**Cdc-streams-tasks**

This document outlines the design and implementation of a real-time **Change Data Capture (CDC) streaming platform** that captures, processes, and distributes database changes across multiple systems. The solution enables near real-time data synchronization, event-driven architectures, and real-time analytics by streaming database changes from transactional systems to various downstream consumers including data lakes, data warehouses, and microservices.

The platform addresses the critical business need for low-latency data availability across the organization, replacing traditional batch-based ETL processes with a modern, event-driven approach.

**2. Objectives**

The primary objectives of this CDC streaming platform are:

* **Real-time Data Availability:** To reduce data latency from hours/days to seconds/minutes for critical business data.
* **Decoupled Architecture:** To establish an event-driven architecture that decouples source systems from downstream consumers.
* **Scalable Change Processing:** To build a platform capable of handling high-volume database change events with guaranteed delivery and ordering.
* **Schema Evolution Support:** To implement robust schema management that accommodates database schema changes without breaking downstream consumers.
* **Operational Reliability:** To ensure exactly-once processing semantics and comprehensive monitoring for all CDC streams.

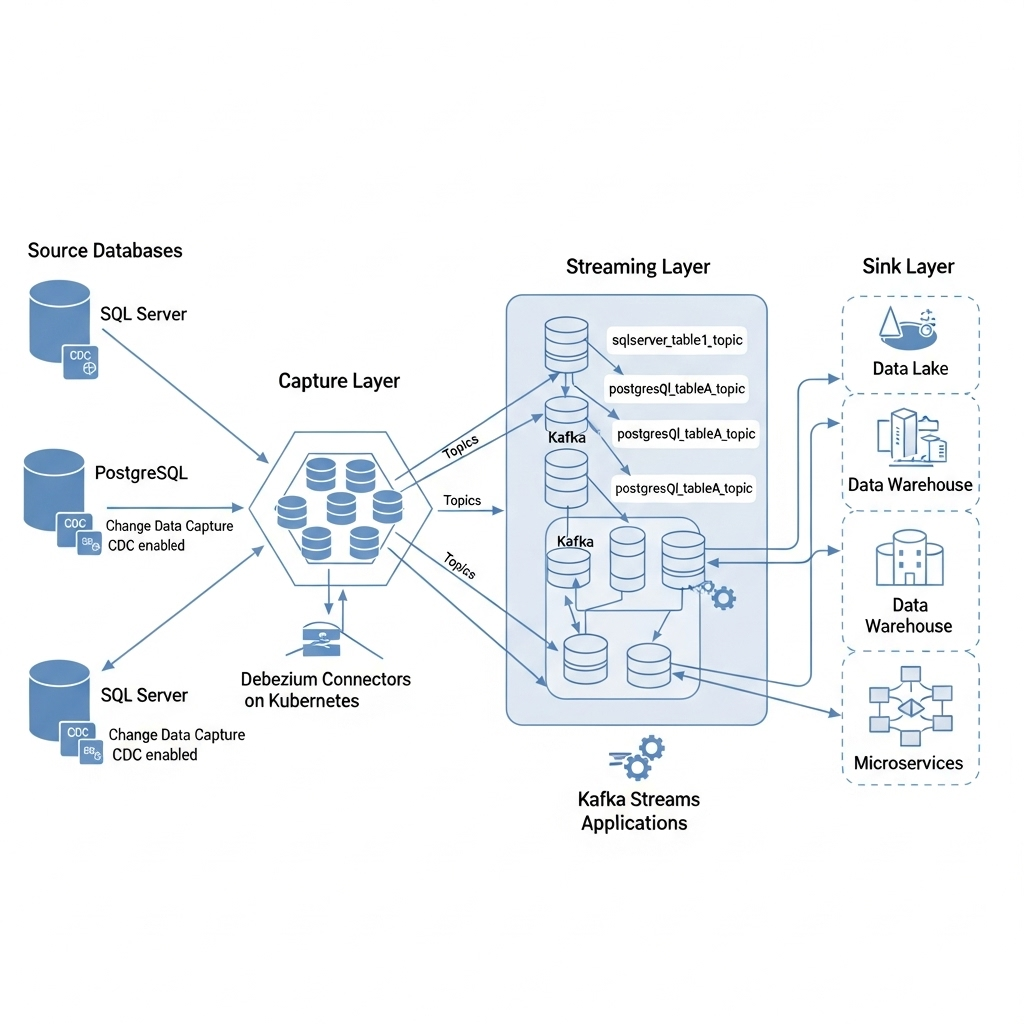
**3. Technology Stack**

| Category | Technology / Service | Justification |
| --- | --- | --- |
| **CDC Capture** | **Debezium** | Open-source CDC platform for capturing database changes with low latency. |
| **Message Broker** | **Apache Kafka** | Distributed event streaming platform for durable, ordered message delivery. |
| **Stream Processing** | **Kafka Streams** | Lightweight library for building real-time applications and microservices. |
| **Storage** | **Azure Data Lake Gen2** | Landing zone for raw CDC events and processed data. |
| **Orchestration** | **Kubernetes (AKS)** | Container orchestration for scalable deployment of CDC connectors. |
| **Monitoring** | **Grafana & Prometheus** | Real-time monitoring and alerting for CDC pipeline health. |
| **Schema Registry** | **Confluent Schema Registry** | Centralized schema management and evolution control. |

**4. System Architecture & Design**

**4.1. High-Level Architecture**

The CDC platform follows a distributed, event-driven architecture with clear separation between change capture, stream processing, and data consumption layers.



**4.2. Data Flow Design**

**4.2.1. Change Capture Flow**

1. **Database Configuration:** CDC enabled on source databases with appropriate retention policies
2. **Debezium Connectors:** Debezium monitors transaction logs and captures changes
3. **Kafka Topics:** Changes published to Kafka topics following naming convention: {server}.{database}.{table}

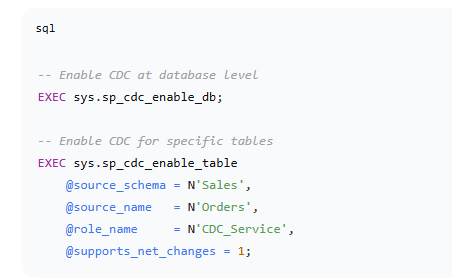
**4.2.2. Stream Processing Flow**

1. **Raw CDC Events:** Initial capture of change events in Debezium format
2. **Schema Validation:** Schema registry validates and manages schema evolution
3. **Event Transformation:** Kafka Streams applications transform and enrich events
4. **Multi-destination Routing:** Processed events routed to appropriate sinks

**5. Implementation**

**5.1. Source Database Configuration**

**5.1.1. SQL Server CDC Setup**



**5.1.2. Debezium Connector Configuration**

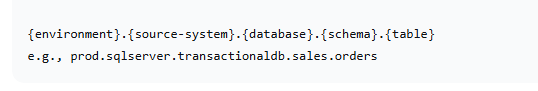
**5.2. Kafka Topics & Partitioning Strategy**

**5.2.1. Topic Configuration**

* **Retention Policy:** 7 days for raw CDC topics
* **Replication Factor:** 3 for production resilience
* **Partition Strategy:** Partitioned by primary key for ordering guarantees

**5.2.2. Topic Naming Convention**

text

**5.3. Stream Processing Implementation**

**5.3.1. Kafka Streams Application for Data Enrichment**

java

**6. Data Quality & Reliability**

**6.1. Exactly-Once Processing**

* **Kafka Transactions:** Enabled for exactly-once semantics
* **Idempotent Producers:** Prevent duplicate message production
* **Consumer Offsets:** Atomic commitment of processing results with offset updates

**6.2. Schema Evolution & Compatibility**

* **Forward Compatibility:** New fields can be added without breaking consumers
* **Backward Compatibility:** Consumers can read data produced by older schemas
* **Schema Validation:** All messages validated against registered schemas

**6.3. Monitoring & Alerting**

**6.3.1. Key Metrics Monitored**

* **Consumer Lag:** Difference between latest and consumed offsets
* **Connector Health:** Debezium connector status and error rates
* **Processing Latency:** End-to-end latency from source change to consumer delivery
* **Error Rates:** Failed processing and delivery attempts

**7. Testing & Validation**

**7.1. Testing Strategy**

**7.1.1. Performance Testing**

* **Throughput Validation:** Verified handling of 10,000+ events per second
* **Latency Testing:** Confirmed sub-5-second end-to-end latency
* **Load Testing:** Sustained performance under peak database activity

**7.1.2. Reliability Testing**

* **Failover Testing:** Validated connector recovery after pod restarts
* **Network Partition:** Tested behavior during network disruptions
* **Schema Evolution:** Verified compatibility with backward-incompatible changes