UNIVERSITY OF LIMERICK
OLLSCOIL LUIMNIGH

Implementation and considerations of a Stock price prediction on Amazon Web Services (AWS)

Module Assignment for

CS5024 - Theory and Practice of Advanced AI Ecosystems

Student Name: **JAAIE KADAM**

Student ID: **23222441**

Revision Timestamp: 02/05/2024 23:15:12

Abstract

I'm implementing stock price prediction using XG-Boost Algorithm and various AWS services to provide a solution that is suited for actual financial requirements by utilising AWS's dependable infrastructure and AI tools. Motivated by the demand for reliable forecasting tools in finance, we can assist investors in navigating market uncertainties. Precise stock predictions are necessary for more profitable investing, risk mitigation, and smart investment decisions. We harness AWS's scalability and reliability in response to the demand for reliable predictive analytics in financial markets. SageMaker makes model building and deployment easier, while S3 guarantees secure data storage. DynamoDB provides a reliable database solution, and Lambda facilitates smooth integration. Secure access control is guaranteed by IAM roles, and effective prediction communication is facilitated using AWS SNS. CloudWatch provides monitoring and alerts. The ability to use AWS AI services for practical financial applications is demonstrated in this project.

Contents

Introduction

As an Investor in stocks, I'm excited to learn more about how the AWS AI Ecosystem can help solve practical problems into the fields of finance and artificial intelligence. A challenge that particularly interests me is predicting stocks, which is a fundamental role in the financial markets. This means predicting stock prices with accuracy, which is essential for maximising returns and refining investing strategies in a constantly shifting market environment. Precise forecasting is crucial for enhancing investment tactics, controlling hazards, and optimising profits. Using the scalable infrastructure and AI tools offered by AWS, my goal is to create a system that can match the requirements of practical finance applications. The goal of this project is to show how AWS AI services can be used to solve real-world stock prediction problems in an efficient manner, giving investors useful information to help them navigate markets that are unpredictable.

My initial approach involves using AWS's robust infrastructure and machine learning capabilities to utilise the XGBoost algorithm, which is well-known for its efficiency and performance in predictive modelling jobs. Popular for its accuracy and scalability in regression and classification problems, XGBoost is a great option for stock prediction. I'll be diving into this project using Amazon SageMaker, a feature-rich machine learning tool, which is part of the AWS AI Ecosystem. SageMaker streamlines the entire process by offering a seamless platform for training, deploying, and developing models.
My approach starts with preprocessing historical stock data, using the supervised infrastructure of SageMaker to train an XGBoost model, and then deploy the trained model to generate a prediction endpoint. By incorporating this endpoint into my model, I hope to provide subscribers (investors) with timely and reliable one day ahead stock price forecasts.

To ensure scalability and reliability, I will use a variety of AWS services. The secure and long-lasting storage of historical stock data will be achieved through the utilisation of Amazon S3. DynamoDB, a NoSQL database renowned for effective data management, will host AWS Lambda functions to manage backend tasks like data retrieval, prediction execution, and output storing. To protect sensitive data and resources, IAM roles will be used to enforce secure access control and permissions management across all AWS services.

CloudWatch will monitor the deployed model's performance and provide real-time insights into its behaviour to ensure that it performs at its best. Furthermore, AWS API Gateway will function as an interface to provide smooth communication and integration between the prediction endpoint and external systems. This will improve usability and accessibility by making it simple for programmes and outside users to utilise the prediction service. I'm also using AWS SNS (Simple Notification Service) to send timely notifications to investors and stakeholders, to stay updated on important changes in stock predictions.

In this project, I illustrate how the AWS AI Ecosystem effectively addresses real-world stock prediction difficulties. Using AWS's machine learning and cloud architecture, my objective is to create a scalable and effective solution that meets the needs of practical financial applications.

AI Ecosystem Architecture Used

The following diagram illustrates the architecture designed to support the implementation of a powerful AI ecosystem for the Stock prediction model. This architecture combines different AWS components and services to produce a strong framework for prediction challenges. To understand how this architecture supports effective and scalable machine learning solutions, let's examine the main elements and how they interact.

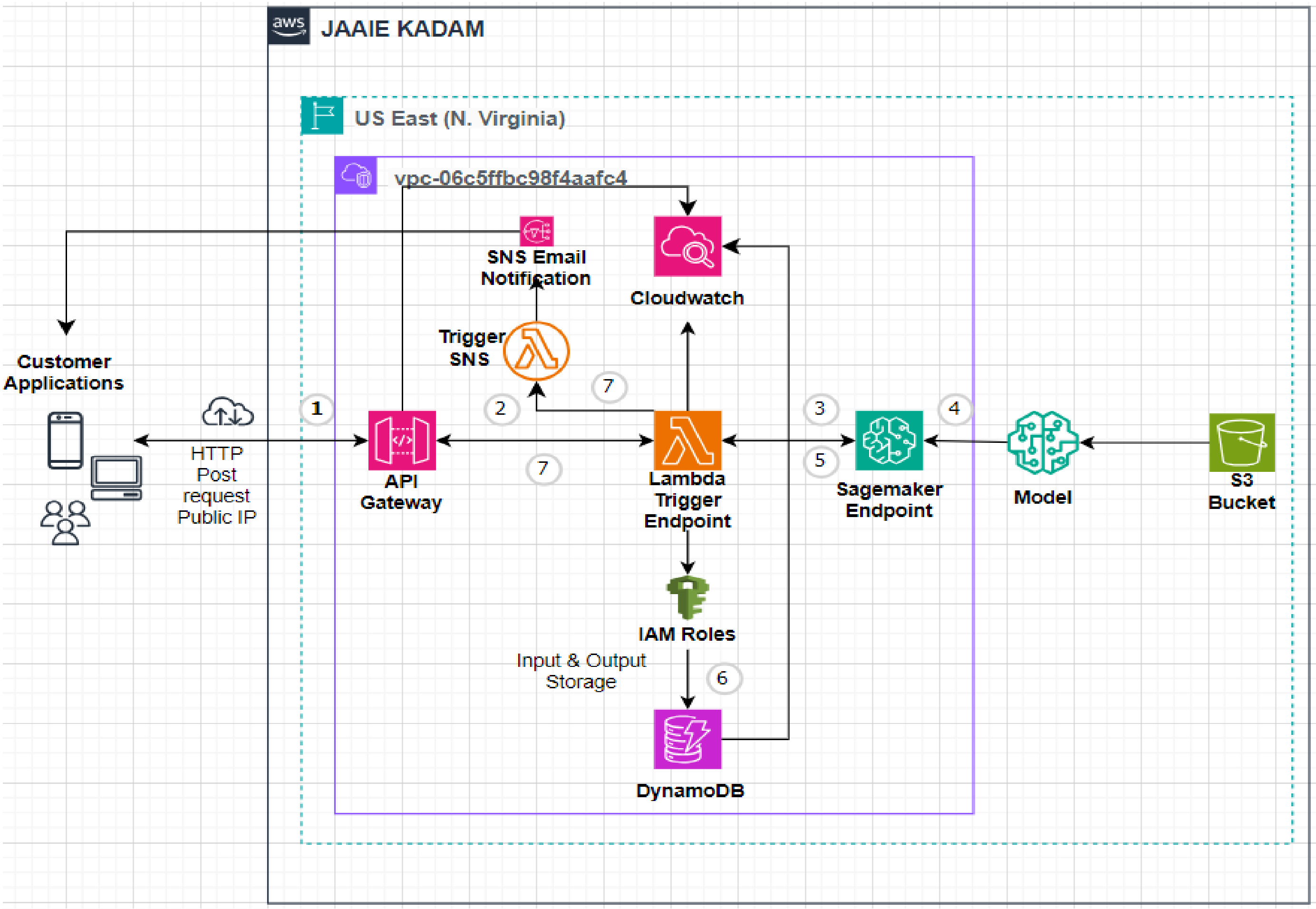


Figure 1: Architecture used to create AI/ML ecosystem for Stock price prediction model.

In our AWS ecosystem, end-users interact with a client application, initiating a process where a REST-style request is sent to an **API Gateway endpoint**. This triggers a **Lambda** function responsible for preprocessing the input data before invoking the model endpoint on **Amazon SageMaker**. The SageMaker endpoint performs inference on the model (This involves retrieving the model through **S3 bucket** and loading it into the memory for execution, generating predictions or outputs based on the model's input features.), and the Lambda function then receives the inference result. After the prediction is done, the output is saved in a **DynamoDB** table for examination together with the input data and timestamps. After that, the Lambda function maps the result to a response, which is sent back to the client via 2 ways: API Gateway with a JSON response and through **SNS** with email notification. This design makes sure that user requests are handled effectively by orchestrating response mapping, model execution, and data preprocessing. The system is also scalable, making use of the capacity of DynamoDB, the rate-limiting of API Gateways, and the scalable nature of Lambda functions to handle different workloads. **IAM roles** with restricted access and **CloudWatch** for extensive monitoring and logging is also used, to ensure system reliability and data security.

Rationale behind choosing the above services for stock price prediction: -

1. **Amazon Sagemaker: -** It has an integrated instance of a Jupyter notebook so you can quickly access your data sources for investigation and analysis. SageMaker facilitates easy interaction with other AWS services in my architecture, such as AWS Lambda for managing the model deployment process and Amazon S3 for storing training data and model artefacts.
2. **S3: -** Amazon S3 is an ideal choice for this model since any files, models, input data for training models, etc. in a very secure, durable, and scalable manner. It's scalability effectively handles massive volumes of data, and its reliability guarantees that the data will be accessible when needed.

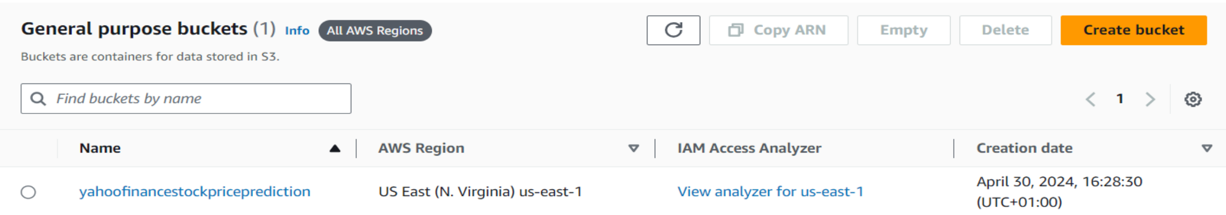


Figure 2: S3 Bucket for Stock Price Prediction Model.

3. **Lambda: -** Lambda was chosen over EC2 for this project due to its serverless architecture. For any volume of traffic, Lambda precisely and automatically distributes compute execution power and executes your code in response to incoming requests or events.

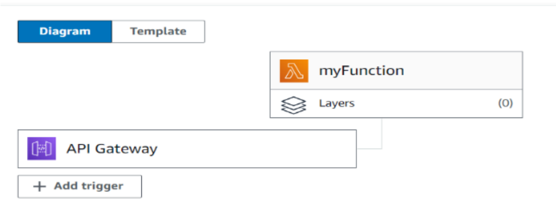


Figure 3: Lambda Diagram

4. **API Gateway: -** For my project, I decided to use API Gateway since it makes it easier to access the stock prediction model that is hosted on AWS. Here, I've sent the request to the server for processing using the POST method.

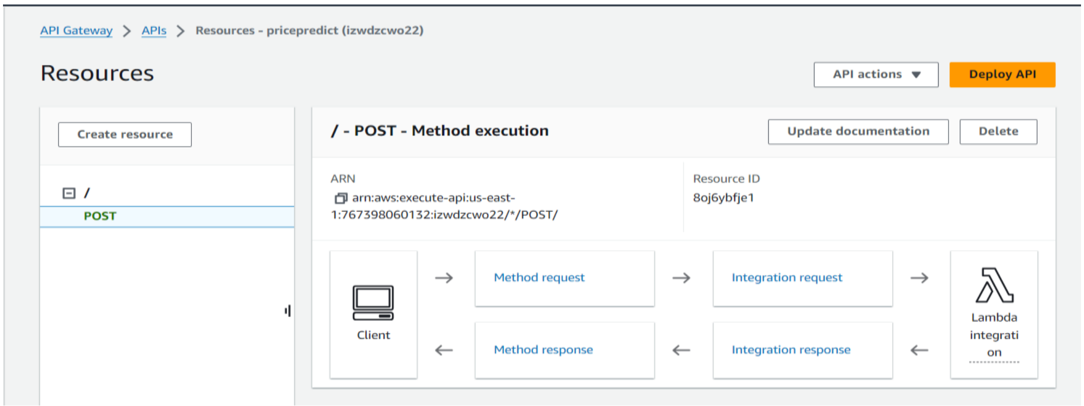


Figure 4: API Gateway for Stock Prediction Service

5. **Amazon SNS: -** I chose Amazon SNS (Simple Notification Service) for this project because it efficiently sends timely notifications to subscribers (investors and stakeholders).

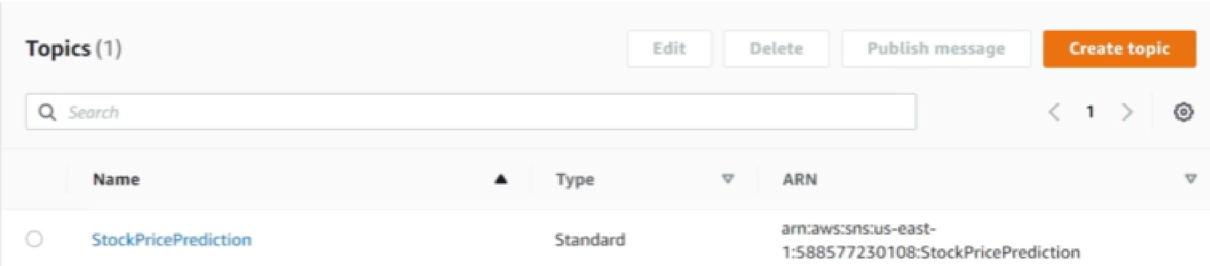


Figure 5: Amazon SNS Topic Creation

6. **CloudWatch: -** Because CloudWatch provides real-time insights into the performance of the deployed model and strict monitoring, I selected it for this project. I can monitor important metrics with CloudWatch to make sure everything is running smoothly and to detect problems quickly.

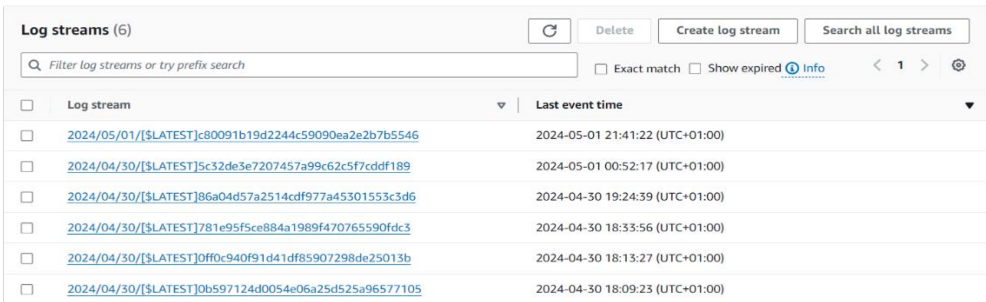
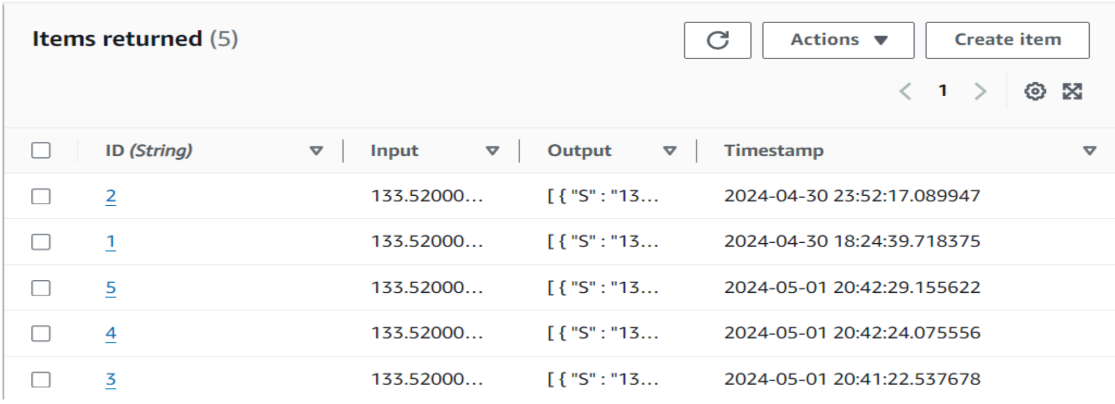


Figure 6: CloudWatch Logs

7. **DynamoDB: -** DynamoDB's scalable and effective NoSQL database features make it the best option for this project. The input text, predicted values, and timestamps are recorded in the 'HISTORY' database, which makes it easier to analyse user inputs and model performance.

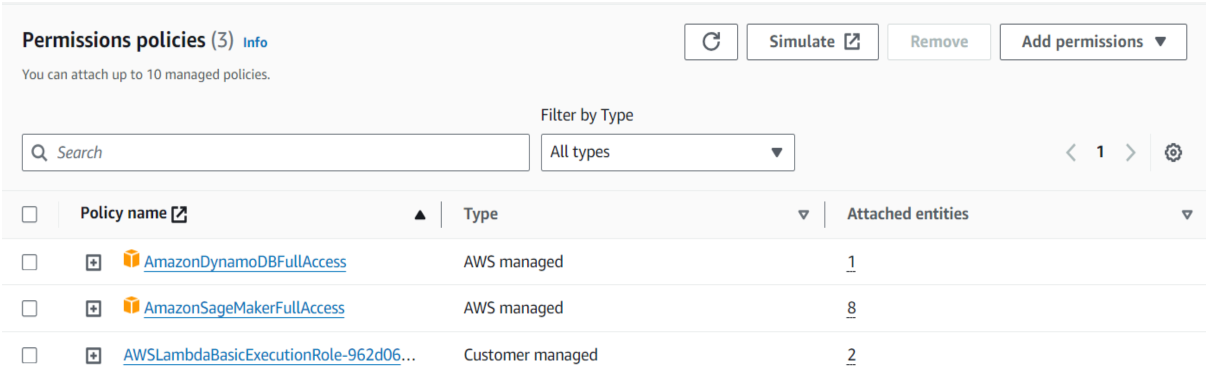


Items returned (5)

ID (String)	Input	Output	Timestamp
2	133.52000...	[{ "S" : "13...	2024-04-30 23:52:17.089947
1	133.52000...	[{ "S" : "13...	2024-04-30 18:24:39.718375
5	133.52000...	[{ "S" : "13...	2024-05-01 20:42:29.155622
4	133.52000...	[{ "S" : "13...	2024-05-01 20:42:24.075556
3	133.52000...	[{ "S" : "13...	2024-05-01 20:41:22.537678

Figure 7: "HISTORY" table entries.

8. **IAM Roles: -** This is crucial to any AWS architecture since it manages permissions and offers safe access control for AWS services. The following are some of my architecture roles.



Permissions policies (3) Info

You can attach up to 10 managed policies.

Policy name	Type	Attached entities
AmazonDynamoDBFullAccess	AWS managed	1
AmazonSageMakerFullAccess	AWS managed	8
AWSLambdaBasicExecutionRole-962d06...	Customer managed	2

Figure 8: IAM Roles

Model Description

The code for stock price prediction was adapted from (Academy, 2022)

For this project, I have used the **XGBoost algorithm**, which is available in Amazon Sagemaker as a built-in algorithm. Located within the Sagemaker library of algorithms, XGBoost is a widely used algorithm for stock predictions because of it's ability to handle large datasets, capture intricate patterns, and deliver accurate predictions.

Boto3 is also used, which is an AWS SDK for Python which allows you to build, edit, and delete AWS resources directly from your Python scripts and allows you to easily link your Python application/library/script with AWS services like Amazon S3, Amazon EC2, Amazon DynamoDB, and more.

For the **Dataset**, I have used the Yahoo Finance API via the 'yfinance' library to access historical stock price data for Apple Inc. (AAPL) over a specific time frame, from January 1, 2019, to January 1, 2021. The dataset comprises various attributes such as the opening price, closing price, highest and lowest prices, and trading volume for each trading day within this period.

	Date	Open	High	Low	Close	Adj Close	Volume
0	2019-01-02	38.722500	39.712502	38.557499	39.480000	37.845039	148158800
1	2019-01-03	35.994999	36.430000	35.500000	35.547501	34.075394	365248800
2	2019-01-04	36.132500	37.137501	35.950001	37.064999	35.530045	234428400
3	2019-01-07	37.174999	37.207500	36.474998	36.982498	35.450970	219111200
4	2019-01-08	37.389999	37.955002	37.130001	37.687500	36.126770	164101200
...	...	...	...	...	...	...	...
500	2020-12-24	131.320007	133.460007	131.100006	131.970001	129.514450	54930100
501	2020-12-28	133.990005	137.339996	133.509995	136.690002	134.146683	124486200
502	2020-12-29	138.050003	138.789993	134.339996	134.869995	132.360504	121047300
503	2020-12-30	135.580002	135.990005	133.399994	133.720001	131.231918	96452100
504	2020-12-31	134.080002	134.740005	131.720001	132.690002	130.221069	99116600

505 rows × 7 columns

Figure 9: 'Yahoo Finance' Dataset

1. **Data Preprocessing: -** In order to prepare the obtained data for model training, unnecessary columns are removed, and columns are rearranged. The data is divided into features and targets, with the target column ('Close' price) moving to the first position. By doing this, it is made sure that the model is trained to forecast stock values based on historical data.

	Target	Open	High	Low	Close	Volume
0	35.994999	38.722500	39.712502	38.557499	39.480000	148158800
1	36.132500	35.994999	36.430000	35.500000	35.547501	365248800
2	37.174999	36.132500	37.137501	35.950001	37.064999	234428400
3	37.389999	37.174999	37.207500	36.474998	36.982498	219111200
4	37.822498	37.389999	37.955002	37.130001	37.687500	164101200
...	...	...	...	...	...	...
499	131.320007	132.160004	132.429993	130.779999	130.960007	88223700
500	133.990005	131.320007	133.460007	131.100006	131.970001	54930100
501	138.050003	133.990005	137.339996	133.509995	136.690002	124486200
502	135.580002	138.050003	138.789993	134.339996	134.869995	121047300
503	134.080002	135.580002	135.990005	133.399994	133.720001	96452100

Figure 10: Tailored Dataset with Target column.

Finally, to avoid bias in the training process, the dataset is randomised and separated into training and testing sets, with 80% of the data designated for training and 20% for testing.

2. **Uploading Data to S3: -** The training and testing datasets are then saved in CSV format and uploaded to an S3 bucket for use during model training. The folders generated for training, validation, and output are shown below. When the model is constructed with parameters, it will be stored in the output folder.

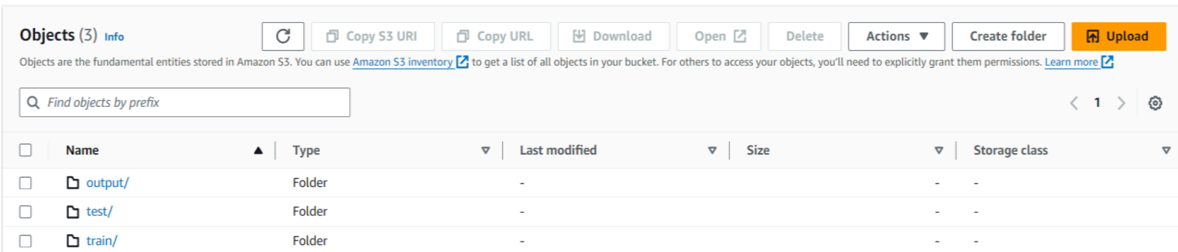


Figure 11: S3 Train, Test, Output Folders

Next, we configure a SageMaker Estimator for training an XGBoost model. It starts by finding the appropriate XGBoost container image for the AWS region, using “image_uris.retrieve” . Then it specifies hyperparameters. The trained model is stored in Amazon S3 via an output path. Next, the code creates a SageMaker Estimator object (estimator = sagemaker.estimator.Estimator()), which includes parameters such as the container image, hyperparameters, instance type, and output path. Spot instances are employed to save money, and timeout configurations control job runtime.

3. **Output & KPI: -** For regression models, such as those used in stock price prediction, generally RMSE is used as an evaluation metric as it directly measures the model's accuracy in predicting the stock prices. Better performance is indicated by a lower RMSE, which shows that the model's predictions are closer to that of the actual stock prices.

   Hence, I have chosen Root-Mean-Squared-Error (RMSE) as a means of KPI for evaluating the effectiveness of the model.

```
[988]#011train-rmse:0.66464#011validation-rmse:1.58349
[989]#011train-rmse:0.66466#011validation-rmse:1.58344
[990]#011train-rmse:0.66465#011validation-rmse:1.58346
[991]#011train-rmse:0.66466#011validation-rmse:1.58369
[992]#011train-rmse:0.66469#011validation-rmse:1.58375
[993]#011train-rmse:0.66466#011validation-rmse:1.58369
[994]#011train-rmse:0.66472#011validation-rmse:1.58380
[995]#011train-rmse:0.66476#011validation-rmse:1.58386
[996]#011train-rmse:0.66483#011validation-rmse:1.58396
[997]#011train-rmse:0.66487#011validation-rmse:1.58400
[998]#011train-rmse:0.66469#011validation-rmse:1.58375
[999]#011train-rmse:0.66471#011validation-rmse:1.58378

2024-04-30 16:27:34 Uploading - Uploading generated training model
2024-04-30 16:27:34 Completed - Training job completed
Training seconds: 128
Billable seconds: 55
Managed Spot Training savings: 57.0%
```

Figure 12: Training Output

an RMSE value of around 32.65 suggests that, on average, the model's predictions deviate from the actual stock prices by approximately $32.65 to $32.72.

```
Train RMSE: 32.65014479032582
Validation RMSE: 32.721638968323084
```

Figure 13: KPI for stock prediction model evaluation.

Once the training is complete, the model is deployed as an endpoint using SageMaker.

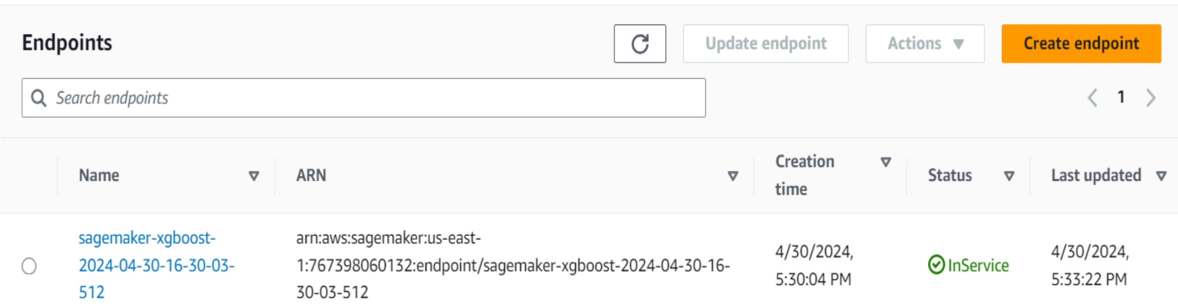


Figure 14: Endpoint for stock prediction.

4. **Inference: -** The code used, is using 3 methods of making predictions using the deployed model:

    i. Directly sending serialized input to the endpoint and decoding the response.

    ii. Using SageMaker's CSVSerializer to serialize the input.

    iii. Implementing a Lambda function to handle inference requests.

5. **Testing Inference: -** It involves utilising a sample input to test the Lambda function and sending a POST request with JSON data to an API endpoint to generate predictions. The model endpoint is called by the lambda function, which then generates predictions based on the input and then those predictions are sent to the users via email.

```python
import boto3
ENDPOINT_NAME = 'sagemaker-xgboost-2024-04-30-16-30-03-512'
runtime = boto3.client('runtime.sagemaker')

def lambda_handler(event, context):
    inputs = event['data']
    result = []
    for input in inputs:
        serialized_input = ','.join(map (str, input))
        response = runtime.invoke_endpoint (EndpointName=ENDPOINT_NAME,
                                            ContentType='text/csv',
                                            Body = serialized_input)
        result.append(response['Body'].read().decode())
    return result
```

Figure 15: Lambda Function

```
▼ Execution results                                  Status: Succeeded | Max memory used: 79 MB | Time: 1193.78 ms
Test Event Name
Test

Response
[
  "132.52377319335938\n",
  "132.52377319335938\n",
  "132.52377319335938\n"
]

Function Logs
START RequestId: 1d6e60df-40fb-487b-a0d3-fb292eca9359 Version: $LATEST
Inside the function
count :  2
Data stored successfully in DynamoDB: {'ResponseMetadata': {'RequestId': '5GD8537AUV8G87M22EFUUHVDBFVV4KQNSO5AEMVJF66Q9ASUAAJG', 'HTTPStatusCode': 200, 'HTTPHeaders': {'s
END RequestId: 1d6e60df-40fb-487b-a0d3-fb292eca9359
REPORT RequestId: 1d6e60df-40fb-487b-a0d3-fb292eca9359  Duration: 1193.78 ms    Billed Duration: 1194 ms    Memory Size: 128 MB Max Memory Used: 79 MB  Init Duration: 48

Request ID
1d6e60df-40fb-487b-a0d3-fb292eca9359
```

Figure 16: Lambda Function Execution Result

From: **AWS Notifications** <no-reply@sns.amazonaws.com>
Date: Wed, May 1, 2024 at 2:14 AM
Subject: eMC Finance - Daily Predictions
To: <jaaiekadam@gmail.com>

Predictions is['132.52377319335938\n', '132.52377319335938\n', '132.52377319335938\n']

--
If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:
https://sns.us-east-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:us-east-1:471112857624:MyTopic:3bf96531-cf6f-4f9f-8dce-89e4bcccc3ed&Endpoint=jaaiekadam@gmail.com

Figure 17: Predictions Email Notification

Scalability Considerations

When implementing the XGBoost model for stock price prediction on AWS, there are a number of scalability issues that must be taken into account to guarantee reliable and efficient performance as the service expands.

a. **Manage Services:** For model deployment and training, using services such as Amazon SageMaker. Developers can concentrate on developing models rather than managing infrastructure since SageMaker takes care of infrastructure provisioning and scaling automatically. (Guide, 2024)
b. **Managing Multiple Requests at Once:** We must make sure our system can manage these requests without sacrificing efficiency to support more users accessing our application at once. Amazon Lambda and AWS API Gateway are good options for efficiently handling and processing several requests at once.
c. **Handling large amounts of Data:** Efficient large-scale data management becomes crucial when data volumes possibly rise. We can effectively store and retrieve data as our application grows thanks to AWS's scalable storage choices, such as Amazon S3 and Amazon DynamoDB.
d. **Fault Tolerance:** Ensuring the reliability of our application is dependent upon maintaining fault tolerance and high availability, particularly in the event of unplanned outages or surges in demand. Resources are automatically adjusted to assure availability and monitor the health of applications with the help of Amazon CloudWatch and AWS Auto Scaling. (Guide, 2024)
e. **Optimising the setting of lambda functions:** To guarantee effective management of fluctuating workloads, Lambda function setup parameters, such as memory allocation and timeout settings, must be optimised.

I chose the following ways to scale my application:

1. **Lambda Auto Scaling: -** AWS Lambda handles scaling automatically, with a default limit of 1000 concurrent requests. However, free tier accounts are limited to 10 concurrent requests. Lambda provisions execution environments for each request, downloading and processing the model. Concurrency values can be adjusted dynamically based on load via CloudWatch. The default Lambda response time is 3 seconds, extended to 1 minute to prevent timeout errors. (Guide, 2023)

Figure 18: Lambda Auto-Scaling

2. **DynamoDB Auto Scaling: -** In order to auto-scale DynamoDB, enable auto-scaling for the table's read and write capacities. (Guide, 2024)
3. **CloudWatch Monitoring: -** I used it in this project to regulate and maintain tabs on resources like SageMaker endpoints, Lambda functions, and API Gateways. CloudWatch was helpful in setting up alerts to track important metrics like Lambda function invocations, API Gateway

latency, and SageMaker endpoint performance, even though EC2 instances were not used in my architecture. (Guide, 2024)

Bibliography

Academy, e. C. (2022). *AWS Machine Learning (SageMaker) Specialization*. Retrieved from https://www.youtube.com/watch?v=3XTmwgjO5DM&list=PLMWIyphKbqfwW4RmL1G29Q_7LOnUV-cE6&index=1

Arunachalam, A. (2022, April 15). Retrieved from From Jupyter Notebook to AWS SageMaker endpoint: Classification and Regression model demo: https://ajay-arunachalam08.medium.com/from-jupyter-notebook-to-aws-sagemaker-endpoint-classification-and-regression-model-demo-157fc2ebc429

Gillela, M. R. (2021, October 18). Retrieved from Serverless workflow for Crypto and Stock price volatility on AWS Cloud: https://manideepreddy1116.medium.com/serverless-workflow-for-crypto-and-stock-price-volatility-on-aws-cloud-f86191eeab2f

Guide, A. D. (2023). *Lambda function Scaling*. Retrieved from https://docs.aws.amazon.com/lambda/latest/dg/lambda-concurrency.html: https://docs.aws.amazon.com/lambda/latest/dg/lambda-concurrency.html

Guide, A. D. (2024). Retrieved from Configure model auto scaling with the console.: https://docs.aws.amazon.com/sagemaker/latest/dg/endpoint-auto-scaling-add-console.html

Guide, A. D. (2024). Retrieved from Using the AWS Management Console with DynamoDB auto scaling: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/AutoScaling.Console.html

Guide, A. D. (2024). Retrieved from Monitor CloudWatch metrics for your Auto Scaling groups and instances: https://docs.aws.amazon.com/autoscaling/ec2/userguide/ec2-auto-scaling-cloudwatch-monitoring.html

Guide, A. D. (2024). Retrieved from Automatically Scale Amazon Sagemaker models: https://docs.aws.amazon.com/sagemaker/latest/dg/endpoint-auto-scaling.html