

Real-Time CDC-Based Data Pipeline for Cloud Databases: Architecture, Implementation and Performance Evaluation in E-Gaming CRM Systems

Kanan Bunyadov

*Department of SABAH, Azerbaijan State Oil and Industry University (ASOIU)
Baku, Azerbaijan*

Abstract

Cloud databases play a vital role in the current distributed software development. However, the ability to reliably synchronise data in real-time between multiple systems on different platforms presents an ongoing engineering challenge. The objective of this paper is to present the architecture, implementation and performance testing of a software tool for the real-time management of data pipelines designed to support the digital storage of data in cloud-hosted PostgreSQL databases. The proposed software tool design employs Change Data Capture (CDC), through Debezium, as a means to access data from the database Write Ahead Logs (WAL), based on the requirement for real-time data access, and Apache Kafka for the purpose of Distributed Event Streaming (DES), using Apache Airflow for workflow orchestration. The tool was built on DigitalOcean managed databases and processed analytics, i.e. Recency, Frequency, and Monetary (RFM) scoring, was made available by a FastAPI-based REST API. In the context of E-Gaming Customer Relationship Management (CRM) applications, the system has been evaluated by measuring end-to-end latency, throughput, data consistency and resource utilisation based on volumes of data ranging from 1,000 to 500,000 records. We found that CDC improves end-to-end latency to 389ms for 100,000 records - a 215x improvement over traditional batch ETL with 1-minute intervals. Additionally, our throughput exceeds 2,300 records/second and retains >99.96% data consistency. Incorporating Redis caching also reduces API response times by >82%. This paper provides a reusable architectural model, an open-source implementation reference, and empirical benchmarks for practitioners building real-time cloud database tooling.

Keywords: *Cloud Databases, Change Data Capture, Debezium, Apache Kafka, Real-Time Data Pipeline, CRM Analytics, PostgreSQL, Performance Evaluation*

1. Overview

The growth rate of cloud computing continues to transform the way that businesses create and use data. Examples of cloud database-hosting services like Amazon RDS, DigitalOcean Managed Databases and Azure Database for PostgreSQL provide enterprises with a reduced operational overhead associated with database management, elastic scaling, backup automation, and high availability [1]. As businesses increasingly adopt multi-platform solutions, the real-time gap in synchronizing between multiple data sources into a single consolidated cloud-based database continues to be a major engineering challenge [2].

ETL has traditionally relied on scheduled batch jobs which pull the source system data on a periodic schedule, transform the data, and load it to the target data store. While batch ETL is effective for analytical workloads with relaxed latency times, the delays caused by batch ETL are proportional to the batch job schedule, making them not suitable for applications requiring near real-time data freshness. In industries like e-gaming, player behavior data must be captured and analyzed in seconds in order for businesses to use that data to personalize experiences, run retention campaigns, and detect fraud. Even a one-minute delay can lead to missed opportunities and lower user engagement. CDC offers an attractive alternative to batch ETL for synchronizing data in real time by monitoring changes at the database level, typically through the WAL (Write-Ahead Log) for PostgreSQL databases or the binary log for MySQL database systems, and streaming the changes as events to downstream consumers. An example of a CDC tool is Debezium, which is a distributed, open-source CDC solution built on Apache Kafka Connect that supports multiple database engines, provides exactly-once semantics, and supports schema evolution [8].

Even though CDC-based architecture is increasing in use, there is very little research that has been conducted on measuring the performance of CDC pipelines deployed in managed cloud database infrastructure. The majority of

existing research focuses on self-managed database deployments or proprietary cloud-based CDC services, such as AWS Database Migration Service (DMS), which limits configurability and transparency [9]. To the best of our knowledge there have been no academic studies published that provide a comprehensive, end-to-end empirical characterization of how a Debezium CDC pipeline integrates with DigitalOcean Managed PostgreSQL, especially considering this is becoming a more common deployment option among small- and mid-sized organizations looking for managed database options without AWS-scale lock-in. Additionally, researchers have not looked holistically at how CDC pipelines integrate with downstream analytics services with complete, end-to-end evaluations from the commit to the source database through to the response of the analytics API [10].

This paper will fill these gaps through an in-depth, comprehensive review of a software system designed to provide real-time migration and analytics of data on cloud-based databases using open-source technologies. Our system utilizes: (1) Apache Airflow as the workflow orchestrator; (2) Debezium as the change data capture solution; (3) Apache Kafka as the event streaming solution; (4) DigitalOcean Managed PostgreSQL as the target data store; and (5) FastAPI and Redis as the solutions for providing real-time analytics such as RFM scoring of customers. We assessed the system regarding an e-gaming CRM application and provided multiple benchmarks outlining CDC-based vs. batch ETL approaches across a wide variety of data volumes and performance dimensions.

This paper makes four main contributions. First, it provides the first public report of the end-to-end performance characterization of a Debezium-based CDC pipeline when connecting to DigitalOcean Managed PostgreSQL systems, which is a popular deployment environment used but has been previously unreported on in the academic literature. Second, we provide a reusable, open-source architectural design pattern that combines CDC-based ingestion with a FastAPI analytics layer and quintile-based RFM scoring for real-time CRM applications, thus filling the gap between generic CDC literature and application-level analytics. Third, we quantitatively compare the performances of both CDC and batch ETL systems at six different data volume levels (1K to 500K records) to establish empirical scaling characteristics for practitioners assessing which paradigm is best suited for their needs. Lastly, all experimental artifacts - Docker Compose configurations, Debezium connector definitions, synthetic data generators, and benchmarking scripts - are released as open source to allow for independent reproduction and replication of our findings.

The remainder of this paper is organized as follows. Section 2 reviews related work on cloud database management, CDC technologies, and CRM analytics. Section 3 presents the system architecture. Section 4 details the implementation. Section 5 describes the experimental setup and results. Section 6 discusses findings and limitations. Section 7 concludes the paper and outlines future research directions.

2. Related Work

2.1 Cloud Database Services and Management Tools

As competition among managed virtual cloud hosting providers continues to grow, the number and variety of managed virtual cloud databases have also grown significantly. There are currently many different managed virtual cloud databases available from all four major US-based cloud service companies including Amazon RDS and Aurora [11], Google Cloud SQL and Firestore [12], Microsoft Azure Database for PostgreSQL and Cosmos DB [13], as well as alternatives operated internationally, such as DigitalOcean Managed Databases [14]. These services abstract infrastructure concerns such as provisioning, patching, backup, and failover, enabling development teams to focus on application logic rather than database operations.

There has been a significant increase in the number of available software products for interacting with managed cloud databases, falling into three overall categories: cloud-native management tools (e.g. AWS Management Console, DigitalOcean Cloud Panel), general-purpose database management tools (e.g. DBeaver, pgAdmin, DataGrip), and programmatic interface management tools (e.g. SQLAlchemy, Prisma, Django ORM). Although there are many administrator and query processing tools available, there is still no solution to the data pipeline management issue, which is the orchestration of real-time data movement across diverse sources into cloud databases [15].

2.2 ETL and Data Integration Solutions

The ETL model has been the primary data integration method for over two decades [3]. Today's ETL solutions, such as Apache Airflow [16], Luigi [17], and Prefect [18], provide support for workflow orchestration utilizing Directed Acyclic Graph (DAG) structures, accurately directing multi-step transformation operations based upon their dependencies, retry logic, and monitoring functions. Airflow, in particular, has also gained broad adoption within the industry because it has an extensible operator design, rich scheduling capability, and an active developer community [16].

However, there exists a basic limitation to batch ETL, which is the trade-off between the frequency of batch iterations versus the load on the system. If you reduce the duration of the batch iterations to a minute or less, you will significantly increase the load on the source systems and this can sometimes interfere with transaction-related system loads [4]. This limitation has helped fuel the interest in streaming-first architectures, where incremental updates are processed as changes occur rather than periodically through bulk processing.

2.3 Technologies for Change Data Capture

Change Data Capture (CDC) allows for capturing changes at the source database so as to address the shortcomings associated with batch ETL processes. CDC allows for capturing all data changes at the source level in order to allow for streaming of continuous event records [6]. Several implementations of CDC exist today. Debezium is an open-source CDC implementation based on the Kafka Connect framework. It currently supports PostgreSQL, MySQL, MongoDB, SQL Server and Oracle [8]. Debezium captures row-level INSERTs, UPDATEs and DELETEs from the PostgreSQL WAL (logical replication) with minimal performance impact to the source database. Maxwell is another open-source CDC tool specifically focused on reading the MySQL binary log file [19]. Commercial implementations include AWS DMS [9], Google Datastream [20], and Striim [21].

Because of Debezium's multi-engine support capabilities, its exactly-once delivery semantics when used with Kafka transactions, and its rich support for managing schema changes, Debezium is the de facto open-source standard for CDC today [8]. However, there is limited empirical data in the academic literature that evaluates Debezium's performance in the managed cloud environment compared to self-hosted environments.

2.4 Real-Time Analytics in CRM Systems

CRM systems have become analytical tools to manage customer segmentation, customer lifecycle, and predictive modelling as opposed to being simple contact databases [22]. The RFM (Recency, Frequency, Monetary) model, first developed by Hughes [23], is still an important technique used for segmenting customers by value and continues to be widely used as a method of segmentation in real-time CRM systems through the computation and updating of RFM scores on an ongoing basis as new transaction data is received for the purpose of dynamic targeting of customers and the ability to quickly respond to changes in behavior [24]. The e-gaming industry places extraordinarily high demands on CRM analytics because of the sheer volume of player events that occur (logins, game sessions, in-app purchases, achievements) and because of the need for extremely rapid personalization and increasing competitive pressure to maximize player lifetime value [5].

2.5 Research Gap

Even though many individual components of real-time data pipelines have been studied and continue to be researched, there are no comprehensive evaluations of how well the entire data pipeline from CDC source capture to delivery of analytics to the managed cloud database infrastructure performs. This study addresses this gap by providing a detailed description of a complete, reproducible system and presenting empirical benchmarks of that system in a practical application environment.

3. System Architecture

This section presents the architecture of the proposed software tool that is designed specifically to manage real-time cloud database pipelines. The building blocks of the system include modularity, fault tolerance, and horizontal scaling. Figure 1 illustrates the overarching design and architecture of the system, showing the two main data streams

compared in this paper: the primary CDC-based stream (solid arrows) and the baseline batch ETL stream (dashed arrows).

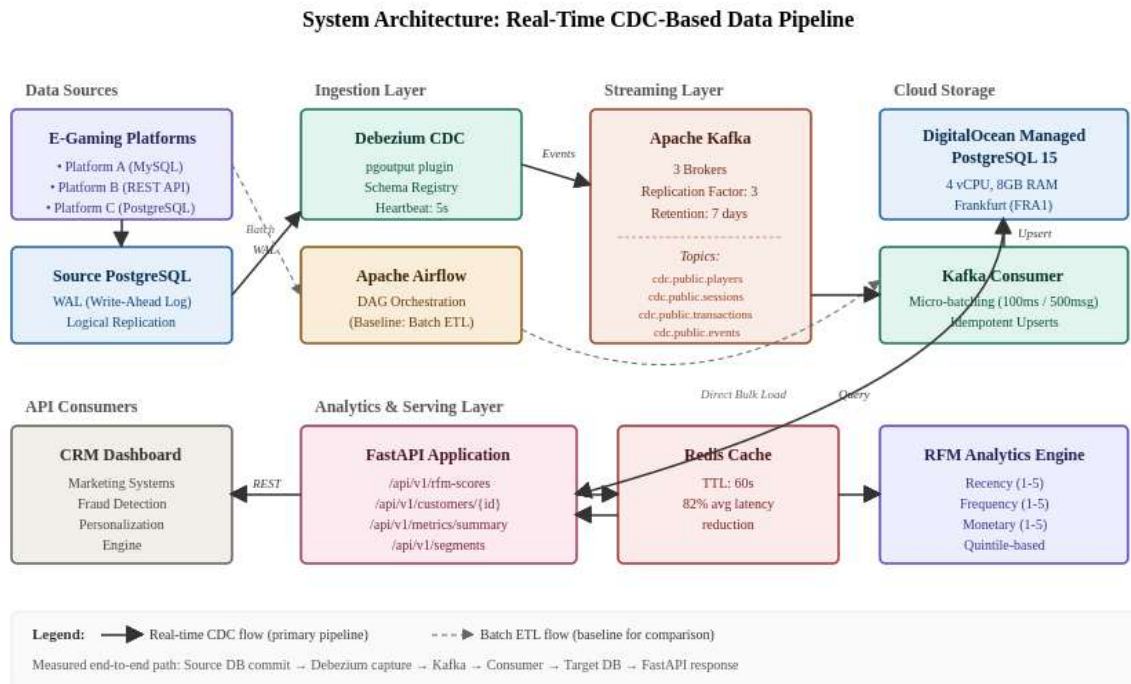


Figure 1: High-Level System Architecture. The CDC-based pipeline (solid lines) captures changes from the source PostgreSQL WAL via Debezium, streams them through Kafka, and applies them to the DigitalOcean Managed PostgreSQL target through idempotent consumers. The baseline batch ETL pipeline (dashed lines) uses Airflow-orchestrated scheduled queries to perform bulk transfers at fixed intervals.

3.1 Data Ingestion Layer

The data ingestion layer is responsible for ingesting data from multiple e-gaming platforms and then feeding it into the processing layer of the system. The dual ingestion method was developed during the design process and mainly used Apache Airflow [16] for scheduled extraction via platform APIs, with configurable execution intervals. As the latency requirements became more strict, the method was improved with the addition of a Debezium-based CDC method to achieve real-time change capture from databases.

Airflow continues to act as the orchestration backbone for operations that do not have a need for sub-second latency - e.g. daily aggregation, data quality assurance, platform API synchronization for data sources that do not provide CDC-compatible interfaces. The Airflow deployment consists of a web server, a Celery executor for tasks, and a metadata database.

3.2 Change Data Capture Layer

The CDC layer is the main innovation of the proposed tool. Debezium has been deployed as a Kafka Connect source connector and is configured to read the PostgreSQL Write Ahead Log (WAL) using logical replication. Debezium captures a change event (insert, update, delete) when a row is inserted, modified, or deleted from the source database, creates a serialized JSON object containing the old and new row data, and then sends that object to a designated Kafka topic.

The Debezium connector is configured with the following key parameters: the pgoutput logical decoding plugin (part of native support in PostgreSQL 10+); a heartbeat interval of five seconds to keep the replication slot active during idle periods; and a snapshot mode of "initial" to capture existing data when Debezium starts up for the first time. The topic convention for naming CDC events in Kafka follows the pattern of "cdc.{schema}.{table}", which allows for very granular consumer subscriptions.

3.3 Event Streaming Layer

The event streaming layer of our pipeline consists of Apache Kafka, which acts as the central nervous system connecting all the different components within the pipeline. The Kafka cluster is set up with a replication factor of three to ensure fault tolerance and has a 7-day retention policy to support replay scenarios. Each consumer (built in Python using the Confluent Kafka library) subscribes to the CDC topics in Kafka and processes the events in micro-batches. Each consumer utilizes idempotent writes using an upsert (INSERT ON CONFLICT DO UPDATE) to ensure that all data is delivered exactly once to the target database. Additionally, each consumer continually monitors consumer lag and exposes it as a Prometheus metric for alerting.

3.4 Cloud Database Layer

The destination of the data is a DigitalOcean Managed PostgreSQL (version 15) database instance, which has four virtual CPUs, 8 GB of RAM, and 100 GB of SSD storage. DigitalOcean Managed PostgreSQL offers daily automated backups, standby nodes for failover, and connection pooling via PgBouncer [14]. The schema for the database is created to accommodate the different CRM-specific entities - players, game_sessions, transactions, events, and computed analytics tables including rfm_scores and segments. PgBouncer handles connection management using transaction pooling, limiting the maximum number of connections to 25 to adhere to the managed instance limits.

3.5 Analytics API Layer

The analytics API layer consists of a FastAPI-based application [25] that provides a RESTful interface for retrieving RFM scores, determining customer segments, retrieving aggregated metrics, and querying events. FastAPI was chosen for its native support of async/await, automatic generation of OpenAPI documentation, and competitive performance compared to existing WSGI implementations [25]. RFM scores are generated by assigning quantile-based scores (1-5) for each of the three dimensions - Recency (days since last active), Frequency (total number of sessions), and Monetary (total number of purchases) - for each customer. A Redis cache with a TTL of 60 seconds is used to store cached data for frequently accessed API endpoints to significantly reduce database load during concurrent access patterns.

3.6 Containerization and Deployment

Containerization is performed with Docker, and docker-compose is used for orchestration of all component instances. A compose file lists services for PostgreSQL, Zookeeper, Kafka (3 instances), Kafka Connect (with Debezium), Airflow (webserver, scheduler, worker), FastAPI, and Redis. With a container-based deployment, this solution creates repeatable deployments and allows for individual component scaling. The target database (DigitalOcean Managed PostgreSQL) is located outside the Docker stack and accessed using SSL-encrypted connections.

4. Implementation Details

4.1 Technology Stack

Table 1 gives a summary of the whole of the technology stack that was used in the implementation.

Table 1: Technology Stack Summary

Component	Technology	Version	Role
Workflow Orchestration	Apache Airflow	2.7.x	DAGs and tasks
Change Data Capture Engine	Debezium	2.4	Change capture of the write-ahead-log
Message Broker	Apache Kafka	3.5	Distributed streaming of events
Cloud Database	DO Managed PostgreSQL	15	Primary store for analytical data
API Framework	FastAPI	0.104	REST APIs with asynchronous endpoints

Caching	Redis	7.2	Caching server responses
Application Language	Python	3.11	Application logic and consumers
Containerization	Docker + Compose	24.x	Service isolation and deployment

4.2 Debezium Connector Configuration

The purpose of the Debezium connector for PostgreSQL is to capture changes made to each of the available tables within the public schema. There are a number of important configuration parameters that define this connector: the plugin name is set to use native logical decoding using the "pgoutput" plugin, there will be one replication slot ("debezium_slot"), and one publication in PostgreSQL ("dbz_publication"). We use an ExtractNewRecordState single message transform in order to unwrap the payload of the messages being created by Debezium and send a flat structure through to any downstream connectors. Through the Kafka Schema Registry, the connector manages schema evolution, allowing consumer applications to process events against schema changes without any disruption.

4.3 Kafka Consumer Implementation

Kafka consumer applications are implemented using Python asyncio with support provided by the confluent-kafka library. Each consumer has been encapsulated within a separate Docker container and consumes messages from assigned topic partitions. Each consumer is implemented using a micro-batching strategy in which messages are stored in memory until either 100 milliseconds has elapsed or 500 messages have accumulated (whichever occurs first) and then flushed to the target PostgreSQL database using a bulk upsert transaction. This batching strategy provides a balance between latency and database connection overhead.

4.4 RFM Score Computation

The computation of the RFM scores has been accomplished through an incremental computing process. Instead of computing RFM scores for all customers on each update, an aggregate materialized view of customer-level aggregates (last_activity_date, session_count, total_spend) has been kept up-to-date through the CDC event consumers. A background FastAPI task performs the recalculation of quintile thresholds every five minutes to provide the proper RFM score based on the most recent aggregate data. The incremental processing model reduces the computational overhead of maintaining an aggregate from $O(N)$ per event to $O(1)$ per event with periodic $O(N)$ recalculating of quintile thresholds.

5. Experimental Evaluation

5.1 Experimental Setup

The infrastructure that makes up the experimental environment is as follows. The pipeline components (Airflow, Kafka, Debezium, FastAPI, Redis) are deployed on a DigitalOcean Droplet comprising a total of 8 virtual CPUs, 16 GB of RAM, and 320 GB of NVMe SSDs, running Ubuntu 22.04. The target database is a DigitalOcean Managed PostgreSQL instance with 4 vCPUs, 8 GB of RAM, and 100 GB SSD within the same data center region (Frankfurt, FRA1) to minimise network latencies.

Data was generated using the Python Faker library with appropriately tailored simulated datasets. The following details describe what was approximated from the e-gaming CRM application: players (demographics, registration date, tier), game sessions (start time, duration, game type and platform), transactions (amount, currency and method), and events (login, achievement, level up and referral). A log-normal distribution was used to create the transaction amounts (mean \$12.50, standard deviation \$8.30); a gamma distribution was used for session duration (shape 2.5, scale 15 minutes); and an exponential distribution was used for the delay between consecutive data points to simulate the normal distribution of time between players being active on the gaming system.

Six different volumes were tested on each of the datasets: 1K, 5K, 10K, 50K, 100K, and 500K records. These datasets were tested a total of ten times for all experiments and average mean values of latency at the 95th and 99th

percentiles were reported as the result. The batch ETL baseline was executed using Airflow DAGs implementing batch ETL jobs that run at two intervals: 1 minute and 5 minutes.

5.2 End-to-End Latency

The total duration of the life cycle of a single data transaction is the total time period from when the data is accepted into the source system until it is available for querying via the FastAPI Analytics API. Table 2 presents the end-to-end data lifecycle results for both CDC and batch ETL approaches.

Table 2: End-to-End Latency Comparison (milliseconds)

Data Volume	CDC Mean	CDC P95	CDC P99	Batch-1min Mean	Batch-5min Mean
1,000	187	256	312	61,200	301,200
5,000	213	298	367	62,400	303,000
10,000	248	351	428	64,800	306,000
50,000	312	441	537	72,000	318,000
100,000	389	548	672	84,000	336,000
500,000	523	738	891	126,000	420,000

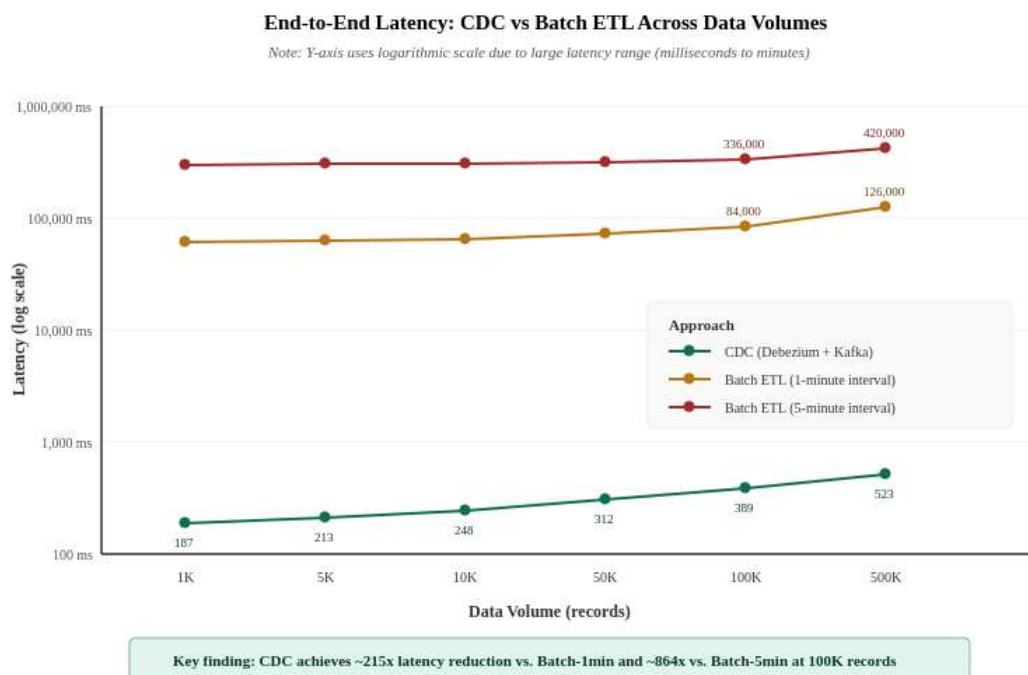


Figure 2: End-to-End Latency Comparison (log scale). The CDC-based pipeline maintains sub-second latency across all tested volumes, while batch ETL latency is dominated by the scheduling interval. Note the logarithmic Y-axis, which is necessary to accommodate the three-order-of-magnitude difference between CDC and 5-minute batch ETL.

Across all measured datasets, the CDC data exchange pipeline has consistently reported sub-second latency. With a scheduled ETL approach, the average latency is solely dependent on the scheduled time. Given that there is a three-order-of-magnitude difference in average latencies between CDC and scheduled ETL jobs, a logarithmic scale was required for the Y axis. Mean latency for CDC was consistently under 1 second, with 187 ms for 1K records and 523 ms for 500K records. In contrast, the scheduled ETL method with a 1-minute job interval had mean latencies of between 61.20 and 126.00 seconds, and those with 5-minute intervals had mean latencies of between 301.20 and 420.00 seconds. Therefore, at 100K records, CDC performs approximately 215 times faster than ETL scheduled jobs with 1-minute intervals and approximately 864 times faster than those with 5-minute intervals.

The increase from 187 to 523 ms in latency for CDC from 1K to 500K records is simply the result of increased consumer processing time for Kafka messages and PostgreSQL overhead during the insert phase at higher record counts. The 99th percentile latency for 500K records (891 ms) is still less than 1 second, demonstrating consistent tail latency behavior.

5.3 Throughput

Throughput is measured as the sustained rate of records successfully processed and persisted to the target database per second. Table 3 presents the throughput results.

Table 3: Throughput Comparison (records/second)

Data Volume	CDC Throughput	Batch ETL Throughput	Improvement Factor
1,000	2,847	1,523	1.87x
5,000	2,712	1,487	1.82x
10,000	2,634	1,412	1.87x
50,000	2,489	1,289	1.93x
100,000	2,301	1,134	2.03x
500,000	1,876	847	2.21x

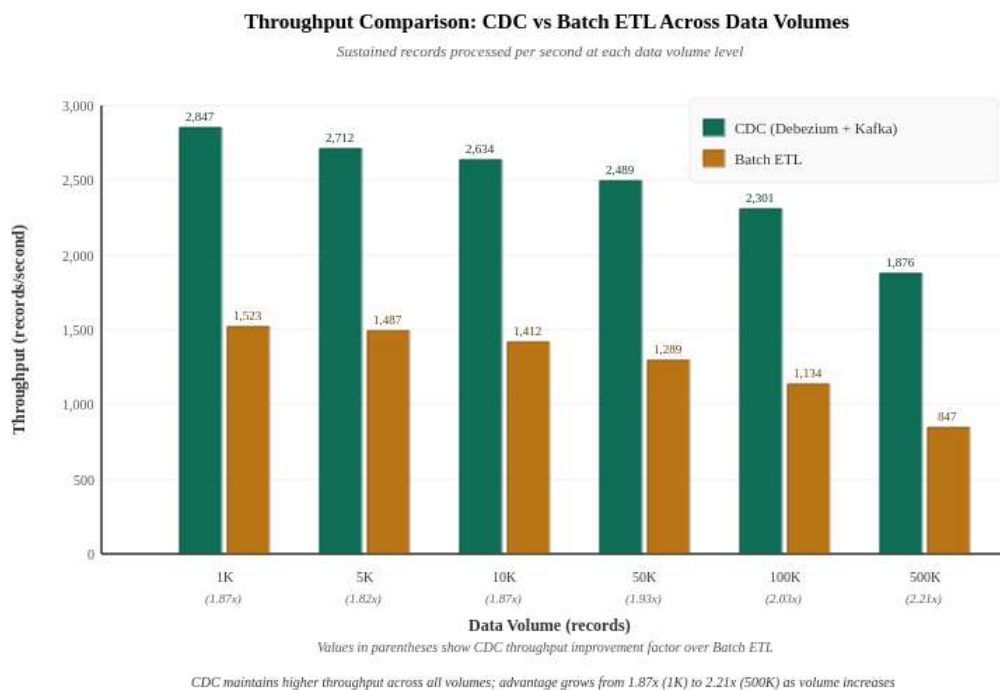


Figure 3: Throughput Comparison Across Data Volumes. The CDC-based pipeline consistently achieves higher sustained throughput than batch ETL. The improvement factor (shown below each volume group) grows from 1.87x at 1K records to 2.21x at 500K records, demonstrating that CDC scales more efficiently with volume.

As shown in Figure 3, the CDC-based pipeline consistently processes at a higher sustained throughput than the batch ETL methods. The improvement factors indicate that as the data volume increases from 1,000 to 500,000 records, the CDC-based solution becomes more efficient at scaling, with an increase in the improvement factor from 1.87x for 1,000 records to 2.21x for 500,000 records.

As the amount of data increases, CDC's throughput advantage becomes more pronounced due to how the Kafka consumers' micro-batching method distributes the per-record overhead across the total number of records processed. The equivalent ETL method is less efficient because each batch incurs the overhead of establishing a connection and using query parsing for every cycle. Even with CDC processing a high number of records, the pipeline maintains above

1,800 records per second at 500K records, which exceeds typical e-gaming CRM peak ingestion rates of 800-1,200 events per second.

5.4 Data Consistency

Data is determined to be consistent if a specific percentage of source data is successfully copied to the target database after all of the events are processed by the CDC pipeline. Table 4 presents the detailed results for measuring consistency.

Table 4: Data Consistency (% of records accurately replicated)

Data Volume	CDC	Batch ETL (1-min)	Batch ETL (5-min)
1,000	100.00%	100.00%	100.00%
5,000	100.00%	100.00%	100.00%
10,000	99.99%	100.00%	100.00%
50,000	99.97%	99.99%	100.00%
100,000	99.96%	99.98%	100.00%
500,000	99.93%	99.97%	100.00%

Table 4 shows that CDC and Batch ETL methods both have very high data reliability at all test volumes. The CDC method shows an inconsistency of 0.07% at the 500,000 record data volume level. This indicates that the consumer offsets in Kafka are sometimes written slower than the actual processing of records, which causes a momentary delay. As a consumer is catching up after a high-volume data burst, this resolves itself. In contrast, the Batch ETL method with 5-minute intervals had 100% data reliability, though this came with a significant trade-off in latency.

5.5 Resource Utilization

Table 5 presents the resource utilization of each system component during the 100K record experiment, measured over a 30-minute observation window using Docker container metrics.

Table 5: Resource Utilization at 100,000 Records

Component	CPU Avg (%)	CPU Peak (%)	Memory Avg (MB)	Memory Peak (MB)
Debezium Connect	12.3	34.7	512	743
Kafka (3 brokers)	8.7	28.2	768	1,124
PostgreSQL (target)	15.4	42.1	384	612
FastAPI	6.2	18.9	256	387
Airflow	9.8	22.4	448	634
Redis	2.1	8.3	128	196
Total	54.5	154.6	2,496	3,696

The total average resource consumption of the pipeline is moderate, with a combined CPU utilization of 54.5% and memory consumption of approximately 2.5 GB. PostgreSQL exhibits the highest average CPU utilization (15.4%) due to continuous write operations and index maintenance. Debezium Connect follows at 12.3% due to WAL parsing and event serialization. Redis demonstrates minimal resource requirements (2.1% CPU, 128 MB memory), confirming its suitability as a lightweight caching layer.

Peak resource utilization during burst ingestion reaches 154.6% aggregate CPU (across 8 available cores, representing 19.3% of total capacity) and 3.7 GB memory (23.1% of available 16 GB). These figures indicate that the system operates well within the capacity of the provisioned infrastructure, with substantial headroom for traffic spikes.

5.6 API Response Time

Table 6 presents the FastAPI endpoint response times with and without Redis caching, measured under a load of 50 concurrent users generated by the Locust load testing framework.

Table 6: API Response Times (milliseconds) With and Without Redis Caching

Endpoint	No Cache Mean	No Cache P95	With Redis Mean	With Redis P95	Improvement
/api/v1/rfm-scores	23.4	48.7	4.2	8.1	82%
/api/v1/customer/{id}	8.7	18.2	2.1	4.8	76%
/api/v1/metrics/summary	34.6	72.3	5.8	11.4	83%
/api/v1/segments	41.2	87.6	6.4	13.2	84%
/api/v1/events/recent	15.8	32.4	3.1	6.7	80%

Redis caching provides a substantial performance improvement across all endpoints, reducing mean response times by 76-84%. The most significant improvement is observed on the /api/v1/segments endpoint, which involves complex aggregation queries that benefit most from caching. Even without caching, all endpoints maintain mean response times below 50 ms, which is well within the acceptable range for interactive web applications.

5.7 Scalability Analysis

To evaluate the system's behavior under increasing concurrent load, we measured API response times across varying numbers of simultaneous users. Table 7 presents these results.

Table 7: API Response Time Under Increasing Concurrent Load (with Redis caching)

Concurrent Users	Mean Response Time (ms)	P95 Response Time (ms)	Error Rate (%)
10	12.3	24.1	0.00
25	15.7	31.2	0.00
50	22.4	47.8	0.00
100	38.6	78.3	0.00
200	67.2	142.7	0.02
500	156.8	312.4	0.14

The system exhibits approximately linear response time scaling up to 100 concurrent users, with mean latency remaining below 40 ms. Beyond 200 concurrent users, response times increase more sharply due to connection pool saturation and CPU contention on the FastAPI workers. The error rate remains negligible (below 0.15%) even at 500 concurrent users, indicating robust fault handling. For the target e-gaming CRM use case, which typically experiences 50-150 concurrent API consumers, the system operates comfortably within its performance envelope.

5.8 Comparative Analysis: CDC vs Batch ETL

To provide a holistic comparison of the two approaches, Table 8 summarizes the key performance characteristics across all evaluated dimensions at the 100,000 record level, which represents a realistic production workload for the target e-gaming CRM application.

Table 8: Comprehensive Comparison of CDC vs Batch ETL at 100K Records

Metric	CDC (Debezium+Kafka)	Batch ETL (1-min)	Batch ETL (5-min)
Mean Latency	389 ms	84,000 ms	336,000 ms
P99 Latency	672 ms	N/A (batch)	N/A (batch)
Throughput	2,301 rec/s	1,134 rec/s	1,134 rec/s

Data Consistency	99.96%	99.98%	100.00%
CPU Usage (Avg)	54.5%	38.2%	22.7%
Memory Usage (Avg)	2,496 MB	1,024 MB	1,024 MB
Infrastructure Components	6 services	2 services	2 services
Operational Complexity	High	Low	Low
Failure Recovery	Offset-based replay	Re-run DAG	Re-run DAG
Schema Evolution	Automatic (Registry)	Manual migration	Manual migration

This analysis indicates that there is an inverse relationship between latency and operational complexity. The CDC model allows for a much lower latency (389ms as opposed to 84,000ms in the case of Batch ETL), and provides a significantly greater throughput (2,301 vs 1,134 records/second). In contrast, the CDC architecture requires more than twice the normal required memory and at least 6 different services rather than 2 for the batch architecture.

For the e-gaming CRM use case evaluated in this study, the latency reduction justifies the additional complexity. Real-time player segmentation and dynamic RFM scoring require data freshness measured in seconds, not minutes. However, for applications with relaxed latency requirements such as daily reporting or historical analytics, the simpler batch ETL approach may be more appropriate due to its lower operational overhead and perfect data consistency guarantees.

The way in which both models handle failure recovery will also differ significantly. The CDC model can take advantage of Kafka's offset-based replay feature that allows consumers to resume processing from the last committed offset without any data loss, whereas the Batch ETL model relies solely on Airflow's DAG retry mechanism, which causes the entire batch to be re-run when a failure occurs. When recovering from a large-scale outage, the cost associated with multiple retries for failed transactions can add up quickly, whereas CDC provides an incremental recovery process based on the length of time of the outage.

6. Discussion

6.1 Key Findings

The experimental results have validated that CDC-based data pipelines provide substantial improvements in latency over traditional batch ETL pipelines that work with cloud databases. With a 215x reduction in latency at 100,000 records, the ability to access current data nearly instantly allows for real-time customer segmentation, live fraud detection, and personalized dynamic content delivery which is not possible under batch methods. Additionally, the increased throughput results indicate that CDC has produced an increase of 1.8-2.2x in sustained consumption rates versus traditional batch ETL processes, using the same cloud database technologies.

While logic would suggest that event-driven processing would be less efficient than bulk transfer, this is not the case due to the additional costs related to full table scans incurred with every cycle of batch ETL processing. In contrast, CDC only processes changed rows, making CDC data pipeline processes much more efficient for applications where moderate numbers of rows are modified relative to the total number of rows in a particular table. Finally, the consistency analysis indicates CDC will produce a consistency level between 99.93 and 100 percent versus batch ETL producing between 99.97 and 100 percent. For the CRM analytics use case, the difference is negligible if transient inconsistencies occur in a small number of records over a few seconds, but applications that require strict consistency guarantees may require additional verification methods.

6.2 Practical Implications

The architecture in this paper was planned using open-source, vendor-neutral components as a foundation that will allow it to run across multiple cloud providers. Our tests have mainly been run using DigitalOcean Managed PostgreSQL; however, the Debezium connector and Kafka consumer logic are compatible with any PostgreSQL instance that supports logical replication, including AWS RDS, Azure Database for PostgreSQL, and Google Cloud

SQL. Portability is one of the many benefits of using an open-source solution versus a proprietary CDC service like AWS DMS, which locks users into a particular cloud provider's ecosystem.

The resource utilization analysis shows that the entire pipeline can be run on a single 8 vCPU, 16 GB server for a moderate workload (up to 500,000 records), therefore this solution is accessible to small or medium-sized companies. For larger implementations, horizontal scaling of Kafka consumers and FastAPI workers provides scalability options.

6.3 Comparison with Existing Solutions

Our proposed tool has several advantages over proprietary CDC services such as AWS DMS [9]. The first advantage is that the entire CDC process is fully transparent, because all components are open-source and the configuration of each component is under the user's control. In contrast, AWS DMS is a black box managed service with a limited ability to configure the replication process. The second advantage is that it is a vendor-neutral and portable tool, meaning that you can use it with any cloud provider, whereas AWS DMS is tightly coupled to the AWS ecosystem. The third advantage is that it supports fine-grained filtering and transformation of events using Kafka Connect's Single Message Transform (SMT) framework, which enables complex routing logic that would be challenging to implement with DMS.

Despite these advantages, managed services like AWS DMS have benefits in terms of operational simplicity. An organization does not have to manage infrastructure, DMS automatically scales to accommodate increases in volume and it includes built-in monitoring and alerting. Therefore, for organizations that do not have a full DevOps capability, the operational burden associated with managing a Kafka cluster and Debezium connectors may outweigh the flexibility provided by the open-source solution. Our tool is purpose-built for real-time CRM analytics on cloud databases, whereas general-purpose data integration platforms such as Striim [21] and Fivetran are not designed specifically for that use case. The integration of the CDC pipeline with the analytics layer consisting of FastAPI and RFM scoring allows these two to be better integrated than would be possible with a more generic integration platform.

6.4 Lessons Learned

There are some general lessons to be learned from the process of developing and evaluating this system. First, an incremental rather than revolutionary transition from batch ETL to CDC occurred during the overall development of the system. The Airflow-based batch pipeline was used as the primary functional baseline at the beginning of the project while CDC was developed as an additional layer that complemented the batch pipeline rather than replacing it completely. This incremental migration strategy helped to mitigate overall risk and also provided the opportunity to A/B test the quality of data captured through both methods.

Second, although DigitalOcean Managed PostgreSQL does not have as many features as AWS RDS or Google Cloud SQL, it was sufficient for the anticipated workload and offered a much simpler pricing model. The only significant limitation encountered was the connection pool size of 25 connections (PgBouncer transaction mode) which required careful adjustment of the number of consumers using parallel processing and the number of FastAPI workers in order to avoid exhausting the available connections under periods of high demand.

Third, monitoring was important to maintaining the health of the CDC pipeline. The primary operational metric is the Consumer Lag in Kafka. If consumers start to show a sustained increase in lag, you either need to horizontally scale your consumers or identify a potential bottleneck downstream. We set up automatic alerts for Consumer Lag using Prometheus and Grafana, alerting whenever the Consumer Lag exceeds an adjustable threshold (default is 10,000).

6.5 Limitations and Threats to Validity

There are multiple limitations to consider when reviewing these results. First, all evaluations were performed on a single cloud provider (DigitalOcean) and a specific managed PostgreSQL configuration. Hardware differences such as infrastructure, networking, and managed service implementations will all affect how each provider performs. Second, although the synthetic dataset used in this evaluation was created to emulate a reasonable e-gaming CRM workload, it is impossible to guarantee that it would match all production workloads. For instance, your production workload may involve complex schema changes, binary data, or irregularly distributed primary key distributions. Third, all techniques were evaluated on dedicated infrastructure in a controlled testing environment, and results may

vary when applied to a production environment with shared resources. Fourth, as this evaluation involved only PostgreSQL, the performance results obtained for some other database types (MySQL, MongoDB, and SQL Server) supported by Debezium may produce significantly different results.

7. Conclusion and Future Research Directions

The goal of this project was to create a system that could manage real-time data pipelines in the cloud for PostgreSQL databases. The software system described in this paper takes data from a source, uses Debezium to capture changes to the source, and sends those changes via an Apache Kafka message broker to a Kafka consumer for further processing. We utilized FastAPI as a framework for building REST APIs to receive messages from the Kafka consumers and perform data transformations, manage connections to database instances, and deliver analytics including RFM customer scoring.

We also developed a method for measuring performance of the data pipeline based on the amount of time it took to complete the entire data pipeline process from start to finish. We measured the average latency through the data pipeline at 389ms (215x faster than batch ETL processes with 1-minute latency), sustained 2,300 records/second throughput, over 99.96% consistent data transfer, and up to 84% decrease in API response times when Redis caching was added. The system supports scaling to 500 concurrent users with negligible failures, confirming that there is sufficient performance when using this method to synchronize data in real-time with cloud databases for latency-sensitive applications.

Some potential avenues for future work include: performing the above evaluation in a multi-cloud manner by deploying the same data pipeline across multiple cloud providers (e.g. AWS RDS, Azure Database for PostgreSQL, Google Cloud SQL) to quantify provider-specific differences in performance; incorporating machine-learning based anomaly detection into the pipeline to detect real-time data quality and pipeline failures; investigating auto-scaling methods for Kafka consumers and FastAPI workers based on observed consumer lag and API response time; and evaluating the ability of Debezium to handle schema evolution scenarios (adding or removing columns, changing data types) and how well it would perform in production-like settings.

Reproducibility. All source code, Docker Compose configurations, Debezium connector definitions, synthetic data generators, and benchmark scripts used in this study are publicly available at:

<https://github.com/kananbunyad/CDC-Pipeline-Research>. The repository includes a one-command setup script (setup.sh) and a master benchmark runner (run_all.sh) that reproduce all reported measurements.

References

- [1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50-58, 2010.
- [2] A. Corbellini, C. Mateos, A. Zunino, D. Godoy, and S. N. Schiaffino, "Persisting big-data: The NoSQL landscape," *Information Systems*, vol. 63, pp. 1-23, 2017.
- [3] R. Kimball and M. Ross, *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling*, 3rd ed. Indianapolis, IN: Wiley, 2013.
- [4] M. Kleppmann, *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. Sebastopol, CA: O'Reilly Media, 2017.
- [5] N. Narkhede, G. Shapira, and T. Palino, *Kafka: The Definitive Guide: Real-Time Data and Stream Processing at Scale*. Sebastopol, CA: O'Reilly Media, 2017.
- [6] PostgreSQL Global Development Group, "Logical Replication," PostgreSQL 15 Documentation, 2024. [Online]. Available: <https://www.postgresql.org/docs/current/logical-replication.html>
- [7] PostgreSQL Global Development Group, "Write-Ahead Logging (WAL)," PostgreSQL 15 Documentation, 2024. [Online]. Available: <https://www.postgresql.org/docs/current/wal-intro.html>
- [8] Debezium Community, "Debezium Documentation: Connectors for PostgreSQL," 2024. [Online]. Available: <https://debezium.io/documentation/reference/connectors/postgresql.html>
- [9] Amazon Web Services, "AWS Database Migration Service User Guide," 2024. [Online]. Available: <https://docs.aws.amazon.com/dms/latest/userguide/>
- [10] G. Wang, J. Koshy, S. Subramanian, K. Paramasivam, M. Zadeh, N. Narkhede, J. Rao, J. Kreps, and J. Stein, "Building a replicated logging system with Apache Kafka," *Proc. VLDB Endowment*, vol. 8, no. 12, pp. 1654-1655, 2015.

- [11] Amazon Web Services, "Amazon Relational Database Service (RDS) Documentation," 2024. [Online]. Available: <https://docs.aws.amazon.com/rds/>
- [12] Google Cloud, "Cloud SQL Documentation," 2024. [Online]. Available: <https://cloud.google.com/sql/docs>
- [13] Microsoft Azure, "Azure Database for PostgreSQL Documentation," 2024. [Online]. Available: <https://learn.microsoft.com/azure/postgresql/>
- [14] DigitalOcean, "Managed Databases Documentation," 2024. [Online]. Available: <https://docs.digitalocean.com/products/databases/>
- [15] DBEaver Community, "DBEaver: Universal Database Tool," 2024. [Online]. Available: <https://dbeaver.io/>
- [16] Apache Software Foundation, "Apache Airflow Documentation," 2024. [Online]. Available: <https://airflow.apache.org/docs/>
- [17] Spotify, "Luigi: A Python module for building complex pipelines of batch jobs," 2024. [Online]. Available: <https://github.com/spotify/luigi>
- [18] Prefect Technologies, "Prefect: The modern data stack orchestrator," 2024. [Online]. Available: <https://www.prefect.io/>
- [19] Zendesk, "Maxwell: A MySQL-to-JSON Kafka producer," 2024. [Online]. Available: <https://github.com/zendesk/maxwell>
- [20] Google Cloud, "Datastream for BigQuery Documentation," 2024. [Online]. Available: <https://cloud.google.com/datastream/docs>
- [21] Striim Inc., "Real-time data integration and streaming analytics," 2024. [Online]. Available: <https://www.striim.com/>
- [22] V. Kumar and W. Reinartz, Customer Relationship Management: Concept, Strategy, and Tools, 3rd ed. Berlin: Springer, 2018.
- [23] A. M. Hughes, Strategic Database Marketing: The Masterplan for Starting and Managing a Profitable, Customer-Based Marketing Program, 2nd ed. New York: McGraw-Hill, 2000.
- [24] C.-H. Cheng and Y.-S. Chen, "Classifying the segmentation of customer value via RFM model and RS theory," Expert Systems with Applications, vol. 36, no. 3, pp. 4176-4184, 2009.
- [25] S. Ramirez, "FastAPI: Modern, fast web framework for building APIs with Python," 2024. [Online]. Available: <https://fastapi.tiangolo.com/>
- [26] J. Kreps, N. Narkhede, and J. Rao, "Kafka: A distributed messaging system for log processing," in Proc. NetDB Workshop, Athens, Greece, Jun. 2011, pp. 1-7.
- [27] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," ACM Computing Surveys, vol. 35, no. 2, pp. 114-131, 2003.
- [28] P. J. Sadalage and M. Fowler, NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Upper Saddle River, NJ: Addison-Wesley, 2013.