Tutorial: Moving Data In and Out of Kafka

This tutorial provides a hands-on look at how you can move data into and out of Apache Kafka® without writing a single line of code. It is helpful to review the concepts for Kafka Connect in tandem with running the steps in this guide to gain a deeper understanding. At the end of this tutorial you will be able to:

- Use Confluent CLI to manage Confluent services, including starting a single connect worker in distributed mode and loading and unloading connectors.
- Read data from a file and publish to a Kafka topic.
- Read data from a Kafka topic and publish to file.
- Integrate Schema Registry with a connector.

To demonstrate the basic functionality of Kafka Connect and its integration with the Confluent Schema Registry, a few local standalone Kafka Connect processes with connectors are run. You can insert data written to a file into Kafka and write data from a Kafka topic to the console. If you are using JSON as the Connect data format, see the instructions here for a tutorial that does not include Schema Registry.

Tip

These instructions assume you are installing Confluent Platform by using the Confluent CLI. For more information, see On-Premises Deployments.

Start the services

Prerequisites

- Confluent Platform
- Confluent CLI (requires separate installation)

In this guide, we are assuming services will run on `localhost` with default properties.

Tip

If not already in your PATH, add Confluent's `bin` directory by running: `export PATH=<path-to-confluent>/bin:$PATH`

Now that Confluent's `bin` directory is now included in your `PATH` variable, you can start all the services with the Confluent CLI `confluent local` commands:

```
confluent local start
```

Important

The confluent local commands are intended for a single-node development environment and are not suitable for a production environment. The data that are produced are transient and are intended to be temporary. For production-ready workflows, see Install and Upgrade.

Every service will start in order, printing a message with its status:

```
Starting zookeeper
zookeeper is [UP]
Starting kafka
kafka is [UP]
Starting schema-registry
schema-registry is [UP]
Starting kafka-rest
kafka-rest is [UP]
Starting connect
connect is [UP]
Starting ksql-server
ksql-server is [UP]
Starting control-center
control-center is [UP]
```

You may choose to open Connect's log to make sure the service has started successfully:

```
confluent local log connect
```

If an error occurred while starting services with Confluent CLI confluent local commands, you may access the logs of each service in one place by navigating to the directory where these logs are stored. For example:

```
# Show the log directory
confluent local current

/tmp/confluent.w1CpYsaI
# Navigate to the log directory
cd /tmp/confluent.w1CpYsaI
# View the log
less connect/connect.stderr
```

For complete details on getting these services up and running see the Confluent Platform installation documentation.

Read File Data with Connect

To startup a FileStreamSourceConnector that reads structured data from a file and exports the data into Kafka, using Schema Registry to inform Connect of their structure, we will use one of the supported connector configurations that come pre-defined with Confluent CLI confluent local commands. To get the list of all the pre-defined connector configurations, run:

```
  confluent local list connectors

Bundled Predefined Connectors (edit configuration under etc/):
   elasticsearch-sink
   file-source
   file-sink
   jdbc-source
   jdbc-sink
   hdfs-sink
   s3-sink
```

The pre-configured connector we will use first is called `file-source` and its configuration file is located at `./etc/kafka/connect-file-source.properties`. Below is an explanation of the contents:

```
# User defined connector instance name.
name=file-source
# The class implementing the connector
connector.class=FileStreamSource
# Maximum number of tasks to run for this connector instance
tasks.max=1
# The input file (path relative to worker's working directory)
# This is the only setting specific to the FileStreamSource
file=test.txt
# The output topic in Kafka
topic=connect-test
```

If choosing to use this tutorial without Schema Registry, you must also specify the `key.converter` and `value.converter` properties to use

`org.apache.kafka.connect.json.JsonConverter`. This will override the converters' settings for this connector only.

We are now ready to load the connector, but before we do that, let's seed the file with some sample data. Note that the connector configuration specifies a relative path for the file, so you should create the file in the same directory that you will run the Kafka Connect worker from.

```
for i in {1..3}; do echo "log line $i"; done > test.txt
```

Next, start an instance of the FileStreamSourceConnector using the configuration file you defined above. You can easily do this from the command line using the Confluent CLI confluent local commands as follows:

```
  confluent local load file-source

{
  "name": "file-source",
  "config": {
    "connector.class": "FileStreamSource",
    "tasks.max": "1",
    "file": "test.txt",
    "topics": "connect-test",
    "name": "file-source"
  },
  "tasks": []
}
```

Upon success it will print a snapshot of the connector's configuration. To confirm which connectors are loaded any time, run:

```
  confluent local status connectors

[
  "file-source"
]
```

Tip

With the Confluent Platform running and the CLI installed, you can find the path to the Kafka Connect log file at: `$(confluent local current)/connect/connect.stdout`. For example, to search the file for errors you can run: `cat $(confluent local current)/connect/connect.stdout | grep ERROR`

You will get a list of all the loaded connectors in this worker. The same command supplied with the connector name will give you the status of this connector, including an indication of whether the connector has started successfully or has encountered a failure. For instance, running this command on the connector we just loaded would give us:

```
  confluent local status file-source

{
  "name": "file-source",
  "connector": {
    "state": "RUNNING",
    "worker_id": "192.168.10.1:8083"
  },
  "tasks": [
    {
      "state": "RUNNING",
      "id": 0,
      "worker_id": "192.168.10.1:8083"
    }
  ]
}
```

Soon after the connector starts, each of the three lines in our log file should be delivered to Kafka, having registered a schema with Schema Registry. One way to validate that the data is there is to use the console consumer in another console to inspect the contents of the topic:

```
kafka-avro-console-consumer --bootstrap-server localhost:9092 --topic connect-test --from-beginning
"log line 1"
"log line 2"
"log line 3"
```

Note that we use the `kafka-avro-console-consumer` because the data has been stored in Kafka using Avro format. This consumer uses the Avro

converter that is bundled with Schema Registry in order to properly lookup the schema for the Avro data.

Write File Data with Connect

Now that we have written some data to a Kafka topic with Connect, let's consume that data with a downstream process. In this section, we will load a sink connector to the worker in addition to the source that we started in the last section. The sink will write messages to a local file. This connector is also pre-defined in Confluent CLI confluent local commands under the name `file-sink`. Below is the connector's configuration as it is stored in `etc/kafka/connect-file-sink.properties`:

```
# User defined name for the connector instance
name=file-sink
# Name of the connector class to be run
connector.class=FileStreamSink
# Max number of tasks to spawn for this connector instance
tasks.max=1
# Output file name relative to worker's current working directory
# This is the only property specific to the FileStreamSink connector
file=test.sink.txt
# Comma separate input topic list
topics=connect-test
```

Note that the configuration contains similar settings to the file source. A key difference is that multiple input topics are specified wit `topics` whereas the file source allows for only one output topic specified with `topic`.

Now start the FileStreamSinkConnector. The sink connector will run within the same worker as the source connector, but each connector task will have its own dedicated thread.

```
  confluent local load file-sink

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

To make sure the sink connector is up and running, use the Confluent CLIconfluent local status command to get the state of this specific connector:

```
  confluent local status file-sink

{
  "name": "file-sink",
  "connector": {
    "state": "RUNNING",
    "worker_id": "192.168.10.1:8083"
  },
  "tasks": [
    {
      "state": "RUNNING",
      "id": 0,
      "worker_id": "192.168.10.1:8083"
    }
  ]
}
```

as well as the list of all loaded connectors:

```
  confluent local status connectors

[
  "file-source",
  "file-sink"
]
```

Tip

Because of the rebalancing that happens between worker tasks every time a connector is loaded, a call to `confluent local status <connector-name>` might not succeed immediately after a new connector is loaded. Once rebalancing completes, such calls will be able to return the actual status of a connector.

By opening the file `test.sink.txt` you should see the two log lines written to it by the sink connector.

Now, with both connectors running, we can see data flowing end-to-end in real time. To check this out, use another terminal to tail the output file:

```
tail -f test.sink.txt
```

and in a different terminal start appending additional lines to the text file:

```
for i in {4..1000}; do echo "log line $i"; done >> test.txt
```

You should see the lines being added to `test.sink.txt`. The new data was picked up by the source connector, written to Kafka, read by the sink connector from Kafka, and finally appended to the file.

```
"log line 1"
"log line 2"
"log line 3"
"log line 4"
"log line 5"
 ...
```

After you are done experimenting with reading from and writing to a file with Connect, you have a few options with respect to shutting down the connectors:

- Unload the connectors but leave the Connect worker running.

```
confluent local unload file-source
confluent local unload file-sink
```

- Stop the Connect worker altogether.

```
  confluent local stop connect
Stopping connect
connect is [DOWN]
```

- Stop the Connect worker as well as all the rest Confluent services.

```
confluent local stop
```

Your output should resemble:

```
Stopping control-center
control-center is [DOWN]
Stopping ksql-server
ksql-server is [DOWN]
Stopping connect
connect is [DOWN]
Stopping kafka-rest
kafka-rest is [DOWN]
Stopping schema-registry
schema-registry is [DOWN]
Stopping kafka
kafka is [DOWN]
Stopping zookeeper
zookeeper is [DOWN]
```

- Stop all the services and wipe out any data of this particular run of Confluent services.

```
confluent local destroy
```

Your output should resemble:

```
Stopping control-center
control-center is [DOWN]
Stopping ksql-server
ksql-server is [DOWN]
Stopping connect
connect is [DOWN]
Stopping kafka-rest
kafka-rest is [DOWN]
Stopping schema-registry
schema-registry is [DOWN]
Stopping kafka
kafka is [DOWN]
Stopping zookeeper
zookeeper is [DOWN]
Deleting: /var/folders/ty/rqbqmjv54rg_v10ykmrgd1_80000gp/T/confluent.PkQpsKfE
```

Both source and sink connectors can track offsets, so you can start and stop the process any number of times and add more data to the input file and both will resume where they previously left off.

The connectors demonstrated in this tutorial are intentionally simple so no additional dependencies are necessary. Most connectors will require a bit more configuration to specify how to connect to the source or sink system and what data to copy, and for many you will want to execute on a Kafka Connect cluster for scalability and fault tolerance. To get started with Kafka Connect you'll want to see the user guide for more details on running and managing Kafka Connect, including how to run in distributed mode. The Connectors section includes details on configuring and deploying the connectors that ship with Confluent Platform.

Tip

The easiest way to create, configure, and manage connectors is with Confluent Control Center. To learn more about Control Center, see Confluent Control Center.

Please report any inaccuracies on this page or suggest an edit.

8 Votes

Last updated on Oct 17, 2019.