Exercise: Kafka Connectors

1. Source Connector Configuration

For the source connector, the reference configuration is available at *$CONFLUENT_HOME/etc/kafka/connect-file-source.properties*:

```
name=local-file-source
connector.class=FileStreamSource
tasks.max=1
topic=connect-test
file=test.txt
```

For this to work then, let's create a basic file with some content:

```
echo -e "foo\nbar\n" > $KAFKA_HOME/test.txt
```

Note that the working directory is $CONFLUENT_HOME.

2. Sink Connector Configuration

For our sink connector, we'll use the reference configuration at *$CONFLUENT_HOME/etc/kafka/connect-file-sink.properties*:

```
name=local-file-sink
connector.class=FileStreamSink
tasks.max=1
file=test.sink.txt
topics=connect-test
```

3. Worker Config

Finally, we have to configure the Connect worker, which will integrate our two connectors and do the work of reading from the source connector and writing to the sink connector.

For that, we can use *$CONFLUENT_HOME/etc/kafka/connect-standalone.properties*:

```
bootstrap.servers=localhost:9092
key.converter=org.apache.kafka.connect.json.JsonConverter
value.converter=org.apache.kafka.connect.json.JsonConverter
key.converter.schemas.enable=false
value.converter.schemas.enable=false
offset.storage.file.filename=/tmp/connect.offsets
```

```
offset.flush.interval.ms=10000
plugin.path=/share/java
```

Note that *plugin.path* can hold a list of paths, where connector implementations are available

As we'll use connectors bundled with Kafka, we can set *plugin.path* to *$CONFLUENT_HOME/share/java*. Working with Windows, it might be necessary to provide an absolute path here.

4. Kafka Connect in Standalone Mode

And with that, we can start our first connector setup:

```
$$CONFLUENT_HOME/bin/connect-standalone \
  $CONFLUENT_HOME/etc/kafka/connect-standalone.properties \
  $CONFLUENT_HOME/etc/kafka/connect-file-source.properties \
  $CONFLUENT_HOME/etc/kafka/connect-file-sink.properties
```

First off, we can inspect the content of the topic using the command line:

```
$$CONFLUENT_HOME/bin/kafka-console-consumer --bootstrap-server
localhost:9092 --topic connect-test --from-beginning
```

As we can see, the source connector took the data from the *test.txt* file, transformed it into JSON, and sent it to Kafka:

```
{"schema":{"type":"string","optional":false},"payload":"foo"}
{"schema":{"type":"string","optional":false},"payload":"bar"}
```

And, if we have a look at the folder *$CONFLUENT_HOME*, we can see that a file *test.sink.txt* was created here:

```
cat $CONFLUENT_HOME/test.sink.txt
foo
bar
```

As the sink connector extracts the value from the *payload* attribute and writes it to the destination file, the data in *test.sink.txt* has the content of the original *test.txt* file.

Now let's add more lines to *test.txt.*

When we do, we see that the source connector detects these changes automatically.

We only have to make sure to insert a newline at the end, otherwise, the source connector won't consider the last line.

At this point, let's stop the Connect process, as we'll start Connect in *distributed mode* in a few lines.

5. Kafka Connect in Distributed Mode

5.1. Starting Connect

A reference configuration for distributed mode can be found at *$CONFLUENT_HOME/etc/kafka/connect-distributed.properties*.

Parameters are mostly the same as for standalone mode. There are only a few differences:

We can start Connect in distributed mode as follows:

```
$CONFLUENT_HOME/bin/connect-distributed $CONFLUENT_HOME/etc/kafka/connect-distributed.properties
```

5.2. Adding Connectors Using the REST API

Now, compared to the standalone startup command, we didn't pass any connector configurations as arguments. Instead, we have to create the connectors using the REST API.

To set up our example from before, we have to send two POST requests to *http://localhost:8083/connectors* containing the following JSON structs.

First, we need to create the body for the source connector POST as a JSON file. Here, we'll call it *connect-file-source.json*:

```
{
    "name": "local-file-source",
    "config": {
        "connector.class": "FileStreamSource",
        "tasks.max": 1,
        "file": "test-distributed.txt",
        "topic": "connect-distributed"
    }
}
```

Note how this looks pretty similar to the reference configuration file we used the first time.

And then we POST it:

```
curl -d @"$CONFLUENT_HOME/connect-file-source.json" \
  -H "Content-Type: application/json" \
  -X POST http://localhost:8083/connectors
```

Then, we'll do the same for the sink connector, calling the file *connect-file-sink.json*:

```
{
    "name": "local-file-sink",
    "config": {
        "connector.class": "FileStreamSink",
        "tasks.max": 1,
        "file": "test-distributed.sink.txt",
        "topics": "connect-distributed"
    }
```

```
}
```

And perform the POST like before:

```
curl -d @$CONFLUENT_HOME/connect-file-sink.json \
  -H "Content-Type: application/json" \
  -X POST http://localhost:8083/connectors
```

If needed, we can verify, that this setup is working correctly:

```
$CONFLUENT_HOME/bin/kafka-console-consumer --bootstrap-server
localhost:9092 --topic connect-distributed --from-beginning
{"schema":{"type":"string","optional":false},"payload":"foo"}
{"schema":{"type":"string","optional":false},"payload":"bar"}
```

And, if we have a look at the folder *$CONFLUENT_HOME*, we can see that a file *test-distributed.sink.txt* was created here:

```
cat $CONFLUENT_HOME/test-distributed.sink.txt
foo
bar
```

After we tested the distributed setup, let's clean up, by removing the two connectors:

```
curl -X DELETE http://localhost:8083/connectors/local-file-source
curl -X DELETE http://localhost:8083/connectors/local-file-sink
```

6. Transforming Data

To test some transformation features, let's set up the following two transformations:

- First, let's wrap the entire message as a JSON struct
- After that, let's add a field to that struct

Before applying our transformations, we have to configure Connect to use schemaless JSON, by modifying the *connect-distributed.properties*:

```
key.converter.schemas.enable=false
value.converter.schemas.enable=false
```

After that, we have to restart Connect, again in distributed mode:

```
$CONFLUENT_HOME/bin/connect-distributed $CONFLUENT_HOME/etc/kafka/connect-
distributed.properties
```

Again, we need to create the body for the source connector POST as a JSON file. Here, we'll call it *connect-file-source-transform.json.*

Besides the already known parameters, we add a few lines for the two required transformations:

```
{
    "name": "local-file-source",
    "config": {
        "connector.class": "FileStreamSource",
        "tasks.max": 1,
        "file": "test-transformation.txt",
        "topic": "connect-transformation",
        "transforms": "MakeMap,InsertSource",
        "transforms.MakeMap.type":
"org.apache.kafka.connect.transforms.HoistField$Value",
        "transforms.MakeMap.field": "line",
        "transforms.InsertSource.type":
"org.apache.kafka.connect.transforms.InsertField$Value",
        "transforms.InsertSource.static.field": "data_source",
        "transforms.InsertSource.static.value": "test-file-source"
    }
}
```

After that, let's perform the POST:

```
curl -d @$CONFLUENT_HOME/connect-file-source-transform.json \
  -H "Content-Type: application/json" \
  -X POST http://localhost:8083/connectors
```

Let's write some lines to our *test-transformation.txt*:

```
Foo
Bar
```

If we now inspect the *connect-transformation* topic, we should get the following lines:

```
{"line":"Foo","data_source":"test-file-source"}
{"line":"Bar","data_source":"test-file-source"}
```