# Detecting Fraudulent Transactions in Real-Time

GitHub Link to Project: [kartik08/ssense\_assignment](https://github.com/kartik08/ssense_assignment)

### Overview

This document outlines the design and implementation of a real time fraud detection pipeline for an e-commerce company. The pipeline is design to ingest live transaction data, perform feature engineering, and scoring/detecting fraudulent transaction using simple logistic regression traditional machine learning model. The pipeline leverages AWS Cloud ecosystem services for scalability, reliability and performance. This document also includes automation of infrastructure deployment, model deployment using GitHub actions.

### Key Objective

* Ingest live transaction data in real time using AWS Kinesis.
* Perform Feature engineering on data using Lambda Function.
* Storing transaction data for historical analysis and model fine tuning.
* Detection of fraud using a ML model deployed via AWS SageMaker and Inference using AWS Lambda Function.
* Trigger alerts for flagged transaction using AWS SNS.
* Automation using Terraform and GitHub Action.

### Architecture Overview:

* **Data Ingestion:** Amazon Kinesis Data Streams collects real-time transactions.
* **Processing Layer:** AWS Lambda as a consumer, and than orchestration of Data Validation, Feature Generation, Fraud Detection and Data loading using Step Function and Lambda Function.
* **Fraud Detection:** AWS SageMaker runs a trained ML model to classify transactions.
* **Storage Layer:** Amazon S3 stores historical transactions, while DynamoDB holds real-time lookups.
* Alerting Layer (Optional): SNS, Event Bridge to Cancel Transaction.

### Diagram:

A diagram of a software system

AI-generated content may be incorrect.

### Assumption for Design:

* Location data is stored in RDS for location mismatch, and no third-party location API is used.
* All Data is encrypted at rest and while transit using KMS Key.
* User previous orders information is stored in RDS/Feature Store to calculate the high spending rate feature.
* All features required for inferencing are collected in the feature processing step.
* On-schedule, data will be moved from DynamoDB to S3.
* Scaling Lambda functions: Hot start and concurrent scaling will be implemented.
* Fraud-detected transaction consumption to the application server from DynamoDB is not displayed.
* Separate IAM role are designed for each service mentioned in Diagram.
* Monitoring is integrated via Cloud Watch or any 3rd party application like Datadog, Dynatrace, etc.

### Trade Offs:

* Amazon Kinesis Data Streams:
  1. The cost for this service will be higher for high-volume transactions.
  2. Kafka provides more customizations compared to Kinesis.
  3. Maximum 7 days of data retention.
  4. Enhanced fan-out incurs additional costs.
  5. Data Ordering: Ensuring data ordering can be complex, especially with multiple shards.
* AWS Lambda and Step Functions
  1. Cold Start of Lambda will introduce latency. So will require hot start of Lambda Functions.
  2. We need to be mindful of Lambda's concurrency limits, especially during peak loads.
  3. Implement robust error handling and retries within Step Functions to ensure reliable execution.
* Amazon S3 and DynamoDB
  1. Delayed data for historical data analysis from S3.

### Infrastructure Setup:

* Used Terraform to set up infrastructure.
* Built a small pipeline using Kinesis, Lambda, and S3 for data loading.
* Attached the code as an appendix, along with a ZIP file and a demo video.

### CI/CD Pipeline:

1. Integrated Terraform with GitHub Actions.
2. GitHub Action Steps:
   1. When to run the GitHub Action: On push to the master branch, with input comments as apply or destroy in the PR.
   2. Set up a Terraform backend with an S3 bucket to store the state of resources deployed, as GitHub runners don’t have persistent storage.
   3. Run the GitHub plan.
   4. Run GitHub apply or destroy based on the comments.
   5. Run GitHub output.
3. Attached the code as an appendix, along with a ZIP file and a demo video.

### Machine Learning Workflow:

* Create a simple Scikit-learn Logistic Regression Model.
* Steps:
  1. Generated synthetic data.
  2. Prepared data (mainly label encoding) and performed feature selection.
  3. Split the data into training and test sets using a 75-25 ratio.
  4. Trained the model.
  5. Ran the model on the test data.
  6. Generated a confusion matrix and metrics like accuracy and precision.
  7. Saved the model as a .pkl file.
* Attached the code as an appendix, along with a ZIP file and a demo video.
* Output From Code:
  1. accuracy: 0.904
  2. confusion matrix:

[[2260 0]

[ 240 0]]

* 1. precision: 0.90

### Step to Run Code:

#### Terraform

* 1. Configure your AWS Credentials.
  2. Run Terraform Init to initialize the repo.
  3. Run Terraform plan
  4. Run Terraform apply to deploy all resource.
  5. Run Terraform destroy to remove all resource

#### Pipeline Test

1. Configure AWS Cred at your local laptop.
2. Install all libs from requirement.txt file
3. Run KinesisTest.py file
4. Check file in S3 Bucket.

#### Model File

1. Update location in Data Generator File
2. Run Data Generator file.
3. Update location in Model file
4. Run Model File

# Appendix

### Infrastructure Setup:

#### Terraform Code:

##### IAM

resource "aws\_iam\_role" "lambda\_role" {

  name = "lambda\_execution\_role"

  assume\_role\_policy = jsonencode({

    Statement = [{

      Action = "sts:AssumeRole"

      Effect = "Allow"

      Principal = { Service = "lambda.amazonaws.com" }

    }]

  })

}

resource "aws\_iam\_policy" "lambda\_cloudwatch" {

  name        = "LambdaCloudWatchPolicy"

  description = "IAM policy for Lambda to access Cloudwatch"

  policy      = jsonencode({

    Version = "2012-10-17",

    Statement = [

      {

        Effect = "Allow",

        Action = [

          "logs:CreateLogGroup",

          "logs:CreateLogStream",

          "logs:PutLogEvents"

        ],

        Resource = "arn:aws:logs:ca-central-1:279754971650:\*"

      }

    ]

  })

}

resource "aws\_iam\_policy" "lambda\_kinesis" {

     name        = "LambdaKinesisPolicy"

     description = "IAM policy for Lambda to access Kinesis"

     policy      = jsonencode({

     Version = "2012-10-17",

     Statement = [

      {

        Effect = "Allow",

        Action = [

          "kinesis:Subscribe\*",

          "kinesis:Get\*",

          "kinesis:List\*",

          "kinesis:Describe\*"

        ],

        Resource = aws\_kinesis\_stream.transaction\_stream.arn

      }

    ]

  })

}

resource "aws\_iam\_policy" "lambda\_s3" {

     name        = "LambdaS3Policy"

     description = "IAM policy for Lambda to access S3"

     policy      = jsonencode({

     Version = "2012-10-17",

     Statement = [

      {

        Effect = "Allow",

        Action = [

          "s3:\*",

        ],

        Resource = "${aws\_s3\_bucket.fraud\_data.arn}/\*"

      }

    ]

  })

}

resource "aws\_iam\_role\_policy\_attachment" "lambda\_policy\_attachment\_s3" {

  policy\_arn = aws\_iam\_policy.lambda\_s3.arn

  role = aws\_iam\_role.lambda\_role.name

}

resource "aws\_iam\_role\_policy\_attachment" "lambda\_policy\_attachment\_cloudwatch" {

    policy\_arn = aws\_iam\_policy.lambda\_cloudwatch.arn

    role = aws\_iam\_role.lambda\_role.name

}

resource "aws\_iam\_role\_policy\_attachment" "lambda\_policy\_attachment\_kinesis" {

    policy\_arn = aws\_iam\_policy.lambda\_kinesis.arn

    role = aws\_iam\_role.lambda\_role.name

}

##### S3

resource "aws\_s3\_bucket" "fraud\_data" {

  bucket = "fraud-detection-data-kartik"

  acl    = "private"

}

resource "aws\_s3\_bucket\_policy" "fraud\_data\_policy" {

  bucket = aws\_s3\_bucket.fraud\_data.id

  policy = jsonencode({

    Version = "2012-10-17",

    Statement = [

      {

        Effect = "Deny",

        Principal = "\*",

        Action = "s3:\*",

        Resource = [

          "${aws\_s3\_bucket.fraud\_data.arn}/\*",

          "${aws\_s3\_bucket.fraud\_data.arn}"

        ],

        Condition = {

          Bool = {

            "aws:SecureTransport": "false"

          }

        }

      },

      {

        Effect = "Allow",

        Principal = {

          AWS = aws\_iam\_role.lambda\_role.arn

        },

        Action = ["s3:Get\*", "s3:Put\*"],

        Resource = "${aws\_s3\_bucket.fraud\_data.arn}/\*"

      }

    ]

  })

}

##### Lambda:

resource "aws\_lambda\_function" "kinesis\_consumer" {

  function\_name = "FraudDetectionConsumer"

  handler       = "lambda\_function.lambda\_handler"

  runtime       = "python3.8"

  role          = aws\_iam\_role.lambda\_role.arn

  filename      =  "lambda\_artifacts/lambda\_function.zip"

  environment {

    variables = {

      KINESIS\_STREAM\_NAME = aws\_kinesis\_stream.transaction\_stream.name

      S3\_BUCKET\_NAME = aws\_s3\_bucket.fraud\_data.bucket

    }

  }

}

resource "aws\_lambda\_event\_source\_mapping" "kinesis\_trigger" {

  event\_source\_arn  = aws\_kinesis\_stream.transaction\_stream.arn

  function\_name     = aws\_lambda\_function.kinesis\_consumer.arn

  starting\_position = "TRIM\_HORIZON"

}

##### Kinesis:

resource "aws\_kinesis\_stream" "transaction\_stream" {

  name             = "fraud-detection-stream"

  retention\_period = 24

   stream\_mode\_details {

    stream\_mode = "ON\_DEMAND"

  }

  tags = {

    Environment = "test"

  }

}

#### CI/CD:

##### Terraform

name: Terraform Deployment

on:

  issue\_comment:

    types: [created]

jobs:

  terraform:

    runs-on: ubuntu-latest

    steps:

      # Check if the comment is on a pull request and contains /apply or /destroy

      - name: Check if PR comment contains /apply or /destroy

        if: github.event.issue.pull\_request != null && (contains(github.event.comment.body, '/apply') || contains(github.event.comment.body, '/destroy'))

        run: |

          echo "PR comment contains /apply or /destroy, proceeding with the respective action."

      - name: Checkout code

        uses: actions/checkout@v2

      # Set up Terraform

      - name: Set up Terraform

        uses: hashicorp/setup-terraform@v1

         # Zip the file before running Terraform

      - name: Zip a specific file

        run: |

          zip lambda\_artifacts/lambda\_function.zip lambda\_artifacts/lambda\_function.py

      # Configure AWS

      - name: Configure AWS credentials

        uses: aws-actions/configure-aws-credentials@v1

        with:

          aws-access-key-id: ${{ secrets.AWS\_ACCESS\_KEY\_ID }}

          aws-secret-access-key: ${{ secrets.AWS\_SECRET\_ACCESS\_KEY }}

          aws-region: ${{ secrets.AWS\_DEFAULT\_REGION }}

      - name: Setup Terraform backend

        run: terraform init -backend-config="bucket=terraform-kartik" -backend-config="key=terraform.tfstate" -backend-config="region=${{ secrets.AWS\_DEFAULT\_REGION }}"

      # Plan Terraform Deployment

      - name: Terraform Plan

        run: terraform plan

     # Run Terraform Apply if the comment contains /apply

      - name: Terraform Apply

        run: terraform apply -auto-approve

        if: github.event.issue.pull\_request != null && contains(github.event.comment.body, '/apply')

      # Run Terraform Destroy if the comment contains /destroy

      - name: Terraform Destroy

        run: terraform destroy -auto-approve

        if: github.event.issue.pull\_request != null && contains(github.event.comment.body, '/destroy')

      # Optional: Terraform Output

      - name: Terraform Output

        run: terraform output

### Machine Learning Workflow:

#### Data Generator

import uuid

import random

from datetime import datetime, timedelta

import pandas as pd

locations = ["California, USA", "New York, USA", "Texas, USA", "Ontario, Canada", "London, UK"]

device\_types = ["mobile", "desktop", "tablet"]

card\_types = ["credit", "debit", "prepaid"]

def generate\_transaction():

    return {

        "transaction\_id": f"T{uuid.uuid4().hex[:6].upper()}",

        "user\_id": f"U{random.randint(10000, 99999)}",

        "timestamp": (datetime.utcnow() - timedelta(minutes=random.randint(1, 10000))).isoformat() + "Z",

        "amount": round(random.uniform(10, 1000), 2),

        "device\_type": random.choice(device\_types),

        "location": random.choice(locations),

        "is\_vpn": random.choice([True, False]),

        "card\_type": random.choice(card\_types),

        "status": random.choice(["approved", "declined"]),

        # For 5%  Fraud Data

        "is\_fraud": random.choices([0, 1],weights=[90, 10])[0]

    }

noOfRows = 10000

dataFrame = pd.DataFrame()

for i in range(noOfRows):

    data = pd.DataFrame([generate\_transaction()])

    dataFrame = pd.concat([dataFrame,data],ignore\_index=True)

dataFrame.to\_csv("D:/SSense/data.csv")

#### Model Code

import pandas as pd

import numpy as np

import joblib

from datetime import datetime, timedelta

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report, roc\_auc\_score

#Read data

data = pd.read\_csv("D:\SSense\data.csv")

#Data Cleaning and prep

data.drop(["transaction\_id", "user\_id", "timestamp", "status"], axis=1, inplace=True)

for col in ["device\_type", "location", "card\_type"]:

    le = LabelEncoder()

    data[col] = le.fit\_transform(data[col])

data["is\_vpn"] = data["is\_vpn"].astype(int)

#SplitData

features = data.drop("is\_fraud", axis=1)

output = data["is\_fraud"]

x\_train,x\_test,y\_train,y\_test = train\_test\_split(features,output,test\_size=.25)

# ScaleFeatures

scaler = StandardScaler()

x\_train = scaler.fit\_transform(x\_train)

x\_test = scaler.transform(x\_test)

# TrainLogisticModel

model = LogisticRegression()

model.fit(x\_train, y\_train)

# MakePredictions

y\_pred = model.predict(x\_test)

y\_prob = model.predict\_proba(x\_test)[:, 1]

# Model Evaluation

print("accuracy:", accuracy\_score(y\_test, y\_pred))

print("confusion natrix:\n", confusion\_matrix(y\_test, y\_pred))

print("classificaion report:\n", classification\_report(y\_test, y\_pred))

print("ROC AUC Score:", roc\_auc\_score(y\_test, y\_prob))

#Saving model

joblib.dump(model,"D:/SSense/model.pkl")

### Data Ingestion Pipeline and Lambda S3 Write

#### Data Ingestion Pipeline

import boto3

import json

import uuid

import random

from datetime import datetime, timedelta

import time

kinesis = boto3.client('kinesis', region\_name='ca-central-1')

stream\_name = 'fraud-detection-stream'

locations = ["California, USA", "New York, USA", "Texas, USA", "Ontario, Canada", "London, UK"]

device\_types = ["mobile", "desktop", "tablet"]

card\_types = ["credit", "debit", "prepaid"]

def generate\_transaction():

    return {

        "transaction\_id": f"T{uuid.uuid4().hex[:6].upper()}",

        "user\_id": f"U{random.randint(10000, 99999)}",

        "timestamp": (datetime.utcnow() - timedelta(minutes=random.randint(1, 10000))).isoformat() + "Z",

        "amount": round(random.uniform(10, 1000), 2),

        "device\_type": random.choice(device\_types),

        "location": random.choice(locations),

        "is\_vpn": random.choice([True, False]),

        "card\_type": random.choice(card\_types),

        "status": random.choice(["approved", "declined",])

    }

while True:

    # Generating Random Data

    dataGenerated = generate\_transaction()

    # Putting Kinesis stream

    try:

        response = kinesis.put\_record(

            StreamName=stream\_name,

            Data=json.dumps(dataGenerated),

            PartitionKey=str(dataGenerated['user\_id'])

        )

    except:

        print("Error")

    print(response)

    print(f"Sent: {dataGenerated}")

    time.sleep(0.5)

    break

#### Lambda S3 Write

import base64

import json

import boto3

import os

def lambda\_handler(event, context):

    bucket\_name =os.environ['S3\_BUCKET\_NAME']

    s3\_client = boto3.client('s3')

    if 'Records' in event:

        for record in event['Records']:

            data = base64.b64decode(record['kinesis']['data']).decode('utf-8')

            data\_dict = json.loads(data)

            file\_key = f'{data\_dict["user\_id"]}/{data\_dict["timestamp"]}.json'

            print(f"Processed: {data\_dict}")

            data\_json = json.dumps(data\_dict)

            try:

                s3\_client.put\_object(

                    Bucket=bucket\_name,

                    Key=file\_key,

                    Body=data\_json,

                    ContentType='application/json'

                )

            except Exception as e:

                print("Error",e)

        return 'Processed all records'

    else:

        print("No records found in the event")

        return 'No records to process'