# Monitoring Connectors

Kafka Connect's REST API enables administration of the cluster. This includes APIs to view the configuration of connectors and the status of their tasks, as well as to alter their current behavior (e.g. changing configuration and restarting tasks).

Once a Kafka Connect cluster is up and running, you can monitor and modify it. In this section, we go over a few common management tasks done via the REST API.

## Using the REST Interface

Since Kafka Connect is intended to be run as a service, it also supports a REST API for managing connectors. By default this service runs on port `8083`. When executed in distributed mode, the REST API will be the primary interface to the cluster. You can make requests to any cluster member; the REST API automatically forwards requests if required.

Although you can use the standalone mode just by submitting a connector on the command line, it also runs the REST interface. This is useful for getting status information, adding and removing connectors without stopping the process, and more.

---

## Connector and Task Status

You can use the REST API to view the current status of a connector and its tasks, including the ID of the worker to which each was assigned.

Connectors and their tasks publish status updates to a shared topic (configured with `status.storage.topic`) which all workers in the cluster monitor. Because the workers consume this topic asynchronously, there is typically a short delay before a state change is visible through the status API. The following states are possible for a connector or one of its tasks:

- `UNASSIGNED`: The connector/task has not yet been assigned to a worker.
- `RUNNING`: The connector/task is running.
- `PAUSED`: The connector/task has been administratively paused.
- `FAILED:` The connector/task has failed (usually by raising an exception, which is reported in the status output).

In most cases, connector and task states will match, though they may be different for short periods of time when changes are occurring or if tasks have failed. For example, when a connector is first started, there may be a noticeable delay before the connector and its tasks have all transitioned to the `RUNNING` state. States will also diverge when tasks fail since Connect does not automatically restart failed tasks.

It's sometimes useful to temporarily stop the message processing of a connector. For example, if the remote system is undergoing maintenance, it would be preferable for source connectors to stop polling it for new data instead of filling logs with exception spam. For this use case, Connect offers a pause/resume API. While a source connector is paused, Connect will stop polling it for additional records. While a sink connector is paused, Connect will stop pushing new messages to it. The pause state is persistent, so even if you restart the cluster, the connector will not begin message processing again until the task has been resumed. Note that there may be a delay before all of a connector's tasks have transitioned to the `PAUSED` state since it may take time for them to finish whatever processing they were in the middle of when being paused. Additionally, failed tasks will not transition to the `PAUSED` state until they have been restarted.

---

## Common REST Examples

Below are a few common activities that are done over the REST API. These examples are shown using a worker running on localhost configurations and a connector named `local-file-sink`. We use the utility `jq` in some of the examples to format the response, but this is not

Expand Content

required. The examples are intentionally simple. Advanced use of the REST API should consider the REST API references.

Get the worker's version information

```
  curl localhost:8083/ | jq
{
  "version": "0.10.0.1-cp1",
  "commit": "ea5fcd28195f168b"
}
```

List the connector plugins available on this worker

```
  curl localhost:8083/connector-plugins | jq
[
  {
    "class": "io.confluent.connect.replicator.ReplicatorSourceConnector"
  },
  {
    "class": "org.apache.kafka.connect.file.FileStreamSourceConnector"
  },
  {
    "class": "io.confluent.connect.jdbc.JdbcSinkConnector"
  },
  {
    "class": "io.confluent.connect.jdbc.JdbcSourceConnector"
  },
  {
    "class": "io.confluent.connect.hdfs.HdfsSinkConnector"
  },
  {
    "class": "org.apache.kafka.connect.file.FileStreamSinkConnector"
  }
]
```

Listing active connectors on a worker

```
  curl localhost:8083/connectors
["local-file-sink"]
```

Restarting a connector

```
  curl -X POST localhost:8083/connectors/local-file-sink/restart
(no response printed if success)
```

Getting tasks for a connector

```
  curl localhost:8083/connectors/local-file-sink/tasks | jq
[
  {
    "id": {
      "connector": "local-file-sink",
      "task": 0
    },
    "config": {
      "task.class": "org.apache.kafka.connect.file.FileStreamSinkTask",
      "topics": "connect-test",
      "file": "test.sink.txt"
    }
  }
]
```

Restarting a task (no response printed if success)

```
curl -X POST localhost:8083/connectors/local-file-sink/tasks/0/restart
```

Pausing a connector (no response printed if success) (useful if downtime is needed for the system the connector interacts with)

```
curl -X PUT localhost:8083/connectors/local-file-sink/pause
```

Resuming a connector (no response printed if success)

```
curl -X PUT localhost:8083/connectors/local-file-sink/resume
```

Updating connector configuration

```
  curl -X PUT -H "Content-Type: application/json" --data '{"connector.class":"FileStreamSinkConnector","file":"test.sin
k.txt","tasks.max":"2","topics":"connect-test","name":"local-file-sink"}' localhost:8083/connectors/local-file-sink/co
nfig
{
  "name": "local-file-sink",
  "config": {
    "connector.class": "FileStreamSinkConnector",
    "file": "test.sink.txt",
    "tasks.max": "2",
    "topics": "connect-test",
    "name": "local-file-sink"
  },
  "tasks": [
    {
      "connector": "local-file-sink",
      "task": 0
    },
    {
      "connector": "local-file-sink",
      "task": 1
    }
  ]
}
```

Getting connector status

```
  curl localhost:8083/connectors/local-file-sink/status | jq
{
  "name": "local-file-sink",
  "connector": {
    "state": "RUNNING",
    "worker_id": "192.168.86.101:8083"
  },
  "tasks": [
    {
      "state": "RUNNING",
      "id": 0,
      "worker_id": "192.168.86.101:8083"
    },
    {
      "state": "RUNNING",
      "id": 1,
      "worker_id": "192.168.86.101:8083"
    }
  ]
}
```

Getting connector configuration

```
  curl localhost:8083/connectors/local-file-sink | jq
{
  "name": "local-file-sink",
  "config": {
    "connector.class": "FileStreamSinkConnector",
    "file": "test.sink.txt",
    "tasks.max": "2",
    "topics": "connect-test",
    "name": "local-file-sink"
  },
  "tasks": [
    {
      "connector": "local-file-sink",
      "task": 0
    },
    {
      "connector": "local-file-sink",
      "task": 1
    }
  ]
}
```

Deleting a connector (no response printed if success)

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

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

**2 Votes**
★★★★★

Last updated on Oct 17, 2019.