Configuring Connectors

Connector configurations are key-value mappings. For standalone mode these are defined in a properties file and passed to the Connect process on the command line. In distributed mode, they will be included in the JSON payload sent over the REST API for the request that creates (or modifies) the connector. When choosing to deploy in standalone or distributed mode, it is useful to review the recommendations here. Both standalone and distributed connectors can be configured by using the Confluent Control Center which gives a graphical interface to update connector configurations. Most configurations are connector dependent, but there are a few settings common to all connectors:

- `name` - Unique name for the connector. Attempting to register again with the same name will fail.
- `connector.class` - The Java class for the connector
- `tasks.max` - The maximum number of tasks that should be created for this connector. The connector may create fewer tasks if it cannot achieve this level of parallelism.
- `key.converter` - (optional) Override the default key converter class set by the worker.
- `value.converter` - (optional) Override the default value converter class set by the worker.

To pass configuration parameters to key and value converters, prefix them with `key.converter.` or `value.converter.` as you would in the worker configuration when defining default converters. Note that these are only used when the corresponding converter configuration is specified in the `key.converter` or `value.converter` properties.

Note that these parameters are not used unless the corresponding converter configuration is specified in the connector configuration.

Sink connectors also have one additional option to control their input:

- `topics` - A list of topics to use as input for this connector

For other options, consult the documentation forindividual connectors.

Standalone Example

Below is an example of a standalone connector configuration for the supported FileSinkConnector. Note the common configuration `name`, `connector.class`, `tasks.max`, and `topics`, and one FileStreamSinkConnector specific configuration `file` is specified. The file containing this configuration should be added as shown here.

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

Distributed Example

You can use the REST API to manage the connectors running on workers in distributed mode. Here is a simple example that creates a FileSinkConnector as in the standalone example. Note the URL is pointing to localhost indicating that the connect worker has been started on the same host where the *curl* command is run. Instructions for starting a worker in distributed mode arehere.

```
  curl -X POST -H "Content-Type: application/json" --data '{"name": "local-file-sink", "config": {"connector.class":"Fi
leStreamSinkConnector", "tasks.max":"1", "file":"test.sink.txt", "topics":"connect-test" }}' http://localhost:8083/con
nectors
# Or, to use a file containing the JSON-formatted configuration
# curl -X POST -H "Content-Type: application/json" --data @config.json http://localhost:8083/connectors
```

Expand Content

To create a connector, you start the workers and then make a REST request to create a connector as above. Unlike many other systems, all nodes in Kafka Connect can respond to REST requests, including creating, listing, modifying, and destroying connectors (see the REST API section for details).
