Replicator Overview

Replicator

Confluent Replicator allows you to easily and reliably replicate topics from one Kafka cluster to another. In addition to copying the messages, Replicator will create topics as needed preserving the topic configuration in the source cluster. This includes preserving the number of partitions, the replication factor, and any configuration overrides specified for individual topics. Replicator is implemented as a connector.

Features

Replicator supports the following features:

- Topic selection using whitelists, blacklists, and regular expressions.
- Dynamic topic creation in the destination cluster with matching partition counts, replication factors, and topic configuration overrides.
- Automatic resizing of topics when new partitions are added in the source cluster.
- Automatic reconfiguration of topics when topic configuration changes in the source cluster.
- Timestamp Preservation, Using Provenance Headers to Prevent Duplicates or Cyclic Message Repetition, and Consumer Offset Translation (supported on Confluent Platform 5.0.1 and later).
- You can migrate from MirrorMaker to Replicator on existing datacenters (Confluent Platform 5.0.0 and later). Migration from MirrorMaker to Replicator is not supported in earlier versions of Confluent Platform (pre 5.5.0).
- At least once delivery, meaning the Replicator connector guarantees that records are delivered at least once to the Kafka topic. If the connector restarts, there may be some duplicate records in the Kafka topic.

Multi-Datacenter Use Cases

Replicator can be deployed across clusters and in multiple datacenters. Multi-datacenter deployments enable use-cases such as:

- Active-active geo-localized deployments: allows users to access a near-by datacenter to optimize their architecture for low latency and high performance
- Active-passive disaster recover (DR) deployments: in an event of a partial or complete datacenter disaster, allow failing over applications to use Confluent Platform in a different datacenter.

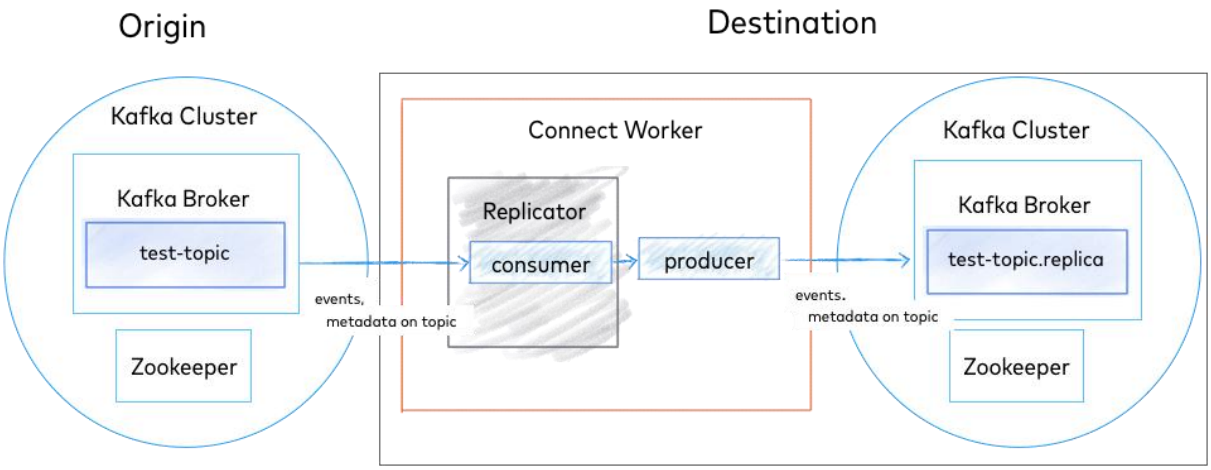
- Centralized analytics: Aggregate data from multiple Apache Kafka® clusters into one location for organization-wide analytics
- Cloud migration: Use Kafka to synchronize data between on-prem applications and cloud deployments

Replication of events in Kafka topics from one cluster to another is the foundation of Confluent's multi datacenter architecture.

Replication can be done with Confluent Replicator or using the open source Kafka MirrorMaker. Replicator can be used for replication of topic data as well as migrating schemas in Schema Registry.

Architecture

The diagram below shows the Replicator architecture. Replicator uses the Kafka Connect APIs and Workers to provide high availability, load-balancing and centralized management.

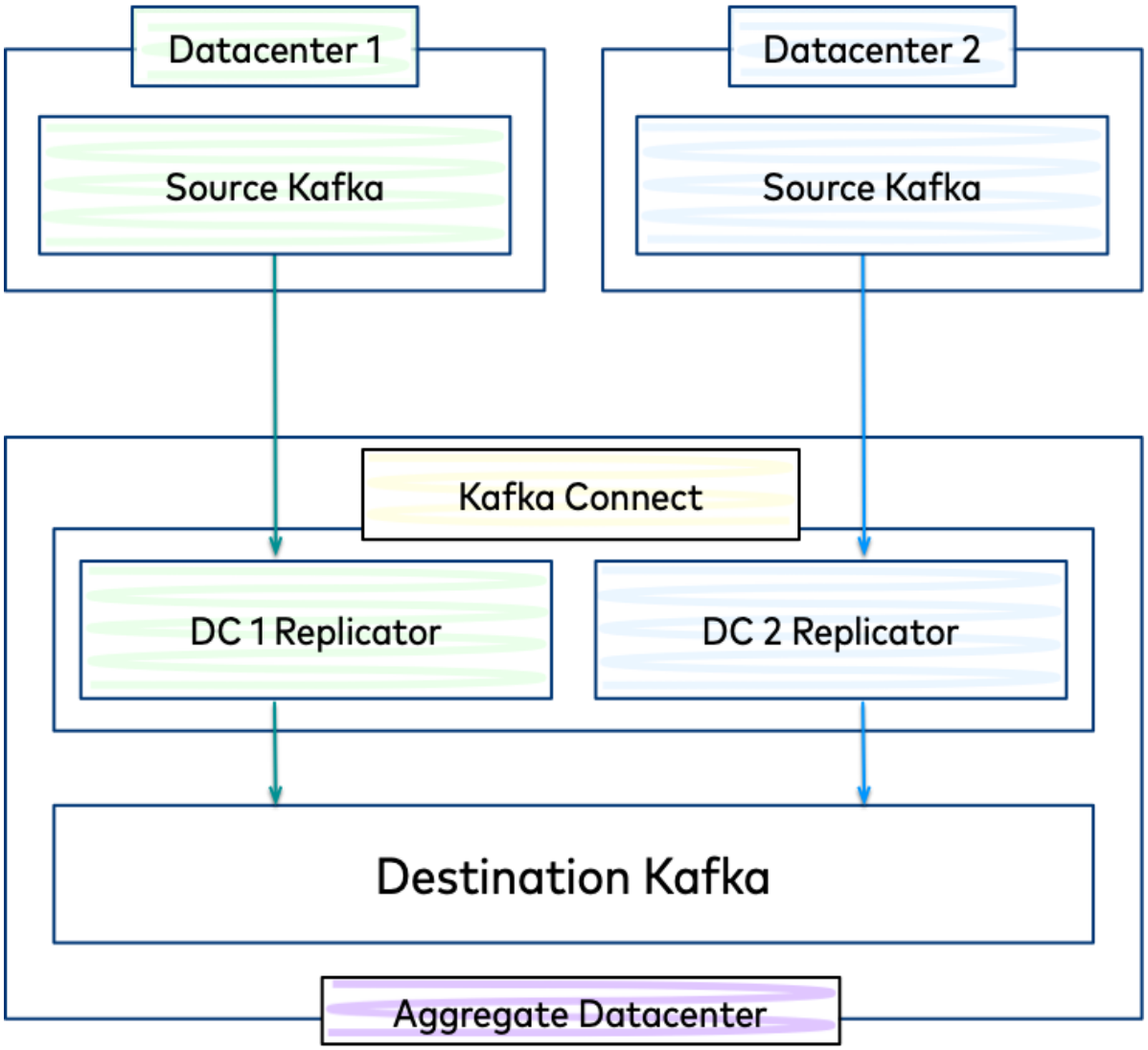


Replicator Architecture

Example Deployment

In a typical multi-datacenter deployment, data from two geographically distributed Kafka clusters located in separate datacenters is aggregated in a separate cluster located in another datacenter. The origin of the copied data is referred to as the "source" cluster while the target of the copied data is referred to as the "destination."

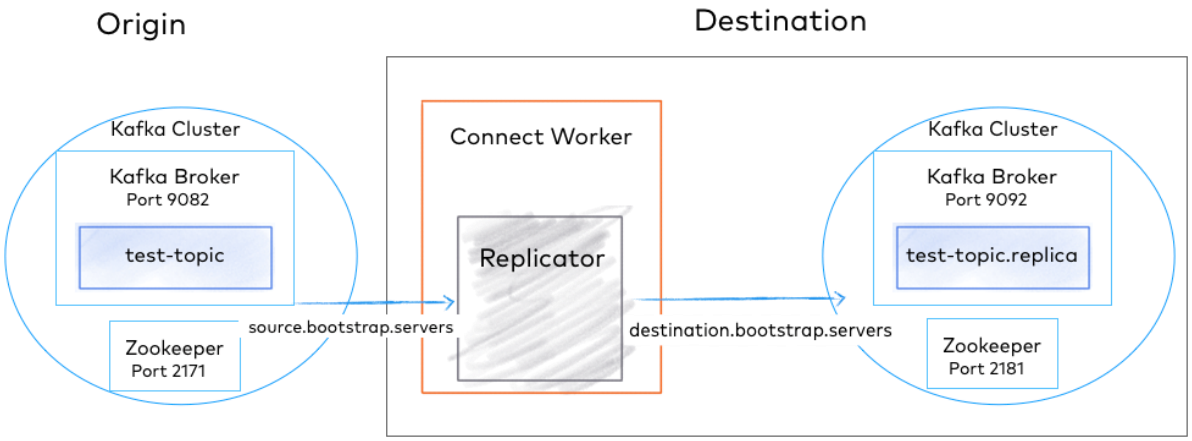
Each source cluster requires a separate instance of Replicator. For convenience you can run them in the same Connect cluster, located in the aggregate datacenter.



Replication to an Aggregate Cluster

Tutorial: Replicating Data Across Clusters

This guide describes how to start two Apache Kafka® clusters and then a Replicator process to replicate data between them. Note that for tutorial purposes, we are running both clusters on the same machine. In order to do that, we jump through a hoop or two to make sure each cluster has its own ports and data directories. You will not need to perform these changes on the ZooKeeper and Broker configuration if you are running in a normal environment where each cluster has its own servers.



Replicator Quick Start Configuration

Tip

About install prerequisites and command examples

- These instructions assume you have a local installation of Confluent Platform, the Confluent CLI, and Java 1.8 or 1.11 (needed for Confluent Platform). If you are new to Confluent Platform, work through the Quick Start for Confluent Platform (Local install) first, and then return to this tutorial.
- The commands shown to start servers and Replicator assume you are running from your Confluent Platform home directory or have Confluent Platform in your CLASSPATH. The examples also assume that your properties files are in the default locations on your Confluent Platform installation, except as otherwise noted. This should make it easier to copy/paste example commands directly into your terminal in most cases. For example, this command:

```
./bin/zookeeper-server-start etc/kafka/zookeeper.properties
```

  Is meant to be the same as:

```
$CONFLUENT_HOME/bin/zookeeper-server-start $CONFLUENT_HOME/etc/kafka/zookeeper
.properties
```

Ports for Kafka brokers and Confluent Platform components

The examples in this tutorial define the following port configurations.

	Origin	Destination
Kafka Brokers	9082	9092
ZooKeeper	2171	2181
Connect Replicator worker		8083 (copies topics from origin -> destination)
Control Center		9021

Start the destination cluster

1. First, start a ZooKeeper server. In this guide, we are assuming services will run on `localhost`.
   - Start ZooKeeper by running this command in its own terminal.

     ```
     ./bin/zookeeper-server-start etc/kafka/zookeeper.properties
     ```

2. Next, start a Kafka broker that will serve as our single node Kafka cluster.
   - Start Kafka by running this command in its own terminal.

     ```
     ./bin/kafka-server-start etc/kafka/server.properties
     ```

Note

The destination cluster should be running the same (or higher) version of Confluent Platform as the source cluster. Replicator runs within a Connect cluster linked to the destination cluster, and reads messages from the source cluster. Therefore, Replicator will not be able to interpret the message format if the destination is running an older version.

Start the origin cluster

While you configured the destination cluster to run on default ports, you will need to run the origin cluster on a different port to avoid collisions. The Kafka broker in the origin cluster is configured on port 9082, ZooKeeper is configured on 2171, as shown in the ports mapping. Copy the configuration files to a temporary location and modify them so they do not conflict with the destination cluster.

1. Create a “quickstart” directory under `$CONFLUENT_HOME` for the tutorial properties files; for example, `my-examples/`.

Tip

- `$CONFLUENT_HOME` represents `<path-to-confluent>`. On Linux systems, you can use this notation to set a shell environment variable.
- The rest of the examples in this section assume you are running commands from `$CONFLUENT_HOME` and using `my-examples/` to store your properties files. If you follow this model, you can use the copy-paste options on the code blocks.

2. Copy the config files to `my-examples` (or a directory of your choice).

```
cp etc/kafka/zookeeper.properties my-examples/zookeeper_origin.properties

cp etc/kafka/server.properties my-examples/server_origin.properties
```

3. Update the port numbers.

```
sed -i '' -e "s/2181/2171/g" my-examples/zookeeper_origin.properties
sed -i '' -e "s/9092/9082/g" my-examples/server_origin.properties
sed -i '' -e "s/2181/2171/g" my-examples/server_origin.properties
sed -i '' -e "s/#listeners/listeners/g" my-examples/server_origin.properties
sed -i '' -e "s/8090/8091/g" my-examples/server_origin.properties
sed -i '' -e "s/#confluent.metadata.server.listeners/confluent.metadata.server
.listeners/g" my-examples/server_origin.properties
```

4. Update the broker ID for origin.

```
sed -i '' -e "s/broker.id=0/broker.id=1/g" my-examples/server_origin.propertie
s
```

Tip

This is not entirely necessary, as the brokers are in two different clusters, but it is nice to have unique broker IDs if you want to manage them with Kafka from the command line.

5. Update data directories.

```
sed -i '' -e "s/zookeeper/zookeeper_origin/g" my-examples/zookeeper_origin.pro
perties
sed -i '' -e "s/kafka-logs/kafka-logs-origin/g" my-examples/server_origin.prop
erties
```

6. From here, you can start up the origin cluster.
   - Start ZooKeeper by running this command in its own terminal.

     ```
     ./bin/zookeeper-server-start my-examples/zookeeper_origin.properties
     ```

   - Start Kafka by running this command in its own terminal.

     ```
     ./bin/kafka-server-start my-examples/server_origin.properties
     ```

Create a topic

Bring up a new command window to run Kafka commands.

Create a topic named “test-topic” in the origin cluster with the following command:

```
kafka-topics --create --topic test-topic --replication-factor 1 --partitions 1 --b
ootstrap-server localhost:9082
```

Tip

Depending on your environment, you may have to use the `.sh` extension (for example, `kafka-topics.sh`) for Kafka commands.
You can verify that the topic was created as follows:

```
kafka-topics --list --bootstrap-server localhost:9082
```

Your output should look similar to this (the `_confluent` topics are internal topics):

```
__confluent.support.metrics
_confluent-command
test-topic
```

When we configure and run Replicator, this “test-topic” will get replicated to the destination cluster (on port `2181`) with the exact configuration we defined above. For the sake of this example, the test topic was created with just one partition. Replicator will work with any number of topics and partitions.

Configure and run Replicator

Confluent Replicator can run as an executable or as a Connector in the Kafka Connect framework. For this quick start, start Replicator as an executable.

Create consumer, producer, and replicator configuration files

The Replicator executable script expects three configuration files:

- Configuration for the origin cluster
- Configuration for the destination cluster
- Replicator configuration

Tip

You can place these config files in any directory you choose, but these steps assume a path of `$CONFLUENT_HOME/my-examples/`, so as not to conflict with the default properties files that ship with Confluent Platform. (The default configs for `producer.properties` and `consumer.properties` are in `etc/kafka/`, and `replicator.properties` is in `etc/kafka-connect-replicator/`.)
Create the following files in `$CONFLUENT_HOME/my-examples/`:

1. Configure the origin cluster in a new file named `consumer.properties`. Edit the file and make sure it contains the addresses of brokers from the **origin** cluster. The default broker list will match the origin cluster you started earlier.

```
# Origin cluster connection configuration
bootstrap.servers=localhost:9082
```

2. Configure the destination cluster in a new file named `producer.properties`. Edit the file and make sure it contains the addresses of brokers from the **destination** cluster. The default broker list will match the destination cluster you started earlier.

```
# Destination cluster connection configuration
bootstrap.servers=localhost:9092
```

3. Define the Replicator configuration in a new file named `replication.properties` for the Connect worker. This quick start shows a configuration for `topic.rename.format` but any of the Confluent Replicator Configuration Properties that are not connection related can be supplied in this file.

```
# Replication configuration
topic.rename.format=${topic}.replica
replication.factor=1
config.storage.replication.factor=1
```

```
offset.storage.replication.factor=1
status.storage.replication.factor=1
confluent.topic.replication.factor=1
```

Tip

- If no port is defined in `replication.properties`, this worker runs on its default port `8083`, which is the desired config for this deployment.
- The replication factor properties (all set to `1`) are used because these test clusters are small. The recommended minimum cluster size in production is `3` and this is the default for these properties.

Start the replicator

After you have created the necessary configuration files, start Replicator executable in its own terminal with the command below (assuming the properties files are in `my-examples`).

```
./bin/replicator --cluster.id replicator --consumer.config my-examples/consumer.properties --producer.config my-examples/producer.properties --replication.config my-examples/replication.properties --whitelist 'test-topic'
```

Some Replicator executable parameters can be passed on the command line:

- `--cluster.id` - An identifier used to determine which Replicator cluster this executable should join. Multiple Replicator executable instances with the same `cluster.id` will work together.
- `--consumer.config` - The path to the origin cluster configuration
- `--producer.config` - The path to the destination cluster configuration
- `--replication.config` - The path to a file containing any non connection specific configuration. Command line arguments will override these configurations.
- `--whitelist` - A list of topics to replicate from origin to destination

Look for success messages related to starting the source task and creating the replicated topic that indicate Replicator is up and running, and copying topics.

Verify topic replication across the clusters

When Replicator finishes initialization, it checks the origin cluster for topics that need to be replicated.

In this case, it finds `test-topic` and creates the corresponding topic in the destination cluster. You can verify this with the following command.

```
./bin/kafka-topics --describe --topic test-topic.replica --bootstrap-server localh
ost:9092
```

Note that you are checking the existence of `test-topic.replica` because `test-topic` was renamed when it was replicated to the destination cluster, according to your configuration.

Your output should look similar to this:

```
./bin/kafka-topics --describe --topic test-topic.replica --bootstrap-server localh
ost:9092
Topic: test-topic.replica    PartitionCount: 1       ReplicationFactor: 1    Confi
gs: message.timestamp.type=CreateTime,segment.bytes=1073741824
      Topic: test-topic.replica      Partition: 0    Leader: 0       Replicas: 0
Isr: 0  Offline: 0
```

You can also list and describe the topics on the destination cluster. Replicated topics, like `test-topic.replica` will be listed.

```
./bin/kafka-topics --list --bootstrap-server localhost:9092
```

Tip

- To list topics on the origin cluster, run `kafka-topics --list` against `localhost:9082`.
- To view a description of the original topic, run `kafka-topics --describe` but look for `test-topic` and target `localhost:9082`.

At any time after you've created the topic in the origin cluster, you can begin sending data to it using a Kafka producer to write to `test-topic` in the origin cluster. You can then confirm that the data has been replicated by consuming from `test-topic.replica` in the destination cluster. For example, to send a sequence of numbers using Kafka's console producer, run the following command in a new terminal window:

```
seq 10000 | ./bin/kafka-console-producer --topic test-topic --broker-list localhos
t:9082
```

You can confirm delivery in the destination cluster using the console consumer in its own terminal window:

```
./bin/kafka-console-consumer --from-beginning --topic test-topic.replica --bootstr
ap-server localhost:9092
```

If the numbers 1 to 10,000 appear in the consumer output, this indicates that you have successfully created multi-cluster replication.

Press `Ctl-C` to end the consumer readout and return to the command prompt.

Use Control Center to monitor replicators

You can use Control Center to monitor the replicators in your current deployment:

1. Stop Replicator and brokers on both the origin and destination clusters, and then stop the ZooKeeper instances (in that order).

   Press `Ctl-C` in the each command window to stop the processes, but keep the windows open to make it easy to restart each one.
2. Activate the monitoring extension for Replicator by doing the following.

   - Add the full path to `replicator-rest-extension-<version>.jar` to your CLASSPATH.
   - Add `rest.extension.classes=io.confluent.connect.replicator.monitoring.ReplicatorMonitoringExtension` to `my-examples/replication.properties`.
3. Uncomment or add the following lines to the Kafka configuration files for *both* the destination and origin, `etc/kafka/server.properties` and `my-examples/server_origin.properties`, respectively. The configuration for `confluent.metrics.reporter.bootstrap.servers` must point to `localhost` on port `9092` in both files, so you may need to edit one or both of these port numbers. (Searching on `confluent.metrics` will take you to these lines in the files.)

```
confluent.metrics.reporter.topic.replicas=1
metric.reporters=io.confluent.metrics.reporter.ConfluentMetricsReporter
confluent.metrics.reporter.bootstrap.servers=localhost:9092
```

   - The first line indicates to Control Center that your deployment is in development mode, using a replication factor of `1`.

- The other two lines enable metrics reporting on Control Center, and provide access to the Confluent internal topic that collects and stores the monitoring data.

Tip

- For this example, the metrics reporter must point to the cluster that Confluent Control Center bootstraps to, which is the destination cluster. If this is not set properly, metrics on source topics will not show up in Control Center. This is why `etc/kafka/server.properties` and `my-examples/server_origin.properties` must have the same configuration for `confluent.metrics.reporter.bootstrap.servers=localhost:9092`.
- When adapting these steps to more complex, real-world environments, you may decide to use a different approach. For example, in a deployment with multiple instances of Control Center for source and destination, each monitoring its own respective cluster, `confluent.metrics.reporter.bootstrap.servers` should point to source or destination, as appropriate.

4. Edit `my-examples/producer.properties` to add the monitoring interceptor for the producer:

```
# Monitoring interceptor for producer
interceptor.classes=io.confluent.monitoring.clients.interceptor.MonitoringProd
ucerInterceptor
```

5. Edit `my-examples/consumer.properties` to add the monitoring interceptor for the consumer:

```
# Monitoring interceptor for consumer
interceptor.classes=io.confluent.monitoring.clients.interceptor.MonitoringCons
umerInterceptor
```

6. Edit `etc/confluent-control-center/control-center-dev.properties` to add the following two lines that specify origin and destination bootstrap servers for Control Center, as is required for monitoring multiple clusters. (A convenient place to add these lines is near the top of the file under “Control Center Settings”, immediately after the line that specifies `confluent.controlcenter.id`.)

```
# multi-cluster monitoring
confluent.controlcenter.kafka.origin.bootstrap.servers=localhost:9082
confluent.controlcenter.kafka.destination.bootstrap.servers=localhost:9092
```

Tip

Control Center requires the host and port of the Connect REST endpoint to know where to look for Replicator monitoring metrics. In the config file used for this

example (`control-center-dev.properties`), this is configured for you on the default port, and so works out-of-the-box:

```
# A comma separated list of Connect host names
confluent.controlcenter.connect.cluster=http://localhost:8083
```

The production-ready config file (`control-center-production.properties`) has the default commented out. If you use this file instead, have multiple Connectors, or want to configure Connect clusters differently, you must specify the Connect endpoint(s), either by uncommenting the default or specifying hosts for your own Connect clusters.

Restart the ZooKeeper instances on the destination and origin clusters with the same commands used above, for example:

```
./bin/zookeeper-server-start etc/kafka/zookeeper.properties

./bin/zookeeper-server-start my-examples/zookeeper_origin.properties
```

7. Restart the brokers on the destination and origin clusters with the same commands used above, for example:

```
./bin/kafka-server-start etc/kafka/server.properties

./bin/kafka-server-start my-examples/server_origin.properties
```

8. Restart Replicator and the Connect worker with the same command as above. For example:

```
./bin/replicator --cluster.id replicator --consumer.config my-examples/consume
r.properties --producer.config my-examples/producer.properties --replication.c
onfig my-examples/replication.properties --whitelist 'test-topic'
```

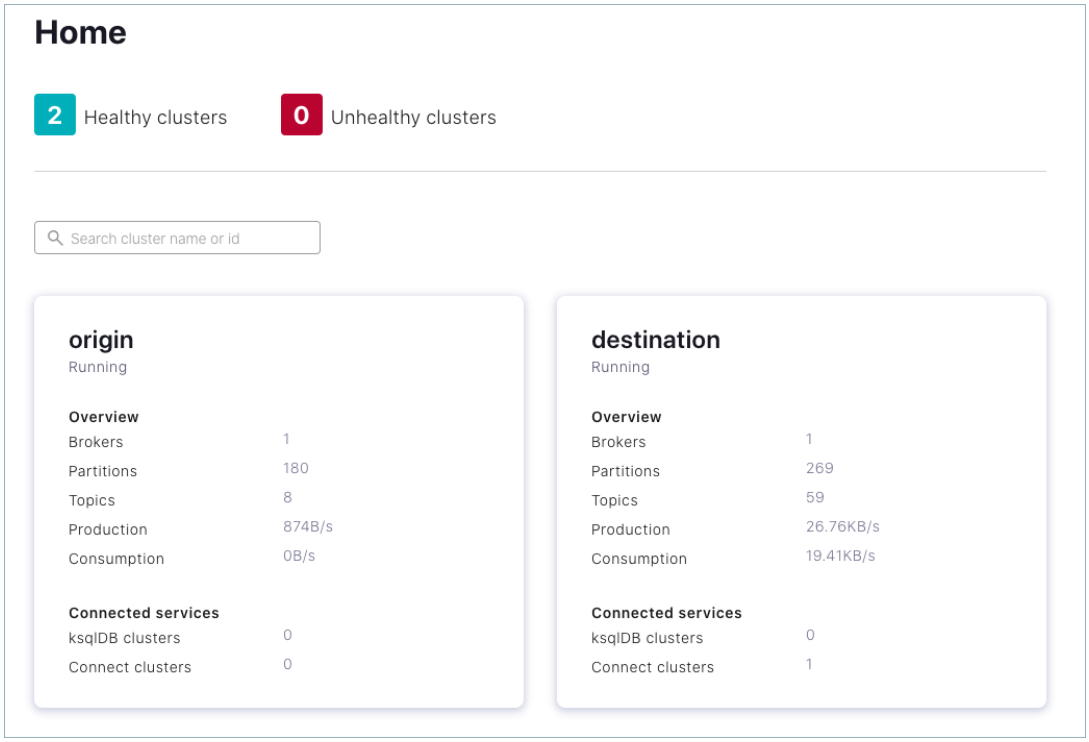
9. Launch Control Center with the following command.

```
./bin/control-center-start etc/confluent-control-center/control-center-dev.pro
perties
```

If no port is defined in `control-center-dev.properties`, Control Center runs by default on port `9021`. This is the desired config for this deployment.

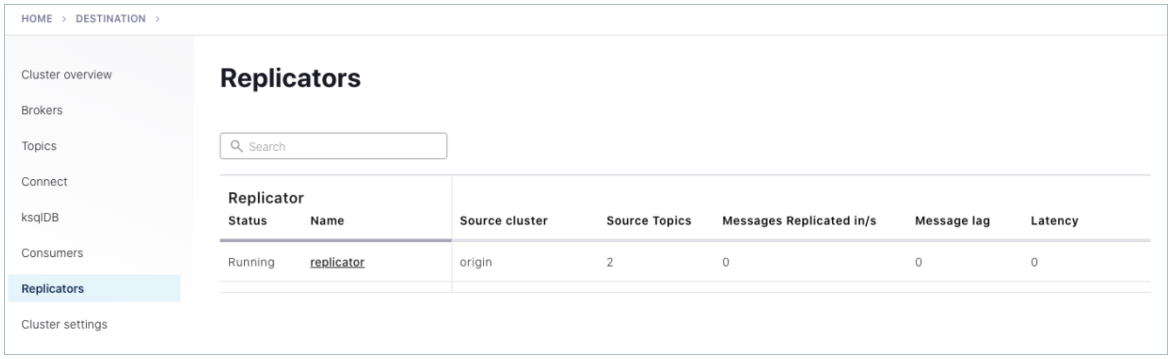
10. Open Control Center at http://localhost:9021/ in your web browser.

The clusters are rendered on Control Center with auto-generated names, based on your configuration.



11. (Optional) On Control Center, edit the cluster names to suit your use case.

12. On Control Center, select the destination cluster, click **Replicators** on the navigation panel, and use Control Center to monitor replication performance and drill down on source and replicated topics.



To see messages produced to both the original and replicated topic on Control Center, try out `kafka-consumer-perf-test` in its own command window to auto-generate test data to `test-topic`.

```
kafka-producer-perf-test \
   --producer-props bootstrap.servers=localhost:9082 \
   --topic test-topic \
   --record-size 1000 \
   --throughput 1000 \
   --num-records 3600000
```

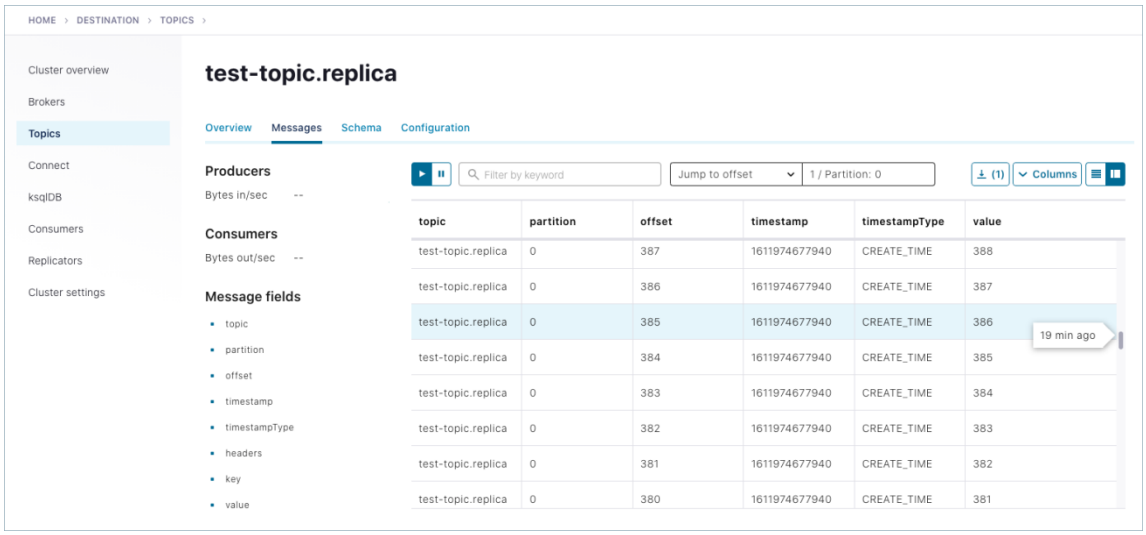
The command provides status output on messages sent, as shown:

```
4999 records sent, 999.8 records/sec (0.95 MB/sec), 1.1 ms avg latency, 240.0
ms max latency.
5003 records sent, 1000.2 records/sec (0.95 MB/sec), 0.5 ms avg latency, 4.0 m
s max latency.
5003 records sent, 1000.2 records/sec (0.95 MB/sec), 0.6 ms avg latency, 5.0 m
s max latency.
5001 records sent, 1000.2 records/sec (0.95 MB/sec), 0.3 ms avg latency, 3.0 m
s max latency.
5001 records sent, 1000.0 records/sec (0.95 MB/sec), 0.3 ms avg latency, 4.0 m
s max latency.
5000 records sent, 1000.0 records/sec (0.95 MB/sec), 0.8 ms avg latency, 24.0
ms max latency.
5001 records sent, 1000.2 records/sec (0.95 MB/sec), 0.6 ms avg latency, 3.0 m
s max latency.
...
```

Like before, you can consume these messages from the command line, using kafka-console-consumer to verify that the replica topic is receiving them:

```
./bin/kafka-console-consumer --from-beginning --topic test-topic.replica --boo
tstrap-server localhost:9092
```

You can also verify this on Control Center. Navigate to `test-topic` on the origin cluster to view messages on the original topic, and to `test-topic.replica` on the destination to view messages on the replicated topic.



13. To learn more about monitoring Replicators in Control Center.

14. When you have completed your experiments with the tutorial, be sure to perform clean up as follows:

    - Stop any producers and consumers using `Ctl-C` in the each command window.

- Use `Ctl-C` in each command window to stop each service in reverse order to which you started them (stop Control Center first, then Replicator, Kafka brokers, and finally ZooKeepers).

Troubleshooting

If you run into trouble at any point with getting things up and running or with Control Center monitoring of replicators, here are some things to check:

- Make sure the configurations in all properties files are correct, and port numbers match origin and destination ports.
- For monitoring with Control Center, make sure that your configurations match the monitoring requirements. If you are using the production-ready Control Center configuration file instead of the "dev" version shown in this example, make sure you have specified the Connect endpoint.
- Verify that the monitoring extension is installed per Replicator Monitoring Extension and is in your CLASSPATH, especially in the shells where you start the Kafka brokers. Check this by running `echo $CLASSPATH` in open command windows.
- If you are using the `systemctl` command to start the monitoring service, make sure you follow the steps in systemctl Command Configurations. If you don't configure the environment variables properly, Connect will fail to start.
- To retry the tutorial:
  1. Use `Ctl-C` in each command window to stop each service in reverse order to which you started them (stop Control Center first, then Replicator, Kafka brokers, and finally ZooKeepers).
  2. Delete stale log and data files in `/tmp` that may conflict with a new run of the clusters and topics. For example, remove these files: `/tmp/confluent/control-center/`, `/tmp/zookeeper`, `/tmp/zookeeper_origin`, `/tmp/kafka-logs`, `/tmp/kafka-logs-origin`, `/tmp/control-center-logs`
  3. From here, you can start again with your current install of Confluent Platform or even reinstall Confluent Platform and try the entire process from scratch.