DataStream API

Connectors

Apache Flink® Training

dataArtisans

Flink v1.2.0 – 27.02.2017

Streaming Connectors

- **Basic data sources**
  - Collections
  - Sockets
  - Filesystem
- **Twitter Stream (source)**
- **Queuing systems (sources and sinks)**
  - Apache Kafka
  - Amazon Kinesis
  - RabbitMQ
  - Apache NiFi
- **Data stores (sinks)**
  - Rolling files (HDFS, S3, …)
  - Elasticsearch
  - Cassandra
- **Custom connectors**

Add'l connectors in Apache Bahir

- **Netty (source)**
- **ActiveMQ (source and sink)**
- **Akka (sink)**
- **Flume (sink)**
- **Redis (sink)**

Basic Connectors

Basic Data Sources: Collections

```
StreamExecutionEnvironment env =
  StreamExecutionEnvironment.getExecutionEnvironment();


// read from elements
DataStream<String> names =
  env.fromElements("Some", "Example", "Strings");


// read from Java collection
List<String> list = new ArrayList<String>();
list.add("Some");
list.add("Example");
list.add("Strings");

DataStream<String> names = env.fromCollection(list);
```

Basic Data Sources: Sockets

```
StreamExecutionEnvironment env =
    StreamExecutionEnvironment.getExecutionEnvironment();


// read text socket from port
DataStream<String> socketLines = env
    .socketTextStream("localhost", 9999);
```

Basic Data Sources: Files

```
StreamExecutionEnvironment env =
    StreamExecutionEnvironment.getExecutionEnvironment();

DataStream<String> lines = env.readTextFile("file:///path");

DataStream<String> lines =
  env.readFile(inputFormat, "file:///path");
```

Data Sources: Monitored Files & Directories

```
StreamExecutionEnvironment env =
    StreamExecutionEnvironment.getExecutionEnvironment();

// monitor directory, checking for new files
// every 100 milliseconds
TextInputFormat format = new TextInputFormat(
    new org.apache.flink.core.fs.Path("file:///tmp/dir/"));

DataStream<String> inputStream = env.readFile(
    format,
    "file:///tmp/dir/",
    FileProcessingMode.PROCESS_CONTINUOUSLY,
    100,
    FilePathFilter.createDefaultFilter());
```

Note: if you modify a file (e.g. by appending to it), its entire contents will be reprocessed! This will break exactly-once semantics.

Basic Data Sinks

Print to the standard output

- `stream.print()`

Write as text file using toString()

- `stream.writeAsText("/path/to/file")`

Write as CSV file

- `stream.writeAsCsv("/path/to/file")`

Emit to socket

- `stream.writeToSocket(host, port, SerializationSchema)`

Execution

- Keep in mind that programs are lazily executed

```
DataStream<T> result;

// nothing happens
result.writeToSocket(...);

// nothing happens
result.writeAsText("/path/to/file", "\n", "|");

// Execution really starts here
env.execute();
```

Unbundled Connectors

Linking with the Unbundled Connectors

- Note that many of the available streaming connectors are not bundled with Flink by default
- This prevents dependency clashes with your code
- To use these modules, you can either
  - Copy the JAR files into the lib folder of each TaskManager
  - Or package them with your code (recommended)
- Docs

  https://ci.apache.org/projects/flink/flink-docs-release-1.2/dev/cluster_execution.html#linking-with-modules-not-contained-in-the-binary-distribution

Connecting to Apache Kafka

Kafka and Flink

- “Apache Kafka is a distributed, partitioned, replicated commit log service”
- Kafka maintains feeds of messages in categories called topics
- Flink can read a Kafka topic to produce a DataStream and write a DataStream to a Kafka topic
- Flink coordinates with Kafka to provide recovery in the case of failures

Reading Data from Kafka

- Add a DataStream source from a Kafka topic

```
Properties props = new Properties();
props.setProperty("zookeeper.connect", "localhost:2181");
props.setProperty("bootstrap.servers", "localhost:9092");
props.setProperty("group.id", "myGroup");

// create a data source
DataStream<String> data= env.addSource(
    new FlinkKafkaConsumer010<String>(
        "myTopic",                  // Kafka topic
        new SimpleStringSchema(), // deserialization schema
        props)                      // Consumer config
);
```

Writing Data to Kafka

- Add a Kafka sink to a DataStream by providing
  - the broker address
  - the topic name
  - a serialization schema

```
DataStream<String> aStream = …
aStream.addSink(
     new FlinkKafkaProducer010<String>(
         "localhost:9092",          // default local broker
         "myTopic",                 // Kafka topic
          new SimpleStringSchema()) // serialization schema
);
```

Writing to Elasticsearch

Elasticsearch

- Distributed search engine, based on Apache Lucene
- Part of an ecosystem that also includes Kibana for exploration and visualization
- Often used to store and index JSON documents
- Has good defaults, but you can not modify an index mapping (schema) after inserting data
- Elasticsearch has an HTTP-based REST API

Elasticsearch and Flink

- Flink has separate Sink connectors for Elasticsearch 1.x and 2.x (and 5.x in Flink 1.3)
- The Flink connectors use the Transport Client to send data
- You'll need to know your
  - cluster's network address
  - cluster name
  - index name

Implementing Custom Connectors

References

- Sources
  - http://ci.apache.org/projects/flink/flink-docs-master/api/java/org/apache/flink/streaming/api/functions/source/SourceFunction.html
  - Also RichSourceFunction, ParallelSourceFunction, and RichParallelSourceFunction
- Sinks
  - https://ci.apache.org/projects/flink/flink-docs-master/api/java/org/apache/flink/streaming/api/functions/sink/SinkFunction.html
- For a real example, look at NifiSource and NifiSink

Custom Connectors

```
StreamExecutionEnvironment env =
  StreamExecutionEnvironment.getExecutionEnvironment();

// read data stream from custom source function
DataStream<<Tuple2<Long, String> stream = env
  .addSource(new MySourceFunction());

// emit data with a custom sink function
stream.addSink(new MySinkFunction());
```

Sources

- Basic interface has four methods:
  - run – runs as long as necessary, emitting elements
  - cancel – called when run must stop
  - snapshotState – called during checkpointing
  - restoreState – called when rolling back

- Must not update state during checkpointing – a lock object is provided for this

- ParallelSourceFunction interface adds methods for coordination among multiple instances

Sinks

- Very simple interface
  - invoke method is called for every record

- For exactly once end-to-end semantics either
  - the underlying data store must support transactions, or
  - the updates must be idempotent

References

- Documentation
  - https://ci.apache.org/projects/flink/flink-docs-release-1.2/dev/connectors/index.html

- Blog posts
  - http://data-artisans.com/kafka-flink-a-practical-how-to/
  - https://www.elastic.co/blog/building-real-time-dashboard-applications-with-apache-flink-elasticsearch-and-kibana