Basic API Concepts (Source: https://ci.apache.org/projects/flink/flink-docs-release-1.0/apis/common/index.html)

Flink programs are regular programs that implement transformations on distributed collections (e.g., filtering, mapping, updating state, joining, grouping, defining windows, aggregating). Collections are initially created from sources (e.g., by reading files, kafka, or from local collections). Results are returned via sinks, which may for example write the data to (distributed) files, or to standard output (for example the command line terminal). Flink programs run in a variety of contexts, standalone, or embedded in other programs. The execution can happen in a local JVM, or on clusters of many machines.

Depending on the type of data sources, i.e. bounded or unbounded sources you would either write a batch program or a streaming program where the DataSet API is used for the former and the DataStream API is used for the latter.

Linking with Flink (Source: https://ci.apache.org/projects/flink/flink-docs-release-1.0/apis/common/index.html)

To write programs with Flink, you need to include the Flink library corresponding to your programming language in your project.

<properties>

<flink.version>1.0.3</flink.version>

</properties>

<dependency>

<groupId>org.apache.flink</groupId>

<artifactId>flink-core\_2.11</artifactId>

<version>${flink.version}</version>

</dependency>

<dependency>

<groupId>org.apache.flink</groupId>

<artifactId>flink-java\_2.11</artifactId>

<version>${flink.version}</version>

</dependency>

<dependency>

<groupId>org.apache.flink</groupId>

<artifactId>flink-clients\_2.11</artifactId>

<version>${flink.version}</version>

</dependency>

<dependency>

<groupId>org.apache.flink</groupId>

<artifactId>flink-streaming-java\_2.11</artifactId>

<version>${flink.version}</version>

</dependency>

<dependency>

<groupId>org.apache.flink</groupId>

<artifactId>flink-connector-kafka-0.9\_2.11</artifactId>

<version>${flink.version}</version>

</dependency>

DataSet and DataStream (Source: https://ci.apache.org/projects/flink/flink-docs-release-1.0/apis/common/index.html)

Flink has the special classes DataSet and DataStream to represent data in a program. You can think of them as immutable collections of data that can contain duplicates. In the case of DataSet the data is finite while for a DataStream the number of elements can be unbounded.

These collections differ from regular Java collections in some key ways. First, they are immutable, meaning that once they are created you cannot add or remove elements. You can also not simply inspect the elements inside.

A collection is initially created by adding a source in a Flink program and new collections are derived from these by transforming them using API methods such as map, filter and so on.

Lazy Evaluation (Source: https://ci.apache.org/projects/flink/flink-docs-release-1.0/apis/common/index.html)

All Flink programs are executed lazily: When the program’s main method is executed, the data loading and transformations do not happen directly. Rather, each operation is created and added to the program’s plan. The operations are actually executed when the execution is explicitly triggered by an execute() call on the execution environment. Whether the program is executed locally or on a cluster depends on the type of execution environment

The lazy evaluation lets you construct sophisticated programs that Flink executes as one holistically planned unit.

Flink DataStream API Programming Guide (Source: https://ci.apache.org/projects/flink/flink-docs-release-1.0/apis/streaming/index.html)

DataStream programs in Flink are regular programs that implement transformations on data streams (e.g., filtering, updating state, defining windows, aggregating). The data streams are initially created from various sources (e.g., message queues, socket streams, files). Results are returned via sinks, which may for example write the data to files, or to standard output (for example the command line terminal). Flink programs run in a variety of contexts, standalone, or embedded in other programs. The execution can happen in a local JVM, or on clusters of many machines.