



WHITEPAPER

Choose The Right Stream Processing Engine For Your Data Needs

Technical and operational factors that are crucial to the decision making process

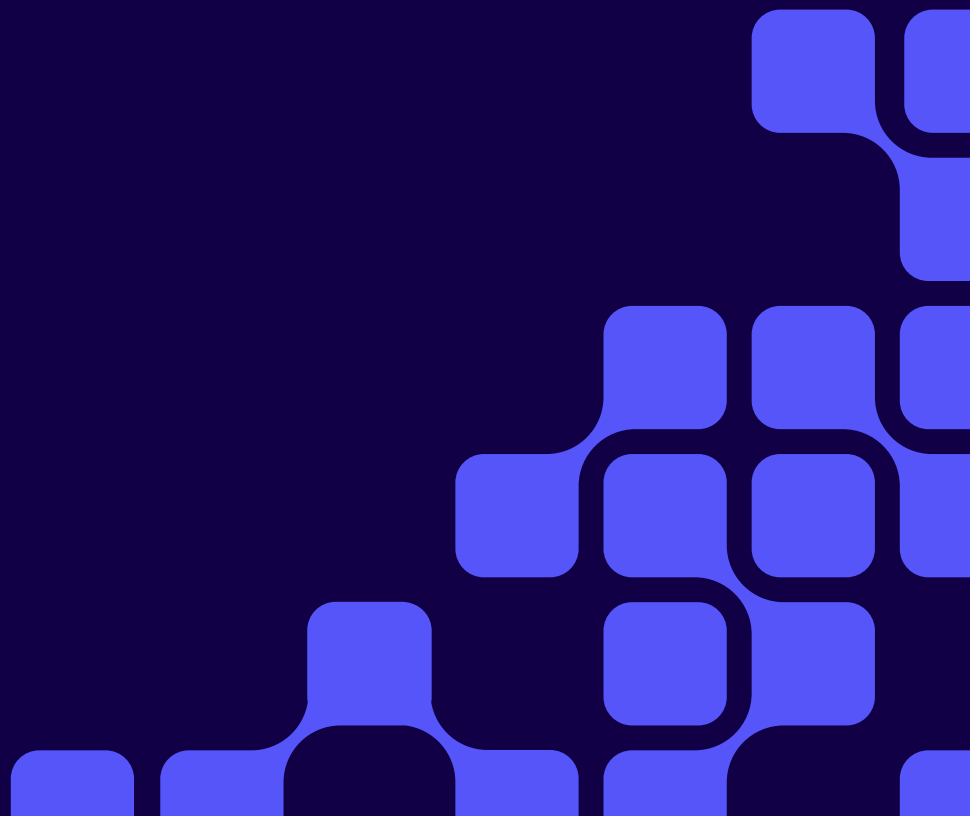
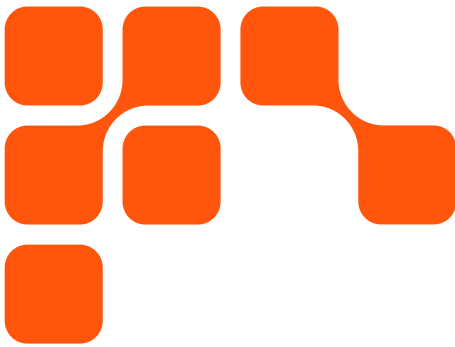


Table of Contents

Process Data Streams at the Speed of Business and at the Scale of IT	3
Address Challenges Through Informed Decision Making	4
Streaming Challenges	4
Decision Making Process: Technology and Operational Considerations	5
Technology Considerations for Stream Processing Engine Evaluation	5
Functional Aspects	5
Developmental Control	6
Implementation and Beyond	6
Operational Considerations for Stream Processing Engine Evaluation	8
Enterprise Adoption	8
Enterprise Operations	8
Spark, Kafka, or Flink? Which to Use?	9
Flink Use Cases	10
Cybersecurity and Log Processing	10
Outage Classification for Telecom Companies	11
Financial Services: Mainframe Offloading	11
IoT for Manufacturing	12
Fraud Monitoring for banking	12
Technical Features Table	13
Operational Features Table	14
Customer Success	15
Ensure Fit for Purpose and Enterprise Wide Adoption	15
About Cloudera	16



Process Data Streams at the Speed of Business and at the Scale of IT

Business opportunities that directly impact revenue or boost operational efficiency need to be addressed in near real-time. Digital transformation initiatives and the advancement in mobility, IoT and streaming technologies has led to enterprises being inundated with data. Key business requirements determine how such high volumes of high-speed data should be processed in real-time to provide actionable intelligence. This directly leads to IT having to evaluate which stream processing engine is best fit for purpose for their enterprise needs. Other determining factors include return on investment, its dexterity to be applied across multiple use cases, and its level of maturity for enterprise-wide adoption.

This paper is meant to help technology architects and developers choose the right stream processing engine for their needs. We do this by analyzing key technical and operational differentiators between three modern stream processing engines from the Apache open source community:

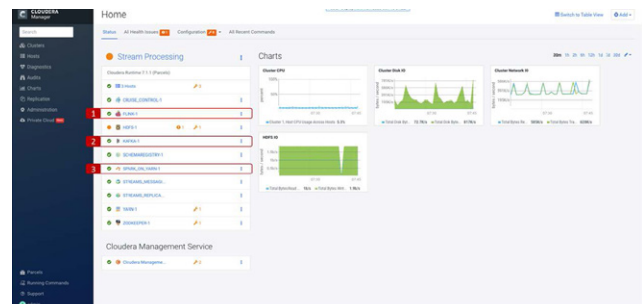
- Kafka Streams;
- Spark Structured Streaming;
- Flink.

This paper also highlights some of the capabilities that are key to any data streaming use case, such as:

- Watermarks to handle late and out of order delivery;
- Windowing semantics to structure the streams;
- Complex event processing; and
- Capabilities that enhance operational efficiency.

Cloudera offers all of the engines listed here, because we believe that you should use the best tool for the job. Sometimes that tool is a very simple one, but more often than not, you will need the advanced capabilities for your specific use cases.

There are a variety of ways by which to address data stream processing challenges. The solution comes down to the fundamental way in which the engine works and how your organization implements it.



Cloudera Manager provides one view to manage all of your resources including stream processing engines. Here we see Flink (1), Kafka (2), and Spark (3) resources in one comprehensive view. Source: Cloudera.

Address Challenges Through Informed Decision Making

Global digitization has resulted in a vast array of new products and services with such high levels of convenience that it fuels a continuous loop of greater expectations for immediacy. Next day delivery and real-time payments are demands driven by consumers at the point of service that then pressures downstream services to respond faster. Processing and analyzing billions of events per second across geographies is becoming an ordinary affair.

In response, technology teams have been pivoting from large monolithic database architectures to event-driven applications and microservices design as a way to reduce the inevitable latency of inputs and outputs across networks by bringing the state of an event closer to the application itself.

Central to this effort are modern data stream processing engines like Kafka Streams, Spark Structured Streaming, and Flink. Of these three, Flink is the oldest, while Kafka Streams is the newest. The Spark Structured Streaming community is large while Flink's is growing rapidly. Knowledge of an engine's development community can help gauge how self-sufficient and productive your team can be.

The engine that is best for you depends on your organization's use cases, team makeup, and various technology and operational factors. This paper is meant to help you in that evaluation process.

Streaming Challenges

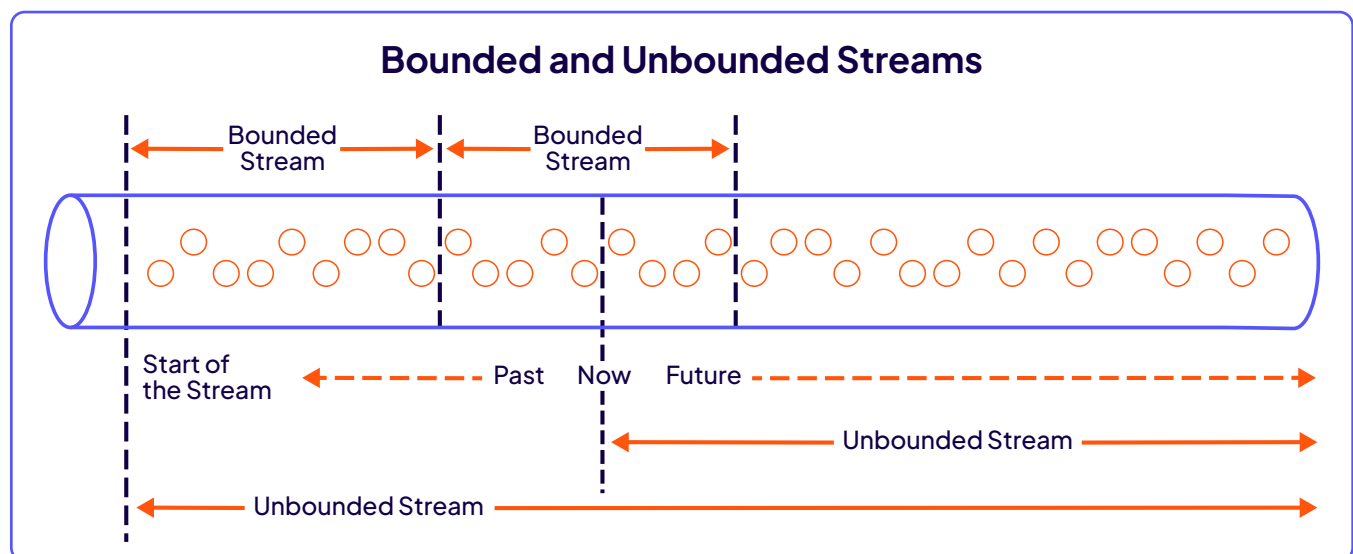
The following are reminders of streaming challenges you've undoubtedly had or will come across.

Event time and processing time — The chances that streaming events come in without any delays and with predictable patterns is low, because you can't control the myriad of input sources that exist across collections of networks that vary in type and quality. Even with the very best networks and the fastest collection mechanisms, there will always be latency between the time an event happens in the real world (event time) and the time your system processes it (processing time).

Bounded and unbounded streams — Bounded streams have a beginning and an end, so it is easy to reason about time and correctly sort events, akin to batch. Unbounded streams are harder to reason about because, without an end, you don't know if another live event is yet to come. Calculations, aggregations or pattern detection in unbounded streams is very tricky. To handle both scenarios, it is helpful to follow a "streaming first" principle (see sidebar next page), and to consider capabilities like watermarks to handle late and out of order events (see sidebar next page).

Simple and complex events — Complex events are derived from simple events that have been aggregated, patterned, and evaluated to trigger a response or present a result, often on data that continuously moves under your feet. Decisioning on unbounded streams requires the state of events to be stored and analyzed.

Stateless and stateful — Stream processing engines excel when analytics require a reassessment of events within the context of time. That is considered stateful, while stateless represents a self-contained fire-and-forget paradigm. There are acceptable trade-offs between stateless high throughput engines and stateful engines that need to address aggregation, enrichment, and other requirements.



Decision Making Process: Technology and Operational Considerations

There is often an over reliance on streaming benchmarks when choosing a stream processing engine. These benchmarks primarily concentrate on latency, throughput, and hardware utilization, neglecting functional requirements and the level of control developers possess when implementing a solution. Additionally, benchmarks often overlook crucial operational, staffing, and other nonfunctional criteria. The remainder of the paper outlines vital technological and operational factors necessary for making informed decisions and encouraging enterprise-wide adoption of the chosen solution.

Technology Considerations for Stream Processing Engine Evaluation

The next section compares the different stream processing engines with regard to functional, developmental, and implementation considerations.

Functional Aspects

The functional capabilities of stream processing engines as they pertain to approach, streaming model, and time support are used to solve specific business requirements.

Approach — The type of approaches that development communities took at the inception of an engine's development include: “streaming first”, “message broker first”, and “batch first”. The distinction helps in understanding what the engine was originally meant for.

Flink took a “streaming first” approach and is regarded as being the modern leader in this space.

Spark Structured Streaming followed a “batch first” approach, while Kafka Streams was initially developed as a “message broker first”. Streaming capabilities are popular add-ons for both.

Streaming Model — Earlier, we described the concept of “stateless” and “stateful” and that it is critical to distinguish between the two, and the trade-offs between throughput and latency.

The “Streaming First” Approach

Stream processing engines have followed different paths in their approach to solving unique time reasoning challenges.

Flink is a “streaming first” distributed system. This means that it has always focused on solving the difficult unbounded stream use cases over bounded stream and batch scenarios.

It turns out that algorithms that work on unbounded streams, also work on bounded streams by treating the latter as a special case of the former. As a result, Flink addresses micro-batch use cases as well.

Watermarks To Handle Late Delivery

Watermarks offer a robust method for handling late or out-of-order arrivals by providing a set of trigger messages that are injected alongside the data stream.

For unbounded streams, in which you don't have a definitive end, watermarks delineate points at which you would expect all of the events to have occurred. It is from here that you can establish some logic.

A collection of watermarks forms windows, providing structure to your data streams and enabling the application of reasoning (refer to [Window Semantics](#) in the sidebar on page 7).

Flink provides extensive control over watermark generation. This provides more options as to how completely you want to capture events that may or may not arrive. This mechanism can also extend to very sophisticated functionality like leveraging upstream and downstream materialized views or using batch engines to reprocess and incorporate late data.

Flink, Kafka Streams, and Spark Structured Streaming are all stateful, but with slight differences. Having taken a “batch first” approach, Spark Structured Streaming handles events as micro-batches and it excels when high throughput is necessary but low latency is not a big requirement. The two other stateful engines differ in how they store state. Kafka Streams depends on the Kafka ecosystem, while Flink provides more storage options. Both process messages one event at a time and are considered low-latency solutions.

Time Support — All of the stream processing engines described in this paper are able to distinguish event time from processing time. The nuance is in how much control you have to address some of the trickier use cases. Flink provides a great deal of control with capabilities such as watermarking and session windows (see sidebars on [page 5](#) and [page 7](#)).

Developmental Control

A common task in every data processing use case is to import data from one or multiple systems, apply transformations, and then export results to another system. Considering the ubiquitousness of streaming data applications, unified integration with machine learning, graph databases, and complex event processing is becoming more common.

Processing Abstraction — To help your engineering team be productively focused on business logic instead of advanced streaming concepts, it’s important to evaluate the stream processing engine’s abstraction capabilities.

Spark Structured Streaming has a rich set of libraries to implement machine learning use cases. If you are already developing within a Spark ecosystem with frameworks such as MLlib, choosing Spark Structured Streaming as your stream processing engine will make it for a much easier adoption.

Special attention should be paid to the engine’s SQL abstraction. From an analytics democratization point of view, SQL abstraction is a very important basis for comparison. While many senior developers prefer sophisticated languages like Scala for intricate analytical tasks, the expressiveness and simplicity of SQL can get the job done more easily and it is accessible to a wider range of developer, and even business, resources. This accessibility is a critical factor in enabling domain ownership of data apps.

When it comes to the comparison on the basis of SQL, the more standard the better. Flink has the most mature and production tested OpenSource SQL-on-Streams implementation and is fully ANSI compliant. Kafka’s ksqlDB, formerly known as KSQL, has matured signifi-

cantly and is now fully open-source. However it is not ANSI-compliant nor as feature rich as Flink’s offering. Spark Structured Streaming SQL, is well adopted, and has ANSI compliance dating back to 2003.

Implementation and Beyond

Application development is only as good as its implementation. Below, we cite aspects that need to be considered to move beyond the idea and development stage.

Delivery Guarantee — This is a key factor to consider as it relates to your expectations of latency, throughput, correctness, and fault tolerance of message delivery.

At-least-once delivery ensures that a message is delivered at least once, although multiple delivery attempts may result in duplicates. This approach offers high performance and minimal overhead since delivery tracking state is not maintained. Operating in a fire-and-forget manner, at-least-once delivery satisfies low latency and high throughput requirements while guaranteeing message delivery, but may not sufficiently address data duplication concerns.

Some applications, such as financial transactions, have stricter demands and may require that messages are received and processed exactly once. This requires retries to counter transport losses, which means keeping state at the sending end and having an acknowledgment mechanism at the receiving end. Exactly-once is optimal in terms of correctness and fault tolerance, but comes at the expense of added latency.

All the engines described in this paper provide exactly-once delivery guarantee, though Kafka Streams is limited to the Kafka ecosystem and can’t control downstream systems. Flink and Spark Structured Streams guarantee exactly-once delivery from any replayable upstream source but also with downstream platforms in some cases, if they have transactional support.

State Management — The aforementioned trade-off between exactly-once delivery guarantee and the inevitable latency of state storage may drive the selection process based on the state management capabilities that come with the engine.

Kafka Streams is good for things that are Kafka centric, because it tends to rely heavily on Kafka storage for state. Like Flink, it uses a local RocksDB but checkpoints the state as a Kafka topic and that limits flexibility as to how you store and access the history of that state. Within a Kafka ecosystem, a good linear access mechanism is provided, making everything nice and tame. This works great for simple use cases, but it doesn’t provide the flexibility and operational capabilities that some of the other engines do.

Apache Flink provides more complete state management capabilities compared to Kafka Streams, offering features such as native support for different state backends, efficient checkpointing, and built-in fault tolerance mechanisms. While both Flink and Kafka Streams can handle stateful stream processing, Flink's state management system is designed to scale across distributed environments seamlessly, allowing for automatic state redistribution and recovery during rescaling or failures. This makes Flink particularly well-suited for large-scale, complex, and state-heavy streaming applications that require high levels of fault tolerance and flexibility in state management.

Fault Tolerance/Resilience — The demand to mitigate operational disruption is so strong that the concept of “resiliency” attracts regulatory oversight across industries. Streaming architecture capabilities such as checkpointing, savepoints, redistribution, and state management (see above) are crucial to the stream processing engine selection process.

Spark Structured Streaming has built-in capabilities, while Kafka Streams requires you to “build your own”, using ZooKeeper to replace a failed broker for example.

Flink's fault tolerance mechanism uses checkpoints to draw consistent snapshots to which the system can fall back in case of a failure. The aforementioned state management capabilities ensure that even in the presence of failures, the program's state will eventually reflect every record from the data stream exactly once.

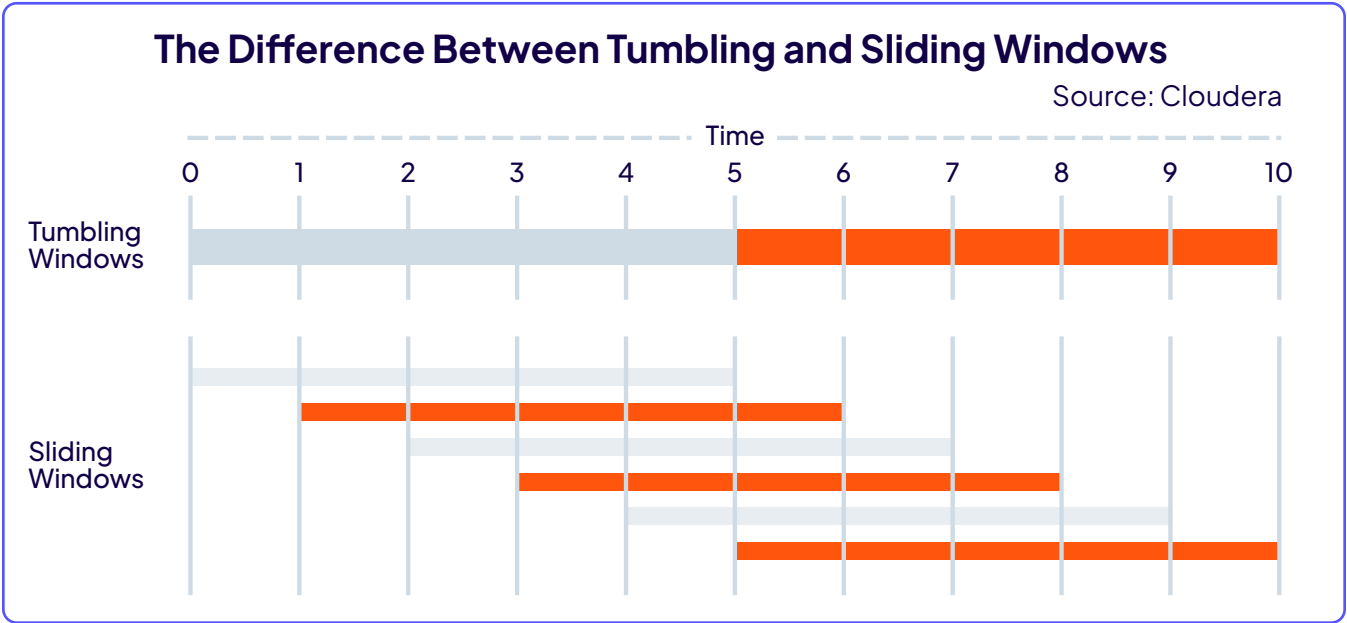
Leveraging Apache Flink's advanced checkpointing capabilities offers a multitude of operational advantages, such as seamless job versioning, effortless cluster migration, streamlined application and infrastructure upgrades, and the flexibility to transition workloads between cloud and on-premises environments. As a testament to its efficacy, this robust approach is increasingly being embraced by the wider stream processing engine community, further solidifying Flink's position as a trailblazer in stateful stream processing.

Window Semantics

Flink allows you to customize a window structure so you are not limited to pure linear time. By utilizing Session Windows, for instance, you can define windows based on gaps between events or the number of events. This offers considerable flexibility in assigning events to different windows before processing.

Windows are logically essential for analytics, as they provide the structure upon which analysis is based.

Two other windowing examples are tumbling windows (that slice a stream into even chunks) or sliding windows (that enable your aggregations and analytics to move with time).



Complex Event Processing (CEP)

Processing real-time events and extracting information to identify more meaningful events, like understanding behaviors, probably ranks as one of the most interesting streaming use cases.

Flink's statefulness and window handling capabilities is the foundation on which advanced CEP is crafted. What makes Flink all the more compelling is that CEP is accessible to a wider range of developer resources through standard SQL abstraction.

For example, the `Match_Recognize SQL` statement can be very helpful when you are looking for patterns built up through sequences of events that can't be distinguished by simple counting methods.

The standard SQL abstraction of Flink makes it a compelling choice for use cases that require "simple" complex event processing.

Operational Considerations for Stream Processing Engine Evaluation

All organizations look to control costs by doing more with less. Budget approval for your selected stream processing engine may be contingent on its utility across streaming pipelines, reusability of talent, and synergizing tech stacks.

Enterprise Adoption

The best application is no good if it can't be efficiently and safely deployed across your organization, or if there is a dependency on hard-to-find development talent. Effective solutions are those that can be adopted across the entire enterprise.

Deployment Model — There is a better chance of adoption if teams don't have limited deployment options. Flink can be deployed in clustered, Kubernetes, YARN, Kafka, HDFS, Docker, S3, and microservices environments, while Structured Streaming and Kafka Streams are more limited (see [Technical Features Table](#), page 13). Kafka Streams is the most lightweight for microservices, at the cost of out-of-the-box features, and the Flink library is nearly as lightweight.

Kafka Streams doesn't provide a scheduler or full deployment framework out-of-the-box. While it provides efficient ways of writing simple applications, you are left to your own devices on how to launch, run, orchestrate and operate those applications.

Community Maturity and Documentation — To ensure that your developer resources are self-sufficient and productive, the maturity of the developer community and quality of documentation is a very important aspect to consider.

Kafka Streams is relatively young with very strong community growth and extensive documentation and examples. While the Spark Structured Streaming community is large and busy with extensive documentation and examples, they still need help from more reviewers and committers, something that Cloudera is helping to drive forward.

Flink is the fastest growing community with strong research and production deployments. Documentation and working examples are good and will broaden considerably as the community matures. "For the 5 year period from 2017–2022 Flink has been the most active community in terms of code commits and community discussion". Also, some of the biggest brand name companies have already invested in large deployments of Flink for their real-time stream processing needs.

Enterprise Operations

If you are looking to establish a legacy of successful, fit for purpose data streaming solutions, it is important to know that you've selected an engine that completely integrates into your organization's security framework, provides comprehensive monitoring and metrics, and can scale up and down in line with business demand.

Enterprise Management — At Cloudera, we've invested a good deal of our time to integrate the stream processing engines described in this paper into [Cloudera](#) to make sure that they are all enterprise ready for their respective purposes. Both Flink and Spark Structured Streaming have rich Operations Support Systems (OSS) with enhanced vendor offerings. Kafka Streams has minimal OSS via some vendor offerings.

Over time, Cloudera has enhanced Spark Structured Streaming with important metadata, logging, metrics, and operational capabilities that are also starting to find its way to Flink but, for now, the former has the edge on Cloudera.




As it relates to Kafka, the original security implementation was done by Cloudera (via integration with Apache Ranger for role-based authorization and auditing) and still provides leading security capabilities via scale-tested role-based and attribute-based access control models, and integrated data governance with [Apache Atlas](#) in Cloudera.

The other advantage of Cloudera for these stream processing engines is the fine-grained integrated single pane of glass security control.

Scaling Up / Scaling Down — Another consideration is that streaming workflows tend to be multi-modal and unbalanced throughout the day, so the scaling capabilities are absolutely crucial. Flink and Spark Structured Streaming are developing auto-scaling approaches to automatically maintain steady and predictable performance. They each have a solid orchestration platform underneath, which tends to give them an edge over the “build your own” type of approach that you get with Kafka Streams.

Spark, Kafka, or Flink? Which to Use?

Boiling this down to high level guidance and decision making points, below is the heuristic that Cloudera tends to work with when advising customers.

	<p>Spark Structured Streaming is best for developer accessibility and whole platform solutions where low latency and advanced streaming are not required, e.g. combining batch and stream where response time is measured in seconds to minutes.</p>	<ul style="list-style-type: none"> • Spark Structured Streaming is already standard at your organization. • You need a unified batch/stream solution. • The highest levels of throughput is crucial. • Low latency is not necessary. • Advanced time/state features would be overkill. 	
	<p>Kafka Streams is best for Kafka Streams-only architectures without advanced streaming features.</p>	<ul style="list-style-type: none"> • You only need microservices. • Throughput is essential. • Low latency is crucial. • Time/state features are not needed out-of-the-box. • Application operational and resilience requirements are simple or handled elsewhere. 	
	<p>Flink is best for covering the full range of streaming pipeline requirements.</p>	<ul style="list-style-type: none"> • You need flexibility across microservices, batch and streaming. • High throughput is necessary. • Low latency is crucial. • Use cases call for advanced windowing and state capabilities. • You are not scared of new solutions, especially those that are best-in-class. 	<p>Both Flink and Kafka can be used as libraries in microservice architectures.</p>

Operational Efficiency

How would you understand what contributed to an unexpected value in a complex calculation while data continues to stream in?

Use the state processing API in Flink to recreate states as transactional snapshots and then dig in as much as needed to explain the origin and reasoning behind that dynamic calculation.

Checkpoints and savepoints are key to understanding how this works.

A Checkpoint provides a recovery mechanism in case of unexpected job failures. It creates a consistent image of the streaming job's execution state called a savepoint. You can use savepoints to stop-and-resume, fork, or update your jobs.

A fundamental aspect of Flink since its inception has been the separation of the state into local pieces that are linked together through consistent checkpointing levels. This minimizes latency but also provides all sorts of operational efficiency gain. For example, you can:

- Deploy new application versions that are preloaded from the current production state.
- Do accurate A/B testing of new algorithms because you can easily bring up new instances against a solid starting point.

To deal with supersets of data, the states saved locally in a high performance key value store (RocksDB) are checkpointed down to HDFS, a cloud blob store, or other durable repository.

Flink Use Cases

To get a practical understanding of how Flink is used in the real world, we have described a variety of use cases below.



Cybersecurity and Log Processing

A classic streaming data challenge is to identify and act upon intrusions and fraudulent events that are hidden within terabytes of dynamic machine logs. Throughput and latency are obviously important factors to consider because you want to identify an issue as quickly as possible. But effective action against criminals that doesn't alienate good customers requires understanding the behaviors of both good and bad actors.

The robust state management features of Flink serve as a cornerstone for cutting-edge cybersecurity solutions. By harnessing the performance advantages of localized state, Flink empowers efficient enrichment functions and intricate event processing. Advanced session windows and watermarking techniques enable in-depth analysis of dynamic data streams, facilitating thorough investigations for a secure digital landscape.



Outage Classification for Telecom Companies

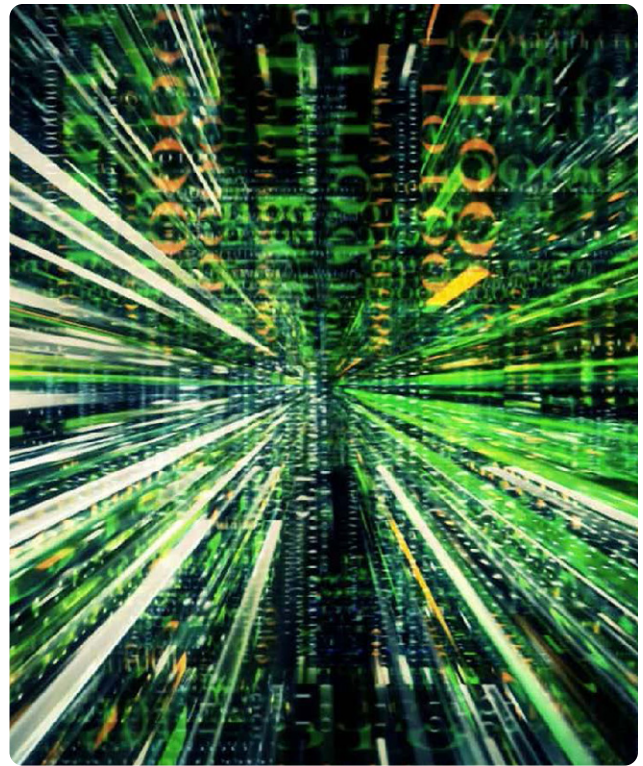
For decades, telecom companies have focused on their network infrastructure and preventative maintenance but often as a reaction to past events. Customer and regulatory demands require a dynamic approach to predict and mitigate spotty performance and outages.

Adapting network strength and mass availability is crucial and requires aggregated analysis on vast amounts of data over a wide array of networks to find anomalies, predict where failures are likely to occur, or even just record the state of their current network at any point of time.

5G has only increased the volume and variety of metrics available, so having the ability to scale and perform analytics on incoming events quickly is absolutely critical to identifying problems before customers do.

Financial Services: Mainframe Offloading

Today's consumers expect swift and seamless service experiences. Traditional, overburdened mainframes struggling with low-latency user interactions often hinder banks' efficiency, especially in the era of Open Banking directives, which require them to share customer data with third-party providers. To address this challenge, financial institutions are transitioning



customer relationship functions from mainframes to agile stateful stream processing engines like Flink. This shift enables tailored product offerings based on real-time spending patterns, delivering an enhanced and personalized customer experience.

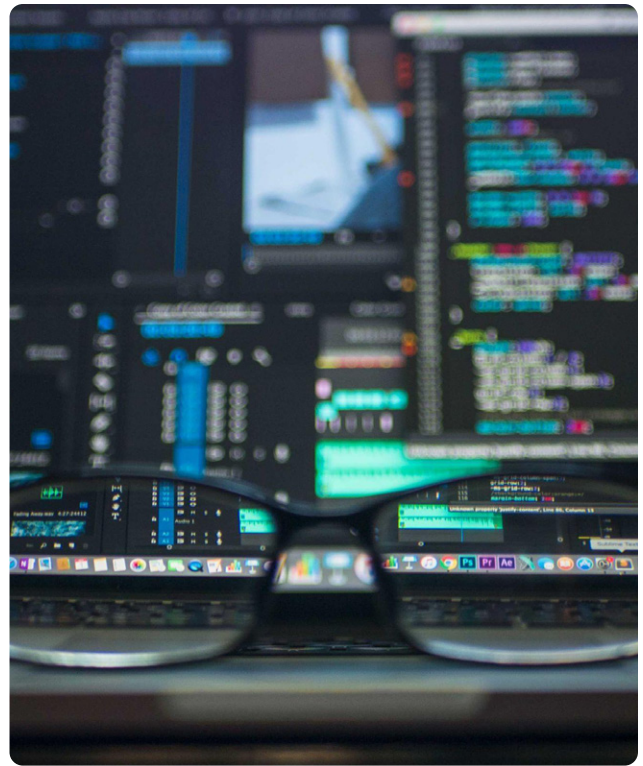
A key driver of success is data consistency. Flink's exactly-once delivery guarantee ensures that data is processed exactly once, even in the event of failures. This is critical for applications that need to track customer spending behavior and balances, as well as those that need to make real-time decisions based on the latest data. Flink's ability to combine exactly-once delivery with complex event processing (CEP) makes it a powerful tool for real-time marketing. By analyzing real-time data streams, Flink can identify patterns that indicate when a customer is likely to be interested in a particular product or offer. This allows banks to deliver the right offer at the right time, which can lead to increased customer engagement and sales. Flink's ability to perform data access, enrichment, and decisioning locally eliminates the need to access external data sources. This can significantly improve response times, as data does not need to be transferred over the network. This is essential for applications that require millisecond-level response times, such as fraud detection and risk assessment.



IoT for Manufacturing

IoT devices streamline supply chain operations within a manufacturing facility. Today, manufacturers are leveraging advanced monitoring sensors and real-time technologies to track quality of goods, automate the visual inspection of goods, and customized manufacturing for individual partners.

Flink's advanced windowing and state capabilities make it an ideal platform for processing sensor data in manufacturing. By aggregating and comparing sensor data over time, Flink can identify patterns that indicate potential problems with machines. This allows manufacturers to take corrective action before a problem causes a costly outage. Additionally, Flink's flexibility makes it a good fit for the different use cases in manufacturing IoT. Flink can be used to process data from both streaming and batch sources, and it can be deployed as a microservice or a standalone application. This makes it easy to integrate Flink with other manufacturing systems. Overall, Flink is a powerful platform for processing sensor data in manufacturing. Its advanced windowing and state capabilities, as well as its flexibility, make it a good fit for the diverse needs of manufacturing.



Fraud Monitoring for Banking

Today's financial services demand instant approval which requires data from various sources while transaction streams carry ever increasing volumes of data. Fraudsters try to exploit these conditions and continue to evolve tactics to try to stay one step ahead.

Flink's ultra-low latency unified processing makes it possible to process transaction streams in real time while executing table look-ups for historical customer data. In order to identify fraudulent patterns, transactions must be processed in context of customer profile and transaction history. Furthermore, with Flink SQL API's providing a useful abstraction layer, fraud monitoring logic can be expressed in the language of data, making the capabilities of Flink more accessible to less technical fraud analysts so they can adapt their monitoring techniques to stay ahead of the fraudsters.

Technical Features Table

The table below gives technical comparison across four modern stream processing engines. Refer to it when evaluating the functional and developmental aspects of your project.

	Flink 1.16	Kafka Streams 2.4	Spark Structured Streaming 2.4
Approach, position	<ul style="list-style-type: none"> ● Streaming first ● Modern class-leader 	<ul style="list-style-type: none"> ◆ Message-broker first ● Popular streaming add-on 	<ul style="list-style-type: none"> ◆ Batch first ● Popular streaming add-on
Streaming model, throughput, type	<ul style="list-style-type: none"> ● Stateful (First class requirement) ● <500 milliseconds ● Event-at-a-time 	<ul style="list-style-type: none"> ● Stateful ● <500 milliseconds ● Event-at-a-time 	<ul style="list-style-type: none"> ◆ Stateful ● Greater than 1 second ● Microbatch
Time support	<ul style="list-style-type: none"> ● Event time ● Processing time ● Customizable for greater control 	<ul style="list-style-type: none"> ● Event time ● Processing time 	<ul style="list-style-type: none"> ● Event time ● Processing time
Processing abstractions	<ul style="list-style-type: none"> ◆ Table ● SQL (ANSI Standard) ● Complex event processing ● Graph ● Machine learning ● Batch (experimental) 	<ul style="list-style-type: none"> ◆ Table ● SQL-like DSL (KSQL) (not ANSI compliant) ● No batch 	<ul style="list-style-type: none"> ● Table ● ANSI SQL:2003 in Spark Structured Streaming 2.3 ● Graph ● Machine learning ● Unified APIs for batch and stream
Delivery guarantee	<ul style="list-style-type: none"> ● Upstream: Exactly-once ● Downstream: Some capabilities depending on the downstream system 	<ul style="list-style-type: none"> ◆ Upstream: Exactly-once Kafka Streams only ● Downstream: No 	<ul style="list-style-type: none"> ● Upstream: Exactly-once ● Downstream: Some capabilities depending on the downstream system
State management	<ul style="list-style-type: none"> ● RocksDB ● Configurable snapshots ● Queryable 	<ul style="list-style-type: none"> ● BYO RocksDB ● Snapshot to Kafka Streams topic ● Queryable 	<ul style="list-style-type: none"> ◆ RocksDB Databricks only ● OSS sync on HDFS
Fault tolerance and resilience	<ul style="list-style-type: none"> ● Built-in ● Checkpoints ● Savepoints ● Redistributable 	<ul style="list-style-type: none"> ◆ BYO Microservice 	<ul style="list-style-type: none"> ● Built-in

● Great fit for purpose ◆ Fits with some work ► Fits with a lot of work

Operational Features Table

The table below gives an operational comparison across four modern stream processing engines. Refer to it when evaluating the nonfunctional aspects of your project.

	Flink 1.16	Kafka Streams 2.4	Spark Structured Streaming 2.4
Deployment model	<ul style="list-style-type: none"> • Clustered • Kubernetes • YARN • Kafka • Docker • S3 • Microservices 	<ul style="list-style-type: none"> ♦ Not clustered • Kubernetes • Microservices 	<ul style="list-style-type: none"> ♦ Clustered • Kubernetes
Documentation	<ul style="list-style-type: none"> ♦ Good technical documentation • Growing examples • Stack overflow coverage 	<ul style="list-style-type: none"> • Extensive documentation • Extensive examples • Stack overflow coverage 	<ul style="list-style-type: none"> • Extensive documentation • Extensive examples • Stack overflow coverage
Maturity/ community	<ul style="list-style-type: none"> • Smaller but fastest growing community with strong research and production deployments 	<ul style="list-style-type: none"> • Newest, strong community with strong growth 	<ul style="list-style-type: none"> • Spark Structured Streaming community is strong, but Streaming is a small, quiet corner
Use cases	<ul style="list-style-type: none"> • Unbounded and bounded streams • Batch • Complex event processing • IoT • Microservices • Others 	<ul style="list-style-type: none"> • Microservice/event driven, embedded in another application 	<ul style="list-style-type: none"> • Unified ETL, semi-RT processing
Enterprise management	<ul style="list-style-type: none"> • Rich OSS • Enhanced vendor offerings 	<ul style="list-style-type: none"> ♦ Minimal OSS • Some via vendor offerings 	<ul style="list-style-type: none"> • Rich OSS • Enhanced vendor offerings
Push button security	<ul style="list-style-type: none"> ♦ Complex • Some OSS support • Limited vendor offerings 	<ul style="list-style-type: none"> • Simple, some OSS support, good vendor offerings 	<ul style="list-style-type: none"> • Complex • Good OSS support • Good vendor offerings
Logging/metrics	<ul style="list-style-type: none"> ♦ Usual OSS integrations, some vendor offerings 	<ul style="list-style-type: none"> ♦ BYO microservices 	<ul style="list-style-type: none"> • Good logging integration
Scaling up/down	<ul style="list-style-type: none"> • Not yet autoscaling, but all requirements available 	<ul style="list-style-type: none"> ♦ BYO microservice, scaling limits e.g.: shuffle sort 	<ul style="list-style-type: none"> • Not yet autoscaling, but all requirements available

● Great fit for purpose ♦ Fits with some work ► Fits with a lot of work

Customer Success

To provide insights into the business impact that can be drawn through a comprehensive data-in-motion solution, we provide these customer success examples.

1. An international communications company serving consumers and businesses in ten countries, deployed the Cloudera streaming data platform to tackle a variety of critical use cases including, stream processing, log aggregation, large-scale messaging and customer insights.

Results included improved overall customer experience through strategic use of data analysis, reduced infrastructure management costs and TCO, and enabled real-time actions to improve business outcomes.

2. A large European bank specializing in agriculture financing and sustainability oriented banking across global markets leveraged Cloudera's streaming data platform to run sophisticated real-time algorithms and financial models to help customers manage their financial obligations, including loan repayments.

By implementing the platform and gaining the ability to stream real-time data, the bank can now detect warning signals in extremely early stages of where clients may go into default. Through their new, governed data lake, the bank's account managers are also able to access an in-depth overview of customer data, enabling them to generate liquidity overviews and advise customers on how to avoid defaulting. Through rapid data processing, better models are created that more accurately predict warning signals.

3. Afterpay is a frictionless payment solution for customers. By providing instant loans, AfterPay is able to give customers a convenient "buy now, pay later" option. The speed at which data is captured, streamed and processed is what allows AfterPay to offer this service to customers. Behind the scenes AfterPay is processing up to 800 million transactions daily that are all processed by the fraud monitoring team before approvals are made. The speed of Flink data processing helps AfterPay stop bad actors who would seek to initiate two loan requests in different geos simultaneously. This is completely unbeknownst to the customer who gets an instant no hassle approval and is now able to use their loan at the point of sale. The data from this transaction is then also persisted in a large scale, redundant cloud database for future analysis and to create behavioral models or simply for letting the customer know what they purchased in the past.

Ensure Fit for Purpose and Enterprise Wide Adoption

Streaming and time based reasoning applications are confronted with both simple and complex sets of challenges. Functional business requirements dictate the manner in which data must be processed, thereby guiding the evaluation and selection of the most suitable stream processing engine to meet your specific needs.

This paper described a number of capabilities that would address both simple and complex scenarios, while keeping in mind acceptable trade-offs. You don't want to over-engineer a solution, but you want to know that it can grow to support an evolving business. To support that growth, there are a number of technical and operational factors that are crucial to the decision making process.

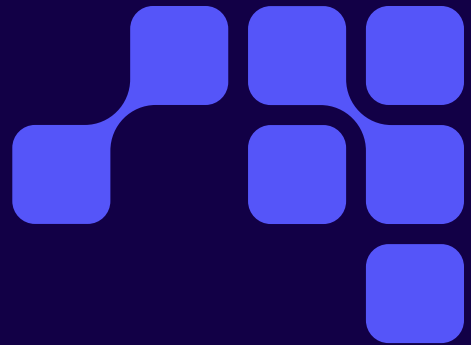
We also suggested that you take a broad perspective that also considers nonfunctional aspects such as how your team can deliver on the solution's promise, how it integrates into your organization's security framework, operational processes, support structure, and how it can scale up and down in line with business demand.

In summary, the view expressed here will help ensure that you choose the right stream processing engine that is both fit for purpose to the business challenge at hand, and will also enjoy enterprise wide adoption.

About Cloudera

Cloudera is the only true hybrid platform for data, analytics, and AI. With 100x more data under management than other cloud-only vendors, Cloudera empowers global enterprises to transform data of all types, on any public or private cloud, into valuable, trusted insights. Our open data lakehouse delivers scalable and secure data management with portable cloud-native analytics, enabling customers to bring GenAI models to their data while maintaining privacy and ensuring responsible, reliable AI deployments. The world's largest brands in financial services, insurance, media, manufacturing, and government rely on Cloudera to be able to use their data to solve the impossible — today and in the future.

To learn more, visit [Cloudera.com](https://cloudera.com) and follow us on [LinkedIn](#) and [X](#). Cloudera and associated marks are trademarks or registered trademarks of Cloudera, Inc. All other company and product names may be trademarks of their respective owners.



CLUDERA

Cloudera, Inc. | 5470 Great America Pkwy, Santa Clara, CA 95054 USA | cloudera.com