

# Kyra Works Internship

## Onboarding Task

Submitted by: Bigyan Kumar Piya  
[bigyan.piya@gmail.com](mailto:bigyan.piya@gmail.com)

## Build predictive analytics for theft prevention

Task: Define Basic Log Aggregation Strategy Problem: Outline a strategy for collecting and centralizing logs from different system components. Expected Outcome: Document describing the log aggregation approach. Tools: Text editor. Research: Log management. Consider: Sources, format, storage.

### Objective

To collect, centralize, and standardize logs from multiple system components (e.g., applications, servers, databases, IoT devices, and security systems) to enable predictive analytics for theft prevention.

#### 1. Sources of Logs

Identify where important information comes from:

- Application logs: User behavior, failed login attempts, accesses of sensitive areas.
- System logs: OS-level events, authentication logins, process runs.
- Database logs: Execution of queries, data access patterns.
- IoT/sensor logs: Access control, motion sensors, camera operations.
- Network logs: Firewall, VPN, router traffic, anomaly events.

#### 2. Log Format Standardization

- Use **structured formats** like JSON or CSV to ensure logs are machine-readable.
- Include **essential fields** for predictive analysis:
  - Timestamp (with timezone)
  - Source/component
  - Event type (e.g., login, motion detected)
  - User or device ID
  - Metadata (IP address, location, access point)

#### 3. Log Collection & Transmission

- Use **lightweight agents** installed on source systems to forward logs.
- Transport logs securely (TLS/SSL) to central storage.
- Examples:
  - **Fluentd/Logstash**: For transforming and shipping logs.
  - **Filebeat**: Lightweight shipper for system/application logs.

#### 4. Centralized Storage

- Central log server or cloud storage:
  - Elasticsearch, Splunk, or cloud storage options (AWS S3 + Athena, GCP Logging).
- Make sure indexing and searchability to query historical patterns in a timely manner.
- Use retention policies for historical logs (e.g., 90 days, as per regulations).

#### 5. Preprocessing & Normalization

- Parse logs into a **common schema**.
- Remove redundant or irrelevant fields.
- Anonymize sensitive data (for compliance) if needed.

## 6. Analytics & Alerting

- Feed centralized logs to predictive models to detect anomalies (e.g., unusual access patterns, repeated failed logins, suspicious sensor events).
- Set up **real-time alerting** on suspicious events.

## 7. Security & Compliance

- Ensure logs are **tamper-proof** (write-once storage or checksum verification).
- Restrict access to logs to authorized personnel only.
- Comply with local privacy/data regulations.

## 8. Implementation Tools Overview

Task	Tool/Method
Log collection	Filebeat, Fluentd, Logstash
Storage	Elasticsearch, Splunk, AWS S3, GCP Logging
Processing & Normalization	Logstash, custom Python scripts
Analytics	Python (Pandas, Scikit-learn), MLflow
Alerting	Kibana alerts, Grafana, custom scripts

## Conclusion

A basic log aggregation method is to collect logs from multiple sources, normalize formats, store securely centrally, and preprocess for analysis. The setup accommodates predictive analytics for theft detection where the system is able to detect anomalies and send notifications in advance. Strong indexing, security, and compliance are essential to preserve integrity and use of logs.

## Build model update/deployment automation

Task: Store Feedback Data Problem: Design and implement storage for the alert feedback data. Expected Outcome: Database schema/table for feedback records. Tools: Database. Research: Data modeling. Consider: Linking to original alert, user info.

### Database Schema Design

We need to capture:

- **Original alert** details (linking back to alerts generated by the system).
- **User feedback** (confirmation, false positive, severity, comments).
- **User info** (who gave the feedback).
- **Timestamps** (when feedback was provided).

### Tables

#### 1. alerts

Holds all system-generated alerts.

```
CREATE TABLE alerts (  
  alert_id SERIAL PRIMARY KEY,  
  alert_type VARCHAR(100),  
  alert_message TEXT,  
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP);
```

#### 2. users

Tracks users who can provide feedback.

```
CREATE TABLE users (  
  user_id SERIAL PRIMARY KEY,  
  username VARCHAR(100) UNIQUE NOT NULL,  
  email VARCHAR(150),  
  role VARCHAR(50),  
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP);
```

#### 3. alert\_feedback

Stores feedback on alerts.

```
CREATE TABLE alert_feedback (  
  feedback_id SERIAL PRIMARY KEY,  
  alert_id INT NOT NULL,  
  user_id INT NOT NULL,  
  feedback_status VARCHAR(50) CHECK (feedback_status IN  
(true_positive, false_positive, false_negative, uncertain)),  
  severity_level INT CHECK (severity_level BETWEEN 1 AND 5),  
  comments TEXT,  
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
  
  FOREIGN KEY (alert_id) REFERENCES alerts(alert_id) ON DELETE CASCADE,  
  FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE SET NULL);
```

### Example Workflow

1. System raises an alert → stored in alerts.
2. User reviews alert → stores feedback in alert\_feedback.
3. Later, feedback data can be used for:
  - Measuring **false positive/negative rates**.
  - Training improved **alert classification models**.
  - **Prioritizing retraining** based on severity/uncertainty.

## Example Insert

-- Insert sample alert

```
INSERT INTO alerts (alert_type, alert_message)
VALUES ('Intrusion', 'Suspicious login from unknown IP');
```

-- Insert sample user

```
INSERT INTO users (username, email, role)
VALUES ('alice', 'alice@example.com', 'analyst');
```

-- Insert feedback

```
INSERT INTO alert_feedback (alert_id, user_id, feedback_status, severity_level, comments)
VALUES (1, 1, 'false_positive', 2, 'Login was from a trusted VPN');
```

## Outcome

This schema ensures **feedback data is properly linked** to alerts and users. It supports auditability, enables tracking model performance drift, and provides structured data for retraining models in deployment automation.

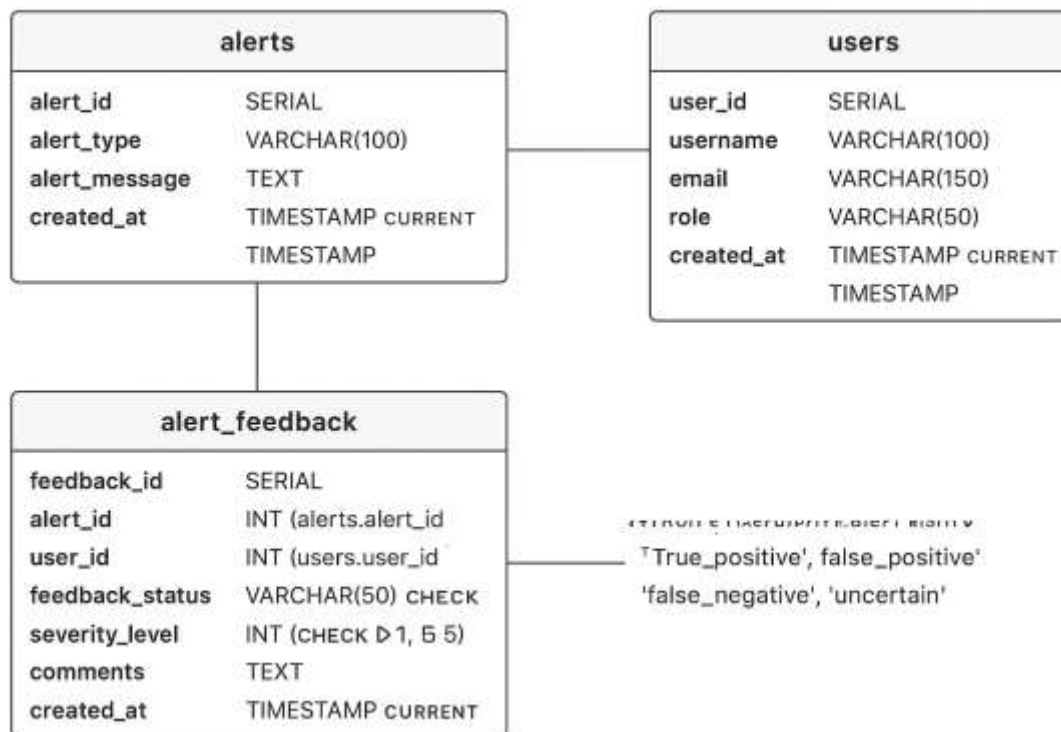


Fig: Database Schema