**Support documents:**

<https://medium.com/expedia-group-tech/mongo-change-streams-in-production-97a07c7c0420>

<https://cwiki.apache.org/confluence/display/KAFKA/KIP-618%3A+Exactly-Once+Support+for+Source+Connectors>

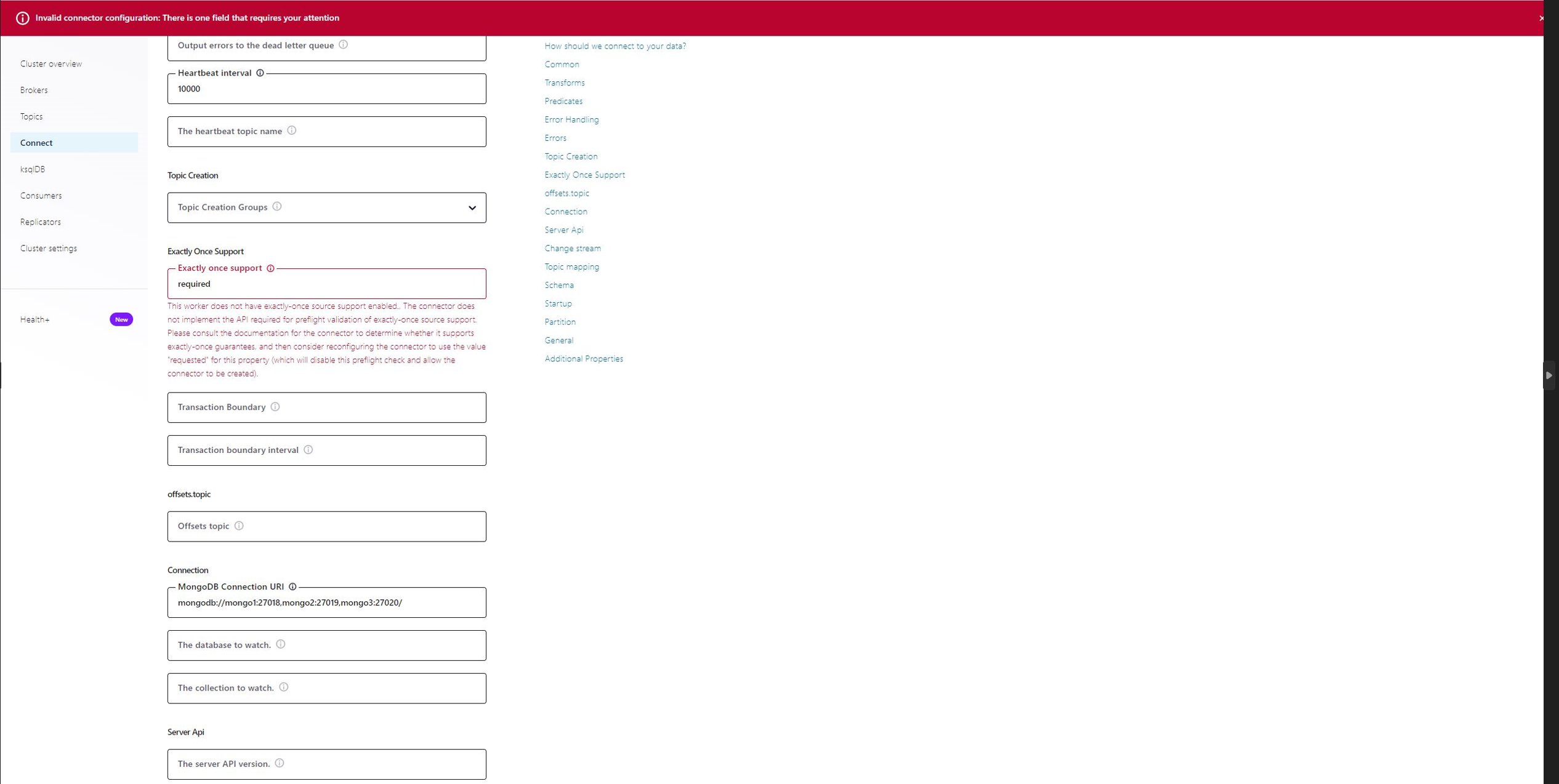
<https://stackoverflow.com/questions/59389861/can-kafka-connect-mongo-source-run-as-cluster-max-tasks-1>

**Prerequisites:**

* MongoDB **4.0 or higher** is needed because ChangeStream implementation must be present
* MongoDB must be present in replicaSet configuration because the ChangeStream watches the **oplog**, which is present only in ReplicaSet configuration

**Limitations:**

* The MongoSource connector **uses only 1 task per worker node**, cannot be configured for more(please see versions used below)
* EOS is not supported for MongoSource connector(please see versions used below)



**Versions of libraries and images used:**

* MongoSource connector: mongodb/kafka-connect-mongodb:1.9.1
* Kafka Connect image: confluentinc/cp-server-connect-base:latest
* MongoDB image: mongo:4.4
* Kafka brokers image: confluentinc/cp-server:7.3.1

**Infrastructure used :**

Attached docker-compose.yml file used to spin up:

* Mongodb replicaSet – which will be used as the source for the connector.
* Kafka cluster – used as target for the connector
* Kafka Connect cluster with 2 worker nodes – used to deploy the source connector
* Kafka Control Center to monitor the source connector and topics created

**Create new source connector**:

New source connector can be created setting the following properties: <https://www.mongodb.com/docs/kafka-connector/current/source-connector/configuration-properties/#std-label-source-configuration-index>

**First Connector for collection testCollection1:**

*curl -X POST -H "Content-Type: application/json" --data '*

*{*

*"name": "mongo-cursor1",*

*"config": {*

*"name": "mongo-cursor1",*

***"connector.class": "com.mongodb.kafka.connect.MongoSourceConnector",***

*"tasks.max": "1",*

*"key.converter": "org.apache.kafka.connect.json.JsonConverter",*

*"value.converter": "org.apache.kafka.connect.json.JsonConverter",*

*"errors.retry.timeout": "20000",*

*"errors.retry.delay.max.ms": "1000",*

***"connection.uri": "mongodb://mongo1:27018,mongo2:27019,mongo3:27020/",***

*"pipeline": "",*

***"batch.size": "100",***

***"heartbeat.interval.ms": "10000",***

***"database": "test",***

***"collection": "testCollection1",***

***"offset.partition.name": "mongo-source-token-invalid.0"***

*}*

*}' http://localhost:8083/connectors -w "\n"*

**Second Connector for collection testCollection2:**

*curl -X POST -H "Content-Type: application/json" --data '*

{

"name": "mongo-cursor2",

"config": {

"name": "mongo-cursor2",

**"connector.class": "com.mongodb.kafka.connect.MongoSourceConnector",**

"tasks.max": "1",

"key.converter": "org.apache.kafka.connect.json.JsonConverter",

"value.converter": "org.apache.kafka.connect.json.JsonConverter",

"errors.retry.timeout": "20000",

"errors.retry.delay.max.ms": "1000",

**"connection.uri": "mongodb://mongo1:27018,mongo2:27019,mongo3:27020/",**

"pipeline": "",

**"batch.size": "100",**

**"heartbeat.interval.ms": "10000",**

**"database": "test",**

**"collection": "testCollection2",**

**"offset.partition.name": "mongo-source-token-invalid.0"**

}

}' http://localhost:8083/connectors -w "\n"

Each connector instance opens a ChangeStream for the collection(s) used. In our scenario there are 2 cursors opened, one for each collection.

In case we need only one cursor(opening many cursors is not recommended!), we can create a connector for both collections using pipeline property as below:

*curl -X POST -H "Content-Type: application/json" --data '*

*{*

*"name": "mongo-cursor1",*

*"config": {*

*"name": "mongo-cursor1",*

*"connector.class": "com.mongodb.kafka.connect.MongoSourceConnector",*

*"tasks.max": "3",*

*"key.converter": "org.apache.kafka.connect.json.JsonConverter",*

*"value.converter": "org.apache.kafka.connect.json.JsonConverter",*

*"errors.retry.timeout": "20000",*

*"errors.retry.delay.max.ms": "1000",*

*"connection.uri": "mongodb://mongo1:27018,mongo2:27019,mongo3:27020/",*

***"pipeline": "[{\"$match\":{\"$and\":[{\"ns.db\":{\"$regex\":/^test$/}},{\"ns.coll\":{\"$regex\":/^test.\*/}},{\"docId\":{\"$mod\" : [2,0]}}]}}]",***

*"batch.size": "100",*

*"topic.prefix": "mongo",*

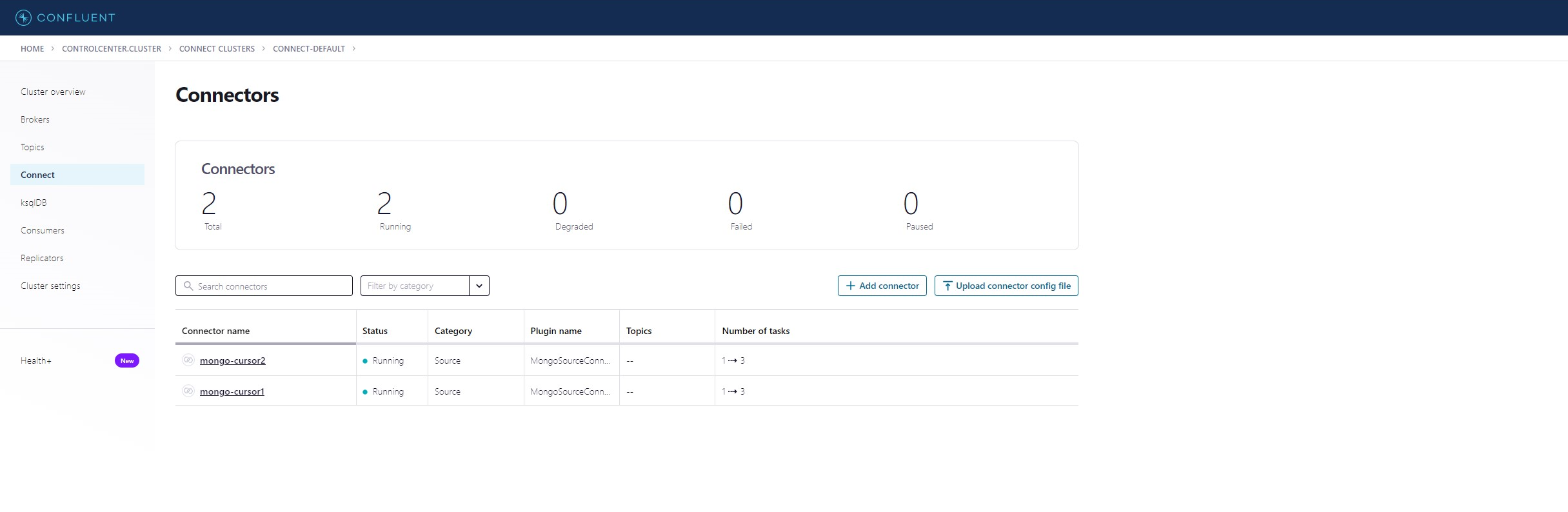
*"heartbeat.interval.ms": "10000",*

*"offset.partition.name": "mongo-source-token-invalid"*

*}*

*}' http://localhost:8083/connectors -w "\n"*

These 2 connectors will be noticed in Control Center:



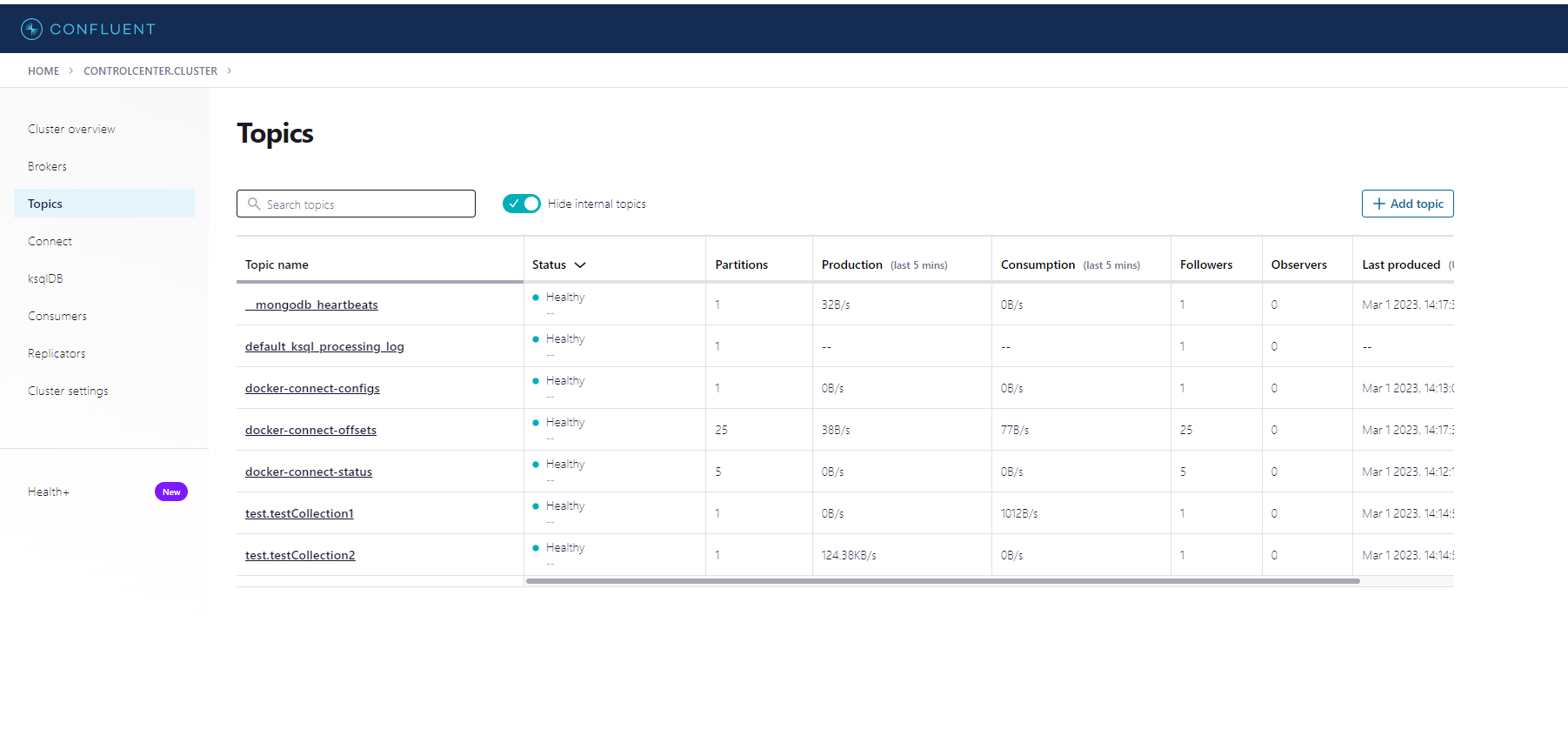
**Topics created by Kafka Connect inside Kafka Cluster:**

Below topics will automatically be created on Kafka cluster(their names are either defaults or passed in environment variables as per docker-compose file:

* \_mongodb\_heartbeats -> to keep the offsets refreshed during heartbeats
* docker-connect-configs\_> to store connector and task configuration data. This must be the same for all workers with the same group.id(which form the cluster of worker nodes)
* docker-connect-offsets -> to store offset data for connectors. This must be the same for all workers with the same group.id(which form the cluster of worker nodes)
* docker-connect-status -> to store state for connectors. This must be the same for all workers with the same group.id(which form the cluster of worker nodes)

Topics that hold the data retrieved from MongoDB are created:

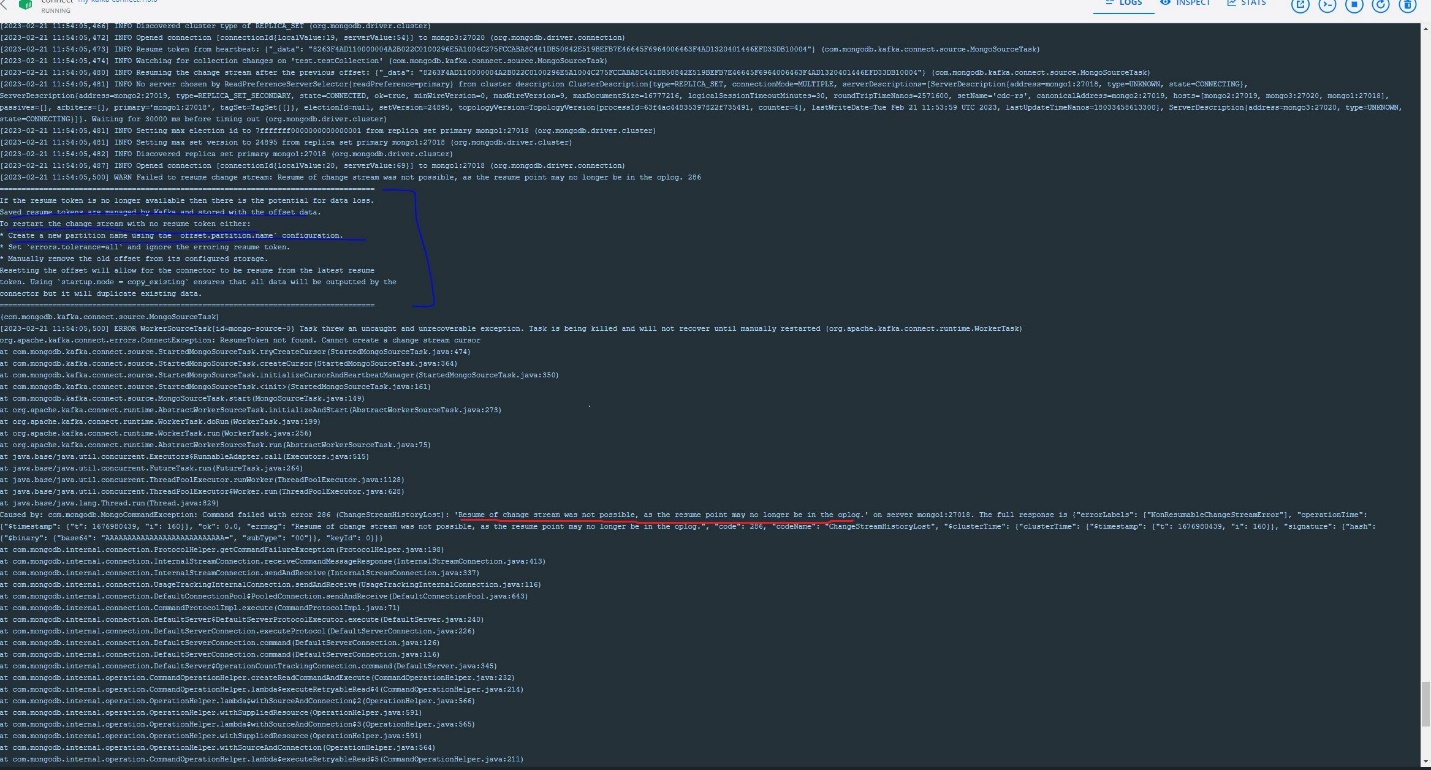
* test.testCollection1
* test.testCollection2



**Problematic Scenarios:**

1. When oplog reaches it’s max configuration(1G in our case), new messages are written over the older messages.

In case the connector was paused/stopped, when it will be restarted, the resumeToken(offset) will not be available anymore in MongoDB and an error will be displayed:

****

In order to resume from this error, offset.partition.name must be filled as per <https://www.mongodb.com/docs/kafka-connector/current/troubleshooting/recover-from-invalid-resume-token/#std-label-invalid-resume-token-prevention>

in order to restart the stream. The stream will be restarted with no resumeToken, data previously written to the oplog will be lost!

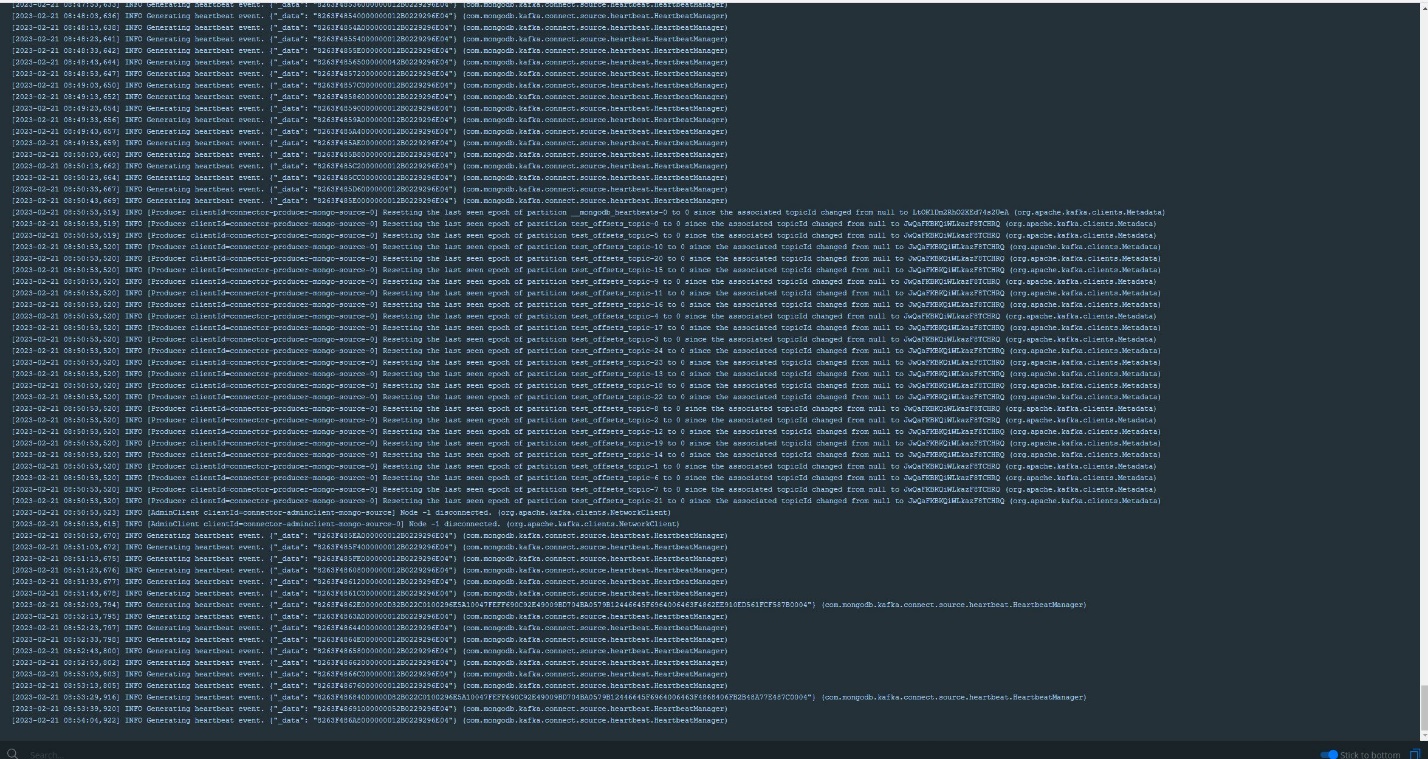
Solution:

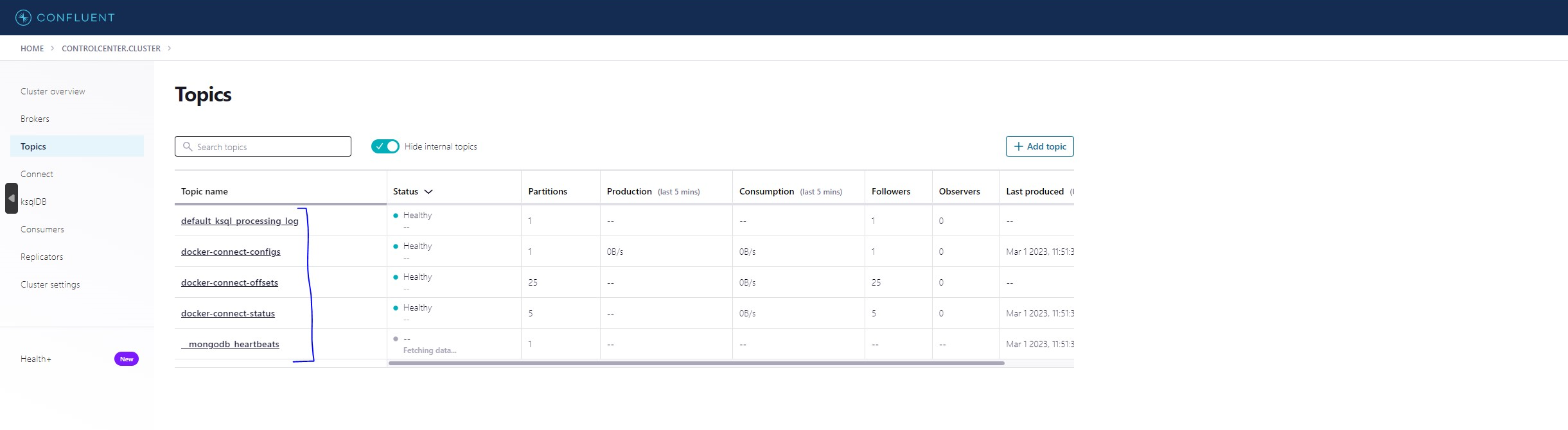
* make the size of oplog big enough to hold changes for a longer period in which the connector is down
* monitor the connector metrics to make sure alerts are raised when connector is down(please see connector’s metrics section)

1. frequent namespace changes can cause resumeTokens to be invalid as per <https://www.mongodb.com/docs/kafka-connector/current/troubleshooting/recover-from-invalid-resume-token/#std-label-invalid-resume-token-prevention>

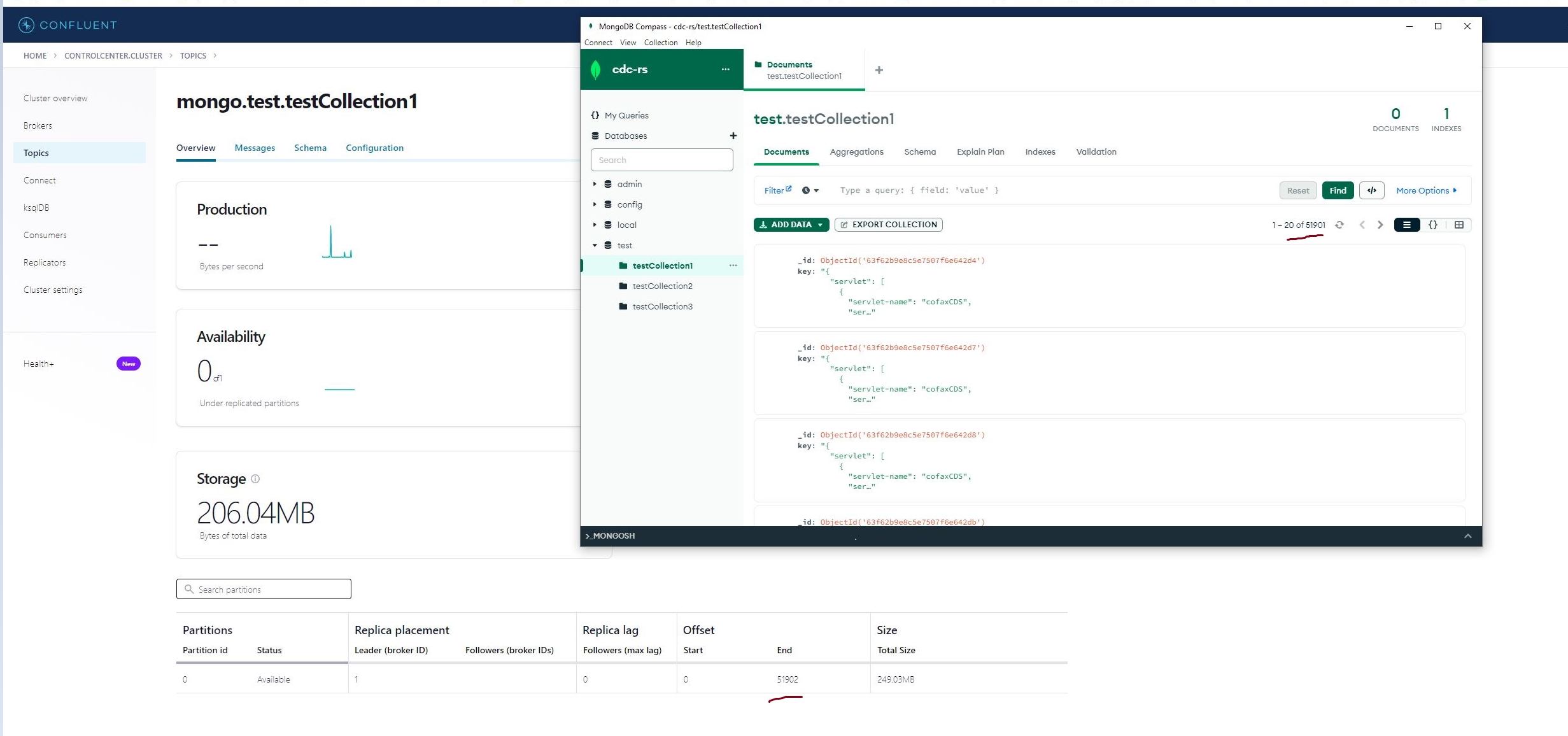
Solution:

* set **heartbeat.interval.ms** to refresh the resumeToken. This will create a Kafka topic as below, where herbeat offsets are stored and activity is resumed safely

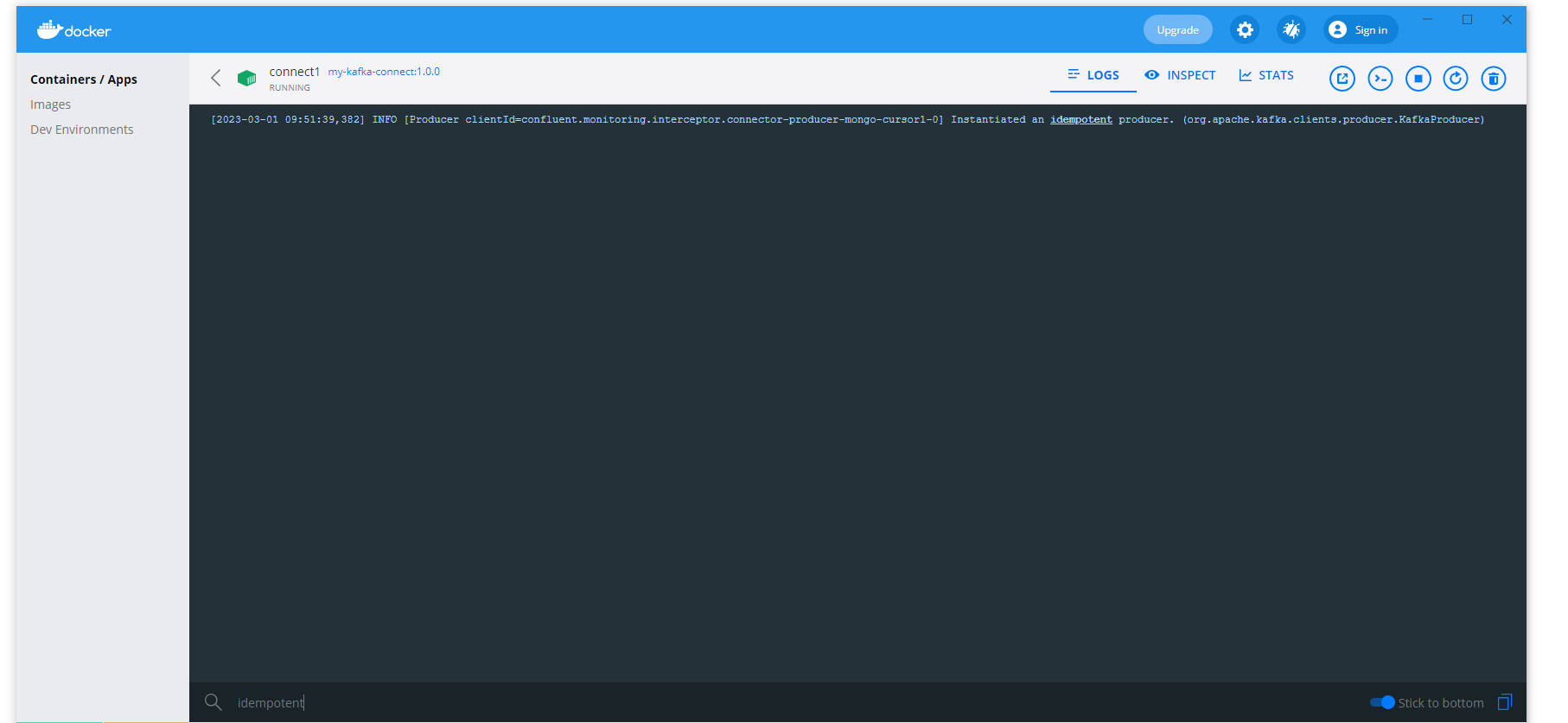




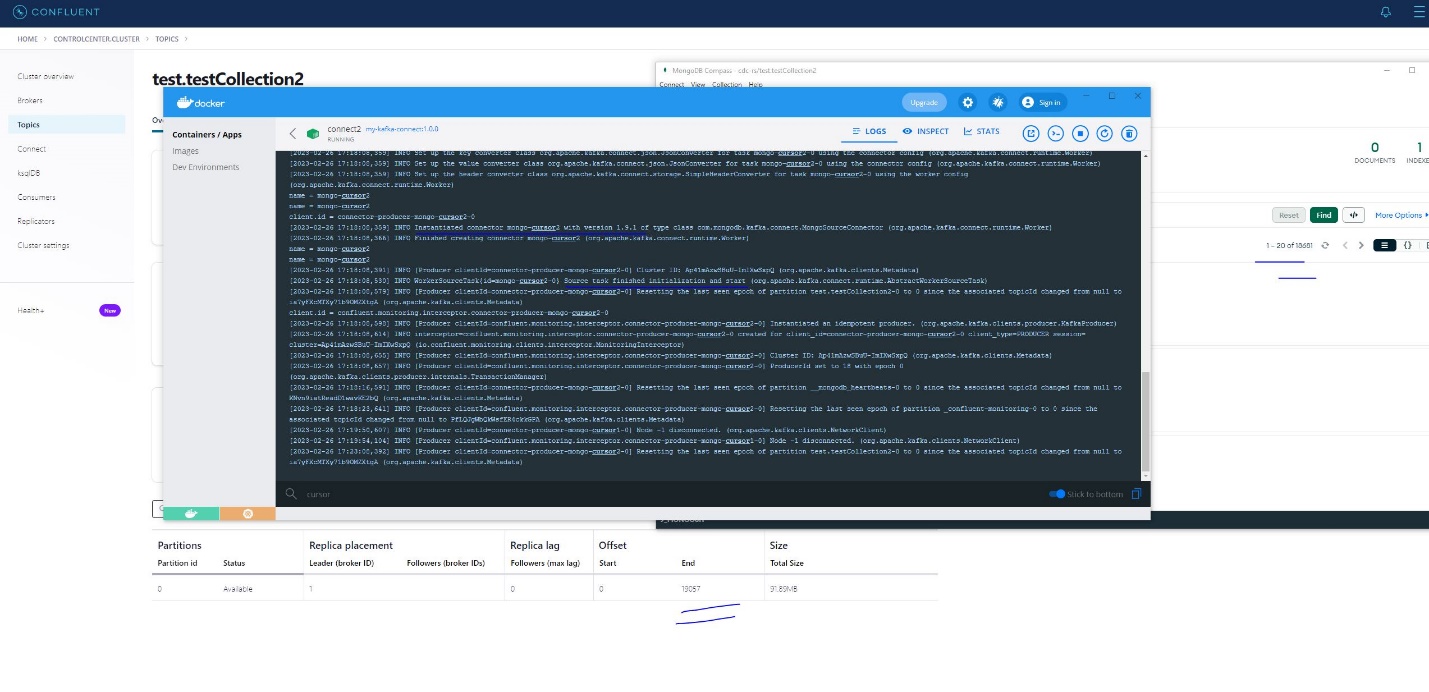
1. When Kafka Broker goes down and the connector succeeds to write the data but not the offset, the process will be resumed with an older offset. Hence, same data from source will be sent twice



The producer is idempotent by default



4) When worker node goes down and did not manage to write offsets but managed to write data, the other worker that will take over the workload will start with an older offset and hence same data will be collected again from MongoDb and duplicate data will be written to Kafka topic

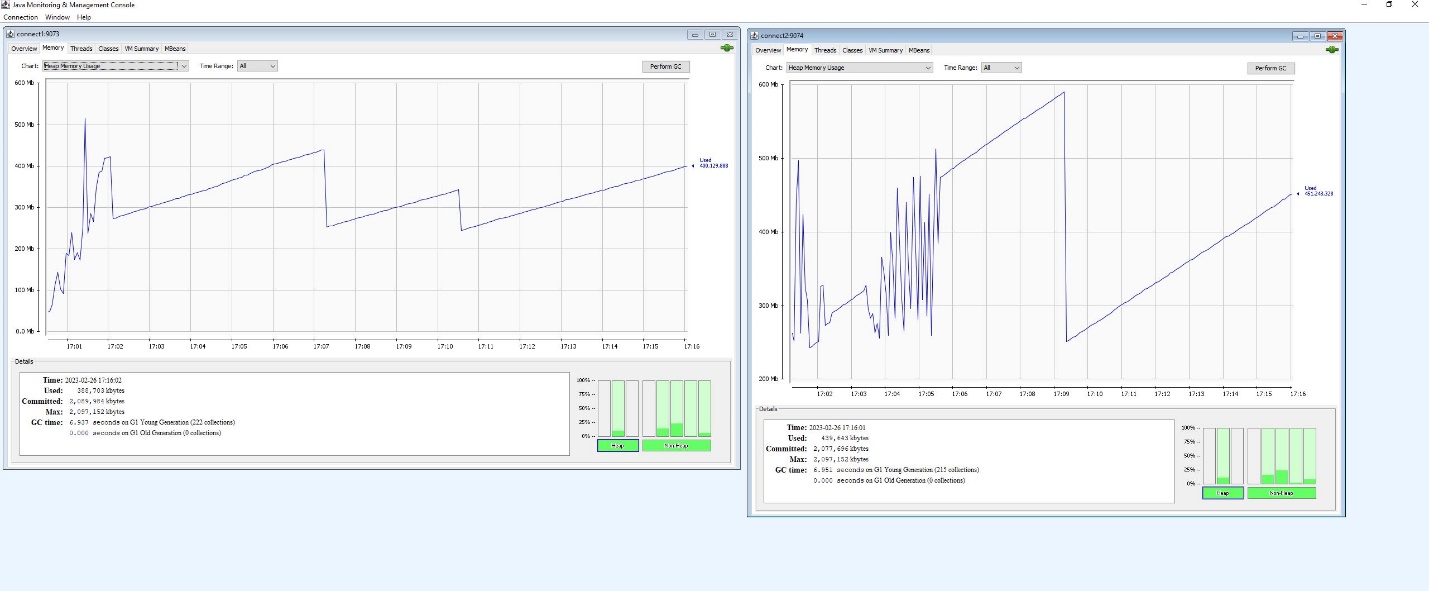


**Normal behavior of the connector in cluster mode**

The worker nodes inside Kafka Connect cluster communicate and split work between each other. In order to be able to form a cluster, the workers must belong to the same group.id, that can be passed as environment variable as per docker-compose.yml file.

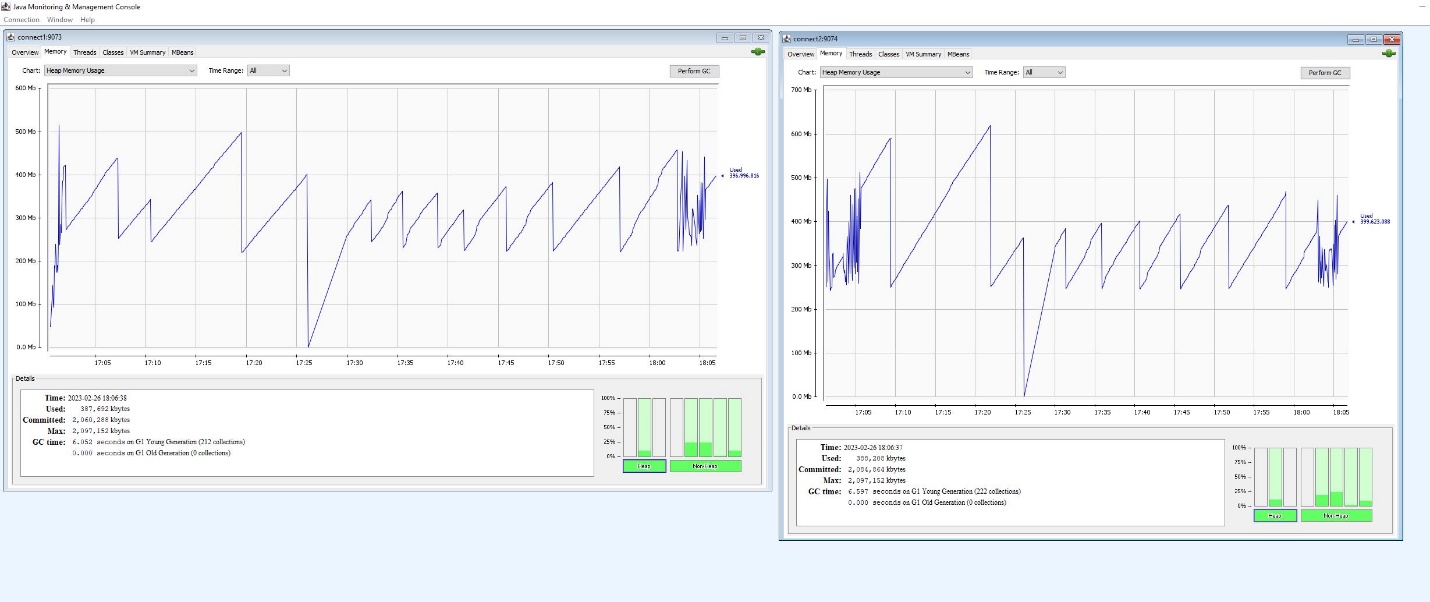
1. If we have two worker nodes and only one cursor(one ChangeStream), only one worker will work, the other will be a standby in case it fails.

As per screenshot below, only worker node **connect2:9074** works between 17:04 and 17:06 when the load was generated.

****

1. In case we have 2 cursors open, the work will be split between the 2 worker nodes

In screenshot below both worker nodes **connect1:9073** and **connect2:9074** work independently, each on one cursor, between 18:00 and 18:05, when the workload was generated.

****

**Monitor Kafka Connect:**

In order to monitor Kafka Connect cluster, JMX options must be activated as per docker-compose.yaml file. In tab MBeans from JConsole there are metrics that belong to each worker node

