**Kafka Connect**

Kafka Connect is a framework for connecting Kafka with external systems such as databases, key-value stores, search indexes, and file systems, using so-called Connectors.

**Features of kafka connect are:**

A common framework for Kafka connectors

* Distributed and standalone modes
* REST interface
* Automatic offset management
* Distributed and scalable by default
* Streaming/batch integration

The main reasons to consider kafka connect are

* Data Centric Pipeline
* Flexibility and Scalability
* Reusability and Extensibility
* Auto-recovery After Failure
* Auto-failover
* Simple Parallelism

Kafka Connect Core Concepts

Kafka Connect Core Concepts include

* Connectors – the high-level abstraction that coordinates data streaming by managing tasks
* Tasks – the implementation of how data is copied to or from Kafka
* [Workers](https://docs.confluent.io/current/connect/concepts.html#connect-workers) – the running processes that execute connectors and tasks
* Converters – the code used to translate data between Connect and the system sending or receiving data
* Transforms – simple logic to alter each message produced by or sent to a connector
* Dead Letter Queue – how Connect handles connector errors

Here, we are going to discuss connectors

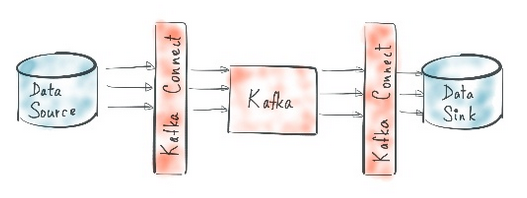
Connectors in Kafka Connect define where data should be copied to and from.

A **connector instance** is a logical job that is responsible for managing the copying of data between Kafka and another system.

All of the classes that implement or are used by a connector are defined in a **connector plugin**.

Both connector instances and connector plugins may be referred to as “connectors”, but it should always be clear from the context which is being referred to (e.g., “install a connector” refers to the plugin, and “check the status of a connector” refers to a connector instance).





Understanding Source and Sink Connector

**Source Connector**

A source connector ingests entire databases and streams table updates to Kafka topics. It can also collect metrics from all of your application servers and store these in Kafka topics, making the data available for stream processing with low latency.

**Sink Connector**

A sink connector delivers data from Kafka topics into secondary indexes such as Elasticsearch, or batch systems such as Hadoop for offline analysis.

**Configurations**

There are three main Parameters that are needed for every single connection in Kafka. These are

Name – Name of connector  
Connector.class – class/type of connector  
Task.max – maximum number of task that it can create.

Other important configurations needed for kafka connect include:  
Adv.host - Hostname to publish to ZooKeeper for clients to use  
Run test = 0 – to start faster

Ports should be free to use and the settings include port numbers for   
Zookeeper  
Landoop UI  
Rest Proxy  
Schema registry

Kafka connect ports  
JMX port  
Kafka broker

Then we need to provide worker.properties which has following parameters

Bootstrap.servers – IP  
Key.converter – to convert all data to json  
Key.converter.schema.enable – Boolean  
Value.converter  
Value.converter.schema.enable

(Note that internal key, value converters to be alwas json)

Then comes the rest API. Its must to be mentioned

Rest.port  
Rest.host.name

Then comes worker properties as per standalone or distributed. Here its mentioned as per standalone

Offset.storage.file.filename = standalone.offsets  
Offset.flush.intervals.ms = 10000 (or whatever required)

For state storage, we use

Config.storage.topic  
Status.storage.topic

**Two important questions to be answered before executing the connector:**

How I am serializing and deserializing my data?

How am I running kafka connect?

**Transforms in kafka connect**

A transform is a simple function that accepts one record as an input and outputs a modified record. All transforms provided by Kafka Connect perform simple but commonly useful modifications.

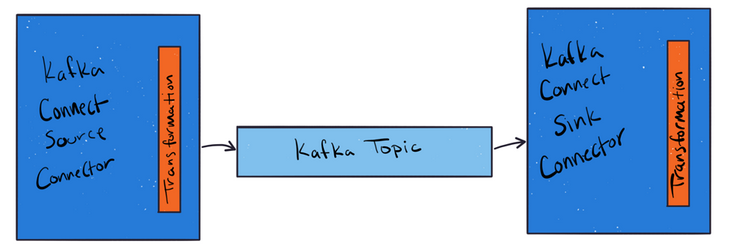
Connectors can be configured with transformations to make simple and lightweight modifications to individual messages. This can be convenient for minor data adjustments and event routing, and multiple transformations can be chained together in the connector configuration. However, more complex transformations and operations that apply to multiple messages are best implemented with KSQL and Kafka Streams.

**Single Message Transformations (SMTs)**

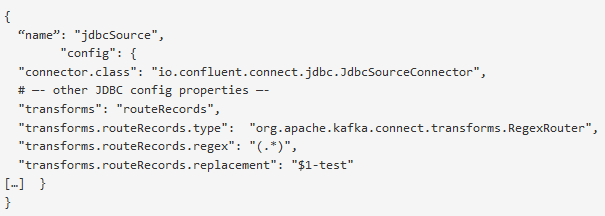
The Kafka Connect API also provides a simple interface for manipulating records as they flow through both the source and sink side of your data pipeline. This API is known as Single Message Transforms (SMTs), and as the name suggests, it operates on every single message in your data pipeline as it passes through the Kafka Connect connector.

Connectors are classified as either sources or sinks—they either pull data from a system upstream of Kafka or push data downstream from Kafka. These connectors can be configured to take advantage of transforms on either side. Source connectors pass records through the transformation before writing to the Kafka topic, and sink connectors pass records through the transformation before writing to the sink.

* Source Transforms before writing to topic
* Sink Transforms before writing to target

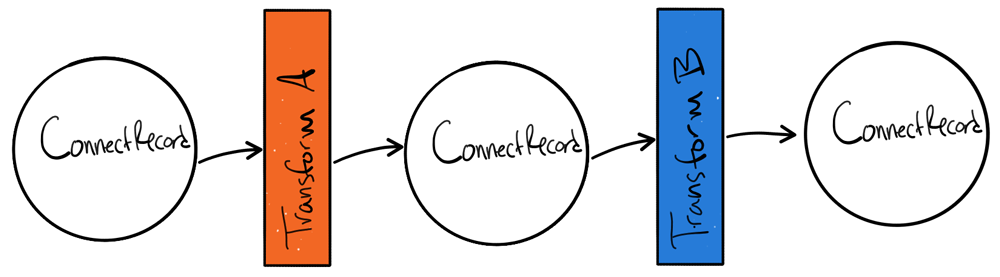


Some common uses for transforms are:

* Renaming fields
* Masking values
* Routing records to topics based on a value
* Converting or inserting timestamps into the record
* Manipulating keys, like setting a key fro m a field’s value  
    SMTs Configuration snippet

**Performing multiple transformations**

Sometimes more than one transformation is necessary. Kafka Connect supports defining multiple transformations that are chained together in the configuration. These messages flow through the transformations in the same order in which they are defined in the transforms property.



Multiple transformation example snippet

“transforms”:”createKey,extractInt”, “transforms.createKey.type”:”org.apache.kafka.connect.transforms.ValueToKey”,

“transforms.createKey.fields”:”c1”, “transforms.extractInt.type”:”org.apache.kafka.connect.transforms.ExtractField$Key”,

“transforms.extractInt.field”:”c1”

There are many kinds of transformations available. Note that Transforms are a powerful concept, but they should only be used for simple, limited mutations of the data. Don’t call out to external APIs or store state, and don’t attempt any heavy processing in the transform. Heavier transforms and data integrations should be handled in the stream processing layer between connectors using a stream processing solution such as Kafka Streams or KSQL. Transforms cannot split one message into many, nor can they join other streams for enrichment or do any kinds of aggregations. Such activities should be left to stream processors.

The available transformations are as follows

| **Transform** | **Description** |
| --- | --- |
| Cast | Cast fields or the entire key or value to a specific type (for example, to force an integer field to a smaller width). |
| Drop | Drop either a key or a value from a record and set it to null. |
| ExtractField | Extract the specified field from a Struct when schema present, or a Map in the case of schemaless data. Any null values are passed through unmodified. |
| ExtractTopic | Replace the record topic with a new topic derived from its key or value. |
| Filter | Include or drop records that match the filter.condition predicate. |
| Flatten | Flatten a nested data structure. This generates names for each field by concatenating the field names at each level with a configurable delimiter character. |
| HoistField | Wrap data using the specified field name in a Struct when schema present, or a Map in the case of schemaless data. |
| InsertField | Insert field using attributes from the record metadata or a configured static value. |
| MaskField | Mask specified fields with a valid null value for the field type. |
| MessageTimeStampRouter | Update the record’s topic field as a function of the original topic value and the record’s timestamp field. |
| RegexRouter | Update the record topic using the configured regular expression and replacement string. |
| ReplaceField | Filter or rename fields. |
| SetSchemaMetadata | Set the schema name, version, or both on the record’s key or value schema. |
| TimestampConverter | Convert timestamps between different formats such as Unix epoch, strings, and Connect Date and Timestamp types. |
| TimestampRouter | Update the record’s topic field as a function of the original topic value and the record timestamp. |
| TombstoneHandler | Manage tombstone records. A tombstone record is defined as a record with the entire value field being null, whether or not it has ValueSchema. |
| ValueToKey | Replace the record key with a new key formed from a subset of fields in the record value. |