Operationalizing an AWS ML Project

Notebook instance:

For the Notebook instance, I used ‘ml.m5.xlarge’ because it’s more affordable than most GPU-based instances while still providing enough compute power for general data processing and training smaller models.

To train the model on an EC2 instance, I chose the ‘g4dn.xlarge’ instance. This is one of the most cost-effective options that still offers a GPU, which significantly speeds up training. The GPU allows for efficient matrix operations (through vectorization), which are essential for Deep Neural Nets.

Difference Notebook / EC2:

While the code logic for both environments is mostly similar, there are some key differences. For example, Sagemaker notebooks make it very easy to set up multi-instance training because AWS manages the underlying infrastructure for you. In contrast, EC2 gives more manual control but requires you to manage more steps yourself. Furthermore, deploying a model to an endpoint is much simpler in Sagemaker than in a pure EC2 setup, where you’d have to handle that part manually.

Lambda function:

The function uses the Sagemaker runtime to invoke the deployed model endpoint. It passes the endpoint name, the content/accepted return type, and the body to the invoke_endpoint() method. The input data is serialized as JSON, and the response is expected in JSON format as well. After receiving the response, it’s decoded and parsed back into a usable format for further processing. The Lambda function returns a standard HTTP response, including the predicted values and some metadata.

The Lambda function returns:

```
{
  "statusCode": 200,
  "headers": {
    "Content-Type": "text/plain",
    "Access-Control-Allow-Origin": "*"
```

},

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

"COntent-Type-In": "LambdaContext([aws_request_id=4282978a-3581-439a-8821-5577788768ef,log_group_name=/aws/lambda/lambda-operationalizing-ml,log_stream_name=2025/04/23/[$LATEST]7c1b377a1a9147bcb5a401e019659931,function_name=lambda-operationalizing-ml,memory_limit_in_mb=128,function_version=$LATEST,invoked_function_arn=arn:aws:lambda:us-east-1:311454078555:function:lambda-operationalizing-ml,client_context=None,identity=CognitoIdentity([cognito_identity_id=None,cognito_identity_pool_id=None])])",

"body": "[[-6.578413009643555, -5.360112190246582, -2.8626160621643066, -2.742889881134033, -6.944191932678223, -8.18106460571289, -2.532848596572876, -1.5910323858261108, -7.777955055236816, -2.492629051208496, -1.8516863584518433, -3.801038980484009, -1.5943701267242432, 1.3407608270645142, -2.6001152992248535, -1.5743176937103271, -7.494557857513428, -4.158069133758545, -5.750823974609375, 0.5179995894432068, -5.810633182525635, -1.0484752655029297, -6.641811370849609, -8.730829238891602, -1.9371569156646729, -8.6724271774292, -2.5087664127349854, -0.7652319669723511, -5.9724507331848145, -3.9978537559509277, -4.478230953216553, -3.3013956546783447, -9.561565399169922, -3.6301934719085693, -8.491765975952148, -8.624963760375977, -4.808907508850098, -4.208901882171631, -2.982218027114868, -4.053686618804932, -3.4872307777404785, -2.3552305698394775, -2.3227932453155518, -3.9935224056243896, -2.2260262966156006, -8.704375267028809, -3.0434906482696533, -0.6313292980194092, -3.13830828666687, -1.0911691188812256, -3.91373348236084, -8.818476676940918, -10.927680969238281, -2.318544626235962, -8.176907539367676, -0.8300705552101135, -4.252604961395264, -5.3475213050842285, -3.869982957839966, -2.824632406234741, -6.977392673492432, -6.60130500793457, -7.915271282196045, -7.003939628601074, -3.1176929473876953, -9.75128173828125, 0.4585524201393127, -9.154943466186523, -3.0139987468719482, -0.22123372554779053, 1.5057463645935059, -5.767995834350586, -6.249962329864502, -7.334660530090332, -5.383360862731934, -5.94774866104126, -5.563496112823486, -3.804082155227661, -8.020214080810547, -3.097028970718384, -0.4139789342880249, -6.959142208099365, 1.3020350933074951, -0.7333729267120361, -4.438535213470459, -7.173788547515869, -2.549084424972534, -7.496339321136475, -3.797830581665039, -2.03141188621521, -7.934144496917725, -7.632876396179199, -7.770069122314453, -

8.753503799438477, -10.11751651763916, -4.658415794372559, -3.9731802940368652, -5.008951187133789, -5.391707897186279, -4.891912460327148, -5.764043807983398, -2.400820732116699, -4.813380718231201, -2.744311809539795, -2.987330675125122, -10.674338340759277, -1.7667014598846436, -2.1184844970703125, -4.0006818771362305, -1.784851312637329, -2.4924843311309814, -1.9711248874664307, -10.34094524383545, -6.145237445831299, -4.4118523597717285, -1.7080047130584717, -9.892416000366211, -1.7550373077392578, -4.7798075675964355, 0.5348824262619019, -3.8346076011657715, -4.979280948638916, -3.4436192512512207, -6.692900657653809, -7.476945400238037, -5.1981201171875, 0.21117901802062988, -1.7645456790924072, -5.716482162475586, -7.6267194747924805, -8.030180931091309, -1.096584677696228, -2.43524169921875]]" }

Security:

Security is often a key concern when deploying ML models. IAM roles should always follow the Principal of Least Privilege – roles should only have the permissions they absolutely need, and nothing more. Broad permissions like 'FullAccess' may be convenient, but are not ideal from a security viewpoint. Additionally, unused roles or outdated policies should be regularly reviewed and – if necessary – removed.

Concurrency and auto-scaling:

For the Lambda function, I chose reserved concurrency of 4 and provisioned concurrency of 2. This setup balances cost and performance, hopefully handling medium levels of traffic with low latency. Since half of the concurrency is provisioned, the cold-start latency should be low enough to work just fine.

For auto-scaling the model endpoint, I used the following settings:

- Min/Max instance count: 1-4
- Target value: 10 simultaneous requests
- Scale-in/Scale-out cooldown: 30s each

This configuration provides a good user experience with low latency during usage spikes, while also being cost-efficient by only scaling when absolutely necessary.