National College of Ireland

Configuration Manual

MSc Industrial Internship

Fraud-Proof Finance: Harnessing Cloud Technology and Secure Finance Infrastructure

Jeevitha BS

Student ID: x23162384

School of Computing

National College of Ireland

Supervisor: Kamil Mahajan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Jeevitha BS
Student ID:	X23162384
Programme:	MSc Cybersecurity **Year:** 2024-25
Module:	MSc Industrial Internship
Lecturer:	Dr. Kamil Mahajan
Submission Due Date:	31/08/2025
Project Title:	Fraud-Proof Finance: Harnessing Cloud Technology and Secure Finance Infrastructure
Word Count:	1106 **Page Count:** 15

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Jeevitha BS
Date:	31-8-2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Jeevitha BS
X23162384

1. Introduction

The manual is a comprehensive document describing how to recreate, audit, and comprehend a cloud-based fraud detection system developed with AWS. Compliant to the actual in real-life security standards designed (GDPR, PCI-DSS), this guide is a documentation on one hand that can be submitted to an academic focus and a blueprint of deployment on the other.

2. System Configuration

Services	Details
AWS Region	eu-west-1
IAM Policy	Least privilege, custom roles for Lambda/SageMaker
Web Application Firewall	AWS WAF -Blocks Malicious Traffic
API Gateway	RESTAPI endpoint for transaction ingestion
Lambda Functions	Ingestion and Prediction Function
Kinesis Data Stream	Real time streaming, buffer transactions
SageMaker Endpoint	ML Model Hosting
Data Storage	S3 Bucket
Log Processing	CloudWatch (live), S3 (archival)
Monitoring	CloudWatch Logs, CloudTrail

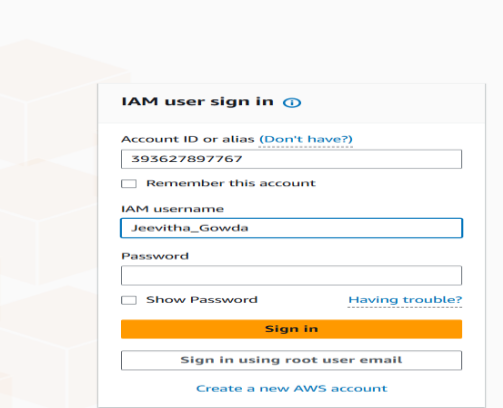
Table 1. Environment setup

3. Detailed Component Setup

3.1 AWS Account Setup

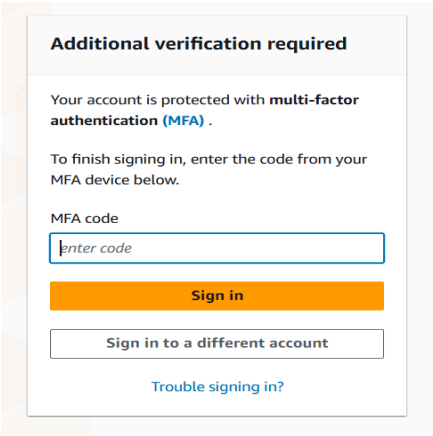
3.1.1 AWS Account Creation:

- A new AWS account was set up for deployment of the fraud detection pipeline.
- As this provides a secure, cloud-based environment with the required access to AWS services



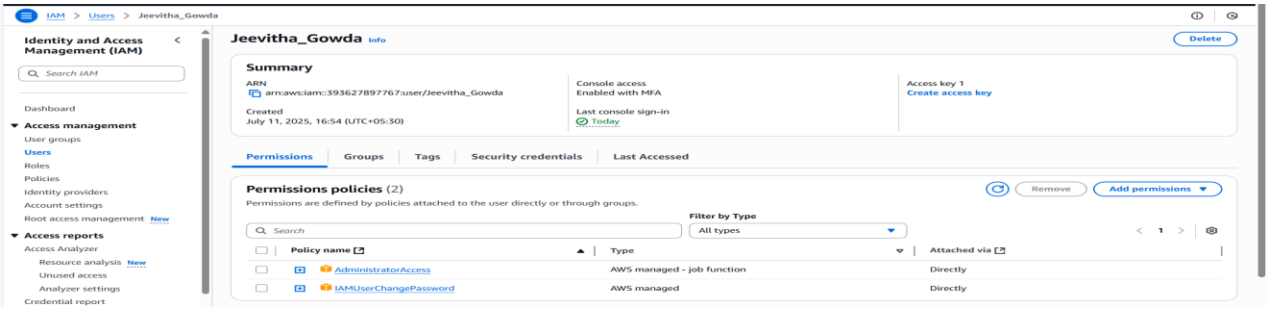
3.1.2 Multi-Factor Authentication (MFA):

- MFA was enabled on the account as it adds a layer of security protects against unauthorized access.



3.1.3 IAM User Creation:

- Separate IAM users were created for developers and administrators and Admin access was assigned.



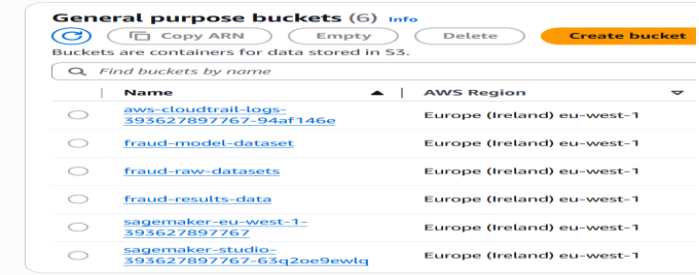
3.1.3 Region Selection:

All the deployment was performed in eu-west-1 (Ireland).

3.2 Storage Setup -Amazon S3 Bucket

3.2.1Dataset Bucket:

- Bucket were created for testing and training datasets and storing the result.



3.2.3 Logs Bucket:

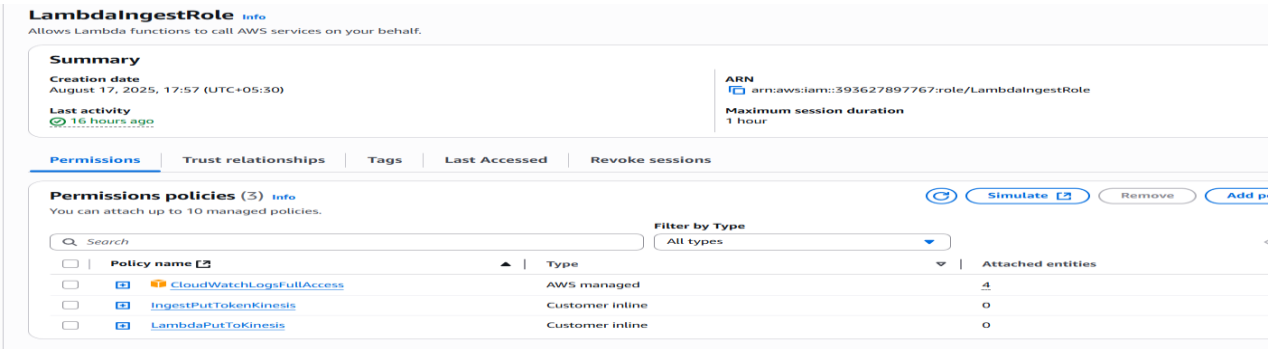
- Bucket fraud-results-data is created for storing the result of fraud detection logs and predictions. As this helps in auditing, tracking model performance

3.3. Identity & Access Management (IAM)

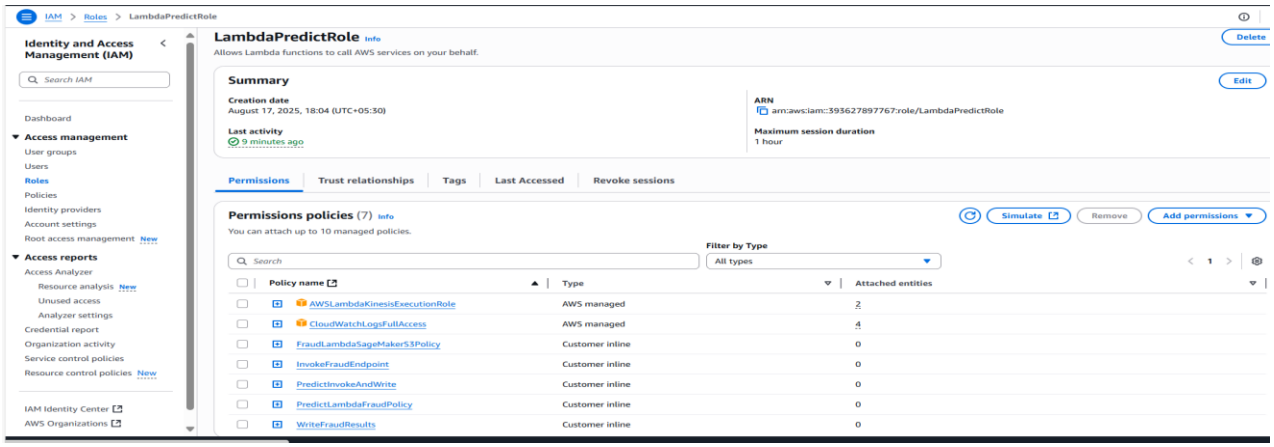
3.3.1 Lambda Role:

Assigned polices to Lambda function for both Ingestion function and Prediction function to pass data between services and send logs to CloudWatch.

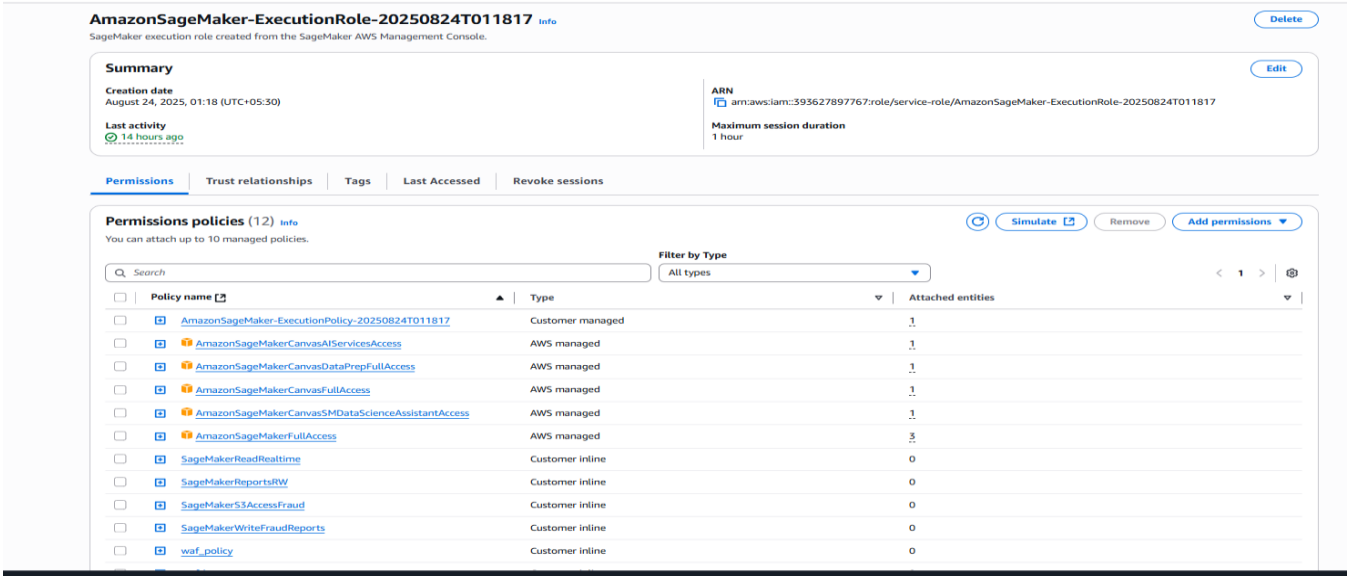
- Ingestion Function



- Prediction Function

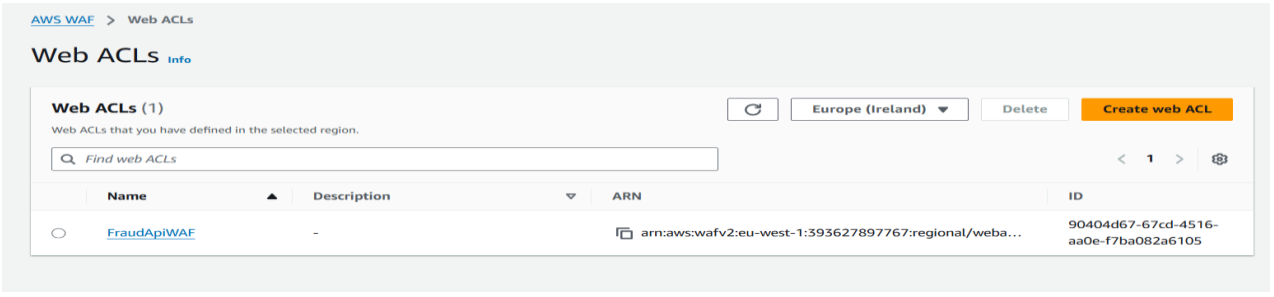


3.3.2 SageMaker Service Role:



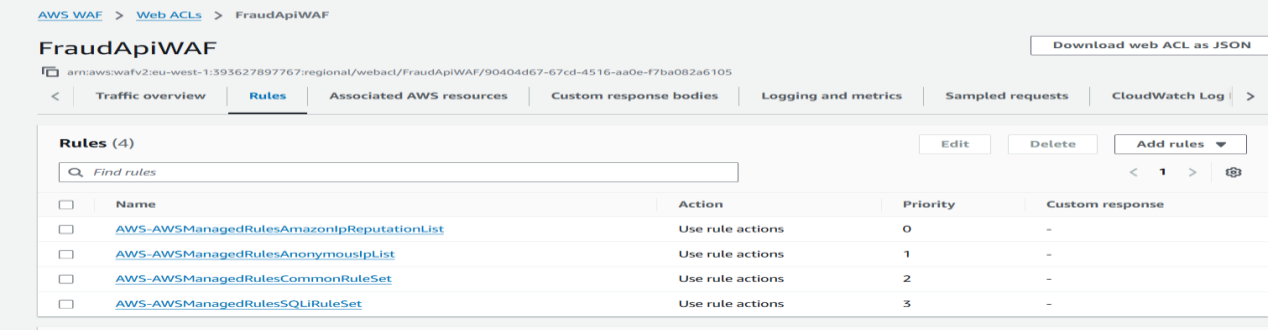
3.4. Web Application Firewall (AWS WAF)

FraudApiWAF, a Web ACL was created on eu-west-1 region to act as first line of defence.



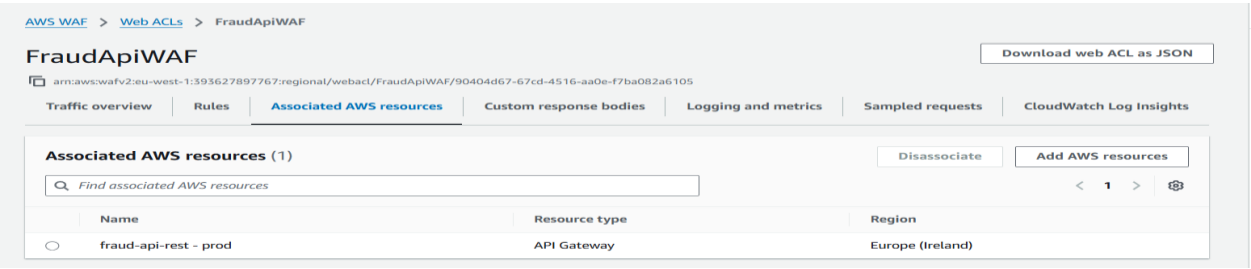
3.4.1 Rules Setup:

- AWS Managed rule was enabled to block the incoming malicious traffic



3.4.2 Integration with API Gateway:

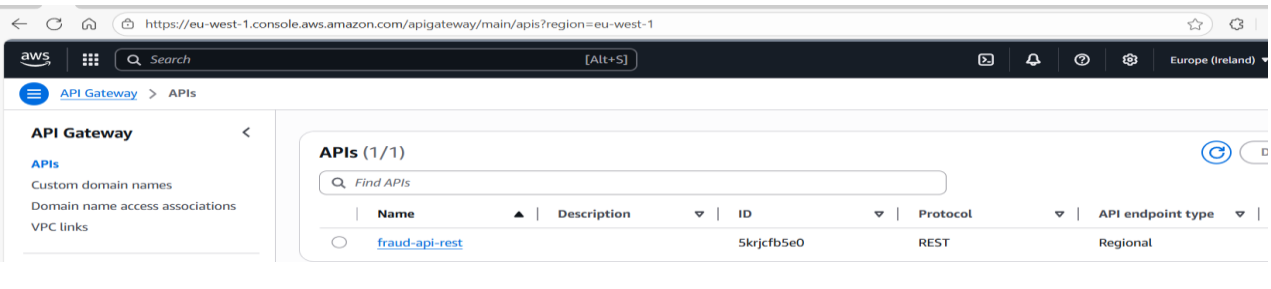
- WAF was associated with API to ensure that malicious requests are blocked.

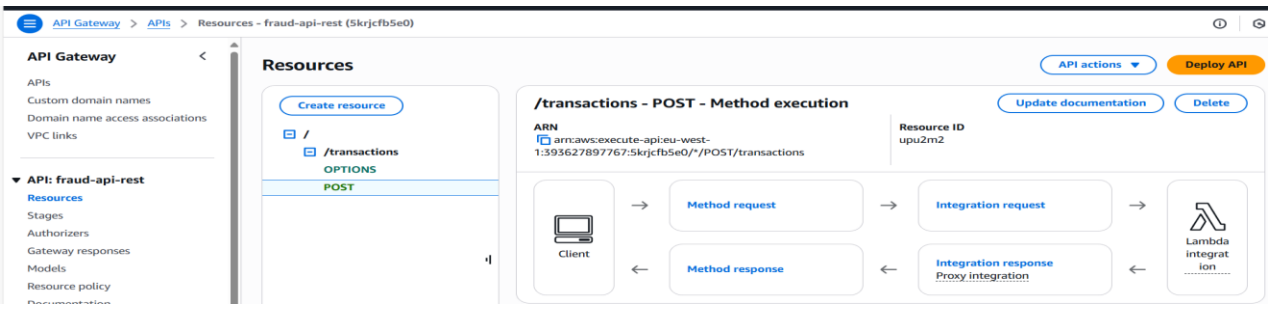


3.5. API Gateway Configuration

3.5.1 Endpoint Creation:

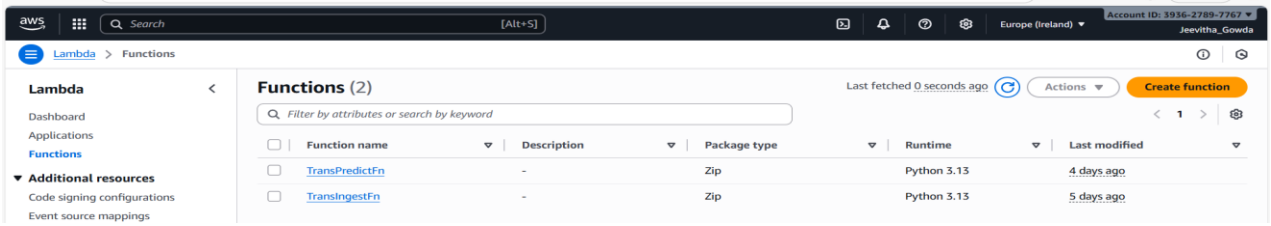
A REST API was created to have a secure entry point for external application to send transaction data with POST method.





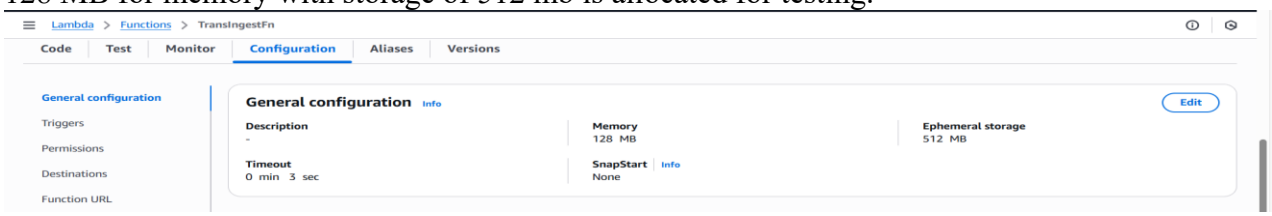
3.6. AWS Lambda Functions

Two functions were created 1. TransIngestFn – 2. TransPredictFn

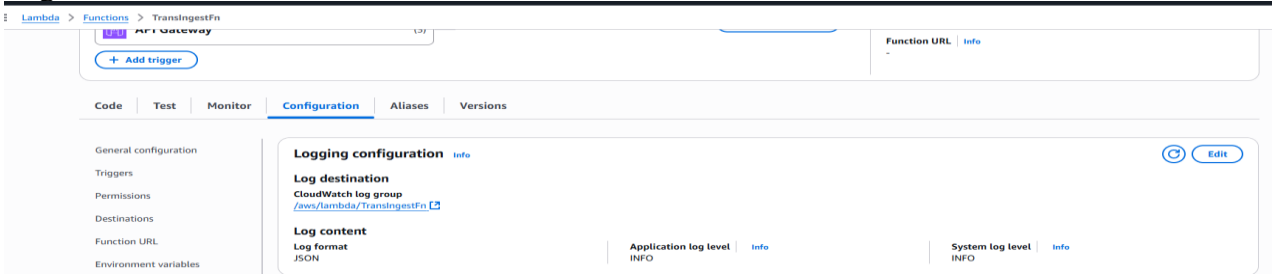


3.6.1 Ingestion Lambda:

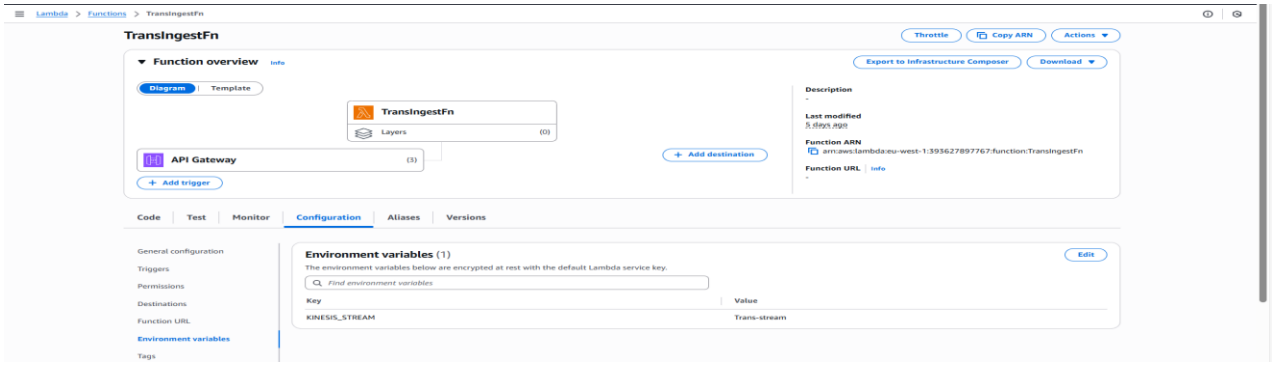
- This function validates all the request payloads and it forwards the events to Kinesis stream as this prepares data for streaming and ensures the data fed is clean.
- 128 MB for memory with storage of 512 mb is allocated for testing.



- Logs are enabled to track and monitor all the activities

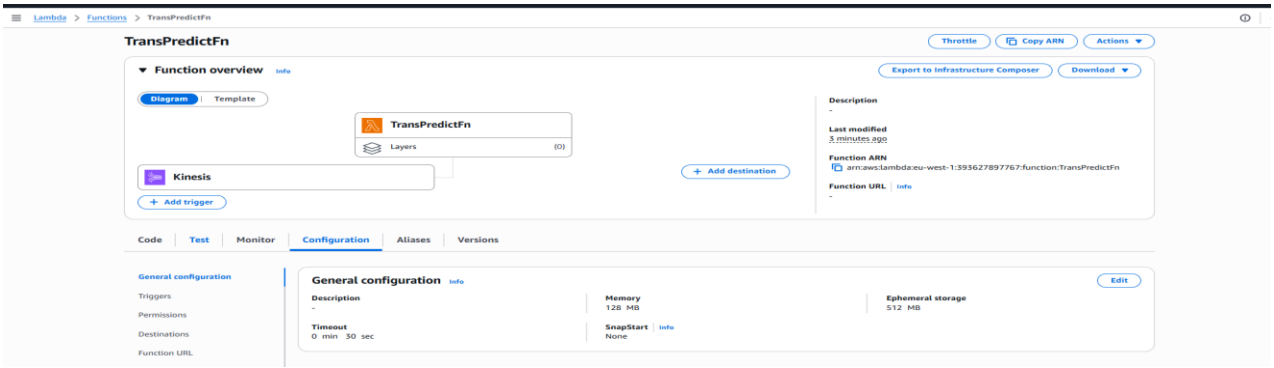


- Variables -KINESIS-STREAM is key for the value TRANS-STREAM which is kinesis stream which will be used by lambda to connect to it.

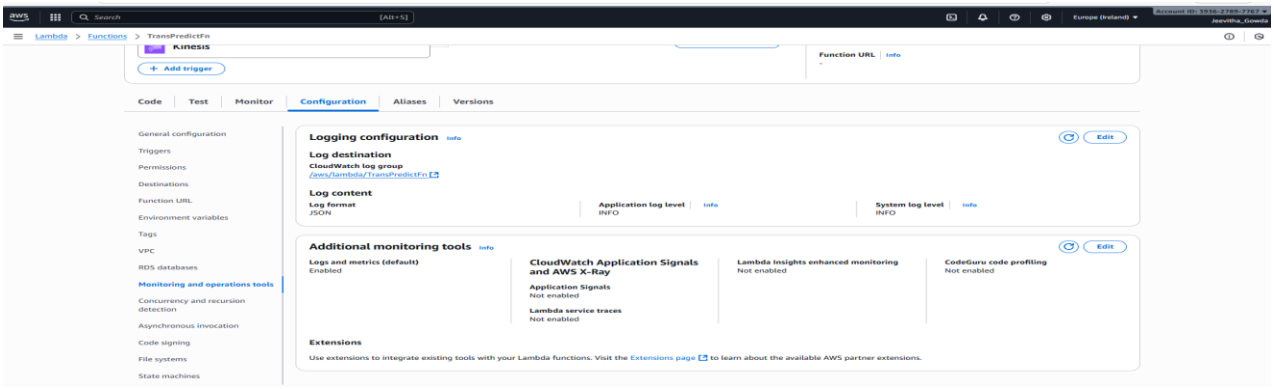


3.6.2 Prediction Lambda:

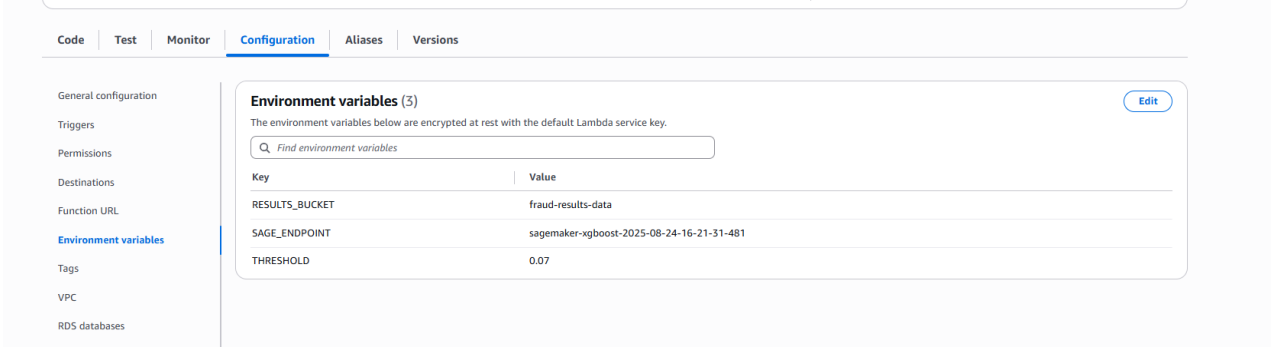
- It reads events from Kinesis, invokes SageMaker, and stores results in S3.
- 128 MB for memory with storage of 512 mb is allocated for testing.



- Logs are enabled to track and monitor all the activities



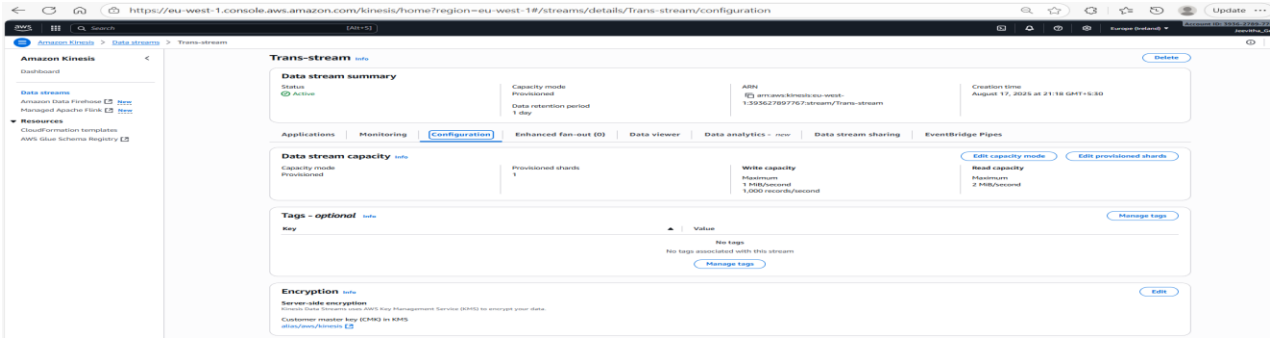
- Variables : key–value pairs is created that the function uses to connect to resources like an S3 bucket, a SageMaker endpoint.



3.7. Streaming Layer (Kinesis Data Stream)

3.7.1Stream Setup:

- Configured Trans-stream (Kinesis stream) with 1 shard it is scalable as needed.
- Encryption is Enabled



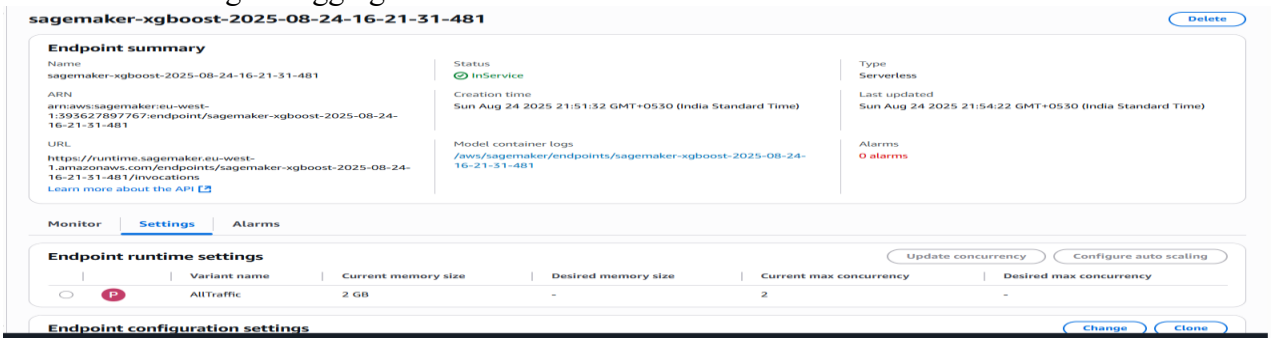
3.8Amazon SageMaker

3.8.1 Endpoint Summary (second screenshot)

- **Status**: InService → the endpoint is live and ready to serve predictions.
- **Name:** sagemaker-xgboost-2025-08-24-16-21-31-481.
- **Type**: Serverless SageMaker handles scaling).
- **URL**: For REST API endpoint for invoking predictions-https://runtime.sagemaker.eu-west-1.amazonaws.com/endpoints/sagemaker-xgboost-2025-08-24-16-21-31-481/invocations

The Lambda function will call this URL to send transaction data and receive fraud predictions.

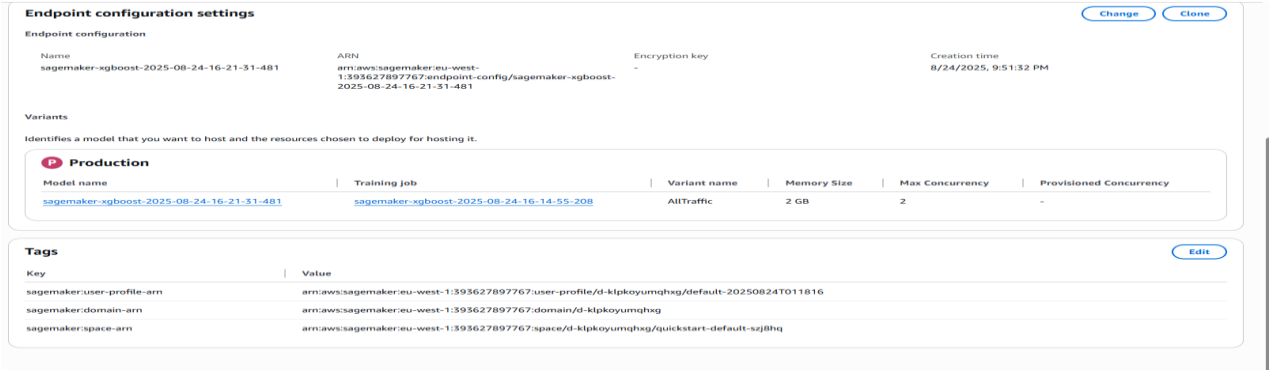
- **Logs**: It is Stored in CloudWatch (/aws/sagemaker/endpoints/...) for monitoring/debugging.



3.8.2 Endpoint Configuration Settings

- **Name:** sagemaker-xgboost-2025-08-24-16-21-31-481.
- **Production Variant**:

  - Training Job- Trained with job sagemaker-xgboost-2025-08-24-16-14-55-208
  - Variant-AllTraffic
  - Memory Size-2 GB allocated for container.
  - Max Concurrency- 2



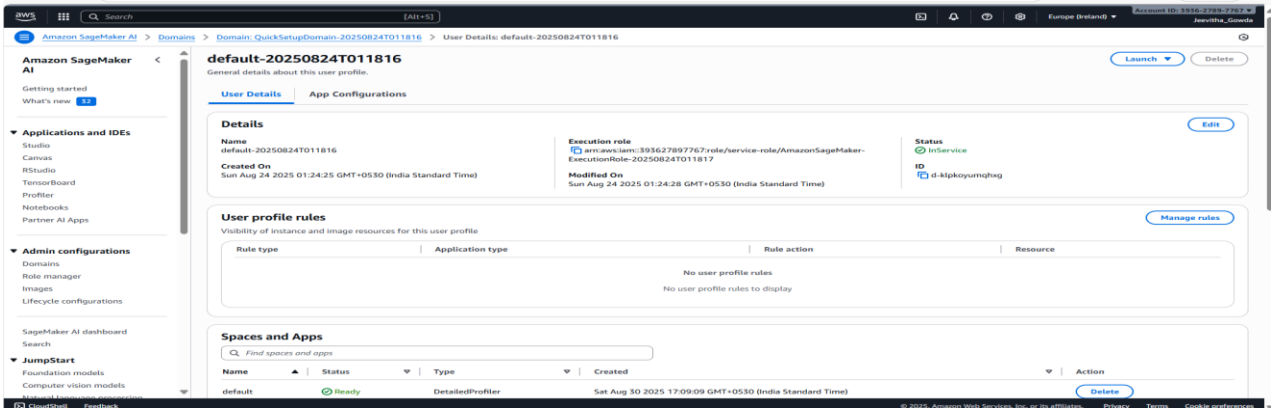
3.8.3. SageMaker Domain, User Profile & Studio Setup

1 SageMaker Domain Setup

- **Step 1:** A Domain was created to provide a centralized environment for developing and collaboration.

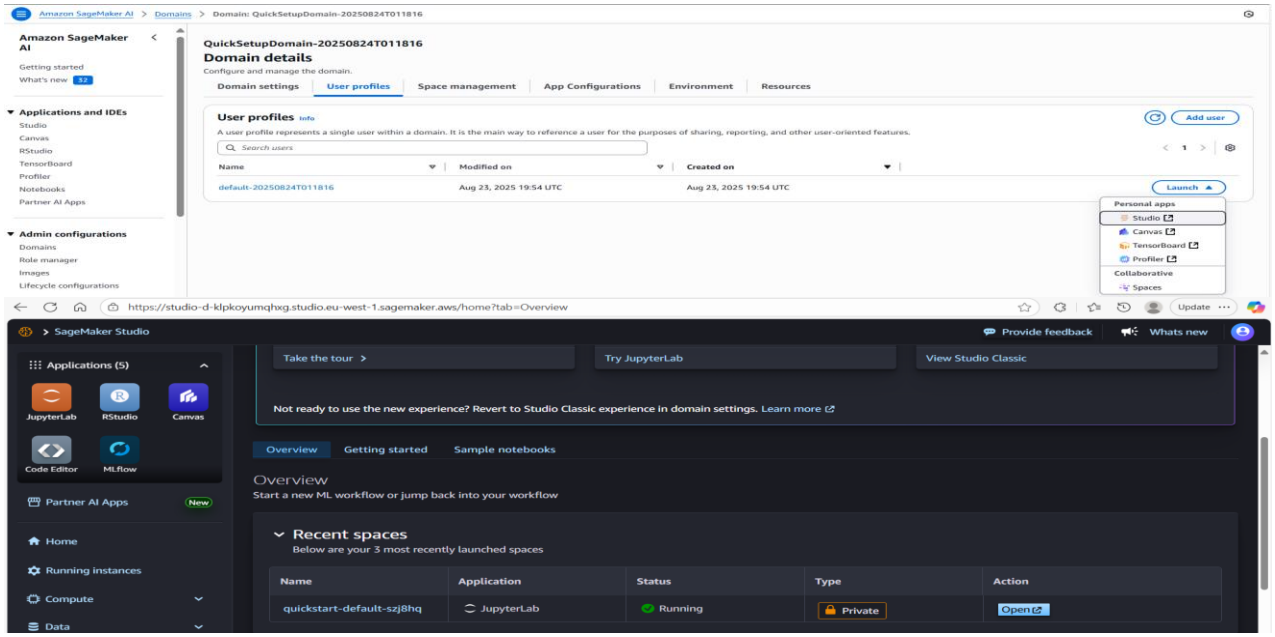
2 User Profile Creation

- **Step 1:** A **User Profile** was created in the Domain and for the project researcher.



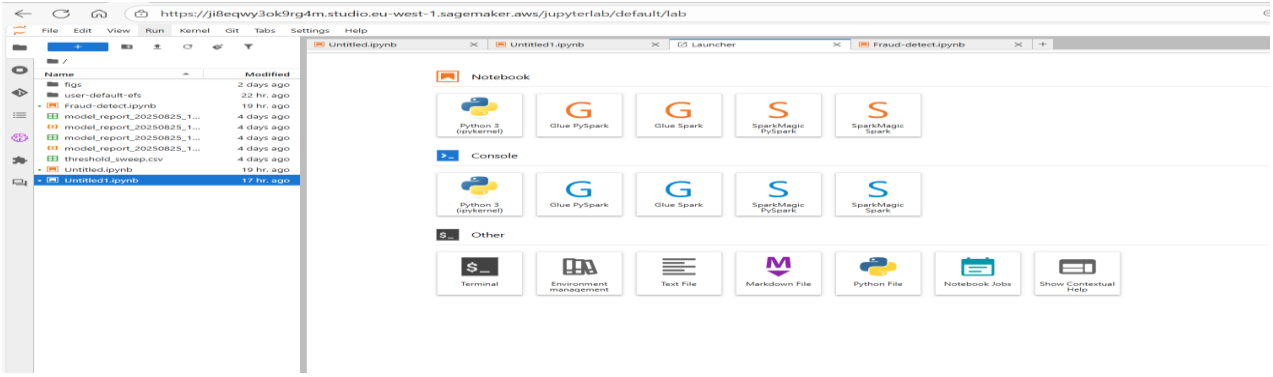
3 SageMaker Studio Launch

- **Step:** Once the User Profile was created SageMaker Studio was launched which enabled access to an integrated ML IDE as this provides interface for building, training, and deploying models. Where notebooks, deployment and experiment tracking can be seen in a single workspace as this reduces setup complexity and time consumption



4 Jupyter Notebook Environment

- **Step:** A Jupyter Notebook was used for this project inside SageMaker Studio for purpose of training and testing the fraud detection model.Python 3 is used as Kernel for the Notebook.



5.Model setup in SageMaker

1. Install Dependencies This command interpolates the necessary Python packages to develop the models. It comes with boto3 (aws sdk), pandas (data management), matplotlib (figure generation), and scikit-learn (ML tools). The command -quiet is used to have the installation silently take place without including the elaborate logs.

```
[1]: !pip install --quiet boto3 pandas matplotlib scikit-learn
```

3. Upload Raw Data to S3 The training and testing CSVs are raw datasets are uploaded in S3 bucket to be accessed by SageMaker. The files are placed in all under defined keys (e.g.,xgb/xgb_train.csv and xgb/xgb_test.csv). The final S3 paths are printed by the code to allow re-use in training.

[7]:
```
import boto3, sagemaker
bucket_raw   = "fraud-raw-datasets"
bucket_model = "fraud-model-dataset"
region = sagemaker.Session().boto_region_name
s3c = boto3.client("s3", region_name=region)

train_key = "xgb/xgb_train.csv"
test_key  = "xgb/xgb_test.csv"

s3c.upload_file("/tmp/xgb_train.csv", bucket_raw, train_key)
s3c.upload_file("/tmp/xgb_test.csv",  bucket_raw, test_key)

train_s3 = f"s3://{bucket_raw}/{train_key}"
test_s3  = f"s3://{bucket_raw}/{test_key}"
print("Uploaded:", train_s3, "and", test_s3)
```

```
Uploaded: s3://fraud-raw-datasets/xgb/xgb_train.csv and s3://fraud-raw-datasets/xgb/xgb_test.csv
```

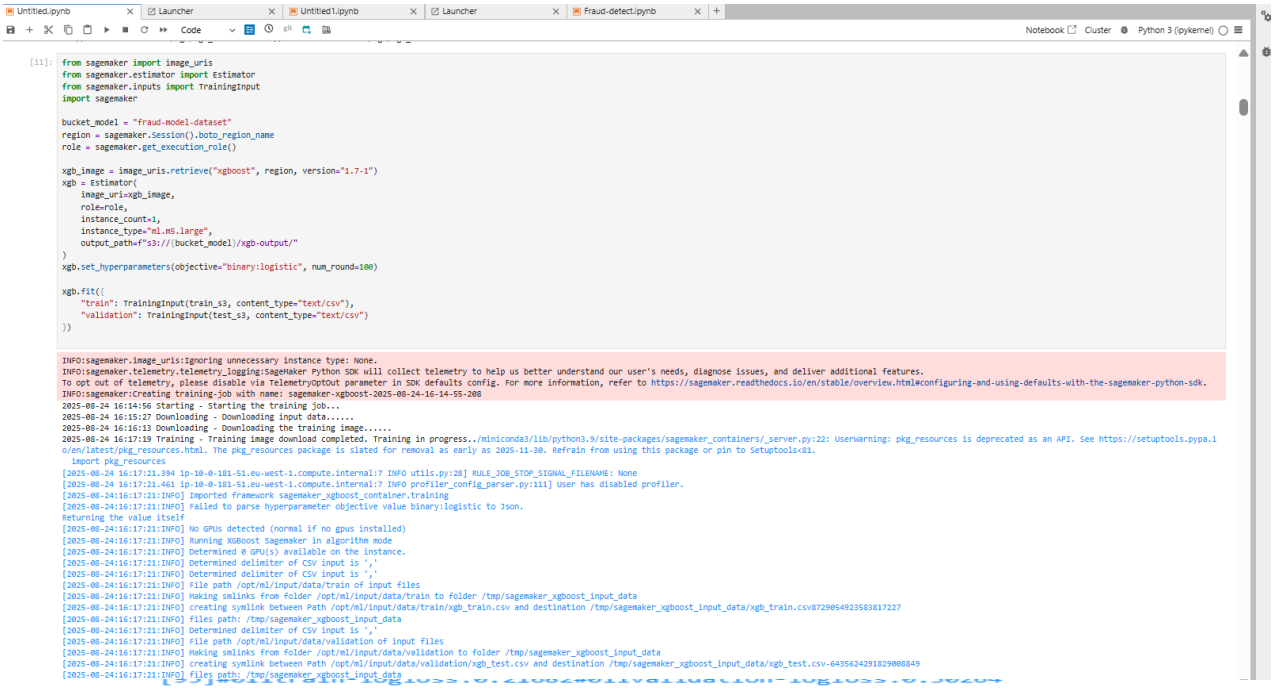
4. Data Preprocessing Raw datasets downloaded are processed into a clean and normalized and fixed into XGBoost-style format. Label values are illegibly mapped with 0 (non-fraud) and 1 (fraud), non-numeric attributes are discarded, and missing values are imputed. Introduction of final training and testing files is then saved as CSVs, that are headed which are uploaded in S3 again.

[6]:
```
import pandas as pd, numpy as np, boto3, sagemaker

# paths & buckets you already use
bucket_raw   = "fraud-raw-datasets"
train_key    = "fraud_traindataset.csv"
test_key     = "fraud_testdataset.csv"

region = sagemaker.Session().boto_region_name
s3 = boto3.client("s3", region_name=region)

# 1) download raw
s3.download_file(bucket_raw, train_key, "/tmp/train.csv")
s3.download_file(bucket_raw, test_key,  "/tmp/test.csv")

# 2) load & normalize columns
train_df = pd.read_csv("/tmp/train.csv")
test_df  = pd.read_csv("/tmp/test.csv")
train_df.columns = [c.strip().lower() for c in train_df.columns]
test_df.columns  = [c.strip().lower() for c in test_df.columns]

# 👉 if your label column name is different, set it here:
label_col = "label"
assert label_col in train_df.columns, f"Label column '{label_col}' not found. Columns: {train_df.columns.tolist()}"

# 3) show current labels
print("Unique labels (train) BEFORE:", pd.unique(train_df[label_col])[:20])

# 4) convert labels to {0,1}
def to_binary(series):
    mapping = {
        "1":1,"0":0, 1:1,0:0,
        "true":1,"false":0,
        "yes":1,"no":0,
        "fraud":1,"fraudulent":1,
        "legit":0,"legitimate":0,"non-fraud":0,"nonfraud":0
    }
    def _map(v):
        s = str(v).strip().lower()
```

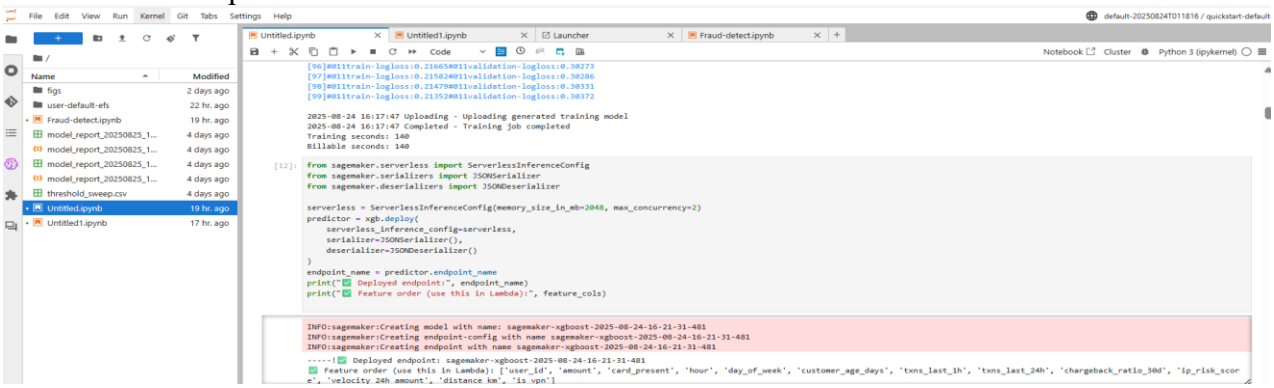
```
        s = str(v).strip().lower()
        return mapping[s] if s in mapping else v
    out = series.map(_map)
    # if still not numeric, try cast; then clamp to {0,1}
    out = pd.to_numeric(out, errors="coerce").fillna(0)
    out = out.clip(lower=0, upper=1).astype(int)
    return out

train_df[label_col] = to_binary(train_df[label_col])
test_df[label_col]  = to_binary(test_df[label_col])

print("Unique labels (train) AFTER:", pd.unique(train_df[label_col]))

# 5) keep ONLY numeric features (strings break training)
num_train = train_df.select_dtypes(include=[np.number]).copy()
num_test  = test_df.select_dtypes(include=[np.number]).copy()

assert label_col in num_train.columns, "Label not numeric after conversion."
bad = set(num_train[label_col].unique()) - {0,1}
assert not bad, f"Labels must be 0/1; found {bad}"

# 6) fill NaNs / inf
for df in (num_train, num_test):
    df.replace([np.inf, -np.inf], np.nan, inplace=True)
    df.fillna(0, inplace=True)

# 7) build feature list (everything numeric except label)
feature_cols = [c for c in num_train.columns if c != label_col]
print("Feature columns:", feature_cols[:20], "... (total:", len(feature_cols), ")")

# 8) write XGBoost CSVs (no header, label last)
pd.concat([num_train[feature_cols], num_train[[label_col]]], axis=1)\
  .to_csv("/tmp/xgb_train.csv", index=False, header=False)
pd.concat([num_test[feature_cols],  num_test[[label_col]]],  axis=1)\
  .to_csv("/tmp/xgb_test.csv", index=False, header=False)

print("✅ Prepared /tmp/xgb_train.csv and /tmp/xgb_test.csv")
```

```
Unique labels (train) BEFORE: [0 1]
Unique labels (train) AFTER: [0 1]
Feature columns: ['user_id', 'amount', 'card_present', 'hour', 'day_of_week', 'customer_age_days', 'txns_last_1h', 'txns_last_24h', 'chargeback_ratio_30d', 'ip_risk_score', 'velocity_24h_amoun
t', 'distance_km', 'is_vpn'] ... (total: 13 )
 ✅ Prepared /tmp/xgb_train.csv and /tmp/xgb_test.csv
```

5. Model Training: Training is performed by the SageMaker built-in XGBoost algorithm. All the datsets from S3 are fetched and trained for model evaluation.



6. Model Deployment The trained model is deployed as an endpoint of SageMaker, as a serverless endpoint with 2048 MB memory and maximum concurrency of 2. In SageMaker endpoint name is automatically created. The feature order is printed so that the input structure would be correct when it comes to prediction or Lambda calls.



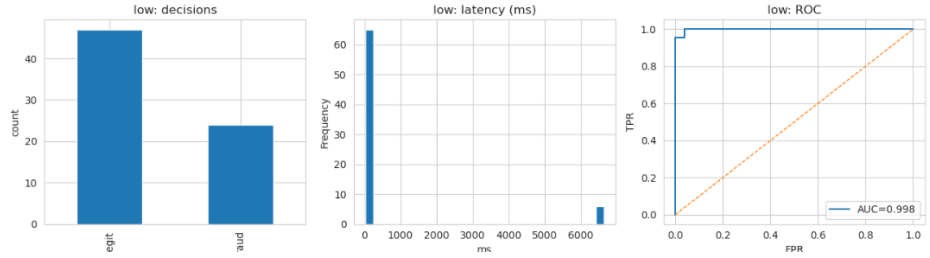
7. Model Evaluation The model is tested in several traffic situations (low, medium, high). Such measurements as accuracy, precision, recall, F1-score and latency are calculated. Outcomes indicate great recall and accuracy, that is, the model identifies the majority of fraud and the minimal false negativity

```
                avg_ms=round(avg_ms,2) if avg_ms==avg_ms else None,
                p95_ms=round(p95_ms,2) if p95_ms==p95_ms else None,
                avg_tps=round(tps,2) if tps else None,
                accuracy=round(acc,4), precision=round(prec,4),
                recall=round(rec,4), f1=round(f1,4),
                TP=TP, FP=FP, FN=FN, TN=TN)

BEST_THR = None   # use Lambda's decisions; set e.g. 0.10 to re-threshold from score

summary_rows = []
dfs_labeled = {}

for name, d in dfs.items():
    d2 = add_label_and_pred(d, thr=BEST_THR)
    dfs_labeled[name] = d2
    summary_rows.append({"scenario": name, **metrics_for_df(d2)})

summary = pd.DataFrame(summary_rows)
summary
```

[25]:

	scenario	rows	legit_pred	fraud_pred	avg_ms	p95_ms	avg_tps	accuracy	precision	recall	f1	TP	FP	FN	TN
0	low	71	47	24	643.31	6508.0	2.85	0.9577	0.8750	1.0000	0.9333	21	3	0	47
1	med	1000	675	325	349.09	661.1	16.69	0.9750	0.9231	1.0000	0.9600	300	25	0	675
2	high	9932	6656	3276	412.61	779.0	20.79	0.9705	0.9118	0.9987	0.9532	2987	289	4	6652

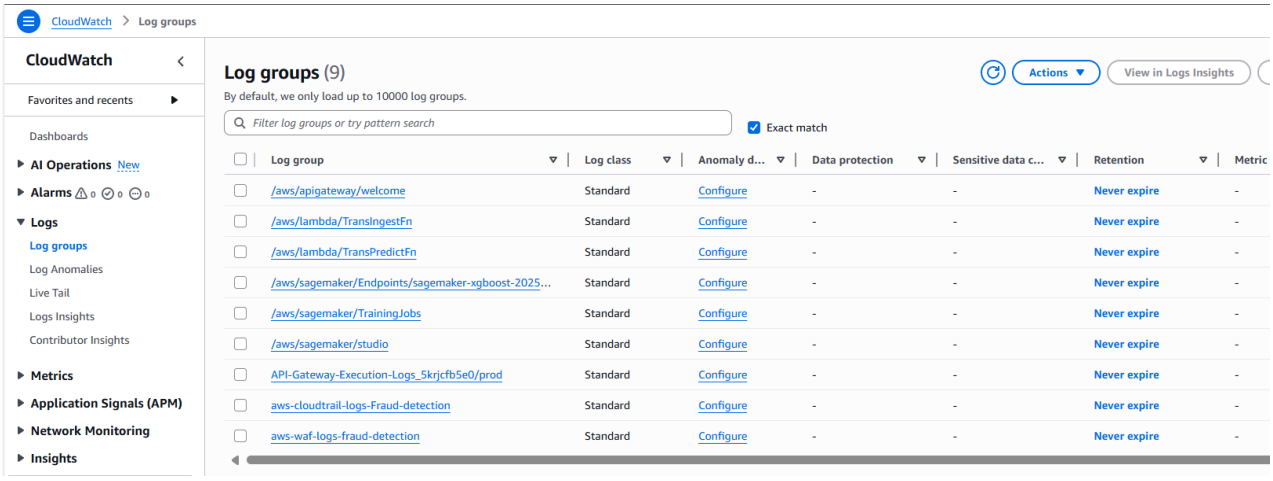
[26]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

for name, d in dfs_labeled.items():
    if d.empty:
        print(f"{name}: no data to plot")
        continue

    fig, axes = plt.subplots(1,3, figsize=(13,4))

    # decisions
    d["decision"].value_counts().reindex(["legit","fraud"]).plot(kind="bar", ax=axes[0], title=f"{name}: decisions")
    axes[0].set_xlabel("decision"); axes[0].set_ylabel("count")

    # Latency
    if "latency_ms" in d and d["latency_ms"].notna().any():
        d["latency_ms"].plot(kind="hist", bins=30, ax=axes[1], title=f"{name}: latency (ms)")
        axes[1].set_xlabel("ms")

    # ROC from scores (if present)
    if "score" in d and d["score"].notna().any():
        yy  = d["label"].values
        scr = pd.to_numeric(d["score"], errors="coerce").fillna(0).values
        fpr, tpr, thr = roc_curve(yy, scr)
        roc_auc = auc(fpr, tpr)
        axes[2].plot(fpr, tpr, label=f"AUC={roc_auc:.3f}")
        axes[2].plot([0,1],[0,1],"--", linewidth=1)
        axes[2].set_title(f"{name}: ROC")
        axes[2].set_xlabel("FPR"); axes[2].set_ylabel("TPR"); axes[2].legend()

    plt.tight_layout(); plt.show()
```

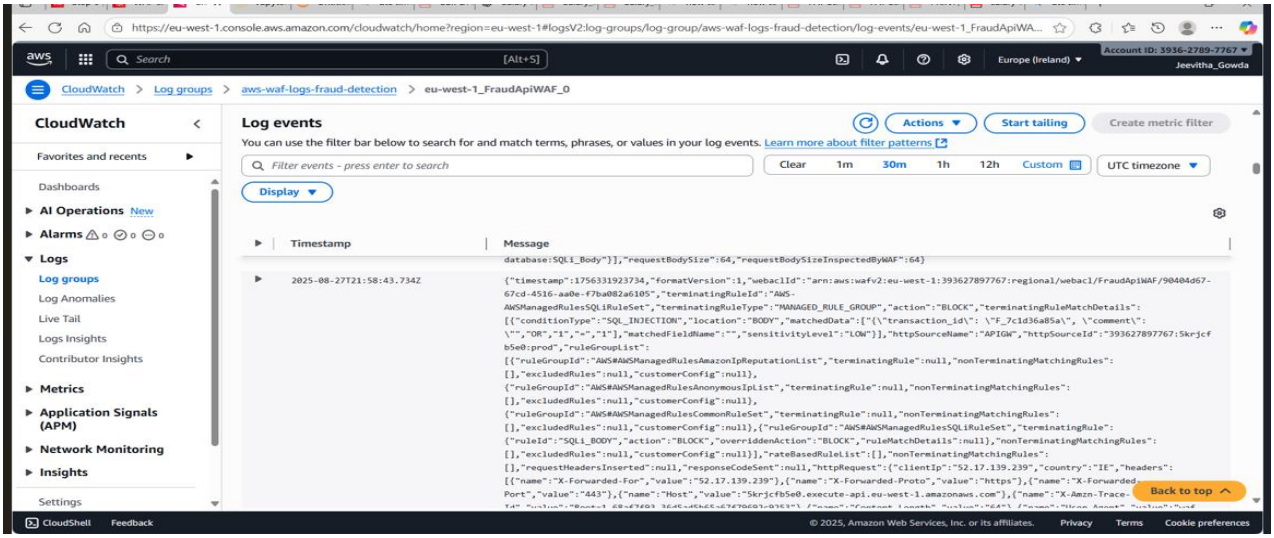


4. Monitoring & Auditing

4.1CloudWatch Logs:

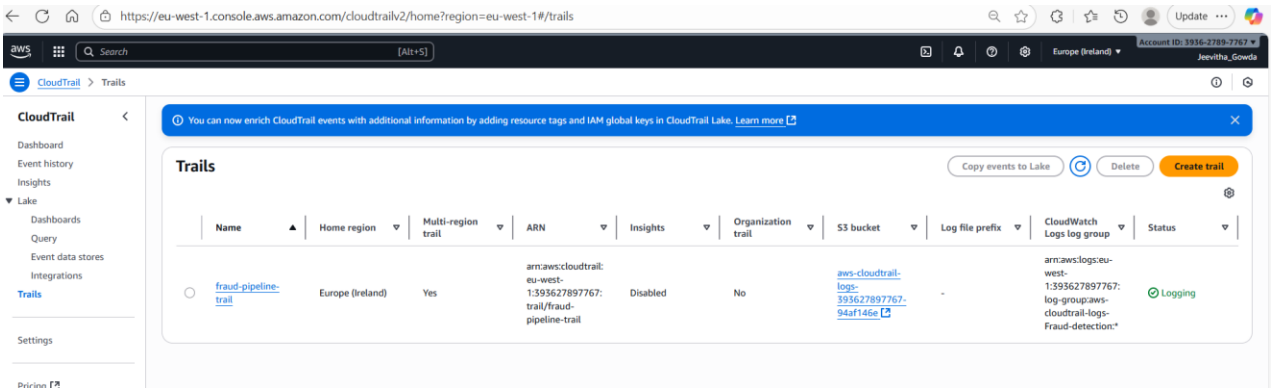
CloudWatch is used to capture the logs for tracking purpose and auditing , log groups are enabled for each component and retention period is set to never expire





2.CloudTrail:

Fraud-pipeline-trail is created where logs are sent both to S3 buckets and Cloud watch for monitoring and storing**.**



Reference

Domain user profiles - Amazon SageMaker AI. (n.d.). https://docs.aws.amazon.com/sagemaker/latest/dg/domain-user-profile.html

Create your first Lambda function - AWS Lambda. (n.d.). https://docs.aws.amazon.com/lambda/latest/dg/getting-started.html

aws. “AWS WAF - Web Application Firewall - Amazon Web Services (AWS).” *Amazon Web Services, Inc.*, 2024, aws.amazon.com/waf/.

aws. “Amazon API Gateway.” *Amazon Web Services, Inc.*, 2019, aws.amazon.com/api-gateway/.

Amazon Kinesis.” *Amazon Web Services, Inc.*, 2019, aws.amazon.com/kinesis/.

“Amazon SageMaker Studio (12/4).” *Amazon Web Services, Inc.*, 29 Aug. 2025, aws.amazon.com/sagemaker/ai/studio/. Accessed 31 Aug. 2025.