**Ref:** [**https://medium.com/larus-team/how-to-setup-mirrormaker-2-0-on-apache-kafka-multi-cluster-environment-87712d7997a4**](https://medium.com/larus-team/how-to-setup-mirrormaker-2-0-on-apache-kafka-multi-cluster-environment-87712d7997a4)

**How to setup MirrorMaker 2.0 on Apache Kafka multi-cluster environment**

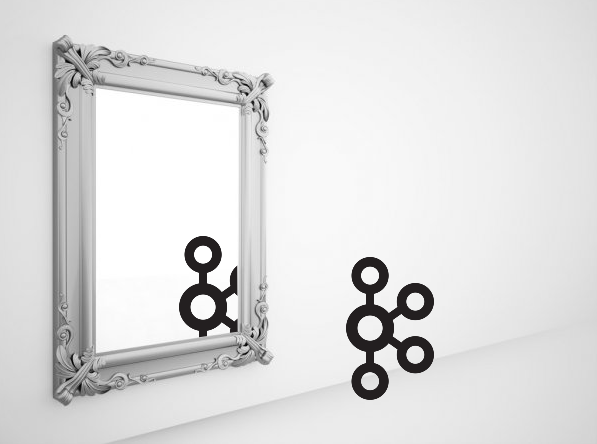
[Mauro Roiter](https://medium.com/@mauro.roiter?source=post_page---byline--87712d7997a4---------------------------------------)

7 min read

·

May 7, 2021

Disaster Recovery strategies, and more generally business continuity plans, are one of the most critical points to be implemented in order to minimize data-centers downtime and data loss.



Apache Kafka provides a solution called MirrorMaker which gives the ability to mirror data from a source Kafka cluster to a destination Kafka cluster. Despite this, MirrorMaker has some weaknesses which do not make it the ideal solution for disaster recovery cases.

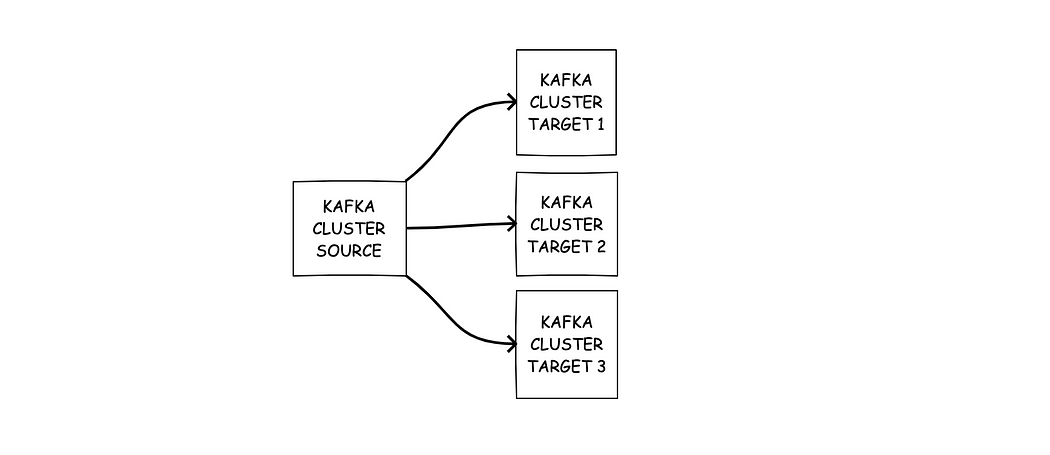
MirrorMaker 2.0 (MM2), based on the Kafka Connect framework, is the new open-source solution able to manage multi-cluster environments and cross-data-center replication. With the [KIP-382](https://cwiki.apache.org/confluence/display/KAFKA/KIP-382%3A+MirrorMaker+2.0) article, MM2 is accepted as the official solution by Apache Kafka.

Basically, MM2 introduces the following features:

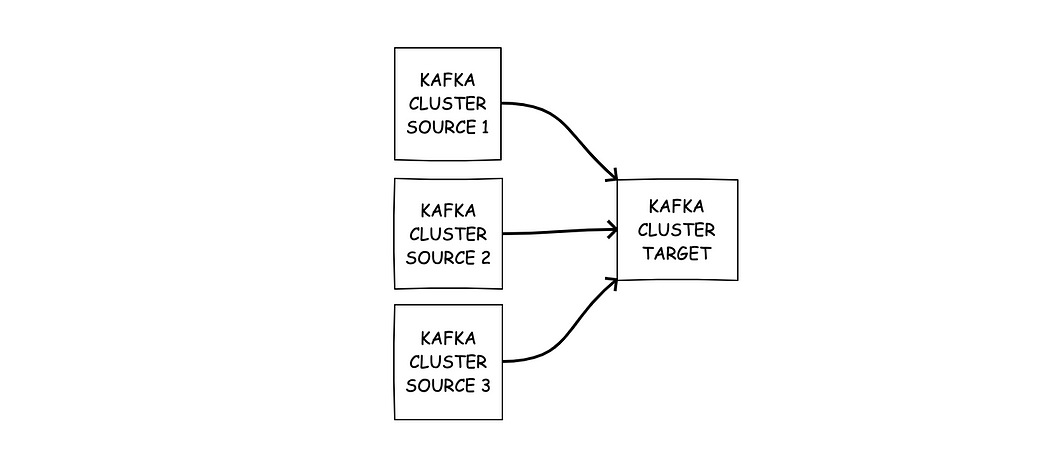
* Offset translation
* Timestamp-based recovery (consumer group checkpoint)
* Cross-cluster metrics to monitor
* Single place (a properties file) where to configure the entire cross-cluster replication
* Topics configuration, partitions and ACLs synced

MM2 can replicate data in directional flows expressed with the notation **source->target**, for example:

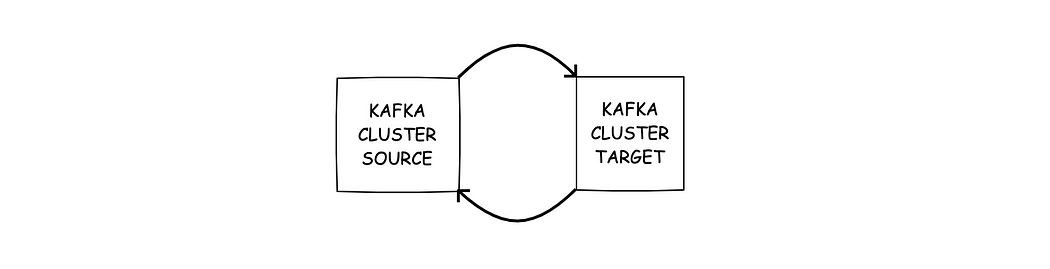
* **fan-out**, where you have a single source cluster and multiple target clusters



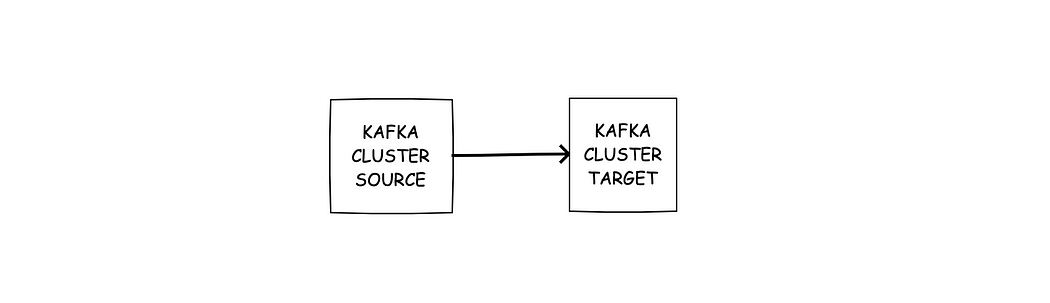
* **aggregation**, where you have multiple source clusters and a single target cluster



* **active/active**, where you have a bidirectional flow between a source cluster and a target cluster



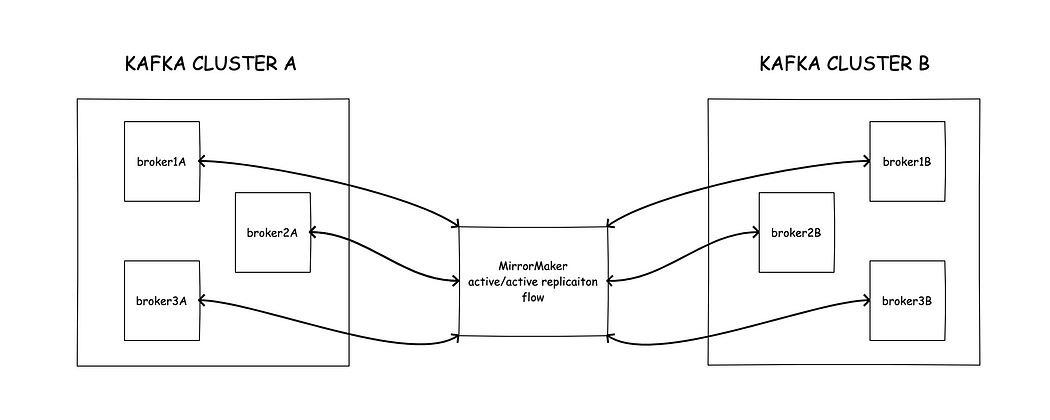
* **active/passive**, where you have a unidirectional flow from a source cluster to a target cluster



A typical use-case of MM2 is a multi-cluster environment where clusters could be in the same data center or across multiple data centers.

**Hands on**

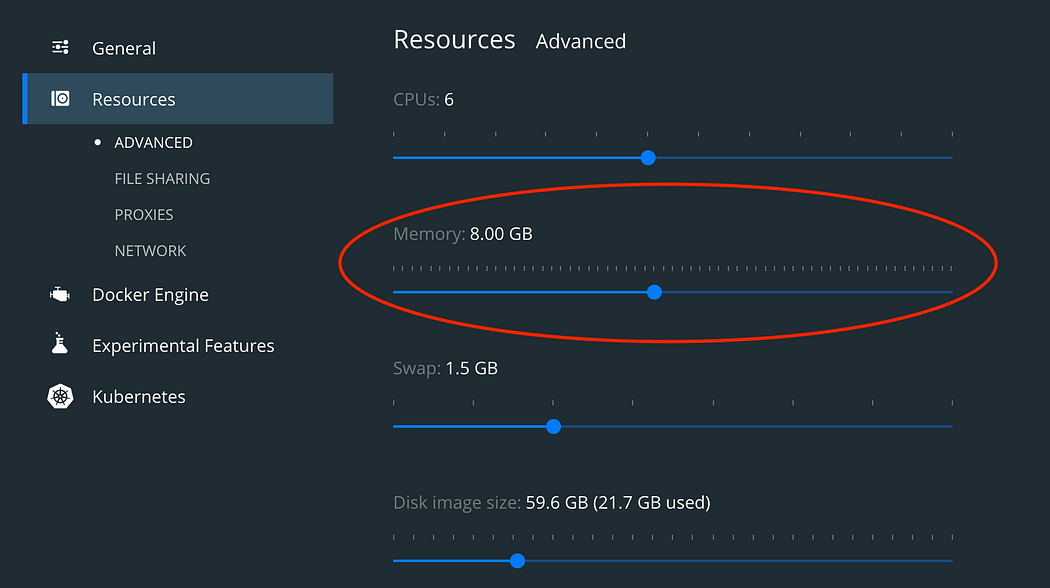
Let’s explore the features of this powerful tool with a practical example, that can be found [here](https://github.com/mroiter-larus/kafka-docker-mm2), about how to setup MM2 for an active/active replication flow in a local Dockerized environment.



With the provided docker-compose file we are going to simulate two Kafka clusters, named “*clusterA*” and “*clusterB*” (as specified in *mm2.properties* file), and we are going to use MM2 in order to replicate data across them in an “active/active” scenario. The *mm2.properties* file to configure MirrorMaker is the following:

**Step By Step**

Before we start, make sure that your Docker environment has at least 8 GB of allocated memory in order to avoid out-of-memory errors.



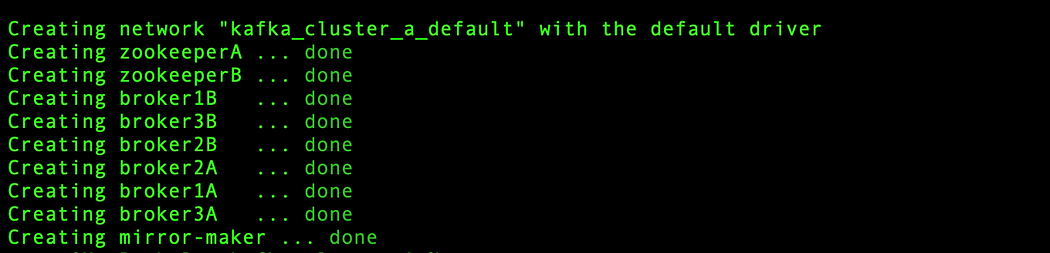
1. Clone the GitHub repository

git clone <https://github.com/mroiter-larus/kafka-docker-mm2>

2. Open a Terminal window and go to the root of the repository, then start the test environment by running:

docker-compose up -d

At the end of the process you should see something like:



3. Connect to Kafka container dedicated to MirrorMaker:

docker exec -it mirror-maker /bin/bash

4. From the inside of the container, start MirrorMaker:

connect-mirror-maker /tmp/kafka/config/mm2.properties

If you get a *FileNotFoundException*, like

java.io.FileNotFoundException: /usr/bin/../config/connect-log4j.properties (No such file or directory)

don’t panic! If you want to remove that exception you just need to define the connect-log4j.properties file position in an environment variable as follow

export KAFKA\_LOG4J\_OPTS="-Dlog4j.configuration=file:/etc/kafka/connect-log4j.properties"

and rerun the step-4 command. However, this is not mandatory, but it is helpful if you would like to explore MirrorMaker logs.

5. Open a new Terminal window and connect to a broker of the source cluster:

docker exec -it broker1A /bin/bash

6. Let’s create a topic “*topic1*” into both clusters:

kafka-topics --bootstrap-server broker1A:29092 --create --topic topic1 --partitions 3 --replication-factor 3

7. Open a new Terminal window, then connect to one of the broker in the target cluster

docker exec -it broker1B /bin/bash

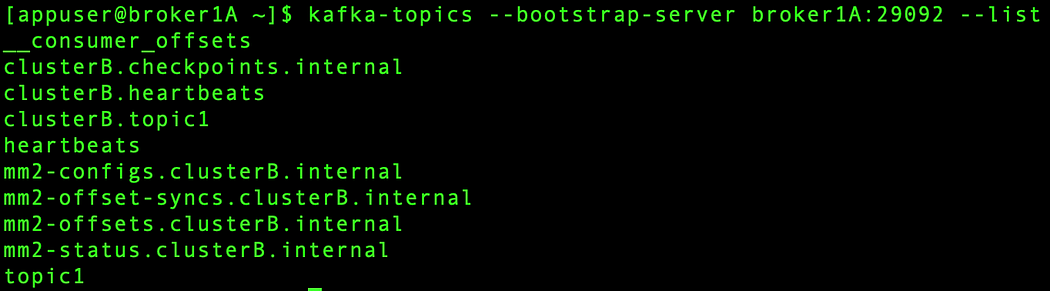
create a topic “*topic1*”:

kafka-topics --bootstrap-server broker1B:29093 --create --topic topic1 --partitions 3 --replication-factor 3

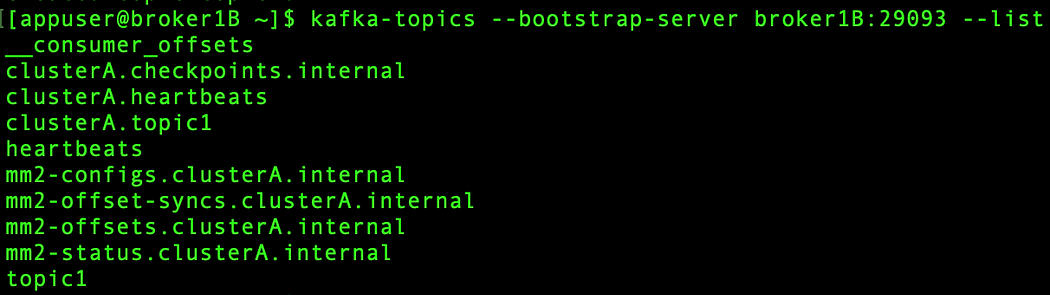
and checkout the topics list into both clusters by running

kafka-topics --bootstrap-server broker1A:29092 --list  
kafka-topics --bootstrap-server broker1B:29093 --list

From *clusterA* you will get:



From *clusterB* you will get:



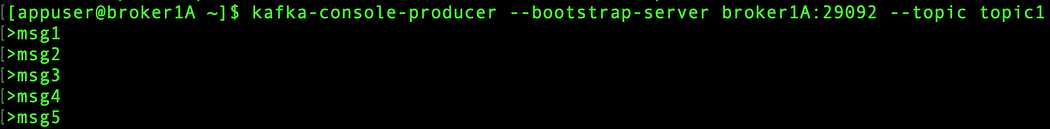
As you can see, the *topic1* created in the *clusterA* has been replicated on the *clusterB* with the addition of the prefix “*clusterA.*”, and the *topic1* created in the *clusterB* has been replicated on the *clusterA* with the addition of the prefix “*clusterB.*”, so that you can easily identify every topic avoiding any conflicts.

The reason why two *topic1* were created is because we want to leverage MirrorMaker features in order to have a disaster recovery plan. What could happen in a production scenario is that one of the two clusters goes down for some reasons. Without MirrorMaker, this could mean data loss and significant downtime. With MirrorMaker, we just need to send the same events on both *topic1* and MM2 will take care of keeping the two clusters synchronized. Doing so, if a cluster goes down, all the data are still available on the other one.

Let’s simulate the above scenario:

8. From the *broker1A* Docker container, to which we connected at step 5, run a console-producer and send some sample events to *topic1*:

kafka-console-producer --bootstrap-server broker1A:29092 --topic topic1

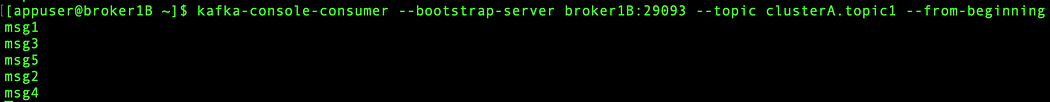


9. From the *broker1B* Docker container, to which we connected at step 8, do the same thing:

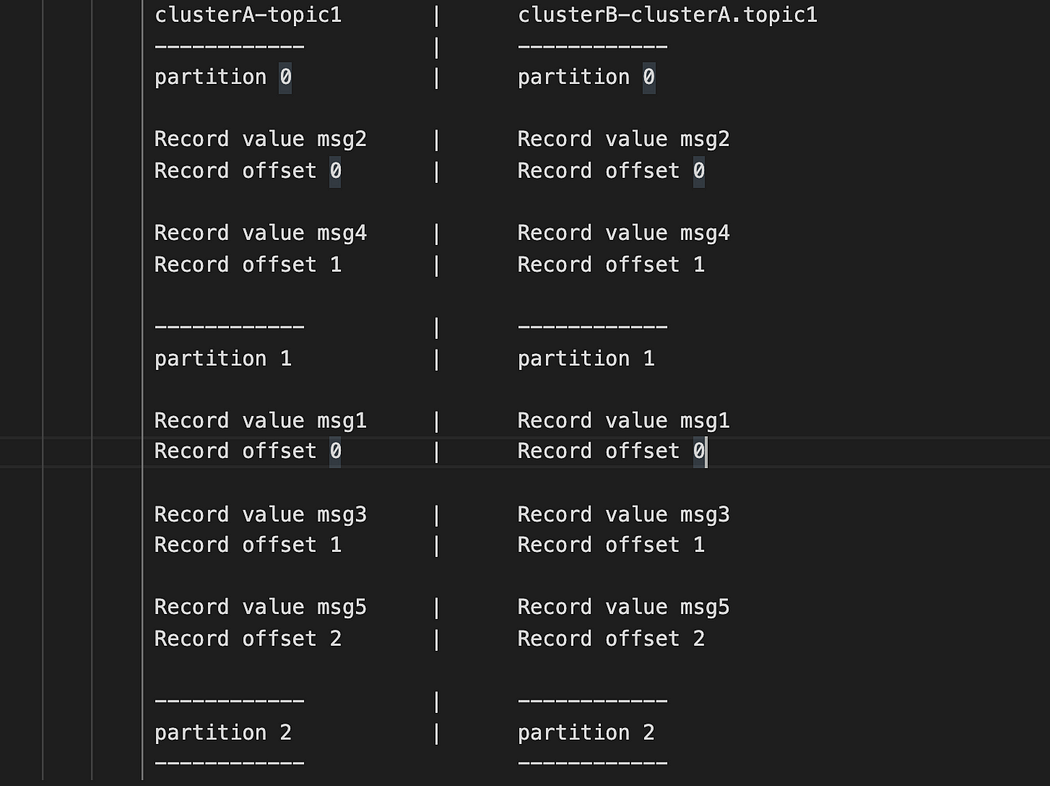
kafka-console-producer --bootstrap-server broker1B:29093 --topic topic1

10. From the *broker1B* Docker container, check if the data sent into *topic1* (inside clusterA) has been replicated as expected:

kafka-console-consumer --bootstrap-server broker1B:29093 --topic clusterA.topic1 --from-beginning



As you can see all the events have been automatically replicated. Don’t worry about the output order of the events, because MirrorMaker replicates also topic configurations, including checkpoints and offsets. This means that every message has his corresponding offset, so that the event ordering will be guaranteed. If you want to make sure about this, just run the simple Java application I’ve provided in the GitHub repository linked above and checkout the offsets for every *topic1* and *clusterA.topic1* partition. Just be sure to change the bootstrap url, the topic name and the partition accordingly. Here is a summary about the result:



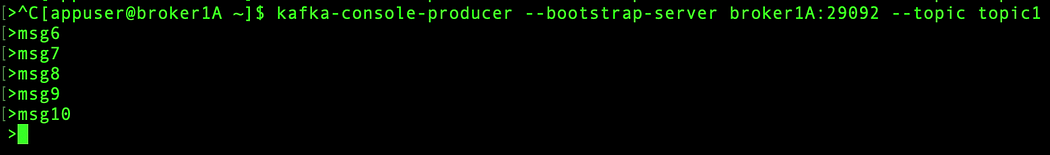
11. Imagine one of the two cluster goes down accidentally. We can simulate this scenario by stopping the containers of the *clusterB,* for example:

docker-compose stop broker1B broker2B broker3B

Our hypothetical application will still be available for both reading and writing since we can still produce and consume events into *topic1* (inside *clusterA*). After the downtime period, even if *topic1* (inside clusterB) will have remained misaligned with respect to *topic1* (inside clusterA), it can be realigned starting from *clusterA.topic1* which will be automatically replicated as soon as the *clusterB* will come back online.

12. Send some other events into *topic1*

kafka-console-producer --bootstrap-server broker1A:29092 --topic topic1



13. Restart the *clusterB*

docker-compose start broker1B broker2B broker3B

Once the cluster is up and running, MM2 automatically resync topics and all its configurations. Let’s verify if the message sent during the downtime period has been replicated:

kafka-console-consumer --bootstrap-server broker1B:29093 --topic clusterA.topic1 --from-beginning

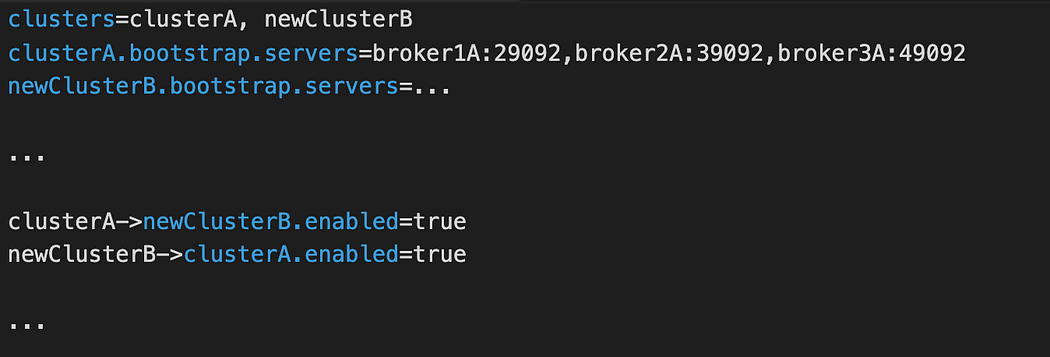


If you don’t see the replicated events immediately, just wait until MirrorMaker resync data between clusters (it may take few minutes). If you want to immediately replicate the data, you just need to restart MirrorMaker (re-running step 4) and the game is done.

**Worst case**

The worst scenario could be that the cluster is permanently lost. In this case, we can recover the lost *clusterB* from the *clusterA:*

* setting up a *newClusterB*
* editing *mm2.properties*, replacing all the configurations that referenced the old cluster with the new values



* and finally restarting the MirrorMaker

**Conclusion**

That’s it!! We’ve seen how to setup MirrorMaker 2.0 in a dedicated instance. This running mode does not need a running Connect cluster: it leverages a high-level driver which generates a set of Connect workers based on the *mm2.properties* configuration file.

If you run into any issues, [please raise a GitHub issue](https://github.com/mroiter-larus/kafka-docker-mm2/issues).

[Apache](https://medium.com/tag/apache?source=post_page-----87712d7997a4---------------------------------------)

[Apache Kafka](https://medium.com/tag/apache-kafka?source=post_page-----87712d7997a4---------------------------------------)

[Setup](https://medium.com/tag/setup?source=post_page-----87712d7997a4---------------------------------------)

[Mirrormaker](https://medium.com/tag/mirrormaker?source=post_page-----87712d7997a4---------------------------------------)

[[](https://medium.com/larus-team?source=post_page---post_publication_info--87712d7997a4---------------------------------------)](https://medium.com/larus-team?source=post_page---post_publication_info--87712d7997a4---------------------------------------)

**[Published in LARUS](https://medium.com/larus-team?source=post_page---post_publication_info--87712d7997a4---------------------------------------)**

[47 followers](https://medium.com/larus-team/followers?source=post_page---post_publication_info--87712d7997a4---------------------------------------)

·[Last published Nov 29, 2022](https://medium.com/larus-team/the-most-loved-in-memory-store-an-introduction-to-redis-6852834bdd5a?source=post_page---post_publication_info--87712d7997a4---------------------------------------)

We are data engineers and data scientists team passionate about data and top quality data insight-driven software solutions. We share contents around NoSQL, Graph Technologies and stream processing platform

[[](https://medium.com/@mauro.roiter?source=post_page---post_author_info--87712d7997a4---------------------------------------)](https://medium.com/@mauro.roiter?source=post_page---post_author_info--87712d7997a4---------------------------------------)

**[Written by Mauro Roiter](https://medium.com/@mauro.roiter?source=post_page---post_author_info--87712d7997a4---------------------------------------)**

[17 followers](https://medium.com/@mauro.roiter/followers?source=post_page---post_author_info--87712d7997a4---------------------------------------)

·[18 following](https://medium.com/@mauro.roiter/following?source=post_page---post_author_info--87712d7997a4---------------------------------------)

Senior Software Developer @AgileLARUS

[What are your thoughts?](https://medium.com/m/signin?operation=register&redirect=https%3A%2F%2Fmedium.com%2Flarus-team%2Fhow-to-setup-mirrormaker-2-0-on-apache-kafka-multi-cluster-environment-87712d7997a4&source=---post_responses--87712d7997a4---------------------respond_sidebar------------------)

[chandrashekhar c pawali](https://medium.com/@cpawali?source=post_page---post_responses--87712d7997a4----0-----------------------------------)

[Oct 31, 2023](https://medium.com/@cpawali/nice-article-i-was-trying-to-add-mm2-along-with-jmx-configuration-to-monitor-kafka-mm2-b72a468375b6?source=post_page---post_responses--87712d7997a4----0-----------------------------------)

Nice article!! i was trying to add MM2 along with JMX configuration to monitor Kafka MM2,

i used your repo and updated docker-compose.yml to include JMX config and it worked.

Thanks a lot

here are changes which i made for my requirement

mirror-maker:

[Vishnu](https://medium.com/@vishnu24297069?source=post_page---post_responses--87712d7997a4----1-----------------------------------)

Hii Mauro,

This blog final taught me how to setup mirror maker for kafka recovery. And i followed each steps as mentioned and it was working awesomely. But i had one doubt regarding when i was adding Security to my kafka cluster using SASL and ACL it…

[Rahulbhatia1998](https://medium.com/@rahulbhatia1998?source=post_page---post_responses--87712d7997a4----2-----------------------------------)