Choice of Notebook Instance:

For the notebook instance I choose the AWS ml.m5.xlarge. The AWS ml.m5.xlarge instance is one of the many instance types available on AWS that is specifically designed for machine learning workloads.

The ml.m5.xlarge instance offers a balance of compute, memory, and networking resources, making it suitable for a wide range of machine learning tasks.

Here are some key features and benefits of the ml.m5.xlarge instance:

1. Compute Power: The ml.m5.xlarge instance provides a good amount of compute power, making it suitable for running machine learning models that require moderate computational resources.
2. Memory: It offers a decent amount of memory, which is important for handling large datasets or running memory-intensive algorithms.
3. Network Performance: The instance type provides high network performance, enabling fast data transfer between instances and other AWS services. This is particularly useful when working with large datasets or distributed machine learning frameworks.
4. Cost-Effective: The ml.m5.xlarge instance is cost-effective compared to some other instance types. It strikes a balance between performance and cost, making it a popular choice for many machine learning workloads

Description and Reasoning for EC2 Instance:

I also choose the ml.m5.xlarge for the EC2 instance for the reasons above and for the network performance which is described below:

For EC2 instances:

Network Performance: EC2 instances often require good network performance for data transfer between instances and other AWS services. The ml.m5.xlarge instance provides high network performance, facilitating efficient communication.

The ml.m5.xlarge instance is a versatile choice for both SageMaker notebook instances and EC2 instances due to its balanced resources and cost-effectiveness.

Lambda test output:

Test Event Name
lambda_function_test_1

Response

{
  "statusCode": 200,
  "headers": {
    "Content-Type": "text/plain",
    "Access-Control-Allow-Origin": "*"
  },
  "type-result": "<class 'str'>",
  "COntent-Type-In": "<__main__.LambdaContext object at 0x7fce1ff4fb80>",
  "body": "[[0.32303309440612793, 0.43427303433418274, 0.7346514463424683,
0.41949740052223206, 0.7398744225502014, 0.6043280959129333, 0.4677397310733795,
0.9804325103759766, -0.2377524971961975, 0.3775336742401123, 0.49289074540138245,
0.9390873908996582, 0.42134881019592285, 0.5386292338371277, 0.43434005975723267,
0.5506890416145325, 0.05780079588294029, -0.24294757843017578, 0.6838098168373108,
0.8392728567123413, 0.5001637935638428, 0.13068601489067078, 0.5291597843170166,
0.22412922978401184, 0.06394549459218979, -0.04505985602736473,
0.49245744943618774, 0.1458319127559662, 0.3799089193344116, 0.1608961969614029,
0.5580585598945618, 0.7963208556175232, 0.12813441455364227, 0.5728402733802795,
0.35184189677238464, 0.2106768786907196, 0.524625301361084, 0.3412991464138031,
0.5806666016578674, -0.1603090912103653, 0.24978488683700562, 0.5636273622512817,
0.03612860292196274, 0.24845433235168457, 0.4253509044647217, 0.6912871599197388,
0.1952333003282547, 0.16979077458381653, -0.16326461732387543, 0.33599165081977844,
0.10863150656223297, 0.012270927429199219, 0.6500461101531982, 0.3805588483810425,
0.4388788938522339, 0.798782467842102, 0.6591728329658508, 0.6324992775917053,
0.11446990072727203, 0.5448489785194397, 0.7983440160751343, 0.14579035341739655,
0.6746119856834412, 0.23397456109523773, 0.2815587520599365, -0.26202651858329773,
-0.3084242641925812, 0.5174111723899841, 0.019834274426102638, -
0.04875708371400833, 0.4453558325767517, 0.3438114523887634, 0.006522052921354771,
-0.2333928942680359, 0.13286380469799042, 0.592589259147644, -0.01930384151637554,
0.23889046907424927, -0.15798963606357574, -0.2749941945075989, 0.4812035858631134,
0.11813672631978989, 0.5028713345527649, -0.20220081508159637, 0.06583335995674133,
0.5642567276954651, 0.5718668103218079, 0.3168664276599884, 0.4377765655517578,
0.6897336840629578, 0.18809416890144348, 0.3209178149700165, -0.05408415198326111,
-0.004332324489951134, 0.35060086846351624, 0.10981206595897675,
0.3089914619922638, -0.05985371768474579, -0.007153026759624481,
0.14518919587135315, -0.1329272985458374, -0.5716119408607483, 0.4758630096912384,
0.12477868795394897, -0.2501007914543152, 0.12711270153522491, 0.1447596549987793,
-0.2744748592376709, 0.33094435930252075, -0.257753849029541, 0.3553982079029083,
0.6466200947761536, -0.20088942348957062, -0.2154862880706787, 0.290467232465744, -
0.30835914611816406, -0.0195404253900051 1, 0.4015783965587616, -0.165511354804039, -
0.004145502112805843, -0.21788839995861053, -0.330916166305542, -
0.27143317461013794, 0.05789840221405029, 0.043438345193862915, -
0.18053625524044037, 0.4010483920574188, -0.4685504734516144, 0.06431137770414352,
0.07979275286197662, 0.04480786621570587, -0.2870643138885498, -
0.17551589012145996]]"

}

Function Logs
Loading Lambda function
START RequestId: dcc9e784-c1b4-4cb8-b854-ee962a4187cf Version: $LATEST
Context::: <__main__.LambdaContext object at 0x7fce1ff4fb80>
EventType:: <class 'dict'>
END RequestId: dcc9e784-c1b4-4cb8-b854-ee962a4187cf
REPORT RequestId: dcc9e784-c1b4-4cb8-b854-ee962a4187cf Duration: 1464.64 ms Billed Duration: 1465 ms Memory Size: 128 MB Max Memory Used: 76 MB Init Duration: 400.42 ms

Request ID
dcc9e784-c1b4-4cb8-b854-ee962a4187cf

IAM Security Vulnerabilities:

For IAM setups, there are a few potential security risks that can arise from full access policies, old or inactive roles, and roles with policies for functions that are no longer in use. Let's review these three vulnerabilities:

1. Full Access Policies: Granting full access policies to IAM users or roles can be risky because it provides unrestricted access to AWS resources. This means that if an attacker gains access to an IAM user or role with full access, they can potentially compromise or misuse all resources within the AWS account.
2. Old or Inactive Roles: Roles that are no longer in use or have become inactive can pose security risks. If these roles still have permissions attached to them, they can be exploited by attackers.
3. Roles with Unused Policies: Roles that have policies for functions or services that are no longer being used can introduce security vulnerabilities. It is important to remove unused policies.

In the course material we learned ways to mitigate these risks:

- Follow the principle of least privilege and grant only the necessary permissions to IAM users and roles.
- Implement strong password policies and enable multi-factor authentication (MFA) for IAM users.
- Monitor and log IAM activities to detect any suspicious or unauthorized access attempts.

Concurrency and Autoscaling:

1. Traffic Considerations:

- Concurrency: Concurrency refers to the number of simultaneous requests or tasks that a system can handle. Higher concurrency allows for more simultaneous requests to be processed, for handling spikes in traffic. With higher concurrency permissions the system can scale and handle increased demand.

- Auto-scaling: Auto-scaling automatically adjusts the resources allocated to a system (like our lambda function) based on the incoming traffic. By dynamically scaling resources up or down, auto-scaling ensures that the system can meet demand during peak traffic periods and avoid over-provisioning during low traffic periods.

2. Cost Considerations:

- Concurrency: Higher concurrency comes with more resources and a higher cost. As a machine leaning engineer It is important to carefully chose the right amount of concurrency balancing increased performance with cost management.

- Auto-scaling: Auto-scaling can help optimize costs by dynamically adjusting resources based on traffic patterns. It allows for scaling up during peak periods and scaling down during low traffic periods, ensuring that resources are allocated efficiently. This helps avoid over-provisioning and reduces unnecessary costs.

3. Efficiency Considerations:

- Concurrency: Efficiently managing concurrency involves optimizing resource utilization and minimizing response times. Techniques such as load balancing, caching, and optimizing code can help improve concurrency efficiency.

- Auto-scaling: By dynamically scaling resources, auto-scaling helps maintain performance and responsiveness while minimizing resource waste.

References:

This write up was created using assistance from Udacity GPT and the course materials.