SUMMARY

Flink

1. Installation git clone https://github.com/apache/flink

   Build with maven mvn clean install -DskipTests

   Another option is to download the release version from the download page.

2. Once built start the Flink engine as a CLI. /bin/start-local.sh I searched for this file and found it at ./flink-dist/target/flink-1.2-SNAPSHOT-bin/flink-1.2-SNAPSHOT/bin/start-local.sh. I ran from this location

   Validate that the local flink instance is running by checking the log file ./log/ or Job managers web instance http://localhost:8081/#/overview

   To stop the local instance ./bin/stop-local.sh

3. Run samples

I cd to the flink home cd ./flink-dist/target/flink-1.2-SNAPSHOT-bin/flink-1.2-SNAPSHOT

Ran an example ./bin/flink run ./examples/batch/WordCount.jar

1. What does the example WordCount do? https://ci.apache.org/projects/flink/flink-docs-release-0.8/programming_guide.html This page has the code for the sample flink program and the basic flink constructs. The program does the below 5 steps a. Get a execution environment b. Get the data from file or static sample c. run the transformations on the data ser d. write out put e. run the program by calling execute

   i. Set up your IDE http://dataartisans.github.io/flink-training/devSetup/handsOn.html

   ii. Create a Flink project and the WordCount program in the IDE.

   iii. Understand the programming constructs in the Sample WordCount https://ci.apache.org/projects/flink/flink-docs-release-0.7/programming_guide.html a. DataSet b. Tuple2 c.

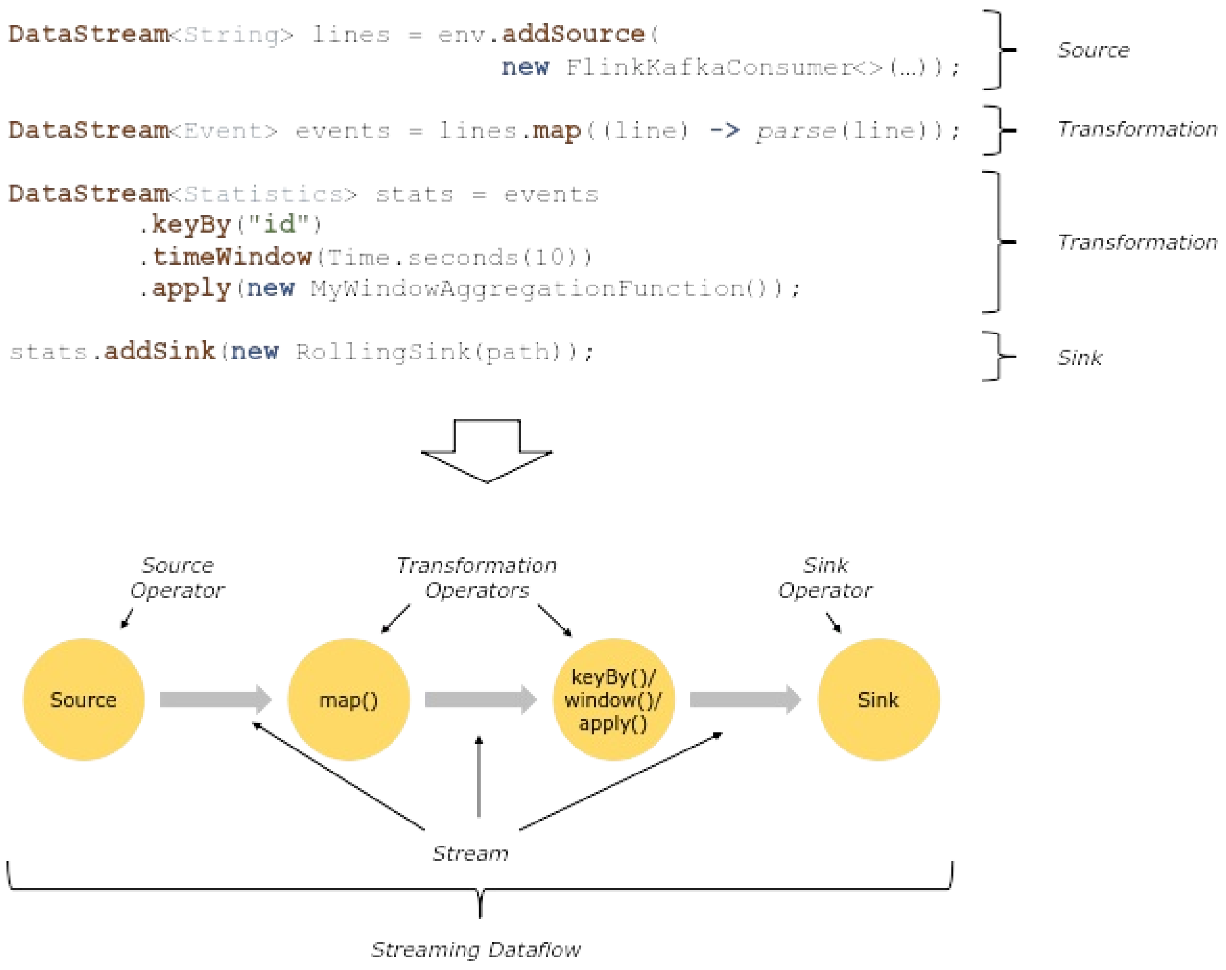
References: https://ci.apache.org/projects/flink/flink-docs-release-0.8/programming_guide.html - What does the example word count do?

Flink Concepts: https://ci.apache.org/projects/flink/flink-docs-master/concepts/index.html

The basic building blocks of Flink programs are streams and transformations (note that a DataSet is internally also a stream). A stream is an intermediate result, and a transformation is an operation that takes one or more streams as input, and computes one or more result streams from them.

When executed, Flink programs are mapped to streaming dataflows, consisting of streams and transformation operators. Each dataflow starts with one or more sources and ends in one or more sinks. The dataflows may resemble arbitrary directed acyclic graphs (DAGs). (Special forms of cycles are permitted via iteration constructs, we omit this here for simplicity).

In most cases, there is a one-to-one correspondence between the transformations in the programs and the operators in the dataflow. Sometimes, however, one transformation may consist of multiple transformation operators.

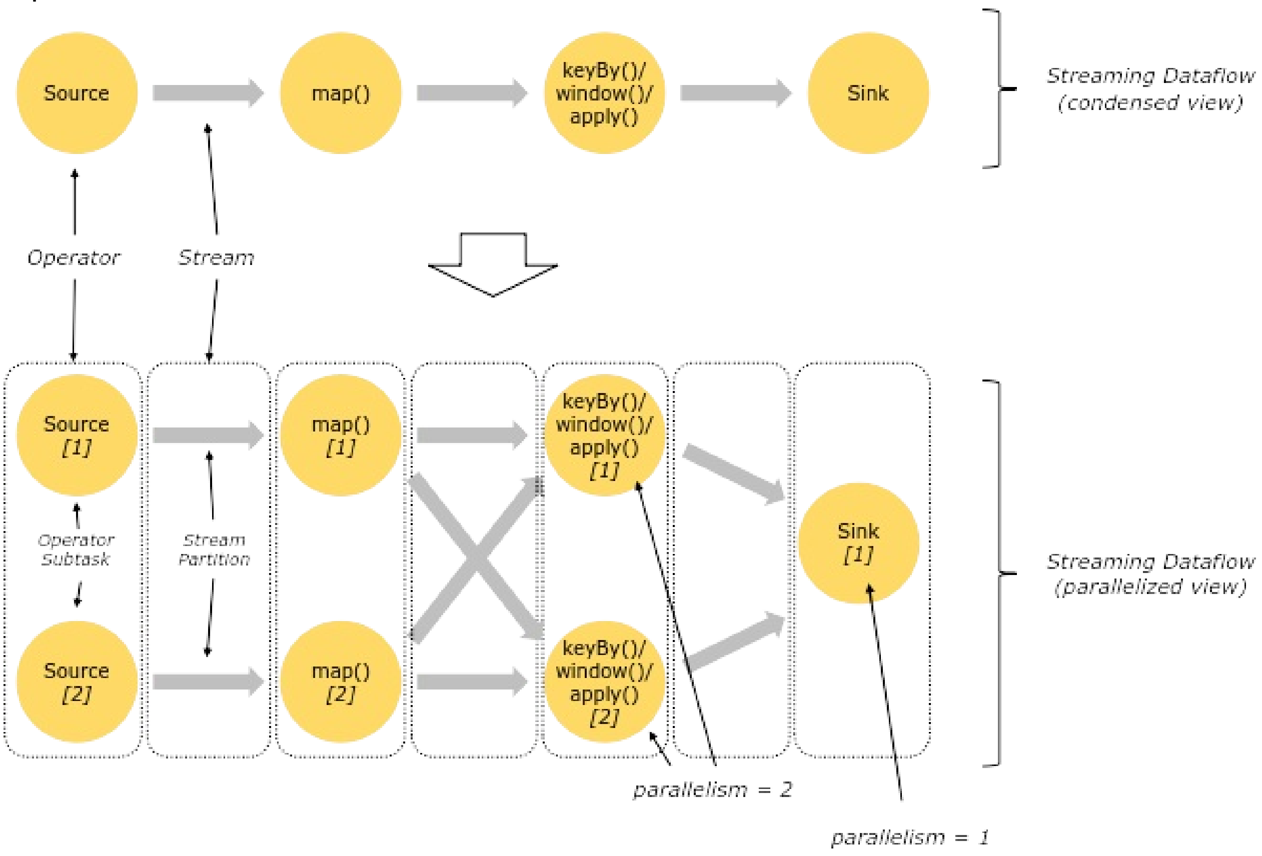


```
DataStream<String> lines = env.addSource(
                               new FlinkKafkaConsumer<>(…));

DataStream<Event> events = lines.map((line) -> parse(line));

DataStream<Statistics> stats = events
        .keyBy("id")
        .timeWindow(Time.seconds(10))
        .apply(new MyWindowAggregationFunction());

stats.addSink(new RollingSink(path));
```

Parallel Dataflows Programs in Flink are inherently parallel and distributed. Streams are split into stream partitions and operators are split into operator subtasks. The operator subtasks execute independently from each other, in different threads and on different machines or containers.

The number of operator subtasks is the parallelism of that particular operator. The parallelism of a stream is always that of its producing operator. Different operators of the program may have a different parallelism.

A parallel dataflow


Source
map()
keyBy()/ window()/ apply()
Sink
Streaming Dataflow (condensed view)
Operator
Stream
Source [1]
map() [1]
keyBy()/ window()/ apply() [1]
Operator Subtask
Stream Partition
Sink [1]
Streaming Dataflow (parallelized view)
Source [2]
map() [2]
keyBy()/ window()/ apply() [2]
parallelism = 2
parallelism = 1

Streams can transport data between two operators in a one-to-one (or forwarding) pattern, or in a redistributing pattern:

One-to-one streams (for example between the source and the map() operators) preserves partitioning and order of elements. That means that subtask[1] of the map() operator will see the same elements in the same order, as they were produced by subtask[1] of the source operator.

Redistributing streams (between map() and keyBy/window, as well as between keyBy/window and sink) change the partitioning of streams. Each operator subtask sends data to different target subtasks, depending on the selected transformation. Examples are keyBy() (re-partitions by hash code), broadcast(), or rebalance() (random redistribution). In a redistributing exchange, order among elements is only preserved for each pair of sending- and receiving task (for example subtask[1] of map() and subtask[2] of keyBy/window).

Streaming window: https://flink.apache.org/news/2015/12/04/Introducing-windows.html

Spark:

The primary value of Spark is its ability to control an expandable cluster of machines, and make them available to the user as though they were a single machine ecosystem. At the core of Spark functionality are two main components: the data storage/interaction format and the execution engine.

This is the answer to "What is XYZ?".

This is the answer to "How can I do X?".