TDMQ for CKafka

FAQs

Product Documentation

Contents

FAQs

Instance-Related

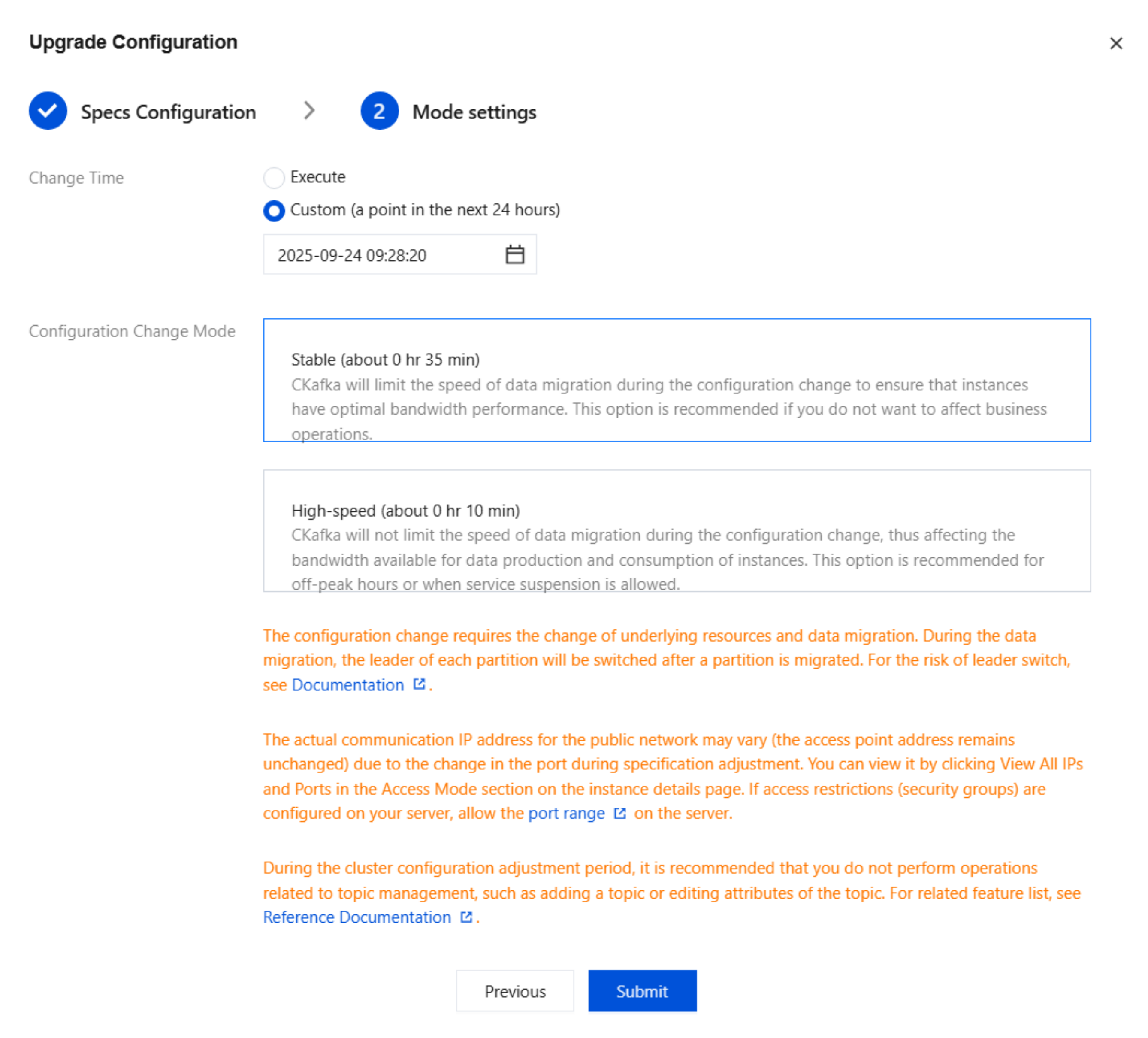
Last updated：2025-09-28 16:17:41

Instance Upgrade Modes and Key Differences

When the system algorithm detects that the current dedicated resource pool has insufficient quota for the upgraded configuration, it must automatically add resources to the resource pool and migrate the instance data. Therefore, there are two upgrade modes:

- When resources are sufficient, the instant upgrade mode increases quota and completes within minutes.
- When resources are insufficient, the data migration mode depends on the accumulated data size in the cluster and may take a few minutes to hours.

When data migration is required, the migration time can be customized.



When data migration is required, there will be two options: migration mode and scheduled migration. The migration mode is divided into two types as follows:

- Stable mode: CKafka limits the data migration speed during the upgrade process to maximize the bandwidth of an instance, suitable for scenarios where business interference is not desired.
- High-speed mode: CKafka does not limit the data migration speed during the upgrade process, which may affect the production and consumption bandwidth of the instance. It is suitable for off-peak hours or scenarios where service interruption is allowed.

Because migration may occupy cluster bandwidth and disk resources, customers may worry about migration affecting business during peak hours. Therefore, scheduled migration can be configured, such as setting it to perform upgrades at midnight. For details, see Changing Instance Specifications.

Why Is Traffic Throttling Triggered When Cluster Bandwidth Percentage Is below 100%

From the monitoring interface, you can see that the production or consumption bandwidth percentage of the cluster instance does not exceed 100%, but the maximum production or consumption bandwidth of the instance has already exceeded the purchased cluster bandwidth. Meanwhile, the cluster throttling count is not 0. This symptom occurs for the following two reasons:

1. The current cluster uses a throttling mechanism that adopts a soft delayed packet response for high-volume scenarios like Kafka. In extreme cases, the maximum production/consumption bandwidth of an instance may exceed the purchased cluster bandwidth. For details, refer to Throttling Mechanism Description.
2. The bandwidth percentage metric uses a peak-shaving and valley-filling way to present the current cluster utilization ratio. Our calculation method obtains the average per unit time, which leads to the above problem.

Maximum Production/Consumption Bandwidth Exceeds Purchased Cluster Bandwidth but No Throttling Occurs

The current cluster's throttling mechanism employs a relatively soft delayed packet response for high-volume scenarios like Kafka. In extreme cases, the instance's maximum production/consumption bandwidth may exceed the purchased cluster bandwidth. For details, please refer to Throttling Mechanism Description.

Does the Instance Support Data Compression

Currently CKafka supports open source snappy and lz4 message compression formats. Gzip compression is not yet supported due to its high CPU consumption. Enabling compression may increase server pressure in some cases, such as when the producer version is high and the consumer protocol edition is low, leading to protocol conversion during consumption. This can cause higher server pressure. So if necessary for testing, it is recommended that customers turn off message compression parameters for testing.
Method to enable configuration: Set compression.type = snappy or lz4 in the Producer configuration file. The default setting is none.
It is recommended to select Snappy compression first. After enabling compression, it may lead to an increase in server CPU and higher production and consumption duration. Use with caution.

Topic-Related

Last updated: 2025-09-28 16:18:04

How do I choose an appropriate number of CKafka replicas?

We recommend that you select two or three replicas for data storage when creating a topic to ensure data reliability. A created topic has two replicas by default. If your business requires higher availability, you can select three replicas. If you need more replicas, you can submit a ticket for assistance.
To improve data security, CKafka does not recommend creating single-replica topics. If you have a single-replica topic in your account, please migrate it as follows:

1. Create a topic, select the same partition parameter, and select "dual-replica".
2. Produce messages in the new topic while the existing single-replica topic continues to be consumed.
3. Modify the consumer configuration after consumption is completed to subscribe to the new topic for consumption.

Why can messages still be queried after the message retention period set for the topic elapsed?

1. A timestamp field and timestamp types are added to messages. Currently, two timestamp types are supported: `CreateTime` and `LogAppendTime`. The former indicates the time when the message is created by the producer, and the latter indicates the time when the broker receives the message. If the timestamp data of the time when a client produces a message is invalid, data deletion on the broker server will be affected.
2. If there are too many partitions in the topic and too little message data, and only one log segment file exists in the partition, messages will not be deleted.
3. The log deletion task checks whether the current log size exceeds the set threshold, i.e., 1 GB per segment. If the maximum timestamp data in the log segment is still within the retention period, messages will not be deleted.

Why is the number of topics (total number of partitions) limited?

A high number of topics (total number of partitions) in Kafka will compromise the cluster performance and stability.
As the number of partitions that a server can sustain is limited, if you want more partitions, you need to add more nodes, which incur more fees. Kafka's storage and coordination mechanisms work by partition. If there are too many partitions, storage fragmentation will be severe, random writes on individual servers will increase, the efficiency of leader switch in the cluster will decrease, controller node failover will slow down, and other problems may occur, which lower the cluster performance and stability.

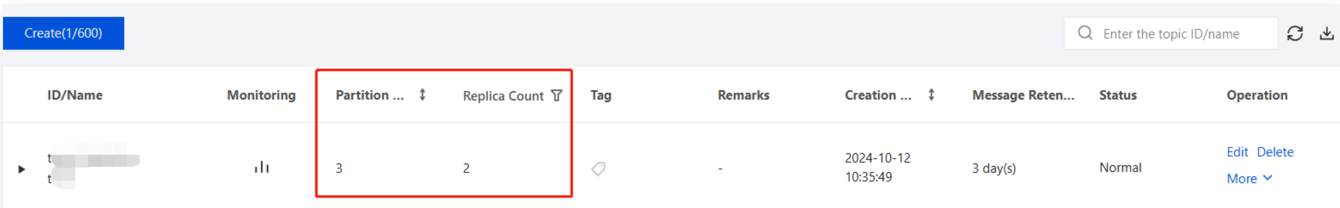
What is the relationship between the number of topics and the number of partitions?

CKafka uses partition as an allocation unit.
Total number of partitions = topic A *number of replicas + topic B* number of replicas + ... + topic N * number of replicas.
Replicas are also counted into the total number of partitions. For example, if you create 1 topic with 6 partitions and 2 replicas for each partition, then you have a total of 12 partitions (1 *6* 2).

- Partition count: it is a concept in physical partition, where one topic can contain one or more partitions.
- Replica Count: The number of partition replicas is used to ensure the high availability of the partition. To ensure data reliability, creating a single-replica topic is not supported. Two replicas are enabled by default.

What to Do If Topic Creation Fails

- **The sum of partition numbers of all created Topics has reached the maximum number of partitions for the instance specification.**
  It is advisable to scale out the Kafka instance or delete unnecessary Topics.



- **Create a Topic with the same name during the delete Topic period**
  Topic deletion is an asynchronous operation. After issuing the delete command, the system will asynchronously remove the Topic metadata. If you create a topic with the same name during this period, the system will notify you that the Topic already exists. Please try again later at that time. Here, you have to wait about 1 minute before proceeding with the reconstruction operation.
- **Topic already exists in the cluster**
  Topic already exists in the cluster. Creating a topic with the same name will prompt that the Topic already exists. It is recommended to reset the topic name.

Consumer Group-Related

Last updated：2025-09-28 16:18:29

How to Set a Reasonable Consumer Count

The relationship between consumer group and consumer is as follows:
- A consumer can subscribe to multiple topics at the same time.
- A topic can contain one or multiple partitions.
- A partition can be consumed only by one consumer at a time.

Therefore, the maximum number of consumers in a consumer group is the total number of partitions in all topics.
Consumer refers to a consumer object in programming. There can be multiple consumers on a server. This is because a thread contains a consumer, and if multiple threads are initiated, there will be multiple consumers as shown below:

```
Properties props = new Properties();
props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrap);
KafkaConsumer<String, String> consumer = new KafkaConsumer<>
(props);
consumer.subscribe(Arrays.asList(topic));
while (true) {
            ConsumerRecords<String, String> records =
consumer.poll(100);
}
```

You can configure the initial number of consumers based on the client resources and configure heap alarms for consumer groups. If messages are heaped, you can add more consumers. For more information on how to configure an alarm, see Configuring Alarm.

Why You Will Not See Consumer Detail

Consumer detail includes the Topic consumed by the consumer, partition progress, client information, etc. This information is stored by the server after the consumer processes and submits the offset. The client will not automatically submit the offset storage request to the server in the following scenarios:
1. Use the custom partition consumption mode.
2. Using Flink + CKafka for big data scenarios, the Flink cluster does not submit offsets or perform partition rebalance through the coordinator, so the server has no consumption status or pod IP information.

If the above scenarios are excluded, refer to Consumer Group list details missing to perform troubleshooting.

Consumer Group List Details Missing What to Do

Symptom Overview

The CKafka consumer group list shows group names, but no consumption details when opened. For example: the CR consumer group in the figure below displays no details.

Possible Causes

Kafka supports two data consumption modes: consumer group mode and custom partition consumption mode.

- When using consumer group mode, the client coordinates consumption through the consumption coordinator. Upon completion, it submits a storage request for the offset to the server. At this point, the server stores information such as the consumed Topic, partition progress, and client details.
- When using custom partition consumption mode, the client does not automatically submit offset storage requests to the server. If the client does not actively initiate an offset submission request, the server will not see the relevant consumption information.
- After a Topic has ACL set up, specific instances may be unable to view the consumer group details. If you cannot view the details, please first check whether there is an ACL. If there is, you are required to submit a ticket for processing.

Solution

1. View the consumer group list of an instance.

```
]$ bin/kafka-consumer-groups.sh --bootstrap-server
9.146.153.249:9092 --list
CR
```

   See current ALL consumer group names.
2. View the specific details of a consumer group for an instance.

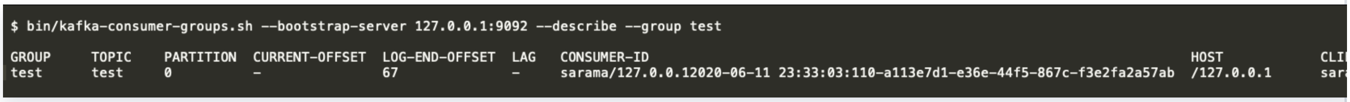
```
]$ bin/kafka-consumer-groups.sh --bootstrap-server
9.146.153.249:9092 --describe --group CR
Note: This will not show information about old Zookeeper-based
consumers.
```

You will find no details for this consumer group. This indicates the consumer client does not use the consumerGroup mechanism to consume data. The client does not submit consumption details to the server, and the server does not store consumption data, so it will not display normally.

3. Locate whether the issue is on the server side.
4. Using the built-in consumer group command, specify the consumer group name test1 to consume, as follows:

```
]$ bin/kafka-console-consumer.sh --bootstrap-server
127.0.0.1:9092 --from-beginning --topic test --group test1
```

For consumer groups that display normally on the console, you can view details using the `--describe` command, as shown below:



Subscribe Type Consumer Group and Assign Type Consumer Group Can Reuse the Same Group Id

In CKafka, the management methods for subscribe-type and assign-type consumer groups are entirely different. The information of subscribe-type consumer groups is fully managed by the CKafka server, including consumer meta-information, submitted offsets, and rebalance management. In contrast, for assign-type consumer groups, the logic of partition assignment for each consumer is entirely managed by the client itself, with the server only storing the offsets submitted by the assign-type consumer group. Therefore, if subscribe-type and assign-type consumer groups reuse the same group id, some unexpected behaviors may occur as follows:

1. If a subscribe-type consumer group is in stable status and consuming, then starting an assign-type consumer with the same group id may cause the assign-type consumer's offset submission to fail.
2. If an assign-type consumer group is consuming and submitting offsets, then starting a subscribe-type consumer group with the same group id may cause the subscribe-type group's submitted offsets to overwrite those of the assign-type group, leading to offset confusion and submission failure for the assign-type consumer group.
3. If a subscribe-type consumer group stops consumption after use and its status is empty, then starting an assign-type consumer with the same group id may allow normal consumption and offset submission. However, since the server stores this group id as a subscribe type, including its consumer meta-information, when the meta-information expires (default: 7 days), the consumer group will be directly deleted via subscribe-type expiration cleanup. This can cause

synchronous deletion of the group's offsets, and the ongoing assign-type consumer group may encounter issues due to offset loss.

Therefore, we should avoid this improper usage. If you must use the same group id name while the consumer group types differ before and after, you need to first delete the consumer group in the console and then enable a new type of consumer group for consumption.

What to Do If Consumer Group Persistently Occurs in PreparingRebalance State

Possible Causes

1. A new consumer joins the consumer group.
2. When a running consumer stops functioning and leaves the consumer group. Common situations include consumer restart, consumer application crash, or heartbeat timeout reported by the consumer process (For details, refer to CKafka Common Parameter Configuration Guide).
3. When the number of partitions changes (add or delete).

Solution

Two situations, 1 and 3, cannot avoid rebalance. Under normal conditions, rebalance completes within 30s. If a long-term rebalance occurs, submit a ticket to handle it.
If the issue is caused by heartbeat timeout or two times poll interval too large, you can adjust the following parameters:

```
# When using the Kafka consumer grouping mechanism, the consumer timeout period. When the Broker does not receive a heartbeat from the consumer within this time, the consumer is considered to have failed, and the Broker initiates the Rebalance process. Currently, this value must be set in the Broker configuration between group.min.session.timeout.ms=6000 and group.max.session.timeout.ms=300000.
session.timeout.ms=10000

# When using the Kafka consumer grouping mechanism, the interval for consumers to send heartbeat. This value must be less than session.timeout.ms, normally less than one-third of it.
heartbeat.interval.ms=3000

# When using the Kafka consumer grouping mechanism, the maximum interval for subsequent calls to poll. If poll is not called again within this time, the consumer is considered to have
```



```
failed, and the Broker will initiate Rebalance again to assign
its partitions to other consumers.
max.poll.interval.ms=300000
```

If a consumer subscribes to many topics, you can try to reduce the number of topics subscribed by the Group.

Client-Related

Client use problem

Last updated：2025-09-28 16:03:23

What should I do if no messages are displayed in the Kafka console when I test the client?

- If the "latest" option is used, the consumer can only get the last messages, and production needs to be ongoing so that the messages can be seen.
- Change to the "earliest" option for data consumption.

What should I do if a production/consumption error occurs after a new client is connected to the service?

- Check whether telnet works. It might be a network issue. Check if Kafka and the producer are in the same network.
- Check whether the accessed “vip - port” is correctly configured.
- Check whether the topic allowlist is enabled. If yes, you need to configure the correct IP for access.

How can I ensure that messages produced by the client are sequential in the same partition?

- If the topic has only one partition, messages will be stored in the order in which they are received by the server. This makes messages sequential in the partition.
- If a topic has multiple partitions, you can specify keys for messages of the same type on the producer. If such messages are sent with the same key, CKafka will select a partition to store them using the hash algorithm according to the key. As a partition can be listened on and consumed by only one consumer, messages will be consumed in the sending order.

A single producer produces messages to a single partition sequentially.

How many connections does a client generally establish to brokers for message production?

For a single client instance (a producer object created with the `new` command), the total number of connections established between it and all servers ranges from one to n (n refers to the number of brokers).

Each Java producer manages TCP connections as follows:

1. The `Sender` thread will be initiated when a `KafkaProducer` instance is created to establish TCP connections to all brokers in `bootstrap.servers`.
2. After the metadata on the `KafkaProducer` instance is updated, TCP connections to all brokers in the cluster will be established again.
3. If no TCP connections are found when a producer sends a message to a broker, a connection will be established immediately.
4. If you set the `connections.max.idle.ms` parameter on the producer to a value above 0, TCP connections established in step 1 will be closed automatically. The parameter value is 9 minutes by default; that is, if no requests are sent through a TCP connection in 9 minutes, Kafka will automatically close the connection. If you set the parameter to –1, TCP connections established in step 1 cannot be closed and will become "zombie" connections.

How do I know whether a message is successfully sent from a client?

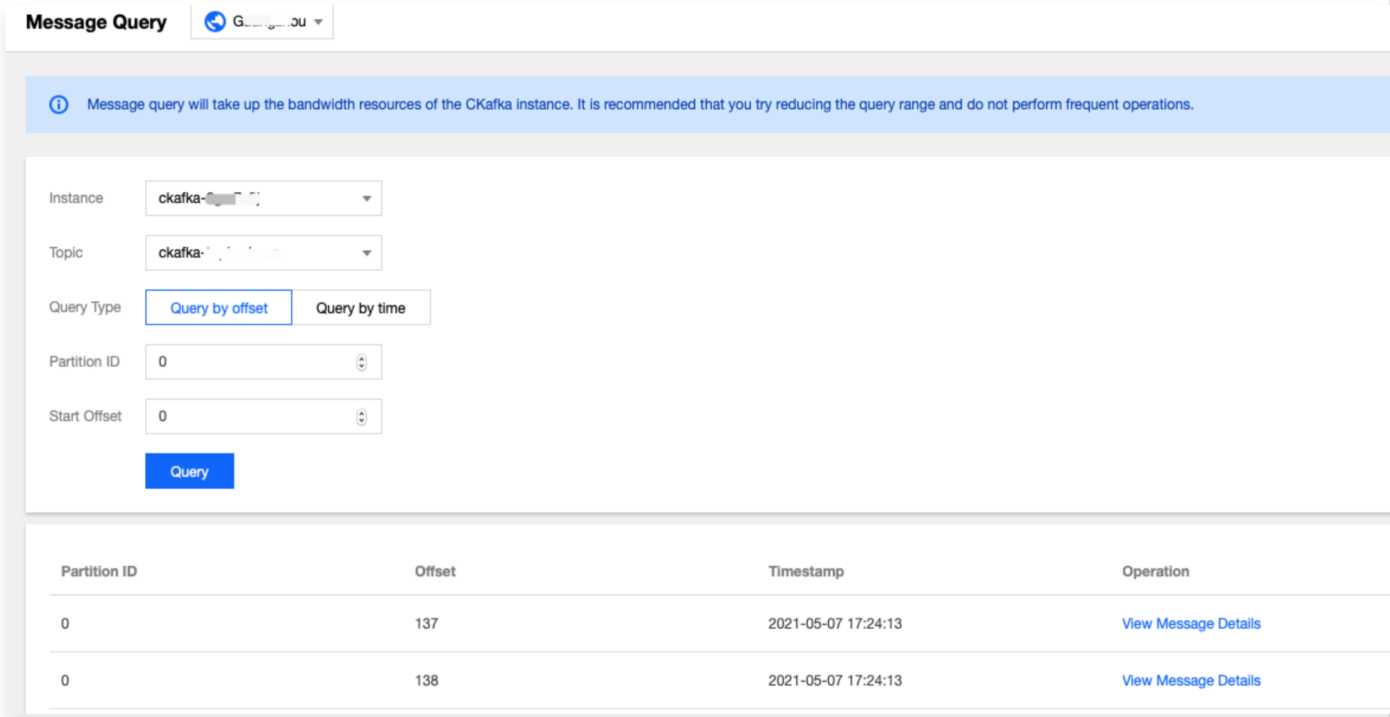
After sending a message, most clients will return a `Callback` or `Future`. A successful callback indicates that the message is successfully sent.
You can also check whether a message is successfully sent in the console by the following methods:

- View the topic partition status to see the number of messages in each partition in real time.
- View the topic traffic monitoring data to see the traffic curve of the produced messages.

You can print the partition information returned by the `send` method to check whether the message is successfully sent:

```
Future<RecordMetadata> future = producer.send(new
ProducerRecord<>(topic, messageKey, messageStr));
RecordMetadata recordMetadata = future.get();
log.info("partition: {}", recordMetadata.partition());
log.info("offset: {}", recordMetadata.offset());
```

If the partition and offset information can be printed out, the currently sent message has been correctly saved on the server. At this time, you can use the message query tool to query the information of the relevant offset.
If the partition and offset information cannot be printed out, the message has not been saved on the server, and the client needs to retry.



What is leader switch?

When a topic is created, the Kafka broker cluster specifies a leader for each partition, and the topic partitions are evenly distributed to each broker.
As time elapses, the partitions may become unevenly distributed across brokers, and the client may throw exceptions such as `BrokerNotAvailableError` and `NotLeaderForPartitionError` during the production or consumption process.
Generally, such issues occur because the partition leader has been switched as described in the following scenarios:

- When the broker where a partition leader resides cannot communicate with the broker controller due to some exceptions such as network disconnections, program crashes, and hardware failures,
- **leader switch** will occur, that is, the current topic partition leader will be replaced by a new leader elected from among the follower partitions.
- When the Kafka cluster sets `auto.leader.rebalance.enable = true` to automatically trigger rebalancing or when rebalancing is manually triggered by the increase/decrease of the number of brokers,
- **leader switch** will also occur due to partition rebalancing.

When the leader is switched due to the broker's accidental disconnection:

- If the client has set `ack = all` and `min.insync.replicas > 1`, messages won’t get lost as they have been acknowledged by both the leader and follower partitions.
- If the client has set `ack = 1`, **some messages may get lost** as they may fail to be synced to

the follower partitions within the specified `replica.lag.time.max.ms` .

Messages won't get lost if the broker works normally and the leader is switched due to the rebalancing manually/automatically triggered by instance upgrade, the single- to multi-AZ deployment mode switchover, or instance migration. This is because:

- If the client has set `ack = all` and `min.insync.replicas > 1` , messages won't get lost as they have been acknowledged by both the leader and follower partitions.
- If the client has set `ack = 1` , messages won't get lost as the partition offset will be automatically synced when the leader is switched.

Can a Kafka Client Connect Directly to CKafka Service

You can connect to the CKafka service through mainstream open-source Kafka clients and deploy code in Tencent Cloud service to produce or consume messages.

Kafka Java Client Issue

Last updated：2025-09-28 16:11:42

What Is the Cause of Uneven Producer Partitioner

Before version 2.3, Kafka Java Producer used the polling partition policy by default for sending messages. Starting from version 2.4, it offers optional partition selectors, providing two partition policies: RoundRobinPartitioner and UniformStickyPartitioner. By default, it uses the UniformStickyPartitioner polling policy for partition selection. Among them, RoundRobinPartitioner has an unrealized method, onNewBatch(), which may lead to uneven message distribution where only partial partitions perform sending. To address such issues, a custom partitioner, MyRoundRobinPartitioner, can be considered for use. Just specify this class when initializing the producer. ```
Since the community recommends UniformStickyPartitioner as the default partitioner, but due to the open-source Kafka Java SDK (version >=2.4.1 && <3.3.0), the default sticky partition policy of Kafka Producer has a data skew problem, especially when linger.ms is 0 (messages sent consecutively in a short period of time are all written to the same partition). If customers are sensitive to hotspot data skew, they can consider upgrading the Kafka Java SDK to version 3.3.0 or higher. Kafka Java SDK Producer in version 3.3.0 developed the Strictly Uniform Sticky Partitioner to optimize this. For higher versions: always specify ack, idempotence, and linger.ms when writing.

```
/*
 * Licensed to the Apache Software Foundation (ASF) under one or
more
 * contributor license agreements. See the NOTICE file
distributed with
 * this work for additional information regarding copyright
ownership.
 * The ASF licenses this file to You under the Apache License,
Version 2.0
 * (the "License"); you may not use this file except in
compliance with
 * the License. You may obtain a copy of the License at
 *
 *    http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing,
software
 * distributed under the License is distributed on an "AS IS"
BASIS,
```



```
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express
or implied.
 * See the License for the specific language governing
permissions and
 * limitations under the License.
 */
package org.apache.kafka.clients.producer;

import java.util.List;
import java.util.Map;
import java.util.Queue;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.ConcurrentLinkedQueue;
import java.util.concurrent.ConcurrentMap;
import java.util.concurrent.atomic.AtomicInteger;

import org.apache.kafka.common.Cluster;
import org.apache.kafka.common.PartitionInfo;
import org.apache.kafka.common.utils.Utils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * The "Round-Robin" partitioner - MODIFIED TO WORK PROPERLY WITH
STICKY PARTITIONING (KIP-480)
 * <p>
 * This partitioning strategy can be used when user wants to
distribute the writes to all
 * partitions equally. This is the behaviour regardless of record
key hash.
 */
public class MyRoundRobinPartitioner implements Partitioner {
    private static final Logger LOGGER =
LoggerFactory.getLogger(RoundRobinPartitioner.class);
    private final ConcurrentMap<String, AtomicInteger>
topicCounterMap = new ConcurrentHashMap<>();
    private final ConcurrentMap<String, Queue<Integer>>
topicPartitionQueueMap = new ConcurrentHashMap<>();

    public void configure(Map<String, ?> configs) {
```



```
    }

    /**
     * Compute the partition for the given record.
     *
     * @param topic      The topic name
     * @param key        The key to partition on (or null if no
key)
     * @param keyBytes   serialized key to partition on (or null
if no key)
     * @param value      The value to partition on or null
     * @param valueBytes serialized value to partition on or null
     * @param cluster    The current cluster metadata
     */
    @Override
    public int partition(
        String topic, Object key, byte[] keyBytes, Object value,
byte[] valueBytes, Cluster cluster) {
        Queue<Integer> partitionQueue =
partitionQueueComputeIfAbsent(topic);
        Integer queuedPartition = partitionQueue.poll();
        if (queuedPartition != null) {
            LOGGER.trace("Partition chosen from queue: {}",
queuedPartition);
            return queuedPartition;
        } else {
            List<PartitionInfo> partitions =
cluster.partitionsForTopic(topic);
            int numPartitions = partitions.size();
            int nextValue = nextValue(topic);
            List<PartitionInfo> availablePartitions =
cluster.availablePartitionsForTopic(topic);
            if (!availablePartitions.isEmpty()) {
                int part = Utils.toPositive(nextValue) %
availablePartitions.size();
                int partition =
availablePartitions.get(part).partition();
                LOGGER.trace("Partition chosen: {}", partition);
                return partition;
            } else {
```

```
                // no partitions are available, give a non-
available partition
                return Utils.toPositive(nextValue) %
numPartitions;
            }
        }
    }

    private int nextValue(String topic) {
        AtomicInteger counter =
            topicCounterMap.computeIfAbsent(
                topic,
                k -> {
                    return new AtomicInteger(0);
                });
        return counter.getAndIncrement();
    }

    private Queue<Integer> partitionQueueComputeIfAbsent(String
topic) {
        return topicPartitionQueueMap.computeIfAbsent(topic, k ->
{
            return new ConcurrentLinkedQueue<>();
        });
    }

    public void close() {
    }

    /**
     * Notifies the partitioner a new batch is about to be
created. When using the sticky partitioner,
     * this method can change the chosen sticky partition for the
new batch.
     *
     * @param topic         The topic name
     * @param cluster       The current cluster metadata
     * @param prevPartition The partition previously selected for
the record that triggered a new
     *                      batch
```



```
     */
    @Override
    public void onNewBatch(String topic, Cluster cluster, int
prevPartition) {
        LOGGER.trace("New batch so enqueuing partition {} for
topic {}", prevPartition, topic);
        Queue<Integer> partitionQueue =
partitionQueueComputeIfAbsent(topic);
        partitionQueue.add(prevPartition);
    }
}
```

Why Is Consumer Coordinator Unable to Refresh and Reconnect

For Kafka Java consumer clients (version >=2.6.x && <3.2.1), after a commitOffsetsAsync exception, the connection to the coordinator cannot be restored, leading to subsequent inability to commit offsets. This issue is fixed in version 3.2.1 and later. Clients affected by this problem are advised to either upgrade to 3.2.1 to solve it or roll back to a version <2.6.x.

Why Does Consumer Sticky Assigner Cause Repeat Consumption

For Kafka Java consumer clients with versions <2.3.0, if using the Sticky assignor, partition duplication may occur—where a partition could be simultaneously allocated to different consumers within the same consumption group, leading to duplicate consumption. If using the Sticky assignor, it is recommended to upgrade to version 2.3.0 or higher to avoid this issue.

Go Sarama Client

Last updated：2025-09-28 16:13:08

How to Set Go Sarama Metadata Refresh Parameters

1. The Go Sarama client Metadata update parameter is controlled by Metadata.Full. For ease of design and implementation, it is set to true by default, which updates Metadata for all topics and may impact performance. In scenarios with a large number of topics, this might affect Kafka production and consumption performance. We recommend customers modify this configuration to false.
2. The default Metadata refresh time parameter for the Go Sarama client is controlled by Metadata.RefreshFrequency, with a default value of 10 minutes, which can easily trigger the Broker's default connection idle timeout (10min) and cause connection interruption. It is recommended to adjust it to 5 minutes to maintain parameter consistency with the Kafka java standard sdk.

How to Set Go Sarama Offset Submission Expiration Time

The Go Sarama client CommitOffset request can specify the expiry date of the offset. This parameter is controlled by Retention.Minutes, with a default value of 0, which will cause the offset to expire immediately after submission. It is recommended to set this value consistent with the CKafka offset expiry date, preferably 7 days.

What to Do If Unable to Produce or Consume Using Go Sarama Cluster Client

Currently, Sarama cluster is no longer maintained and has multiple defects. The following known issues exist: the underlying code of the consumer client, PartitionConsumer, unpacks FetchResponse by calling FetchRequest. At this point, the sarama cluster implementation does not follow the standard protocol and has defects. Even if the server actually returns complete metadata, the client may encounter incomplete parsed metadata, leading to production or consumption failures. The error typically appears as: `response did not contain all the expected topic/partition blocks`.

Scenario example:
During a configuration adjustment operation on a customer instance, adding nodes triggers a client-side metadata update. If the metadata returns all partitions in the order test-0, a-0, test-1, a-1, test-2 (3 partitions for test and 2 for a), although the broker returns complete metadata, there is a certain probability of gaps in the sequence. The sarama cluster client may parse incomplete

metadata, such as only resolving a–1 and test–2, causing the reported error where the response does not contain all expected partition metadata, leading to production or consumption failures.

For this situation, clients with standard protocol implementation will not encounter such issues, such as Java or Confluent go. It is recommended that customers check whether they are using this client before scaling, because scaling can frequently trigger metadata updates. If they are, strongly recommend upgrading to the confluent go client before proceeding with scaling. If upgrading is not possible, immediately restarting the client after the issue occurs may restore it to normal status.

Common Client Errors and Solutions

Last updated：2025-09-28 16:17:04

Client Configuration or Service Abnormality

The following exception belongs to client configuration or service abnormality, and the client will not retry automatically.

Abnormal	Description	Analysis and Description
UnknownServerException	An unknown error occurred while the server was processing the request.	Older versions of flow control will return this error; the new version might be due to a server bug.
RecordTooLargeException	File too large.	Current configuration: message.max.bytes=1000012.
InvalidRequiredAcksException	The acks parameter in producer configuration is invalid.	-
InconsistentGroupProtocolException	The Group protocol is inconsistent.	Check whether the Consumer and Connector are configured with the same group.id. They use different protocols and cannot join the same group.
InvalidGroupIdException	The Consumer Group ID is invalid.	Recommend using [a-zA-Z0-9._-] characters with length not exceeding 128.
InvalidTopicException	The Topic is invalid.	After enabling the auto-create Topic option, the client will receive this exception if the Topic is invalid. Check whether the Topic contains invalid characters or exceeds the length limit.
InvalidSessionTimeoutException	The session.timeout.ms in the consumer configuration is invalid.	The current minimum value allowed on the server side is group.min.session.timeout.ms=6000, and the maximum value is group.max.session.timeout.ms=300000.
InvalidCommitOffsetSizeException	Offset submission is too large, exceeding max message size	Current configuration: message.max.bytes=1000012.

	and cannot be written to __consumer_offsets.	
OffsetMetadataTooLarge	The Offset submit request contains metadata that is too large.	Server configuration: offset.metadata.max.bytes=4096.
UnsupportedVersionException	The Broker does not support requests for this version.	It is recommended to use client version 0.10.2.x.

Program Running Normally with Temporary Exception

The following exception may briefly appear during normal program operation, and the client will retry automatically. If it persists, the service is abnormal.

Abnormal	Description	Analysis and Description
TimeoutException	Request timeout.	First-time connection reports request timeout. First check whether the address is correct, then use telnet to confirm network connectivity. This exception occasionally thrown during program execution may belong to network jitter.
CorruptRecordException	The message is invalid.	CRC check may fail due to invalid data size. In addition, if the compression method uses **GZip** or versions below 0.9 **with compression**, it may also lead to this error.
UnknownTopicOrPartitionException	The Topic or Partition does not exist.	Check to console whether the corresponding topic has been created. Note: The client uses TopicName for production consumption rather than TopicId. In addition, the client may report the topic as nonexistent if it has no permission to access it.
LeaderNotAvailableException	Partition has no Leader.	When a Topic is just created and the server has not yet elected a suitable Leader, this error will be returned to the client. The client will automatically retry to retrieve the Leader info. **This issue only occurs in earlier versions and has been removed in 0.10.2.1**.

NotLeaderForPartitionException	The Leader of the Partition is unavailable.	Since the client caches the Topic's Metadata, when the Leader switchover of a Partition occurs, production or consumption requests may still be sent to the old Leader. In this case, this error will be returned to the client, and the client will auto-update the Metadata info.
NetworkException	The client connection is closed by the server side.	Network exception or the number of connections exceeds the limit.
NotEnoughReplicasException	Insufficient ISR count.	When writing data, the number of ISRs in the Partition is less than the min.insync.replicas configured for the Topic, which may be due to ISR fluctuations.
NotEnoughReplicasAfterAppendException	After data is written locally to the Broker, ISR fluctuations may cause the min.insync.replicas requirement to not be met.	-
BrokerNotAvailableError	Leader not found for the partition.	Since the client caches the Topic's Metadata, when the Leader switchover of a Partition occurs, production or consumption requests may still be sent to the old Leader. In this case, this error will be returned to the client. The client will auto-update the Metadata info. After the switch, new production requests sent to the old Leader should automatically adjust to the new Leader after reporting errors, theoretically without affecting the integrity of data write and consumption.
NotLeaderForPartitionError	Leader not found for the partition.	Since the client caches the Topic's Metadata, when the Leader switchover of a Partition occurs, production or consumption requests may still be sent to the old Leader. In this case, this error will be returned to the client, and the client will auto-update the Metadata info. After the switch, new production

		requests sent to the old Leader should automatically adjust to the new Leader after reporting errors, theoretically without affecting the integrity of data write and consumption.

Logging Configuration at DEBUG Level Exception

The following exception may appear when logging configuration is at DEBUG level, and the client will handle it automatically.

Abnormal	Description	Analysis and Description
OffsetOutOfRangeException	The Offset passed in by the consumer when pulling messages exceeds the limit.	If the client sets the Offset reset policy (earliest or latest), the client will reset the Offset based on policies. Otherwise, the user program is required to fix this exception.
GroupLoadInProgressException	The corresponding Coordinator for the ConsumerGroup is loading.	The server side may briefly appear during upgrade, and the client will retry automatically.
GroupCoordinatorNotAvailableException	The Coordinator is unavailable.	The server side may briefly appear during upgrade, and the client will retry automatically.
NotCoordinatorForGroupException	The current node is not the Coordinator of this ConsumerGroup. The Coordinator has migrated to another node.	The server side may briefly appear during upgrade, and the client will retry automatically.
IllegalGenerationException	The ConsumerGroup generation is invalid.	Heartbeat timeout may occur or new consumers may join. The consumer will automatically retry joining the ConsumerGroup.
RebalanceInProgressException	ConsumerGroup rebalance is currently in progress.	Heartbeat timeout may occur or new consumers may join. The consumer will automatically retry joining the ConsumerGroup.

Network-Related

Last updated：2025-09-28 16:19:32

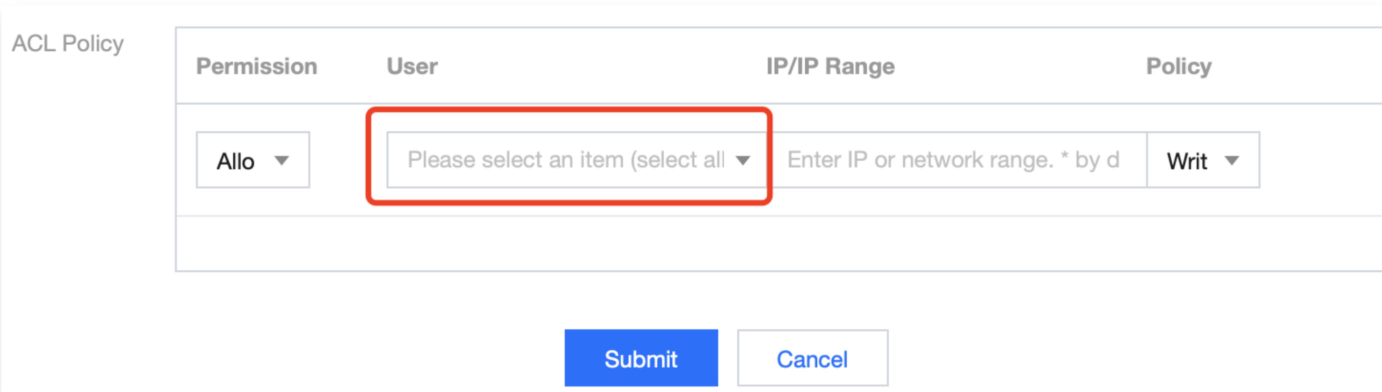
Does CKafka support public network access?

CKafka currently does not support public network access.

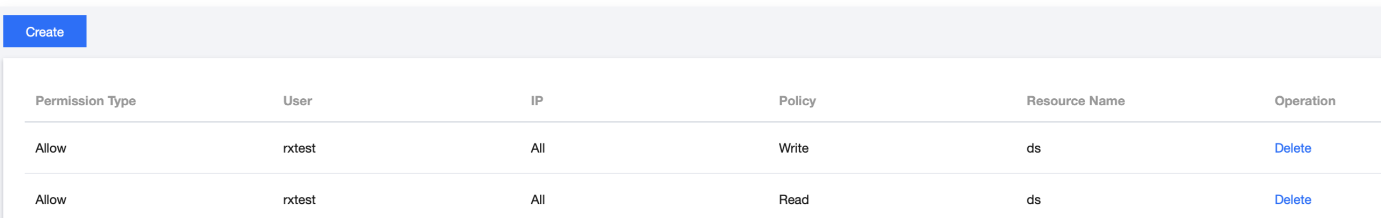
How do I continue accessing CKafka over VPC after SASL is enabled?

If you use the PLAINTEXT method to access CKafka while enabling another SASL access routing, the ACL previously set for the topics will still take effect. If you want PLAINTEXT access to be unaffected, please add the read/write permissions of all users for the topics that PLAINTEXT needs to access.

> **Note:**
> When adding an ACL policy, you don't need to select any user, and read/write permissions are added to **all users** by default.



The effect after addition is as follows:



Create

Permission Type	User	IP	Policy	Resource Name	Operation
Allow	rxtest	All	Write	ds	Delete
Allow	rxtest	All	Read	ds	Delete

Monitoring-Related

Last updated: 2025-09-28 16:19:49

Why No Monitoring Data for Topic

- **The client may not have production consumption**
  Check whether the client has production and consumption behavior. You can test with native production and consumption commands, then view monitoring data. For specific operations, refer to running the Kafka client.
- **The monitoring collection system has a failure**
  If the monitoring collection system has a failure, submit a ticket to fix it.

Why Does Message Queue Depth Persist in Specific Partitions during Monitoring?

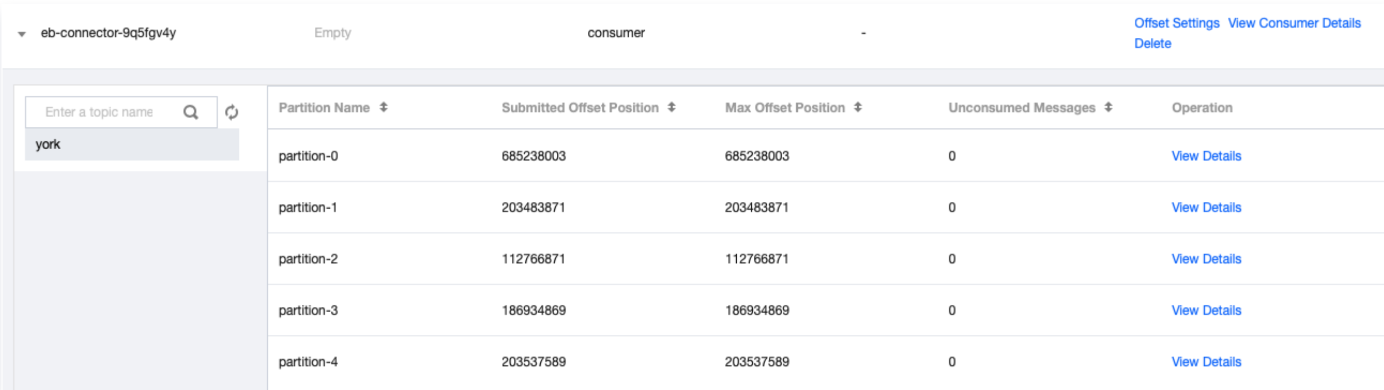
- **The consumer does not consume messages from this partition, while message production continues, leading to persistent aggregated data.**
  If you encounter a situation where specific partitions are not being consumed, you can attempt to resolve the issue by restarting the client. If restarting resolves the problem, first troubleshoot the client log. You can also submit a ticket to request assistance.
- **Insufficient consumer count in the consumer group or consumption speed too slow causes data accumulation.**
  You can try increasing the number of consumers to improve consumption speed.
- **The consumer program may contain bugs, which may lead to not consuming specific partitions.**
  At this point, check the local log at the consumer end to see if there is any exception info.

Message-Related

Last updated: 2025-09-28 16:20:09

How do I calculate the number of unconsumed messages?

Number of unconsumed messages = maximum offset - submitted offset, as shown below:



eb-connector-9q5fgv4y Empty consumer - Offset Settings View Consumer Details Delete

Enter a topic name

york

Partition Name	Submitted Offset Position	Max Offset Position	Unconsumed Messages	Operation
partition-0	685238003	685238003	0	View Details
partition-1	203483871	203483871	0	View Details
partition-2	112766871	112766871	0	View Details
partition-3	186934869	186934869	0	View Details
partition-4	203537589	203537589	0	View Details

img

Can the message retention period be automatically adjusted?

CKafka allows you to add dynamic message retention policies. After such a policy is set, if the disk utilization reaches the specified percentage, a certain proportion of existing data will automatically expire in case of message surges, helping avoid situations where normal production and consumption stop after the disk space is used up. For detailed directions, please see Adding Dynamic Message Retention Policy.

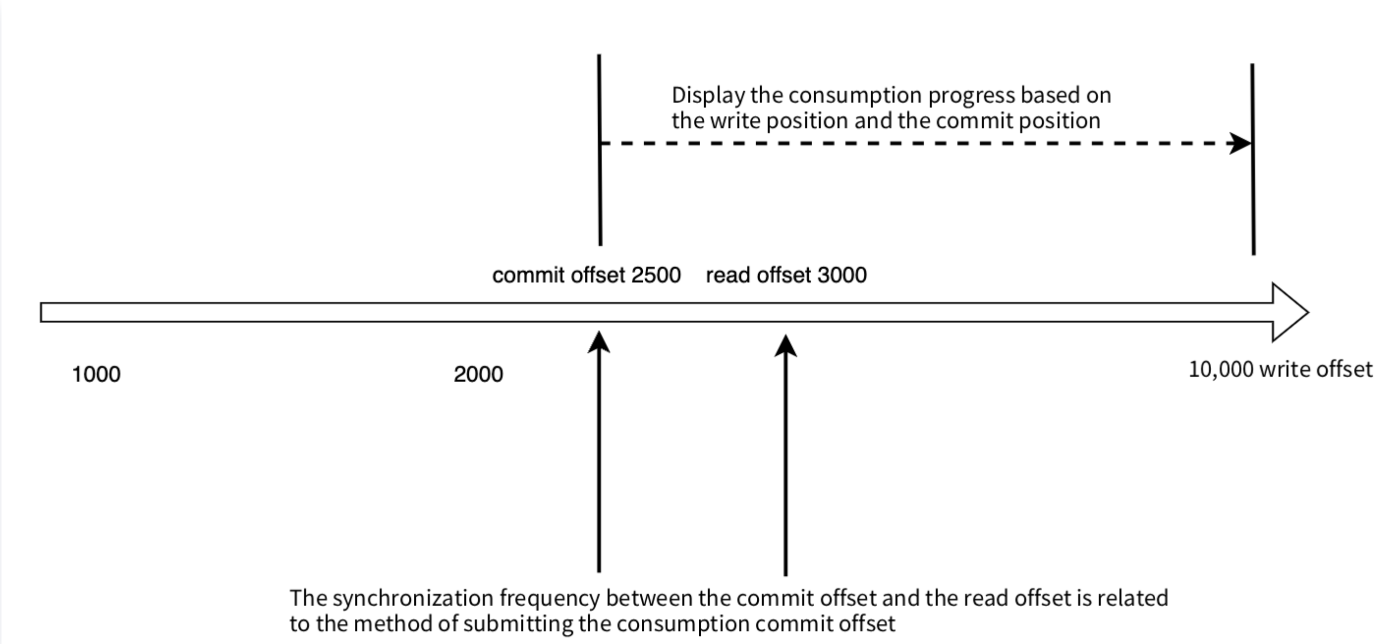
Why Is Consumption Progress Not Synchronized to the Server in Time

Symptom

CKafka consumption progress is not updated on the server in time, and the consumption speed cannot be seen via console.

Pre-Reserve

The CKafka cluster shows consumption progress based on the commit frequency of the client consumption offset to estimate consumption speed and progress. Therefore, prioritize troubleshooting client consumption and offset submission in such problems.

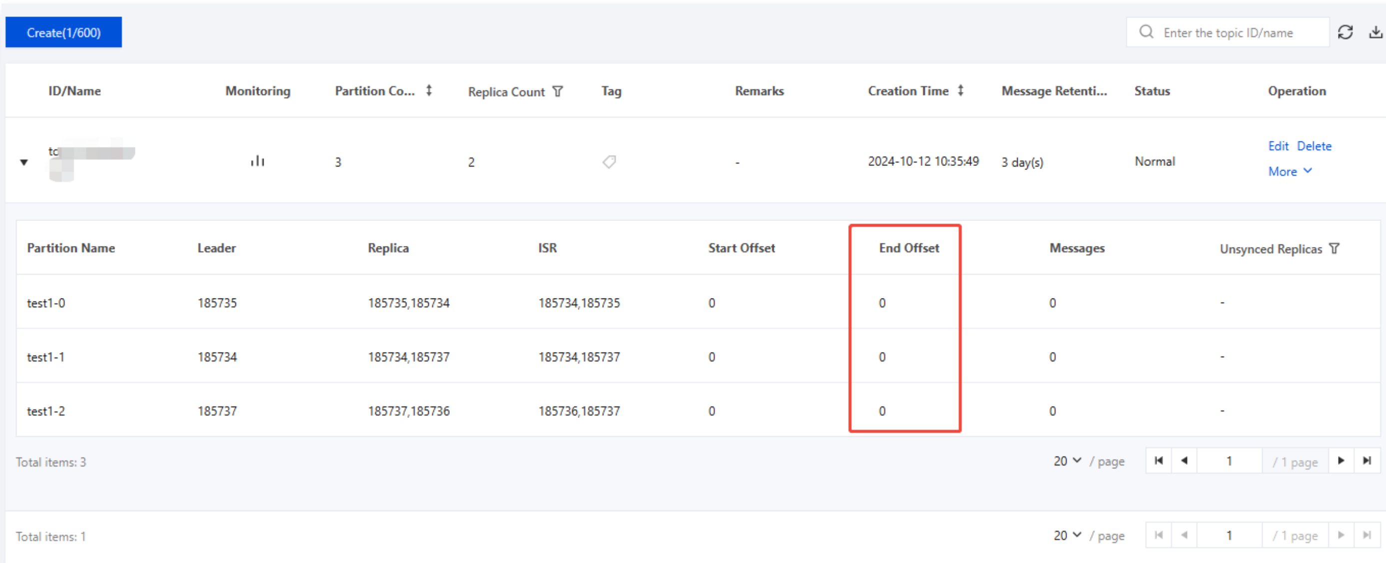


Troubleshooting Guide

Step 1: Check Whether There Is Production Data Written

If no accumulation or consumption speed is detected, check the Topic and production monitoring metrics to see if data is being written.

1. Enter the console, click **instance list** > **ID/Name** > **Topic list**, and check whether the end offset of related partitions updates.



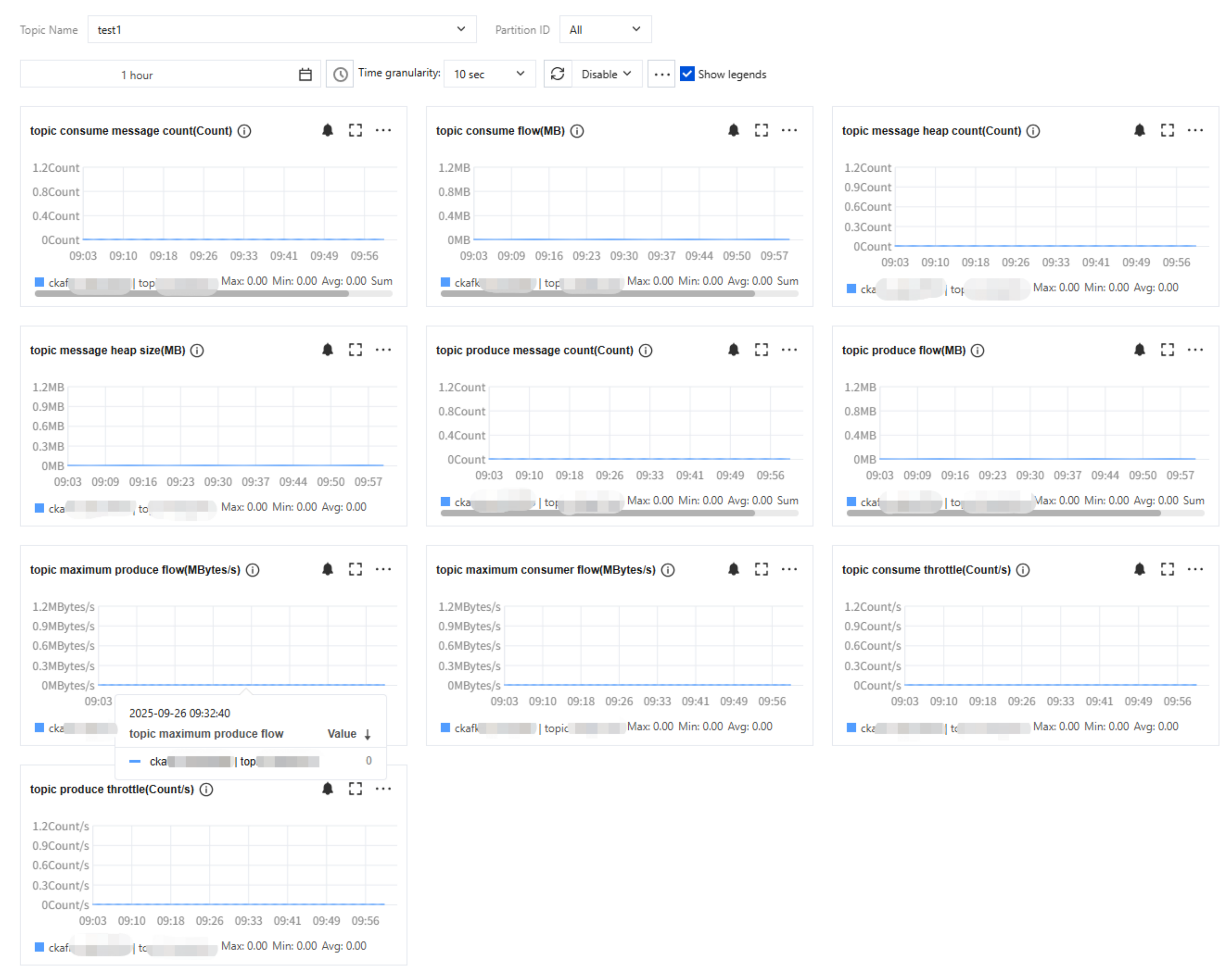
ID/Name	Monitoring	Partition Co...	Replica Count	Tag	Remarks	Creation Time	Message Retenti...	Status	Operation
tc		3	2		-	2024-10-12 10:35:49	3 day(s)	Normal	Edit Delete More

Partition Name	Leader	Replica	ISR	Start Offset	End Offset	Messages	Unsynced Replicas
test1-0	185735	185735,185734	185734,185735	0	0	0	-
test1-1	185734	185734,185737	185734,185737	0	0	0	-
test1-2	185737	185737,185736	185736,185737	0	0	0	-

Total items: 3 20 / page 1 / 1 page

Total items: 1 20 / page 1 / 1 page

2. Enter the console, click **instance list** > **ID/Name** > **monitor > Topic**, and check whether the production flow and total amount of messages occupying disk exceed 0.



If no real-time data is written to the Topic, there are two cases:

- Create a consumer group, set the `auto.offset.reset=latest` behavior to start consuming from the latest. Since there is no real-time data write, it will not generate `commit offset` action;
- Topic data exceeds the retention time. If you set the `auto.offset.reset=earliest` behavior, it will start consuming from the beginning. Since the data has been `retention` , consumption failure occurs, and no `commit offset` action will be generated.

The above scenario does not generate `commit offset` action, so the server will not update the consumption progress.

Step 2: Check Client Configuration

For Kafka native Client, need to check Consumer configuration, whether auto commit `offset: enable.auto.commit` is enabled.

Native Kafka Java Client Example

A direct way is to check the client configuration, or you can search for ConsumerConfig in the program logs, as shown below. This log is only printed during KafkaConsumer initialization and may be unable to find if the log retention time is short.

```
2024-11-21 10:33:25,668 INFO  org.apache.kafka.clients.consumer.ConsumerConfig            [] - ConsumerConfig values:
    allow.auto.create.topics = true
    auto.commit.interval.ms = 5000
    auto.offset.reset = latest
    bootstrap.servers = [redacted]
    check.crcs = true
    client.dns.lookup = use_all_dns_ips
    client.id = [redacted]
    client.rack =
    connections.max.idle.ms = 540000
    default.api.timeout.ms = 60000
    enable.auto.commit = true
    exclude.internal.topics = true
    fetch.max.bytes = 52428800
```

- Based on client consumption features, if auto commit offset can be enabled, you can turn on this parameter, and client consumption progress will be updated to the server;
- If manual offset submission management is required, such as stricter requirements for duplicate consumption or data loss, when `enable.auto.commit` is off, check whether the client program synchronously or asynchronously performs offset submission actions during consumption, for example consumer.commitSync()

```
while (true) {
            ConsumerRecords<String, String> consumerRecords =
consumer.poll(500);
            for (ConsumerRecord record : consumerRecords) {
                System.out.printf("receive message: partition:%d,
offset:%d, time:%d, key:%s, value:%s\n", record.partition(),
record.offset(), record.timestamp(), record.key(),
record.value());
            }
            consumer.commitSync();
        }
```

After troubleshooting and adjustment with the Native Kafka Client, observe again whether Client consumption progress is updated to the server.

Targeting Kafka Streaming Scenarios with Relatively High Flink Occurrence Frequency

For the Flink consumer client, its underlying layer integrates with Kafka through the Flink Kafka Source API, combining with the Flink framework's fault tolerance mechanism, such as checkpoint and savepoint mechanisms. Here, two cases arise: Kafka Client's underlying enable.auto.commit handling and Flink checkpoint's offset processing.

- Kafka Source submits the current consumer offset upon checkpoint completion to ensure Flink checkpoint status matches the commit offset on the Kafka broker.
- If checkpoint is not enabled, Kafka Source depends on the Kafka consumer's internal scheduled auto-commit logic for offsets. The autocommit feature is configured by the two Kafka consumer configuration items: enable.auto.commit and auto.commit.interval.ms. Notably,

If the user does not manually specify enable.auto.commit in the configuration when using Flink Kafka Source, the Flink framework will override enable.auto.commit=false, disabling auto-commit of offsets. If Flink checkpoint is not enabled, the user is required to manually enable this parameter.

Expired Messages Not Promptly Deleted What Is the Cause

Cause Analysis

The message deletion mechanism in Kafka can cause expired messages to not be promptly deleted in specific business scenarios. Lack of understanding of the mechanism may lead to doubt. For example, a significant gap exists between message timestamps in partition 0 and partition 7, with expired messages in partition 0 not being deleted in time.

Kafka Message Deletion Mechanism

Kafka data storage is physically stored on disk in three dimensions: Topic, partition, and data segment. Message data deletion conditions are as follows:

- Message data is deleted based on retention time, with deletion performed at the data segment level.
- Each data segment is set to 1GB in size. When it reaches 1GB, a new segment is generated by rolling.
- All messages in a data segment must expire before the segment is deleted.
- If a message in the data segment is within the retention time, such as the last line of the segment file, this segment file will not be deleted.

Due to some reasons, message write is skewed, and data ingestion is concentrated in specific partitions, such as partition 7, while data for some partitions is seldom, such as partition 0. At this point, the size of the data segment in partition 0 does not reach 1GB, so no scrolling occurs. However, since there is data within the retention time in the entire segment, messages in partition 0 will not be deleted.