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
);
```

When are Kafka offsets committed?

- If Flink checkpointing is disabled, then the Properties auto.commit.enable and auto.commit.interval.ms control this behavior

- If checkpointing is enabled, then the autocommit Properties are ignored, and Flink commits the offsets whenever a checkpoint is completed

Kafka timestamps

- Since Kafka 0.10, Kafka messages can carry timestamps
- Flink can use these timestamps; see https://ci.apache.org/projects/flink/flink-docs-release-1.2/dev/connectors/kafka.html#using-kafka-timestamps-and-flink-event-time-in-kafka-010 for details
- You will still need to arrange for watermarks to be emitted

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

Fault Tolerance

Fault Tolerance

- What happens if a worker thread goes down?
- Flink supports different guarantee levels for failure recovery:
- Exactly once
  - Each event affects the declared state of a program exactly once.
  - **Note**: This does *not* mean that events are processed exactly once!
- At least once
  - Each event affects the declared state of a program at least once
- Deactivated / None / At most once
  - All state is lost in case of a failure

Source & Sink Requirements

- “Exactly once” & “at least once” guarantees require replayable sources
  - Data must be replayed in case of a failure
- “End-to-End exactly once” guarantees require
  - Transactional sinks, or
  - Idempotent writes

Guarantees of Data Sources

Source	Guarantee
Apache Kafka	Exactly once
AWS Kinesis Streams	Exactly once
RabbitMQ	None (v 0.10) / Exactly once (v 1.0)
Twitter Streaming API	None
Collections	Exactly once
Files	Exactly once
Sockets	None

Guarantees of Data Sinks

Sink	Guarantee
HDFS rolling sink	Exactly once
Cassandra	Exactly once for idempotent updates
Elasticsearch	Exactly once for idempotent indexing
Kafka	At least once
AWS Kinesis Streams	At least once
File sinks	At least once
Socket sinks	At least once
Standard output	At least once
Redis	At least once

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