How Does Apache Flink Work?

Apache Flink is a framework and distributed processing engine for stateful computations over unbounded and bounded data streams. Flink has been designed to run in all common cluster environments, and perform computations at in-memory speed, at any scale. Developers build applications for Flink using APIs such as Java or SQL, which are executed on a Flink cluster by the framework.

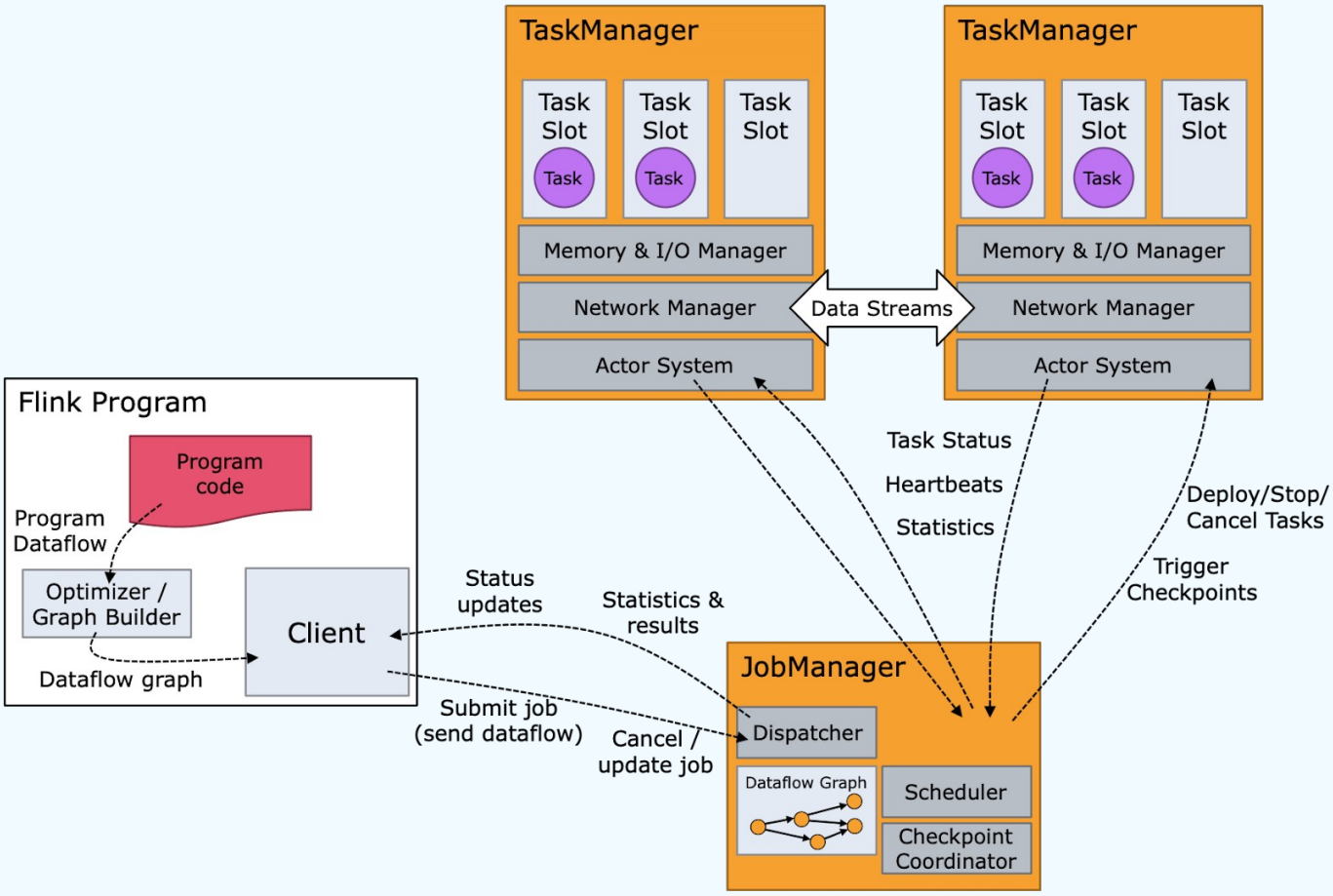
While other popular data processing systems like Apache Spark and Kafka Streams are limited to data streaming or batch processing, Flink supports both. This makes it a versatile tool for businesses in industries such as finance, e-commerce, and telecommunications who can now perform both batch and stream processing in one unified platform. Businesses can use Apache Flink for modern applications such as fraud detection, personalized recommendations, stock market analysis, and machine learning.

Apache Flink Architecture and Key Components

While Kafka Streams is a library that runs as part of your application, Flink is a standalone stream processing engine that is deployed independently. Apache Flink runs your application in a Flink cluster that you somehow deploy. It provides solutions to the hard problems faced by distributed stream processing systems, such as fault tolerance, exactly-once delivery, high throughput, and low latency. These solutions involve checkpoints, savepoints, state management, and time semantics.

Flink's architecture is designed for both stream and batch processing. It can handle unbounded data streams and bounded data sets, allowing for real-time data processing and data analysis. Flink also ensures data integrity and consistent snapshots, even in complex event processing scenarios.

The diagram below shows the Flink components as well as the Flink runtime flow. The program code or SQL query is composed into an operator graph which is then submitted by the client to a job manager. The job manager breaks the job into operators which execute as tasks on nodes that are running task managers. These tasks process streaming data and interact with various data sources, such as the Hadoop Distributed File System ﬁHDFSﬀ and Apache Kafka.



Advantages of Using Apache Flink

Apache Flink is a popular choice due to its robust architecture and extensive feature set.

Its features include sophisticated state management, savepoints, checkpoints, event time processing semantics, and exactly-once consistency guarantees for stateful processing.

Flink's stateful stream processing allows users to define distributed computations over continuous data streams. This enables complex event processing analytics on event streams such as windowed joins, aggregations, and pattern matching.

Flink can handle both bounded and unbounded streams, unifying batch and stream processing under the same umbrella. This makes it useful for various data processing needs, like real-time streaming applications that require processing both types of data.

Flink is also designed to run stateful applications at virtually any scale. It can process data parallelized into thousands of tasks distributed across multiple machines, and efficiently handle large data sets. This makes it ideal for applications that need to scale up to thousands of nodes with minimal latency and throughput loss.

Flink also features layered APIs for handling streams at different levels of abstraction. This gives developers the flexibility required to handle common, as well as hyper-specialized stream processing use cases.

Flink has connectors for messaging and streaming systems, data stores, search engines, and file systems like Apache Kafka, OpenSearch, Elasticsearch, DynamoDB, HBase, and any database providing a JDBC client.

Flink also provides various programming interfaces, including the high-level Streaming SQL and Table API, the lower-level DataStream API, and the ProcessFunction API for fine control. This flexibility allows developers to use the right tool for each problem and supports both unbounded streams and bounded data sets.

It also supports multiple programming languages, including Java, Scala, Python, and other JVM languages like Kotlin, making it a popular choice among developers.

Use Cases of Apache Flink

Although built as a generic data processor, Flink's native support of unbounded streams contributed to its popularity as a stream processor. Flink's common use cases are very similar to Kafka's use cases, although Flink and Kafka serve slightly different purposes. Kafka usually provides the event streaming while Flink is used to process data from that stream. Flink and Kafka are commonly used together for:

- Batch processingss Flink is really good at processing bounded data sets, which makes it suitable for traditional batch processing tasks where data is finite and processed in chunks.
- Stream processingss Flink is designed to handle unbounded data streams. This allows continuous data processing in real time, making it ideal for applications requiring real-time analytics and monitoring.

- Event-driven applications Flink's capabilities in processing event streams make it a valuable tool for building event-driven applications like fraud and anomaly detection systems, credit card transaction systems, business process monitoring, etc.
- Update stateful applications (savepoints) Flink's stateful processing and savepoints enable the updating and maintaining of stateful applications. This ensures consistency and continuity, even during failures.
- Streaming applications Flink supports a wide range of streaming applications, from simple real-time data processing to complex event processing and pattern detection.
- Data analytics (batch, streaming) Flink's ability to handle both batch and streaming data makes it suitable for data analytics. This includes real-time data analysis and historical data processing.
- Data pipelines/ETL Flink is used in building data pipelines for ETL processes, where data is extracted from various sources, transformed, and loaded into data warehouses or other storage systems for further analysis.

Challenges with Apache Flink

Apache Flink has a complex architecture. It can be difficult to learn, and challenging for even seasoned practitioners to understand, operate and debug. Flink developers and operators often find themselves struggling with complexities around custom watermarks, serialization, type evolution, and so on.

Perhaps a bit more than most distributed systems, Flink poses difficulties with deployment and cluster operations, such as performance tuning to account for hardware selection and job characteristics. Common concerns involve reasoning about the root causes of performance issues such as backpressure, slow jobs, and savepoint restoration from unreasonably large states. Other common issues include fixing checkpoint failures, and debugging job failures such as out-of-memory errors.

Organizations using Flink tend to require teams of experts dedicated to developing stream processing jobs and keeping the stream processing framework operational. For this reason, Flink has only been economically feasible for large organizations with complex and advanced stream processing needs.

Flink vs. Kafka vs. Spark, and When to Use Them

Kafka Streams is a popular client library used for stream processing, particularly when the input and output data are stored in a Kafka cluster. Because it's part of Kafka, it leverages the benefits of Kafka natively.

ksqlDB layers the simplicity of SQL onto Kafka Streams, providing a nice starting point for stream processing, and broadening the audience for it.

Kafka users and Confluent customers often turn to Apache Flink for stream processing needs for a number of reasons. For example, complex stream processing often means

large intermediate state that does not always make sense to use Kafka for, because when it gets sufficiently large, this state becomes its own thing that impacts the resources and planning needed to operate the Kafka cluster. Also, there can be a requirement to process streams from multiple Kafka clusters in different locations, streams in Kafka with streams outside of Kafka, and so on.

Because Flink is its own distributed system with its own complexities and operational nuances, it has been unclear when the benefits of working with Apache Flink outweigh the complexities incurred by it, or the costs associated with it, especially when other stream processing technologies such as Kafka Streams or Apache Spark are available.

Benefits of Fully Managed Flink with Confluent

But the cloud changes that calculation, and allows us to fully embrace Apache Flink. By offering Apache Flink as a fully managed cloud service, we get to bring the benefits that Confluent Cloud brings to Kafka—to Flink as well. The operational complexities and nuances that make Apache Flink complex and costly, such as instance type or hardware profile selection, node configuration, state back end selection, managing snapshots, savepoints and so on, are handled for you, enabling developers to focus exclusively on their application logic, rather than Flink specific nuances.

Not only does this bring the capabilities of Flink to Confluent Cloud, but it also changes the economics behind when Flink makes sense to use for stream processing. This means Flink can be used for more use cases earlier in an organization's streaming maturity. It also means that the developer can pick and choose the stream processing layer that works for them, allowing the developer to stick with one platform as their needs change or their complexity increases.

Now that Flink has a mature and robust SQL interface, it makes sense to start there, not only for end users adopting stream processing, but also for Confluent adopting Apache Flink.

Confluent Chronicles: The Force of Kafka + Flink Awakens

Introducing the second issue of the Confluent Chronicles comic book! Trust us, this isn't your typical eBook. Inside you'll find a fun and relatable approach to learn about the challenges of streaming processing, the basics of Apache Flink, and why Flink and Kafka are better together.

View Now