Example: Testing & Other Data Formats

In this lesson, we'll discuss the testing with Kafka and the data format Avro.

We'll cover the following

- Tests with embedded Kafka
- Avro as data format

Tests with embedded Kafka

In a JUnit test, an *embedded Kafka server* can be used to analyze the functionality of the microservices. In this case, a Kafka server runs in the same Java Virtual Machine (JVM) as the test. Thus, it is not necessary to build up an infrastructure for the test, and consequently, there is no need to tear down the infrastructure again after the test.

This requires two things essentially:

- An embedded Kafka server has to be started. With a class rule, JUnit can be triggered to start a Kafka server prior to the tests and to shut it down again after the tests. Therefore, a variable with `@ClassRule` must be added to the code.

```
@ClassRule
public static KafkaEmbedded embeddedKafka =
  new KafkaEmbedded(1, true, "order");
```

- The Spring Boot configuration must be adapted in such a manner that Spring Boot uses the Kafka server. This code is found in a method annotated with `@BeforeClass`, so that it executes before the tests.

```
@BeforeClass
public static void setUpBeforeClass() {
  System.setProperty("spring.kafka.bootstrap-servers",
   embeddedKafka.getBrokersAsString());
}
```

Avro as data format

Avro (http://avro.apache.org/) is a data format quite frequently used with Kafka (https://www.confluent.io/blog/avro-kafka-data/) and Big Data solutions from the Hadoop area. Avro is a binary protocol, but also offers a JSON-based representation. There are Avro libraries for Python, Java, C#, C++, and C.

Avro supports schemas, meaning that each dataset is saved or sent together with its schema. For optimization, a reference to a schema from the schema repository can be used rather than a copy of the complete schema. Thereby, it is clear which format the data has. The schema contains a documentation of the fields. This ensures the long-term interpretation of the data, and that the semantics of the data are clear.

In addition, the data can be converted to another format upon reading. This facilitates the schema evolution (https://martin.kleppmann.com/2012/12/05/schema-evolution-in-avro-protocol-buffers-thrift.html). New fields can be added when default values are defined so that the old data can be converted into the new schema by using the default value. When fields are deleted, a default value can be given so that new data can be converted into the old schema. In addition, the order of the fields can be changed because the field names are stored.

An advantage of the flexibility associated with schema migration is that old records can be processed with current software and the current schema. Also, software based on an old schema can process new data. Message-oriented middleware (MOM) typically does not have such requirements because messages are not stored for very long. Only upon long-term record storage does schema evolution turn into a challenge.

QUIZ

Q Lots of complex additional infrastructure is required to run tests with Kafka.

A) True

B) False

Submit Answer

Reset Quiz

In the next lesson, we'll discuss some variations of the messaging and Kafka recipe.

Back

Example: Topics & Partitions

Next

Variations & Experiments

Mark as Completed

Report an Issue