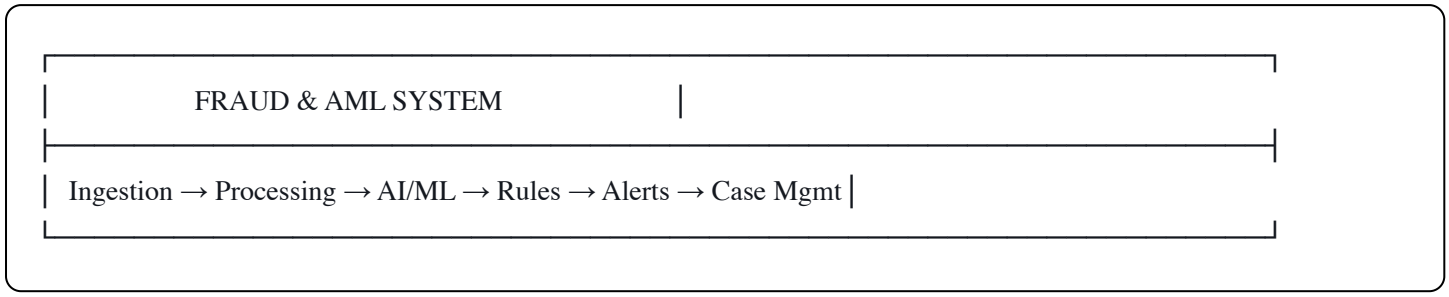
Fraud & AML Detection System - Java + AI Architecture Guide

System Overview

A modern fraud and AML system that combines traditional rule-based detection with AI/ML models for intelligent pattern recognition.



Architecture Components

1. Data Ingestion Layer

- Real-time transaction streaming
- Batch customer data processing
- External data enrichment

2. AI/ML Detection Layer

- Anomaly detection
- Pattern recognition
- Risk scoring
- Behavioral analysis

3. Rules Engine

- Regulatory compliance (FATF, FinCEN)
- Threshold-based alerts
- Watchlist screening
- Sanctions checking

4. Case Management

- Alert investigation
- Suspicious Activity Reports (SARs)
- Workflow automation

5. Analytics & Reporting

- Real-time dashboards
- Compliance reporting
- Model performance monitoring

Technology Stack Recommendation

Core Java Stack

```
├── Spring Boot 3.x          (Application framework)
├── Spring Cloud             (Microservices)
├── Apache Kafka             (Event streaming)
├── PostgreSQL               (Transactional data)
├── Redis                    (Caching, real-time scoring)
├── Elasticsearch            (Search, analytics)
├── Apache Flink             (Stream processing)
└── Docker + Kubernetes      (Containerization)
```

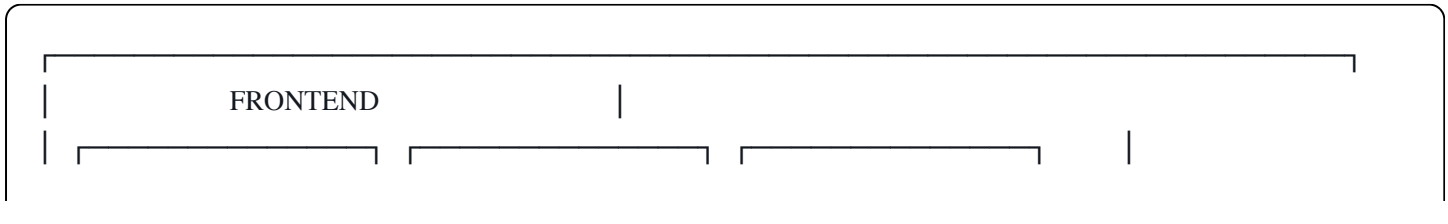
AI/ML Integration

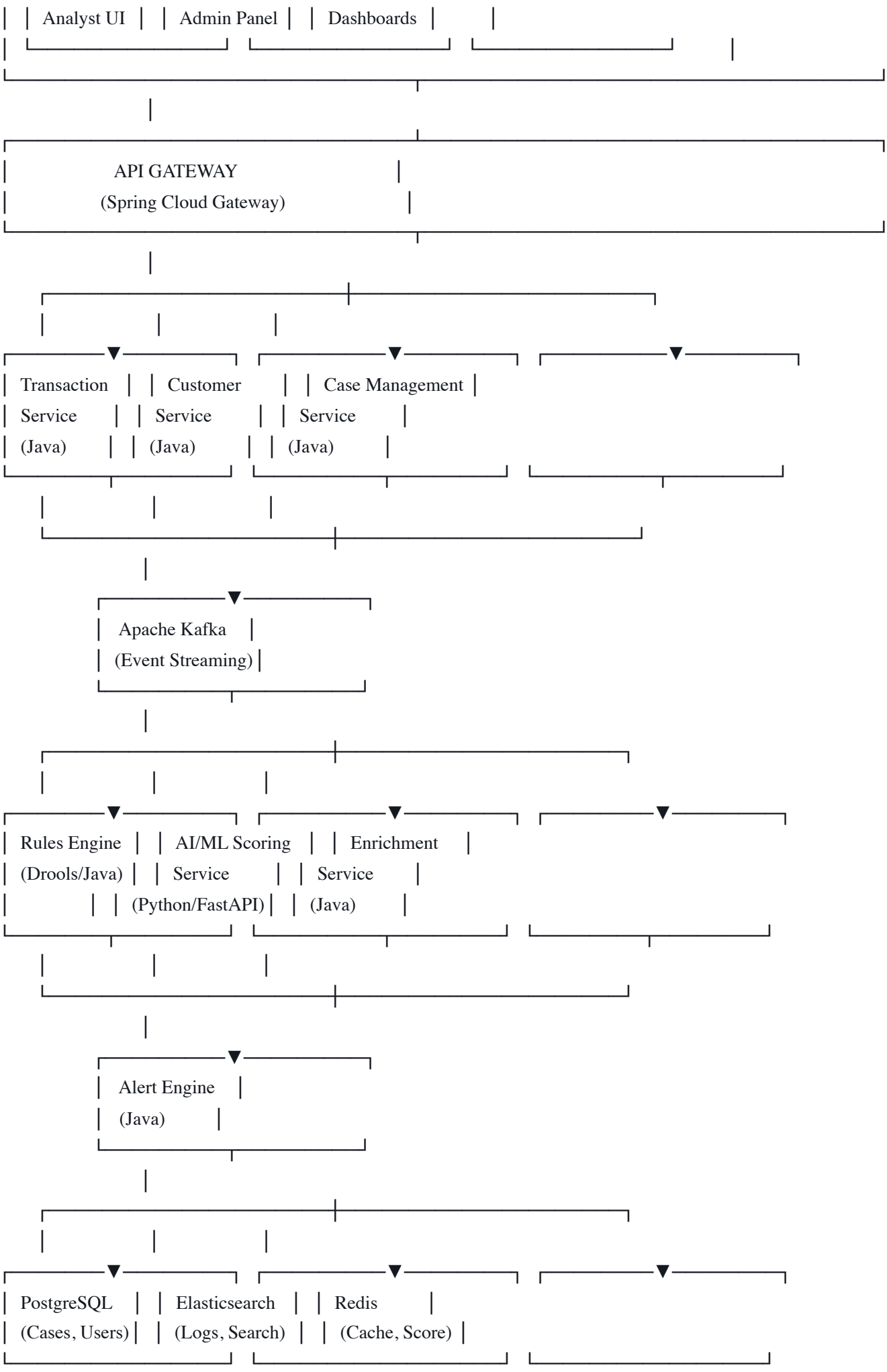
```
├── Python (ML Models)       (Scikit-learn, TensorFlow, PyTorch)
├── MLflow                   (Model management)
├── FastAPI                  (ML model serving)
├── Seldon Core / KServe     (Model deployment on K8s)
├── Apache Spark             (Big data ML training)
└── H2O.ai / Databricks      (AutoML platforms)
```

Alternative: Java-Native ML

```
├── Deeplearning4j           (Deep learning in Java)
├── Tribuo (Oracle)          (ML library)
├── Weka                     (Classic ML)
└── Apache Mahout            (Distributed ML)
```

System Architecture Diagram




Analyst UI
Admin Panel
Dashboards
API GATEWAY
(Spring Cloud Gateway)
Transaction Service (Java)
Customer Service (Java)
Case Management Service (Java)
Apache Kafka
(Event Streaming)
Rules Engine (Drools/Java)
AI/ML Scoring Service (Python/FastAPI)
Enrichment Service (Java)
Alert Engine
(Java)
PostgreSQL
(Cases, Users)
Elasticsearch
(Logs, Search)
Redis
(Cache, Score)

Core Java Services Implementation

1. Transaction Monitoring Service

java

```java
// TransactionMonitoringService.java
package com.fraudaml.monitoring;

import org.springframework.stereotype.Service;
import org.springframework.kafka.annotation.KafkaListener;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;

@Service
@Slf4j
@RequiredArgsConstructor
public class TransactionMonitoringService {

    private final RulesEngineService rulesEngine;
    private final MLScoringService mlScoringService;
    private final AlertService alertService;
    private final EnrichmentService enrichmentService;

    @KafkaListener(topics = "transactions", groupId = "fraud-detection")
    public void processTransaction(Transaction transaction) {
        log.info("Processing transaction: {}", transaction.getId());

        try {
            // 1. Enrich transaction data
            EnrichedTransaction enriched = enrichmentService.enrich(transaction);

            // 2. Get AI/ML risk score (async)
            CompletableFuture<MLScore> mlScore =
                mlScoringService.scoreTransactionAsync(enriched);

            // 3. Run rules engine
            RuleResult ruleResult = rulesEngine.evaluate(enriched);

            // 4. Combine ML score with rules
            MLScore score = mlScore.join(); // Wait for ML result

            // 5. Generate alert if needed
            if (shouldGenerateAlert(ruleResult, score)) {
                Alert alert = createAlert(enriched, ruleResult, score);
                alertService.publish(alert);
            }

            // 6. Store results
            storeAnalysisResult(enriched, ruleResult, score);

        } catch (Exception e) {
```

```
            log.error("Error processing transaction: {}", transaction.getId(), e);
            // Dead letter queue handling
        }
    }

    private boolean shouldGenerateAlert(RuleResult ruleResult, MLScore mlScore) {
        // High ML score OR rule violations
        return mlScore.getScore() > 0.7 || ruleResult.hasViolations();
    }

    private Alert createAlert(EnrichedTransaction tx,
                              RuleResult ruleResult,
                              MLScore mlScore) {
        return Alert.builder()
            .transactionId(tx.getId())
            .alertType(determineAlertType(ruleResult, mlScore))
            .severity(calculateSeverity(mlScore.getScore()))
            .riskScore(mlScore.getScore())
            .violatedRules(ruleResult.getViolations())
            .features(mlScore.getFeatures())
            .timestamp(Instant.now())
            .status(AlertStatus.NEW)
            .build();
    }
}
```

2. Transaction Entity

java

```java
// Transaction.java
package com.fraudaml.domain;

import lombok.Data;
import java.math.BigDecimal;
import java.time.Instant;

@Data
public class Transaction {
    private String id;
    private String customerId;
    private String accountId;

    // Transaction details
    private BigDecimal amount;
    private String currency;
    private TransactionType type; // DEPOSIT, WITHDRAWAL, TRANSFER, PAYMENT
    private String channel; // ATM, ONLINE, MOBILE, BRANCH

    // Counterparty
    private String beneficiaryId;
    private String beneficiaryName;
    private String beneficiaryCountry;

    // Location
    private String ipAddress;
    private GeoLocation location;
    private String deviceId;

    // Timing
    private Instant timestamp;
    private String merchantCategory;

    // Additional
    private Map<String, Object> metadata;
}

@Data
class GeoLocation {
    private Double latitude;
    private Double longitude;
    private String country;
    private String city;
}
```

3. Rules Engine (Drools Integration)

java

```java
// RulesEngineService.java
package com.fraudaml.rules;

import org.kie.api.runtime.KieSession;
import org.springframework.stereotype.Service;
import lombok.RequiredArgsConstructor;

@Service
@RequiredArgsConstructor
public class RulesEngineService {

    private final KieSession kieSession;

    public RuleResult evaluate(EnrichedTransaction transaction) {
        RuleResult result = new RuleResult();

        // Insert facts into working memory
        kieSession.insert(transaction);
        kieSession.insert(result);

        // Fire all rules
        kieSession.fireAllRules();

        // Dispose session
        kieSession.dispose();

        return result;
    }
}
```

4. Sample Drools Rules

java

```
// fraud-rules.drl
package com.fraudaml.rules;

import com.fraudaml.domain.EnrichedTransaction;
import com.fraudaml.rules.RuleResult;
import java.math.BigDecimal;

// Rule 1: High Value Transaction
rule "High Value Transaction"
    when
        $tx : EnrichedTransaction(amount > 10000)
    then
        RuleResult result = (RuleResult) kcontext.getKieRuntime()
            .getGlobal("result");
        result.addViolation("HIGH_VALUE",
            "Transaction exceeds $10,000: " + $tx.getAmount());
end

// Rule 2: Rapid Fire Transactions
rule "Rapid Fire Transactions"
    when
        $tx : EnrichedTransaction()
        $count : Number(intValue >= 5) from accumulate(
            EnrichedTransaction(
                customerId == $tx.customerId,
                timestamp > $tx.timestamp.minusMinutes(5)
            ),
            count(1)
        )
    then
        RuleResult result = (RuleResult) kcontext.getKieRuntime()
            .getGlobal("result");
        result.addViolation("RAPID_FIRE",
            "5+ transactions in 5 minutes");
end

// Rule 3: Unusual Location
rule "Unusual Location"
    when
        $tx : EnrichedTransaction(
            location.country != customerProfile.homeCountry,
            timeSinceLastTransaction < 2 hours
        )
    then
        RuleResult result = (RuleResult) kcontext.getKieRuntime()
            .getGlobal("result");
```

```
        result.addViolation("LOCATION_ANOMALY",
            "Transaction from unusual location");
end

// Rule 4: Structuring Detection (AML)
rule "Structuring Detection"
    when
        $count : Number(intValue >= 3) from accumulate(
            Transaction(
                customerId == $customerId,
                amount > 9000 && amount < 10000,
                timestamp > today.minusDays(1)
            ),
            count(1)
        )
    then
        result.addViolation("STRUCTURING",
            "Multiple transactions just below reporting threshold");
end

// Rule 5: Sanctioned Country
rule "Sanctioned Country"
    when
        $tx : EnrichedTransaction(
            beneficiaryCountry in ("IR", "KP", "SY", "CU")
        )
    then
        result.addViolation("SANCTIONS",
            "Transaction to sanctioned country: " +
            $tx.getBeneficiaryCountry());
end
```

AI/ML Integration - Python Service

ML Scoring Service (FastAPI)

python

```python
# ml_scoring_service.py
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
import numpy as np
import joblib
from typing import Dict, List
import logging

app = FastAPI(title="Fraud Detection ML Service")
logger = logging.getLogger(__name__)

# Load trained models
fraud_model = joblib.load('models/fraud_detector_v1.pkl')
aml_model = joblib.load('models/aml_detector_v1.pkl')
scaler = joblib.load('models/scaler.pkl')

class Transaction(BaseModel):
    id: str
    amount: float
    customer_id: str
    transaction_type: str
    channel: str
    beneficiary_country: str
    hour_of_day: int
    day_of_week: int
    customer_age_days: int
    avg_transaction_amount_30d: float
    transaction_count_24h: int
    customer_risk_score: float
    ip_country: str
    device_fingerprint: str

class MLScore(BaseModel):
    transaction_id: str
    fraud_score: float
    aml_score: float
    combined_score: float
    features: Dict[str, float]
    model_version: str
    explanation: List[str]

@app.post("/score", response_model=MLScore)
async def score_transaction(transaction: Transaction):
    """
    Score a transaction for fraud and AML risk
    """
```

```
    try:
        # Feature engineering
        features = extract_features(transaction)

        # Scale features
        X = scaler.transform([features])

        # Get predictions
        fraud_prob = fraud_model.predict_proba(X)[0][1]
        aml_prob = aml_model.predict_proba(X)[0][1]

        # Combined score (weighted average)
        combined_score = 0.6 * fraud_prob + 0.4 * aml_prob

        # Feature importance for explainability
        feature_importance = get_feature_importance(X, fraud_model)

        # Generate explanation
        explanation = generate_explanation(
            transaction,
            fraud_prob,
            aml_prob,
            feature_importance
        )

        return MLScore(
            transaction_id=transaction.id,
            fraud_score=round(fraud_prob, 4),
            aml_score=round(aml_prob, 4),
            combined_score=round(combined_score, 4),
            features=feature_importance,
            model_version="v1.0.0",
            explanation=explanation
        )

    except Exception as e:
        logger.error(f"Error scoring transaction {transaction.id}: {e}")
        raise HTTPException(status_code=500, detail=str(e))

def extract_features(tx: Transaction) -> List[float]:
    """
    Extract and engineer features for ML model
    """
    features = [
        tx.amount,
        tx.hour_of_day,
        tx.day_of_week,
```

```
        tx.customer_age_days,
        tx.avg_transaction_amount_30d,
        tx.transaction_count_24h,
        tx.customer_risk_score,

        # Derived features
        tx.amount / (tx.avg_transaction_amount_30d + 1),  # Amount ratio
        1 if tx.hour_of_day < 6 or tx.hour_of_day > 22 else 0,  # Unusual hours
        1 if tx.beneficiary_country != tx.ip_country else 0,  # Country mismatch

        # One-hot encoded categorical features
        1 if tx.transaction_type == "WITHDRAWAL" else 0,
        1 if tx.transaction_type == "TRANSFER" else 0,
        1 if tx.channel == "ONLINE" else 0,
        1 if tx.channel == "ATM" else 0,
    ]

    return features

def get_feature_importance(X, model) -> Dict[str, float]:
    """
    Get top feature contributions for explainability
    """
    feature_names = [
        "amount", "hour", "day_of_week", "customer_age",
        "avg_amount_30d", "tx_count_24h", "customer_risk",
        "amount_ratio", "unusual_hours", "country_mismatch"
    ]

    # Get feature importance from model
    if hasattr(model, 'feature_importances_'):
        importance = model.feature_importances_
    else:
        # For linear models
        importance = np.abs(model.coef_[0])

    # Normalize and create dict
    importance_norm = importance / np.sum(importance)

    return {
        name: round(float(imp), 4)
        for name, imp in zip(feature_names, importance_norm)
    }

def generate_explanation(tx: Transaction,
                         fraud_score: float,
                         aml_score: float,
```

```
                features: Dict[str, float]) -> List[str]:
    """
    Generate human-readable explanation
    """
    explanations = []

    # High risk factors
    if fraud_score > 0.7:
        explanations.append("⚠️ High fraud risk detected")

        if features.get('amount_ratio', 0) > 0.15:
            explanations.append("• Transaction amount significantly exceeds customer norm")

        if features.get('unusual_hours', 0) > 0.1:
            explanations.append("• Transaction at unusual time")

        if tx.transaction_count_24h > 10:
            explanations.append(f"• High transaction velocity: {tx.transaction_count_24h} in 24h")

    if aml_score > 0.7:
        explanations.append("⚠️ High AML risk detected")

        if features.get('country_mismatch', 0) > 0.1:
            explanations.append("• IP location differs from beneficiary country")

    return explanations if explanations else ["✓ No significant risk factors detected"]

@app.get("/health")
async def health_check():
    return {"status": "healthy", "models_loaded": True}

@app.get("/models/info")
async def model_info():
    return {
        "fraud_model": "RandomForestClassifier",
        "aml_model": "XGBoostClassifier",
        "version": "v1.0.0",
        "last_trained": "2024-01-15"
    }
```

Java ML Service Client

java

```java
// MLScoringService.java (Java client)
package com.fraudaml.ml;

import org.springframework.stereotype.Service;
import org.springframework.web.reactive.function.client.WebClient;
import reactor.core.publisher.Mono;
import lombok.RequiredArgsConstructor;

@Service
@RequiredArgsConstructor
public class MLScoringService {

    private final WebClient mlServiceClient;

    public CompletableFuture<MLScore> scoreTransactionAsync(
            EnrichedTransaction transaction) {

        return mlServiceClient
            .post()
            .uri("/score")
            .bodyValue(convertToMLRequest(transaction))
            .retrieve()
            .bodyToMono(MLScore.class)
            .toFuture();
    }

    public MLScore scoreTransaction(EnrichedTransaction transaction) {
        return mlServiceClient
            .post()
            .uri("/score")
            .bodyValue(convertToMLRequest(transaction))
            .retrieve()
            .bodyToMono(MLScore.class)
            .block();
    }

    private MLTransactionRequest convertToMLRequest(EnrichedTransaction tx) {
        return MLTransactionRequest.builder()
            .id(tx.getId())
            .amount(tx.getAmount().doubleValue())
            .customerId(tx.getCustomerId())
            .transactionType(tx.getType().name())
            .channel(tx.getChannel())
            .beneficiaryCountry(tx.getBeneficiaryCountry())
            .hourOfDay(tx.getTimestamp().get(ChronoField.HOUR_OF_DAY))
            .dayOfWeek(tx.getTimestamp().get(ChronoField.DAY_OF_WEEK))
```

```
            .customerAgeDays(tx.getCustomerProfile().getAgeDays())
            .avgTransactionAmount30d(tx.getCustomerProfile().getAvgAmount30d())
            .transactionCount24h(tx.getCustomerProfile().getTransactionCount24h())
            .customerRiskScore(tx.getCustomerProfile().getRiskScore())
            .ipCountry(tx.getLocation().getCountry())
            .deviceFingerprint(tx.getDeviceId())
            .build();
    }
}
```

Alert Management Service

java

```java
// AlertService.java
package com.fraudaml.alert;

import org.springframework.stereotype.Service;
import org.springframework.kafka.core.KafkaTemplate;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;

@Service
@Slf4j
@RequiredArgsConstructor
public class AlertService {

    private final KafkaTemplate<String, Alert> kafkaTemplate;
    private final AlertRepository alertRepository;
    private final NotificationService notificationService;

    public void publish(Alert alert) {
        // 1. Save to database
        Alert saved = alertRepository.save(alert);

        // 2. Publish to Kafka for real-time processing
        kafkaTemplate.send("alerts", alert.getId(), saved);

        // 3. Send notifications based on severity
        if (alert.getSeverity() == Severity.CRITICAL) {
            notificationService.sendUrgentAlert(alert);
        }

        log.info("Alert published: {} - Severity: {}, Score: {}",
                alert.getId(),
                alert.getSeverity(),
                alert.getRiskScore());
    }

    public List<Alert> getAlertsForInvestigation(String analystId) {
        return alertRepository.findByStatusAndAssignedTo(
            AlertStatus.NEW,
            analystId
        );
    }

    public Alert investigateAlert(String alertId, Investigation investigation) {
        Alert alert = alertRepository.findById(alertId)
            .orElseThrow(() -> new AlertNotFoundException(alertId));
```

```
        alert.setStatus(AlertStatus.INVESTIGATING);
        alert.setInvestigation(investigation);
        alert.setAssignedTo(investigation.getAnalystId());

        return alertRepository.save(alert);
    }

    public Alert closeAlert(String alertId, AlertResolution resolution) {
        Alert alert = alertRepository.findById(alertId)
            .orElseThrow(() -> new AlertNotFoundException(alertId));

        alert.setStatus(AlertStatus.CLOSED);
        alert.setResolution(resolution);
        alert.setClosedAt(Instant.now());

        // If SAR is needed, trigger filing workflow
        if (resolution.isFileSAR()) {
            sarFilingService.initiateSAR(alert);
        }

        return alertRepository.save(alert);
    }
}
```

Case Management

java

```java
// CaseManagementService.java
package com.fraudaml.cases;

import org.springframework.stereotype.Service;
import lombok.RequiredArgsConstructor;

@Service
@RequiredArgsConstructor
public class CaseManagementService {

    private final CaseRepository caseRepository;
    private final CustomerRepository customerRepository;
    private final TransactionRepository transactionRepository;

    public Case createCase(Alert alert) {
        Customer customer = customerRepository.findById(alert.getCustomerId())
            .orElseThrow();

        Case investigationCase = Case.builder()
            .customerId(customer.getId())
            .customerName(customer.getName())
            .alerts(List.of(alert))
            .priority(calculatePriority(alert))
            .status(CaseStatus.OPEN)
            .createdAt(Instant.now())
            .build();

        return caseRepository.save(investigationCase);
    }

    public CaseView getCaseDetails(String caseId) {
        Case case = caseRepository.findById(caseId)
            .orElseThrow(() -> new CaseNotFoundException(caseId));

        // Gather all related data
        List<Transaction> transactions = transactionRepository
            .findByCustomerId(case.getCustomerId());

        Customer customer = customerRepository
            .findById(case.getCustomerId())
            .orElseThrow();

        List<Alert> relatedAlerts = alertRepository
            .findByCustomerId(case.getCustomerId());

        // Build comprehensive view
```

```
        return CaseView.builder()
            .caseInfo(case)
            .customer(customer)
            .transactions(transactions)
            .alerts(relatedAlerts)
            .timeline(buildTimeline(case, transactions, relatedAlerts))
            .riskAssessment(assessRisk(customer, transactions))
            .build();
    }

    public void escalateCase(String caseId, String reason) {
        Case case = caseRepository.findById(caseId)
            .orElseThrow();

        case.setPriority(Priority.CRITICAL);
        case.setEscalated(true);
        case.setEscalationReason(reason);
        case.setEscalatedAt(Instant.now());

        caseRepository.save(case);

        // Notify senior analysts
        notificationService.notifyEscalation(case);
    }
}
```

Key ML Models to Implement

1. Anomaly Detection

```
python
```

```python
# Isolation Forest for anomaly detection
from sklearn.ensemble import IsolationForest

def train_anomaly_detector(normal_transactions):
    model = IsolationForest(
        contamination=0.01,  # 1% expected anomalies
        random_state=42
    )
    model.fit(normal_transactions)
    return model

# Usage
anomaly_score = model.predict(transaction_features)
# -1 = anomaly, 1 = normal
```

2. Fraud Classification

```python
# XGBoost for fraud detection
import xgboost as xgb

def train_fraud_classifier(X_train, y_train):
    model = xgb.XGBClassifier(
        max_depth=6,
        learning_rate=0.1,
        n_estimators=100,
        scale_pos_weight=50  # Handle imbalanced data
    )
    model.fit(X_train, y_train)
    return model
```

3. Network Analysis (AML)

```python
```

```
# Graph neural network for money laundering detection
import networkx as nx

def detect_laundering_networks(transactions):
    # Build transaction graph
    G = nx.DiGraph()

    for tx in transactions:
        G.add_edge(
            tx.sender,
            tx.receiver,
            amount=tx.amount,
            timestamp=tx.timestamp
        )

    # Detect suspicious patterns
    # 1. Circular flows
    cycles = nx.simple_cycles(G)

    # 2. Layering (many hops)
    paths = nx.all_simple_paths(G, source, target, cutoff=10)

    # 3. Rapid movement
    velocity = calculate_money_velocity(G)

    return suspicious_patterns
```

Database Schema

```
sql
```

```sql
-- PostgreSQL Schema

-- Transactions
CREATE TABLE transactions (
    id UUID PRIMARY KEY,
    customer_id VARCHAR(50) NOT NULL,
    account_id VARCHAR(50) NOT NULL,
    amount DECIMAL(15,2) NOT NULL,
    currency VARCHAR(3) NOT NULL,
    transaction_type VARCHAR(20) NOT NULL,
    channel VARCHAR(20) NOT NULL,
    beneficiary_id VARCHAR(50),
    beneficiary_name VARCHAR(200),
    beneficiary_country VARCHAR(2),
    ip_address INET,
    device_id VARCHAR(100),
    merchant_category VARCHAR(50),
    timestamp TIMESTAMP NOT NULL,
    created_at TIMESTAMP DEFAULT NOW(),
    INDEX idx_customer (customer_id),
    INDEX idx_timestamp (timestamp),
    INDEX idx_amount (amount)
);

-- Alerts
CREATE TABLE alerts (
    id UUID PRIMARY KEY,
    transaction_id UUID REFERENCES transactions(id),
    customer_id VARCHAR(50) NOT NULL,
    alert_type VARCHAR(50) NOT NULL,
    severity VARCHAR(20) NOT NULL,
    risk_score DECIMAL(5,4) NOT NULL,
    violated_rules JSONB,
    ml_features JSONB,
    status VARCHAR(20) NOT NULL,
    assigned_to VARCHAR(50),
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW(),
    closed_at TIMESTAMP,
    INDEX idx_status (status),
    INDEX idx_customer (customer_id),
    INDEX idx_risk_score (risk_score DESC)
);

-- Cases
CREATE TABLE cases (
```

```
    id UUID PRIMARY KEY,
    customer_id VARCHAR(50) NOT NULL,
    customer_name VARCHAR(200),
    priority VARCHAR(20) NOT NULL,
    status VARCHAR(20) NOT NULL,
    escalated BOOLEAN DEFAULT FALSE,
    escalation_reason TEXT,
    resolution VARCHAR(20),
    resolution_notes TEXT,
    sar_filed BOOLEAN DEFAULT FALSE,
    sar_id VARCHAR(50),
    created_at TIMESTAMP DEFAULT NOW(),
    closed_at TIMESTAMP,
    INDEX idx_status (status),
    INDEX idx_customer (customer_id)
);

-- Customer Risk Profiles
CREATE TABLE customer_risk_profiles (
    customer_id VARCHAR(50) PRIMARY KEY,
    base_risk_score DECIMAL(5,4) NOT NULL,
    transaction_count_30d INT,
    avg_transaction_amount DECIMAL(15,2),
    max_transaction_amount DECIMAL(15,2),
    high_risk_countries JSONB,
    kyc_status VARCHAR(20),
    pep_status BOOLEAN DEFAULT FALSE,
    sanction_hit BOOLEAN DEFAULT FALSE,
    updated_at TIMESTAMP DEFAULT NOW()
);

-- Model Performance
CREATE TABLE model_performance (
    id SERIAL PRIMARY KEY,
    model_name VARCHAR(50) NOT NULL,
    model_version VARCHAR(20) NOT NULL,
    precision DECIMAL(5,4),
    recall DECIMAL(5,4),
    f1_score DECIMAL(5,4),
    auc_roc DECIMAL(5,4),
    false_positive_rate DECIMAL(5,4),
    evaluation_date TIMESTAMP DEFAULT NOW()
);
```

Deployment Configuration

Docker Compose

yaml

```yaml
# docker-compose.yml
version: '3.8'

services:
  # Java Services
  transaction-service:
    build: ./transaction-service
    ports:
      - "8081:8081"
    environment:
      - KAFKA_BOOTSTRAP_SERVERS=kafka:9092
      - POSTGRES_URL=jdbc:postgresql://postgres:5432/fraudaml
      - ML_SERVICE_URL=http://ml-service:8000
    depends_on:
      - kafka
      - postgres
      - ml-service

  alert-service:
    build: ./alert-service
    ports:
      - "8082:8082"
    environment:
      - KAFKA_BOOTSTRAP_SERVERS=kafka:9092
      - POSTGRES_URL=jdbc:postgresql://postgres:5432/fraudaml

  # Python ML Service
  ml-service:
    build: ./ml-service
    ports:
      - "8000:8000"
    volumes:
      - ./models:/app/models
    environment:
      - MODEL_PATH=/app/models

  # Infrastructure
  postgres:
    image: postgres:15
    ports:
      - "5432:5432"
    environment:
      - POSTGRES_DB=fraudaml
      - POSTGRES_USER=admin
      - POSTGRES_PASSWORD=secure_password
    volumes:
```

```
    - postgres-data:/var/lib/postgresql/data

  kafka:
    image: confluentinc/cp-kafka:7.5.0
    ports:
      - "9092:9092"
    environment:
      - KAFKA_ZOOKEEPER_CONNECT=zookeeper:2181
      - KAFKA_ADVERTISED_LISTENERS=PLAINTEXT://kafka:9092

  zookeeper:
    image: confluentinc/cp-zookeeper:7.5.0
    environment:
      - ZOOKEEPER_CLIENT_PORT=2181

  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"

  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:8.11.0
    ports:
      - "9200:9200"
    environment:
      - discovery.type=single-node
      - xpack.security.enabled=false

volumes:
  postgres-data:
```

Key Features to Implement

Phase 1: Core Detection

- ✅ Real-time transaction monitoring
- ✅ Rule-based detection
- ✅ ML-based risk scoring
- ✅ Alert generation
- ✅ Basic case management

Phase 2: Advanced ML

- 🔄 Anomaly detection models

- 🔄 Network analysis (graph ML)
- 🔄 Behavioral profiling
- 🔄 Auto-retraining pipeline
- 🔄 Model explainability (SHAP)

Phase 3: AML Compliance

- 📋 Watchlist screening
- 📋 Sanctions checking
- 📋 PEP (Politically Exposed Persons) detection
- 📋 SAR filing automation
- 📋 Regulatory reporting

Phase 4: Advanced Features

- 🚀 Entity resolution
- 🚀 Social network analysis
- 🚀 Real-time streaming ML
- 🚀 Feedback loops
- 🚀 A/B testing for models

Performance Considerations

Scalability Targets

- **Transaction throughput**: 10,000+ TPS
- **Alert latency**: < 500ms
- **ML inference**: < 100ms
- **Dashboard refresh**: Real-time (WebSocket)

Optimization Strategies

1. **Caching**: Redis for customer profiles, risk scores
2. **Batch processing**: Apache Flink for aggregations
3. **Async processing**: Kafka for decoupling
4. **Database partitioning**: Time-series partitioning
5. **ML optimization**: Model quantization, ONNX runtime

Compliance & Regulations

Key Standards

- **FATF** (Financial Action Task Force)
- **Bank Secrecy Act (BSA)**
- **FinCEN** reporting
- **GDPR** (for EU customers)
- **PCI DSS** (payment data)

Audit Trail

- Log every transaction analysis
- Track all alert dispositions
- Record analyst actions
- Maintain model versions
- Store decision rationale

Testing Strategy

```java
```

```java
// TransactionMonitoringServiceTest.java
@SpringBootTest
class TransactionMonitoringServiceTest {

    @Test
    void shouldDetectHighValueTransaction() {
        // Given
        Transaction tx = createTransaction(15000.00);

        // When
        RuleResult result = rulesEngine.evaluate(tx);

        // Then
        assertTrue(result.hasViolation("HIGH_VALUE"));
    }

    @Test
    void shouldDetectRapidFireTransactions() {
        // Given
        List<Transaction> transactions = create5TransactionsIn3Minutes();

        // When
        transactions.forEach(tx -> service.processTransaction(tx));

        // Then
        verify(alertService, times(1))
            .publish(argThat(alert ->
                alert.getViolatedRules().contains("RAPID_FIRE")));
    }

    @Test
    void mlServiceIntegration() {
        // Given
        Transaction tx = createSuspiciousTransaction();

        // When
        MLScore score = mlScoringService.scoreTransaction(tx);

        // Then
        assertThat(score.getFraudScore()).isGreaterThan(0.7);
    }
}
```

Monitoring & Observability

Metrics to Track

```java
// Prometheus metrics
@Timed(value = "transaction.processing.time")
@Counted(value = "alerts.generated")
public void processTransaction(Transaction tx) {
    // ... processing logic
}

// Key metrics:
// - Transactions processed per second
// - Alert generation rate
// - False positive rate
// - Average investigation time
// - Model accuracy over time
// - System latency (p50, p95, p99)
```

Next Steps

1. **Start with MVP**
    - Basic transaction ingestion
    - Simple rules engine
    - Alert management UI
2. **Add ML Gradually**
    - Start with pre-trained models
    - Collect labeled data
    - Train custom models
3. **Iterate Based on Feedback**
    - Tune rules based on false positives
    - Retrain models monthly
    - Add new detection patterns
4. **Scale Infrastructure**
    - Add more Kafka partitions
    - Scale ML service horizontally
    - Implement caching layers

Resources & References

- **AML Regulations**: FATF Recommendations
- **ML Libraries**: Scikit-learn, XGBoost, TensorFlow
- **Java Frameworks**: Spring Boot, Apache Kafka, Drools
- **Model Explainability**: SHAP, LIME
- **Graph Analysis**: Neo4j, NetworkX
- **Testing Data**: Synthetic transaction generators

Good luck building your fraud and AML system! This is a complex but highly rewarding project. Start small, iterate quickly, and always prioritize compliance and accuracy! 🚀 🔒