Real-Time Fraud Detection in Online Transactions using AWS Fraud Detector

Team Members: 1) Vanguri Charan -23P31A0561

2) Ganti Karthik - 23P31A0516

3) Penugonda Sandeep -23P31A0545

4) Nagulapalli Manikantaraj -23P31A0581

5) Gatti Greshma -23P31A04E4

6) Villuri Mahi -23P31A0467

1. Abstract

This project implements a real-time fraud detection system for online financial transactions using Amazon Web Services (AWS). The system uses Amazon Fraud Detector to analyze transactions, detect suspicious patterns, and trigger alerts in real time. Fraudulent transactions are flagged based on machine learning models and predefined rules.

2. Introduction

Online transaction fraud is a major issue in banking and e-commerce. Detecting fraud in real time prevents losses and builds user trust. This project leverages AWS Fraud Detector and other AWS services to create a serverless, scalable fraud detection pipeline.

3. Architecture

The architecture integrates multiple AWS services to process, analyze, and respond to potential fraud in real time. Transactions are streamed through Amazon Kinesis, processed by AWS Lambda, evaluated by Amazon Fraud Detector, and results are stored and alerted via DynamoDB and SNS.

[Insert Architecture Diagram Here]

4. AWS Services Used

• Amazon Fraud Detector – Detects fraud using ML.

• Amazon Kinesis – Streams incoming transactions.

• AWS Lambda – Executes fraud detection logic.

• Amazon SNS – Sends SMS/email alerts.

• Amazon DynamoDB – Stores suspicious transaction records.

• Amazon S3 – Stores dataset and logs.

• Amazon CloudWatch – Monitors the system.

5. Implementation Steps

• Created historical transaction dataset (CSV) & uploaded to S3.

• Configured Fraud Detector: Created entity type, event type, and variables. Trained model using CSV data. Built detector with rules.

• Set up Kinesis Stream for real-time transactions.

• Developed Lambda function to process transactions and call Fraud Detector.

• Connected SNS, DynamoDB, and CloudWatch for alerts, storage, and monitoring.

6. Dataset

• Total Transactions: 200
• Fraudulent: 45
• Legit: 155

Fields: transaction_id, user_id, transaction_amount, country, ip_address, device_id, payment_method, timestamp, is_fraud

7. Rules & Outcomes

• Model Score > 700 → FRAUD

• Model Score 500–700 → REVIEW

• Model Score < 500 → APPROVE

8. Testing & Results

• Fraudulent transactions were flagged correctly.
• Alerts sent via SNS.
• Suspicious transactions stored in DynamoDB.
• Logs available in CloudWatch.

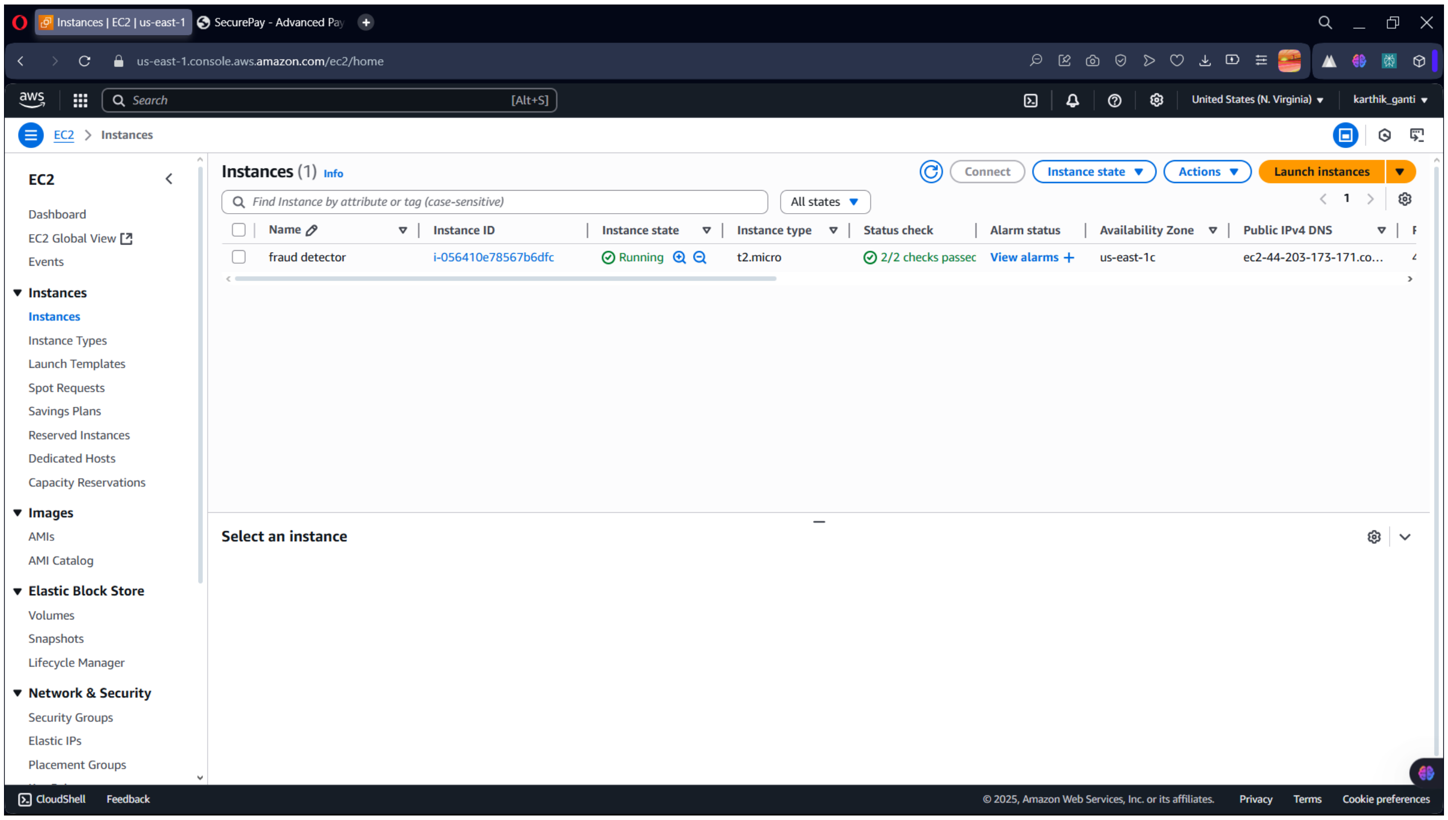
9. Benefits

• Real-time fraud detection.
• No server management (serverless).
• Scalable for high transaction volume.

10. Limitations

• AWS Fraud Detector available in limited regions.
• Model accuracy depends on quality of training data.

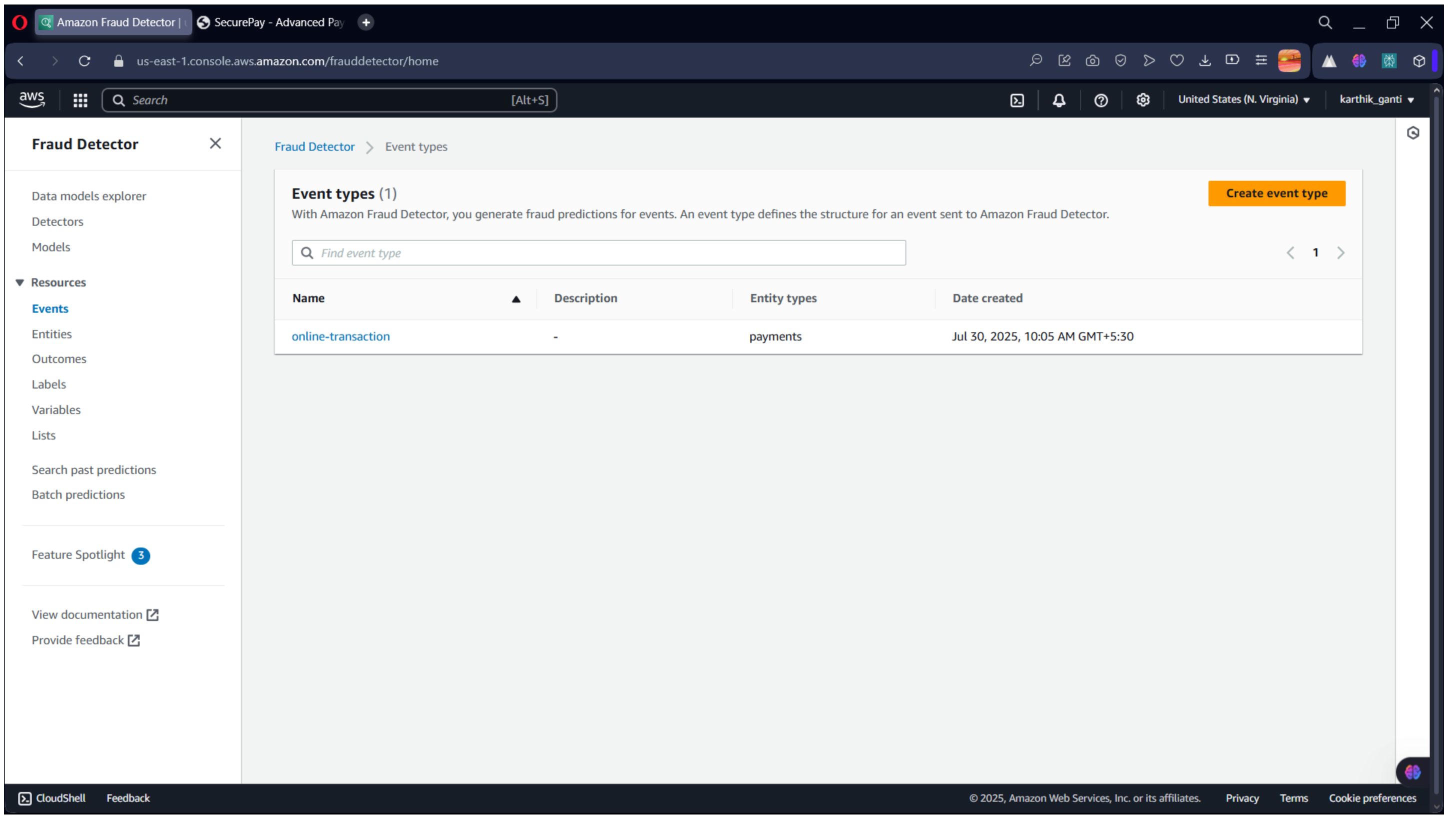
11.STEP TO STEP GUIDES



Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS
fraud detector	i-056410e78567b6dfc	Running	t2.micro	2/2 checks passec	View alarms +	us-east-1c	ec2-44-203-173-171.co...

Screenshot 1: AWS EC2 Instances Page

- This page shows the *AWS EC2 Console* for the region US East (N. Virginia).
- You currently have one EC2 instance running, named "fraud detector", with the instance ID "i-056410e78567b6dfc".
- The instance is of type *t2.micro*, is in the *running* state, has passed both status checks, and is in the *us-east-1c* availability zone.
- The public IPv4 DNS is also visible, allowing you to access the instance remotely.
- This setup is typical for hosting backend services, running applications, or supporting cloud-based solutions like your fraud detection service.



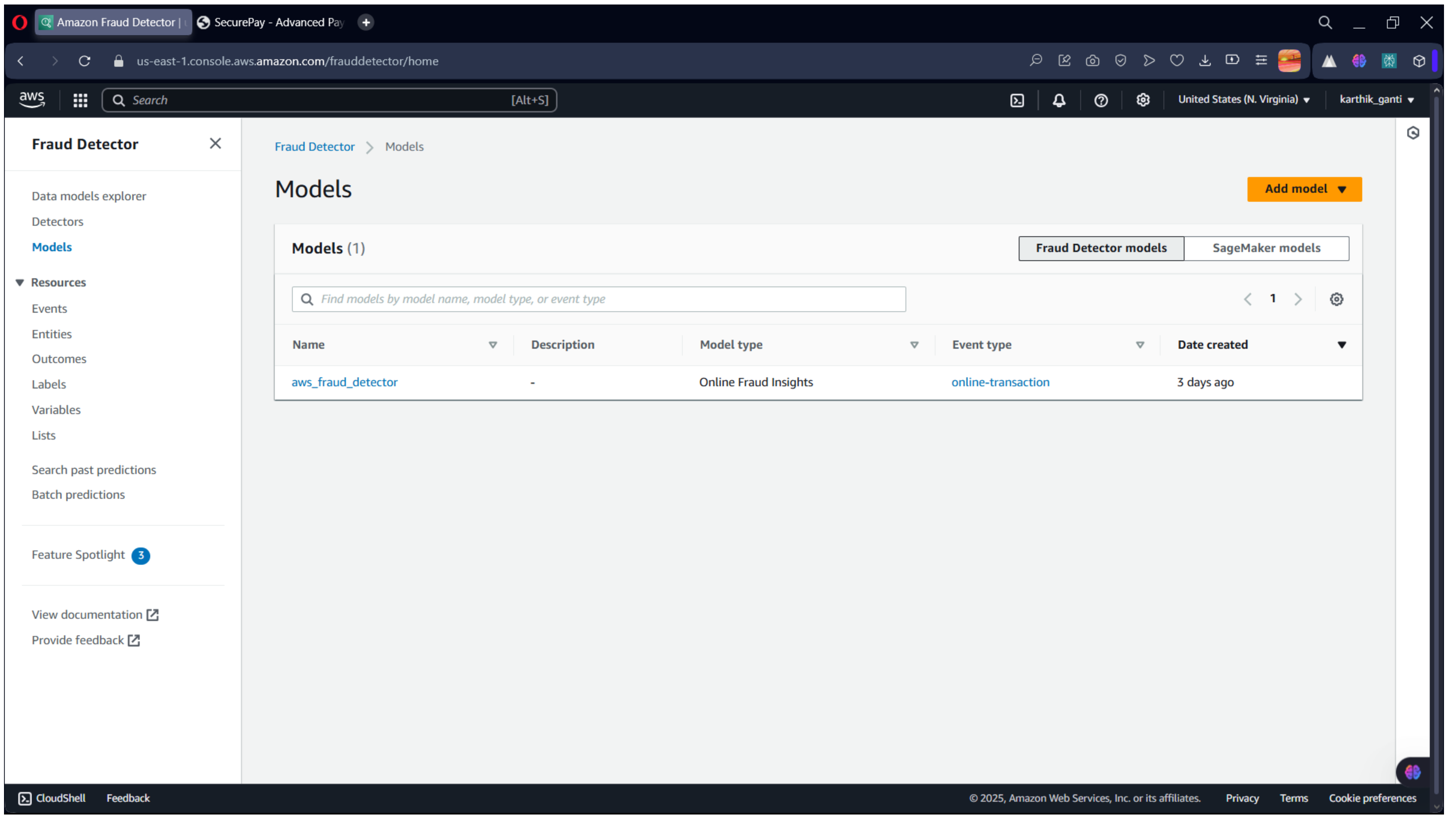
Event types (1)

With Amazon Fraud Detector, you generate fraud predictions for events. An event type defines the structure for an event sent to Amazon Fraud Detector.

Name	Description	Entity types	Date created
online-transaction	-	payments	Jul 30, 2025, 10:05 AM GMT+5:30

Screenshot 2: AWS Fraud Detector – Event Types

- This page is from the *AWS Fraud Detector* service under the "Event types" section.
- There is one event type configured: "online-transaction".
- The event type is associated with "payments" as its entity type and was created on July 30, 2025.
- *Event types* are used by AWS Fraud Detector to define the structure of input events (in this case, online payment transactions) for which you want to predict fraud.

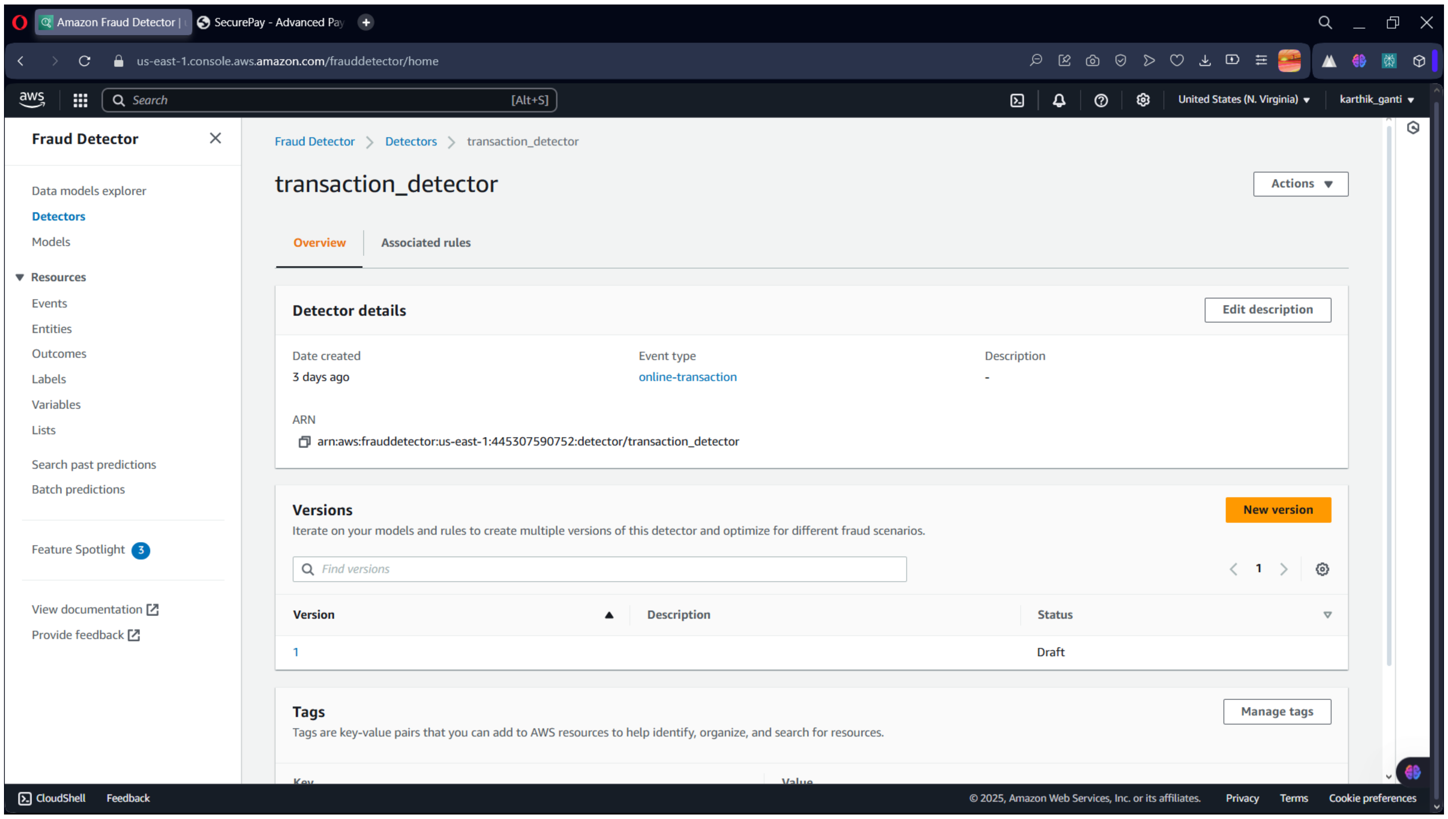


Models (1)

Name	Description	Model type	Event type	Date created
aws_fraud_detector	-	Online Fraud Insights	online-transaction	3 days ago

Screenshot 3: AWS Fraud Detector – Models

- This screen is under the "Models" tab of AWS Fraud Detector.
- There is one model listed: "aws_fraud_detector".
- The model type is "Online Fraud Insights", and it's associated with the "online-transaction" event type.
- This model was created three days ago and is used to generate fraud prediction scores for new online transaction events.
- The model is likely trained on historical transaction data, learning to distinguish between normal and fraudulent activities.



transaction_detector

Detector details

Date created	Event type	Description
3 days ago	online-transaction	-

ARN
arn:aws:frauddetector:us-east-1:445307590752:detector/transaction_detector

Versions
Iterate on your models and rules to create multiple versions of this detector and optimize for different fraud scenarios.

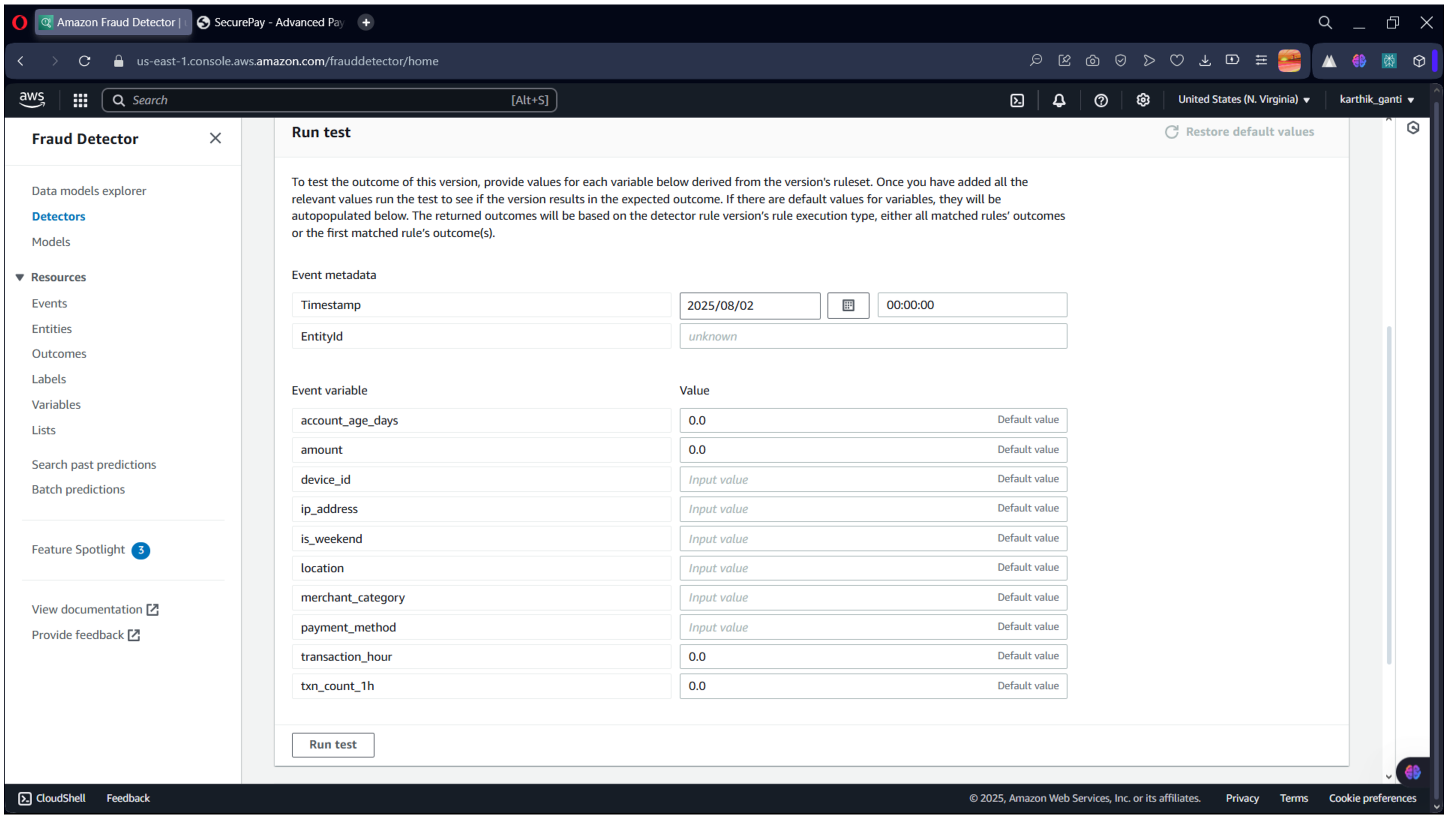
Version	Description	Status
1		Draft

Tags
Tags are key-value pairs that you can add to AWS resources to help identify, organize, and search for resources.

Key Value

Screenshot 4: AWS Fraud Detector – Detector Details

- This shows the detail page for a detector named "transaction_detector" in AWS Fraud Detector.
- The detector is linked to the "online-transaction" event type and was created three days ago.
- Each detector can have multiple versions and rules for evaluating event data and triggering actions based on model predictions.
- The ARN is displayed, uniquely identifying this detector resource.
- As shown, the current version is still in *draft* status, meaning it might still be under configuration or testing.
- Detectors are responsible for applying fraud detection models and evaluating rules when transaction events are received.



To test the outcome of this version, provide values for each variable below derived from the version's ruleset. Once you have added all the relevant values run the test to see if the version results in the expected outcome. If there are default values for variables, they will be autopopulated below. The returned outcomes will be based on the detector rule version's rule execution type, either all matched rules' outcomes or the first matched rule's outcome(s).

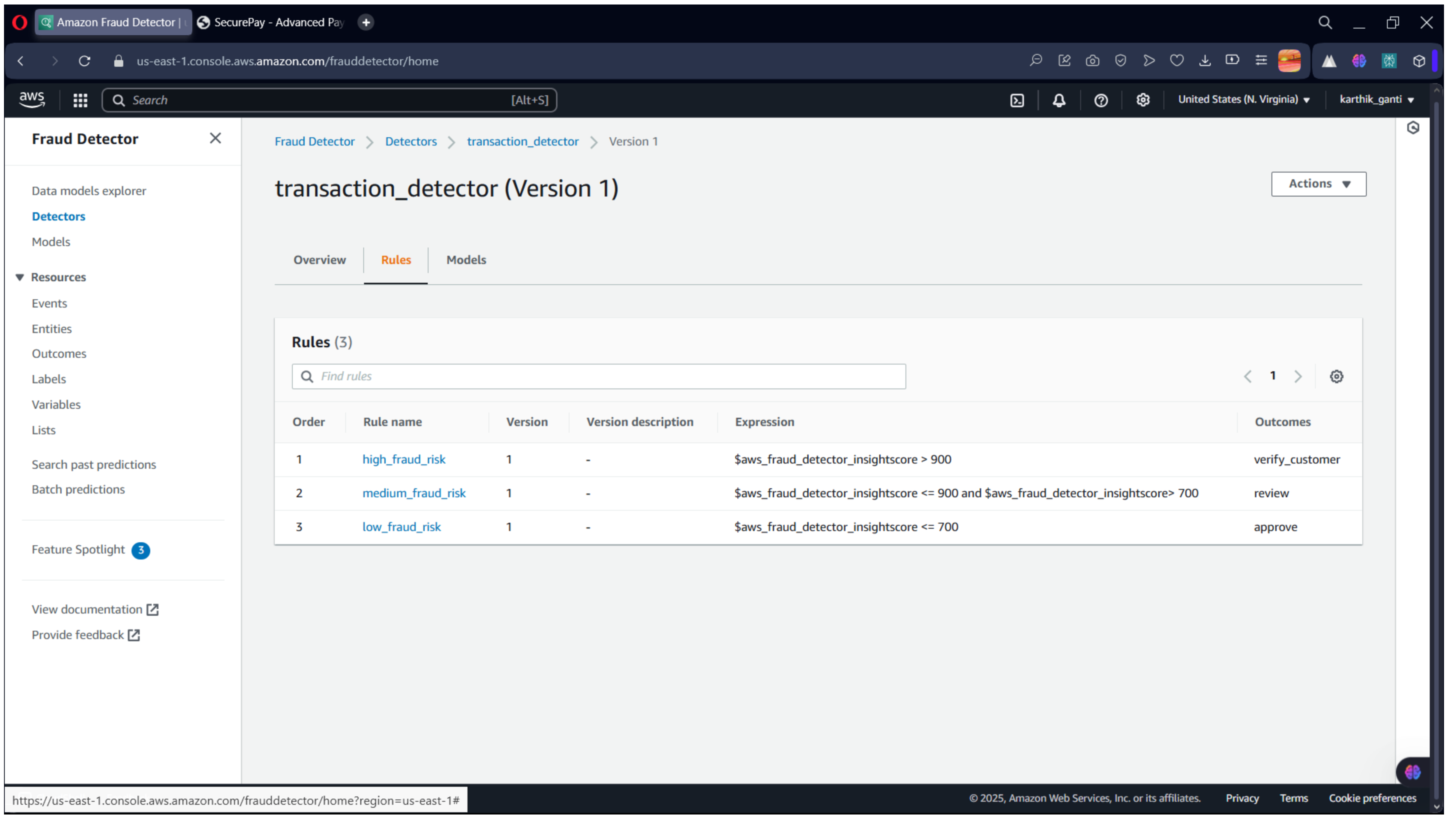
Event metadata

Timestamp	2025/08/02 00:00:00
EntityId	unknown

Event variable	Value	
account_age_days	0.0	Default value
amount	0.0	Default value
device_id	Input value	Default value
ip_address	Input value	Default value
is_weekend	Input value	Default value
location	Input value	Default value
merchant_category	Input value	Default value
payment_method	Input value	Default value
transaction_hour	0.0	Default value
txn_count_1h	0.0	Default value

Screenshot 5: AWS Fraud Detector – Run Test Page

- This page allows you to **test the output of a specific detector version** in AWS Fraud Detector by manually inputting sample data for each defined variable.
- You can set event metadata fields like *Timestamp* and *EntityId*, as well as numerous event variables such as:
  - account_age_days, amount, device_id, ip_address, is_weekend, location, merchant_category, payment_method, transaction_hour, and txn_count_1h.
- After entering values (or using defaults), the "Run test" button lets you check the model/rules outcomes, helping you verify if the detector's logic is producing the expected results with your test inputs.
- This is useful for validating your fraud detection pipeline without using live transaction data.



transaction_detector (Version 1)

Rules (3)

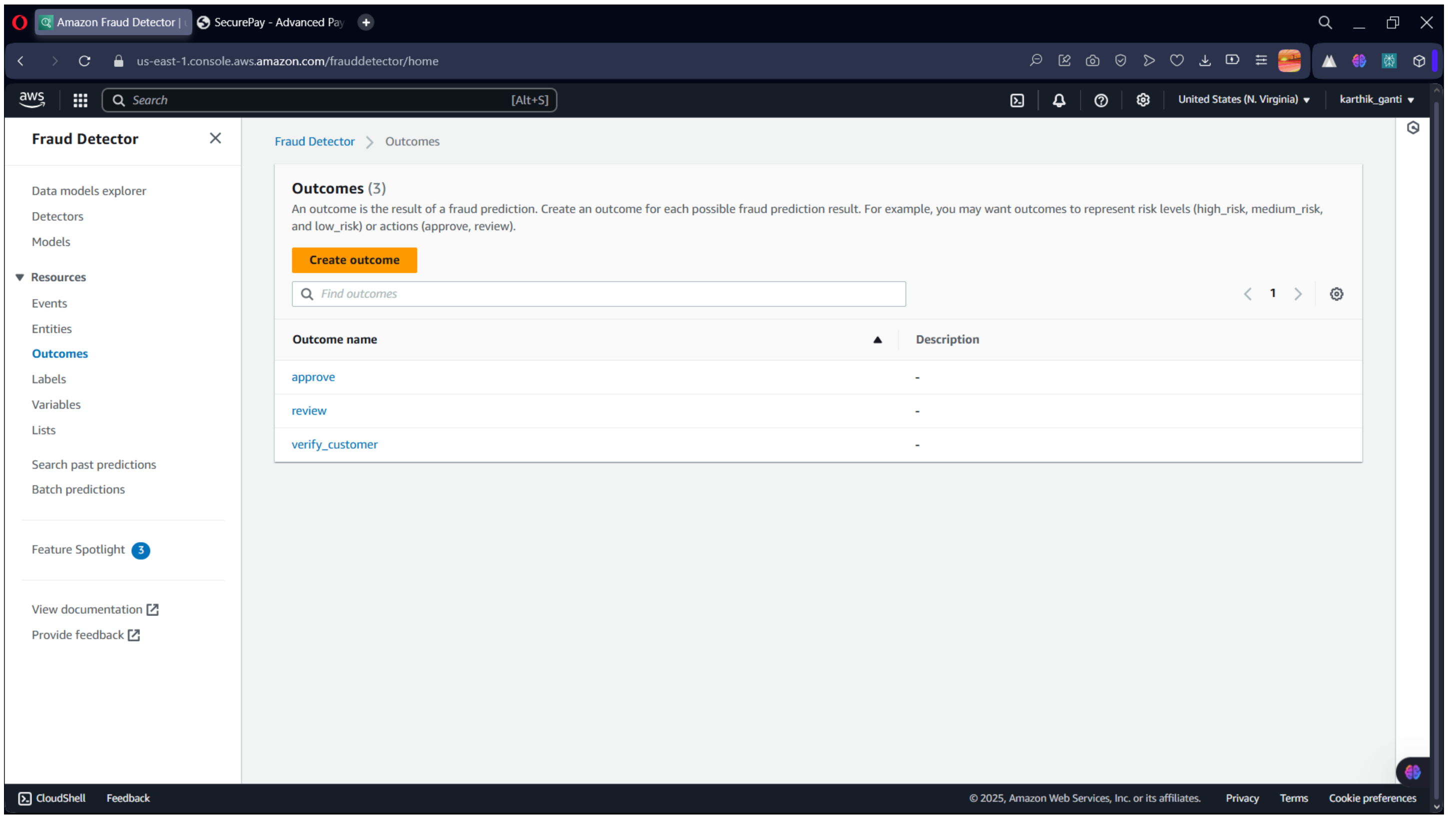
Order	Rule name	Version	Version description	Expression	Outcomes
1	high_fraud_risk	1	-	$aws_fraud_detector_insightscore > 900	verify_customer
2	medium_fraud_risk	1	-	$aws_fraud_detector_insightscore <= 900 and $aws_fraud_detector_insightscore> 700	review
3	low_fraud_risk	1	-	$aws_fraud_detector_insightscore <= 700	approve

https://us-east-1.console.aws.amazon.com/frauddetector/home?region=us-east-1#

© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Screenshot 6: AWS Fraud Detector – Detector Rules (transaction_detector Version 1)

- This is the "Rules" tab for **Version 1** of the "transaction_detector" in AWS Fraud Detector.
- Three rules are presented, each associated with a fraud risk level and an outcome:
    1. **high_fraud_risk**: if the fraud score is greater than 900, outcome → verify_customer.
    2. **medium_fraud_risk**: if the score is between 700 and 900, outcome → review.
    3. **low_fraud_risk**: if the score is 700 or less, outcome → approve.
- These rules determine what action to take depending on the fraud model's prediction score, automating the fraud decision process for each transaction analyzed.



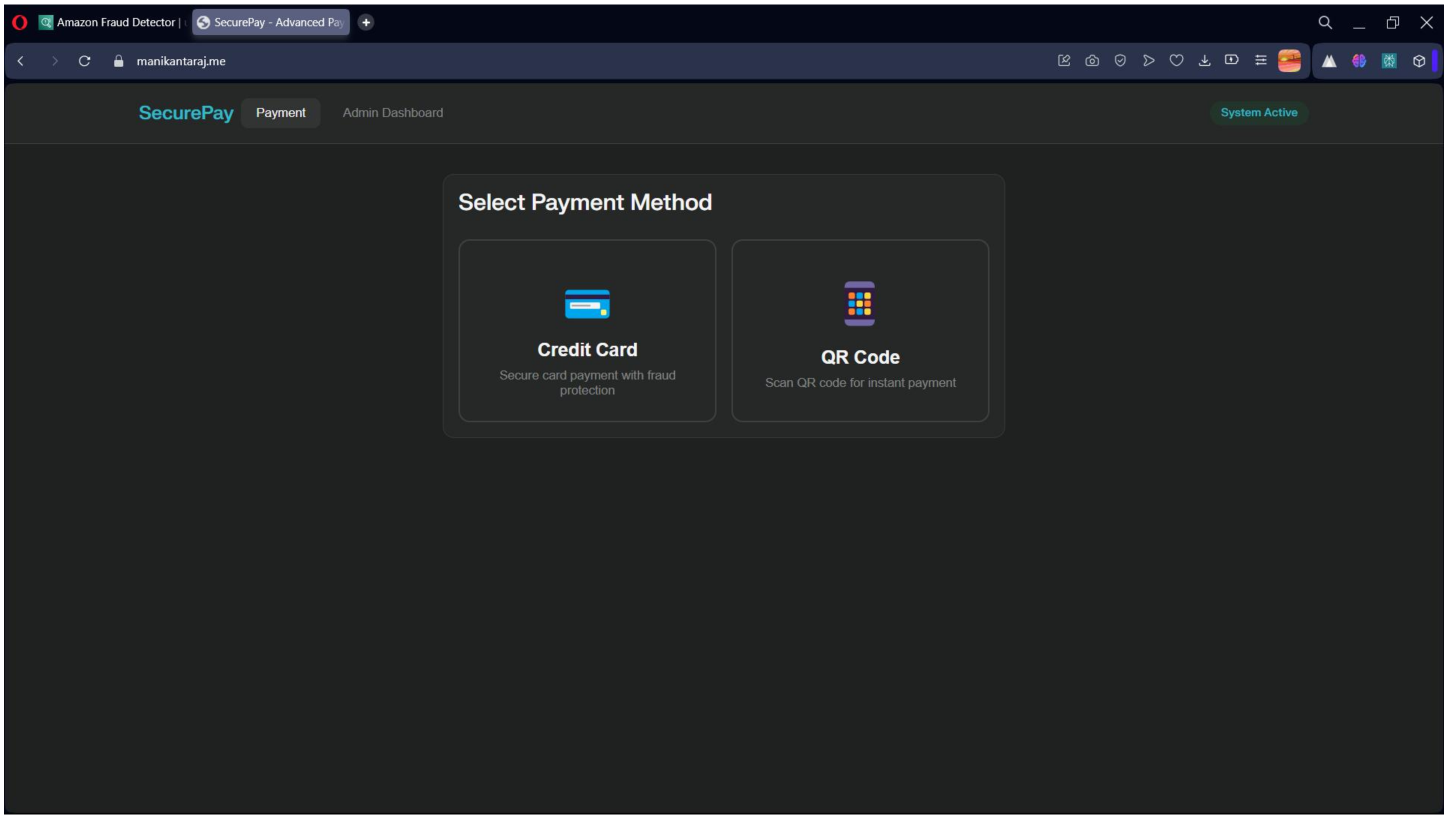
Outcomes (3)

An outcome is the result of a fraud prediction. Create an outcome for each possible fraud prediction result. For example, you may want outcomes to represent risk levels (high_risk, medium_risk, and low_risk) or actions (approve, review).

Outcome name	Description
approve	-
review	-
verify_customer	-

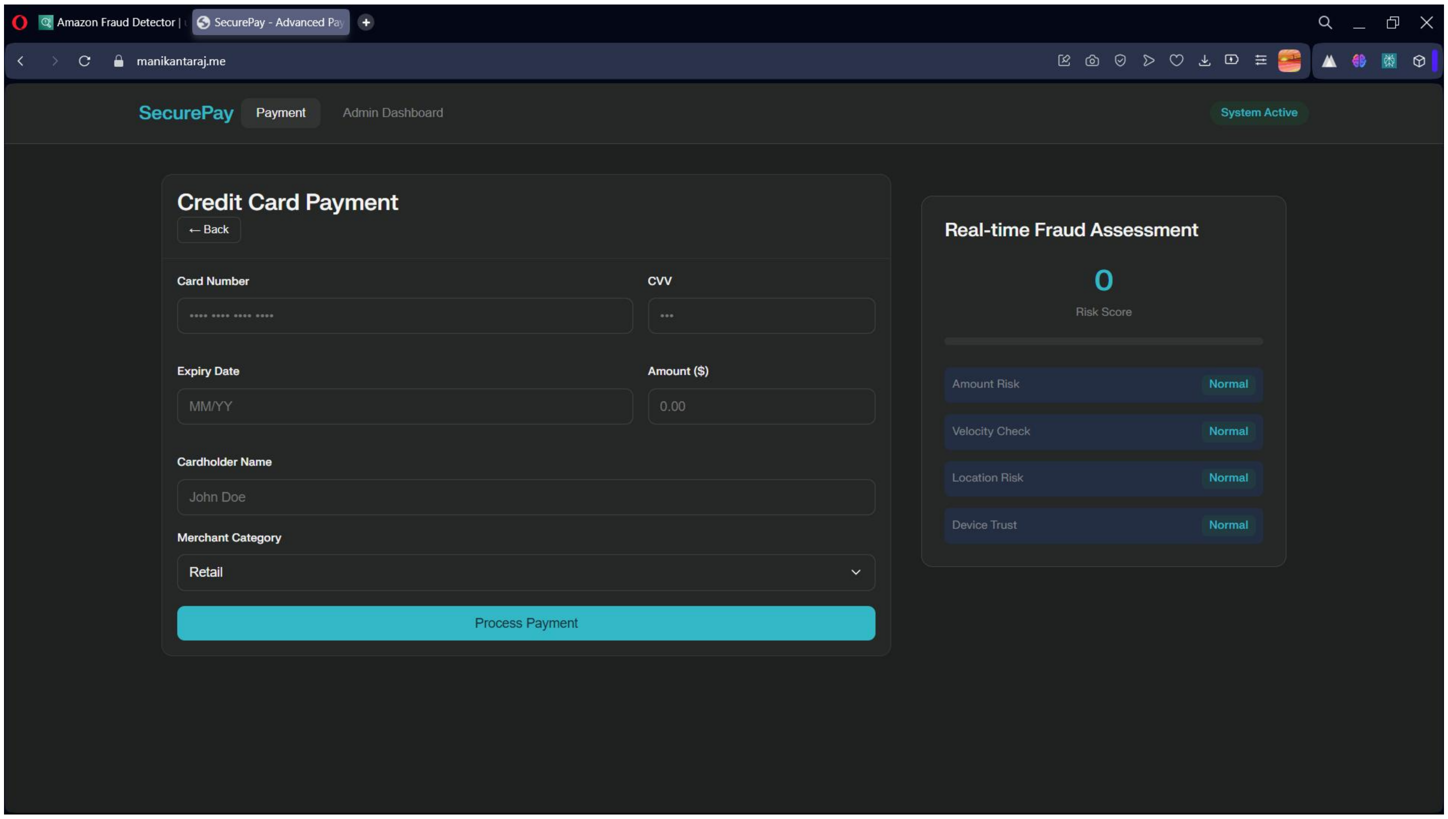
Screenshot 7: AWS Fraud Detector – Outcomes List

- In AWS Fraud Detector, an **outcome** defines the action or label resulting from a fraud prediction or rule.
- This page lists three outcomes:
    - approve
    - review
    - verify_customer
- No detailed descriptions are present, but typically:
    - "approve" means the transaction passes automatically,
    - "review" means it's flagged for manual checking,
    - "verify_customer" could prompt extra authentication or information before approval.
- These outcomes are linked to the rule evaluations and model predictions, controlling what happens after each transaction is scored.



Screenshot 8: SecurePay Payment Page – Select Payment Method

- This is a **frontend UI for a payment gateway called "SecurePay"**.
- The screen displays two payment options:
    - Credit Card (with a note: Secure card payment with fraud protection)
    - QR Code (with a note: Scan QR code for instant payment)
- The system status at the top right shows "System Active".
- This UI is likely integrated with your fraud detection backend (perhaps via AWS Fraud Detector) to provide secure, protected transaction processing.



Credit Card Payment

← Back

Card Number	•••• •••• •••• ••••
CVV	•••
Expiry Date	MM/YY
Amount ($)	0.00
Cardholder Name	John Doe
Merchant Category	Retail

Process Payment

Real-time Fraud Assessment

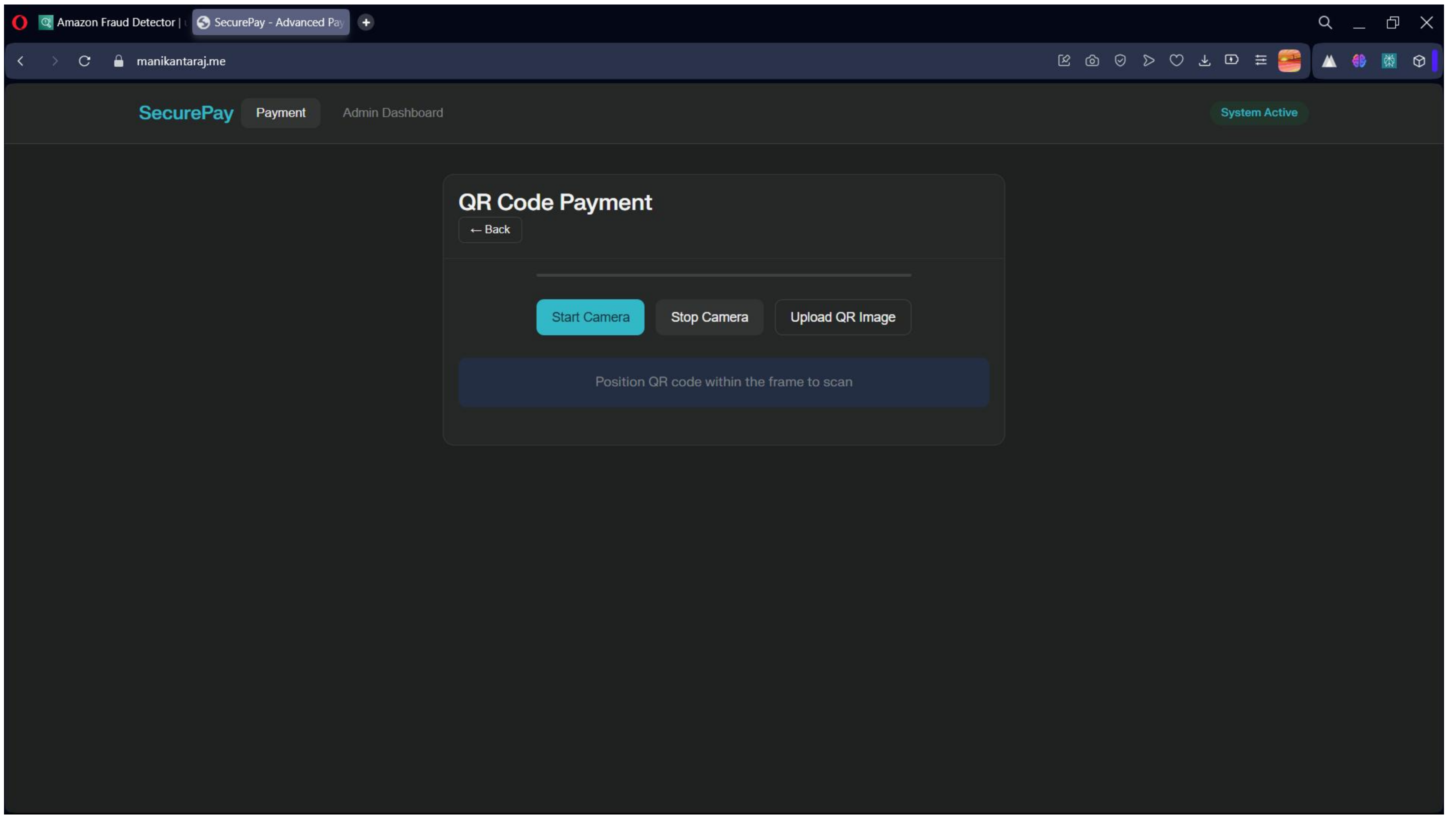
0

Risk Score

Amount Risk	Normal
Velocity Check	Normal
Location Risk	Normal
Device Trust	Normal

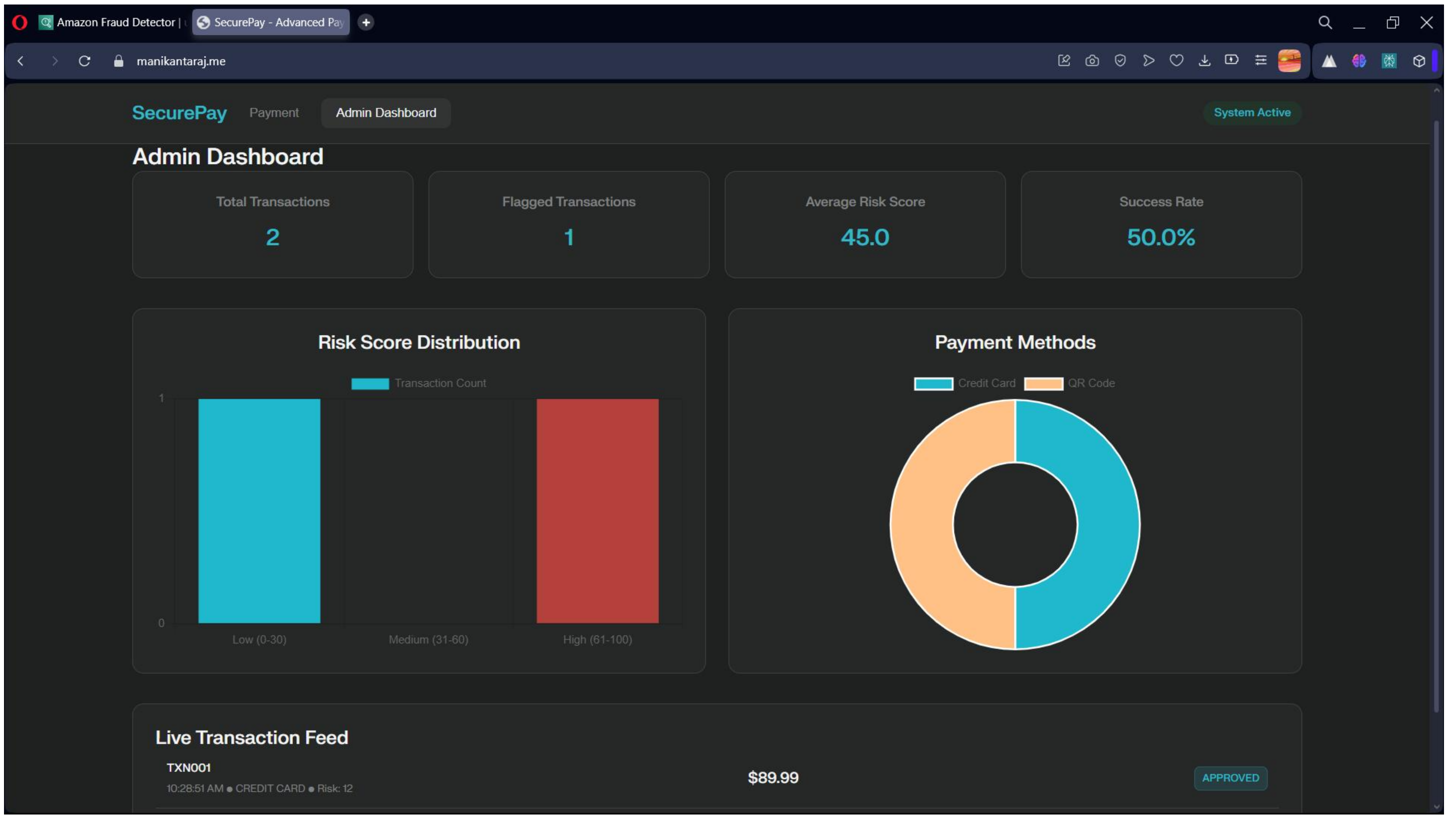
Screenshot 9: SecurePay – Credit Card Payment with Real-Time Fraud Assessment

- This screen shows the **Credit Card Payment** input form from the SecurePay payment platform.
- Users enter their card details (number, CVV, expiry date), amount, cardholder name, and merchant category to process a payment.
- To the right, there's a **Real-time Fraud Assessment** panel showing:
    - Risk Score: Displays the current risk as "0".
    - Amount Risk, Velocity Check, Location Risk, Device Trust: All displayed as "Normal".
- This interface demonstrates direct integration between payment capture and real-time fraud analysis—likely using an automated backend such as AWS Fraud Detector. The risk assessment helps inform decision-making before even completing the payment.



Screenshot 10: SecurePay – QR Code Payment

- This is the **QR Code Payment** page from the SecurePay system.
- Users can start or stop their camera, or upload a QR image to scan and pay.
- The system prompts users to position the QR code within a frame for scanning.
- This feature is common in instant/mobile payment platforms and supports rapid checkouts, especially for e-commerce and physical retail payments.



Admin Dashboard

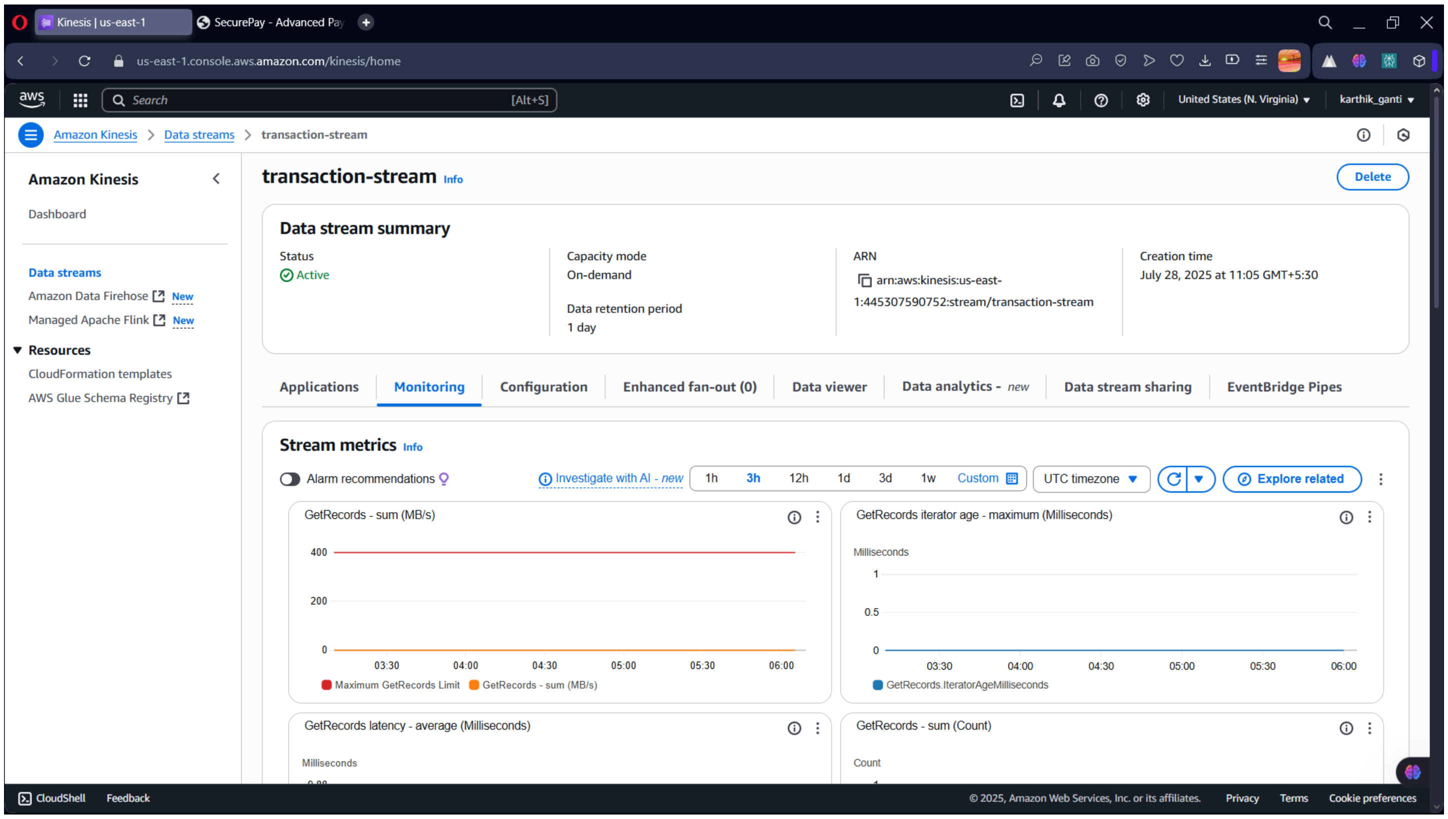
Total Transactions	Flagged Transactions	Average Risk Score	Success Rate
2	1	45.0	50.0%

Live Transaction Feed

TXN001
10:28:51 AM • CREDIT CARD • Risk: 12
$89.99
APPROVED

Screenshot 11: SecurePay – Admin Dashboard Overview

- This is the **Admin Dashboard** of SecurePay, providing operational insights at a glance.
- Top KPIs: Total Transactions (2), Flagged Transactions (1), Average Risk Score (45), Success Rate (50.0%).
- Charts:
  - Risk Score Distribution: Shows transaction counts by risk range (Low, Medium, High).
  - Payment Methods: Donut chart splitting usage between Credit Card and QR Code payments.
- Live Transaction Feed: Details individual transactions, including timestamp, payment method, risk score, and status (example: TXN001 for $89.99, Risk: 12, APPROVED).
- This dashboard is valuable for monitoring system activity, detecting fraud trends in real time, and keeping track of risk and performance for all payments processed via SecurePay.



transaction-stream Info

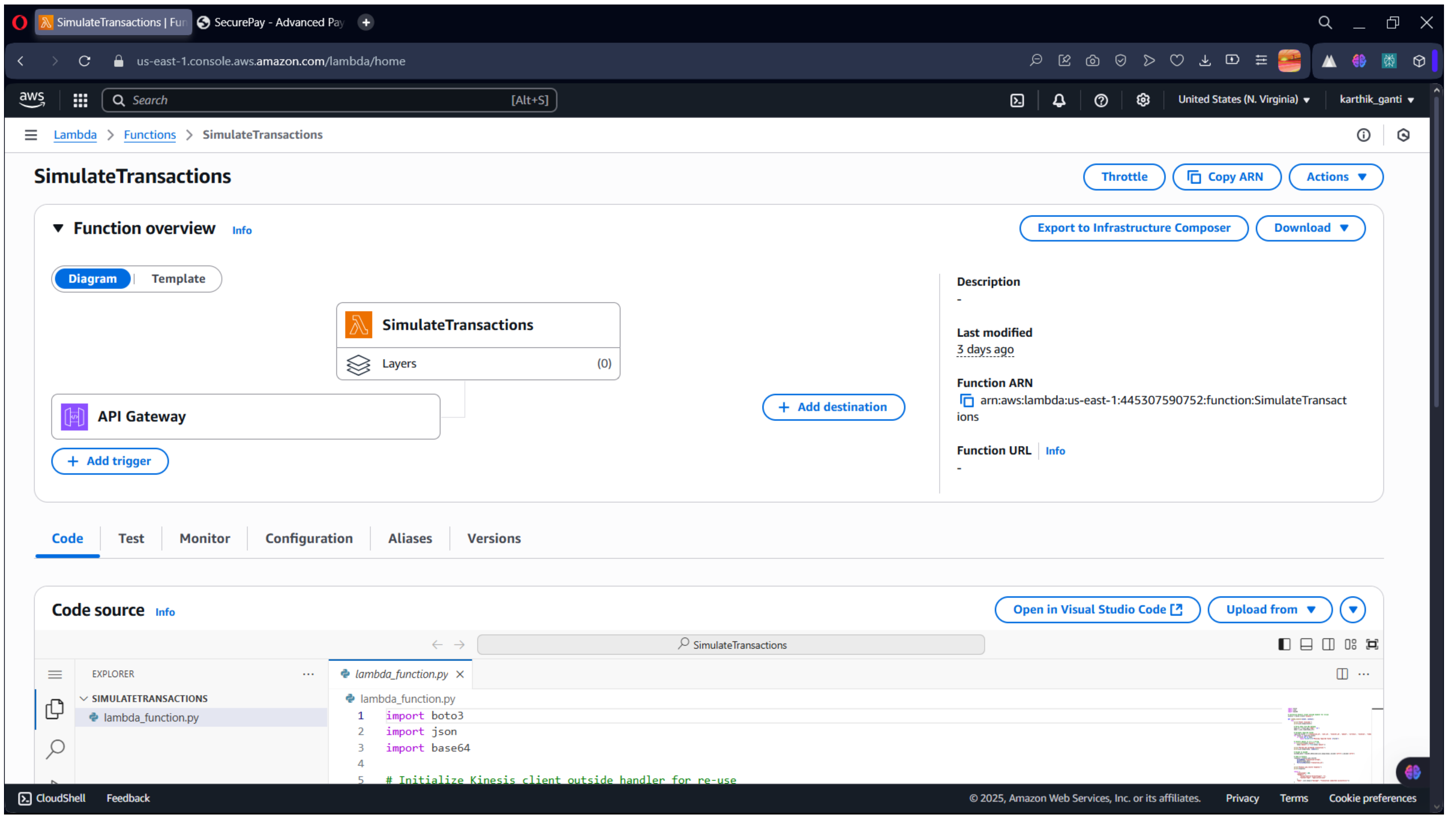
Data stream summary

Status: Active
Capacity mode: On-demand
Data retention period: 1 day
ARN: arn:aws:kinesis:us-east-1:445307590752:stream/transaction-stream
Creation time: July 28, 2025 at 11:05 GMT+5:30

Stream metrics Info

Screenshot 12: AWS Kinesis – Monitoring a Data Stream ("transaction-stream")

- This AWS Kinesis dashboard shows metrics for a stream named "transaction-stream".
- Status is "Active," and the stream uses On-demand capacity mode with 1-day retention.
- The page is on the **Monitoring** tab, presenting real-time stream metrics such as:
    - GetRecords sum (MB/s)
    - Iterator age (ms)
    - Latency
    - Count of GetRecords
- Kinesis streams like this one are frequently used to capture, process, and route transaction or event data (for example, payment events) to other AWS services—such as for fraud detection, analytics, or alerting pipelines.



SimulateTransactions

Description
-

Last modified
3 days ago

Function ARN
arn:aws:lambda:us-east-1:445307590752:function:SimulateTransactions

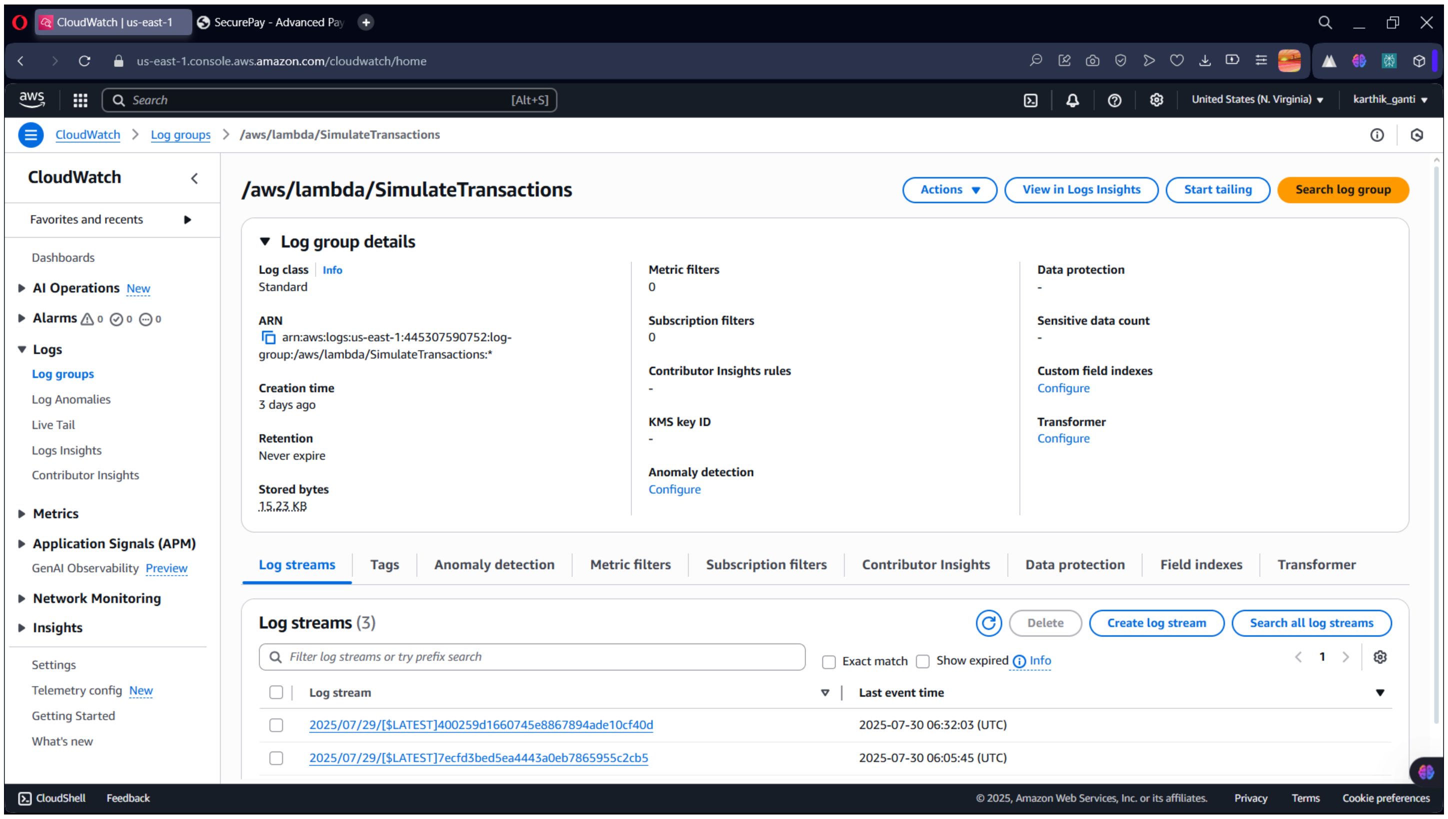
Function URL
-

```python
import boto3
import json
import base64

# Initialize Kinesis client outside handler for re-use
```

Screenshot 13: AWS Lambda – SimulateTransactions Function

- This is the AWS Lambda console for a function called "SimulateTransactions".
- The architecture diagram shows that the function is triggered by API Gateway, meaning it can be called via an HTTP API.
- You see the Code source tab with the actual Python script open (lambda_function.py).
- Metadata indicates it was last modified 3 days ago.
- This function likely generates or simulates payment/transaction data (for testing the overall fraud detection workflow, including event ingestion and risk analysis).



/aws/lambda/SimulateTransactions

Log group details

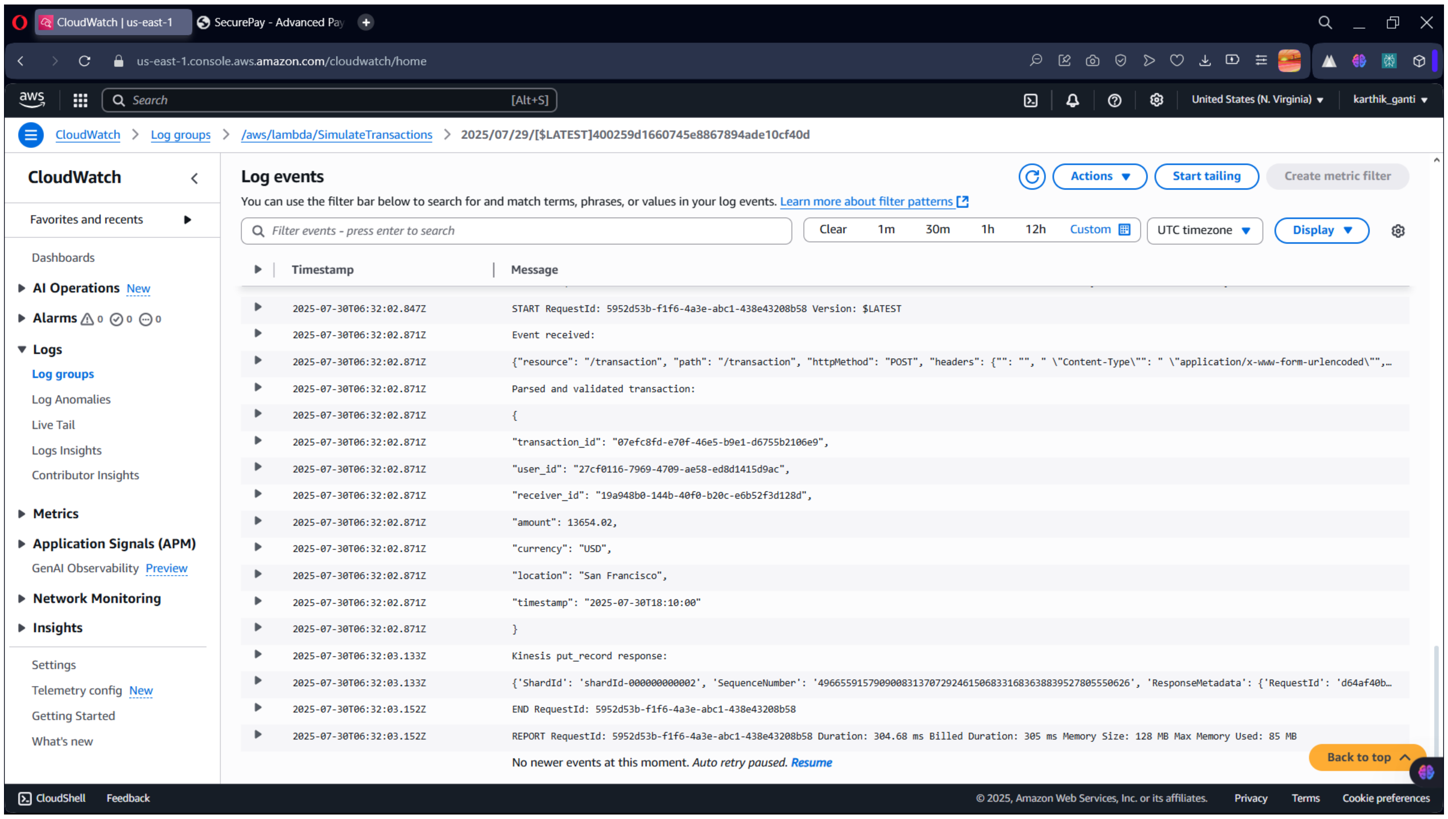
Log class Info: Standard	Metric filters: 0	Data protection: -
ARN: arn:aws:logs:us-east-1:445307590752:log-group:/aws/lambda/SimulateTransactions:*	Subscription filters: 0	Sensitive data count: -
Creation time: 3 days ago	Contributor Insights rules: -	Custom field indexes: Configure
Retention: Never expire	KMS key ID: -	Transformer: Configure
Stored bytes: 15.23 KB	Anomaly detection: Configure	

Log streams (3)

Log stream	Last event time
2025/07/29/[$LATEST]400259d1660745e8867894ade10cf40d	2025-07-30 06:32:03 (UTC)
2025/07/29/[$LATEST]7ecfd3bed5ea4443a0eb7865955c2cb5	2025-07-30 06:05:45 (UTC)

Screenshot 14: Amazon CloudWatch – Log Group Details for SimulateTransactions

- This displays the **log group details** page for logs generated by the Lambda function "/aws/lambda/SimulateTransactions".
- Information includes:
  - Log class: Standard (default retention and usage)
  - ARN, creation time, bytes stored, and that retention is set to "Never expire"
  - The "Log streams" tab lists individual session logs (each stream corresponds to a separate Lambda execution context)
- From here, admins can set up anomaly detection, metric filters, subscription filters, and more advanced monitoring rules for Lambda execution and transaction flow.



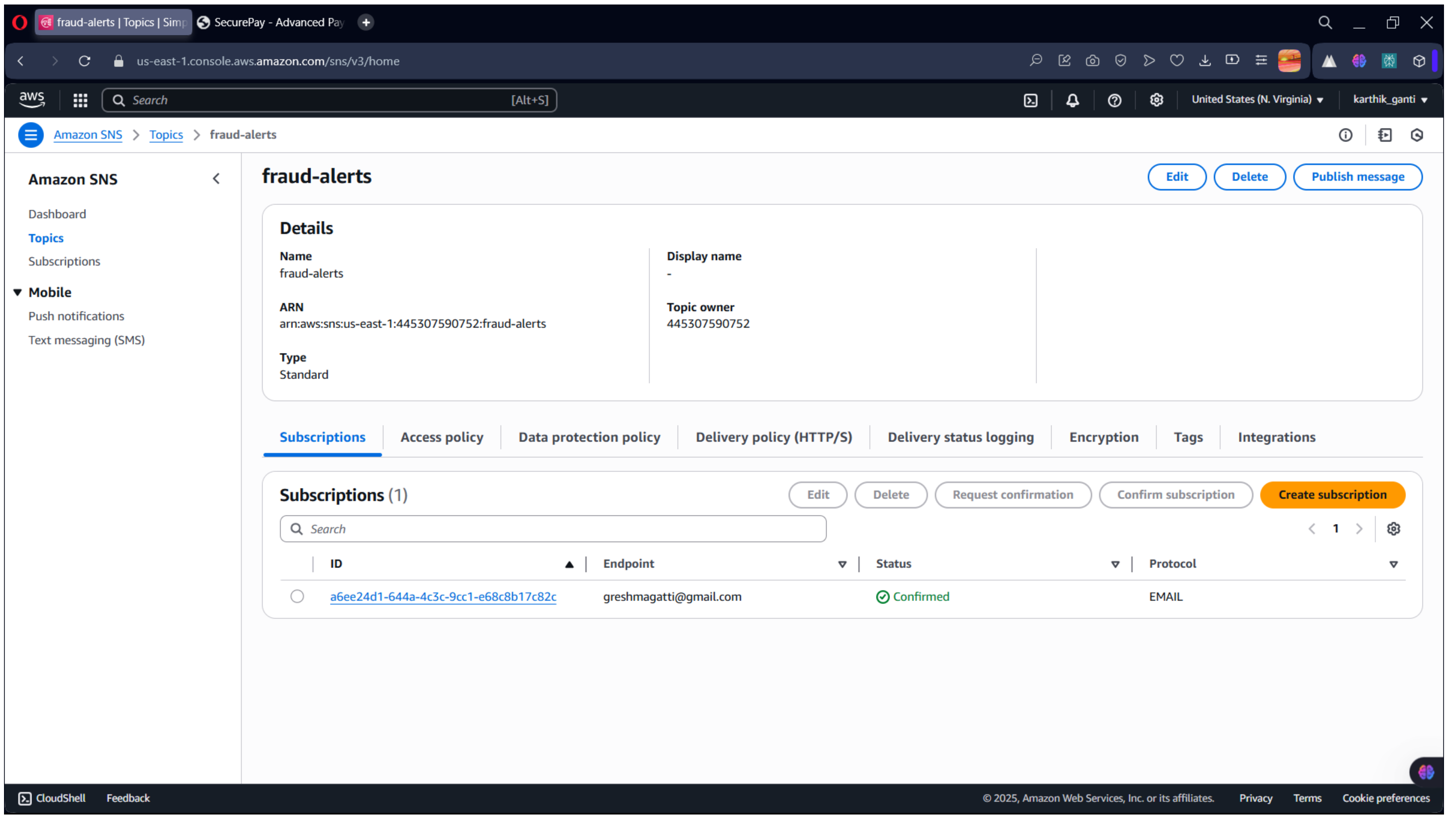
You can use the filter bar below to search for and match terms, phrases, or values in your log events.

```
Timestamp                   Message
2025-07-30T06:32:02.847Z    START RequestId: 5952d53b-f1f6-4a3e-abc1-438e43208b58 Version: $LATEST
2025-07-30T06:32:02.871Z    Event received:
2025-07-30T06:32:02.871Z    {"resource": "/transaction", "path": "/transaction", "httpMethod": "POST", "headers": {"": "", " \"Content-Type\"": " \"application/x-www-form-urlencoded\"",…
2025-07-30T06:32:02.871Z    Parsed and validated transaction:
2025-07-30T06:32:02.871Z    {
2025-07-30T06:32:02.871Z    "transaction_id": "07efc8fd-e70f-46e5-b9e1-d6755b2106e9",
2025-07-30T06:32:02.871Z    "user_id": "27cf0116-7969-4709-ae58-ed8d1415d9ac",
2025-07-30T06:32:02.871Z    "receiver_id": "19a948b0-144b-40f0-b20c-e6b52f3d128d",
2025-07-30T06:32:02.871Z    "amount": 13654.02,
2025-07-30T06:32:02.871Z    "currency": "USD",
2025-07-30T06:32:02.871Z    "location": "San Francisco",
2025-07-30T06:32:02.871Z    "timestamp": "2025-07-30T18:10:00"
2025-07-30T06:32:02.871Z    }
2025-07-30T06:32:03.133Z    Kinesis put_record response:
2025-07-30T06:32:03.133Z    {'ShardId': 'shardId-000000000002', 'SequenceNumber': '49665591579090083137072924615068331683638839527805550626', 'ResponseMetadata': {'RequestId': 'd64af40b…
2025-07-30T06:32:03.152Z    END RequestId: 5952d53b-f1f6-4a3e-abc1-438e43208b58
2025-07-30T06:32:03.152Z    REPORT RequestId: 5952d53b-f1f6-4a3e-abc1-438e43208b58 Duration: 304.68 ms Billed Duration: 305 ms Memory Size: 128 MB Max Memory Used: 85 MB
```

No newer events at this moment. *Auto retry paused.* Resume

Screenshot 15: Amazon CloudWatch – Log Events for SimulateTransactions

- This CloudWatch Log view shows recent log entries from the "SimulateTransactions" Lambda function.
- You can see timestamped logs for each invocation, including:
  - Raw HTTP requests received by the function
  - Parsed transaction details like transaction_id, user_id, amount, currency, location, and timestamp
  - Kinesis record publishing results, showing the processed transaction is pushed into a Kinesis data stream
- These logs are crucial for monitoring, debugging, and validating that simulated transactions are being handled and forwarded as intended.



fraud-alerts

Details

Name fraud-alerts	Display name -
ARN arn:aws:sns:us-east-1:445307590752:fraud-alerts	Topic owner 445307590752
Type Standard	

Subscriptions (1)

ID	Endpoint	Status	Protocol
a6ee24d1-644a-4c3c-9cc1-e68c8b17c82c	greshmagatti@gmail.com	✓ Confirmed	EMAIL

Screenshot 16: Amazon SNS – fraud-alerts Topic

- This page displays a Simple Notification Service (SNS) topic named "fraud-alerts".
- Key information:
  - ARN (Amazon Resource Name): Uniquely identifies this SNS topic for integrations.
  - Type: Standard topic (not FIFO).
- Under **Subscriptions**, there is one subscriber: the endpoint is an email address (greshmangatti@gmail.com), and the protocol is EMAIL with status "Confirmed".
- Purpose: Alerts about suspicious or fraudulent transactions (detected via the fraud detection system) can be sent immediately by publishing to this SNS topic, triggering automated email notifications to recipients.

11. Conclusion

This project demonstrates how AWS Fraud Detector can be integrated into an online transaction system for instant fraud detection. Future enhancements could include face verification with Rekognition and Step Functions for more complex workflows.

12. References

• AWS Fraud Detector Documentation: https://docs.aws.amazon.com/frauddetector/

• AWS Lambda Documentation: https://docs.aws.amazon.com/lambda/

• AWS Kinesis Documentation: https://docs.aws.amazon.com/kinesis/