1. Initial Setup

The Sagemaker instance i create is **ml.m3.medium**. The ml.m3.medium is a general-purpose instance that offers a balanced configuration of compute, memory, and storage at alow cost It provides 1 vCPU and 3.75 GiB of memory. It's good choice for development, testing and lightweight machine learning workload, especially when working with small-to-medium datasets or performing preliminary model training.

2. EC2 Setup

The EC2 instance i created is instance has instance type is **t2.micro** and AMI (Amazon Machine Image) is "*Deep Learning OSS Nvidia Driver AMI GPU PyTorch 2.5.1 (Amazon Linux 2023) 20241120*".

- The t2.micro instance is part of the AWS free tier, making it highly cost-effective for light workloads or small-scale deep learning experiments. It provides 1 vCPU and 1 GiB memory.
- The selected AMI "Deep Learning OSS Nvidia Driver AMI GPU PyTorch 2.5.1" comes pre-installed with:
    - The latest PyTorch 2.5.1 framework
    - GPU drivers for scalable performance
    - An optimized Amazon Linux 2023 environment, which is well-suited for deep learning workflows.

3. EC2 Code

The code i worked with in Step 1 was specifically written to work in Sagemaker. It **designs philosophy and integration with the environment**. The code leverages SageMaker's features, such as hyperparameter optimization, built-in logging of metrics, and interaction with S3 for seamless data storage and retrieval. The hyperparameter tuning process and the use of pre-defined rules for debugging and profiling further enhance the SageMaker workflow for cloud-based model development

In contrast, **ec2train1.py** is designed to run locally or on an EC2 instance. It does not include advanced features like automated hyperparameter tuning or integration with cloud-native storage (e.g., S3). Instead, it relies on standard Python and PyTorch libraries to train and evaluate the model. The logging is less sophisticated, and the training process is hard-coded for fewer epochs and smaller datasets, reflecting the constraints of local or lightweight environments

4. Setting up Lambda function

The provided Lambda function serves as an integration layer between incoming event data and an Amazon SageMaker endpoint name:
pytorch-inference-2024-11-25-17-53-53-884
where the ML model is deployed
When the function is triggered, the *lambda_handler* receives an *event* and a *context*. It processes the input event: decode the input by using base64 and sends the decode data as request to SageMaker endpoint,The response from endpoint is parsed and convert to JSON format with HTTP Status, headers

5. Lambda function testing

To execute the Sagemaker endpoint by lambda function, we need attach the AmazonSageMakerFullAccess policy to **haont1-lambda-function-role-tiak0e85** role.
Response:


```
{
 "statusCode": 200,
 "headers": {
   "Content-Type": "text/plain",
   "Access-Control-Allow-Origin": "*"
 },
 "type-result": "<class 'str'>",
 "COntent-Type-In": "<__main__.LambdaContext object at 0x7fd20e1afb80>",
 "body": "[[-7.186712265014648, -1.9093202352523804, 0.6950663924217224,
2.4817941188812256, 2.4456496238708496, 1.8820074796676636,
2.0392987728118896, -0.19391047954559326, -6.493125915527344,
2.0568480491638184, 1.280523657798767, 0.2914726436138153,
0.4165630340576172, 2.8352458477020264, -2.2367725372314453,
3.1208529472351074, -5.083155632019043, -1.5720800161361694,
-0.15463830530643463, 2.472303628921509, 1.3132493495941162,
2.8746018409729004, -3.0012197494506836, 0.02661910280585289,
-5.652354717254639, -6.248774528503418, -2.089911460876465,
-2.4187519550323486, -0.5695735812187195, -2.799633502960205,
1.2931052446365356, -3.1527419090270996, -6.894887924194336,
-4.345378875732422, -7.1333112716674805, -4.0213623046875,
0.6083419919013977, -3.587517499923706, 2.2315735816955566,
-4.890777587890625, -2.96830415725708, -1.2419970035552979,
2.5835201740264893, -2.539724349975586, 2.06386399269104,
-2.2464566230773926, 0.7242380380630493, 2.4554831981658936,
-1.9356286525726318, -0.7454302310943604, 2.635218620300293,
-4.091686248779297, -2.6923158168792725, 2.4974658489227295,
-4.434329032897949, -1.1531683206558228, -1.9944254159927368,
-3.887418508529663, -4.291004657745361, -2.187577724456787,
```

```
-4.244577884674072, -6.141637802124023, -5.06566047668457,
-6.214316368103027, -1.0288679599761963, -4.648942947387695,
3.5183217525482178, -6.924302101135254, -1.4223833084106445,
0.36098015308380127, 2.6327595710754395, -5.390843391418457,
-2.904773473739624, -4.46401834487915, -4.794497013092041,
1.252897024154663, -11.176520347595215, -4.960510730743408,
1.0609041452407837, -2.3538029193878174, 1.7078369855880737,
-6.879121780395508, 2.0819358825683594, 2.3548145294189453,
-1.8791985511779785, -3.2485146522521973, 0.9334737062454224,
-8.065655708312988, -1.7736318111419678, 0.5537471175193787,
-1.7865140438079834, 2.5039753913879395, -4.738636493682861,
-2.6337740421295166, 1.1038702726364136, -2.879991292953491,
-1.3894251585006714, 0.06456659734249115, -4.783239841461182,
-4.796283721923828, -3.478503465652466, -1.0037827491760254,
-1.0992263555526733, -4.662079811096191, -5.379837512969971,
-2.644301414489746, -2.6402742862701416, 2.3194973468780518,
3.3649306297302246, 2.1754682064056396, -1.1114686727523804,
1.0688281059265137, -3.577500104904175, -4.088151454925537,
0.3891354203224182, 1.0180394649505615, -0.9323839545249939,
2.305504083633423, -6.495588302612305, 1.6162891387939453,
-1.5268229246139526, -2.207836389541626, 1.7975678443908691,
0.41561201214790344, -9.29943561553955, -2.272061824798584,
-3.0698161125183105, 1.1779743432998657, -1.320559024810791,
-0.91138756275177, -8.288350105285645, -1.1483521461486816,
-6.793163776397705]]"
}
```


6. Other security considerations

The lambda function is going to invoke deploy endpoint. But it is only invoke endpoint if it has the proper security policies attached to it. This one only need two security policy.

- Basic Lambda function
- SageMaker endpoint permissions.

Common security vulnerabilities:

- **Overly Permissive Policies**: Attaching "FullAccess" policies to IAM roles can create unnecessary risks, as these roles grant excessive permissions. Instead, it is best to follow the principle of least privilege by granting only the permissions strictly required for Lambda and SageMaker to perform their tasks.
- **Inactive or Obsolete Roles**:.Roles that are no longer actively used can introduce vulnerabilities. For instance, these roles might belong to team members who have left the project or organization, increasing the likelihood

of neglect or misuse. Regularly auditing and removing inactive roles can significantly reduce this risk.

- **Lack of Monitoring and Rotation**: IAM roles should have clear monitoring policies, such as enabling CloudTrail logs to track usage. Additionally, rotating credentials and access keys associated with roles can mitigate risks from leaked or exposed credentials.

7. Concurrency and auto-scaling

- Concurrency configuration for Lambda: I configured **provisioned concurrency** for your Lambda function with a concurrency level of 2. This ensures that two execution environments are pre-initialized and always ready to handle requests. The Lambda function can handle up to two simultaneous requests without delay due to cold starts. This setup might be suitable for workloads with predictable, low-to-moderate traffic
- Auto-Scaling configuration: I configured the minimum instance count as 1 and the maximum instance count as 4, allowing the endpoint to scale between these limits based on the workload. I also enabled the built-in auto-scaling policy, targeting **30 invocations per instance** as the scaling metric. This means the system will scale out or scale in based on this invocation threshold. The scale-in and scale-out cooldown times were set to 30 seconds, ensuring a sufficient buffer period between scaling actions to stabilize traffic changes.