**Real-Time Reporting System Documentation**

**Table of Contents**

1. **System Overview**
2. **Architecture Design**
3. **ETL Pipeline Components**
4. **Data Storage Strategy**
5. **Performance Optimization**
6. **Monitoring & Maintenance**
7. **Disaster Recovery**
8. **Security Considerations**

**1. System Overview**

**Purpose**

This real-time reporting system is designed to process and analyze data from an e-commerce and delivery platform, providing instant insights and metrics for business intelligence and decision-making. The system allows tracking of key metrics such as order fulfillment, revenue analytics, delivery performance, and user engagement.

**Key Features**

* **Real-time data processing** to provide up-to-date insights.
* **Scalable architecture** to handle growing data volume.
* **High availability** through fault tolerance and load balancing.
* **Data consistency** maintained across the system for accurate reporting.
* **Low-latency reporting** for quick access to insights.

**Business Metrics Tracked**

* **Order Fulfillment Rates**: From the orders and delivery\_tracks tables.
* **Inventory Levels**: From the product\_stocks and warehouse tables.
* **Delivery Performance**: From the delivery\_location and drivers tables.
* **User Engagement**: From the users, orders, and notifications tables.
* **Vendor Performance**: From the vendors, vendor\_payments, and vendor\_followers tables.
* **Revenue Analytics**: From the orders, payments, and refund tables.

**2. Architecture Design**

**High-Level Architecture**

The system follows a **tiered architecture** involving several key components:

1. **Source Database**: The system pulls data from PostgreSQL, the relational database used for transactional data.
2. **Change Data Capture (CDC)**: Debezium is used to capture and stream real-time changes from the PostgreSQL database.
3. **Message Queue**: Apache Kafka ensures reliable message streaming and decouples components of the system.
4. **Stream Processing**: Apache Flink processes the data streams in real-time, applying business logic and transformations.
5. **Storage Layer**: The system utilizes multiple data stores, including Redis (for hot storage), TimescaleDB (for time-series data), and Cassandra (for cold storage).
6. **API Layer**: Exposes processed data through an API for dashboards and reporting interfaces.

**Technologies Used & Reasons**

* **PostgreSQL**: Chosen for its relational model and ACID-compliant properties, which are suitable for transactional data.
* **Debezium**: Used for real-time change data capture from PostgreSQL to stream changes to Kafka.
* **Apache Kafka**: A highly scalable distributed message queue that ensures reliable and fault-tolerant data streaming.
* **Apache Flink**: Chosen for its real-time stream processing capabilities, enabling low-latency analytics and complex event processing.
* **Redis**: Used for caching frequently accessed data, offering low-latency responses for real-time metrics.
* **TimescaleDB**: A PostgreSQL extension optimized for storing time-series data, ideal for real-time metrics such as orders and inventory levels.
* **Cassandra**: A distributed NoSQL database for cold storage, chosen for its ability to handle large volumes of data across many nodes with high availability.

**3. ETL Pipeline Components**

**A. Data Ingestion**

The ETL pipeline uses **Debezium** for capturing changes from PostgreSQL and **Kafka** to stream these changes. Kafka topics are set up to organize data based on the type of information (orders, inventory, users). The ingestion system ensures that the data pipeline remains scalable and fault-tolerant.

**B. Stream Processing**

**Apache Flink** is used to process incoming data streams. It enables real-time transformation and analytics on data, such as calculating delivery times, order amounts, and user engagement metrics. Flink provides robust stream processing capabilities, such as windowing, stateful processing, and event-time handling.

**C. Data Transformation Logic**

As data is ingested, it is transformed to fit the needs of the reporting system. For example, order data is enriched with additional business metrics (e.g., delivery time, processing time) before being stored.

**4. Data Storage Strategy**

**A. Hot Storage (Redis)**

**Redis** serves as the cache layer for frequently accessed data that needs to be available in real-time. It is used to store metrics such as daily orders, vendor performance, and inventory levels. Redis provides extremely low-latency data access, which is crucial for real-time dashboards.

**B. Warm Storage (TimescaleDB)**

**TimescaleDB** is used for time-series data. The orders and transaction data, which are naturally time-dependent, are stored in TimescaleDB. Its ability to efficiently handle time-series data and perform complex queries makes it suitable for metrics like hourly sales, order counts, and delivery performance.

**C. Cold Storage (Cassandra)**

**Cassandra** is used for long-term storage of historical data. It is well-suited for storing large volumes of data that do not need frequent access, such as archived orders or past vendor performance.

**Technologies Used & Reasons**

* **Redis**: Provides high-speed caching, making it suitable for real-time metric queries.
* **TimescaleDB**: Optimized for time-series data, it allows for efficient storage and querying of time-based metrics.
* **Cassandra**: A highly scalable NoSQL database for storing massive amounts of data, chosen for its ability to handle high write throughput and large data sets.

**5. Performance Optimization**

**A. Database Indices**

To optimize query performance, particularly for large datasets, indexes are created on frequently queried fields such as order time, vendor ID, and order status. This ensures faster retrieval of relevant data for reporting.

**B. Query Optimization**

**Materialized views** are used to precompute and store results of expensive queries. For example, hourly sales data can be computed once and then refreshed periodically, reducing the load on the database during peak query times.

**6. Monitoring & Maintenance**

**A. Health Checks**

The system includes health checks to monitor key components like Kafka lag, processing delay, storage health, and API response time. These health checks help ensure the system is running smoothly and that data is processed without significant delays.

**B. Alert Configuration**

Alerts are set up to notify system administrators if any critical thresholds are breached, such as Kafka lag exceeding a certain limit or high error rates in the pipeline. Alerts are sent through channels like email, Slack, or PagerDuty, depending on the severity.

**Technologies Used & Reasons**

* **Prometheus**: For system monitoring and alerting, providing time-series data collection and alert configuration.
* **Grafana**: Used for visualizing metrics, allowing easy dashboard monitoring.

**7. Disaster Recovery**

**A. Backup Strategy**

Daily backups are scheduled for all critical systems: PostgreSQL, Redis, and Kafka configurations. The system uses automated backup jobs to ensure data can be restored in the event of failure.

**B. Recovery Procedures**

If a failure occurs, the system follows a predefined recovery procedure. It ensures data consistency and resumes data ingestion from the last known consistent state. The recovery process includes stopping the data ingestion, restoring data, and verifying the system before resuming operations.

**Technologies Used & Reasons**

* **Backup Solutions (e.g., pgBackRest for PostgreSQL, Redis persistence)**: Ensures data recovery in case of failures.
* **Automation**: Ensures smooth and quick recovery with minimal manual intervention.

**8. Security Considerations**

**A. Data Encryption**

All data is encrypted both at rest and in transit using **AES-256** encryption. This ensures that sensitive information is protected from unauthorized access during storage and transmission.

**B. Access Control**

Role-based access control (RBAC) is implemented to restrict access to sensitive data. Different user roles (e.g., read-only, admin) are defined, and access is granted based on the principle of least privilege.

**Technologies Used & Reasons**

* **TLS (Transport Layer Security)**: Ensures secure communication over networks.
* **AES-256**: A strong encryption algorithm for protecting data at rest.
* **PostgreSQL RBAC**: Ensures that only authorized users have access to specific tables and data.