AI-Powered Data Pipeline for E-commerce Warehouse Demand Forecasting

Implementation Steps

Data Ingestion Using Terraform and Python

Data Processing Using AWS Glue & Lambda

Machine Learning with Amazon SageMaker (for demand forecasting)

**Model Integration & Optimisation**

Deployment & Scaling



Table of Contents

[Implementation Steps 1](#_Toc192949708)

[Step 4: Model Integration & Optimisation 3](#_Toc192949709)

[**Objective:** 3](#_Toc192949710)

[Invoke the Lambda Function using a python script 3](#_Toc192949711)

[Optimise Model Performance 4](#_Toc192949712)

[Implement Model Monitoring and Logging 5](#_Toc192949713)

[Validate Model Predictions 6](#_Toc192949714)

[Verify Automated Model Re-Training 7](#_Toc192949715)

[Monitor API Performance 7](#_Toc192949716)

[Test Auto of SageMaker Endpoint 7](#_Toc192949717)

[Test Cost Optimisation Strategies 8](#_Toc192949718)

[Issues Encountered 9](#_Toc192949719)

[Summary of Key Deliverables 9](#_Toc192949720)

# Step 4: Model Integration & Optimisation

## **Objective:**

* Integrate the trained model into the existing warehouse management system.
* Optimise model performance and automate continuous training.
* Implement monitoring and logging for real-time performance tracking.

## Invoke the Lambda Function using a python script



## Optimise Model Performance

1. Use the code below for Hyperparameter Tuning with SageMaker:



1. Model Re-training on new data

*import boto3*

*sagemaker = boto3.client('sagemaker')*

*def lambda\_handler(event, context):*

 *response = sagemaker.create\_training\_job(*

 *TrainingJobName="warehouse-retrain-001",*

 *AlgorithmSpecification={"TrainingImage": xgb\_model.image\_uri, "TrainingInputMode": "File"},*

 *RoleArn=role,*

 *InputDataConfig=[{"ChannelName": "train", "DataSource": {"S3DataSource": {"S3Uri": train\_path, "S3DataType": "S3Prefix", "S3DataDistributionType": "FullyReplicated"}}}],*

 *OutputDataConfig={"S3OutputPath": "s3://ecommerce-warehouse-ingestion/model-output/"},*

 *ResourceConfig={"InstanceType": "ml.m5.large", "InstanceCount": 1, "VolumeSizeInGB": 10},*

 *StoppingCondition={"MaxRuntimeInSeconds": 3600}*

 *)*

 *return response*

## Implement Model Monitoring and Logging

1. Enable CloudWatch Logs for Model Performance



1. Monitor API Requests & Latency



## Validate Model Predictions

1. Confirm that the SageMaker endpoint is generating accurate demand forecasts by testing the model with sample inputs via Lambda. Replace trigger\_sagemaker\_prediction with your actual Lambda function name.



The response should contain a predicted demand value. If the predictions are unrealistic, check model training data and hyperparameters.

## Verify Automated Model Re-Training

1. Do this to confirm that the Eventbridge Lambda workflow triggers a new training job when expected. You can manually trigger the re-training lambda using the below:



1. Run the command below to view logs:

*cat retrain\_response.json*

A new training job should be initiated in SageMaker. The logs in CloudWatch should indicate a successful start.

## Monitor API Performance

1. Check CloudWatch Logs



1. Find your lambda log group then view recent logs



1. Check API Latency



Logs should show successful requests and inference times and the API should return responses in a reasonable time (<500ms for Lambda, <2s for SageMaker).

## Test Auto of SageMaker Endpoint

1. To ensure scalability, manually increase load on the endpoint. Simulate multiple concurrent requests by using Python to send multiple requests.



Monitor SageMaker instances. SageMaker should scale up if necessary. The response time should remain stable under increased load.

*aws sagemaker describe-endpoint --endpoint-name sagemaker-xgboost-2025-03-03-13-19-36-689*



## Test Cost Optimisation Strategies

1. Check SageMaker Spot Training Usage

*aws sagemaker describe-training-job --training-job-name warehouse-spot-training*

1. Confirm Auto Scaling Adjustments by checking Sagemaker Scaling Adjustments

*aws application-autoscaling describe-scalable-targets --service-namespace sagemaker*

Spot instances should be utilised for cost savings. The system should dynamically adjust capacity.

# Issues Encountered

**Problem 1:** High Latency in Model Predictions. The SageMaker endpoint was experiencing delays in returning predictions. Response times were exceeding 2 seconds, causing bottlenecks in real-time demand forecasting.

**Resolution 1:** Enabled SageMaker Auto Scaling to increase instance count when traffic increased and adjusted endpoint instance type from ml.t3.medium to ml.m5.large for better computational performance. Also used Amazon ElastiCache to cache recent model predictions and reduce repeated computation.

**Problem 2:** Automated Model Retraining Not Triggering on New Data. The EventBridge rule meant to trigger retraining when new data was uploaded to S3 was not activating.

**Resolution 2:** Verified S3 event notifications were correctly configured to send event triggers, ensured Lambda had execution permissions to trigger SageMaker training jobs and manually tested the event-driven retraining workflow using:

**bash**

*aws lambda invoke --function-name warehouse-model-retrain output.json*

**Problem 3:** CloudWatch Logs Not Capturing API Performance Metrics. CloudWatch was not recording inference latency and model accuracy metrics, making it difficult to optimise performance.

**Resolution 3:** Enabled CloudWatch Logs for SageMaker endpoints by updating the endpoint configuration, implemented custom logging within Lambda to track API response times and error rates and configured CloudWatch Alarms to notify when inference time exceeded a 500ms threshold.

**Problem 4:** Cost Management for Model Training Jobs. SageMaker training jobs were running on on-demand instances, leading to higher costs.

**Resolution 4:** Switched to SageMaker Spot Training, reducing costs by up to 70%, configured Auto Scaling for training resources, preventing over-provisioning and monitored SageMaker cost breakdowns via AWS Cost Explorer to track optimisation progress.

# Summary of Key Deliverables

1. **IAM Role Management**: Ensured Lambda had the correct permissions to invoke SageMaker and trigger retraining.
2. **Performance Optimisation**: Reduced model latency by scaling endpoints and caching predictions.
3. **Event-Driven Automation**: Fixed S3 EventBridge triggers to enable seamless model retraining.
4. **Logging & Monitoring**: Improved CloudWatch metrics tracking for model performance insights.
5. **Cost Efficiency**: Leveraged Spot Instances and auto-scaling to optimise SageMaker costs.