Project Writeup: Operationalizing an AWS ML Project

1. **SageMaker Instance Type Selection:**
   I selected ml.t3.medium instance for my sagemaker notebook. I chose the same because it is the ideal choice for development environments as it is cost effective and has a fast launch time, also since the notebook is mainly for writing code and only triggering the jobs rather than heavy model training tasks, this is ideal.
   For hyperparameter optimization and single instance training jobs, I selected ml.g4dn.xlarge instance as this is the ideal choice for training CNNs since they reduce training time and increase efficiency.
   Multi instance training was triggered with 4 instances and since we need to show distributed training using multiple general-purpose instances handling the workload concurrently, I used ml.m5.2xlarge.

2. **EC2 Instance Type Justification:**
   I chose g4dn.xlarge instance in a Deep Learning OSS Nvidia Driver AMI GPU PyTorch 2.9 (Amazon Linux 2023) Image for ec2 training because the task at hand was deep learning model training and GPU instances like the g4dn family use specialized hardware effective for matrix multiplication thus in turn making them more efficient and time saving for CNN model training as compare to other CPU instances.

3. **Differences Between SageMaker and EC2 Code:**
   The code used in sagemaker relies heavily on the high level sagemaker python SDK. For the infrastructure management it handles the data from S3 using env variables and model artifact storage while EC2 code is purely a python script which requires manual setup for the data and infrastructure and the model is explicitly saved to a local directory rather than automatically uploading the artifact to s3, and here rather than using sagemaker specific methods like tuner or sagemaker.pytorch, pytorch module functions like torch, torch.nn, torch.optim are used.

4. **Lambda Function Structure:**
   The Lambda function is using boto3 to invoke the deployed endpoint. The input data, that is the image URL is passed as the body of the request as a JSON object with the content type application/Json. It also follows the best practice for the API return with a proper structure containing statusCode (200 for success), Headers (content type and access control), type-result and body (Containing prediction result).

5. **Lambda Function Test Result:**

[[0.3124537765979767, 0.2254045009613037, 0.007378812879323959, 0.22388780117034912, 0.3272993266582489, 0.232872873544693, -0.0521014966070652, 0.20868180692195892, -0.3722832202911377, 0.039797406643629074, 0.1101515144109726, 0.2240876853466034, -0.06571207195520401, 0.25555822253227234, 0.411977618932724, 0.15205027163028717, 0.35454657673835754, 0.2201269268989563, -0.055789634585380554, 0.26093029975891113, 0.16541366279125214, -0.07853928208351135, 0.11542056500911713, 0.11212440580129623, -0.39705994725227356, -0.2519780397415161, 0.268739253282547, -0.42293474078178406, 0.3988823890686035, -0.043938055634498596, 0.02863544411957264, 0.11632513254880905, -0.24722792208194733, 0.16129839420318604, 0.028306614607572556, 0.28955549001693726, 0.05971509963274002, 0.06952015310525894, 0.2764989733695984, -0.0009909180225804448, 0.3519112467765808, 0.07596338540315628, -0.05570568889379501, 0.2968565821647644, -0.1352202445268631, 0.21830181777477264, -0.031574465334415436, 0.04704910144209862, -0.11563912034034729, -0.07163916528224945, 0.18225574493408203, -0.1459144651889801, -0.06984592229127884, 0.07804059237241745, -0.05678929015994072, 0.27736374735832214, 0.17562632262706757, -0.10658738017082214, -0.09044195711612701, 0.21455036103725433, 0.09106216579675674, 0.14125676453113556, -0.10160854458808899, -0.26580357551574707, -0.1531292200088501, -0.386445015668869, -0.3752899467945099, 0.2737617790699005, 0.054105497896671295, -0.08135149627923965, 0.21905097365379333, -0.025552712380886078, -0.20313315093517303, -0.29287928342819214, -0.14965826272964478, 0.01337929256260395, -0.06836339086294174, -0.32252416014671326, 0.07705368846654892, -0.12384490668773651, 0.13973748683929443, 0.19120875000953674, 0.028000032529234886, -0.21641312539577484, -0.4034680426120758, -0.11128730326890945, 0.07247260957956314, -0.10028360038995743, 0.1498919129371643, 0.0668892040848732, 0.01947346143424511, -0.21156883239746094, -0.3119734823703766, -0.04212578386068344, -0.022167690098285675, -0.2048814594745636, 0.0529768168926239, -0.1606474071741104, -0.22173790633678436, -0.19805923104286194, -0.05227692797780037, -0.6424115300178528, 0.06605778634548187, -0.08221089839935303, -0.43296676874160767, 0.006667513865977526, -0.118065245449543, -0.6564592123031616, -0.11383760720491409, -0.4754452109336853, -0.1492895931005478, 0.01957554556429386, -0.33588752150535583, -0.30078840255737305, 0.20986732840538025, -0.4705118238925934, 0.08010596036911011, 0.007129895966500044, -0.36216309666633606, -0.261739581823349, -0.6106287240982056, -0.31152868270874023, -0.22535105049610138, -0.06825900077819824, -0.4748227894306183, -0.5376924872398376, -0.17956672608852386, -0.4404749572277069, -0.15169930458068848, -0.1265004575252533, -0.4368908703327179, -0.6272271275520325, -0.4550662934780121]]

6. **Security Analysis:**

   I checked all the IAM roles and policies in my workspace and saw some vulnerabilities like the overly permissive policies (mine uses AmazonSageMakerFullAccess which violates the principle of least privilege, thus in turn having a risk of allowing an attacker to have control over all sagemaker resources) and Broad resource scope (some of the policies are applying to resource of '*', usually production environments must have restricted permissions to specific resources).

7. **Concurrency and Auto-Scaling Configuration:**
   I configured provisioned concurrency for the lambda function with a setting of 3. My project requires low latency response for real time use. By configuring 3 provisioning concurrency, it is ensured that there are 3 environments are initialized and warm, this preventing cold start latency that occurs when the function is called after a period of inactivity.

   I configured the Autoscaling for the endpoint with 1 minimum instance, 3 maximum instance, 25 target invocations per instance, 180 seconda scale out cooldown and 300 seconds scale in cooldown. The target of 25 target invocations per instance ensures that the system scales out before a single instance gets overwhelmed, the shorter scale cooldown allows the system to react reasonably to traffic spike and a longer scale in cooldown prevents rapid adding and removing of instances thus making sure traffic has genuinely reduced before terminating the resources.