Real-Time Transaction Monitoring & Alerts

Event-driven monitoring system for fraud detection and customer alerts

Version: 1.0

Author: Security & ML Engineering Team

Category: Real-Time Systems | Fraud Detection | Alerting

Last Updated: December 12, 2025

This document details Monzo real-time transaction monitoring system that processes 10+ million transactions daily with sub-100ms latency. The system combines rule-based detection, machine learning fraud models, and intelligent alerting to protect customers while minimizing false positives. Learn how to integrate with monitoring APIs and configure custom alerts.

System Architecture Overview

Real-Time Processing Pipeline

Every transaction flows through a multi-stage monitoring pipeline:

Stage 1: Pre-Authorization Checks (5ms) - Balance verification, spending limits, blocked merchants

Stage 2: Fraud Detection ML (15ms) - Random Forest model scores transaction risk 0-1

Stage 3: Rules Engine (10ms) - Custom rules fire alerts based on patterns

Stage 4: Alert Delivery (50ms) - Push notifications sent to user devices

Total Pipeline Latency: 80ms P50, 120ms P95, 200ms P99

Technology Stack

Component	Technology	Purpose
Event Stream	Apache Kafka	10M+ transactions/day ingestion
Processing	Apache Flink	Stream processing, windowing
ML Inference	TensorFlow Serving	Real-time fraud scoring
Rules Engine	Drools	Complex event processing
Alerting	AWS SNS + Firebase	Multi-channel delivery

Metrics	Prometheus + Grafana	System observability

Alert Categories & Configuration

Built-In Alert Types

Monzo provides pre-configured alerts that customers can enable:

Alert Type	Trigger Condition	Default State	Delivery
Large Purchase	Amount > 100 GBP	Enabled	Push + App
Foreign Transaction	Country != home country	Enabled	Push
Online Purchase	Card-not-present	Disabled	App only
ATM Withdrawal	Any ATM transaction	Disabled	Push + SMS
Declined Transaction	Payment declined	Enabled	Push
Low Balance	Balance < 20 GBP	Enabled	App
Spending Limit	80% of budget reached	Enabled	App
Unusual Activity	ML fraud score > 0.7	Enabled	Push + SMS

Configure Alerts via API

```
# Enable/disable alerts
PUT /api/alerts/preferences
Authorization: Bearer {access_token}

{
  "alert_preferences": {
    "large_purchase": {
      "enabled": true,
      "threshold_gbp": 100,
      "channels": ["push", "app"]
    },
    "foreign_transaction": {
      "enabled": true,
      "channels": ["push"]
    },
    "low_balance": {
      "enabled": true,
      "threshold_gbp": 20,
      "channels": ["app"]
    }
  }
}
```

Custom Alert Rules

Create Velocity-Based Rules

Create custom rules to detect suspicious patterns:

```
POST /api/alerts/rules
Authorization: Bearer {access_token}

{
  "rule_name": "rapid_spending_burst",
  "description": "Alert if 5+ transactions in 10 minutes",
  "condition": {
    "metric": "transaction_count",
    "window": "10m",
    "threshold": 5,
    "operator": "greater_than"
  },
  "actions": [
    {
      "type": "alert",
      "severity": "high",
      "message": "Unusual spending activity detected",
      "channels": ["push", "sms"]
    },
    {
      "type": "block_card",
      "duration_seconds": 300
    }
  ]
}
```

Geographic Velocity Rules

```
# Detect impossible travel
{
  "rule_name": "geographic_impossibility",
  "condition": {
    "type": "velocity",
    "max_speed_kmh": 800,
    "window": "1h"
  },
  "trigger": "two_transactions_different_locations",
  "action": {
    "type": "alert",
    "severity": "critical",
    "message": "Card used in two distant locations",
    "require_confirmation": true
  }
}
```

Receiving Alerts via Webhooks

Register Webhook for Alerts

```
POST /api/webhooks
Authorization: Bearer {access_token}

{
  "url": "https://yourapp.com/monzo/alerts",
  "event_types": [
```

```
    "alert.large_purchase",
    "alert.foreign_transaction",
    "alert.fraud_detected"
  ]
}
```

Alert Webhook Payload

```
{
  "type": "alert.large_purchase",
  "alert_id": "alert_00009kL3mN8pQ2rS",
  "created_at": "2024-12-12T14:30:15Z",
  "data": {
    "transaction_id": "tx_00009kL3mN8pQ2rS",
    "account_id": "acc_00009237aqC8c79",
    "amount": 15000,
    "currency": "GBP",
    "merchant": {
      "name": "Apple Store Regent Street",
      "category": "electronics",
      "location": {
        "city": "London",
        "country": "GB"
      }
    },
    "alert_reason": "amount_exceeds_threshold",
    "threshold_gbp": 100,
    "requires_confirmation": false
  }
}
```

Fraud Detection Integration

Real-Time Fraud Score API

Query fraud scores for transactions in real-time:

```
GET /api/fraud/score/{transaction_id}
Authorization: Bearer {access_token}

Response:
{
  "transaction_id": "tx_00009kL3mN8pQ2rS",
  "fraud_score": 0.23,
  "risk_level": "low",
  "contributing_factors": [
    {
      "factor": "merchant_familiarity",
      "score": 0.1,
      "description": "Customer uses this merchant regularly"
    },
    {
      "factor": "amount_deviation",
      "score": 0.05,
      "description": "Amount within normal range"
    },
    {
      "factor": "velocity",
      "score": 0.08,
```

```
      "description": "Transaction frequency normal"
    }
  ],
  "actions_taken": []
}
```

High-Risk Transaction Handling

When fraud score exceeds 0.85, automatic actions trigger:

```
{
  "fraud_score": 0.92,
  "risk_level": "critical",
  "actions_taken": [
    {
      "action": "transaction_blocked",
      "reason": "high_fraud_probability",
      "timestamp": "2024-12-12T14:30:15Z"
    },
    {
      "action": "card_temporarily_frozen",
      "duration_seconds": 1800,
      "timestamp": "2024-12-12T14:30:15Z"
    },
    {
      "action": "sms_sent",
      "message": "Suspicious transaction detected. "
                 "Reply YES to confirm or NO to block.",
      "timestamp": "2024-12-12T14:30:16Z"
    },
    {
      "action": "fraud_team_notified",
      "priority": "high",
      "timestamp": "2024-12-12T14:30:16Z"
    }
  ]
}
```

Alert Acknowledgment & Response

Confirm or Dispute Transaction

```
POST /api/alerts/{alert_id}/respond
Authorization: Bearer {access_token}

{
  "response": "confirmed",
  "transaction_id": "tx_00009kL3mN8pQ2rS",
  "notes": "Legitimate purchase"
}

# Or dispute transaction
{
  "response": "dispute",
  "transaction_id": "tx_00009kL3mN8pQ2rS",
  "reason": "unauthorized",
  "notes": "I did not make this purchase"
}
```

Unfreeze Card After Fraud Alert

```
POST /api/cards/{card_id}/unfreeze
Authorization: Bearer {access_token}

{
  "reason": "false_positive_confirmed",
  "alert_id": "alert_00009kL3mN8pQ2rS"
}
```

Spending Insights & Analytics

Query Spending Patterns

```
GET /api/analytics/spending
Authorization: Bearer {access_token}
?period=last_30_days
&group_by=category

Response:
{
  "period": "2024-11-12 to 2024-12-12",
  "total_spent": 2847.50,
  "currency": "GBP",
  "breakdown": [
    {
      "category": "groceries",
      "amount": 847.20,
      "percentage": 29.7,
      "transaction_count": 42,
      "trend": "up_15_percent"
    },
    {
      "category": "transport",
      "amount": 425.80,
      "percentage": 15.0,
      "transaction_count": 68,
      "trend": "stable"
    }
  ]
}
```

Budget Progress Monitoring

```
GET /api/budgets/{budget_id}/progress
Authorization: Bearer {access_token}

Response:
{
  "budget_id": "budget_groceries_monthly",
  "category": "groceries",
  "period": "monthly",
  "limit": 500.00,
  "spent": 412.30,
  "remaining": 87.70,
  "percentage_used": 82.5,
  "days_remaining": 8,
  "pace": "on_track",
  "alerts": [
```

```
    {
      "type": "threshold_warning",
      "message": "80% of grocery budget used",
      "triggered_at": "2024-12-10T09:15:00Z"
    }
  ]
}
```

Performance & Reliability

System SLAs

Metric	Target	Actual (30-day)	Monitoring
Alert Latency P50	< 100ms	78ms	Prometheus
Alert Latency P99	< 500ms	342ms	Prometheus
Delivery Success	> 99.5%	99.87%	SNS metrics
False Positive Rate	< 2%	1.3%	Manual review
Uptime	> 99.9%	99.97%	PagerDuty
Processing Throughput	> 3000 TPS	3847 TPS	Kafka lag

Best Practices

1. Configure alerts per user preference - avoid alert fatigue
2. Use webhooks for real-time notifications in your application
3. Implement retry logic with exponential backoff for webhook delivery
4. Verify webhook signatures to prevent spoofing
5. Cache fraud scores for 30 seconds to reduce API calls
6. Provide easy ways for users to confirm/dispute transactions
7. Show spending insights proactively to build trust
8. Test alert flows in sandbox before production deployment
9. Monitor webhook delivery success rates and investigate failures
10. Respect user notification preferences across all channels