CONFLUENT

Whitepaper

A TECHNICAL GUIDE

How to Optimize Data Ingestion into Lakehouses & Warehouses

Introduction

In the rapidly evolving landscape of data management and processing, the concept of "shifting left" has emerged as a key strategy for organizations that want to enhance the efficiency, reliability, and quality of their data operations. Shifting left is both a strategic and technical endeavor, requiring a deep understanding of the underlying principles and methodologies. At its core, shifting left involves proactively cleaning and governing data closer to its point of origin, ensuring higher data quality and faster delivery of valuable insights.

This white paper will equip readers with the knowledge and tools needed to kickstart their shift left initiatives. Through a detailed exploration of the technical components of a data streaming platform and practical guidance on data governance and quality control, readers will gain insights into the transformative potential of shifting left. Whether you're a data engineer, an analytics engineer, or a decision-maker, this resource will help you gain a solid understanding of the shift left concept and guide you through the practical steps for its effective implementation.

Why Shifting Left Is Important

Data is the lifeblood of a company. Every bit of data that's stored—an order, a payment, a customer, a shipment, an interaction—represents an important aspect of the business. Initially, data is stored in the systems that created it, residing on a disk somewhere, purpose-built, and modeled to serve the immediate business use case. These transactional systems, which are part of the operational plane and populated by online transaction processing (OLTP) systems, are extremely efficient at serving their specific business use cases. However, business requirements expand far beyond handling transactions. Some questions require collecting and analyzing data from across the organization in data warehouses, data lakes, or data lakehouses—questions such as "Are we making money? Is our product strategy succeeding? Where should we focus our efforts in the next quarter and the next year?" The analytics plane, home to online analytical processing (OLAP) systems, focuses on evaluating business success, providing actionable insights, and conducting multipurpose analyses of historical data.

There is, however, a significant rift between the teams operating in the operational and analytical planes. Access to operational data is crucial for computing analytics, yet data practitioners in the analytics plane often have to devise their own means of access. Data pipelines, such as the extract-transform-load (ETL) and the extract-load-transform (ELT) processes, have long been the primary methods for transferring data into the analytics plane. Data or analytics engineering teams extract data from the OLTP database, model the data they need (transform), and then push it to the destination data lake or warehouse (load). Despite being traditionally brittle and expensive and requiring ongoing maintenance, ETL pipelines remain in use due to the significant need for data access outside the source.

The challenge is compounded by the fact that those in the analytics domain have little to no control or involvement over the source data model, which is managed by the application developers in the operational domain. Evolution and changes in the application space often lead to frequent disruptions in the data pipelines, causing break-fix work, delayed results, bad data, and dissatisfaction among customers. This problem stems from a mix of social responsibilities, technological limitations, and outdated conventions based on legacy tools.

The analytics plane is not the only area that requires access to high-quality business data. In the era of decoupled cloud computing, microservices, managed SQL engines, and software-as-a-service (SaaS) applications, there is an unprecedented demand for high-quality business data across organizations, including their operational layers. Operational systems often need access to the same data as analytical systems but can't use the same ETL/ELT data pipelines. This has led to the emergence of yet another flavor of batch pipelines—reverse ETL. The advent of technologies like generative artificial intelligence (AI) has intensified the need for data access across businesses. From tuning models to providing query context, easy and reliable access to well-formatted data is crucial for achieving excellent results.

Both operational and analytical use cases face a common challenge: the inability to reliably access relevant, complete, and trustworthy data. They are left to independently devise their means, periodically querying APIs, databases, and SaaS endpoints. While ETL pipelines might partially solve data access for analytics, a RESTful API might accommodate some operational data requests. However, each solution requires its own implementation and maintenance, leading to duplicative efforts, excessive costs, and slightly different datasets.

A better way to make data accessible to the people and systems that need it—whether for operational, analytical, or other purposes—involves rethinking the outdated yet still prevalent ETL patterns, the costly and slow multi-hop data processing architectures, and the "every man for himself" mentality that dominates data access responsibilities. It requires not just a change in thinking but also a transformation of where data processing is performed, who can use the data, and how it's implemented—in essence, a shift left.

What is Shift Left?

The essence of shifting left is straightforward. It involves taking the tasks of extracting, transforming, and remodeling data from your downstream systems, such as your analytics plane, and moving them upstream closer to the source. By doing this, all downstream consumers can benefit from a single source of well-defined and well-formatted data, ensuring consistent, reliable, and timely access to crucial business information. This makes data a first-class asset for powering services, analytics, and AI capabilities.

Shifting left significantly reduces complexity and costs. This strategy eliminates the need for duplicate data pipelines, processing, and storage. It also removes the necessity for parallel maintenance efforts, eradicates the need for fixing broken data pipelines, and allows engineers to focus on more valuable tasks. Since the data cleanup, standardization, and structuring efforts are front-loaded, data can be reused across a business rather than being confined to siloed locations downstream.

In practice, shifting left might be as straightforward as moving SQL cleanup code, which is run in the data lake, closer to the source system that originally generated the data. In other situations, more effort might be needed, both to identify the most valuable data and to implement the shift left strategy. However, the principle remains: Move the data cleanup, structuring, and transformation operations as close as possible to the point of data creation, allowing everyone to benefit from well-prepared data.

Shifting left is not an "all or nothing" approach. Not every piece of data needs to be moved upstream, and there are many small, iterative steps you can take to achieve this goal, gaining value along the way. Techniques and strategies are discussed in **How to Shift Left — Best Practices** of this white paper.

Challenges of Traditional Data Pipeline Approaches

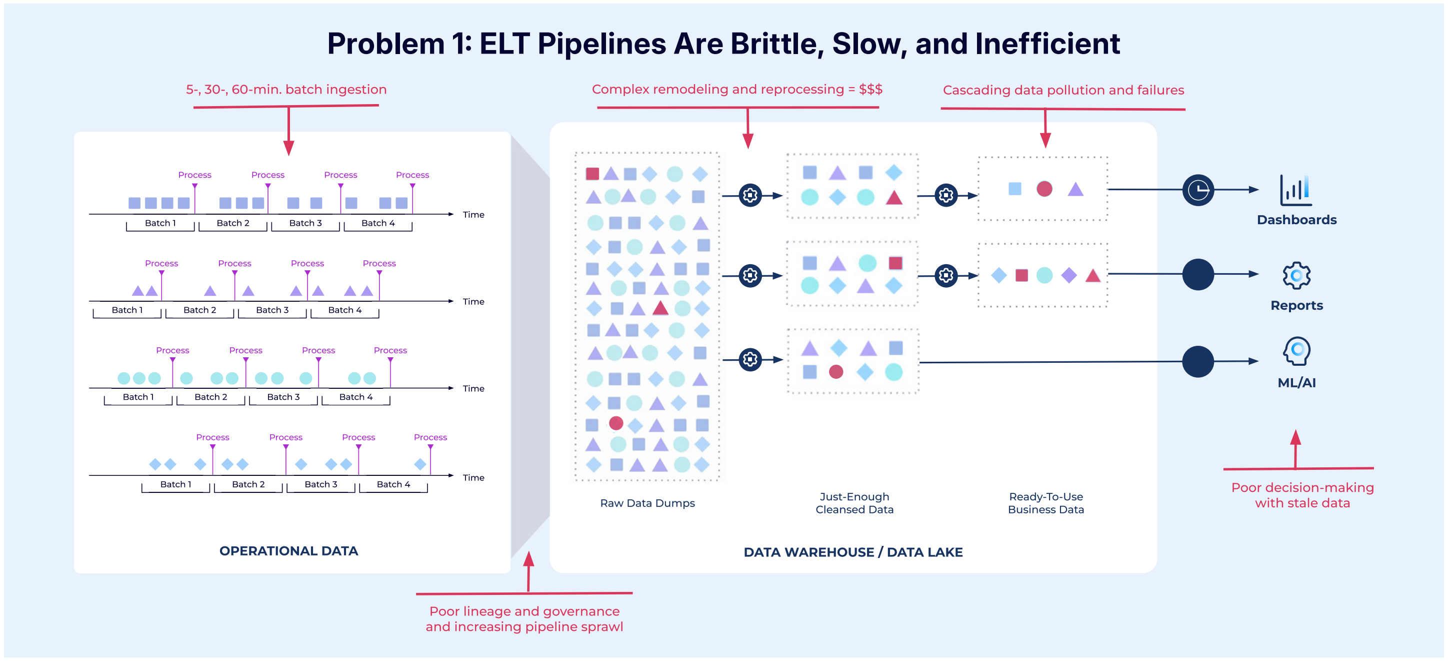
The divide between analytical and operational settings presents a significant data challenge for many companies. The operational plane, characterized by applications and services optimized for low-latency transactions, is where most data is initially created and stored. This data is then readily available for other operational systems to query and update. However, the issue for analytical purposes arises because the data models and database technologies are optimized for OLTP purposes, which does not lend itself well to OLAP. OLAP typically involves complex querying and characterizations of large quantities of data, which OLTP systems are not designed to handle.

On the flip side, data analysts, scientists, and engineers need to query large sets of data, including historical data and data from various systems across the organization. Their work commonly involves large, high-latency queries, which depend on optimization techniques specifically innovated for analytics. A prime example of such an optimization is the columnar data model, such as Apache Parquet™, which stores and accesses data by columns instead of rows. This model significantly speeds up analytical queries that aggregate on a column value, such as calculating the sum of all values in the "paid_amount" column, by requiring the loading of only the relevant column of data.

In contrast, OLTP systems typically use a row-based model, where updates to a specific row are executed per row. While this model is much faster for the random read/write access that operational workloads demand, it's less efficient for analytical purposes.

Data teams (who work on OLAP systems) and operational teams (who work on OLTP systems) are often compartmentalized within their own segments of a company, with separate leadership, business objectives, and cost centers. This segregation can hinder collaboration across teams, making it cumbersome and slow. As a result, a flourishing ETL/ELT industry has developed to cater to the needs of isolated data practitioners. The premise is to build and manage personal pipelines, allowing for the transfer of data from the operational systems into the analytics domain.

The primary objective of ETL and ELT pipelines is to move data from the source systems into the analytical plane to facilitate analytics. In an ETL process, some initial transformations may occur between the extraction and load steps. Conversely, in an ELT process, data is pulled into the analytics plane in its original form from the source and then transformed once it has been imported. There are four inherent challenges with this.


Problem 1: ELT Pipelines Are Brittle, Slow, and Inefficient
5-, 30-, 60-min. batch ingestion
Complex remodeling and reprocessing = $$$
Cascading data pollution and failures
Process
Batch 1
Batch 2
Batch 3
Batch 4
Time
OPERATIONAL DATA
Raw Data Dumps
Just-Enough Cleansed Data
Ready-To-Use Business Data
DATA WAREHOUSE / DATA LAKE
Dashboards
Reports
ML/AI
Poor decision-making with stale data
Poor lineage and governance and increasing pipeline sprawl

First, when data is extracted and processed in batches, the result is low-fidelity snapshots, frustrating inconsistencies, and stale information. Any downstream use of that data is based on outdated inputs.

The second challenge is the cost and complexity of remodeling and reprocessing: Teams often create local versions of the data to which they can apply processing logic in order to meet various use case demands. But as data gets reprocessed over and over, there's a high cost—both in terms of compute and wasted hours maintaining different data sets that don't match. In addition, when data arrives in microbatches, the incremental changes often have to be stitched together with additional processing logic. This can be simple if the data comes from just one source or is used in just one downstream location, but that's not the case. So it's up to the engineer to figure out how to make these incremental updates everywhere that data is used—and that's incredibly hard to do.

The third challenge is data quality and trustworthiness. If the application changes or the schema drifts, the result is garbage data in the data warehouse or data lake. This means all the systems and applications that depend on this data are building on dirty data, and fixing it is usually a multistep, manual, and tedious process.

The fourth challenge is pipeline inheritance. The data engineer often inherits pipelines but no knowledge or documentation about them and no clear lineages. As a result, engineers are fearful of changing existing pipelines and instead prefer to add one more, which worsens the pipeline sprawl.

The reason it's so hard to unlock the value of data from data warehouses and data lakes is that tedious data preparation is required to get access to high-quality, trustworthy data.

Data is often stale and unreliable. When the right data can't be found, it's recreated (incorrectly) over and over, increasing compute costs, data quality issues, and maintenance complexity. A lot of time and resources are spent on acquiring and preparing data—and worst of all, customer experiences are being designed to rely on slow batch-based processes and unreliable data.

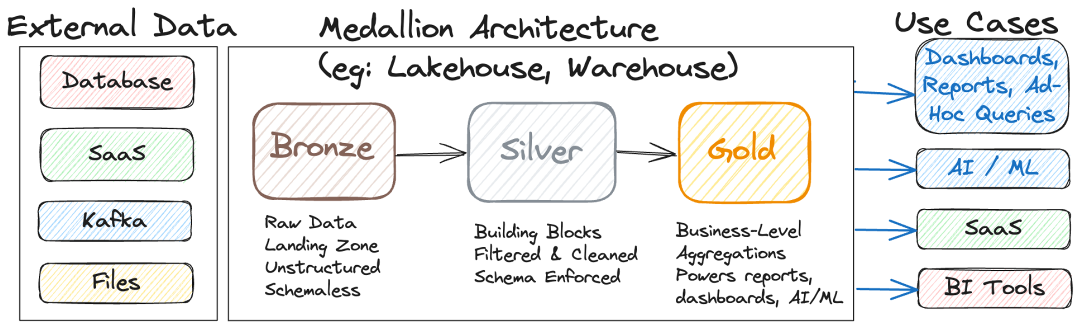
The complexity of acquiring and preparing data increases costs and impedes the ability to be more agile and innovative.

With a Shift Left approach, you can build data pipelines for multiple compute engines and destinations such as Snowflake, Amazon Redshift, Azure Synapse, Google BigQuery and Databricks Delta Lake. The implementation described in this document elaborates how to build data pipelines with data streaming to cloud data warehouses.

Understanding the Medallion Architecture

Before stepping into data cleaning and processing workloads, it's important to first understand how data is prepared for analytical systems. While there are multiple approaches for this process, the most common is a multi-hop medallion architecture.

The medallion architecture is a design pattern used to organize and delineate datasets. It is divided into three medallion classifications or layers, according to the Olympic Medal standard: bronze, silver, and gold. Each of the three layers represents progressively higher quality, reliability, and guarantees—with bronze being the weakest and gold being the strongest.



Bronze: The bronze layer is the landing zone for raw imported data sourced from Apache Kafka® topics, OLTP systems, SaaS endpoints, CSVs, XMLs, or even plain text files. The data commonly arrives as a mirror of the source data model but may also undergo minor transformations as part of the ETL process. Data practitioners then add structure, schemas, enrichment, and filtering to the raw data. The bronze layer is the primary data source for the higher-quality silver layer data sets.

Silver: The silver layer stores filtered, cleaned, structured, standardized, and (re)modeled data that is processed to a minimally acceptable level of quality. The silver layer provides well-defined data suitable for analytics, reporting, and further advanced computations. Silver layer data represents important business entities, models, and transactions that are then used as building blocks for calculating analytics, building reports, and populating dashboards. For example, it may contain datasets representing all business customers, sales, receivables, inventory, and customer interactions, each well-formed, schematized, deduplicated, and verified as "trustworthy canonical data."

Gold: The gold layer delivers business-level and application-aligned datasets that are purpose-built to provide data for specific applications, use cases, projects, reports, or data export. The gold layer data is predominantly denormalized and optimized for reads but still may need to be joined against other datasets at query time. While the silver layer provides building blocks, the gold layer provides the building from the blocks, cementing them together with additional logic.

Despite its broad adoption across different business, there are several significant downsides to the medallion architecture:

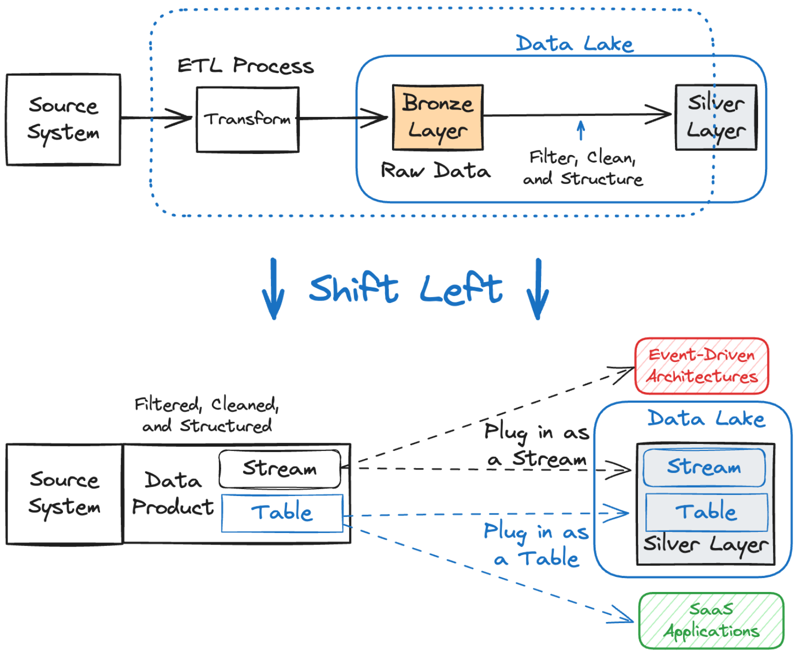
The clean, reliable, and well-formatted data in the silver layer is locked within the data lake or data warehouse, inaccessible to teams in the operational estate of the business or in other analytical systems.

The architecture is inefficient and expensive, with each step incurring costs—loading data, network transfers, writing data, and computing resources—that add up to a large bill.

Downstream consumers are responsible for understanding and cleaning raw data in the bronze layer, despite not having the same domain expertise and understanding as the upstream producer.

The architecture typically relies on batch processing frameworks, which increases the end-to-end processing latency and leads to analytical data becoming stale.

By shifting left the bronze to silver layers (i.e., out of the data lake or data warehouse), teams publish high-quality, reusable data products to be used by all consumers, whether operational, analytical, or something in between.



Think of a data product as a first-class data citizen, published with the rigor, control, and quality of any other product that you or your business creates. You'll be able to publish your own data products for others to use as well as rely on and use data products published by others.

Technical Overview of a Data Streaming Platform

Confluent's data streaming platform provides key capabilities for building data pipelines that shift the processing and governance of your data closer to its source. By migrating bronze to silver workloads away from the downstream analytical systems, teams can improve data portability and consistency across different platforms, optimize pipeline latency and data freshness, and reduce compute costs.

The Core Capabilities of the Data Streaming Platform

Stream

Reimagine data streaming everywhere—on-premise and in every major public cloud.

Connect

Make it easy to on-ramp and off-ramp data from existing systems and apps.

Govern

Make data in motion self-service, secure, compliant, and trustworthy.

Process

Drive greater data reuse with always-on stream processing.

Stream: The foundation of Confluent is built on Kora, our cloud-native Kafka engine. Kora provides the same Kafka protocol that has become the de facto standard for writing and reading data streams for thousands of organizations but without the operational burdens of self-managing the distributed system. Kora powers Confluent's cloud-native service to be 10x more elastic, resilient, faster, and cost-effective than open source Kafka and other cloud-hosted Kafka services. For customers with private cloud or on-premises environments, many of Kora's innovations are available in Confluent Platform and offer similar cloud-native attributes for self-managed workloads.

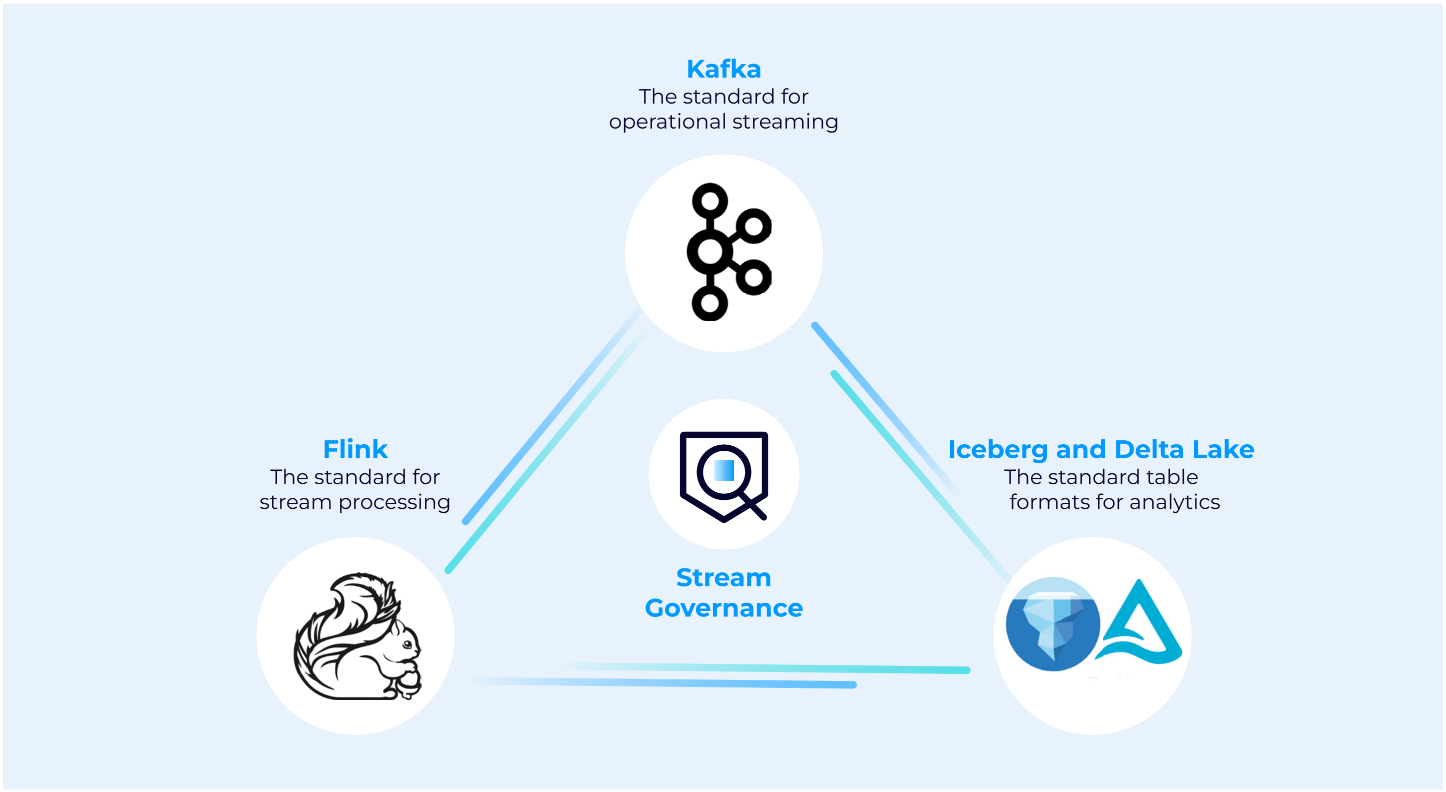
Connect: Confluent provides 120+ connectors built and maintained by data streaming experts to easily integrate popular data systems and applications in real time. With a rich ecosystem of source connectors, organizations can continuously capture real-time data streams from operational databases such as MongoDB and Oracle, SaaS providers such as Salesforce and ServiceNow, messaging queues such as AWS Kinesis and Azure Event Hubs, and more. After those streams are processed and tagged for proper governance controls, they can be shared via sink connectors to downstream data warehouses such as Snowflake and Google BigQuery, data lakes such as Amazon S3 and Azure Blob Storage, and other commonly used analytical systems.

Govern: Stream Governance is a comprehensive suite of features to maintain the quality of data flowing through pipelines and to minimize the data cleaning burden on each downstream consumer. Schema Registry ensures that proper schemas are enforced and evolvable as business requirements inevitably change, maximizing data compatibility between upstream sources and downstream sinks. Schema Registry is further bolstered with features such as Data Quality Rules to enforce semantically correct values (e.g., age must be an integer between 0 and 100) and establish data contracts that minimize silent breakage.

Process: Much like Kafka for data streaming, Apache Flink® has become the de facto standard for stream processing, enabling teams to run computations over their data in-flight. Common data cleaning tasks like deduplication, the dropping of unnecessary fields, denormalization, and more can be completed in real time with Flink rather than periodically running batch jobs with stale data. Confluent's serverless Flink offering is also broadly accessible to developers, supporting Java, Python, and SQL for all common processing steps. In the next section, we provide a deep dive on how Flink can implement common transformations done in analytical systems, including providing simple code snippets as examples.

The capabilities mentioned earlier lay the foundation for transitioning bronze to silver workloads. There is also Tableflow, which enables users to transform Kafka topics on Confluent into Apache Iceberg™ tables. These tables can then be directly accessed by a data lake, data warehouse, or analytics engines such as Trino, Presto, Spark, BigQuery, or Snowflake. Making operational data accessible to the analytical world is traditionally a complex, expensive, and brittle process. But Tableflow makes it push-button simple to better unify the two estates of your data architecture, significantly easing the process of accessing clean, real-time data for analytical workloads.

Think of a data streaming platform as combining three different open source standards—Kafka, Flink, and Iceberg—secured and unified within one platform and underpinned with a strong foundation of Stream Governance.


Kafka
The standard for operational streaming
Flink
The standard for stream processing
Iceberg and Delta Lake
The standard table formats for analytics
Stream Governance

How to Shift Left – Technical Implementation Using Apache Flink®

As organizations embrace the shift left paradigm and move data processing closer to the source, Flink has emerged as a critical tool for real-time stream processing. One of the primary tasks in this paradigm is transforming raw, unprocessed data from the bronze layer into cleaned, structured, and usable datasets for the silver layer. Flink's powerful stream-processing capabilities make it an ideal choice for this task, as it enables continuous, real-time transformations rather than relying on traditional batch-based processes.

In this section, we explore the key transformations that analytical teams perform within their Snowflake data warehouses or Delta lakehouses using processors that come with the challenges discussed in **What is Shift Left?** and **Challenges of Traditional Data Pipeline Approaches**. With a shift left strategy, these same transformations can be applied to data using Flink to create the silver layer. Each transformation will focus on a specific operation that can improve the quality, usability, and value of the data while it's still in flight, allowing downstream consumers to work with clean, reliable data immediately.

These transformations can be broadly categorized as:

- Preprocessing and cleaning
- Enrichment and denormalization
- Pre-aggregation

A streaming ETL pipeline usually consists of transformations from one or more of these categories. In the following sections, we look at each category in more detail and dive into practical examples.

Preprocessing and Cleaning

It all starts with data wrangling, the process of transforming raw data into a more structured form that is suitable for any further processing steps. The most common transformations that we see with customers shifting left include:

- Filtering
- Deduplication
- Normalization
- Array explosion (unnesting)
- Data type conversion (e.g., String to Timestamp)
- Data format conversion (e.g., JSON to Avro)
- Projections

In this white paper, we will dive a bit deeper into deduplication, array explosion, and normalization.

Deduplication

In most real-world systems, the source system or an intermediate component will occasionally introduce duplicate events or transactions into the data stream, usually as a result of partial system failures. Luckily, we can use Flink SQL to eliminate these duplicates right away, before they make their way to the downstream datastore. In the example below, which you can run for yourself as is on Confluent Cloud, we deduplicate the `clicks` stream based on `click_id`, keeping only the first occurrence of each of these events as measured by `$rowtime`. As shown, Flink automatically adds a `$rowtime` system column to every table, which contains the Kafka message timestamp and functions as the default **event time column**.

```
SELECT
      click_id,
      user_id,
      url,
      user_agent,
      view_time,
      $rowtime
FROM (
      SELECT
              *,
              $rowtime,
              ROW_NUMBER() OVER (PARTITION BY user_id ORDER BY
              $rowtime ASC) AS rownum
      FROM `examples`.`marketplace`.`clicks`
      )
WHERE rownum = 1;
```

Flink automatically deals with out-of-order clicks here by forwarding only clicks with the lowest **`$rowtime`** per **`click_id`**, regardless of the order in which they actually arrived and were processed. This is known as "event-time processing" and is crucial for accurate time-based analytics because it accounts for the possibility of late-arriving or out-of-order data.

Considerations and Remarks

When performing deduplication, you need to tell Flink how long you need it to keep historical events that it has already seen in its memory, for comparison to newly arriving data. Do this by setting what's known as a state time-to-live (TTL) configuration in the following clause: **`SET 'sql.state-ttl' = '<duration>'`**. In the example above, if you set a state TTL of 1 hour, Flink can guarantee that any duplicates processed by the query within the same hour will be detected. If you don't set this configuration, Flink will keep the state of these events in memory indefinitely, which can lead to higher costs and degraded performance.

Deduplication is currently supported only for input tables with **`'changelog-mode' = 'append'`**. Deduplication for updating tables is a work in progress.

By changing sort order from **`ASC`** to **`DESC`** in the query above, you can change the way you deduplicate the stream: Instead of always keeping the first click you see per **`click_id`**, keep the most recent.

Array Explosion

Unnesting an array into individual rows (also known as "exploding" an array), is an extremely common transformation in ETL pipelines. To illustrate, let's imagine a raw data stream called "**`visited_pages_per_minute`**" that arrives from the source in the following nested array format:

```
{
  "window_time": 1741161659999,
  "user_id": 3485,
  "urls": [
    "https://www.acme.com/product/wujjy",
    "https://www.acme.com/product/lnmfy",
    "https://www.acme.com/product/msryf",
    "https://www.acme.com/product/amvii",
    "https://www.acme.com/product/ifmip"
  ]
}
```

- To unnest the URLs with Flink SQL, use the **`CROSS JOIN UNNEST`** clause, as shown below.

```
INSERT INTO page_visits
SELECT
            v.window_time,
            v.user_id,
            u.url
FROM visited_pages_per_minute AS v
CROSS JOIN UNNEST(v.urls) AS u(url)
```

When you run the command, you get the following result from the page_visits stream: one URL per record, just as intended. The data is now much easier to work with for users and applications downstream.

```
{
  "window_time": 1741161659999,
  "user_id": 3485,
  "url": "https://www.acme.com/product/wujjy"
}
{
  "window_time": 1741161659999,
  "user_id": 3485,
  "url": "https://www.acme.com/product/lnmfy"
}
{
  "window_time": 1741161659999,
  "user_id": 3485,
  "url": "https://www.acme.com/product/msryf"
}{
  "window_time": 1741161659999,
  "user_id": 3485,
  "url": "https://www.acme.com/product/amvii"
}
{
  "window_time": 1741161659999,
  "user_id": 3485,
  "url": "https://www.acme.com/product/ifmip"
}
```

Data Type Conversion

Imagine a scenario where the leading big box electronics retailer Peak Purchase just acquired its biggest competitor, Voltage Village. The companies publish their orders in Kafka topics/tables, but each one does so slightly differently:

- Peak Purchase's table, **`orders_v1`**, tracks each order price in the **`price`** column with a **`DOUBLE`** data type, while Voltage Village's table, **`orders_v2`**, tracks each one in the **`price_usd`** column with the **`DECIMAL(10,2)`** data type.
- Peak Purchase tracks only the order type in the dedicated **`order_type`** column, while Voltage Village provides additional details as a semistructured JSON string in **`order_details`**.

The schema of each streaming table can be seen below. If you want to standardize the schemas of these two tables so that Peak Purchase can begin offering its shareholders unified sales reports across all of its stores, how might you go about it?

Schema **orders_v1**

order_id	STRING	NOT NULL
customer_id	INT	NULL
product_id	STRING	NULL
price	**DOUBLE**	NULL
order_type	**STRING**	**NULL**

Schema **orders_v2**

order_id	STRING	NOT NULL
customer_id	INT	NULL
product_id	STRING	NULL
price_usd	**DECIMAL(10,2)**	NULL
order_details	**STRING**	**NULL**

The following Flink SQL query brings both companies' order tables into a standardized form using **`UNION ALL`**.

```
INSERT INTO orders_norm
(
SELECT
  order_id,
  customer_id
  product_id,
  CAST(price AS DECIMAL(10,2)) AS price,
  JSON_OBJECT('order_type' VALUE order_type) AS order_details
FROM orders_v1
)
UNION ALL
(
SELECT
  order_id,
  customer_id,
  product_id,
  price_usd AS price,
  oder_details
FROM orders_v2
)
```

Considerations and Remarks

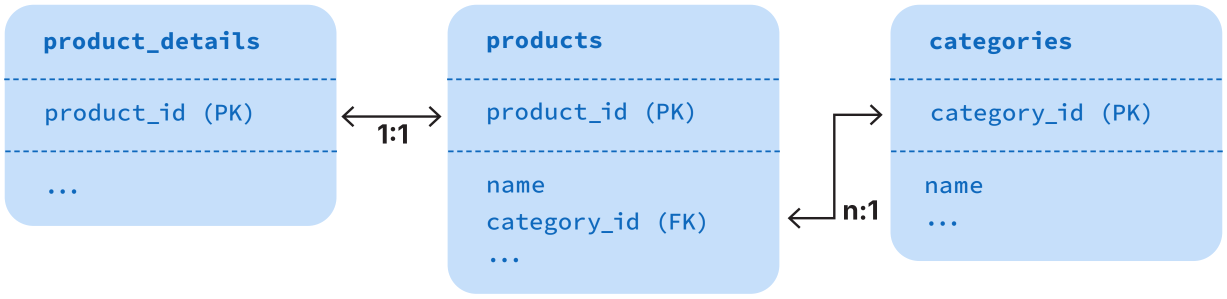
- **`UNION ALL`** does not guarantee ordering across the messages of the different unioned tables. If ordering by time is important, add **`ORDER BY $rowtime ASC`**.
- Flink SQL comes with an ever-growing set of built-in functions (e.g., for **handling date and time logic**, **working with JSON data**, or doing **string transformations**). And if that's not enough, Confluent Cloud for Apache Flink also **supports user-defined scalar and table-valued functions**.

Enrichment and Denormalization

Creating data pipelines for analytics that join together different datasets and tables to enrich them and add context is perhaps one of the most essential tasks that data engineers perform. With stream processing, every new data point may trigger the need for a new join, creating thousands of joins that need to occur incrementally and in real time as new data points flow in continuously. Flink is the perfect tool to use in scenarios like these because it can handle real-time joins with ease. Choosing the right type of join for Flink to perform, however, is absolutely critical for ensuring high performance, as different types of joins have drastically different performance characteristics and different semantics. With Flink, it's important to be familiar with the most common join patterns and know when to apply them, so in the next two sections, we share some key patterns.

Combining Reference Datasets With a Regular Join

Let's say you have three tables with product-related reference data: **`products, product_details, and categories`**.



As all of the tables are continuously updated, look for a single output table that contains the latest product information at any given point in time. Whenever each of the input tables is updated, you want the output table to be updated too. Using Flink SQL, you can accomplish this quite easily with a regular (inner, left/right/full outer) join:

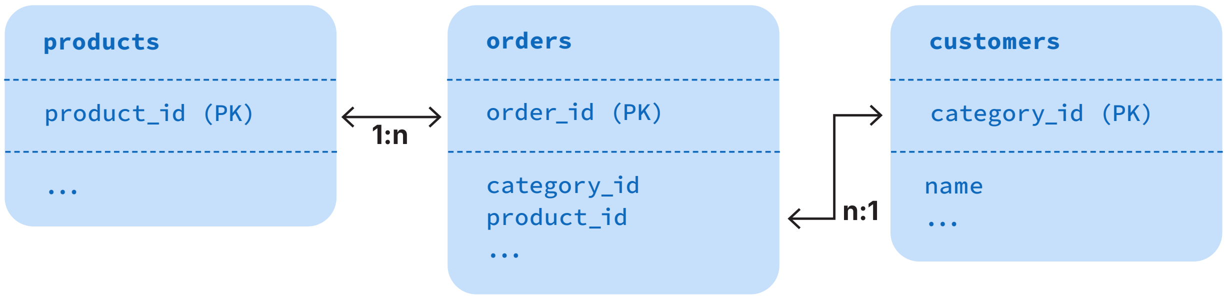
```
INSERT INTO products_full
SELECT
 *
FROM products p
INNER JOIN product_details d ON p.product_id = d.product_id
INNER JOIN categories c ON c.category_id = p.category_id
```

Considerations and Remarks

- Regular joins in Flink are not limited to primary key columns or columns mapped to the Kafka message key. Any column can be used as a join key.
- The state requirements, and ultimately cost, of a regular join are proportional to the cardinality of each of the input tables. This means that if any of the input tables grows quickly and without bounds, you'll have to **configure a state TTL**. In the example above, this is not the case, because the number of products and categories is not huge, and they're not growing quickly (in contrast to a join involving a fast-growing fact table as seen earlier with **`transactions`**, **`orders`**, or **`clicks`**, for example).

Enriching a Fact Table With Slow-Changing Dimensions With a Temporal Table Join

In contrast, let's say you have three tables: **`orders`**, **`products`**, and **`customers`**.



The streaming **`orders`** table contains one row per order and is append-only. Once an order is recorded, it's never updated. On the other hand, the streaming tables **`products`** and **`customers`** serve as reference tables that contain details about the products being ordered and the customers ordering them. These tables have a one-to-many relationship with the **`orders`** table, since one customer can make multiple orders and one product can appear in multiple orders. Unlike the **`orders`** table, these two tables are updated continuously as product details and customer information changes in the business's systems of records.

This presents an interesting challenge for real-time data enrichment: joining dynamic tables with static, append-only tables. First, you want to enrich **`orders`** with the customer and product information *as of the time the order occurred*—not as it exists now. You *don't* want changes to the pricing or customer information tables—that occur after a sale happens—to end up changing the order data after the fact, giving incorrect order information. Customers would be upset if they bought a product for a discount during a flash sale, only to see on their invoice that they paid the regular retail price.

Luckily, Flink is well-equipped for the job. You need a so-called **temporal table join**, with its characteristic **`FOR SYSTEM_TIME AS OF`** syntax. As opposed to a regular join, here you want only new rows in the **`order`** table to trigger new output rows.

```
INSERT INTO orders_full
SELECT
 *
FROM orders o
LEFT JOIN customers FOR SYSTEM_TIME AS OF o.`$rowtime` c ON o.customer_id = c.customer_id
LEFT JOIN products FOR SYSTEM_TIME AS OF o.`$rowtime` p ON o.product_id = p.product_id
```

Considerations and Remarks

In a temporal join, the join key needs to be the primary key of the temporal table (here, **`customer_id`** has to be the primary key of **`customers`**, and **`product_id`** has to be the primary key of **`products`**.)

The left side of the join (here, **`orders`**) can be either an append-only table, as in this case, or an updating table.

- Temporal table joins are highly efficient as they can purge state as time progresses. Flink excels at use cases like these.

Pre-Aggregation

By strategically shifting left the data processing, you can save lots of time and money that you would spend dealing with the storage and compute costs and difficulties of performing post-processing to the right/downstream. Continuous, incremental pre-aggregation of events prior to landing data in the data warehouse is the most effective way to significantly speed up query processing and reduce processing compute costs in the data warehouse itself. The most common forms of pre-aggregation are **time-windowed aggregations** and **sessionization**.

Time-Windowed Aggregations

Using time-windowed aggregations, events/facts can be pre-aggregated within time windows, and only the summary data for each time window will be written to the data warehouse.

Let's assume that, in the downstream data store, users are running analytical queries over orders with a known set of dimensions (such as **`category`**, **`country`**, **`currency`**, and **`payment_method`**) and that the required time resolution is 5 minutes. This means that you can already pre-aggregate the orders on the fly and thereby massively reduce the storage and processing costs in the datastore, as follows:

```
INSERT INTO orders_5m_agg
SELECT
 window_time,
 category,
 country,
 currency,
 payment_method,
 COUNT(*) AS num_orders,
 SUM(order_value_usd) AS total_order_value_usd
 COUNT(DISTINCT product_id) AS num_distinct_products_ordered
FROM TUMBLE(TABLE orders, DESCRIPTOR(´$rowtime`), INTERVAL '5' MINUTES)
GROUP BY
  window_start,
  window_end,
  window_time
  category,
  country,
  currency,
  payment_method;
```

Considerations and Remarks

- Flink SQL automatically accounts for out-of-orderness by firing windows only once the watermark has reached the window end time. Use the watermark strategy to trade off between the delay after which windows are fired and the completeness in the presence of out-of-orderness and late data.
- Besides **tumbling windows**, Flink SQL also supports **cumulating** and **sliding time windows**.

In addition to built-in aggregation functions such as **`AGG`** and **`SUM`**, user-defined aggregate functions are coming soon to Confluent Cloud for Apache Flink. Stay tuned!

Sessionization

Sessionization is the process of aggregating a group of lower-level events into a higher-level logical session based on some time- or event-based criteria. Web sessions are a series of interactions, such as clicks, data entry, and other interactions with a website that happen over a discrete period of time. By choosing the period of time that you're interested in, you can connect what would look like independent events into connected events that are all part of a single session.

The easiest way to assign events into a session is by using session windows. These are non-overlapping windows of time that are defined by a period of user inactivity between sessions—the so-called session gap. Here is an example, using session windows with a session gap of 1 minute:

```
INSERT INTO sessions
SELECT
  user_id,
  window_start AS session_start,
  window_end AS session_end,
  ARRAY_AGG(url) AS visited_pages
FROM SESSION(TABLE clicks PARTITION BY user_id, DESCRIPTOR(´$rowtime`),        INTERVAL '1'
MINUTES)
GROUP BY
  window_start,
  window_end,
  window_time,
  user_id
```

An alternative approach for sessionization is to use the **`MATCH_RECOGNIZE clause`**, which allows you to define specific conditions for the beginning and end of a session, such as the occurrence of dedicated "start" and "end" events. Flink also supports this method.

How to Shift Left – Governance and Apache Iceberg™

When implementing a shift left strategy with Flink, maintaining data integrity and ensuring high-quality data flow are essential. The following practices can help organizations enforce governance and quality control.

Data Lineage and Traceability

Establish clear data lineage tracking to ensure that every piece of data can be traced from its source to its final destination in the pipeline. Confluent's Stream Lineage and Schema Registry integrate seamlessly with Flink, enabling tracking of transformations, schema changes, and data flow. This traceability not only ensures compliance but also improves transparency and aids in debugging data anomalies.

Enforcing Schema Validation and Data Quality Rules

As data streams evolve, it's critical to ensure schema validity and data quality to prevent inconsistencies and to ensure that downstream consumers receive reliable data. Confluent's Schema Registry ensures that data complies with registered schemas by validating the schema ID before it's published to Kafka topics. Additionally, Data Quality Rules within Confluent's Stream Governance suite allow organizations to enforce data contracts and implement rules—such as domain validation and event-condition-action rules—ensuring the integrity and consistency of data. This approach helps safeguard the pipeline against poor-quality or malformed data.

Real-Time Data Validation

Implement real-time data validation to detect and reject corrupt or incomplete data early in the pipeline. Flink's built-in capabilities for watermarking and event-time processing help maintain data integrity, even in the face of out-of-order or late-arriving data. By setting up validation checks—such as outlier detection, duplicate filtering, and format validation—within the transformation pipeline, organizations can ensure that only valid data enters downstream systems.

Leveraging Tableflow to Materialize Data Streams as Open Tables

Tableflow simplifies the process of feeding Kafka data into data lakes, warehouses, or analytics engines as Iceberg tables. This process, traditionally complex and prone to errors, becomes seamless with Tableflow's one-click solution for converting Kafka topics and schemas into Iceberg tables. By reducing redundant efforts, Tableflow enhances both operational and analytical data environments with real-time data products.

By embedding data quality and schema management closer to the data source, Tableflow supports the enforcement of schema validation and data quality rules earlier in the data stream, aligning perfectly with the goals of data governance in the shift left approach.

Moreover, Tableflow's integration into the data governance framework illustrates the practical application of shift left strategies. Using Confluent's Kora Storage Layer, Tableflow transforms Kafka segments into various formats, such as Parquet files, while a metadata materializer leverages Confluent's Schema Registry to automate schema mapping, evolution, and type conversions. This eliminates the need for manual mappings, which can often break with new data.

By enforcing data quality rules upstream, incompatible data is intercepted at the source, significantly improving data integrity throughout the pipeline. In the broader context of the shift left paradigm, Tableflow bridges the gap between operational and analytical data environments. It simplifies the data ingest pipeline, reduces complexity, lowers infrastructure costs, and ensures high data integrity and quality from the outset.

How to Shift Left – Best Practices

Adopting the shift left paradigm requires more than just a technical implementation of real-time stream processing. It necessitates a holistic approach that incorporates best practices to help organizations ensure data integrity, maximize operational efficiency, and unlock the full potential of real-time analytics.

Key Data Categories for Shift Left Implementation

Certain datasets are particularly well-suited for benefiting from the shift left approach, delivering the greatest value in terms of efficiency and data quality.

High-Demand Data

High-demand data includes information that is widely used across multiple systems and departments in an organization. This data is critical for core business operations and directly influences decision-making and actions. Examples include product details, orders, accounts, reviews, inventory, and payments in an ecommerce setting or packages, vehicles, drivers, and warehouses for a delivery service. High-demand data is often replicated across various locations to enable quicker access. For instance, large organizations might use ETL/ELT processes to duplicate data into multiple data lakes, warehouses, or SaaS platforms. This replication introduces significant overhead in terms of pipeline creation, maintenance, and network transfer costs while also increasing the risk of inconsistencies across different data copies.

Critical Data

Critical data encompasses information for which accuracy, reliability, and security are essential. Misinterpretation or mishandling of such data can lead to serious operational or regulatory issues. Key examples include sensitive financial data, personally identifiable information, banking transactions, legal contracts, and accounts receivable. Historically, criticality checks for data have been performed during the transition from operational to analytical stages, benefiting only downstream users. However, by shifting these checks, schema enforcement, and quality rules upstream, all consumers of the data—both operational and analytical—can benefit.

For example, inconsistencies between billing and payment data handled by operations and revenue aggregation data handled by analytics can erode customer trust. A unified approach to managing critical data ensures consistency and reduces the risk of such discrepancies.

High-Volume Data

High-volume data introduces its own set of challenges, particularly in ETL/ELT processes that often involve multiple stages of data replication. A typical "source to bronze to silver" pipeline results in multiple copies of the same data, which can lead to significant costs in terms of storage and network usage. Moreover, replicated data can become out of sync due to latency and differing synchronization intervals. By adopting a more efficient headless model, which reduces replication and maintains data quality in a single pass, organizations can cut costs and improve data consistency.

Fostering Collaboration Across Teams

A successful shift left implementation goes beyond technical changes. It also requires breaking down silos between teams involved in data production, engineering, and consumption. Here are some best practices for promoting effective collaboration.

Cross-Functional Teams

Building cross-functional teams that include data engineers, data scientists, business analysts, and domain experts is essential for aligning business goals with technical capabilities. Involving all relevant teams early in the process helps ensure that both business requirements and technical constraints are addressed from the outset, preventing misaligned expectations or unnecessary complexity down the line.

Common Data Vocabulary and Standards

To streamline collaboration, establish a common data vocabulary and adhere to standardized data definitions across teams. This ensures consistency in how data is understood, processed, and consumed. For instance, if multiple teams are working on customer data, defining the structure and meaning of customer-related fields (e.g., `customer_id`, `customer_name`) avoids ambiguity and reduces the risk of mismatches and misinterpretations.

Data-Driven Development

Encourage a culture of data-driven development where business teams and data engineers work together to define business logic at the start of the pipeline. By using tools such as Flink SQL or Flink’s Table API, engineers can design transformations that directly reflect business requirements. Regular feedback loops and joint working sessions ensure that business needs are met and technical challenges are addressed proactively.

Integrated Testing Environments

Create shared testing environments where both data producers and consumers can validate data transformations before they go live. For example, business analysts should be able to test the output of transformations in a staging environment to ensure that business-specific requirements—such as aggregations and enrichment rules—are properly implemented. This collaborative testing helps ensure that data products meet expectations before they’re deployed.

Feedback Loops and Iterative Improvements

Continuous feedback among all stakeholders is critical for ensuring that real-time data products meet evolving business needs. As data is consumed, business users can provide insights into how well it supports decision-making. Data engineers should be responsive to this feedback, making iterative improvements to the pipeline to adapt to changing requirements.

Collaborative Monitoring and Incident Management

Effective incident management requires collaboration across teams to ensure pipeline reliability. Implement a unified monitoring framework that provides visibility to all stakeholders—engineers, analysts, and business users—so that issues like data spikes, slow processing, or data loss can be identified and resolved quickly. Tools such as Prometheus and Grafana can be integrated with Flink to provide real-time insights and automated alerts, enabling a swift response to incidents.

Documentation and Knowledge-Sharing

Encourage thorough documentation and knowledge-sharing among teams. Detailed data schemas, transformation logic, and metadata should be accessible to everyone involved in the pipeline. A data portal can serve as a central hub, enabling teams to discover, explore, and interact with data in a secure, transparent manner. This ensures smooth collaboration and reduces friction when teams need to access and work with data.

By focusing on key data categories—such as high-demand, critical, and high-volume data—organizations can apply the shift left approach effectively across the pipeline. When combined with strong collaboration among teams, these practices ensure that data governance, quality control, and operational efficiency are maintained from the start, unlocking the full potential of real-time analytics.

Operational Use Cases Powered by Shifting Left and Data Products

As mentioned previously, one of the main advantages of shifting left is creating real-time data products that are usable across both operational and analytical workloads. By leveraging the same source of data, organizations can avoid the inconsistent and redundant processing that often occurs when each downstream team is responsible for implementing its own business logic.

Foundational business data, such as products, orders, payments, accounts, reviews, and inventory, are often used across multiple contexts within businesses, increasing the benefit that they might realize by creating a single universal data product that can be read by multiple teams. Here are some examples of data that's critical to both analytical and operational workloads:

- User reviews are key focus areas for analytical reports to understand how customer satisfaction measures across time and various touchpoints with a business. However, that same data can also be key to powering operational use cases, such as proactively intervening with customers submitting poor reviews. By using tools such as Flink AI Model Inference, you can call external language learning models to subjectively rate customer ratings and filter poor ones for downstream consumption by customer success representatives.
- Analyzing inventory levels over time is critical to optimizing the efficiency of your business and ensuring that orders can be served without tying up too much capital. Many of our retail customers use the same stream of data to power real-time inventory management use cases, automatically triggering orders for additional supply when inventory levels drop below sufficient levels.
- Payment information is used in the analytical domain to tally up revenue and report on core financial information of a business, which naturally requires a high level of correctness and reliability. Similarly, the operational domain uses similar data for denoting which outstanding bills can be closed out. Unifying these use cases to leverage a single common source of data eliminates potential discrepancies and redundant processing.

Customer Stories

Many organizations are embracing a shift left approach to modernize their data architectures, drive real-time insights, and optimize their decision-making processes. By integrating data processing earlier in the pipeline, businesses can streamline their workflows, improve operational efficiency, and unlock new opportunities for growth. Here are two real-world examples of how companies across different industries have successfully leveraged a data streaming platform to accelerate data processing, enhance customer experiences, and achieve measurable results.

American Medical Equipment Company Revolutionizes Respiratory Care With Data Integration

An American medical equipment company dedicated to providing cloud-connectable medical devices for sleep apnea, COPD, and other respiratory conditions faced the daunting challenge of overhauling its data strategy. With the ambitious goal to impact 500 million patients by 2030 and bring AI innovation to the forefront of its services, the company encountered significant hurdles. These included managing data from thousands of disparate sources generating billions of records daily, slow and cumbersome batch-based ELT pipelines, data quality issues, high data warehousing costs in Snowflake, and the inability to leverage AI/machine learning due to the lack of clean, accessible data.

In response, the company embarked on a journey to transform its data management practices by adopting a unified data platform powered by Confluent Cloud. This strategic move involved using fully managed connectors for MongoDB, Snowflake, S3, and more and implementing a Schema Registry from the outset to ensure data quality. By shifting left processing to the stream—including tagging customer IDs across 10 billion daily transactions and cleaning testing data in real time—the company could streamline device certification and display real-time manufacturing dashboards. This approach not only alleviated previous bottlenecks but also set a new standard for operational efficiency and real-time data utilization in the medical device industry.

Key Benefits:

- 40% increase in manufacturing floor productivity through real-time dashboards
- 35% anticipated savings on Snowflake data warehousing costs
- Capability expansion from 30–40 to 1,000 integrations per year
- 95% of workloads streamlined onto the new platform within two years

Online Marketplace Drives Personalized Recommendations With Apache Flink

An online marketplace serving 30 million customers, offering a platform where users create and sell customized products, adopted Confluent Cloud for Apache Flink to optimize its primary data pipeline: clickstream data. The company recognized the critical need to understand user behavior—specifically, their clicking patterns on the website—to provide tailored product recommendations and boost sales.

Previously, redundant datasets created challenges during data lake analysis. By moving to Flink, the company minimized data duplication and enabled the processing of clickstream data closer to its origin. This shift to real-time stream processing allowed the marketplace to gain immediate insights into product popularity and improve the relevance of its recommendations, directly contributing to increased sales and revenue. The company now plans to scale its data pipeline significantly, to increase clickstream data processing by up to 100 times, and to add new data streams, such as customer orders, to the Flink pipeline.

Key Benefits:

- Eliminated redundant data and optimized data processing
- Gained real-time insights into user interactions and product popularity
- Enhanced customer personalization, leading to higher sales and engagement
- Scaled data pipeline to support future growth and additional data streams

These customer stories highlight how adopting a shift left strategy with Confluent Cloud for Apache Flink has empowered organizations to optimize their data processing pipelines, improve operational efficiencies, and drive significant business outcomes.

Conclusion

Implementing the shift left paradigm marks a fundamental change in how organizations approach data integration, governance, and quality control. By bringing data processing and governance closer to the source, organizations can unlock real-time, reliable data to drive insightful business decisions. This white paper explores the key elements and technical components necessary for successful implementation—from understanding data layers in a medallion architecture to the specific Flink transformations required for data refinement. The introduction of Tableflow as a seamless integration tool for streaming data into analytical systems further highlights the potential for operational and analytical unification. Together with real-world customer examples, these discussions provide a comprehensive road map for organizations that want to leverage shift left strategies to enhance their data pipelines.

Embracing the shift left approach offers a clear path to streamlining data processes, improving data quality at the source, and fostering collaboration among all stakeholders in data management. The best practices outlined in this paper provide actionable guidance for organizations navigating the complexities of shift left implementations. As data environments continue to evolve, the principles and techniques shared here will serve as a valuable resource for those seeking to optimize their data ecosystems for speed, efficiency, and integrity. While the journey to more agile, real-time data processing can be challenging, the benefits—improved data quality, reduced costs, and enhanced decision-making capabilities—far outweigh the obstacles.