LEARNING MADE EASY

Confluent & Qlik Special Edition

Apache Kafka® Transaction Data Streaming

for dummies®
A Wiley Brand

Enable flexibility in modern data pipelines

Implement Kafka transaction streaming

Ingest, store, and deliver transaction data

Brought to you by

CONFLUENT™

Qlik

Thornton J. Craig
Kevin Petrie
Tim Berglund

About Confluent

Confluent is the data streaming platform that is pioneering a fundamentally new category of data infrastructure that sets data in motion. Confluent's cloud-native offering is the foundational platform for data in motion—designed to be the intelligent connective tissue enabling real-time data, from multiple sources, to constantly stream across the organization. With Confluent, organizations can meet the new business imperative of delivering rich, digital front-end customer experiences and transitioning to sophisticated, real-time, software-driven backend operations. To learn more, visit **www.confluent.com**

About Qlik

Qlik's vision is a data-literate world, where everyone can use data and analytics to improve decision-making and solve their most challenging problems. Qlik offers real-time data integration and analytics solutions, powered by Qlik Cloud, to close the gaps between data, insights and action. By transforming data into Active Intelligence, businesses can drive better decisions, improve revenue and profitability, and optimize customer relationships. Qlik serves more than 38,000 active customers in over 100 countries. To learn more, visit **www.qlik.com**.

Apache Kafka® Transaction Data Streaming

Confluent & Qlik® Special Edition

by Thornton J. Craig, Kevin Petrie, Tim Berglund

for dummies®
A Wiley Brand

Apache Kafka® Transaction Data Streaming For Dummies®, Confluent & Qlik® Special Edition

Published by
John Wiley & Sons, Inc.
111 River St.
Hoboken, NJ 07030-5774
www.wiley.com

ISBN: 978-1-119-62601-5 (pbk); ISBN: 978-1-119-62603-9 (ebk)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Publisher's Acknowledgments

Some of the people who helped bring this book to market include the following:

Project Editor: Carrie Burchfield-Leighton

Editorial Manager: Rev Mengle

Acquisitions Editor: Steve Hayes

Business Development Representative: Molly Daugherty

Table of Contents

Introduction

Apache Kafka is a highly flexible streaming platform that supports multiple, key use cases in modern data architectures. One critical use for Kafka, and the focus of this book, is building scalable, real-time data pipelines. Streams of unbounded data are effectively ingested, persisted, and delivered using Kafka as a framework. High-volume, high-velocity data from social media feeds, Internet of Things (IoT) sources, and any database transactions are written, replicated, and read in real-time. Change data capture (CDC) database replication technologies leverage Kafka to enable a highly effective platform supporting real-time transactional database streams.

About This Book

Real-time database transactions are a growing and preferred use case for Kafka. Traditional batch methods struggle to meet the demands of modern data pipelines. Replicating high volumes of data from large enterprise databases requires CDC and a high-performance data processing pipeline. *Apache Kafka Transaction Data Streaming For Dummies*, Confluent & Qlik Special Edition, explores how Kafka supports critical database transaction use cases and how to effectively implement these in real-time transaction streaming solutions.

Icons Used in This Book

Remember icons mark the information that's especially important to know.

The Tip icon points out helpful suggestions and useful nuggets of information.

The Technical Stuff icon indicates some additional technical details that may be interesting.

Beyond the Book

This book can help you discover more about implementing Kafka transaction streaming, but if you want resources beyond what this book offers, we have some insight for you:

- `https://www.qlik.com/us/products/qlik-replicate`: Qlik Replicate® empowers organizations to accelerate data replication, ingestion and streaming across a wide variety of heterogeneous databases, data warehouses, and big data platforms.
- `https://cnfl.io/express-scripts`: An on-demand webinar with joint customer, Express Scripts
- `https://www.qlik.com/us/resource-library/streaming-change-data-capture`: This book serves as a practical guide for enterprise architects, data managers and CIOs as they enable modern data lake, streaming and cloud architectures with CDC.
- `www.confluent.io/product/confluent-platform`: Confluent's streaming platform based on Kafka
- `www.confluent.io/confluent-cloud`: Kafka for the cloud
- `https://kafka-tutorials.confluent.io`: A collection of event streaming use cases, with each tutorial featuring an example scenario and several complete code solutions
- `https://docs.confluent.io/current/tutorials/index.html`: Detailed demos of Confluent and Kafka
- `http://kafka.apache.org`: Especially for developers, documentation that offers deeper dives into implementation and code details

IN THIS CHAPTER

- Defining a data stream
- Seeing how transactional data becomes part of a data stream
- Creating opportunities with transaction data streaming

Chapter 1
Understanding Transaction Streaming with Apache Kafka

Modern data requirements are real time. This is the case with processing online transactions or social media feeds, capturing Internet of Things (IoT) data, or generating "just-in-time" or instant analytics on any type of database transaction. Rapid, incremental, and high-volume changes in data require ultra-fast replication or processing. Traditional batch processes operate on a schedule using a fixed time window to process database transactions, often slowing or pausing production operations during off-hours given the high impact of its full-load replication on source databases. Batch processes struggle to support real-time and continuous changes in databases.

Changes in data are commonly generated from source datasets by using change data capture (CDC) processes. CDC focuses on capturing most the most current or real-time changes in data and metadata (for example, data about the datasets). These incremental changes (as transactions) are collected from the source using common CDC methods like log-based capture. These transactions

then create a stream. The stream of changes is then collected, persisted, and delivered using a streaming platform like Kafka.

Transaction streaming is the collection of database changes derived through CDC that are delivered to a streaming platform like Kafka. The key benefits of transaction streaming are

- Support for real-time processing and analytics via continuous replication of critical, high-volume database changes
- Reduction of impact on production workloads via non-intrusive CDC methods
- Low-latency, automated delivery, and persistence of transactions to multiple targets via Kafka

In this chapter, we give you the low-down on the components of transaction streaming.

What is a Data Stream?

A data stream consists of data with three common characteristics. A data stream may include one or more of these characteristics:

- **Unbounded:** No specific beginning or end of the dataset clearly defined
- **Arriving sporadically:** Thousands of records in sub-milliseconds, or very few records over hours, days, or even longer time periods
- **Varying sizes:** Records varying in size from KBs to GBs

Streaming data is a bit different than traditional batch data. Figure 1-1 shows you an illustration of the streaming data process compared with traditional batch data.

The sources for batch and streaming data are often the same, although newer sources like IoT or other real-time feeds are usually associated with streaming data. Even non-real-time data can be a stream, like CDC database transactions. The key differentiator between batch and streaming is that the stream data isn't bounded or captured in a specific unit or group to be processed.

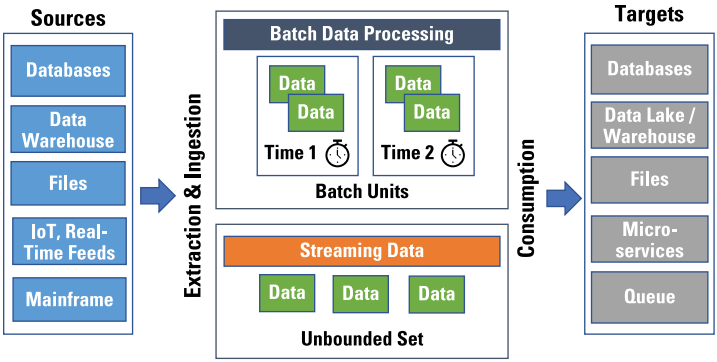

FIGURE 1-1: Comparing streaming data and traditional batch data processes.

Because this data is expected to require processing as a stream, a streaming solution must support real-time ingestion, unbounded (uncategorized/ungrouped) data, and variable sizes and velocities. A primary challenge is ingesting and processing these data streams in real time, efficiently, and at scale. Table 1-1 gives you the modern data characteristics and requirements for processing streaming data.

TABLE 1-1 Modern Data Characteristics

Characteristic	Description/Use Cases
Real-time ingestion	IoT, sensor, video data Social media feeds
(Near) real-time event response	Online shopping Real-time analytics processing Geolocation analysis
Asynchronous data transfer	Immediately available data, consumed when needed
Data distribution	Many users, consumers from single streaming or aggregated stream sources Parallel processing and efficient scaling
Service modularity	Microservices or de-coupled users as consumers
Data extraction and replication	Transactional data CDC

TECHNICAL STUFF

Real time and near real time differ and depend on use cases. For some requirements, a 15-minute delay is real time; for others it needs to be sub-second.

REMEMBER

Apache Kafka enables the capture and process of CDC transactions by ingesting data at any required velocity.

Defining Transaction Data

Transaction data consists of changes to database data/operations or metadata (commonly the schema). Read operations are generally not transaction data because no data or metadata is changed.

TECHNICAL STUFF

Metadata commonly refers to a database schema, but it may also include information about infrastructure, users, data instances, and related information. For more details on metadata, visit `https://www.qlik.com/us/resource-library/streaming-change-data-capture`.

Common transactions include

- **Inserts:** Adding one or more rows to a database. A new row, commonly called a *record,* may summarize the time, date, amount, and name for a recent transaction.

  Kafka also refers to *records* as the base unit of the data log structure in Kafka. Even though they're mostly the same thing, in transaction streaming, they're different.
- **Updates:** Changing fields in one or more existing rows. An address or other data may change in an existing record.
- **Deletes:** Removing one or more rows. If an incorrect transaction is entered, it's erased. (Inserts, updates, and deletes are performed using Data Manipulation Language [DML].)
- **Data Definition Language (DDL) operations:** Removing tables or columns, or altering data types. DDLs are used to change a database's structure.

TECHNICAL STUFF

Traditional Relational Database Management Systems (RDBMS) are frequently sources for transaction data. Alternative or multi-model databases+, such as NoSQL, also generate similar transaction data. Databases may be on-premises, cloud-hosted, or in a hybrid combination of both. Table 1-2 details the different database types and roles.

TABLE 1-2 Database Types and Functions

Database Type	Examples	Roles/Use Cases
RDBMS (SQL Based)	Oracle Microsoft SQL Server DB2 MySQL PostgreSQL	Data warehouse for business reporting Operational databases
Hadoop Ecosystem	Hortonworks Cloudera	Data lakes Unstructured data
NoSQL (Graph, Column, Document, Key-Value, and so on)	MongoDB Marklogic DynamoDB Neo4j	Data warehouse Analytics Social media relationships Specialized or custom use cases where data structures map more closely than relational

REMEMBER

In most cases, database systems support some type of trigger, query, or log-based CDC to allow capture of transaction data operations. The most advanced CDC solutions minimize the impact on source production systems by using a log-based approach that reads backup or recovery logs. Regardless of the CDC capture method, the database changes must be replicated to targets or other consumers. Kafka provides a highly scalable, real-time method of receiving changes from CDC and distributing them in carefully sorted streams to a variety of targets.

Defining Transaction Data Streaming

After transaction data is captured via CDC, there must be a method to ingest, store, and deliver the transaction data streams. Common methods for CDC delivery include direct connections from the CDC platform to targets (also called sinks), which extract data from Kafka. For example, a relational database may rely on CDC mechanisms to capture changes (transactions) and push these into another database, data warehouse, or archival system. This

method is tried and tested, highly effective, and low impact to capture and replicate data changes into target systems. But what if you need to persist the stream of transactions? Or if the stream needs to be consumed by multiple targets? Perhaps some consumers require near real time, while others retrieve infrequently using intermittent jobs.

REMEMBER

A single data transaction stream could be used simultaneously by a data warehouse for analytics, a shared microservices-based application, and an advanced data lake platform for machine learning. *Transaction data streaming* is the unbounded flow of transactions from multiple sources to one or more targets.

Infrastructures grow and quickly require more complex architectures to support these multiple consumption scenarios. The best method to address this common problem is with a new type of architecture that scales horizontally, supports real time, and is running in production at thousands of companies: an event streaming platform.

REMEMBER

Kafka is a proven and well-used enterprise streaming data platform that effectively scales transaction data streams.

The final component of transaction data streaming is where the data stream is delivered — into Kafka. Kafka then replicates and scales the optimized delivery of these streaming transactions. Kafka supports publishing and persistence scenarios as described in Table 1-3.

TABLE 1-3 Kafka Use Cases

Kafka Use Cases	Description
Real-time event processing	Transactions written to cluster are available to be read with millisecond latency.
Streaming ingestion	Cluster can scale to support high volume ingestion of records.
Machine learning	Stream processing (see Chapter 5 for more info) enables analytics using AI/ML operations.
Microservices enablement	Decoupling of producers and consumers enables microservices implementations. Kafka supports both events (triggers or signals from one process to another) and data (movement of data between processes).

Kafka Use Cases	Description
Record or event broker	Kafka performs like a traditional message queue, allowing ingestion and delivery of records or events.
Data persistence	By default, Kafka persists data records for seven days. This timeframe is configurable down to minutes or up to forever, based on demands and use cases.

CDC combined with Kafka enables a complete ecosystem for building a transaction data streaming platform. Chapters 2, 3, and 4 dive much deeper into how this rich solution is fully implemented.

Opportunities Created with Transaction Data Streaming

Transaction data streaming enables flexibility in modern data pipelines. In addition to traditional batch data operations — still having relevant use cases — transaction streaming supports real-time processing and new use cases like advanced analytics.

REMEMBER

Business processes and decision-making functions can be enabled in near real time, without having to wait on batch loads or scheduled processes. CDC functions can be configured to automatically identify and capture source database changes and then write them to Kafka in near real time. The Kafka streaming platform delivers and persists these transactions for immediate processing, archival use cases, or delivery to a data lake, or even consumption by other microservices.

IN THIS CHAPTER

- Studying the Kafka architecture
- Aligning Kafka with transactional data

Chapter 2
Looking Deeper into Apache Kafka

Apache Kafka is a streaming platform. What exactly does that mean? Essentially, a streaming platform ingests, persists, and presents streams of data for consumption. The stream can be any type of data — any source that can be serialized into a record. In this chapter, we dig a little deeper into the anatomy of Kafka, so you can better understand its moving parts and why Kafka is such a powerful streaming platform.

REMEMBER

With a deeper view of Kafka, you see that topics resemble database transaction logs. Transaction data, captured for example via change data capture (CDC), is essentially a log of records that's persisted and made consumable at scale. These records can be consumed for multiple use cases (databases, microservices, immediate stream analytics, and so on), and because they're stored for longer periods, they can be replayed or consumed at slower velocities, and in order. Kafka offers a solution for transaction data through its log-based architecture. For even more information on Kafka, visit `https://kafka.apache.org`.

Looking at the Kafka Architecture

Processing data with Kafka relies on three basic components: producers, brokers, and consumers. Figure 2-1 shows a basic Kafka architecture.

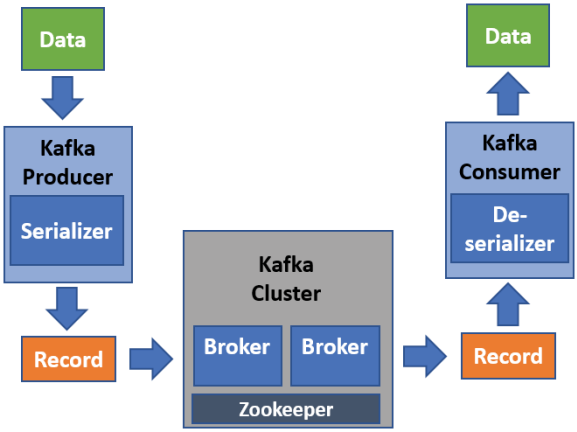

FIGURE 2-1: The basic Kafka architecture.

We cover this structure and its components in this section.

Kafka producers

The *producer* is a piece of code or process that writes data to Kafka. Producers use a serializer and an optional key to create a record containing the source data. The record is then written to Kafka. The producer's components are detailed as follows:

- **Data:** Data is written to Kafka. This can be any type of data that's serialized. Commonly used sources include events such as database transaction logs.
- **Record:** A unit of data in Kafka is a record. Streams of data in Kafka consist of collections of records. One or more interrelated records constitutes a database transaction.
- **Serializer:** Serializers are used by the producer to convert data into a record. Any serializer can be used, provided the same type of de-serializer is used by the consumer.

- **De-serializer:** This is used by the consumer to convert records back into the original data.
- **Key:** A key is an optional identifier used to hash (select) which partition will receive a record. This function is used to target records to specific partitions.

Kafka cluster

A *Kafka cluster* is a collection of brokers. Brokers are the basic functional unit in Kafka. A broker runs the Kafka process and responds to requests from producers and consumers. Brokers rely on, manage, and/or interact with the following components:

- **Topic:** The organizational unit for records is called a topic. Kafka records are written to specific topics. Topics are generally named for the kind of data they contain. Each topic is an immutable, append-only log consisting of a collection of records. Each topic is a logical (and sequential) grouping of a log of records. See Chapter 4 for more information on topics.
- **Partition:** Topics are composed of storage units called partitions. Topics are divided into multiple partitions for scalability. This allows parallel writes and reads and helps enable backup copies of each topic. Partitions are replicated between brokers, providing redundant copies of data in the event of a broker failure. See Chapter 4 for more information on partitions.
- **Apache Zookeeper:** A component that helps Kafka clusters manage distributed consensus is called Apache Zookeeper. Zookeeper keeps track of cluster metadata and helps in the process of electing lead partition replicas.

Kafka consumers

The consumer is an application program that reads records from a topic. Topics can be read by a single consumer, or if the topic is partitioned, a group of consumers can read from its partitions in parallel. This enables faster throughput and lower latency, which allow scaling of reads — an important concept we talk about more with streaming transactions in Chapter 4.

An *offset* is a marked position that Kafka consumers use to keep track of the most recent message they've consumed from each partition. This enables consumers to pause read operations and then pick up where they left off and resume in sequence.

Relating Transaction Data and Kafka

What makes Kafka's architecture so useful for CDC transaction data? We're glad you asked. In this section, you take a look at the life cycle of transactional data and how it aligns with a Kafka data pipeline.

REMEMBER

Here is one example of the life cycle of a transaction:

1. **Database transaction events are captured by low-impact CDC.**
2. **The transaction event is converted by the producer to Kafka record.**
3. **The record is written to the topic (log of records).**
4. **The record is persisted for as long as retention is set in Kafka (minutes to days, or indefinitely).**
5. **The record is stored redundantly (via replicated partitions).**
6. **Multiple consumers read the record over different time periods.**
7. **Consumer groups can read records in parallel from multiple partitions to enable highly scalable reads throughput (although transactional consistency must still be maintained).**

Kafka can scale with multiple topics and partitions to support ingestion and delivery of high-volume CDC transactional data from a variety of sources simultaneously and using the same cluster. Using partition keys, Kafka can also maintain strict ordering of transactions.

ORDER, SEMANTICS, AND CONSISTENCY — OH MY!

Kafka captures database events as transactions by using a producer to create a record containing the details of the event. Each record is written to a Kafka topic. Because records of the same key always land in the same topic, transactions related to the same key are kept strictly in order. Database transactions often require transactional consistency — defined here as being holistic, ordered, and committed only one time (exactly once semantics). You can find more about "exactly once" semantics and options with Kafka Streams at `www.confluent.io/blog/enabling-exactly-once-kafka-streams`. We also want to make the distinction between database transaction consistency and exactly once semantics: They overlap, and you need both, but they're different things. We cover this more in Chapter 5.

Events (records in Kafka) can also be grouped by topic into one partition, guaranteeing the correct order. Other methods include sharing transactional metadata by using headers that are attached to Kafka records. You can read more about this in Chapter 6.

Figure 2-2 shows the anatomy of how Kafka uses topics, partitions, and records.

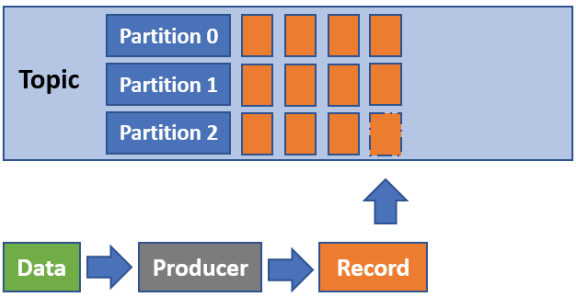

FIGURE 2-2: Kafka topics, partitions, and records.

IN THIS CHAPTER

- Integrating data with Apache Kafka Connect API
- Using change data capture (CDC) to integrate data
- Looking at architectural patterns for integrating data

Chapter 3
Ingesting Data with Apache Kafka

Traditional data integration pipelines focus on Extract, Transform, and Load (ETL) or, as is now common with big data architectures, Extract, Load, and Transform (ELT) processing. Data is extracted using a process like change data capture (CDC) and/or by writing data to another database or target. Data is then either transformed before it is loaded (for example, ETL) or loaded into an analytical or target system and then transformed (ELT). The emergence of data lakes and other big data architectures has blurred the line between ETL and ELT, and both patterns are now commonly used.

Apache Kafka is used to ingest and transport data in an ETL/ELT pipeline. Data is read and written by various databases and other end points using Kafka producers and consumers. Low latency, high availability, scalability, and persistence enable Kafka to serve as a highly efficient pipeline to move data for real-time integration requirements. Kafka also supports in-stream data integration with options like the Kafka Streams API, which we cover in Chapter 5.

Kafka supports common data integration methods, and we cover two of them here: Kafka Connect and CDC. We also cover common data integration architectural patterns in the last section of this chapter.

Kafka Connect

Kafka Connect is an open-source component of Kafka that supports connections to common databases. Connect creates a common framework to develop, deploy, and manage connectors to Kafka. Common scenarios implemented with Connect include database ingestion, collecting metrics from application servers, or database extraction processes. Each of these workflows can be processed incrementally, in near real time, or in batches. For more info on Kafka Connect, visit `https://docs.confluent.io/current/connect/index.html` or `https://kafka.apache.org/documentation/#connect`.

Connectors support simplified, declarative methods to pull or push data with Kafka. Figure 3-1 shows a typical Kafka Connect configuration.

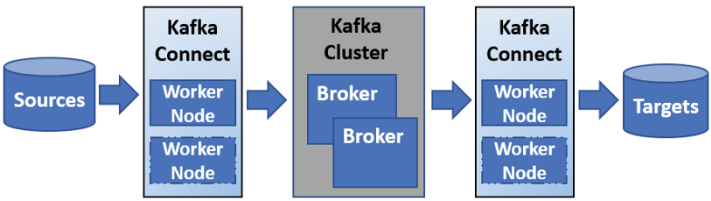

FIGURE 3-1: A typical example of a Kafka Connect configuration.

Kafka Connect is deployed as a separate interface between data sources and targets and the Kafka cluster. Connect uses a distributed cluster, leveraging multiple worker nodes to provide horizontal scaling and fault-tolerance.

A challenge of using Kafka Connect is creating the required connectors to external sources and targets. Connectors are reusable but may need development efforts to build initially. Most organizations leverage a third-party vendor, such as Confluent

(see Chapter 7 for more info), to provide prebuilt connectors for common databases and other data sources. Using a managed service for Kafka Connect also helps provide a fully scalable Kafka Connect infrastructure.

Using Kafka Connect for data integration leverages connectors that easily move data into and out of Kafka. The actual data integration functions (called *transformations*) are still completed by external functions, unless in-stream processing is used. Chapter 5 discusses in-stream processing in detail.

Kafka Connect provides a scalable, reusable, common interface to most data sources and targets. These features facilitate building high-performance data integration pipelines.

Change Data Capture (CDC)

CDC is a method for tracking, capturing, and delivering changes from source datasets. Implemented well, CDC enables data integration by replicating database or data source changes with little or no impact on those sources.

Modern CDC systems use three common data capture methods:

- **Triggers:** A source insert, update, or delete in a database "triggers" writing a change to a separate table (commonly called a *shadow* or *change-capture table*). This table is a separate record of changes used by CDC to capture all data source changes. The change-capture table is then replicated to data targets or into a streaming platform like Kafka.
- **Query-based:** A CDC engine "queries" the source database, looking at timestamps, version numbers, or status columns, and copies new data to targets or into the transaction streaming platform.
- **Log-based:** CDC scans backup or recovery transaction logs and identifies changes to be replicated to targets or into the transaction stream.

Table 3-1 describes the impact and requirements for CDC capture methods, which sometimes depend on data source requirements.

TABLE 3-1 CDC Capture Method Requirements and Impacts

CDC Capture Method	Requirements	Impact
Triggers	Sometimes used when CDC process has no access to transaction logs. Database triggers required to write separate tables can impact performance.	Medium
Query-based	Often used if the source does *not* have change logs, as is generally the case with data warehouses. This method has similar use cases as triggers, but with less impact. The CDC engine queries the database at regular intervals, although the time between intervals can significantly increase latency.	**Low**
Log-based	The CDC engine scans transaction logs to identify and capture real-time updates with minimal or no impact on source production workloads. This is the preferred CDC method because it's efficient and provides minimal impact.	**Minimal**

Common Data Integration Architectural Patterns

CDC combined with Apache Kafka is becoming a primary streaming pipeline to support multiple architectural patterns for data integration. Kafka's topic mechanism is appealing for large enterprise environments that need to create and manage granular data streams between many sources and targets. Table 3-2 summarizes the most common patterns for data integration in modern data platforms.

TABLE 3-2 Data Integration Architectural Patterns

Pattern	Description
Data lake integration	This entails data ingestion into a data lake or other analytics platform. Many sources may feed into a data lake and require CDC + Kafka to effectively ingest and transform data.
Microservices	Data integration is more commonly performed using de-coupled microservices to process events and data separately. CDC + Kafka act as the extraction, persistence, and transit layer between different microservices.
NoSQL	Traditional relational databases leverage structured data and common SQL queries to extract and integrate data. Multi-model databases now support semi-structured data like document or graph databases. Kafka enables connections and data integration between multi-model databases.
Streaming-first architecture	An emerging integration pattern is the concept of streaming first, whereby data is ingested directly into a streaming platform using CDC and Kafka. Kafka then persists this data until it can be consumed by more traditional data warehouses or analytics platforms. The heart of this concept is to put the data in the stream first, instead of loading into a traditional datastore or data warehouse.

IN THIS CHAPTER

- Using CDC and Kafka to deliver real-time results
- Configuring topics and partitions

Chapter 4
Applying CDC to Apache Kafka

The complete change data capture (CDC) process consists of two functions: capturing the changed data and enabling replication of the changed data. Both functions grouped together are commonly referred to as CDC or replication (although data replication is technically speaking a distinct process from CDC). Understanding this two-step process is important as you look at CDC and Apache Kafka in this chapter. CDC captures and presents data changes to Kafka. Kafka becomes the persistence and transit layer in which CDC changes are replicated and delivered.

Delivering in Real Time: CDC and Kafka

Any CDC method can be used to create a stream of records. These records can be written to a streaming platform like Kafka, where they're persisted and consumed by multiple targets. The CDC replication process is shared between the CDC system and Kafka producers. The captured data from the CDC process must be serialized and written to a Kafka topic. CDC systems, such as Qlik's data integration solution, acquired from Attunity (see Chapter 7 for more info), fully automate the configuration of databases as

producers to Kafka, eliminating the need for scripting and greatly simplifying management.

REMEMBER

CDC source metadata, such as Data Definition Language (DDL) schemas, is critical to a fully optimized transaction streaming pipeline. Kafka doesn't see any internal data; it's just moving bytes, so optimizing data definitions and serialization are key concerns for an effective transaction pipeline.

Topic schemas and partition usage (see the next section "Topics and Partitions" for more info) should be aligned with throughput and latency requirements (more on this in Chapter 6). Producers can use optimized schema serializers, such as Apache Avro, to ensure consistency in data stream pipelines. Many organizations leverage dedicated schema registries such as Confluent's Schema Registry. These registries allow for version control and updates of topic schemas while enforcing record consistency.

For more information on Apache Avro in context with Connect, visit `https://kafka.apache.org/documentation/#connectapi`. Head to Chapter 7 to discover more on Confluent's Schema Registry.

Figure 4-1 shows a more detailed CDC and Kafka architecture.

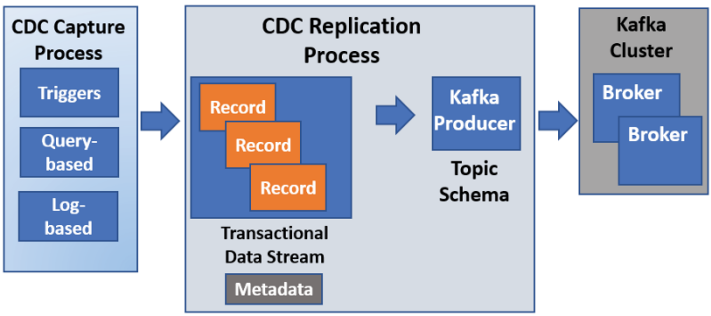

FIGURE 4-1: The CDC and Kafka architecture.

Using CDC to publish database changes as a record stream to Kafka is increasingly considered a best practice. Data engineers can use a familiar product or process because CDC is widely adopted as the preferred method to efficiently replicate database records to analytics and microservices platforms. CDC also reduces scripting work for Kafka developers, especially when a fully automated solution is used.

Getting Organized: Topics and Partitions

Two key goals of a streaming platform are to optimize data throughput and ensure consistency. CDC inserts, updates, deletes, and DDL changes are most commonly captured from database sources. Kafka's architecture leverages topics to provide structure, partitions for scalability (high throughput), and replication for redundancy. Transaction streaming use cases are often a tradeoff between high throughput, guaranteed ordering, and low latency.

Global message order is guaranteed when all transactions go to a single partition. Scaling a topic over multiple partitions allows parallel reads and writes that can improve performance at the expense of global ordering of messages. The solution is to choose a key that provides the key-specific ordering required. Additionally, the size of each Kafka record affects the time required for data ingestion and consumption, as well as overall throughput of transactions.

Table 4-1 compares two Kafka topic configuration options for transactional data. Tables are based on CDC from a relational database.

TABLE 4-1 Kafka Topic Configuration Options

Option	Benefits	Considerations
One topic per database table	Easy to understand categories and locate topics Can isolate and reload one table to check data consistency	Thousands of tables and topics can be complex to manage Best used with auto-topic creation for easier provisioning Transformation required for topic labeling and tracking if databases share schemas or table names
One topic per application (focused on consuming application)	Simplifies provisioning of many tables for database application Easily categorizes DB applications	Must partition carefully for consumers to understand records' schema and table

Table 4-2 compares four common partition configuration options for transaction streaming.

TABLE 4-2 Kafka Partition Configuration Options

Option	Benefits	Considerations
Random partitioning	Spreading data across partitions improves parallelism and performance Ideal for initial loads or insert-only updates	For correct record order, need complex logic to consume all partitions, then sort by key Complex logic required for transaction consistency Randomly assigns partition to each record
Partition by schema/table	Maintains correct record order for each table as all records go to the same partition Multiple ordered tables can be written to one topic	Transformation logic might be required to ensure one table per partition Complex logic also required for transaction consistency between tables Often used for multiple tables per topic
Partition by table primary key	Records with same primary key are ordered correctly in same partition Uses most/all partitions	Additional metadata required to map records to tables, if multiple tables per topic Often used for one topic per table scenarios
Partition by transaction ID	Ensures transaction consistency Consumer can easily read transactions sequentially from one partition	Must map all tables associated with transaction to same topic for accurate consumption

Separating topics by transaction use cases enables a single Kafka cluster to support many CDC and integration pipelines. Tables 4-1 and 4-2 show the potential complexities in determining the best configurations for aligning database tables with Kafka topics and partitions.

TIP

Optimizing Kafka to support transactional data streaming often has tradeoffs between high throughput, guaranteed ordering, and low latency. Focus topic and partition considerations on each use case to ensure an ideal match between transactional requirements and the flexibility of Kafka.

IN THIS CHAPTER

- Defining the core components that support stream processing
- Reaping the benefits of stream processing

Chapter 5
Understanding Stream Processing with Apache Kafka and Its Benefits

Stream processing is a method of performing transformations or generating analytics on transactional data inside a stream. Traditional Extract, Transform, Load (ETL)-based data integration functions are performed on data that will be analyzed in a database, data lake, or data warehouse. Analytics are typically run against a data warehouse with structured and cleansed data. In contrast, streaming platforms like Apache Kafka enable both integration and in-stream analytics against data as it moves through the stream.

Transactional data also benefits from stream processing because it allows both transformation/cleansing functions and rerouting or aggregation of multiple data sources in-stream. Change data capture (CDC) is an optimal mechanism for capturing and delivering transactional data from sources into a streaming platform. Stream processing can then take this CDC generated data and create new streams for additional use cases, or it can generate analysis against the stream of transactions.

In this chapter, you discover the components that support stream processing and learn why stream processing is beneficial.

Stream Processing in Kafka

Kafka includes three core components that support stream processing: The Apache Kafka Streams API, KSQL, and windowing.

The Kafka Streams API

The Streams API is a client library that supports building applications or microservices. Source data is read from Kafka and written back to Kafka. The Streams API lives outside an actual Kafka broker or cluster. You can see a typical Kafka Stream API structure in Figure 5-1.

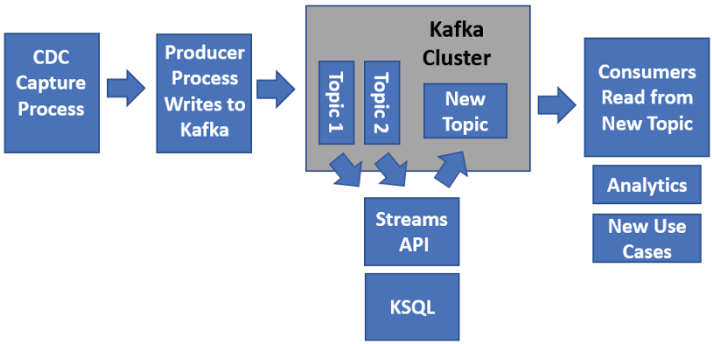

FIGURE 5-1: The structure of a typical Kafka Streams API.

The basic concept of stream processing is defined by the relationship of tables and streams. A relational database uses tables, which may be a familiar concept to you. A simple table is a collection of key-value pairs. A Kafka topic is also a collection of key-value pairs — as a changelog. The idea is that database values are converted to a changelog and then reconstructed or used to create new values by operations done in stream processing. This is also a basic example of how CDC replicates source data to a target.

Key-value pairs are two linked data items, usually expressed as a table: planet:earth, sky:blue, water:H20.

In Figure 5-2, we show you a simple example of the relationship between tables and streams.

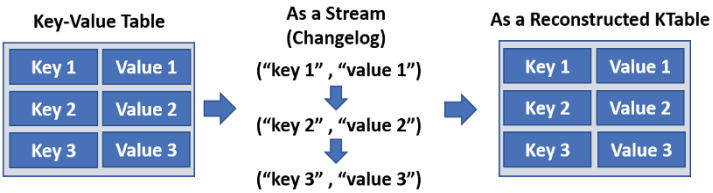

FIGURE 5-2: The stream-table duality.

TECHNICAL STUFF

Stream processing in Kafka uses abstracted concepts of the stream (changelog) called *KStreams*. A KStream is a lightweight abstraction on top of a Kafka topic, with each event in a KStream representing a record in the topic. A *KTable* is a Kafka Streams API abstraction that turns that changelog back into an efficiently accessible, durable, replicated, in-memory key-value store.

KSQL

REMEMBER

KSQL is a SQL-like language for describing stream processing operations on data in Kafka. It uses similar stream and table abstractions as the KStream and KTable classes found in the Kafka Streams API (see the preceding section).

TIP

For more information on KTable and KStream and their differences, visit `docs.confluent.io/current/streams/faq.html`.

Looking into the stream: Windowing

The last core component of stream processing is windowing. *Windowing* uses time-based constraints to determine what subset of records is being viewed. Because streaming data is unbounded, when working with streams, you must define the time window you're referencing to group records for processing.

Commonly used time-based window options are described in Table 5-1.

TABLE 5-1 Windowing Options in Kafka

Window Type	Description
Tumbling (time-based)	Fixed-size, non-overlapping, gap-less
Hopping (time-based)	Fixed size, overlapping
Sliding (time-based)	Fixed size, overlapping using differences in record timestamps
Session (session-based)	Dynamically sized, non-overlapping, data-driven

Benefitting from Stream Processing

Pulling the concepts of stream processing all together, the following key elements of stream processing enable many critical benefits:

- **Exactly-once:** Kafka supports a transaction API that provides for messages to be processed exactly once, even if duplicate messages are delivered to a topic.
- **Stateful and stateless:** Stream processing programs can process only each individual message without regard for previous messages, or they can aggregate state based on the history of the messages in a stream.
- **Time:** Windowing and other time-based joins or aggregations are supported by the Streams API (see the earlier section "Stream Processing in Kafka" for more information).
- **CDC:** CDC leverages log-based extraction of data to build new streams and perform in-stream analytics from data that begins in relational database tables.
- **Real-time:** Stream processing is done in real-time windows on real-time data, processing one record at a time, rather than storing messages in batches for processing later.

Stream processing can be complex but also powerful. Table 5-2 describes common stream processing use cases.

TABLE 5-2 Stream Processing Use Cases

Stream Processing Use Case	Description
Real-time analytics	KSQL or the Kafka Streams API is used to generate analysis against live streams of data, often computed over defined time windows.
Microservices integration	KStreams integrated into microservices allow each service to process inputs from other services.
Log analysis	KSQL allows filtering of records from log data sources to look for anomalies or other critical variations.
Data integration	The Streams API and KSQL both support basic to more-complex data integration scenarios, using simple SQL-like commands and queries.

IN THIS CHAPTER

- Planning transaction data streaming systems
- Maintaining transaction consistency

Chapter 6
Starting Your Journey: Effective Planning

In this chapter, we explain planning and implementing your CDC and Apache Kafka streaming architecture. You explore common characteristics of effective streaming approaches, including careful pipeline design, standardized process, pooled resources, granular monitoring, data governance policies, and productive collaboration. You dive into implementation best practices, including configuration methods to ensure transaction consistency. We also give you a case study from Generali, a Fortune 100 financial services organization, in implementing CDC and Apache Kafka.

Planning an Effective Transaction Data Streaming System

Designing, building, and managing an effective enterprise streaming platform is tricky business. You need to put efficient and scalable processes in place that meet service level agreements (SLAs) and maintain data integrity. The five common, high-level

attributes of effective streaming approaches to transaction data that span people, process, and technology are as follows:

- **Effective data pipeline design:** Design pipelines to address three requirements:
  - Publish transaction record inserts, updates, and deletes, as well as source schema changes, to the appropriate topic. Connectors and CDC offerings can achieve this.
  - Run data integrity checks with an update processor to ensure that inserts/updates/deletes are processed once and only once.
  - Identify and retry errors or processing failures.
- **Standardized process:** Data engineers need clear guidance and guard-rails about how best to operate. This includes a set of steps, addressing planning, design, data quality, testing, and production. Developers need a common dictionary and ontology as they design their streaming analytics algorithms. These steps help data teams meet business requirements and efficiently execute projects on time.


TIP

- **Shared resource pool:** Use cases, resource requirements, and usage patterns will vary by line of business and department. Supporting them with a common cloud-based platform and ideally an automated service menu improves infrastructure utilization. When it comes to cost, an enterprise-wide, cloud-based approach is typically more economical than individual "shadow IT" arrangements.
- **Monitoring:** It's critical to monitor state, memory utilization, throughput, latency, number of partitions, and lags in creation/population of topics. These and other metrics signal your ability to meet SLAs from the business and maintain sustainable loads on the infrastructure.
- **Data governance:** Data governance has several dimensions. For example, data producers are often the right accountable parties for data accuracy, using profiling tools, quality checks, and so on. In this scenario, if Line of Business A publishes transactions to Kafka for use by Line of Business B, A rather than B is responsible for source data quality.

  Compliance is more often a shared responsibility. For example, the General Data Protection Regulation (GDPR) requires that all parties along the pipeline manage European Union citizens' Personally Identifiable Information (PII) only in ways that are

explicitly authorized by the original owners of that data. The central IT organization should define, monitor usage, and enforce policies, and then provide reports to the appropriate internal compliance officer.

To learn more about effective streaming approaches, visit `www.eckerson.com/articles/building-an-effective-transactional-data-streaming-system`.

Transaction Consistency

Most analytics use cases for transaction data require records to be accurate, and this requires transaction consistency, which is the guarantee that committed transactions relate to one another according to consistent rules imposed by databases or other repositories. Transactions must be committed accurately, atomically, in order, and only once. Ideally, you'll maintain the same level of consistency as records travel from source database through each point in the Kafka streaming architecture. Two methods to consider for maintaining transaction consistency include the following:

- **Topic and partition configuration:** Sending all events/messages for a given group of transactions (for example that come from the same table) to a single topic and single partition can be a simple and effective way to maintain transaction consistency. However, with only one topic, you can't easily sort messages, which can flood a given consumer with unneeded records and increase processing overhead. Using one topic and many partitions can alleviate this tradeoff, enabling consumers to scale and not be overburdened with unneeded records. Furthermore, by assigning one message key to each transaction, you can ensure all the messages for a given transaction land in the same partition and therefore the same consumer instance.
- **Metadata:** You also can help ensure transaction consistency by notifying consumer applications when transactions are ready for processing. This is achieved with, for example, simple record headers that provide transaction operation sequence numbers and a "true/false" indicator of whether all events in a transaction have been processed. In addition, you should be sure to always start or recover from the beginning of a transaction to further safeguard transaction consistency.

UNIFYING THE CUSTOMER EXPERIENCE WITH CDC AND KAFKA

Generali Group is one of the leading insurers in the world, with insurance and investment services to 61 million customers in over 50 countries. Generali Switzerland like its Italian parent, believes its business is fundamentally based on data. And its IT organization says unlocking the value of that data is akin to awakening a "sleeping beauty." Generali Switzerland seeks to achieve this by implementing a new data streaming architecture based on CDC and Kafka. With this architecture, the company intends to create a simple, flexible, and unified omni-channel customer experience that reduces churn, increases revenue per customer, and improves operational efficiency.

Generali Switzerland's data team was struggling with several challenges that are familiar to large and established organizations. Its traditional processes for replicating data from core Oracle databases to customer-facing applications were disrupting operations, creating duplicative datasets, and increasing infrastructure requirements. The core systems weren't well integrated with new SaaS-based applications, resulting in days-old customer views, inconsistent data, and complex, siloed customer interactions. This lack of integration, along with increasingly complex core application processes, slowed or even prevented service rollouts.

To right the ship, Generali Switzerland needed to synchronize data on a real-time basis across communication channels for customers and internal stakeholders. The company needed to eliminate inefficient, duplicative datasets, closely integrate legacy and new applications, and enable rapid development of new IT services. The solution? Implementing a "data connection platform" that runs on a transaction data streaming architecture with CDC and Kafka.

Qlik Replicate™ (formerly Attunity Replicate) integrated data and metadata across Kafka topics in a containerized microservices environment. The platform enabled Generali Switzerland to discontinue its legacy approach of repeatedly developing SQL jobs for identifying and copying changed data. The intuitive Qlik Replicate GUI, applicable to all end points, reduced configuration time by replacing custom,

siloed scripted procedures. Meanwhile, Qlik Replicate worked with Confluent Platform to provide efficient transaction data integration across applications, with easy connectors to enable agile development of new software components. You can read more about Confluent Platform in Chapter 7.

The Generali Switzerland Connection Platform now underpins real-time, 360-degree customer views so agents can count on a universally accessible, single source of truth for their interactions. These new capabilities and opportunities all rest on efficient, scalable, and flexible change data capture and Kafka technology. For more information on this case study, visit `www.confluent.io/customers/generali`.

IN THIS CHAPTER

- Understanding Confluent Platform
- Putting Qlik Replicate to work

Chapter 7
Ten Reasons to Choose Confluent and Qlik

Every *For Dummies* book ends in a chapter called the Part of Tens. In this chapter, we've chosen ten reasons why Confluent and Qlik products are good choices for transaction streaming with Apache Kafka. We've divided this chapter into two sections, focusing on five reasons for each company's products. We start with Confluent and end with Qlik.

Seeing How Confluent Event Streaming Works for You

Confluent Platform and Confluent Cloud, build on Kafka, were designed to make the deployment and management of Kafka easier, more reliable, and more secure than Kafka alone. With Confluent, users have the freedom of choice to deploy on any cloud, public or private, stream across on-premises and public clouds, as self-managed software or fully-managed service with Confluent Cloud. Confluent helps you confidently architect and manage a complete streaming solution for the enterprise in five ways.

Development and integration flexibility

Confluent Platform easily integrates with many existing systems as part of a broad and expanding partner ecosystem. The Kafka Connect API and connectors provide developers, data engineers, and operators a simple way to stream records between popular applications, sources, and targets. Some connectors are included in the platform (as JDBC, S3, HDFS, and Elasticsearch) and included as part of Confluent Platform. Many more can be downloaded from Confluent Hub.

Confluent Hub is an extensive library of prepackaged, ready-to-install connectors built by partners, vendors, and the community to help you identify and easily integrate with popular data systems, databases, and applications. Visit `www.confluent.io/hub` for more info.

The Confluent ecosystem also includes extensive producer and consumer clients, with support for languages such as Java, C/C++, Python, Go, and .NET. In addition, a REST Proxy provides universal Kafka access from any network connected device over HTTP.

Comprehensive management, monitoring, and control

A key requirement of transaction streaming is to understand and control Kafka cluster operations. Confluent Control Center provides a curated, GUI-based dashboard that reveals insights about the inner workings of your Kafka clusters and the data flowing through them. You gain key operational and monitoring capabilities, as well as new confidence to meet service level agreements (SLAs). You can assess cluster health, availability, and scalability while tracking KPIs for end-to-end performance and broker/cluster resource utilization across environments based on any Kafka client and any programming language. These KPIs and threshold-based alerts address record delivery status, latency, throughput/consumer lag, and missing/duplicate/delayed events.

Control Center also helps you navigate countless configuration choices to speed Kafka cluster setup and maintenance by proactively recommending configuration parameter values. Users can easily leverage default settings, understand common peer configurations, and run impact assessments. With these insights, you can better manage and optimize the Kafka cluster for maximum

availability, transient high-throughput, or infinite retention. The results are higher confidence and lower risk.

Scalability and disaster recovery

Many organizations need to replicate Kafka topics across multiple data centers and public clouds to serve distributed consumers. Confluent Replicator provides a simple and scalable method of meeting these requirements, seamlessly replicating streams so you can tap the widest possible array of cloud-based services. You can scale horizontally, distributing replication workloads across many CPUs to increase throughput between source and target clusters.

In addition, Replicator automates the disaster recovery failover and switchback process to reduce recovery times across geographically distributed environments. Replicator protects business critical metadata by creating topics as needed while preserving the topic configuration in the source cluster. This configuration can include the number of partitions, replication factors, and any configuration overrides specified for individual topics. Replicator also copies the Confluent Schema Registry data (see the next section for more info), enabling the recovery of schema information in a disaster recovery scenario.

Metadata integration and control

With the Confluent Schema Registry, you can ensure application development compatibility by centralizing event data structures, thereby guaranteeing data compatibility as you extend your streaming platform. Schema Registry provides a RESTful interface for developers to define, share, and adapt standard event schemas over time while staying backward compatible and future proof. Native language libraries also exist for Java, Python, and the languages of the .NET platform to integrate Schema Registry natively into application code. The Schema Registry stores a versioned history of all schemas, allowing the evolution of schemas according to configured compatibility settings. You can increase reliability by seamlessly accommodating changes such as database column updates without breaking inter-dependencies or enduring a manual change process. New schemas and versions are automatically registered, validated, and — if they pose issues — flagged for developers as they adjust record structures over time.

Stream processing interface

Confluent products use a stream processing interface called KSQL. This streaming SQL engine helps developers build powerful continuous stream processing queries against Kafka with familiar SQL-like semantics. KSQL provides an intuitive interactive SQL interface for stream processing on Kafka, without the need to write code in a programming language. You simply perform stream processing tasks using SQL-like statements. KSQL is scalable, elastic, and fault-tolerant, and it supports a wide range of streaming operations, including data filtering, transformations, aggregations, joins, and windowing. It enables use cases such as streaming ETL, scoring, and anomaly detection. (Find out more about KSQL in Chapter 5.)

Pointing out the Perks of the Qlik Solution

When it comes to Kafka, Qlik provides a simple, real-time, and universal solution for converting production databases into live data streams. Data engineers can use the intuitive graphical interface of Qlik Replicate to configure any major database to publish to Kafka systems, applying one consistent process for any end point type. Qlik minimizes impact on source producers, optimizes performance, and easily supports "one to many" publication scenarios. This section tells you how.

Completely automated process

Data engineers use Qlik Replicate to automate the process for configuring databases to publish to Kafka, using a drag and drop approach to create a new target endpoint for Kafka, define the broker server, and browse Confluent Platform environment to select one or more topics. You can design, execute, and monitor this task along with hundreds of other data flows enterprise wide. Qlik Replicate also provides the flexibility to rename schemas or tables, add or drop columns from the producer definition, and filter records that are published to the topic stream.

The same flexible process applies to all end point types, helping you add or remove producers in a modular fashion. Qlik Replicate fully supports and automates all topic and partition configuration

options available with Kafka, executing and monitoring through a centralized, intuitive GUI.

Minimal production impact

Qlik Replicate's change data capture (CDC) technology remotely scans transaction logs to identify and replicate source updates while placing minimal load on source production databases. Row inserts, updates, and deletes, as well as schema changes, all become records in the live transaction stream to the Kafka broker. This log-based CDC approach is faster and has a much lower impact on production workloads than alternatives such as query-based CDC, which periodically queries the source database itself to identify recent changes.

Qlik Replicate also minimizes impact by eliminating the need for software agents on the source system, operating instead on a separate standalone server. Finally, by capturing only incremental changes, Qlik Replicate CDC reduces the bandwidth requirements of data transfer as compared with batch replication, which is especially useful for publication to cloud-based streaming systems. (Check out Chapter 4 for more on CDC.)

Automated data type mapping

Qlik Replicate can automatically map data types from many heterogeneous sources into a single consumption format. In the case of Kafka, Qlik Replicate can map to the Avro format, which serializes data in a compact binary format for downstream usage by Kafka consumers. Data engineers can greatly reduce their administrative burden and project time by configuring these data type conversions with Qlik Replicate's automated, drag-and-drop interface instead of mapping and scripting them individually for each source type. This process makes it faster and easier to generate topic streams that use a single established structure for use cases such as event processing, machine learning, and microservices integration.

Fan-out capabilities

Enterprises often have multiple uses for production transactional data. The same customer purchase table might need to be consumed real-time by dozens or more systems for fraud prevention, next best offer recommendations and supply-chain optimization.

Qlik Replicate enables data engineers to automatically provision a single database producer, once, that Kafka can then map to many streams and consumers, eliminating the need for duplicative configuration processes. These fan-out capabilities have the added benefit of minimizing impact on production workloads.

Metadata integration

Qlik Replicate CDC automatically propagates source schema changes to any supported target, including Kafka. Data engineers ensure that consumers are synchronized with all relevant source structures. For example, Qlik Replicate automatically captures and propagates all the metadata associated with source schema/DDL changes. This includes database level metadata, for example, to correctly identify date or time fields that databases may not otherwise make available to Kafka running on the Kafka Avro and JSON formats.

REMEMBER

Users have two options for metadata integration:

- **Inject all source metadata directly into the stream.** Users do this with an Qlik Replicate "envelope" that encapsulates all metadata, including message type, headers, and schema, for decoding by various consumers.
- **Integrate with the Confluent Schema Registry.** This integration stores and continuously updates schema versions for consumers through a RESTful interface.

Both these approaches ensure that Kafka consumers fully integrate with and incorporate source metadata updates. For more on Confluent Schema Registry, see the earlier section "Metadata integration and control" in this chapter.

Build scalable, real-time data pipelines

Use Apache Kafka® to stream unbounded data, and leverage change data capture (CDC) database replication technologies to enable a platform that supports real-time transactional database streams. These transactions are growing, and this book helps you explore how Kafka supports critical database transaction use cases and how to implement these in real-time streaming solutions.

Inside...

- Explore Kafka
- Discover transaction streaming with Kafka
- Apply CDC to Kafka
- Realize stream processing benefits
- Learn effective planning methods
- Consider Confluent and Qlik solutions

CONFLUENT™

Qlik

Thornton Craig is a data veteran of 20+ years, currently at AWS, previously at Gartner. An analyst at heart, **Kevin Petrie** runs field training at Qlik and writes for Eckerson Group. **Tim Berglund** is a teacher, author, and technology leader at Confluent, serving as Sr. Dir. of Developer Experience.

Go to Dummies.com™
for videos, step-by-step photos, how-to articles, or to shop!

ISBN: 978-1-119-62601-5
Not For Resale

for dummies®
A Wiley Brand

Also available as an e-book

WILEY END USER LICENSE AGREEMENT

Go to www.wiley.com/go/eula to access Wiley’s ebook EULA.