**Kafka Sink Connector**

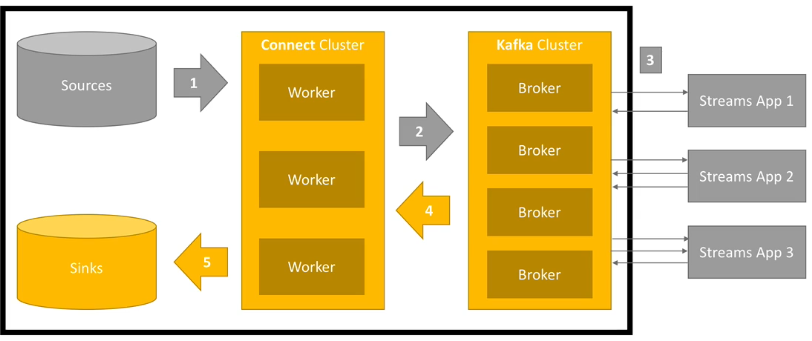
**Source Connector**

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.

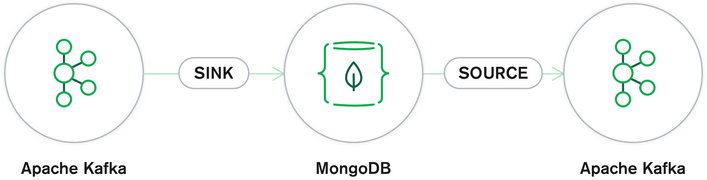
The connect cluster for both source and sink in Kafka is same. The difference lies majorly in configurations that are set for sending data out of kafka to a sink (some database) as shown in step 4 and 5 of the following figure.



**Sink as MongoDB**

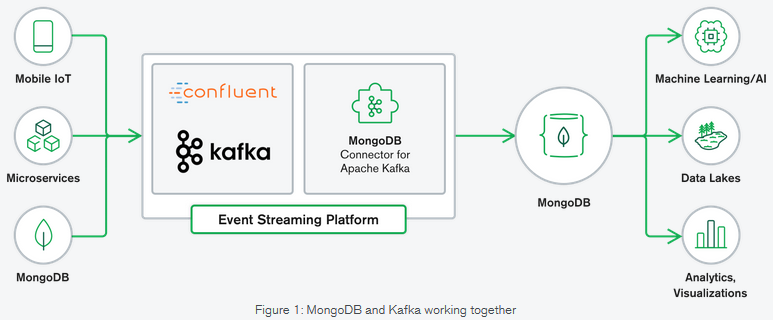
Here, we are going to use sink as MongoDB

MongoDB is the world’s most popular modern database built for handling massive volumes of heterogeneous data. Together MongoDB and Kafka make up the heart of many modern data architectures today.



Integrating Kafka with external systems like MongoDB is done through the use of Kafka Connect. This API enables users to leverage ready-to-use components that can stream data from external systems into Kafka topics, and stream data from Kafka topics into external systems. The official MongoDB Connector for Apache Kafka® is developed and supported by MongoDB engineers and verified by Confluent. The connector, now released in Beta, enables MongoDB to be configured as both a sink and a source for Apache Kafka.

The architecture as per MongoDB becomes



**Why MongoDB and Apache Kafka?**

MongoDB and Kafka are at the heart of modern data architectures. Kafka is designed for boundless streams of data that sequentially write events into commit logs, allowing real-time data movement between your services.

**Using MongoDB as a sink from a Kafka Topic**

Since as per source API of kafka connect, we used to extract data from some source and put in in kafka by giving some identity as kafka topics. Sink and source API are conceptually opposite of each other so we do change the configurations accordingly. The major configuration items remain the same in Sink API as that of source API but since now, we have to send data in a database, we have to make sure the segregation of data in the database tables. For this purpose, we do add configuration parameter “Topics” and “collection” to exclusively identify the which topic data is to be forwarded to sink and where to put it in sink respectively.

The properties and configurations for **source** were as follows

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

file- file name  
topics – topic name

If Distributed mode, no need to have separate file for worker properties. Simply put that here.

Setting up 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)

Internal.key.converter  
Internal.key.converter.schema.enabled  
Internal.value.converter  
Internal.value.converter.schema.enabled

These are all for within kafka connect framework but are not usually exposed to the user

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

Now, Setting up connector

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

topics – topic name

Now comes Key and value converters. Since input topic has a schema so we will need it here

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

Then we prove sink connector based configurations

Connection.url – i.e., connection url as per the database  
Connection.user – same as that is set   
connection.password – same as that is set  
insert.node – upsert is recommended (from insert, update and upsert)

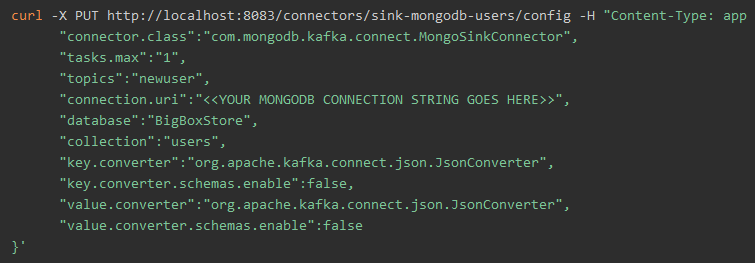
Other properties needed e.g if I want the primary to be offset + partition

Pk.mode = kafka  
pk.fields = \_conncet\_topic,\_connect\_partition,\_connect\_offset  
fields.whitelist=id,created\_at,text,lang,is\_retweeted (to filter data from stream of tweets)  
auto.create=true  
auto.evolve=true

**Using MongoDB as a sink from a Kafka Topic**

Now as we have understood how sink differs from source, we can use sink with any Database needed. Here, we are going to see as per MongoDB.

In this example snippet for MongoDB, the contact information is placed in a Kafka topic, “newuser” for shared use and we would configure MongoDB as a sink to the Kafka Topic. This would allow new users information to propagate to a “users” collection in MongoDB. To configure the connector for this scenario we can issue a REST API call to the Connector service as follows:



Here, we have

Content-Type i.e., app  
connector.class – class/type of connector as we defining it to be MongoSinkConnector  
task.max - maximum number of task that it can create  
“topics” : “newuser” – this is the new parameter to tell sink which topic the data to be taken from kafka  
connection.url – MongoDB url here  
“database” – which database in MongoDB to sink into  
“collection” : “user” – which table in the databse to sink into from topics  
Now the properties to convert the data  
key.converter  
key.converter.schema.enable  
value.converter  
value.converter.schema.enable

**Kafka Connect REST API**

Since Rest API is highly used, Lets see REST API as well

All actions performed by Landoop Kafka Connect UI are actually triggering REST API calls to Kafka connect.

These include

1. Get Worker Info  
   curl -s IP:PORT/ | jq
2. List connectors available on a worker  
   curl -s IP:PORT/connector-plugins |jq
3. Ask about active connectors  
   curl -s IP:PORT/connectors |jq
4. Get info about connector task and config  
   curl -s IP:PORT/connectors/source../tasks | jq
5. Get connector status  
   curl -s IP:PORT/connectors/connector\_name/status | jq
6. Pause, resume and restart a connector

curl -XPUT 'localhost:8083/connectors/sink\_connector/pause'

curl -XPUT 'localhost:8083/connectors/sink/resume'

curl -XPOST 'localhost:8083/connectors/sink/restart'

1. Get connector configuration  
   curl -s IP:PORT/connectors/connector | jq
2. Delete a connector  
   curl -s -X DELETE IP:PORT/connectors/connector
3. Create a new connector  
   curl -s -X POST -H (content type) –data (configurations and parameters) http://IP:PORT/connectors/ |ig
4. Update Connector Configurations  
   curl -s -X PUT -H (content type) –data (configurations and parameters) http://IP:PORT/connectors/ |ig

**Resources**

<https://www.mongodb.com/blog/post/getting-started-with-the-mongodb-connector-for-apache-kafka-and-mongodb-atlas>

<https://www.mongodb.com/kafka-connector>

<https://www.learningcrux.com/video/apache-kafka-series-kafka-connect-handson-learning/5/1>