Cloudera Streaming Analytics

Apache Flink Overview

Date published: 2019-12-17
Date modified: 2019-12-17

https://docs.cloudera.com/

Legal Notice

Contents

What is Apache Flink?

Flink is a distributed processing engine and a scalable data analytics framework. You can use Flink to process data streams at a large scale and to deliver real-time analytical insights about your processed data with your streaming application.

Flink is designed to run in all common cluster environments, perform computations at in-memory speed and at any scale. Furthermore, Flink provides communication, fault tolerance, and data distribution for distributed computations over data streams.

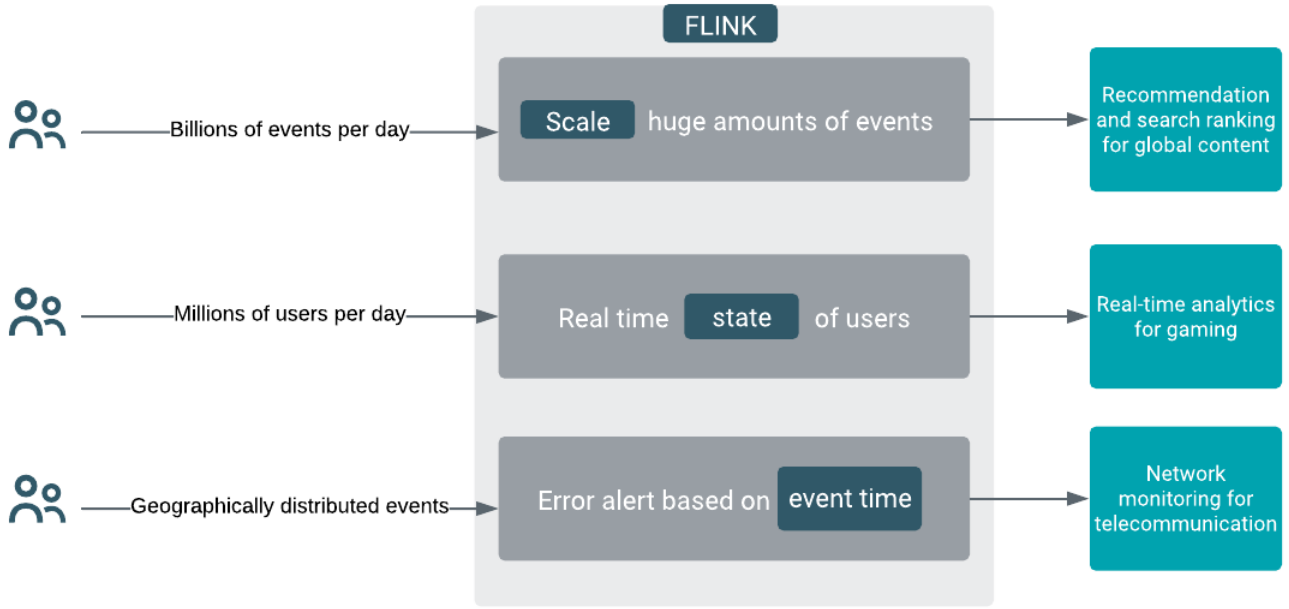
Flink applications process stream of events as unbounded or bounded data sets. Unbounded streams have no defined end and are continuously processed. Bounded streams have an exact start and end, and can be processed as a batch. In terms of time, Flink can process real-time data as it is generated and stored data in storage filesystems. In CSA, Flink is used for unbounded, real-time stream processing.

For more information about Flink, see the Apache Flink documentation.

Streaming use cases with Flink

You can create your Flink streaming applications based on the role of your processed data. Flink use cases include fraud detection, network monitoring, alert triggering, and other solutions to enhance user experience.

A large variety of enterprises choose Flink as a stream processing platform due to its ability to handle scale, stateful stream processing, and event time. See the following illustration for example use



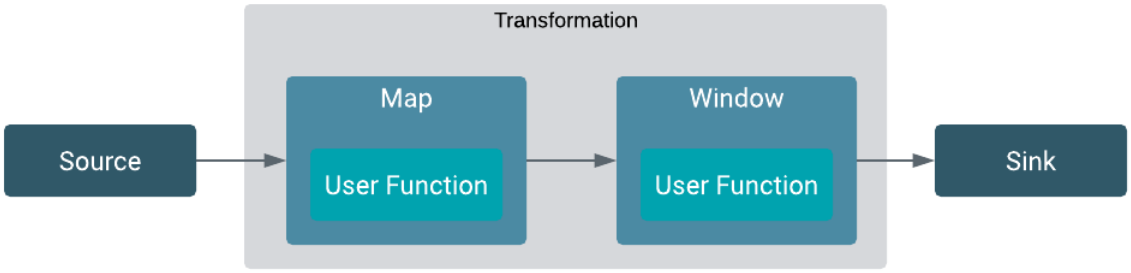
cases.

For specific examples of Apache Flink users, see the Apache Flink Powered by page.

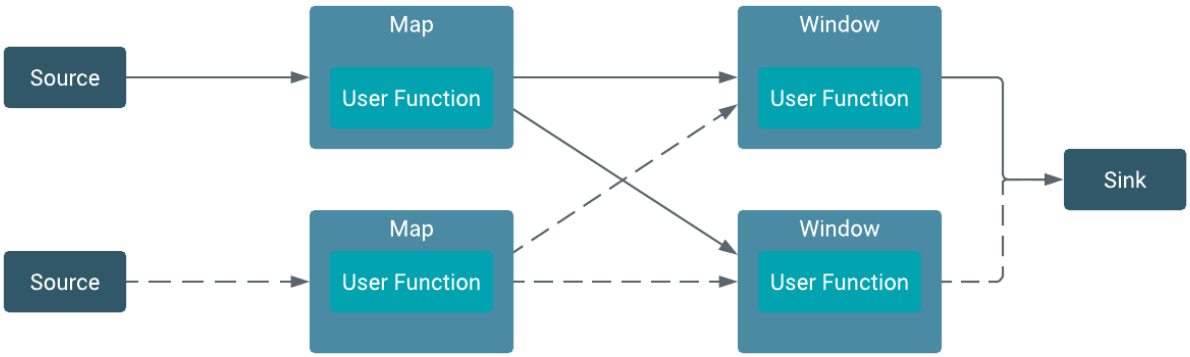
Flink Streaming Applications

You can use APIs to develop Flink streaming applications where the data pipeline consists of one or more data source, data transformation, and data sink. You can build the architecture of your application with parallelism and windowing functions to benefit from the scalability and state handling features of Flink.

The DataStream API is used as the core API to develop Flink streaming applications using Java or Scala programming languages. The core building blocks of a streaming application are datastream and transformation. In a Flink program, the incoming data streams from a source are transformed by a defined operation which results in one or more output streams to the sink as shown in the following illustration.



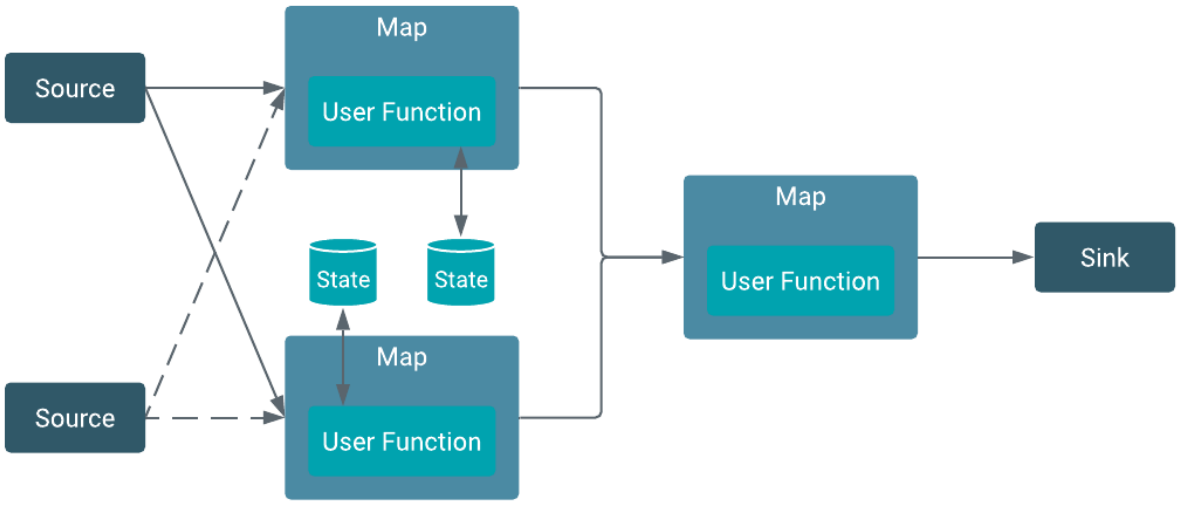
The structure of this dataflow is implemented in a pipeline that gives a Flink application its core logic. On a dataflow one or more operations can be defined which can be processed in parallel and independently to each other. With windowing functions, different computations can be applied to different streams in the defined time window to further maintain the processing of events. The following image illustrates the parallel structure of dataflows.



Handling state in Flink

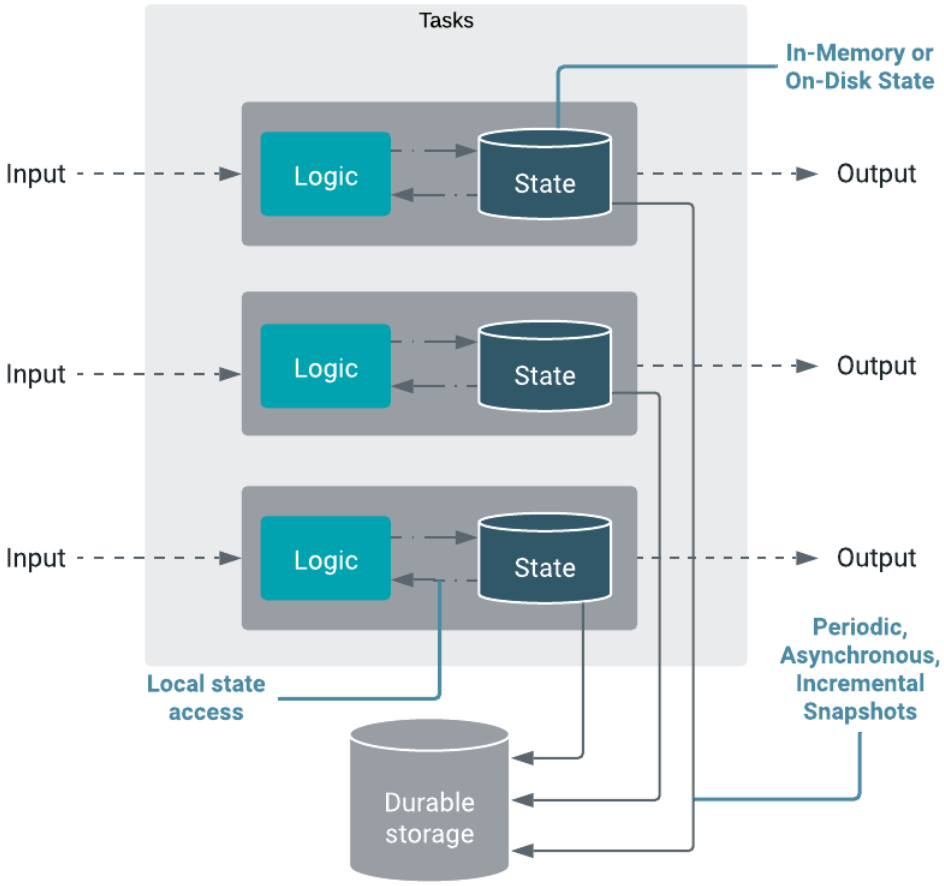
You can use Flink to store the state of your application locally in state backends that guarante lower latency when accessing your processed data. You can also create checkpoints and savepoints to have a fault-tolerant backup of your streaming application on a durable storage.

Stateful applications process dataflows with operations that store and access information across multiple events. There are two basic types of states in Flink: keyed state and operator state. The difference between them is that a keyed state is always bound to keys and can only be used on keyed streams. In operator state, the state is bound to an operator on one parallel substream. Keyed streams are created by defining keys for the elements of a stream. The keyed stream is read by the stateful operator and per key state is stored locally and can be accessed by the operator throughout the data streaming process. A basic stateful application structure is shown in the following illustration.



State backend

Flink applications store and access the working instance of their state locally, and preferably in memory. In Flink, the implementation of these local stores is called state backends. Flink also creates asynchronous and periodic snapshots of the stored state in the application, known as checkpoints. When you enable checkpointing, the snapshots of the states are created and saved into a given durable storage, like HDFS. The following illustration shows how the storing of state and checkpoint is implemented in a Flink application.



CSA supports Java heap and RocksDB state backends. The practical difference between them is that the Heap option is recommended for small states, while the RocksDB option is the production grade solution for large states. RocksDB also supports incremental checkpointing.

Related Information

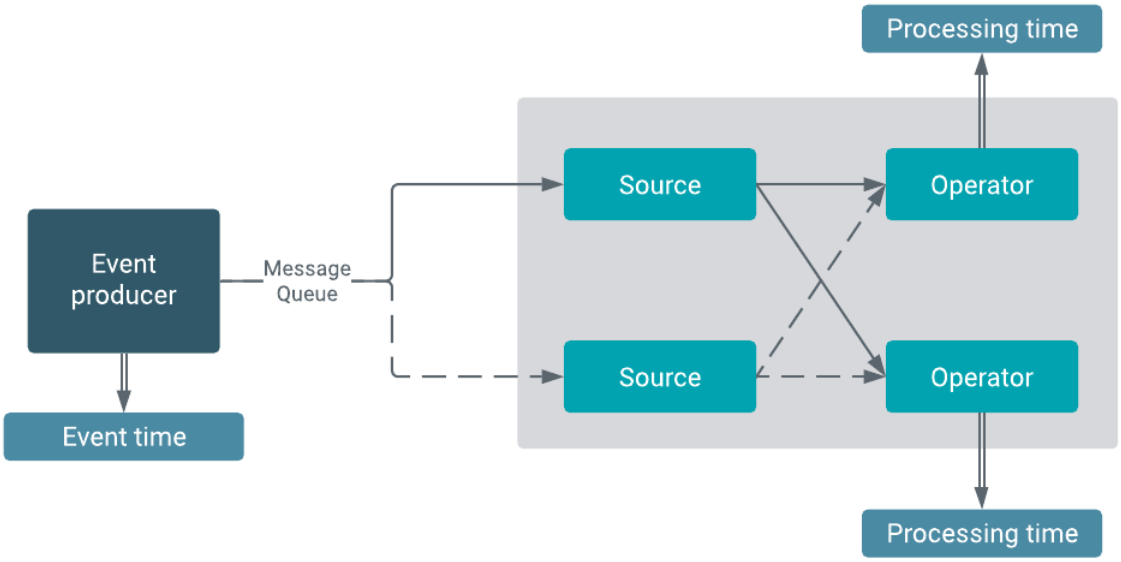
Creating checkpoints and savepoints in Flink

RocksDB state backend configuration

Event-driven applications with Flink

In time-sensitive cases where the application uses alerting or triggering functions, it is important to distinguish between event time and processing time. To make the designing of applications easier, you can create your Flink application either based on the time when the event is created or when it is processed by the operator.

Event-driven applications can be used in cases where the incoming events need to trigger other internal or external action. Consider an event-driven application ingesting events from multiple sources. If timeliness between input sources is valuable information the most straightforward approach to coordinate that is to write the original timestamp of recording the event to the event itself. For Flink, time is distinguished between event time and processing time as shown in the illustration below. Event time is defined by the time the event is created. Event time is already embedded in the record, before entering Flink. Processing time is the time at which the operator processes data. If an application is process time-based, the system clock on the machine is used for the operations.



Sophisticated windowing in Flink

The windowing feature of Flink helps you to determine different time sections of your unbounded data streams. This way you can avoid missing events that arrive late and you can easily apply different transformations on your streaming data.

Windowing can be used to group elements from an unbounded stream together by time, element count, or custom logic. Windows split the incoming stream into “buckets” of finite size, over which you can apply computations. The size of the split can be a period of time (event or processing time) or other custom logic. A window has a trigger and a function defined for it. The function contains the computation that needs to be completed in the given time on the contents of the window. Trigger specifies the condition in which the function is going to be applied. Also, with an evictor, elements can be removed from a window after the trigger, and before or after the function is applied. A window assigner has to be specified for the stream to define how elements are assigned to windows.

The followings are the types of window assigners:

- Tumbling windows
- Sliding windows
- Session windows
- Global windows

Related Information

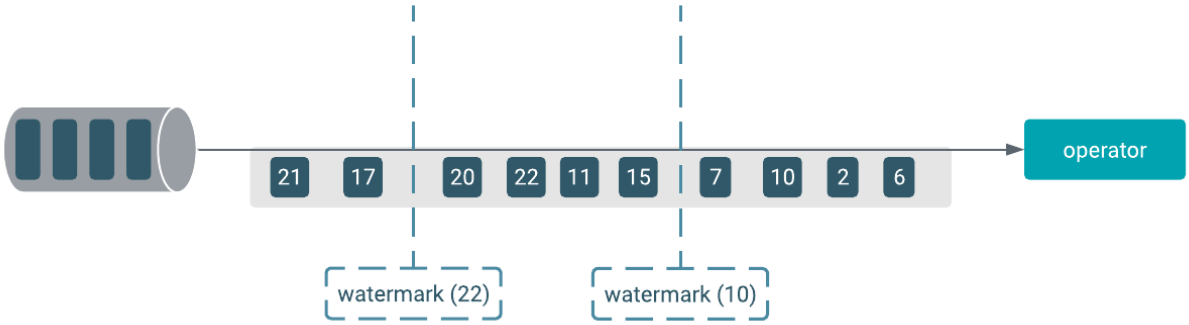
Stateful Tutorial: Creating windowed summaries

Using watermark in Flink

For a streaming application of unbounded data sets, the completeness of all incoming data is crucial. To guarantee that every data is processed, you can use watermarks in Flink applications to track the progress of time for events.

However, with event time, the timestamp only indicates when the event was created. With only the event time, it is not clear when the events are processed in the application. To track the time for an event time based application, watermark can be used. Watermark is a method to measure the progress of the event time. With event time, every input event has an embedded timestamp. This timestamp can be used for watermarks to indicate the time of incoming events to the operator. Like this, you can set the watermark to the time until the operator waits for the events that are being processed.

Let's think of a streaming application with a session window that aggregates data between 10:00 and 11:00. The given watermark will be the time until the data is processed. In this case, the watermark is 11:00. This means the window will progress the events that were created until 11:00.



Creating checkpoints and savepoints in Flink

You can use checkpoints and savepoints to make Flink applications fault tolerant throughout the whole pipeline. With checkpoints and savepoints, you can create a backup mechanism from which you can restore your whole application, with or without state, in case of failure or upgrade.

Flink contains a fault tolerance mechanism that creates snapshots of the data stream continuously. The snapshot includes not only the dataflow, but the state attached to it. In case of failure, the latest snapshot is chosen and the system recovers from that checkpoint. This guarantees that the result of the computation can always be consistently restored.

While checkpoints are created and managed by Flink, savepoints are controlled by the user. A savepoint can be described as a backup from the executed process.

Related Information

Checkpoint

Savepoint