VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

BACKEND TASK

REAL-TIME NETWORK FREQUENCY ANOMALY DETECTION SYSTEM

NAME: SHUBHAM GOYAL

REG NO: 21BBS0096

1. ABSTRACT

This project involves the development of a real-time network frequency anomaly detection system using a message queue (Kafka) for data ingestion, a MongoDB database for data storage, and machine learning models for anomaly detection. The system leverages Prometheus and Grafana for real-time monitoring and visualizations, providing administrators with up-to-the-minute insights into network operations. The system aims to detect network frequency anomalies and trigger alerts in real-time.

2. TECHNOLOGIES USED

- **Programming Languages**:
  - **Golang** for producer/consumer logic and backend processing.
  - **Python** for machine learning model training and Flask API.
- **Data Ingestion**:
  - **Apache Kafka** as a message broker for real-time data streaming.
- **Database**:
  - **MongoDB** for storing network frequency data.
- **Machine Learning**:
  - **Scikit-Learn** for implementing the Isolation Forest model for anomaly detection.
- **Metrics Monitoring**:
  - **Prometheus** for real-time metrics monitoring.
  - **Grafana** for visualizing these metrics in dashboards.
- **Alerts**:
  - **SMTP** for sending email notifications on anomaly detection.
- **Web API**:
  - **Flask (Python)** to provide an API for real-time anomaly prediction using the trained model.
- **Dashboard**:
  - **Grafana** for real-time visualization of network frequency metrics and anomaly trends using data from Prometheus.

3. DETAILED SYSTEM DESCRIPTION

3.1 Data Ingestion: Kafka Producer

The data ingestion layer simulates network frequency values using a **Golang producer** that generates random frequency values between 47.0 Hz and 53.0 Hz. These frequency values are sent to a Kafka topic named "network_frequency". This producer simulates real-time traffic by sending a message every 0.25 seconds.

Producer Key Components:

- **Kafka**: A message queue to handle high-speed data streaming.
- **Random Frequency Generator**: Generates frequency values simulating real-time network operations.

3.2 Data Preprocessing: Kafka Consumer

The **Golang consumer** listens to the Kafka topic. It:

- **Parses the frequency values** received from Kafka.
- **Stores the data** in MongoDB along with a timestamp for future analysis.
- **Normalizes the frequency values** to a fixed number of decimal places.

3.3 Anomaly Detection and Alerts

The consumer code interacts with a **Flask-based Python API** for anomaly detection. The API uses a pre-trained **Isolation Forest model** to predict whether an incoming frequency value is an anomaly.

If an anomaly is detected:

- **Email Alert**: Sent to administrators via **SMTP**.
- **Prometheus Counter Increment**: A custom Prometheus metric (anomaly_detected_total) increments, signaling a detected anomaly. Additionally, the current frequency is monitored with a separate metric (current_frequency).

3.4 Machine Learning Model

The **Isolation Forest** model, trained using network frequency data from MongoDB, detects anomalies in real-time.

Key model characteristics:

- **Algorithm**: Isolation Forest, effective for anomaly detection.

- **Contamination rate**: Set at 5%, representing the expected proportion of anomalies in the data.

3.5 Logging and Error Handling

- Detailed logs for error handling in Kafka consumption, MongoDB connectivity, and email sending.
- Failed messages or formatting issues are recorded for debugging.

3.6 Real-Time Monitoring with Prometheus and Grafana

To provide real-time monitoring and visualization:

- **Prometheus** collects metrics such as current_frequency and anomaly_detected_total.
- **Grafana** visualizes these metrics in user-friendly dashboards, offering insights into system health and network operations.

Key Dashboards:

  - **Current Frequency Monitoring**: Displays the frequency values over time.
  - **Anomalies Over Time**: Shows detected anomalies and how often they occur.
  - **System Health**: Real-time tracking of key performance indicators like data ingestion rate, API performance, and anomaly detection response time.

4. SYSTEM ARCHITECTURE OVERVIEW

- **Producer**:
  - Simulates network frequency.
  - Sends data to Kafka.
- **Consumer**:
  - Consumes messages from Kafka.
  - Stores data in MongoDB.
  - Calls the anomaly detection API.
  - Sends email alerts on anomaly detection.
- **Anomaly Detection API**:
  - Uses the trained Isolation Forest model.
  - Returns predictions on whether a frequency is anomalous.
- **Prometheus & Grafana**:
  - **Prometheus** gathers system metrics.

- **Grafana** visualizes these metrics in dashboards, providing real-time insights.

5. RESULTS

5.1 Real-Time Anomaly Detection

- The system successfully detects anomalies in real-time with less than 1 second of latency.
- **Prometheus metrics** provide real-time system performance and detection insights.

5.2 Real-Time Monitoring

- The integration of **Grafana** with **Prometheus** provides comprehensive real-time monitoring.
- System administrators can visualize anomalies, frequency variations, and overall system health through user-friendly dashboards.

5.3 Performance of Machine Learning Model

- The **Isolation Forest** model detects anomalies effectively with minimal false positives.

5.4 Email Alerting System

- Email alerts are triggered as soon as an anomaly is detected, with notifications sent within seconds of detection.

6. CONCLUSION

This real-time anomaly detection system combines Kafka for high-speed data ingestion, MongoDB for reliable data storage, and machine learning for accurate anomaly detection. **Prometheus** and **Grafana** provide powerful real-time monitoring and visualization tools, ensuring system operators can track performance and respond to anomalies quickly.

The solution is scalable and adaptable, with the potential to handle larger datasets and more complex machine learning models. The Grafana dashboards offer actionable insights, making the system easy to manage in real-world environments.

******