Kafka

**Apache Kafka® is *a distributed streaming platform***

A streaming platform has three key capabilities:

* Publish and subscribe to streams of records, similar to a message queue or enterprise messaging system.
* Store streams of records in a fault-tolerant durable way.
* Process streams of records as they occur.
* **Approach**

**Kafka Connec**t is a tool for scalably and reliably streaming data between Apache Kafka and other systems. It makes it simple to quickly define *connectors* that move large collections of data into and out of Kafka. Kafka Connect can ingest entire databases or collect metrics from all your application servers into Kafka topics, making the data available for stream processing with low latency. An export job can deliver data from Kafka topics into secondary storage and query systems or into batch systems for offline analysis.

Kafka Connect features include:

* **A common framework for Kafka connectors** - Kafka Connect standardizes integration of other data systems with Kafka, simplifying connector development, deployment, and management
* **Distributed and standalone modes** - scale up to a large, centrally managed service supporting an entire organization or scale down to development, testing, and small production deployments
* **REST interface** - submit and manage connectors to your Kafka Connect cluster via an easy to use REST API
* **Automatic offset management** - with just a little information from connectors, Kafka Connect can manage the offset commit process automatically so connector developers do not need to worry about this error prone part of connector development
* **Distributed and scalable by default** - Kafka Connect builds on the existing group management protocol. More workers can be added to scale up a Kafka Connect cluster.
* **Streaming/batch integration** - leveraging Kafka's existing capabilities, Kafka Connect is an ideal solution for bridging streaming and batch data systems
* Since Kafka Connect is intended to be run as a service, it also provides a REST API for managing connectors. The REST API server can be configured using the listeners configuration option. This field should contain a list of listeners in the following format: protocol://host:port,protocol2://host2:port2. Currently supported protocols are http and https. For example:

|  |  |
| --- | --- |
|  | listeners=[http://localhost:8080](http://localhost:8080/),[https://localhost:8443](https://localhost:8443/) |
|  |  |

* By default, if no listeners are specified, the REST server runs on port 8083 using the HTTP protocol. When using HTTPS, the configuration has to include the SSL configuration. By default, it will use the ssl.\* settings. In case it is needed to use different configuration for the REST API than for connecting to Kafka brokers, the fields can be prefixed with listeners.https. When using the prefix, only the prefixed options will be used and the ssl.\* options without the prefix will be ignored. Following fields can be used to configure HTTPS for the REST API:

Following fields can be used to configure HTTPS for the REST API:

* ssl.keystore.location
* ssl.keystore.password
* ssl.keystore.type
* ssl.key.password
* ssl.truststore.location
* ssl.truststore.password
* ssl.truststore.type
* ssl.enabled.protocols
* ssl.provider
* ssl.protocol
* ssl.cipher.suites
* ssl.keymanager.algorithm
* ssl.secure.random.implementation
* ssl.trustmanager.algorithm
* ssl.endpoint.identification.algorithm
* ssl.client.auth

The REST API is used not only by users to monitor / manage Kafka Connect. It is also used for the Kafka Connect cross-cluster communication. Requests received on the follower nodes REST API will be forwarded to the leader node REST API. In case the URI under which is given host reachable is different from the URI which it listens on, the configuration options rest.advertised.host.name, rest.advertised.port and rest.advertised.listener can be used to change the URI which will be used by the follower nodes to connect with the leader. When using both HTTP and HTTPS listeners, the rest.advertised.listener option can be also used to define which listener will be used for the cross-cluster communication. When using HTTPS for communication between nodes, the same ssl.\* or listeners.https options will be used to configure the HTTPS client.

##### [**Connectors and Tasks**](https://kafka.apache.org/documentation.html#connect_connectorsandtasks)

### To copy data between Kafka and another system, users create a Connector for the system they want to pull data from or push data to. Connectors come in two flavors: SourceConnectors import data from another system (e.g. JDBCSourceConnector would import a relational database into Kafka) and SinkConnectors export data (e.g. HDFSSinkConnector would export the contents of a Kafka topic to an HDFS file).

### Connectors do not perform any data copying themselves: their configuration describes the data to be copied, and the Connector is responsible for breaking that job into a set of Tasks that can be distributed to workers. These Tasks also come in two corresponding flavors: SourceTask and SinkTask.

### With an assignment in hand, each Task must copy its subset of the data to or from Kafka. In Kafka Connect, it should always be possible to frame these assignments as a set of input and output streams consisting of records with consistent schemas. Sometimes this mapping is obvious: each file in a set of log files can be considered a stream with each parsed line forming a record using the same schema and offsets stored as byte offsets in the file. In other cases it may require more effort to map to this model: a JDBC connector can map each table to a stream, but the offset is less clear. One possible mapping uses a timestamp column to generate queries incrementally returning new data, and the last queried timestamp can be used as the offset.

##### [**Streams and Records**](https://kafka.apache.org/documentation.html#connect_streamsandrecords)

### Each stream should be a sequence of key-value records. Both the keys and values can have complex structure -- many primitive types are provided, but arrays, objects, and nested data structures can be represented as well. The runtime data format does not assume any particular serialization format; this conversion is handled internally by the framework.

### In addition to the key and value, records (both those generated by sources and those delivered to sinks) have associated stream IDs and offsets. These are used by the framework to periodically commit the offsets of data that have been processed so that in the event of failures, processing can resume from the last committed offsets, avoiding unnecessary reprocessing and duplication of events.

##### [**Dynamic Connectors**](https://kafka.apache.org/documentation.html#connect_dynamicconnectors)

### Not all jobs are static, so Connector implementations are also responsible for monitoring the external system for any changes that might require reconfiguration. For example, in the JDBCSourceConnector example, the Connector might assign a set of tables to each Task. When a new table is created, it must discover this so it can assign the new table to one of the Tasks by updating its configuration. When it notices a change that requires reconfiguration (or a change in the number of Tasks), it notifies the framework and the framework updates any corresponding Tasks.

### 

**Events and Endpoints**

The following are the currently supported REST API endpoints:

* GET /connectors - return a list of active connectors
* POST /connectors - create a new connector; the request body should be a JSON object containing a string name field and an object config field with the connector configuration parameters
* GET /connectors/{name} - get information about a specific connector
* GET /connectors/{name}/config - get the configuration parameters for a specific connector
* PUT /connectors/{name}/config - update the configuration parameters for a specific connector
* GET /connectors/{name}/status - get current status of the connector, including if it is running, failed, paused, etc., which worker it is assigned to, error information if it has failed, and the state of all its tasks
* GET /connectors/{name}/tasks - get a list of tasks currently running for a connector
* GET /connectors/{name}/tasks/{taskid}/status - get current status of the task, including if it is running, failed, paused, etc., which worker it is assigned to, and error information if it has failed
* PUT /connectors/{name}/pause - pause the connector and its tasks, which stops message processing until the connector is resumed
* PUT /connectors/{name}/resume - resume a paused connector (or do nothing if the connector is not paused)
* POST /connectors/{name}/restart - restart a connector (typically because it has failed)
* POST /connectors/{name}/tasks/{taskId}/restart - restart an individual task (typically because it has failed)
* DELETE /connectors/{name} - delete a connector, halting all tasks and deleting its configuration

Kafka Connect also provides a REST API for getting information about connector plugins:

* GET /connector-plugins- return a list of connector plugins installed in the Kafka Connect cluster. Note that the API only checks for connectors on the worker that handles the request, which means you may see inconsistent results, especially during a rolling upgrade if you add new connector jars
* PUT /connector-plugins/{connector-type}/config/validate - validate the provided configuration values against the configuration definition. This API performs per config validation, returns suggested values and error messages during validation.
* **Refferals links**

[**https://kafka.apache.org/documentation.html#connect\_development**](https://kafka.apache.org/documentation.html#connect_development)