

# TDMQ for CKafka

## Migration Guide

### Product Documentation



## Copyright Notice

©2013–2026 Tencent Cloud. All rights reserved.

Copyright in this document is exclusively owned by Tencent Cloud. You must not reproduce, modify, copy or distribute in any way, in whole or in part, the contents of this document without Tencent Cloud's the prior written consent.

## Trademark Notice



All trademarks associated with Tencent Cloud and its services are owned by the Tencent corporate group, including its parent, subsidiaries and affiliated companies, as the case may be. Trademarks of third parties referred to in this document are owned by their respective proprietors.

## Service Statement

This document is intended to provide users with general information about Tencent Cloud's products and services only and does not form part of Tencent Cloud's terms and conditions. Tencent Cloud's products or services are subject to change. Specific products and services and the standards applicable to them are exclusively provided for in Tencent Cloud's applicable terms and conditions.

# Contents

## Migration Guide

Migration Solution Overview

Migrating Cluster Using Open-Source Tool

Step 1: Purchasing Cloud Instance

Step 2: Migrating Topics to the Cloud

Step 3. Migrating Data to Cloud

Solution 1: Single-Write Dual-Consumption Migration

Solution 2: Single-Producer Single-Consumer Migration

Solution 3: Using Mirrmaker for Migration

Solution 4: Migrating Unconsumed Data

# Migration Guide

## Migration Solution Overview

Last updated: 2026-01-20 17:19:21

### Scenarios

This document describes feasible solutions for migrating self-built Kafka clusters or those of other cloud vendors to TDMQ for CKafka (CKafka) clusters. You can select the most suitable migration method based on your actual business requirements.

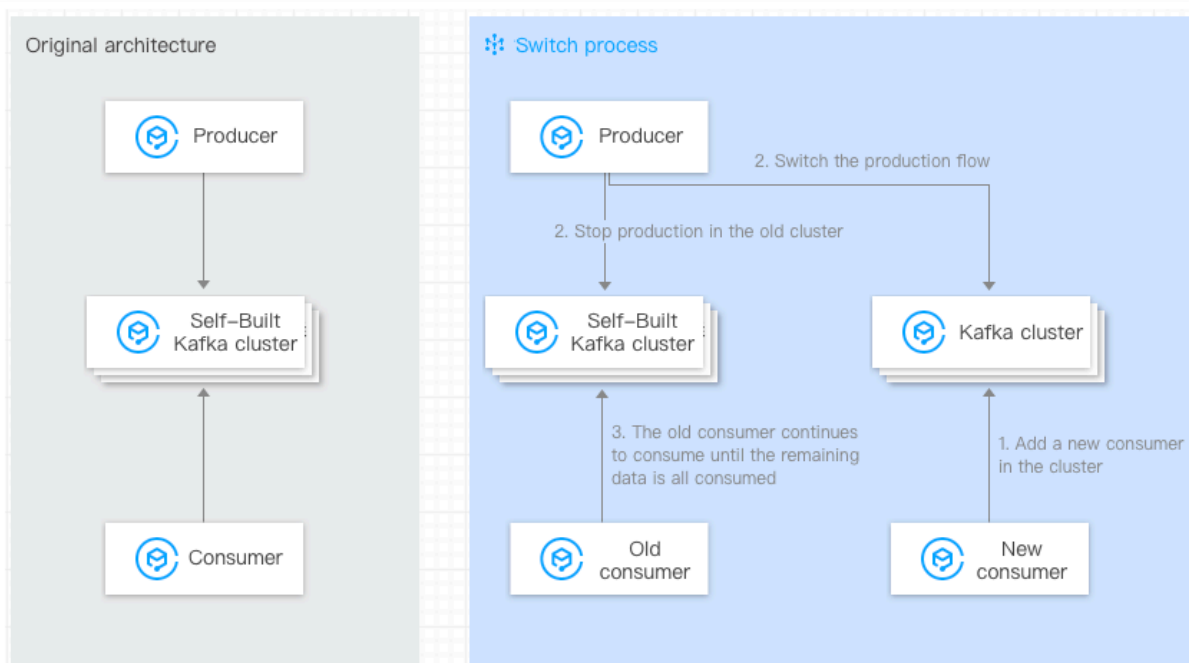
Currently, two migration solutions are provided.

1. Use the connector: Use the connector feature provided by CKafka to achieve migration. It supports multiple scenarios, including cloud and off-cloud (cross-cloud and hybrid cloud) scenarios. For details, see [Configuring Data Synchronization Between Kafka Instances](#).
2. Use an open-source tool: Use an open-source tool to complete migration. The specific operation steps of this solution are introduced below.

### Migrating with an Open-Source Tool

#### Solution 1: Single-Producer Dual-Consumer

This solution is overall simple, clear, and easy to follow. It involves no data backlog and enables a smooth transition.



#### Approach:

1. [Complete the migration of topic metadata](#).

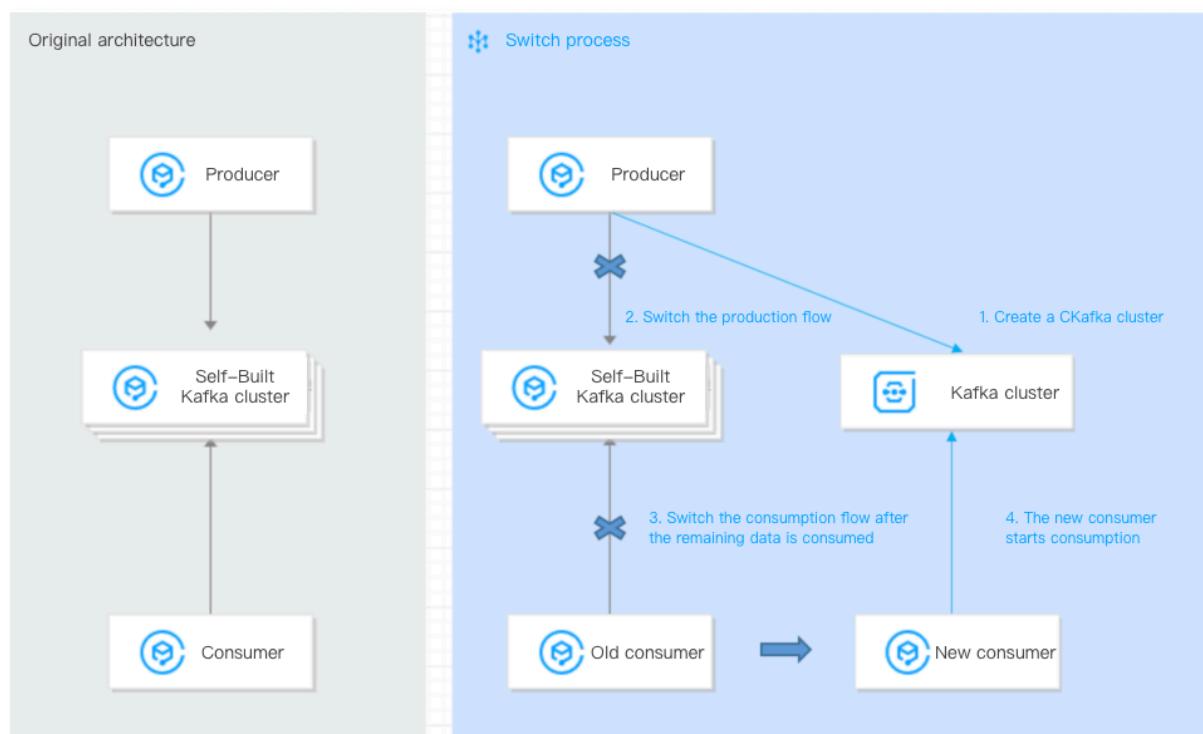
2. The existing consumers in the self-built Kafka cluster remain unchanged.
3. Start a new consumer on the CKafka client, configure the bootstrap-server of the new CKafka cluster, and consume data from the new CKafka cluster.
4. Wait until all consumers have listened to the new CKafka cluster.
5. Switch the production of the self-built cluster to the new CKafka cluster (configure the bootstrap-server of the new CKafka cluster).
6. The existing consumers in the self-built Kafka cluster continue to consume the remaining data in the self-built Kafka cluster, and the original consumers can only be taken offline when all the data has been consumed.

#### Advantages and disadvantages:

- Advantages: The overall migration process is simple, clear, and easy to follow. It involves no data backlog and enables a smooth transition.
- Disadvantages: An additional consumer is required.

## Solution 2: Single-Producer Single-Consumer

This solution is overall simple, clear, and easy to follow.



#### Approach:

1. [Complete the migration of topic metadata](#).
2. Switch the production of the self-built Kafka cluster to the new CKafka cluster (configure the bootstrap-server of the new CKafka cluster).
3. Wait for the consumers in the self-built cluster to finish consuming the remaining data.

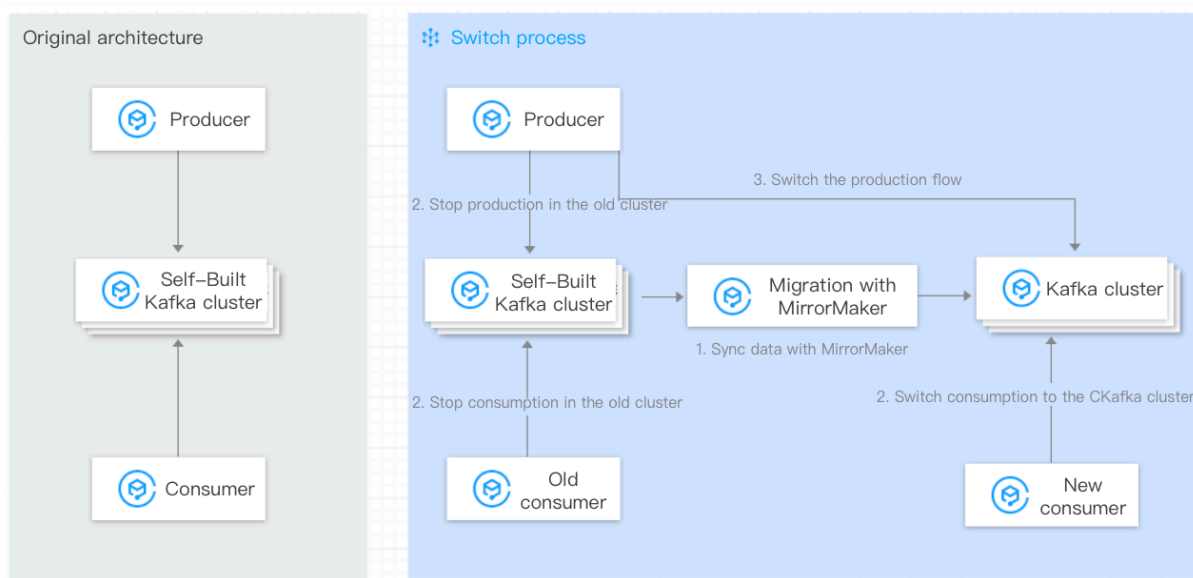
- Switch the existing consumers to consume data from the new CKafka cluster (configure the bootstrap-server of the new CKafka cluster).

#### Advantages and disadvantages:

- Advantages: The overall migration process is simple, clear, and easy to follow. It enables a smooth transition.
- Disadvantages: After production is switched to the CKafka cluster and before the existing consumers are switched to the CKafka cluster, a certain amount of message backlog may accumulate in the CKafka cluster.

### Solution 3: Migrating with Mirrormaker

This solution involves migrating the existing data from a self-built Kafka cluster to a CKafka cluster.



#### Approach:

- Complete the migration of topic metadata.
- The existing consumers in the self-built Kafka cluster remain unchanged.
- Start the data synchronization feature of the [Mirrormaker tool](#).
- Wait for data synchronization to complete, modify consumer configurations, and switch the consumers.
- Wait for data synchronization to complete, modify producer configurations, and switch the producer.
- Migration is completed.

#### Advantages and disadvantages:

- Advantages: The overall migration process is simple, clear, and easy to follow, allowing historical data to be synchronized to the CKafka cluster.
- Disadvantages: After being switched to the target cluster, the consumers need to start consuming data from the beginning. Consumption idempotence is required.

# Migrating Cluster Using Open-Source Tool

## Step 1: Purchasing Cloud Instance

Last updated: 2026-01-20 17:19:21

### Scenarios

When you select specifications for a new instance, base your decision on the current workload (such as peak bandwidth, storage, and the number of partitions) as well as future scalability needs. It is recommended to use monitoring data from the existing instance as a reference to ensure the new instance reserves a 20% to 30% performance buffer and verify version compatibility. TDMQ for CKafka (CKafka) supports online adjustment of instance specifications, allowing you to deploy an instance based on baseline requirements during the initial migration phase, and then scale in or out based on actual business pressure.

- For capability differences between different types of CKafka instances and selection recommendations, see [Capability Comparison](#) and [Selection Recommendations](#).
- For adjustment of CKafka cluster specifications, see [Changing Instance Specifications](#).

### Operation Steps

1. Log in to the [CKafka console](#).
2. In the left sidebar, select **Instance List** and click **Create** to go to the instance purchase page.
3. On the instance purchase page, specify the purchase information based on your business requirements. For detailed parameter descriptions, see [Creating an Instance](#).
4. After confirming the fees, click **Buy Now**. On the order payment page, click **Pay**. After completing the payment, wait for 5–10 minutes to see the created instance on the instance list page.

# Step 2: Migrating Topics to the Cloud

Last updated: 2026-01-20 17:19:21

## Scenarios

This document describes how to use the migration tool provided by TDMQ for CKafka (CKafka) to migrate topics from a self-built Kafka cluster to a CKafka instance.

## Prerequisites

- You have [purchased a cloud instance](#).
- You have [downloaded Python 2](#).

## Operation Steps

1. Download the [migration tool](#) and decompress it to a server that can access the brokers and ZooKeeper (ZK) of your self-built instance.
2. Specify the configuration parameters in the `ckafka-migrate.py` file.

### Note:

- Ensure that the server where migration will be performed maintains network connectivity with both the CKafka instance and the self-built Kafka cluster.
- The user associated with the cloud API key needs to have write permissions for CKafka. It is recommended to use the key pair of the root account.

```
# your local broker ip:port list
# Broker list for self-built instances: ["broker1:port1","ip2:port2"].
bootstrapServers = ["$ip:$port"]

# your local zk ip:port list
# ZK list for self-built instances: ["zk1:port1","zk2:port2"], which
# is optional. If it is not set, bootstrapServers will be used to obtain
# the information of the source cluster.
sourceZk = ["$ip:$port"]
# If authentication is enabled for ZK, this parameter needs to be set
# in the format [("digest", "$user:$password")].
zkAuthData = []
# If the ZK path has a prefix, this parameter needs to be set, such as
"/cluster1".
```

```
zkPathPrefix = ""

# kafka instanceId
# Cloud instance ID "ckafka-xxx".
instanceId = "$yourinstanceId"
# topic regex, just migrate match topics
# Regular expression for a topic name. If it is specified, only
matched topics will be migrated.
topicRegex = ""

# your secretId and secretKey
# Key pair for a Tencent Cloud account.
secretId = "$yoursecretId"
secretKey = "$yoursecretKey"

# your cloud instance region
# Region of a cloud instance. For the available region codes of
CKafka, see Regions and AZs at
https://cloud.tencent.com/document/product/597/44597.

region = "ap-tokyo"

# if you make sure the migrate topic List, please modify checkFlag = 1
# Check marker. When it is set to 0, only the list of topics to be
migrated will be displayed without performing actual migration. First,
set it to 0 to check the list of topics to be migrated. After
confirming that everything is correct, change it to 1 to start
migration.
# 0: The script terminates after the topics to be migrated are listed.
# 1: The topics to be migrated are listed, and migration starts.
checkFlag = 0

# force transfer your cloud-topic config to migrate
# When the parameter is set to 0, inconsistent properties will cause
the failure to migrate a local topic to the cloud. When it is set to
1, the properties of the local topic will be forcibly converted to the
value nearest to the value of a cloud topic.
# For example, a cloud topic only supports 1, 2, or 3 replicas. If a
local topic has 5 replicas, it will not be migrated to the cloud. If
```

force is set to 1, the value nearest to 5 (3 replicas) will be used to create a 3-replica topic in the cloud.

# 0: Local topics whose number of replicas or properties are incompatible with those of cloud topics are skipped.

# 1: Local topics whose number of replicas or properties are incompatible with those of cloud topics are forcibly migrated to the cloud. The properties of cloud topics will be adjusted to the nearest possible values based on the properties of self-built topics (no data from the local self-built source Kafka instance will be modified).

force = 0

Parameter	Description
bootstrapServers	Broker list for self-built instances: ["ip1:port1","ip2:port2"].
sourceZk	ZK list for self-built instances: ["zk1:port1","zk2:port2"], which is optional. If it is not set, bootstrapServers will be used to obtain the information of the source cluster.
zkAuthData	If authentication is enabled for ZK, this parameter needs to be set in the format [("digest", "\$user:\$password")].
zkPathPrefix	If the ZK path has a prefix, this parameter needs to be set, such as "/cluster1".
instanceId	ID of the CKafka instance you purchased in <a href="#">Purchasing a Cloud Instance</a> . It can be copied from the <b>Instance List</b> page in the console.
secretId	Key pair ID for a Tencent Cloud account.
secretKey	Key pair password for a Tencent Cloud account.
region	Deployment region you selected in <a href="#">Purchasing a Cloud Instance</a> , with region codes provided in script comments.
checkFlag	Check marker. When it is set to 0, only the list of topics to be migrated is displayed without starting migration. When it is set to a non-zero value, topic migration starts.
topicRegex	Regular expression for a topic name. If it is not specified, all topics will be migrated. If it is specified, only matched topics will be migrated.
force	Whether to force migration. When it is set to 0, inconsistent properties will cause the failure to migrate a local topic to the cloud. When it is set to 1, topic

properties will be forcibly converted based on the value nearest to the supported value of a cloud topic.

- Set the checkFlag parameter of kafka-migrate.py to 0, run the kafka-migrate.py script using Python, and check the list of topics to be migrated based on the output results.

### Note:

If certain self-built topics are missing, it may be because their names are non-compliant with the naming rules, or their number of replicas or property values are incompatible with those of cloud topics.

```
[root@M-0-21-centos ~/migrateCloud/migrateToCkafkaTool]# python kafka-migrate.py
check topic List to migrate: ["test1", "test0", "test108", "test109", "test5", "test4", "test7", "test6", "test102", "test103", "test100",
"test101", "test106", "test107", "test104", "test105", "test182", "test183", "test180", "test181", "test186", "test187", "test184", "test
185", "test188", "test189", "test46", "test47", "test44", "test45", "test42", "test43", "test40", "test41", "test48", "test49", "test8", "
test179", "test178", "test177", "test176", "test175", "test174", "test173", "test172", "test171", "test170", "test55", "test54", "test57",
"test56", "test51", "test50", "test53", "test52", "test59", "test58", "test168", "test169", "test164", "test165", "test166", "test167", "
test160", "test161", "test162", "test163", "test60", "test61", "test62", "test63", "test64", "test65", "test66", "test67", "test68", "test
69", "test2", "test159", "test158", "test151", "test150", "test153", "test152", "test155", "test154", "test157", "test156", "test79", "tes
t78", "test77", "test76", "test75", "test74", "test73", "test72", "test71", "test70", "test146", "test147", "test144", "test145", "test142
", "test143", "test140", "test141", "test148", "test149", "test82", "test83", "test80", "test81", "test86", "test87", "test84", "test85",
"test88", "test89", "test3", "test133", "test132", "test131", "test130", "test137", "test136", "test135", "test134", "test139", "test138",
"test91", "test90", "test93", "test92", "test95", "test94", "test97", "test96", "test99", "test98", "test35", "test19", "test18", "test11
", "test10", "test13", "test12", "test15", "test14", "test17", "test16", "test120", "test121", "test122", "test123", "test124", "test125",
"test126", "test127", "test128", "test129", "test24", "test25", "test26", "test27", "test20", "test21", "test22", "test23", "test28", "tes
t29", "test", "test119", "test118", "test115", "test114", "test117", "test116", "test111", "test110", "test113", "test112", "test195", "te
st194", "test197", "test196", "test191", "test190", "test193", "test192", "test199", "test198", "test33", "test32", "test31", "test30", "
test37", "test36", "test9", "test34", "test39", "test38"]
```

- Set the checkFlag parameter of kafka-migrate.py to 1 and run the kafka-migrate.py script using Python to start topic migration.

```
[root@M-0-21-centos ~/migrateCloud/migrateToCkafkaTool]# python kafka-migrate.py
preper to migrate these topics ["test1", "test0", "test108", "test109", "test5", "test4", "test7", "test6", "test102", "test103", "test100
", "test101", "test106", "test107", "test104", "test105", "test182", "test183", "test180", "test181", "test186", "test187", "test184", "tes
t185", "test188", "test189", "test46", "test47", "test44", "test45", "test42", "test43", "test40", "test41", "test48", "test49", "test8", "
test179", "test178", "test177", "test176", "test175", "test174", "test173", "test172", "test171", "test170", "test55", "test54", "test57",
"test56", "test51", "test50", "test53", "test52", "test59", "test58", "test168", "test169", "test164", "test165", "test166", "test167", "tes
t160", "test161", "test162", "test163", "test60", "test61", "test62", "test63", "test64", "test65", "test66", "test67", "test68", "test
69", "test2", "test159", "test158", "test151", "test150", "test153", "test152", "test155", "test154", "test157", "test156", "test79", "tes
t78", "test77", "test76", "test75", "test74", "test73", "test72", "test71", "test70", "test146", "test147", "test144", "test145", "test142
", "test143", "test140", "test141", "test148", "test149", "test82", "test83", "test80", "test81", "test86", "test87", "test84", "test85",
"test88", "test89", "test3", "test133", "test132", "test131", "test130", "test137", "test136", "test135", "test134", "test139", "test138",
"test91", "test90", "test93", "test92", "test95", "test94", "test97", "test96", "test99", "test98", "test35", "test19", "test18", "test11
", "test10", "test13", "test12", "test15", "test14", "test17", "test16", "test120", "test121", "test122", "test123", "test124", "test125",
"test126", "test127", "test128", "test129", "test24", "test25", "test26", "test27", "test20", "test21", "test22", "test23", "test28", "tes
t29", "test", "test119", "test118", "test115", "test114", "test117", "test116", "test111", "test110", "test113", "test112", "test195", "te
st194", "test197", "test196", "test191", "test190", "test193", "test192", "test199", "test198", "test33", "test32", "test31", "test30",
"test37", "test36", "test9", "test34", "test39", "test38"] to cloud instance kafka-lxmap3v4 .....

create Migreate to Cloud Task Success!

1/201 migrate topic test1 to kafka-lxmap3v4 success,topicId:topic-9c9q3i2i

2/201 migrate topic test0 to kafka-lxmap3v4 success,topicId:topic-o57vfbw2
```

- Log in to the [CKafka console](#), view the task list on the **Cloud Migration** page, and wait for topic migration to complete.

The task list is as follows:

ID/Name	AZ	Topic Migration	Consumer Group Migration	Data Migration
kafka-lxmap3v4	Changsha Zone 1	0/1	0/0	N/A

The page indicating successful migration is as follows:

```
199/201 migrate topic test34 to ckafka-lxmap3v4 success,topicId:topic-a4bo0ske  
200/201 migrate topic test39 to ckafka-lxmap3v4 success,topicId:topic-bv7pezvi  
201/201 migrate topic test38 to ckafka-lxmap3v4 success,topicId:topic-m96t1b0e  
migrate topics to cloud finished!
```

# Step 3. Migrating Data to Cloud

## Solution 1: Single-Write Dual-Consumption Migration

Last updated: 2026-01-20 17:19:21

### Scenarios

This article mainly introduces the method of using the single-write dual-consumption scheme to migrate data from self-built Kafka clusters to CKafka.

### Prerequisites

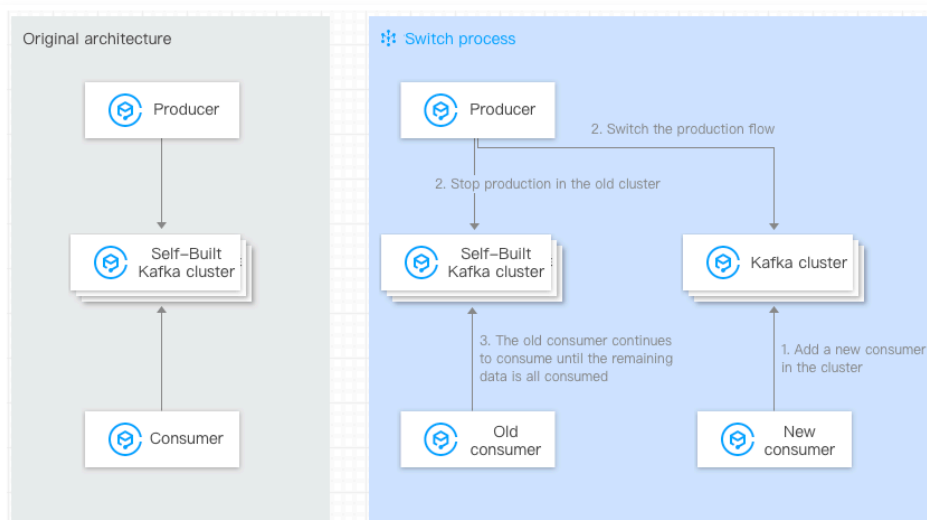
- You have [purchased a CKafka instance](#).
- You have [migrated topics to the cloud](#).

### Operation Steps

When data ordering requirements are not high, the switch can be performed by adopting multiple consumers for parallel consumption.

The single-write dual-consumption approach is simple, clear, and easy to operate, with no data backlog, enabling a smooth transition; however, it requires the business operations side to add an additional set of consumers.

The migration steps are as follows:



1. The old consumers remain unchanged. On the consumer side, start new consumers, configure the bootstrap-server of the new cluster, and consume the new CKafka clusters.

You need to configure the IP address in `--bootstrap-server` to the access network of the

CKafka instance. On the instance details page in the console, in the **Access Mode** module, copy the network information in the Network column.

```
./kafka-console-consumer.sh --bootstrap-server xxx.xxx.xxx.xxx:9092 --  
from-beginning --new-consumer --topic topicName --consumer.config  
../config/consumer.properties
```

2. Switch the production flow so that the producer starts sending data to the CKafka instance. Change the IP address in broker-list to the access network address of the CKafka instance. Set topicName to the Topic name in the CKafka instance:

```
./kafka-console-producer.sh --broker-list xxx.xxx.xxx.xxx:9092 --  
topic topicName
```

3. Existing consumers require no special configuration and continue to consume data from the self-built Kafka clusters. After the consumption completion of the data in the original self-built clusters, the migration is complete.

 **Note:**

The commands provided above are testing commands. In actual business operations, you only need to modify the broker address configured for the corresponding application and then restart the application.

# Solution 2: Single-Producer Single-Consumer Migration

Last updated: 2026-01-20 17:19:21

## Scenarios

This document describes how to migrate data from a self-built Kafka cluster to TDMQ for CKafka (CKafka) using the single-producer single-consumer solution.

## Prerequisites

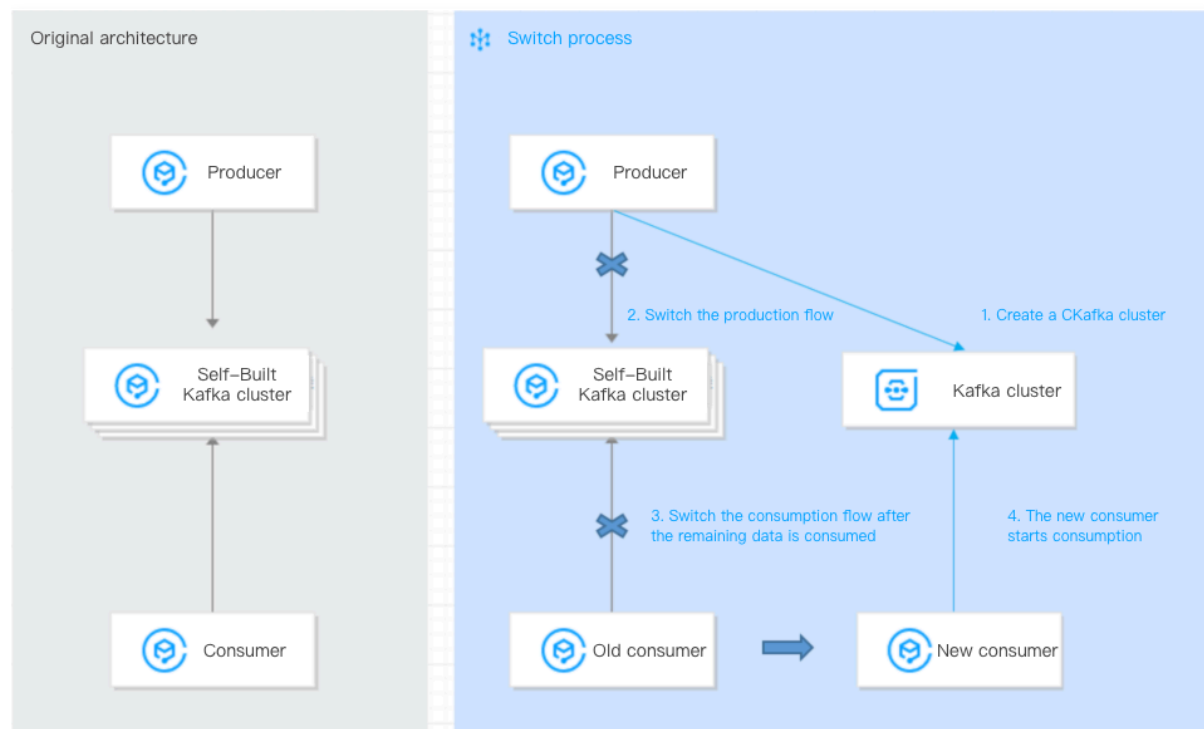
- You have [purchased a CKafka instance](#).
- You have [migrated topics to the cloud](#).

## Operation Steps

The prerequisite for ensuring message order is to strictly control data consumption by a single consumer. Therefore, the timing of the switch is critical.

The single-producer single-consumer method is straightforward and easy to operate. However, after the producer switches to the new cluster and before the old consumer switches to the new cluster, a certain amount of messages will be backlogged in the new cluster.

The migration steps are as follows:



1. Switch the production flow so that the producer starts sending data to the CKafka instance. Change the IP address in `broker-list` to the access network address of the CKafka instance. You can copy it from the Network column in the **Access Method** module on the instance details page in the console. Set `topicName` to the topic name in the CKafka instance.

```
./kafka-console-producer.sh --broker-list xxx.xxx.xxx.xxx:9092 --  
topic topicName
```

2. The original consumer requires no configuration changes. It continues to consume data from the self-built Kafka cluster until all data is consumed.
3. When the original consumer completes consumption, switch the consumer to the new CKafka cluster with the following configuration (a single consumer ensures message order). For the new consumer, change the IP address in `--bootstrap-server` to the access network address of the CKafka instance:

 **Note:**

If the consumer is deployed on a Cloud Virtual Machine (CVM) instance, you can continue using the original consumer for consumption.

```
./kafka-console-consumer.sh --bootstrap-server xxx.xxx.xxx.xxx:9092 --  
from-beginning --new-consumer --topic topicName --consumer.config  
../config/consumer.properties
```

4. The new consumer continues to consume data from the CKafka cluster, and the migration is completed. If the consumer is deployed on a CVM instance, you can continue using the original consumer for consumption.

 **Note:**

The commands provided above are testing commands. In actual business operations, you only need to modify the broker address configured for the corresponding application and then restart the application.

# Solution 3: Using Mirrormaker for Migration

Last updated: 2026-01-20 17:19:22

## Scenarios

This task describes how to migrate data from a self-built Kafka cluster to a TDMQ for CKafka (CKafka) cluster using Mirrormaker.

The Mirrormaker tool of Kafka enables backing up data from a self-built Kafka cluster to a CKafka cluster. The specific mechanism is as follows:

Mirrormaker uses a consumer to consume messages from a self-built Kafka cluster, then sends the data to a CKafka cluster via a producer. Finally, you switch the production/consumption configuration of the client to the access network of the cloud instance to complete data migration from the self-built Kafka cluster to the CKafka cluster.

## Prerequisites

- You have [purchased a CKafka instance](#).
- You have [migrated topics to the cloud](#).

## Operation Steps

1. [Download the Mirrormaker tool](#) and decompress it locally.

### Note:

This document takes [kafka\\_2.11-1.1.1.tgz](#) as an example.

2. Configure the consumer.properties file.

```
# list of brokers used for bootstrapping knowledge about the rest of
the cluster
# format: host1:port1,host2:port2 ...
bootstrap.servers=localhost:9092

# consumer group id
group.id=test-consumer-group

partition.assignment.strategy=org.apache.kafka.clients.consumer.RoundR
obinAssignor
```

```
# What to do when there is no initial offset in Kafka or if the
current
# offset does not exist any more on the server: latest, earliest, none
#auto.offset.reset=
```

Parameter	Description
bootstrap.servers	List of broker access points for self-built instances.
group.id	Consumer group ID used during data migration. It must not conflict with the names of the existing consumers of self-built instances.
partition.assignment.strategy	Partition assignment policy, such as partition.assignment.strategy=org.apache.kafka.clients.consumer.RoundRobinAssignor.

### 3. Configure the producer.properties file.

```
# list of brokers used for bootstrapping knowledge about the rest of
the cluster
# format: host1:port1,host2:port2 ...
bootstrap.servers=localhost:9092

# specify the compression codec for all data generated: none, gzip,
snappy, lz4
compression.type=none
```

Parameter	Description
bootstrap.servers	Access network for cloud instances. You can copy it in the Network column in the <b>Access Mode</b> module on the instance details page in the console.
compression.type	Data compression type. CKafka does not support the GZIP compression format.

### 4. Start the Mirrormaker migration tool in the `.bin` directory to start migration.

```
sh bin/kafka-mirror-maker.sh --consumer.config
config/consumer.properties --producer.config
config/producer.properties --whitelist topicName
```

**Note:**

whitelist is a regular expression. Topics that match the regular expression name are migrated.

- Run `kafka-consumer-groups.sh` in the `.bin` directory to view the consumption offset of the self-built cluster.

```
bin/kafka-consumer-groups.sh --new-consumer --describe --bootstrap-server access point of the self-built cluster --group test-consumer-group
```

**Note:**

group refers to the consumer group ID used during data migration.

```
[root@VM-0-36-centos ~]# /usr/local/services/kafka_2.11-1.1.1/bin/kafka-consumer-groups.sh --bootstrap-server 10.0.0.36:9092 --describe --group mm1-migrate-ckafka
Note: This will not show information about old Zookeeper-based consumers.
```

TOPIC	PARTITION	CURRENT-OFFSET	LOG-END-OFFSET	LAG	CONSUMER-ID	HOST
test26	0	-	461	-	mm1-migrate-ckafka-0-598a37fa-d5f1-4ad4-ba7c-579b8c7a6d5e	/10.0.0.36
test193	0	-	460	-	mm1-migrate-ckafka-0-598a37fa-d5f1-4ad4-ba7c-579b8c7a6d5e	/10.0.0.36
test89	0	-	461	-	mm1-migrate-ckafka-0-598a37fa-d5f1-4ad4-ba7c-579b8c7a6d5e	/10.0.0.36
test114	0	-	461	-	mm1-migrate-ckafka-0-598a37fa-d5f1-4ad4-ba7c-579b8c7a6d5e	/10.0.0.36
test1	3	-	2251	-	mm1-migrate-ckafka-0-598a37fa-d5f1-4ad4-ba7c-579b8c7a6d5e	/10.0.0.36

## Subsequent Processing

After completing data migration, transfer the production and consumption configuration of the client to the access point of the cloud instance.

# Solution 4: Migrating Unconsumed Data

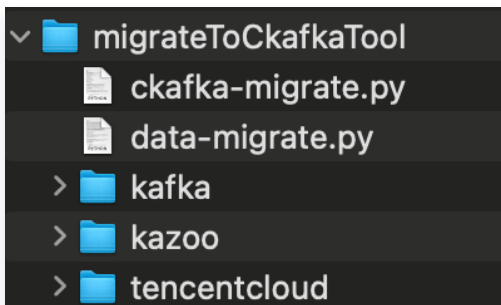
Last updated: 2026-01-20 17:19:22

## Overview

To migrate a cluster to the cloud, you need to migrate unconsumed messages in it to the corresponding topic in the new cluster. You can operate as instructed in this tutorial to synchronize the unconsumed data from the original cluster to the new cluster.

## Prerequisites

1. Ensure that all consumption/production in the original cluster has stopped.
2. Ensure that the retention period for messages to be migrated from the original cluster is long enough so that messages in the topic will not be automatically deleted upon expiration during the migration process.
3. The migration script is a Python script that requires the installation of Python 2. In addition, the Python 2 version should be later than version 2.7.1. Version 2.7.5 is recommended.
4. You have downloaded the migration tool [migrateToCkafkaTool](#). The tool package directory is as follows. Go to the migrateToCkafkaTool directory, modify the configuration in the data-migrate.py file, and run data-migrate.py using Python.



## How It Works

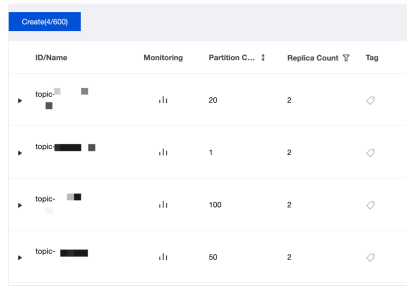
The script scans all group lists in the original cluster and extracts the topic lists that are subscribed to by groups and still contain unconsumed messages. Then, it extracts the **group commit positions** and **topic end positions** for topics that are subscribed to by groups and contain unconsumed messages (if a topic is subscribed to by multiple groups, the smallest group commit position will be used). Messages within these position ranges are then consumed and produced to the corresponding topic partitions in the new cluster.

## Operation Steps

### 1. Creating Corresponding Topics in the Target Cluster

Assume the original cluster is ckafka-47bd7goz, and the target new cluster is ckafka-kzamzogr. As shown in the figure below, topics with the same number of partitions, namely test1, test2, test3, and test4, have been

created for the new cluster.



ID/Name	Monitoring	Partition C...	Replicas Count	Tag
topic-	...	20	2	...
topic-	...	1	2	...
topic-	...	100	2	...
topic-	...	50	2	...

The original cluster kafka-47bd7goz has two groups: test123-group and test34-group. The former subscribes to topics test1, test2, and test3, while the latter subscribes to topics test3 and test4.

## 2. Downloading the Tool Package

After downloading the migration tool, open the script and enter the address configurations for the original and new clusters. Set checkFlag to 0 and run the script to perform a pre-check on the topics to be migrated and target positions.

```
# your local broker ip:port list
# Broker list of the self-built instance ["broker1:port1","ip2:port2"]
bootstrapServers = ["..."] → Source instance

# your cloud kafka instance address
# Cloud instance address "x.x.x.x:9092"
instanceAddress = "... " → Target instance

# A dedicated group for migration data, default migrateLocalToCkafkaGroup, can be customized but should not have
# the same name as the existing consumer group in the cluster, do not delete it after the self-built cluster is
# generated, otherwise the migration progress will be lost
# A dedicated consumer group for data migration, which is 'migrateLocalToCkafkaGroup' by default and can be customized. The name of the custom consumer group cannot be the same as that of any existing consumer
# group. Once you create a custom consumer group in the self-built cluster, don't delete it; otherwise, migration progress data will get lost. migrateGroupId = "migrateLocalToCkafkaGroup"

# topic regex, just migrate match topics
# Topic name regex, which is left empty by default, indicating that all topics will be migrated; otherwise, only topics that match the regex will be migrated.
topicRegex = ""

# The mapping between the local topic and the cloud topic, if not set, the default local and cloud have the same name
# The mapping between the local and cloud topics. If it is not set, the local and cloud topics will have the same name.
# example: localTopicAndCloudTopicMap = {"topic1-local": "topic1-cloud", "topic2-local": "topic2-cloud"}
localTopicAndCloudTopicMap = {}

# if you make sure the migrate topic List, please modify checkFlag = 1
# Check flag. If it is set to 0, the list of topics to be migrated will only be displayed but not migrated. You can set it to 0 first to check the topics to be migrated, confirm that everything is correct, and change it to 1 to start migration.
# 0: List the partition offset data of the topics to be migrated and stop the script.
# 1: List the partition offset data of the topics to be migrated and start migration.
checkFlag = 0 → Set the value to 0 first. If the script runs normally, set the value to 1 and run the script again.

logFileName = "data-migrate"
```

After you run the script, it will output certain information, and a text log file will be written to the current directory.

```
kafka-migrate.py data-migrate_220914_12:09:56.log ← data-migrate.py kafka kazoo tencentcloud
```

## 3. Viewing Output Information

Check the screen output or text log file for the information of **Prepare to migrate**, namely the information of the target migration position.

```
-----Topic:[test3] Partition:(0) migrate-----  
Prepare to migrate local-topic:[test3] partition:0 startOffset:187800 ~ endOffset:264322 data to cloud-topic:[test3]  
  
-----Topic:[test2] Partition:(0) migrate-----  
Prepare to migrate local-topic:[test2] partition:0 startOffset:1873 ~ endOffset:2665 data to cloud-topic:[test2]  
  
-----Topic:[test2] Partition:(1) migrate-----  
Prepare to migrate local-topic:[test2] partition:1 startOffset:1860 ~ endOffset:2621 data to cloud-topic:[test2]  
  
-----Topic:[test2] Partition:(2) migrate-----  
Prepare to migrate local-topic:[test2] partition:2 startOffset:1830 ~ endOffset:2591 data to cloud-topic:[test2]
```

Take test3 as an example. It is subscribed to by both test123-group and test34-group. Check the subscription status of the original cluster.

According to the predefined logic, when a topic is subscribed to by multiple groups, synchronization should start from the smallest commit position, namely 187800. Verify that the output information is consistent with expectations.

```
-----Topic:[test3] Partition:(0) migrate-----  
Prepare to migrate local-topic:[test3] partition:0 startOffset:187800 ~ endOffset:264322 data to cloud-topic:[test3]
```

The other case is that messages in the topic test1 of the original cluster have expired, but the group's commit position falls within the range of the expired messages. Therefore, synchronization will start from the position of the earliest unexpired message in test1.

Take partition 0 of test1 as an example. The script will give a prompt indicating that position 5226 of partition 0 in topic test1 (the minimum position of live messages for the topic) has surpassed the position 3713 of the committed offset subscribed by the group (messages in this position have expired). Therefore, the starting position of synchronization is set to 5226. Since 5226 also represents the current maximum offset for this partition (the total number of messages in this partition is currently 0) and indicates that no messages can be migrated, the script outputs the text "skip migrate...", indicating that data migration for

this partition is skipped.

```

-----Topic:[test1] Partition:(0) migrate-----
the local-topic:[test1] partition:0 topic_beginOffset:5226 large than group_startOffset:3713,and reset startOffset to 5226
skip migrate local-topic:[test1] partition:0 startOffset:5226 ~ endOffset:5226 data to cloud-topic:[test1],because startOffset is equal or more than endOffset

-----Topic:[test1] Partition:(1) migrate-----
the local-topic:[test1] partition:1 topic_beginOffset:5340 large than group_startOffset:3784,and reset startOffset to 5340
skip migrate local-topic:[test1] partition:1 startOffset:5340 ~ endOffset:5340 data to cloud-topic:[test1],because startOffset is equal or more than endOffset

```

## 4. Starting Migration

After confirming that the output information is correct through the above step, set checkFlag to 1 and start migration.

```

-----Topic:[test3] Partition:(0) migrate-----
Prepare to migrate local-topic:[test3] partition:0 startOffset:187800 ~ endOffset:264322 data to cloud-topic:[test3]
begin migrate local-topic:[test3] partition:0 startOffset:187800 ~ endOffset:264322 data to cloud-topic:[test3]
finish migrate local-topic:[test3] partition:0 startOffset:187800 ~ endOffset:264322 data to cloud-topic:[test3]

-----Topic:[test2] Partition:(0) migrate-----
Prepare to migrate local-topic:[test2] partition:0 startOffset:1873 ~ endOffset:2665 data to cloud-topic:[test2]
begin migrate local-topic:[test2] partition:0 startOffset:1873 ~ endOffset:2665 data to cloud-topic:[test2]
finish migrate local-topic:[test2] partition:0 startOffset:1873 ~ endOffset:2665 data to cloud-topic:[test2]

-----Topic:[test2] Partition:(1) migrate-----
Prepare to migrate local-topic:[test2] partition:1 startOffset:1860 ~ endOffset:2621 data to cloud-topic:[test2]
begin migrate local-topic:[test2] partition:1 startOffset:1860 ~ endOffset:2621 data to cloud-topic:[test2]
finish migrate local-topic:[test2] partition:1 startOffset:1860 ~ endOffset:2621 data to cloud-topic:[test2]

-----Topic:[test2] Partition:(2) migrate-----
Prepare to migrate local-topic:[test2] partition:2 startOffset:1830 ~ endOffset:2591 data to cloud-topic:[test2]
begin migrate local-topic:[test2] partition:2 startOffset:1830 ~ endOffset:2591 data to cloud-topic:[test2]
finish migrate local-topic:[test2] partition:2 startOffset:1830 ~ endOffset:2591 data to cloud-topic:[test2]

```

## 5. Checking Whether the Quantity of Migrated Data Is Consistent

Take test3 as an example. 76,522 unconsumed messages from test123-group are expected to be migrated, and all have been successfully written to the test3 topic in the new cluster. Data migration is completed.