

Java-Success.com


Prepare to fast-track, choose & go places with 800+ Java & Big Data Q&As with lots of code & diagrams.

[Home](#) [Why? ▾](#) [300+ Java FAQs ▾](#) [300+ Big Data FAQs ▾](#) [Courses ▾](#)

[👤 Membership ▾](#) [Your Career ▾](#)

[Home](#) > [bigdata-success.com](#) > [Tutorials - Big Data](#) > [TUT - Kafka](#) > 03: Apache Kafka
Connector Tutorial

03: Apache Kafka Connector Tutorial

 Posted on [February 17, 2019](#)

This extends the [Getting started with Apache Kafka on Mac tutorial](#). Kafka works with different kinds of data sources like **file**, **JDBC**, **JMS**, etc.

What is the purpose of a Kafka connector?

Its purpose is to easily add systems to your streamed data pipelines without having to write producer & consumer code. If you want to copy data between Kafka and another system, you instantiate Kafka Connectors for the systems they want to pull data from or push data to.

300+ Java Interview FAQs

300+ Java FAQs



16+ Java Key
Areas Q&As



150+ Java
Architect FAQs



80+ Java Code
Quality Q&As



150+ Java Coding
Q&As



300+ Big Data Interview FAQs

300+ Big Data
FAQs



Tutorials - Big
Data



TUT -  Starting Big
Data

TUT - Starting Spark &
Scala

In this tutorial, let's look at a **File Connector**. The task is to read data from a file and import it into Kafka, and then export that data and save to another file.

Step 1: Start the zookeeper & kafka.

```
1 /usr/local/kafka_2.11-2.1.0]$ ./bin/zookeeper-server-start.sh config/zookeeper.properties
2 /usr/local/kafka_2.11-2.1.0]$ ./bin/kafka-server-start.sh config/server.properties
3
```

Step 2: The properties file used are connect-standalone.properties, connect-file-source.properties and connect-file-sink.properties.

```
1 /usr/local/kafka_2.11-2.1.0]$ ls config/connect*
2
3 config/connect-console-sink.properties      config/
4 config/connect-console-source.properties    config/
5 config/connect-distributed.properties       config/
6
```

connect-file-source.properties

```
1 /usr/local/kafka_2.11-2.1.0]$ cat config/connect-file-source.properties
```

Outputs:

```
1 name=local-file-source
2 connector.class=FileStreamSource
3 tasks.max=1
4 file=test.txt
```

connect-file-sink.properties

```
1 /usr/local/kafka_2.11-2.1.0]$ cat config/connect-file-sink.properties
```

Outputs:

TUT - Starting with Python

TUT - Kafka

TUT - Pig

TUT - Apache Storm

TUT - Spark Scala on Zeppelin

TUT - Cloudera

TUT - Cloudera on Docker

TUT - File Formats

TUT - Spark on Docker

TUT - Flume

TUT - Hadoop (HDFS)

TUT - HBase (NoSQL)

TUT - Hive (SQL)

TUT - Hadoop & Spark

TUT - MapReduce

TUT - Spark and Scala

TUT - Spark & Java

TUT - PySpark on Databricks

TUT - Zookeeper

800+ Java Interview Q&As

300+ Core Java Q&As



300+ Enterprise Java Q&As



150+ Java Frameworks Q&As



120+ Companion Tech Q&As



Tutorials - Enterprise Java



```
1 name=local-file-sink
2 connector.class=FileStreamSink
3 tasks.max=1
4 file=test.sink.txt
5 topics=connect-test
6
```

connect-standalone.properties

```
1 /usr/local/kafka_2.11-2.1.0]$ cat config/connect-s
```

Outputs:

```
1 bootstrap.servers=localhost:9092
2
3 key.converter=org.apache.kafka.connect.json.JsonC
4 value.converter=org.apache.kafka.connect.json.Json
5
6 key.converter.schemas.enable=true
7 value.converter.schemas.enable=true
8
9 offset.storage.file.filename=/tmp/connect.offsets
10 offset.flush.interval.ms=10000
11
```

Run the file connector

```
1 /usr/local/kafka_2.11-2.1.0]$ ./bin/connect-standa
2
```

Step 3: Write something to **source file** "test.txt" file.

```
1 /usr/local/kafka_2.11-2.1.0]$ echo -e "file connect
2
```

Step 4: Whatever written to test.txt is imported into kafka and then exported to the **sink file** "test.sink.txt".

```
1 /usr/local/kafka_2.11-2.1.0]$ cat test.sink.txt
```

```
2  
3 file connector tes  
4
```

Step 5: When you examinr the topic with a console consumer, the output will be json format as it is the converter in the config file.

```
1 /usr/local/kafka_2.11-2.1.0]$ ./bin/kafka-console-c  
2  
3 {"schema":{"type":"string","optional":false},"paylo  
4
```

So, we have loaded data into Kafka from a file and loaded data off from Kafka to a file without writing our own producer or consumer code.

What happens if you restart the connector?

If you restart the connector, it will not read the whole file again as it knows exactly where it left off with the help of **offset**, which in this scenario is persisted to `"/tmp/connect.offsets"`. Offsets are configured through the worker properties named **offset.storage.file.filename**.

```
1 /usr/local/kafka_2.11-2.1.0]$ cat /tmp/connect.off  
2  
3 ??srjava.util.HashMap???`?F  
4 loadFactorI      thresholexp?@  
5 ur[B??T?xp-["local-fi  
6
```

What is the difference between Apache Kafka and Confluent Kafka?

Confluent Platform includes Apache Kafka, and in addition

1) to Java client it also includes C, C++, Python, and Go.

2) to file connector it has connectors for many others like **HDFS, JDBC, JMS, AWS S3, HBase, Elastic Search, MySQL, PostgreSQL, MongoDB, Amazon Kinesis**, etc.

3) Confluent has a REST API, which is by default available at <http://localhost:8083>. A few endpoints are:

GET /connectors – list all connectors in use.

GET /connectors/{name} – details about a given connector.

GET /connectors/{name}/status – current status of the connector.

POST /connectors – creates a new connector by posting a request body in JSON.

DELETE /connectors/{name} – deletes a connector by gracefully stopping all tasks and deleting its configuration.

GET /connector-plugins – lists all connector plugins installed in the Kafka Connect cluster.

When will you use a Connector as opposed to writing your own client in Java?

You will use a Kafka source & sink connector to source/sink systems that you did not write and can't or won't modify their code. For data stores where a connector already exists, Connect can be used by non-developers who will only need to **configure** the connectors.

You will use a Kafka client written in Java or Python when you are a developer, and you want to connect an application to Kafka and can modify the code of the application to push data into Kafka or pull data from Kafka.

What is a kafkacat?

kafkacat is a command line utility that you can use to test and debug Apache Kafka deployments. It is like the kafka-console-producer & kafka-console-consumer we saw earlier, but a lot more powerful. kafkacat is an open-source utility that is available at <https://github.com/edenhill/kafkacat>.

◀ 02: Apache Kafka multi-broker cluster tutorial

More HDFS & NameNode interview Q&As ▶

Disclaimer

The contents in this Java-Success are copyrighted and from EmpoweringTech pty ltd. The EmpoweringTech pty ltd has the right to correct or enhance the current content without any prior notice. These are general advice only, and one needs to take his/her own circumstances into consideration. The EmpoweringTech pty ltd will not be held liable for any damages caused or alleged to be caused either directly or indirectly by these materials and resources. Any trademarked names or labels used in this blog remain the property of their respective trademark owners. Links to external sites do not imply endorsement of the linked-to sites. [Privacy Policy](#).

