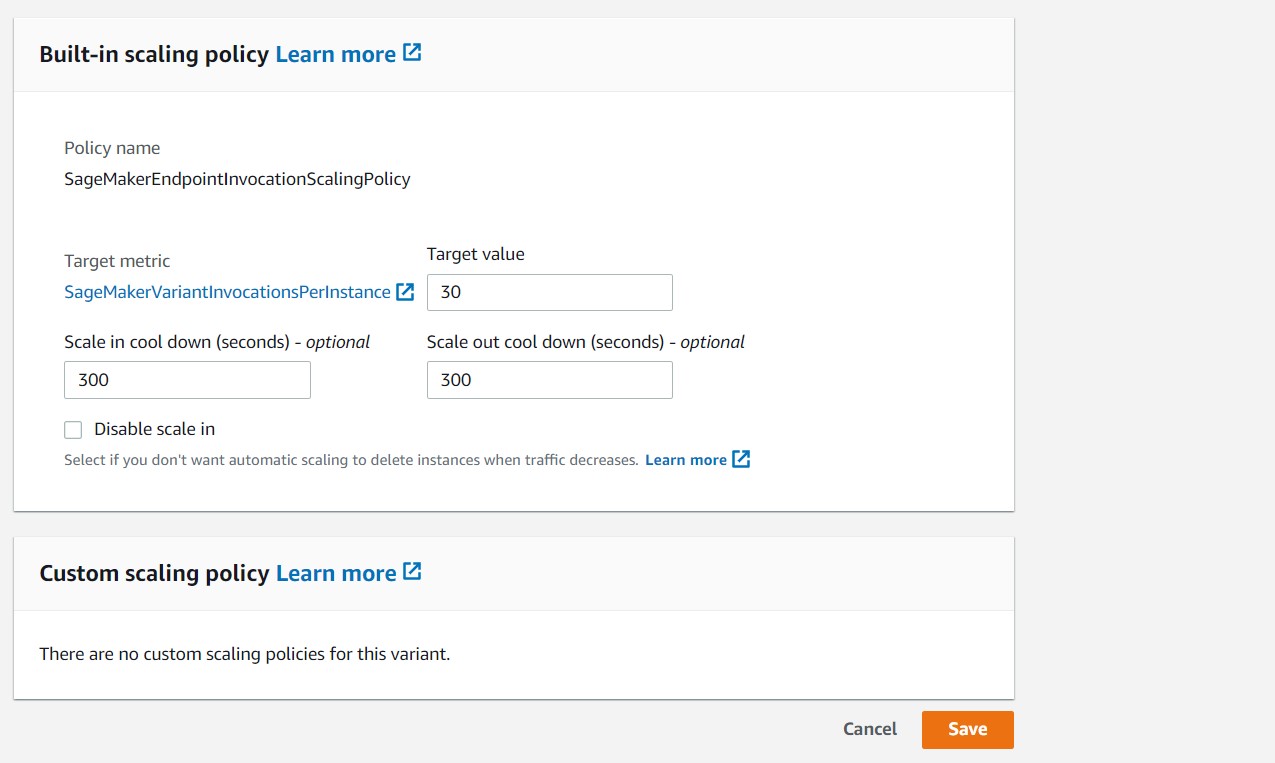
* We were first asked to create a different version for our lambda function after which we were supposed to add concurrency to it.As seen below I have added a provisioned concurrency of 2 to my first version.



**Figure 12- Version(1): Provisioned Concurrency**

* I have set up **Provisioned Concurrency** because it creates instances that are always on and can reply to all traffic without requiring a wait for start-up times.
* Provisioned concurrency has a higher cost than reserved concurrency but since it turns on instances that are always ready to respond to traffic, it can achieve low latency even in very high traffic scenarios.

**Autoscaling:** Setting the target value to be 30 along with default values of 300 seconds for scale in and scale down seemed to sufficient for the current scenario as it would efficiently deal with the traffic



**Figure 13- AutoScale Configuration**