Detecting Fraudulent Transactions in Real Time

Manjot Kaur Dherdi | manjotkaur390@gmail.com

1. Introduction

This Fraud detection solution is designed to flag the fraudulent transactions from a stream of incoming transactions. The solution for it is logically divided into two pipeline flows:

- Real-Time Pipeline: ingests the real–time transaction data, processes it, performs inference using a deployed model and stores the results in appropriate storage solutions. It's described in Pipeline Design section of this document.
- Machine Learning Pipeline: Trains, deploys and monitors the ML model. It involves ingesting historical data from storage solutions, performing feature engineering, and feeding it into ML pipeline. It also includes model monitoring and automated retraining. This part is covered in Machine Learning Workflow section.

Platform Engineering part focuses on AWS platform design to implement both pipelines.

2. Real-Time Pipeline Design

This part of solution focuses on ingesting, processing, inferencing and storing transaction data. It's implemented in following phases:

Data Ingestion

AWS Kinesis is used for ingesting real time transactions. AWS Lambda function is triggered for every new record (which is in json format) in the Kinesis stream which extracts transaction after decoding data and pass it to processing layer.

Alternatives considered / Tradeoff: AWS Kafka:

- Chose Kinesis over Kafka because it integrates with native AWS applications better and needs less operational overhead (fully managed), and offers pay-per-use pricing with auto scaling.

Data Processing

AWS lambda function transforms raw data into a suitable format for detection. Lambda does rule based transformations on data and also retrieves relevant features from AWS SageMaker Feature Store to enrich the data. Lambda also sends derived features to the feature store for future use.

Following validation and processing are done on the raw data:

- Validations: Handle null values, type mismatches, etc. E.g. provide default values for null fields.
- Rule Based Feature Extraction techniques: To extract relevant information from data. *high_value* for transaction amount exceeding threshold, *is_vpn* for flagging vpn as origin, *location_risk* for flagging transactions from high risk locations etc.

- Feature Enrichment Techniques: To add more information/ context for some data. This is done by utilizing historical features stored in Sagemaker Feature Store. For eg.: *time_since_last* for time of previous transaction for this user, *transaction_freq* to count transactions made from this user in decided unit of time, *location_changes* for flagging location unusual to that user.

Data Inference

Lambda invokes the SageMaker endpoint for fraud detection model deployed and uses it to predict if the transaction is fraud or not. The predicted result is returned to lambda for further processing.

Data Storage

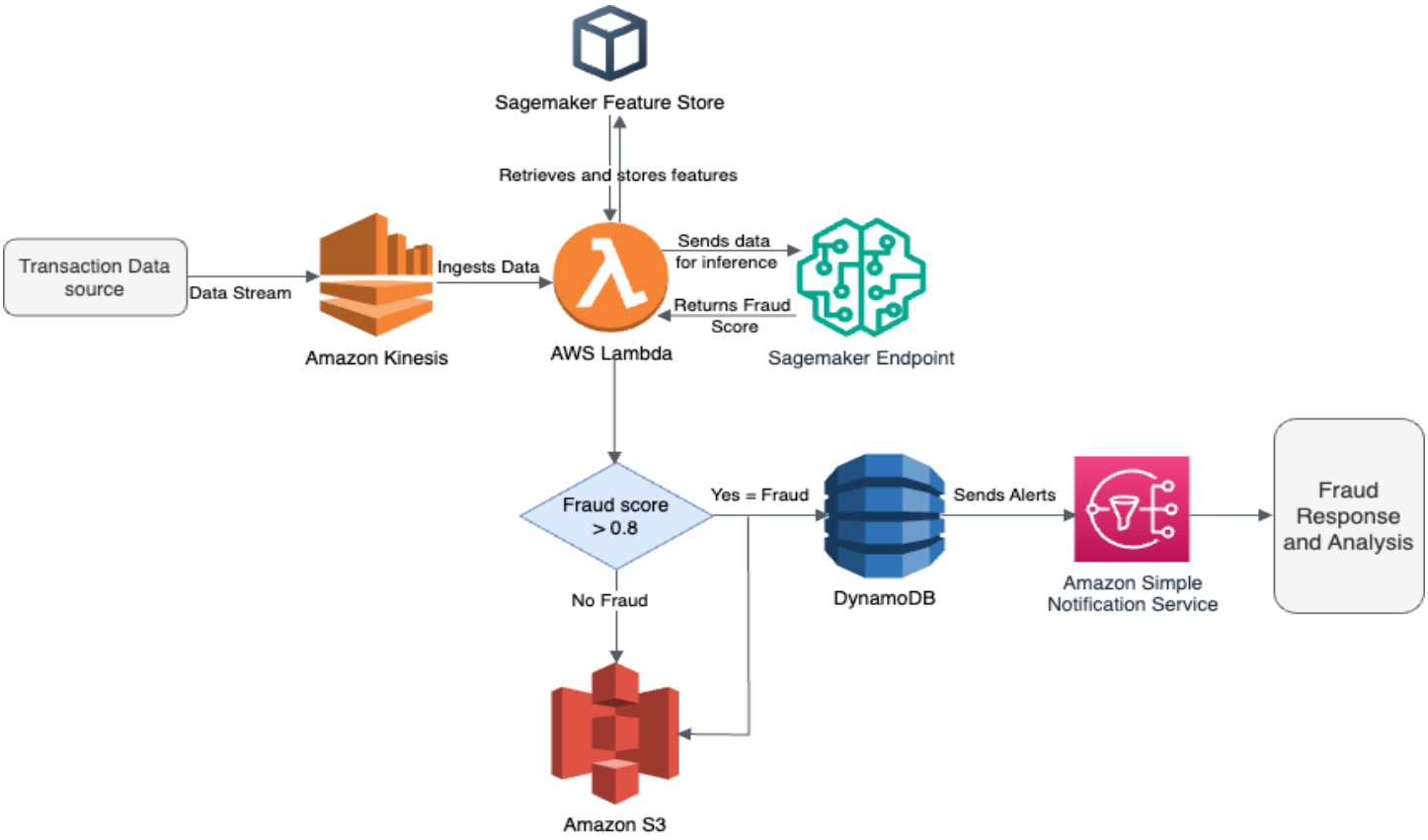
Raw Transactions: The raw transaction data along with fraud prediction is stored in the Amazon S3 bucket for model training.

Fraudulent Transactions: The fraudulent transactions are stored in DynamoDB which in turn triggers response actions using Amazon SNS (notification service) which may include manually reviewing the transaction or blocking the user.

Edge Cases:

- The fraud prediction in s3 data is not verified. So, plan for this is to use glue job during ML Pipeline to verify the fraud staus by cross-referencing S3 data with Dynamodb.

Architecture Diagram:



3. Platform Engineering

Infrastructure as Code (IaC)

This part focuses on the implementation tool for deploying the AWS platform for the pipelines. The IaC tool of choice for this solution is AWS CDK and stacks are defined in Python language. I have chosen this tool because of familiarity and because it's AWS-native solution, it is easy to integrate it. It allows to define infrastructure using familiar programming languages.

- In this solution I have implemented Real Time Pipeline using AWS CDK in which I have deployed following AWS Resources:
    - Amazon Kinesis
    - Amazon DynamoDB
    - AWS Lambda Function
    - Amazon S3 bucket
- Stacks are implemented in separate files to enhance modularity and reusability.
- The implementation is at location and contains stacks for these components:
    - Location- GitHub Repo
    - Stacks for All the above resources.
    - Main app.py file to deploy all the resources.
    - Sample Test stack to verify deployment.

This pipeline is working completely and I have shared the video demo for the same.

Video Link for Demo

CICD Pipeline Generation -

- The implementation of the real time pipeline covers the basics of this ask. I need to manually run the diff and deploy commands but my deployments update any changes to the existing stack or delete the resources if its implementation is removed from the code.
- What more needs to be done to create CICD pipeline using Github Actions.
    - Define Github Actions Workflow by creating a YAML file for cdk deployments.
    - Configure github secrets for AWS credentials like access key, region etc.
    - This file handles CICD part when new code is pushed to the main branch. It installs dependencies and deploys AWS resources without manual intervention.

4. Machine Learning Workflow

This part focuses on training and deploying machine learning model on historical transaction data stored in AWS S3 bucket. The workflow includes data processing, model training, deployment and monitoring.

Data Preprocessing

The raw transaction data stored in S3 bucket is cleaned and prepared for model training. This phase involves AWS Glue for ETL and SageMaker Notebooks for advanced feature engineering.

In Glue Job:

- Handles data inconsistencies, feature extraction and feature enrichment (same as in real-time pipeline).
- Verify the isfraud column of the S3 data by cross-referencing with Amazon DynamoDB. Ensures accuracy of labels for training data.

In SageMaker Notebook:

- Use Label Encoding on categorical values. E.g. location, device_type, card_type).
- Scaling of numerical values like amount using Min-Max Scaler to normalize the feature.

Model Training

Model training is done in SageMaker notebook. Used XGBoost for training fraud detection model.

- Split the transformed data into training (80%) and test (20%) sets. Used stratify parameter to maintain balance of fraud and non-fraud transactions in both sets.
- Use SageMaker Hyperparameter Tuning to optimize the model's hyperparameter like eta, max_depth etc.
- Model is evaluated on accuracy, precision, recall, F1 score, roc_auc_score etc.

Alternates considered/Tradeoffs:

- RandomForestClassifier was also considered but due to synthetic data it was overfitting. XGBoost works well with imbalanced datasets and high performance.

Model Deployment

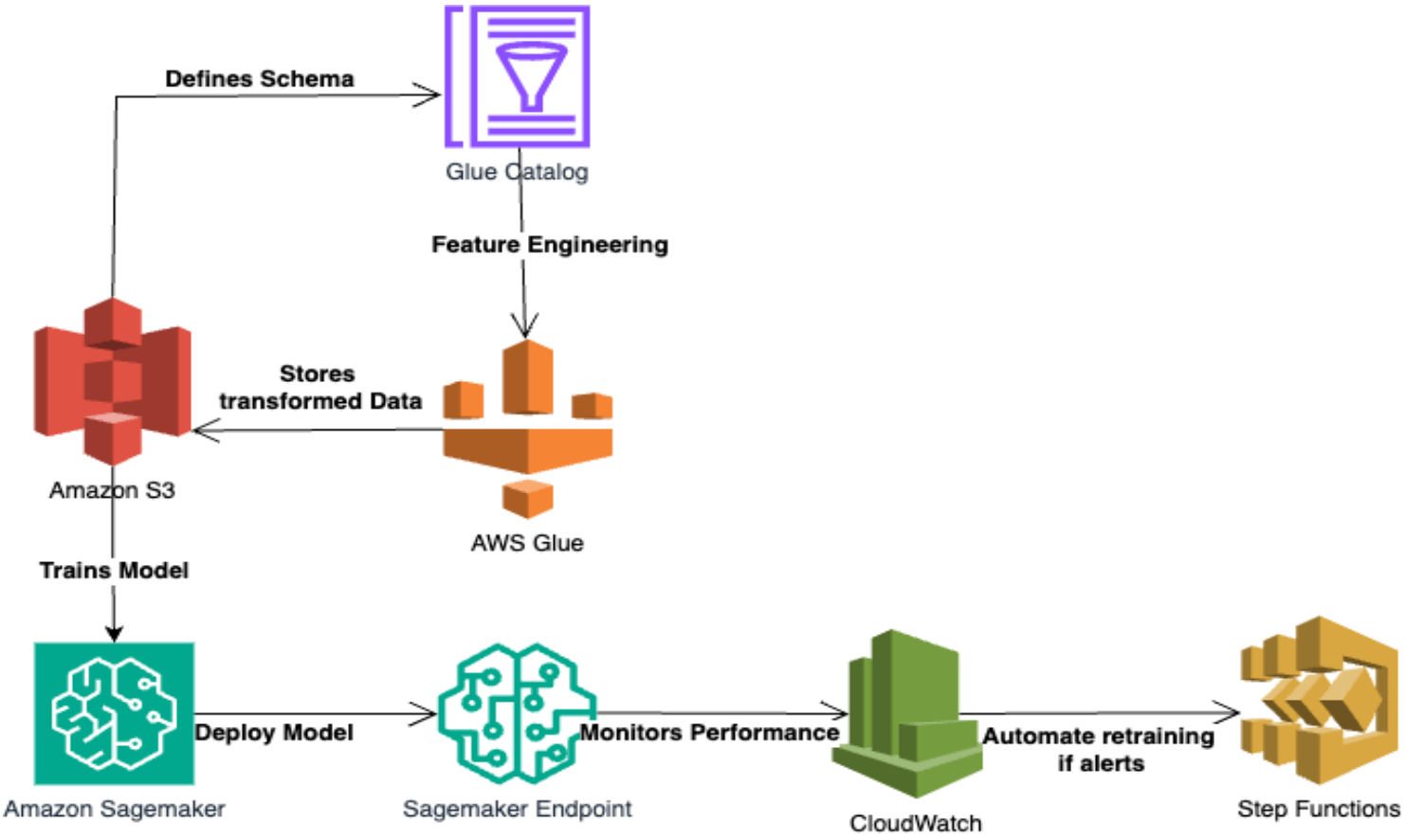
The trained model is deployed as a SageMaker Endpoint which is invoked by lambda to predict real time transactions. The model returns a fraud score to the lambda.

New versions are registered in the SageMaker Model Registry, ensuring versioning and possibility of rollback in case newer model degrades.

Model Monitoring and Retraining

Model metrics are tracked in Cloud Watch for evaluation metrics and alerts are set for notifying significant drift in model performance. If model performance drops below a set threshold, automated retraining is triggered using AWS Step Functions.

Architecture



5. Implementation and Demo

Real Time Pipeline Implementation

This pipeline is working completely and I have deployed AWS Kinesis, DynamoDB, S3 and Lambda using it.

Link for Video of the Demo

The implementation is at location and contains stacks for these components:

- Location- GitHub Repo
- Stacks for All the above resources.
- Main app.py file to deploy all the resources.
- Sample Test stack to verify deployment.

This repo also contains python script to generate 10 mock events and send streaming data to kinesis stream. Python Streaming Script

Steps to Deploy

- Prerequisites – Install node.js, awscli, cdk, python. Configure AWS Credentials
- Clone the githib repo.

- Go to the location of the project.
- Pip install requirements.txt --> install any needed libraries
- cdk synth -> To synthesize cdk stacks to cloud formation templates.
- Pytest tests/test_kinesis_stack --> To conduct unit test for stack deployment
- cdk diff -> To see what resources are being added or deleted
- cdk deploy –all -> it will deploy all the reosurces.
- python tests/test_kinesis_stream.py -->To check the pipeline, run the python script that mocks the streaming data being sent to Kinesis Stream. For this

Machine Learning Workflow Implementation

For this section I have created implementation to generate mock data and developed a ML model in jupyter notebook. I have deployed running model and saved it and stored in S3 bucket.

What's Done - I have inserted the links to the same.

- Synthetic Data Script - Python script to generate synthetic data and stored it in S3 bucket to mimic historical data. (Not shown the method where data is generated)

```
def generate_and_upload_data(num_records=1000):

transactions = [generate_transaction(0) for _ in range(1000)] + [generate_transaction(1) for _ in range(1000)]
s3 = boto3.client("s3")
s3.put_object(
   Bucket=S3_BUCKET,
   Key=S3_KEY,
   Body=json.dumps(transactions),
   ContentType="application/json"
)
print(f"Uploaded {num_records} synthetic transactions to s3://{S3_BUCKET}/{S3_KEY}")
```

- Model Training Notebook - Created jupyter notebook for model training and evaluation.

```
xgb_model = XGBClassifier(
   n_estimators=100,
   max_depth=5,
   learning_rate=0.1,
   subsample=0.8,
   colsample_bytree=0.8,
   random_state=42
)
xgb_model.fit(X_train, y_train)

# Make Predictions
y_pred = xgb_model.predict(X_test)
```

```
# Evaluate Model
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.4f}")
print("Classification Report:\n", report)

#Cross-validation
scores = cross_val_score(model, X, y, cv=5, scoring='accuracy')
print("Cross-Validation Accuracy Scores:", scores)
print("Mean Accuracy:", scores.mean())
```

- Steps to run it -
    - Clone the github repo.
    - Run the python script to generate data and send to S3.
    - Open fraud_detection_model.ipynb in jupyter notebook and run it.

What's Not Done -

- Did not create AWS platform for this pipeline because of time crunch. My knowledge to deploy AWS platform for a pipeline is depicted in the Real-Time Pipeline implementation. I am adept at deploying the platform for this pipeline.

6. Future Enhancements

Platform Engineering

- I will use **ssm parameter store** to export the values or arns of the aws resources created. This eases automation as it gives access to the resources even when not deployed simultaneously.
- I will add **KMS keys** to encrypt S3 buckets, DynamoDB and other valid resources as it gives us capabilities to manages access and encryption.
- I will add **IAM roles** for execution of glue and sagemaker to abide by least access privilege principles insted of providing complete resource accesses.

Real Time Ingestion

- In more elaborate version of this project, I will consider the edge case where in the beginning we don't have any historical data or model. I will use rule-based predictions or patterns to detetct anomalies on the fly in lambda. And when I have enough historical data in S3, I will train and deploy model and modify my lambda to use sagemaker endpoint for inference.
- I will add Response Actions Automation for when a fraudulent transaction lands in DynamoDB. I will use SNS to send mail to customers to verify the transaction's fraud status and update it in DynamoDB.