**Stream processing with streaming dataflows**

Graphical user interface, text, application

Description automatically generated

Let’s say we have a code snippet of a Flink program as shown in the image above.

Step 1: The jar file of this program is distributed amongst all nodes inside the Flink cluster.

Diagram

Description automatically generated

Step 2: Flink converts all the user-defined transformations into a **streaming dataflow**, which is a data structure that encloses the order of operations applied to the input data stream. This dataflow can be represented as a directed graph that always starts with source(s) and ends at sink(s).

Diagram, schematic

Description automatically generated

Step 3: Flink converts the dataflow into an execution graph. This graph not only depicts how the abstract streaming dataflow is mapped to the system’s available resources, it also shows the level of parallelism of the job’s execution process.

**Fault-tolerance with Snapshots**

Flink periodically inserts “markers” into the stream, flowing along with the data. These markers create barriers that split the data stream into 2 parts, containing either old records or new ones. Using this technique, Flink can differentiate the state of the applications at different points in time.

Timeline

Description automatically generated

In Flink’s terminology, “markers” are “checkpoint barriers” and “parts” are “checkpoints”. Checkpoint n encapsulates the state of the operators resulted from having consumed all events happening **only before** checkpoint barrier n. It works like loading a computer game from a saved version. This mechanism provides the abilities to, literally, “time travel”. Flink users can go back in time and recover an earlier but consistent version, or experiment with different implementations at a desired point in time.

**References:**

https://nightlies.apache.org/flink/flink-docs-release-1.17/