



**Programme:** MCA

**Semester:** Winter 24-25

**Course:** Cloud Computing

**Course Code:** PMCA506L

**Faculty:** Dr. Priyaadharshini

**Slot:** E2 + TE2

## **REAL-TIME CREDIT CARD FRAUD DETECTION USING AWS SERVICES**

### **Team Members**

24MCA1018 – Sivas D

24MCA1031 – Gokulnath E

24MCA1033 – Shajith N

## **Abstract**

Credit card fraud is a significant financial issue that results in billions of dollars in losses each year. The need for an efficient, real-time fraud detection system is crucial for financial institutions. This project leverages AWS cloud services and machine learning to build an automated fraud detection system. The system classifies transactions as either fraudulent or legitimate based on various transaction features. Using AWS Lambda, API Gateway, SageMaker, SNS, CloudWatch, IAM, and EC2, we ensure a scalable and efficient solution. The implementation of this project highlights the real-time detection capabilities, alert notifications, and cloud-based security features that improve fraud prevention strategies.

## **1. Introduction**

### **1.1 Project Overview**

Credit card fraud is a growing threat, leading to financial losses and security concerns. This project focuses on utilizing **AWS cloud technologies** to implement a **machine learning-powered fraud detection system** that can process transactions in real-time and provide instant alerts for suspected fraud cases.

### **1.2 Objectives**

- Develop an automated fraud detection system with real-time classification.
- Utilize machine learning models to analyze transaction patterns.
- Deploy the system using AWS Lambda, API Gateway, SageMaker, SNS, CloudWatch, IAM, EC2, and S3.
- Ensure high accuracy in fraud detection.
- Provide instant alerts to users in case of suspected fraud.

## **Detailed Description of the Modules:**

### **1. Amazon S3 (Simple Storage Service)**

#### **Purpose:**

Used for storing datasets used to train the fraud detection model and storing preprocessed transaction data.

#### **Functions:**

- Uploads the raw credit card transaction dataset.
- Lambda fetches and preprocesses data from S3 before model training.
- Ensures durability and easy accessibility for SageMaker.

#### **Interaction:**

Lambda and SageMaker access data from S3 for training or inference.

### **2. Amazon SageMaker**

#### **Purpose:**

To train and host the machine learning model for fraud detection.

#### **Functions:**

- Trains an XGBoost model using historical transaction data.
- Deploys a real-time endpoint for fraud prediction.
- Accepts feature inputs in CSV format and returns a fraud score.

#### **Interaction:**

Lambda invokes the SageMaker endpoint to get predictions during runtime.

### **3. AWS Lambda**

#### **Purpose:**

Acts as the **serverless backend function** that handles business logic.

#### **Functions:**

- Parses API request JSON to extract features.
- Sends features to SageMaker for inference.
- Processes model output to determine if it's fraud.
- Sends alerts to SNS if fraud is detected.
- Logs to CloudWatch for traceability.

#### **Interaction:**

Triggered by API Gateway. Communicates with SageMaker, SNS, and CloudWatch.

## **4. Amazon API Gateway**

### **Purpose:**

Acts as the **front-facing interface** for users and clients to interact with the backend logic.

### **Functions:**

- Accepts HTTP requests from the frontend or external tools (like curl or Postman).
- Forwards the request to Lambda securely.
- Handles CORS and throttling policies.

### **Interaction:**

Receives input from EC2 (Flask UI) or command line, and routes to Lambda.

## **5. AWS SNS (Simple Notification Service)**

### **Purpose:**

Provides **real-time alerting** in the event of a fraud detection.

### **Functions:**

- Sends email or SMS alerts when fraud is detected.
- Helps in rapid incident response or further transaction investigation.

### **Interaction:**

Called by Lambda if the fraud score crosses a defined threshold.

## **6. AWS CloudWatch**

### **Purpose:**

Used for **monitoring and logging** activity across AWS services.

### **Functions:**

- Logs Lambda executions and errors.
- Tracks performance metrics for debugging.
- Can trigger alarms or auto-remediation based on rules.

### **Interaction:**

Lambda automatically pushes logs. IAM policies enable CloudWatch access.

## **7. AWS IAM (Identity and Access Management)**

### **Purpose:**

Provides **secure access control** to AWS services and resources.

### **Functions:**

- Grants permissions to Lambda to invoke SageMaker, access SNS, S3, etc.
- Ensures only authorized actions are allowed.
- Manages roles (like `InvokeSageMakerLambdaRole`) with least-privilege policy.

### **Interaction:**

Every service interaction is validated by IAM permissions behind the scenes.

## **8. Amazon EC2 (Elastic Compute Cloud)**

### **Purpose:**

Hosts a **Flask web interface** for user input and model interaction.

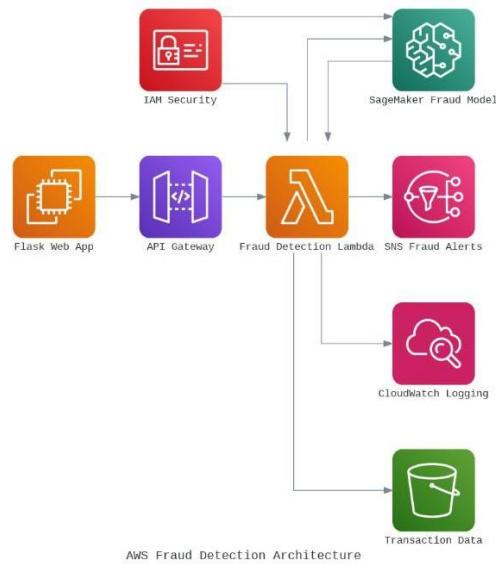
### **Functions:**

- Web server for transaction input form.
- Communicates with API Gateway.
- Uses Gunicorn and Nginx for production-grade serving.

### **Interaction:**

Users access this UI to simulate transactions, which are then routed to the API.

## 2. Architecture & Workflow



### 2.1 Workflow Steps

1. User submits a transaction request via a Flask UI hosted on EC2.
2. The request is forwarded to API Gateway.
3. API Gateway triggers AWS Lambda, which processes the transaction data.
4. Lambda invokes the SageMaker ML model to classify the transaction.
5. If fraud is detected, AWS SNS sends an alert notification.
6. Transaction records and logs are stored in S3 and CloudWatch for monitoring.

#### Individual Descriptions:

**24MCA1033 – Shajith N**

**SageMaker, API Gateway, and Lambda:**

- Developed and trained the XGBoost model using Amazon SageMaker on the credit card fraud dataset.
- Created and deployed a SageMaker endpoint to enable real-time inference from transaction features.

- Built AWS Lambda functions that:
  - Process input data.
  - Invoke the SageMaker endpoint.
  - Interpret the model prediction.
- Configured Amazon API Gateway to expose REST endpoints for transaction inputs.
- Ensured integration between API Gateway and Lambda with proper method and stage deployments.

#### **24MCA1031 – Gokulnath E**

##### **EC2, SNS:**

- Configured the EC2 instance (Ubuntu) with Python, Flask, Gunicorn, and Nginx.
- Developed the Flask-based UI to allow users to input transaction data easily.
- Integrated the frontend with the API Gateway endpoint for real-time requests.
- Set up AWS SNS to send fraud alerts via email/SMS from Lambda.
- Tested the end-to-end communication from UI to backend fraud detection system.

#### **24MCA1018 – Sivas D**

##### **S3, IAM, CloudWatch:**

##### **Handled Amazon S3 storage:**

- Uploaded datasets.
- Enabled access for training and Lambda pre-processing.

##### **Created and managed IAM roles and policies:**

- Defined secure permissions for Lambda, SageMaker, and API Gateway.
- Ensured least privilege access for all services.

## Set up CloudWatch monitoring:

- Verified logs for Lambda executions and SNS triggers.
- Enabled performance and error monitoring.

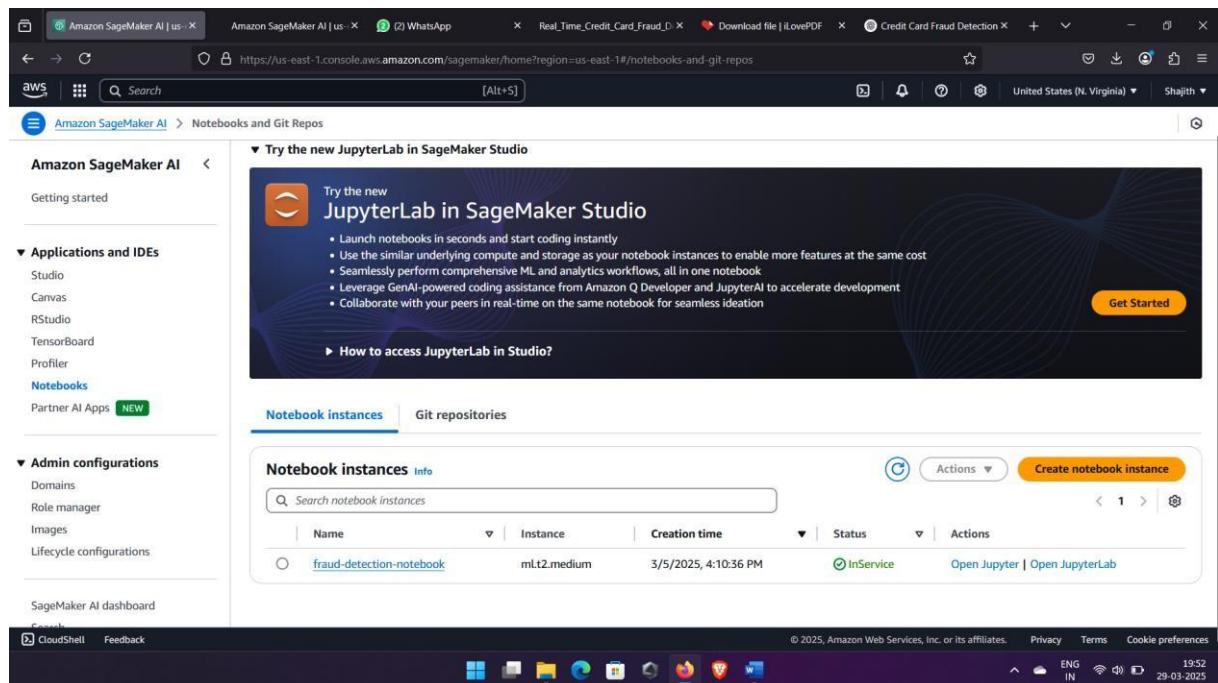
## 3. Implementation

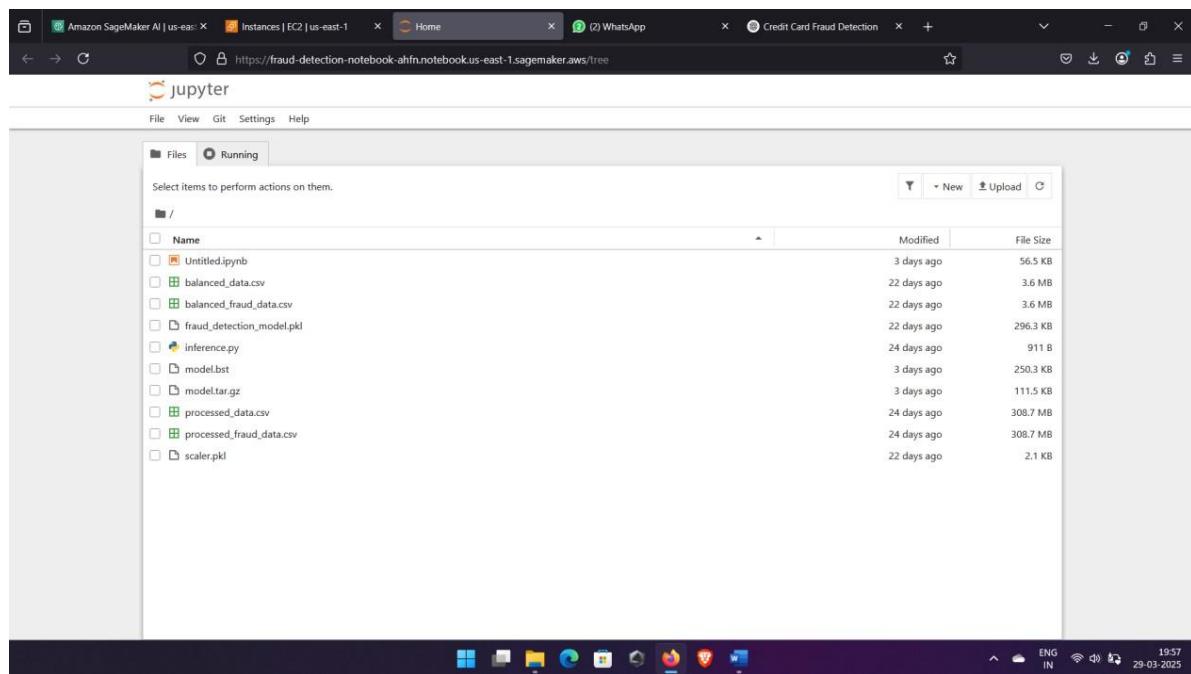
### AWS Services Used

#### AWS SageMaker

- Trained an XGBoost model on a credit card fraud dataset.
- Model predicts a fraud score, classifying transactions as fraudulent or non-fraud

#### Screenshot:





ENG IN WiFi 29-03-2025 19:57

```
[1]: !pip install xgboost
Collecting xgboost
  Downloading xgboost-3.0.0-py3-none-manylinux2014-x86_64.whl.metadata (2.0 kB)
Requirement already satisfied: numpy in /home/ec2-user/anaconda3/envs/python3/lib/python3.10/site-packages (from xgboost) (1.26.4)
Requirement already satisfied: scipy in /home/ec2-user/anaconda3/envs/python3/lib/python3.10/site-packages (from xgboost) (1.15.1)
  Downloading xgboost-3.0.0-py3-none-manylinux2014-x86_64.whl (4.9 MB)
                                         4.9/4.9 MB 101.4 MB/s eta 0:00:00
Installing collected packages: xgboost
Successfully installed xgboost-3.0.0

[2]: !pip install imbalanced-learn
Collecting imbalanced-learn
  Downloading imbalanced-learn-0.13.0-py3-none-any.whl.metadata (8.8 kB)
Requirement already satisfied: numpy<3,>=1.24.3 in /home/ec2-user/anaconda3/envs/python3/lib/python3.10/site-packages (from imbalanced-learn) (1.26.4)
Requirement already satisfied: scipy<2,>=1.10.1 in /home/ec2-user/anaconda3/envs/python3/lib/python3.10/site-packages (from imbalanced-learn) (1.15.1)
Requirement already satisfied: scikit-learn<2,>=1.3.2 in /home/ec2-user/anaconda3/envs/python3/lib/python3.10/site-packages (from imbalanced-learn) (1.6.1)
Collecting sklearn-compat<1,>=0.1 (from imbalanced-learn)
  Downloading sklearn_compat-0.1.3-py3-none-any.whl.metadata (18 kB)
Requirement already satisfied: joblib<2,>=1.1.1 in /home/ec2-user/anaconda3/envs/python3/lib/python3.10/site-packages (from imbalanced-learn) (1.4.2)
Requirement already satisfied: threadpoolctl<4,>=2.0.0 in /home/ec2-user/anaconda3/envs/python3/lib/python3.10/site-packages (from imbalanced-learn) (3.5.0)
  Downloading imbalanced_learn-0.13.0-py3-none-any.whl (238 kB)
Downloaded sklearn_compat-0.1.3-py3-none-any.whl (18 kB)
Installing collected packages: sklearn-compat, imbalanced-learn
Successfully installed imbalanced-learn-0.13.0 sklearn-compat-0.1.3

[3]: import pandas as pd
import xgboost as xgb
from sklearn.model_selection import train_test_split

# Load the dataset
df = pd.read_csv("balanced_fraud_data.csv") # Change filename if needed
```

```
[3]: import pandas as pd
import xgboost as xgb
from sklearn.model_selection import train_test_split

# Load the dataset
df = pd.read_csv("balanced_fraud_data.csv") # Change filename if needed

# Drop unnecessary columns
df = df.drop(columns=["merchant", "job", "trans_num", "cc_num"]) # Remove non-numeric columns

# Convert 'trans_date_trans_time' to Unix timestamp
df["trans_date_trans_time"] = pd.to_datetime(df["trans_date_trans_time"])
df["trans_date_trans_time"] = df["trans_date_trans_time"].astype(int) // 10**9 # Convert to seconds

# Define features (X) and target (y)
X = df.drop(columns=["is_fraud"]) # Remove target column
y = df["is_fraud"]

# Split dataset into training & test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train XGBoost Model
model = xgb.XGBClassifier(use_label_encoder=False, eval_metric="logloss")
model.fit(X_train, y_train)

print("Model trained successfully!")

/home/ec2-user/anaconda3/envs/python3/lib/python3.10/site-packages/xgboost/core.py:377: FutureWarning: Your system has an old version of glibc (< 2.28). We will stop supporting Linux distros with glibc older than 2.28 after **May 31, 2025**. Please upgrade to a recent Linux distro (with glibc >= 2.28) to use future versions of XGBoost.
Note: You have installed the 'manylinux2014' variant of XGBoost. Certain features such as GPU algorithms or federated learning are not available. To use these features, please upgrade to a recent Linux distro with glibc 2.28+, and install the 'manylinux_2_28' variant.
warnings.warn(
Model trained successfully!
```

## Endpoint:

The screenshot shows the Amazon SageMaker AI console with the endpoint details for 'sagemaker-xgboost-2025-03-29-14-29-22-965'. The left sidebar includes sections like JumpStart, Governance, HyperPod Clusters, Ground Truth, Processing, Training, Training Plans, Inference, and more. The main content area shows the endpoint summary with fields such as Name (sagemaker-xgboost-2025-03-29-14-29-22-965), ARN (arn:aws:sagemaker:us-east-1:597088015929:endpoint/sagemaker-xgboost-2025-03-29-14-29-22-965), URL (https://runtime.sagemaker.us-east-1.amazonaws.com/endpoints/sagemaker-xgboost-2025-03-29-14-29-22-965/invocations), Status (InService), Creation time (Sat Mar 29 2025 19:59:26 GMT+0530 (India Standard Time)), Type (Real-time), Last updated (Sat Mar 29 2025 20:06:42 GMT+0530 (India Standard Time)), and Alarms (0 alarms). Below the summary, there are tabs for Monitor, Settings, and Alarms, and a section for Operational Metrics showing CPU Utilization and Memory Utilization.

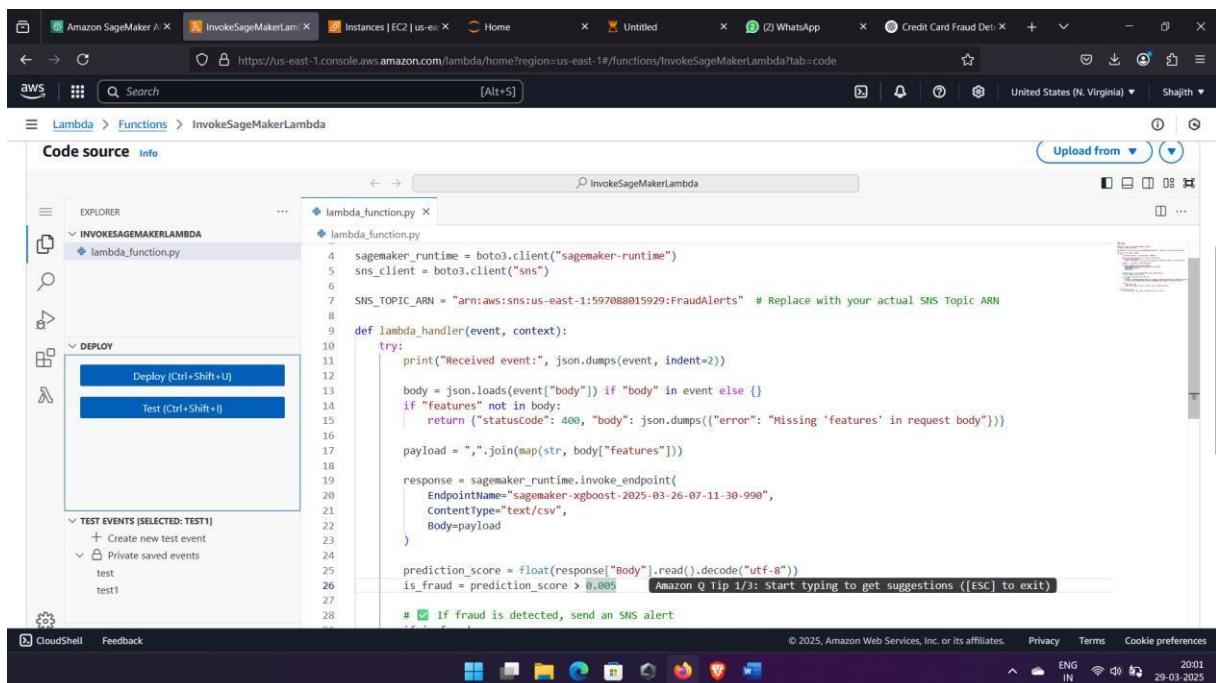
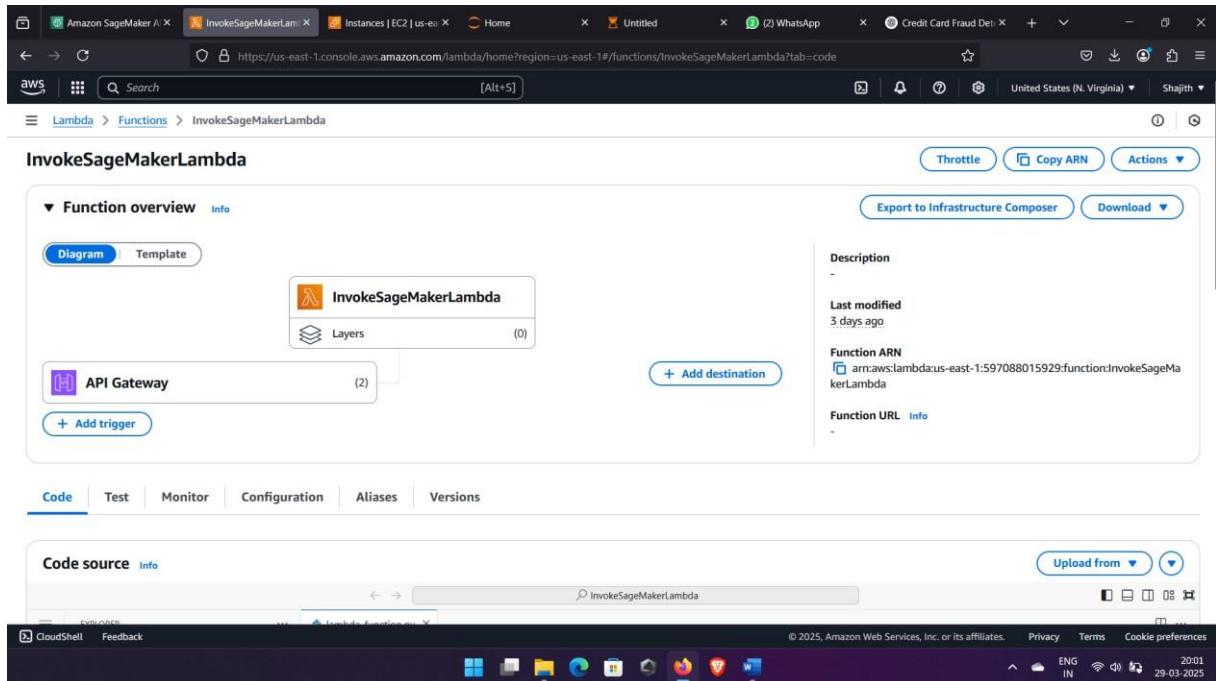
## AWS Lambda

- Processes transaction data and invokes **SageMaker endpoint**.
- Sends alerts using **AWS SNS**.
- Logs details using **CloudWatch**.

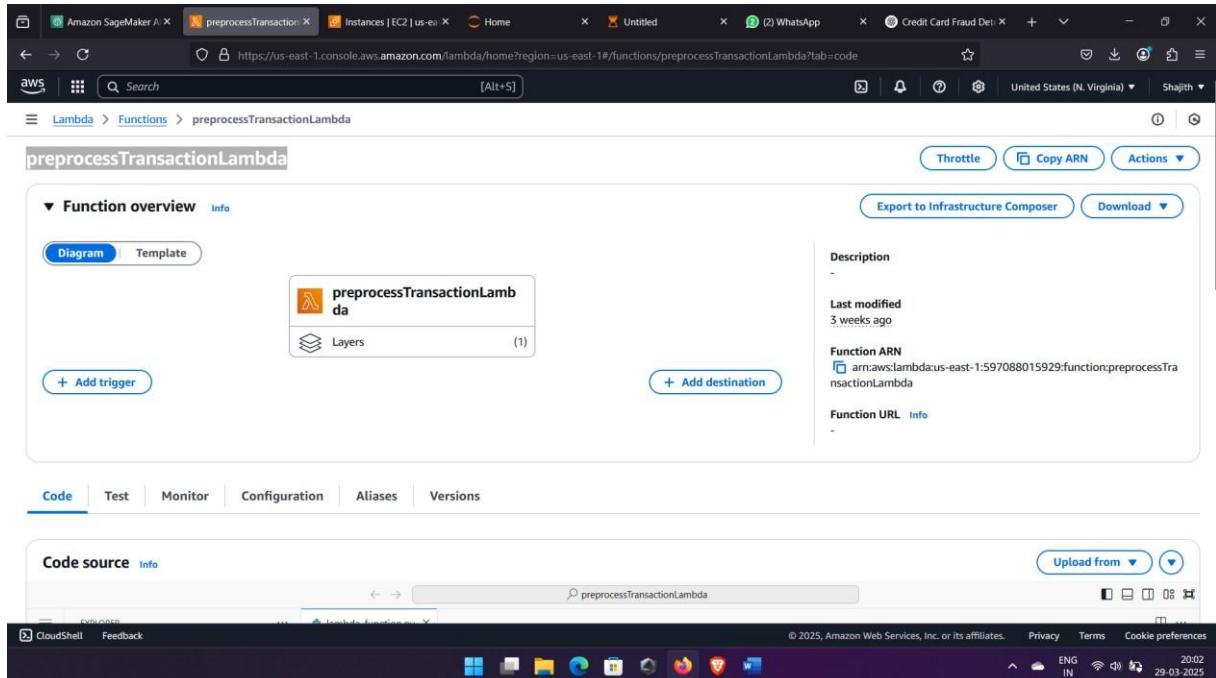
## Screenshot:

### InvokeSageMakerLambda

The screenshot shows the AWS Lambda console with the 'Functions' list. The left sidebar includes sections like Lambda, Additional resources, and Related AWS resources. The main content area shows a table of functions with columns for Function name, Description, Package type, Runtime, and Last modified. There are two entries: 'InvokeSageMakerLambda' (Description: -, Package type: Zip, Runtime: Python 3.9, Last modified: 3 days ago) and 'preprocessTransactionLambda' (Description: -, Package type: Zip, Runtime: Python 3.9, Last modified: 3 weeks ago). A 'Create function' button is located at the top right of the table.



## preprocessTransactionLambda



The screenshot shows the AWS Lambda function overview page for 'preprocessTransactionLambda'. The top navigation bar includes tabs for Amazon SageMaker, Instances | EC2 | us-east-1, Home, Untitled, WhatsApp, Credit Card Fraud Det., and more. The main content area displays the function name 'preprocessTransactionLambda' with a small icon, a 'Layers' section showing '(1)', and a 'Description' field indicating it was last modified 3 weeks ago. Buttons for 'Throttle', 'Copy ARN', and 'Actions' are visible. Below this, tabs for 'Code', 'Test', 'Monitor', 'Configuration', 'Aliases', and 'Versions' are present. The 'Code' tab is selected, showing the 'Code source' section with a code editor containing Python code for the lambda function.

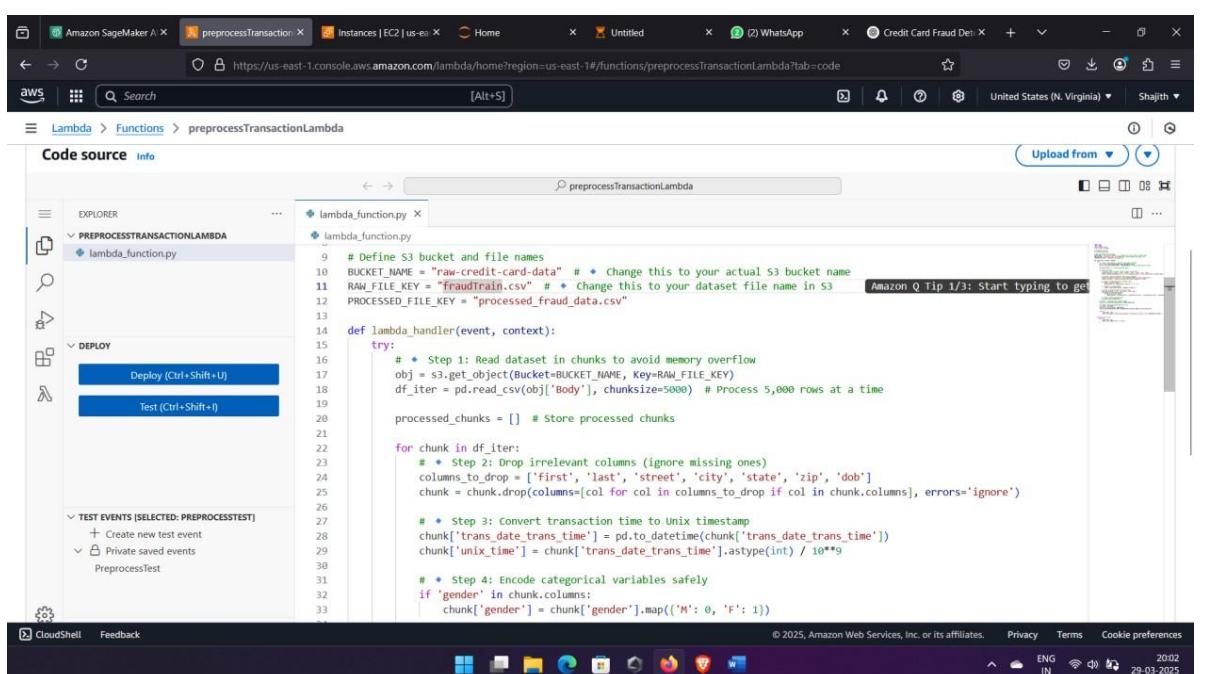
```
preprocessTransactionLambda
# Define S3 bucket and file names
BUCKET_NAME = "raw-credit-card-data" # + Change this to your actual S3 bucket name
RAW_FILE_KEY = "fraudTrain.csv" # + Change this to your dataset file name in S3
PROCESSED_FILE_KEY = "processed_fraud_data.csv"

def lambda_handler(event, context):
    # Step 1: Read dataset in chunks to avoid memory overflow
    obj = s3.get_object(Bucket=BUCKET_NAME, Key=RAW_FILE_KEY)
    df_iter = pd.read_csv(obj['Body'], chunksize=5000) # Process 5,000 rows at a time
    processed_chunks = [] # Store processed chunks

    for chunk in df_iter:
        # Step 2: Drop irrelevant columns (ignore missing ones)
        columns_to_drop = ['first', 'last', 'street', 'city', 'state', 'zip', 'dob']
        chunk = chunk.drop(columns=[col for col in columns_to_drop if col in chunk.columns], errors='ignore')

        # Step 3: Convert transaction time to Unix timestamp
        chunk['trans_date_trans_time'] = pd.to_datetime(chunk['trans_date_trans_time'])
        chunk['unix_time'] = chunk['trans_date_trans_time'].astype(int) / 10**9

        # Step 4: Encode categorical variables safely
        if 'gender' in chunk.columns:
            chunk['gender'] = chunk['gender'].map({'M': 0, 'F': 1})
```



The screenshot shows the AWS Lambda code editor for the 'lambda\_function.py' file. The left sidebar features an 'EXPLORER' view with a tree structure showing 'PREPROCESSTHANSACTIONLAMBDA' and 'TEST EVENTS [SELECTED: PREPROCESSTEST]'. Under 'TEST EVENTS', there are options to 'Create new test event' and 'Private saved events'. The main panel displays the Python code for the lambda function, which reads data from an S3 bucket, processes it in chunks, converts transaction times to Unix timestamps, and encodes categorical variables. A tooltip 'Amazon Q Tip 1/3: Start typing to get...' is visible in the background.

The screenshot shows the AWS Lambda function configuration page for 'preprocessTransactionLambda'. It includes sections for Code properties, Runtime settings, and Layers. Key details include:

- Code properties:** Package size: 1.2 kB, SHA256 hash: 12uqxu9tdqEJezd1CY4NoMcYWkoxyUiZLNA1ikT8E=, Last modified: 3 weeks ago.
- Runtime settings:** Runtime: Python 3.9, Handler: lambda\_function.lambda\_handler, Architecture: x86\_64.
- Layers:** One layer named 'AWSSDKPandas-Python39' is listed, version 5, compatible with Python 3.9 and x86\_64 architecture, ARN: arn:aws:lambda:us-east-1:336392948345:layer:AWSSDKPandas-Python39:5.

## Amazon API Gateway

- Exposes a **REST API** endpoint for transaction requests.
- Secures the API with authentication mechanisms.

### Screenshot:

The screenshot shows the AWS API Gateway APIs list page. It displays a single API named 'frauddetectionapi' with the following details:

Name	Description	ID	Protocol	API endpoint type	Created
frauddetectionapi		9tv4x243g	REST	Regional	2025-03-07

Screenshot of the AWS API Gateway Resources page for the 'frauddetectionapi' API.

**API Gateway** sidebar:

- APIs
- Custom domain names
- Domain name access associations
- VPC links
- APIs**:
  - frauddetectionapi
    - Resources
      - Stages
      - Authorizers
      - Gateway responses
      - Models
      - Resource policy
      - Documentation
      - Dashboard
      - API settings
    - Usage plans
    - API keys
    - Client certificates

**Resources** section:

- Create resource** button
- Path: /fraud-detection
- Method: POST
- Resource details** panel:
  - Path: /fraud-detection
  - Resource ID: r697nu
- Methods (1)** panel:
  - Method type: POST
  - Integration type: Lambda
  - Authorization: None
  - API key: Not required

Bottom navigation bar:

- CloudShell
- Feedback
- © 2025, Amazon Web Services, Inc. or its affiliates.
- Privacy
- Terms
- Cookie preferences
- ENG IN 2004 29-03-2025

Screenshot of the AWS API Gateway Resources page for the 'frauddetectionapi' API, showing the execution flow of a POST request.

**API Gateway** sidebar:

- APIs
- Custom domain names
- Domain name access associations
- VPC links
- APIs**:
  - frauddetectionapi
    - Resources
      - Stages
      - Authorizers
      - Gateway responses
      - Models
      - Resource policy
      - Documentation
      - Dashboard
      - API settings
    - Usage plans
    - API keys
    - Client certificates

**Resources** section:

- Create resource** button
- Path: /fraud-detection
- Method: POST
- /fraud-detection - POST - Method execution** panel:
  - ARN: arn:aws:execute-api:us-east-1:1597088015929:9tv4xz43g/\*/POST/fraud-detection
  - Resource ID: r697nu
- Execution Flow Diagram**:
  - Client → Method request → Integration request → Lambda integration
  - Method response ← Integration response ← Integration response (Proxy integration)
- Method request** tab is selected.
- Method request settings** panel:
  - Authorization: NONE
  - API key required: False

Bottom navigation bar:

- CloudShell
- Feedback
- © 2025, Amazon Web Services, Inc. or its affiliates.
- Privacy
- Terms
- Cookie preferences
- ENG IN 2004 29-03-2025

The screenshot shows the AWS API Gateway console. On the left, the navigation pane is open with the path: API Gateway > APIs > frauddetectionapi (9tv4xz43g) > Stages. The main area displays the 'Stages' configuration for the 'prod' stage. A tree view shows the endpoint structure: prod / /fraud-detection POST. To the right, there's a section titled 'Method overrides' with a note: 'By default, methods inherit stage-level settings. To customize settings for a method, configure method overrides.' It also shows an 'Invoke URL' with the address: https://9tv4xz43g.execute-api.us-east-1.amazonaws.com/prod/fraud-detection. A 'Stage actions' dropdown and a 'Create stage' button are at the top right. The bottom of the screen shows the standard AWS navigation bar.

## AWS SNS (Simple Notification Service)

- Sends fraud alerts via email/SMS.

### Screenshot:

The screenshot shows the AWS SNS console. The navigation pane on the left is open with the path: Amazon SNS > Topics. The main area displays a list of topics under the heading 'Topics (2)'. There are two entries:

Name	Type	ARN
EC2_S3_Alerts	Standard	arn:aws:sns:us-east-1:597088015929:EC2_S3_Alerts
FraudAlerts	Standard	arn:aws:sns:us-east-1:597088015929:FraudAlerts

A blue banner at the top states: '① New Feature Amazon SNS now supports High Throughput FIFO topics. Learn more' with a link icon. A 'Create topic' button is located at the top right. The bottom of the screen shows the standard AWS navigation bar.

The screenshot shows the AWS SNS console with the 'FraudAlerts' topic selected. The left sidebar shows navigation options like Dashboard, Topics, Subscriptions, and Mobile. The main panel displays the 'FraudAlerts' topic details, including its Name (FraudAlerts), ARN (arn:aws:sns:us-east-1:597088015929:FraudAlerts), and Type (Standard). It also shows the 'Display name' field and the 'Topic owner' (597088015929). Below this, the 'Subscriptions' tab is selected, showing one subscription entry:

ID	Endpoint	Status	Protocol
ff5c0336-c709-401c-af28-52b5fb5e72e3	shajith0829@gmail.com	Confirmed	EMAIL

The screenshot shows the AWS SNS console with the 'Subscription: ff5c0336-c709-401c-af28-52b5fb5e72e3' details page. The left sidebar shows the same navigation options as the previous screen. The main panel displays the subscription details, including its ARN (arn:aws:sns:us-east-1:597088015929:FraudAlerts:ff5c0336-c709-401c-af28-52b5fb5e72e3), Endpoint (shajith0829@gmail.com), Topic (FraudAlerts), and Subscription Principal (arn:aws:iam::597088015929:root). It also shows the Status (Confirmed) and Protocol (EMAIL). Below this, the 'Subscription filter policy' tab is selected, indicating 'No filter policy configured for this subscription.' To apply a filter policy, it suggests editing the subscription.

```

{
  "Version": "2008-10-17",
  "Id": "__default_policy_ID",
  "Statement": [
    {
      "Sid": "__default_statement_ID",
      "Effect": "Allow",
      "Principal": "*",
      "AWS": "*"
    },
    {
      "Action": [
        "SNS:GetTopicAttributes",
        "SNS:SetTopicAttributes",
        "SNS:AddPermission",
        "SNS:RemovePermission"
      ],
      "Resource": "arn:aws:sns:us-east-1:597088015929:FraudAlerts"
    }
  ]
}

```

The screenshot shows the AWS SNS console with the 'Access policy' tab selected. The policy document is displayed, granting full permissions to AWS services on the topic.

The screenshot shows the Gmail inbox with three messages from 'AWS Notifications'. The first message is a 'Fraud Detection Alert' about a transaction flagged as fraudulent. The second message is an unsubscribe notice. The third message is another notification about a transaction flagged as fraudulent.

## AWS CloudWatch

- Monitors API calls, Lambda execution logs, and SNS alerts.
- Detects unusual spikes in fraud attempts.

## Screenshot:

The screenshot shows the AWS CloudWatch Log groups interface. On the left, there's a navigation sidebar with sections like AI Operations, Alarms, Logs (Log groups, Log Anomalies, Live Tail, Logs Insights, Contributor Insights), Metrics, and Streams. The main area displays a table titled "Log groups (9)". The table has columns for "Log group", "Log class", "Anomaly d...", "Data pr...", and "Sensitiv...". Each row contains a link to a specific log group, such as "/aws/ecs/containerinsights/cartoon-app/performance" or "/aws/sagemaker/Endpoints/sagemaker-xgboost-2025-03-07-14-12-21-661". There are also "Configure" buttons for each row. At the top right, there are buttons for "Actions", "View in Logs Insights", "Start tailing", and "Create log group". The browser address bar shows the URL: https://us-east-1.console.aws.amazon.com/cloudwatch/home?region=us-east-1#logsV2/log-groups.

## Lambda Log:

The screenshot shows the AWS CloudWatch Log events interface for the Lambda function "InvokeSageMakerLambda". The left sidebar is identical to the previous screenshot. The main area is titled "Log events" and includes a filter bar with a search input, time range buttons (Clear, 1m, 30m, 1h, 12h, Custom, Local timezone), and a "Display" dropdown. Below the filter is a table with columns "Timestamp" and "Message". The table lists several log entries from March 26, 2025, at 19:54:323+05:30. The messages include JSON payloads related to SageMaker invocations. A message at the end indicates "No newer events at this moment. Auto retry paused. Resume". At the bottom right is a "Back to top" button. The browser address bar shows the URL: https://us-east-1.console.aws.amazon.com/cloudwatch/home?region=us-east-1#logsV2/log-groups/log-group/\$25faws\$252fLambda\$252fInvoke\$.

## Preprocessing Log:

The screenshot shows the AWS CloudWatch Log Groups interface. The left sidebar navigation includes 'CloudWatch' (selected), 'Favorites and recents', 'Dashboards', 'AI Operations' (Preview), 'Alarms' (0), 'Logs' (selected), 'Log groups' (New), 'Log Anomalies', 'Live Tail', 'Logs Insights' (New), 'Contributor Insights', 'Metrics' (All metrics), 'Explorer', and 'Streams'. The main content area is titled 'Log events' and displays a table with columns 'Timestamp' and 'Message'. The table shows several log entries from March 2025, such as INIT\_START, START, END, and REPORT requests. A message at the bottom indicates 'Loading newer events'. The top navigation bar shows tabs for 'Amazon SageMail', 'FraudAlerts | Top', 'CloudWatch' (selected), 'preprocessTrans...', 'Instances | EC2', 'Home', 'Untitled', '(2) WhatsApp', 'Credit Card Fraud', and 'Credit Card Fraud'. The address bar is https://us-east-1.console.aws.amazon.com/cloudwatch/home?region=us-east-1#logsV2:log-groups/log-group/\$252faws\$252fLambda\$252fpreprocessTrans... The status bar at the bottom right shows 'ENG IN' and the date '29-03-2025'.

## Sagemaker Jupyter Notebook Log:

The screenshot shows the AWS CloudWatch Log Groups interface, similar to the previous one but for a different log group. The left sidebar navigation is identical. The main content area is titled 'Log events' and displays a table with columns 'Timestamp' and 'Message'. The table shows log entries from March 2025, primarily related to kernel connections and file saving. A message at the bottom indicates 'No newer events at this moment. Auto retry paused. Resume'. The top navigation bar shows tabs for 'Amazon SageMail', 'FraudAlerts | Top', 'CloudWatch' (selected), 'preprocessTrans...', 'Instances | EC2', 'Home', 'Untitled', '(2) WhatsApp', 'Credit Card Fraud', and 'Credit Card Fraud'. The address bar is https://us-east-1.console.aws.amazon.com/cloudwatch/home?region=us-east-1#logsV2:log-groups/log-group/\$252faws\$252fSagemaker\$252fNotebookInstances... The status bar at the bottom right shows 'ENG IN' and the date '29-03-2025'.

## Sagemaker Endpoint Logs:

The screenshot shows the AWS CloudWatch Log Events interface. The left sidebar navigation includes CloudWatch, Favorites and recents, Dashboards, AI Operations (with Alarms, Logs, Metrics), and CloudShell. The main content area displays log events for the path /aws/sagemaker/Endpoints/sagemaker-xgboost-2025-03-29-14-29-22-965. The log events table has columns for Timestamp and Message. The messages show multiple GET requests to the endpoint, all returning a 200 status code and the response "HTTP/1.1". The timestamp for the first event is 2025-03-29T20:08:29.307+05:30.

Timestamp	Message
2025-03-29T20:08:29.307+05:30	169.254.178.2 - - [29/Mar/2025:14:38:24 +0000] "GET /ping HTTP/1.1" 200 0 "-" "AHC/2.0"
2025-03-29T20:08:34.308+05:30	169.254.178.2 - - [29/Mar/2025:14:38:29 +0000] "GET /ping HTTP/1.1" 200 0 "-" "AHC/2.0"
2025-03-29T20:08:39.307+05:30	169.254.178.2 - - [29/Mar/2025:14:38:34 +0000] "GET /ping HTTP/1.1" 200 0 "-" "AHC/2.0"
2025-03-29T20:08:44.346+05:30	169.254.178.2 - - [29/Mar/2025:14:38:39 +0000] "GET /ping HTTP/1.1" 200 0 "-" "AHC/2.0"
2025-03-29T20:08:49.308+05:30	169.254.178.2 - - [29/Mar/2025:14:38:44 +0000] "GET /ping HTTP/1.1" 200 0 "-" "AHC/2.0"
2025-03-29T20:08:54.307+05:30	169.254.178.2 - - [29/Mar/2025:14:38:49 +0000] "GET /ping HTTP/1.1" 200 0 "-" "AHC/2.0"
2025-03-29T20:08:59.307+05:30	169.254.178.2 - - [29/Mar/2025:14:38:54 +0000] "GET /ping HTTP/1.1" 200 0 "-" "AHC/2.0"
2025-03-29T20:09:04.307+05:30	169.254.178.2 - - [29/Mar/2025:14:38:59 +0000] "GET /ping HTTP/1.1" 200 0 "-" "AHC/2.0"
2025-03-29T20:09:09.307+05:30	169.254.178.2 - - [29/Mar/2025:14:39:04 +0000] "GET /ping HTTP/1.1" 200 0 "-" "AHC/2.0"
2025-03-29T20:09:14.307+05:30	169.254.178.2 - - [29/Mar/2025:14:39:09 +0000] "GET /ping HTTP/1.1" 200 0 "-" "AHC/2.0"

## AWS IAM (Identity and Access Management)

- Restricts access to SageMaker, Lambda, and API Gateway.
- Ensures secure API calls.

### Screenshot:

The screenshot shows the AWS IAM Roles page. The left sidebar navigation includes Identity and Access Management (IAM) (with a Search IAM button), Access management (User groups, Users, Roles, Policies), Access reports (Access Analyzer, External access, Unused access, Analyzer settings, Credential report, Organization activity), and CloudShell/Feedback. The main content area displays a table of roles with 11 entries. The columns are Role name, Trusted entities, and Last activity. The last activity column shows times ranging from 7 minutes ago to 24 days ago. Some roles are Service-Linked Roles.

Role name	Trusted entities	Last activity
AmazonSageMaker-ExecutionRole-20250305T160965	AWS Service: sagemaker	7 minutes ago
AWSServiceRoleForAPIGateway	AWS Service: ops.apigateway	-
AWSServiceRoleForAutoScaling	AWS Service: autoscaling	4 hours ago
AWSServiceRoleForECS	AWS Service: ecs	4 hours ago
AWSServiceRoleForLexV2Bots_QLEPGV1JFSA	AWS Service: lexv2	-
AWSServiceRoleForSupport	AWS Service: support	-
AWSServiceRoleForTrustedAdvisor	AWS Service: trustedadvisor	-
creditcardproject	AWS Service: lambda	24 days ago
ecsInstanceRole	AWS Service: ec2	-
ecsTaskExecutionRole	AWS Service: ecs-tasks	4 hours ago
InvokeSageMakerLambda-role-o1lnddk	AWS Service: lambda	3 days ago

The screenshot shows the AWS IAM Policies page. On the left, there's a navigation sidebar for Identity and Access Management (IAM) with sections like Dashboard, Access management, Policies, Access reports, and CloudShell. The main area displays a table titled "Policies (1341) Info" with columns for Policy name, Type, Used as, and Description. The table lists various AWS managed policies such as AccessAnalyzerServiceRole, AdministratorAccess, and AlexaForBusinessGateway. A search bar and filter by type dropdown are at the top of the table.

The screenshot shows the AWS IAM Roles page for the "creditcardproject" role. The left sidebar is identical to the previous screenshot. The main area shows the "Permissions" tab selected. It displays a table for "Permissions policies (3) Info" with columns for Policy name, Type, and Attached entities. Three AWS managed policies are listed: AmazonDynamoDBFullAccess, AmazonS3FullAccess, and AWSLambdaBasicExecutionRole. Below the table, there are sections for "Permissions boundary (not set)" and "Generate policy based on CloudTrail events". The bottom right corner shows standard browser controls and a timestamp of 29-03-2025.

## Amazon S3

- Stores datasets for fraud detection model training.
- Stores preprocessed transaction data from Lambda.

## Screenshot:

The screenshot shows the AWS S3 console interface. On the left, there's a sidebar with navigation links for General purpose buckets, Directory buckets, Table buckets, Access Grants, Access Points, Object Lambda Access Points, Multi-Region Access Points, Batch Operations, IAM Access Analyzer for S3, Storage Lens (Dashboards, Storage Lens groups, AWS Organizations settings), and Feature spotlight (11).

The main content area displays an "Account snapshot - updated every 24 hours" for all AWS Regions. It includes a link to "View Storage Lens dashboard". Below this, there are tabs for "General purpose buckets" and "Directory buckets", with "General purpose buckets" selected. It shows a list of two buckets: "cloudassignmentbucket" and "raw-credit-card-data". Each bucket entry includes its name, AWS Region (US East (N. Virginia) us-east-1), IAM Access Analyzer (links to view analyzers), and Creation date.

At the bottom, there are standard browser controls and a footer with copyright information and language settings (ENG IN).

This screenshot shows the AWS S3 console with the URL https://us-east-1.console.aws.amazon.com/s3/buckets/raw-credit-card-data?region=us-east-1&bucketType=general&tab=objects. The sidebar is identical to the first screenshot.

The main content area shows the "Objects (8)" list. It includes a header with actions like Copy S3 URI, Copy URL, Download, Open, Delete, Actions, Create folder, and Upload. Below the header is a search bar and a table with columns: Name, Type, Last modified, Size, and Storage class. The table lists eight objects: "balanced\_fraud\_data.csv" (csv, March 6, 2025, 21:50:30, 3.6 MB, Standard), "Doc1.docx" (docx, March 8, 2025, 17:03:47, 1.2 MB, Standard), "fraud\_detection\_model.pkl" (pkl, March 5, 2025, 16:21:46, 360.6 KB, Standard), "fraudTest.csv" (csv, March 5, 2025, 15:31:39, 143.4 MB, Standard), "fraudTrain.csv" (csv, March 5, 2025, 15:31:39, 335.0 MB, Standard), "model.tar.gz" (gz, March 29, 2025, 19:59:07, 111.5 KB, Standard), "processed\_fraud\_data.csv" (csv, March 5, 2025, 16:07:26, 308.7 MB, Standard), and "test.txt" (txt, March 8, 2025, 16:58:47, 0 B, Standard).

At the bottom, there are standard browser controls and a footer with copyright information and language settings (ENG IN).

## AWS EC2

- Hosts Flask-based UI for user transactions.
- Runs Nginx and Gunicorn for backend support.

## Screenshot:

The screenshot shows the AWS CloudWatch Instances console. On the left, a sidebar menu is open under the 'EC2' section, showing options like Dashboard, EC2 Global View, Events, Instances, Images, and Elastic Block Store. The 'Instances' option is selected. The main pane displays a table titled 'Instances (1/1)'. A green banner at the top says 'Successfully initiated starting of i-0b341c1ab51ebdb96'. The table has columns for Name, Instance ID, Instance state, Instance type, Status check, Alarm status, Availability Zone, and Public IPv4 DNS. One row is visible for the instance 'fraud-detectio...', which is 'Running', of type 't2.micro', and has an 'Initializing' status check. It is located in 'us-east-1a' with the Public IPv4 DNS 'ec2-34-201-4x'. Below the table, a detailed view for the instance 'i-0b341c1ab51ebdb96 (fraud-detection-app)' is shown, with tabs for Details, Status and alarms, Monitoring, Security, Networking, Storage, and Tags. The 'Details' tab is selected, showing the Instance ID 'i-0b341c1ab51ebdb96', Public IPv4 address '34.201.44.204', Private IPv4 address '172.31.22.15', and Instance state 'Running'. At the bottom of the page, there are links for CloudShell and Feedback, along with a footer with copyright information and system status icons.

```

[ec2-user@ip-172-31-22-15 ~]$ ls
cloud
[ec2-user@ip-172-31-22-15 ~]$ ls cloud
README.md  __pycache__  app.py  templates
[ec2-user@ip-172-31-22-15 ~]$ sudo systemctl status nginx
● nginx.service - The nginx HTTP and reverse proxy server
   Loaded: loaded (/usr/lib/systemd/system/nginx.service; enabled; preset: disabled)
     Active: active (running) since Sat 2025-03-29 14:26:51 UTC; 18min ago
   Process: 2144 ExecStartPre=/usr/bin/rm -f /run/nginx.pid (code=exited, status=0/SUCCESS)
   Process: 2149 ExecStartPre=/usr/sbin/nginx -t (code=exited, status=0/SUCCESS)
   Process: 2163 ExecStart=/usr/sbin/nginx (code=exited, status=0/SUCCESS)
 Main PID: 2172 (nginx)
    Tasks: 2 (limit: 1111)
      Memory: 5.5M
        CPU: 40ms
       CGroup: /system.slice/nginx.service
           └─2172 "nginx: master process /usr/sbin/nginx"
             ├─2173 "nginx: worker process"

Mar 29 14:26:51 ip-172-31-22-15.ec2.internal systemd[1]: Starting nginx.service - The nginx HTTP and reverse proxy server...
Mar 29 14:26:51 ip-172-31-22-15.ec2.internal nginx[2149]: nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
Mar 29 14:26:51 ip-172-31-22-15.ec2.internal nginx[2149]: nginx: configuration file /etc/nginx/nginx.conf test is successful
Mar 29 14:26:51 ip-172-31-22-15.ec2.internal systemd[1]: Started nginx.service - The nginx HTTP and reverse proxy server.
[ec2-user@ip-172-31-22-15 ~]$ 

```

This screenshot shows a terminal window with a black background and white text. It displays the output of several commands: 'ls' showing directory contents 'cloud', 'ls cloud' showing files 'README.md', '\_\_pycache\_\_', 'app.py', and 'templates', and the command 'sudo systemctl status nginx'. The output of 'systemctl status nginx' shows the nginx service is active and running. Log entries at the bottom show the service starting and testing its configuration. The terminal window has a standard Windows-style title bar and taskbar at the bottom.

```
ec2-user@ip-172-31-22-15:~
```

```
location / {
    proxy_pass http://127.0.0.1:5000;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
}

# Settings for a TLS enabled server.
server {
    listen      443 ssl;
    listen      [::]:443 ssl;
    http2       on;
    server_name ;
    root        /usr/share/nginx/html;

    ssl_certificate "/etc/pki/nginx/server.crt";
    ssl_certificate_key "/etc/pki/nginx/private/server.key";
    ssl_session_cache shared:SSL:lm;
    ssl_session_timeout 10m;
    ssl_ciphers PROFILE=SYSTEM;
    ssl_prefer_server_ciphers on;

    # Load configuration files for the default server block.
    include /etc/nginx/default.d/*.conf;

    error_page 404 /404.html;
    location = /404.html {
    }

    error_page 500 502 503 504 /50x.html;
    location = /50x.html {
    }
}
[ec2-user@ip-172-31-22-15 ~]$
```



```
ec2-user@ip-172-31-22-15:~/cloud
```

```
[ec2-user@ip-172-31-22-15 ~]$ cd cloud
[ec2-user@ip-172-31-22-15 cloud]$ ls
README.md  pycache  app.py  templates
[ec2-user@ip-172-31-22-15 cloud]$ gunicorn --workers 3 --bind 0.0.0.0:8000 app:app
[2025-03-29 14:47:40 +0000] [2956] [INFO] Starting gunicorn 23.0.0
[2025-03-29 14:47:40 +0000] [2956] [INFO] Listening at: http://0.0.0.0:8000 (2956)
[2025-03-29 14:47:40 +0000] [2956] [INFO] Using worker: sync
[2025-03-29 14:47:40 +0000] [2957] [INFO] Booting worker with pid: 2957
[2025-03-29 14:47:40 +0000] [2958] [INFO] Booting worker with pid: 2958
[2025-03-29 14:47:40 +0000] [2959] [INFO] Booting worker with pid: 2959
```



**Credit Card Fraud Detection**

Enter transaction details to check for fraud.

Transaction Amount:

Gender (0 = Male, 1 = Female):

Category Entertainment (0 or 1):

Category Food & Dining (0 or 1):

Category Gas & Transport (0 or 1):

Category Grocery Net (0 or 1):

Category Health & Fitness (0 or 1):

Category Home (0 or 1):

Category Travel (0 or 1):

**Fraud Detection Alert**

AWS Notifications <no-reply@sns.amazonaws.com>  
to me

**FRAUD ALERT**  
Transaction flagged as fraudulent  
Fraud Score: 0.008493025787174702

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-1:597088015929:FraudAlerts:f5c0336-c709-401c-af28-52b5b5e72e3&Endpoint=shaikh0829@gmail.com>

Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact us at <https://aws.amazon.com/support>

Enable desktop notifications for Gmail. **OK** **No thanks**

Transaction flagged as fraudulent!

#### **4. Coding:**

##### **Lambda:**

###### **InvokeSageMakerLambda**

```
import json
import boto3

sagemaker_runtime = boto3.client("sagemaker-runtime")
sns_client = boto3.client("sns")

SNS_TOPIC_ARN = "arn:aws:sns:us-east-1:597088015929:FraudAlerts"

def lambda_handler(event, context):
    try:
        print("Received event:", json.dumps(event, indent=2))

        body = json.loads(event["body"]) if "body" in event else {}
        if "features" not in body:
            return {"statusCode": 400, "body": json.dumps({"error": "Missing 'features' in request body"})}

        payload = ",".join(map(str, body["features"]))

        response = sagemaker_runtime.invoke_endpoint(
            EndpointName="sagemaker-xgboost-2025-03-26-07-11-30-990",
            ContentType="text/csv",
            Body=payload
        )

        prediction_score = float(response["Body"].read().decode("utf-8"))
        is_fraud = prediction_score > 0.005

        if is_fraud:
            message = f"⚠️ FRAUD ALERT ⚠️ \nTransaction flagged as fraudulent!\nFraud Score: {prediction_score}"
            sns_client.publish(TopicArn=SNS_TOPIC_ARN, Message=message, Subject="Fraud Detection Alert")

        return {
            "statusCode": 200,
            "body": json.dumps({"is_fraud": is_fraud, "score": prediction_score})
        }

    except Exception as e:
        return {"statusCode": 500, "body": json.dumps({"error": str(e)})}
```

### **preprocessTransactionLambda:**

```
import json
import boto3
import pandas as pd
from io import StringIO

s3 = boto3.client('s3')

BUCKET_NAME = "raw-credit-card-data"
RAW_FILE_KEY = "fraudTrain.csv"
PROCESSED_FILE_KEY = "processed_fraud_data.csv"

def lambda_handler(event, context):
    try:

        obj = s3.get_object(Bucket=BUCKET_NAME, Key=RAW_FILE_KEY)
        df_iter = pd.read_csv(obj['Body'], chunksize=5000)

        processed_chunks = []

        for chunk in df_iter:

            columns_to_drop = ['first', 'last', 'street', 'city', 'state', 'zip', 'dob']
            chunk = chunk.drop(columns=[col for col in columns_to_drop if col in
            chunk.columns], errors='ignore')

            chunk['trans_date_trans_time'] = pd.to_datetime(chunk['trans_date_trans_time'])
            chunk['unix_time'] = chunk['trans_date_trans_time'].astype(int) / 10**9

            if 'gender' in chunk.columns:
                chunk['gender'] = chunk['gender'].map({'M': 0, 'F': 1})

            if 'category' in chunk.columns:
                chunk = pd.get_dummies(chunk, columns=['category'])

            numerical_features = ['amt', 'merch_lat', 'merch_long']
            for feature in numerical_features:
                if feature in chunk.columns:
                    chunk[feature] = (chunk[feature] - chunk[feature].min()) /
                    (chunk[feature].max() - chunk[feature].min())

            processed_chunks.append(chunk)

        final_df = pd.concat(processed_chunks, ignore_index=True)
```

```
csv_buffer = StringIO()
final_df.to_csv(csv_buffer, index=False)
s3.put_object(Bucket=BUCKET_NAME, Key=PROCESSED_FILE_KEY,
Body=csv_buffer.getvalue())

return {
    'statusCode': 200,
    'body': json.dumps(f'Preprocessing Complete! Processed file saved in S3 as
{PROCESSED_FILE_KEY}')
}

except Exception as e:
    return {
        'statusCode': 500,
        'body': json.dumps(f"Error: {str(e)}")
    }
```

## 5. Conclusion

This project successfully implements real-time credit card fraud detection using **AWS cloud services**. The system is capable of accurately identifying fraudulent transactions, alerting users through SNS, and ensuring secure transaction monitoring via CloudWatch and IAM policies. The project demonstrates how AWS services can be effectively integrated to build a scalable, high-performance fraud detection system.

Furthermore, this system provides real-time fraud prevention capabilities that enhance the security of financial transactions. Future improvements will focus on:

- Enhancing machine learning model **accuracy** by training with larger datasets.
- Implementing real-time dashboards using AWS QuickSight for better analytics.
- Utilizing Auto Scaling to dynamically adjust system resources.
- Strengthening **security measures** through enhanced IAM policies and encryption.

By leveraging AWS's cloud-based architecture, this project provides a cost-efficient, scalable, and effective fraud detection system that can be further expanded to meet evolving security challenges in the financial industry.

## **References:**

- [1] Amazon Web Services (AWS) Documentation - <https://docs.aws.amazon.com/>
- [2] Machine Learning for Fraud Detection, AWS Blog - <https://aws.amazon.com/blogs/>
- [3] Real-time Analytics with Amazon Kinesis - <https://aws.amazon.com/kinesis/>
- [4] Secure Cloud Storage with AWS S3 - <https://aws.amazon.com/s3/>
- [5] Best Practices for Fraud Prevention - <https://www.fraudpractice.com/>
- [6] Identity and Access Management (IAM) Best Practices - <https://aws.amazon.com/iam/>
- [7] Implementing Serverless Architectures - <https://aws.amazon.com/serverless/>