# Message Queue CKafka

# Related Documents

# Product Documentation

# Contents

Related Documents
- CKafka Data Reliability Description
- Configuration Guide for Common Parameters in CKafka
- Descriptions of Common Exceptions with CKafka Client

# Related Documents
# CKafka Data Reliability Description

Last updated : 2020-06-30 14:06:34

Existing technologies cannot guarantee that data will never be lost, but you can maximize the reliability of data as possible as you can by using certain configuration parameters.
This document describes the factors that affect the reliability of CKafka from the perspectives of the producer, the service (CKafka), and the consumer, respectively, and provides corresponding solutions.

## Producer

### Causes of Data Loss

When the producer sends data to CKafka, the data may be lost due to network jitters, and CKafka will not receive the data. This may be caused by following reasons:

- The network load is high or the disk is busy, and the producer does not have a retry mechanism.
- The purchased disk capacity is exceeded. For example, if the disk capacity of an instance is 9,000 GB, and it is not expanded promptly after being used up, data cannot be written to CKafka.
- Sudden or continuously increased peak traffic exceeds the purchased peak throughput. For example, if the peak throughput of the instance is 100 MB/s, but it is not scaled up promptly after the peak throughput is exceeded for a long period of time, data writes to CKafka will become slower. In this case, if the producer has a queuing timeout mechanism in place, data cannot be written to CKafka.

### Solutions

- Enable the retry mechanism on the producer for important data.
- For disk usage, set up monitors and alarm policies as preventive measures when configuring the instance.
  When the disk capacity is used up, upgrade the instance timely in the console. Upgrading non-exclusive CKafka instances will not interrupt the service, which also supports expanding the disk capacity alone. You can also shorten the message retention period to reduce disk usage.
- In order to minimize the loss of messages at the producer side, you can fine-tune the size of the buffer using `buffer.memory` and `batch.size` (in bytes). For the buffer, bigger is not always better. If the producer is down for some reason, the more data in the buffer, the more garbage to be recycled, and the slower the recovery. **Pay close attention to the number of messages**

**produced by the producer and average message size** (through the rich set of monitoring metrics available in CKafka).

- Configure ACK for the producer.
  When the producer sends data to the leader, the data reliability level can be set using the `request.required.acks` and `min.isync.replicas` parameters.
  - When acks = 1 (default value), the producer's leader in ISR has successfully received a data entry, and the next data entry can be sent. If the leader is down, as the data may have not been synced to its followers, the data will be lost.
  - When acks = 0, the producer sends the next message without waiting for acknowledgment from the broker. In this case, data transfer efficiency is the highest, but data reliability is the lowest.
  - When acks = -1 or all, the producer needs to wait for acknowledgment of message receipt from all the followers in ISR before sending the next message, which leads to highest reliability.
    Even if ACK is configured as above, there is no guarantee that data will never be lost. For example, when there is only one leader in ISR (the number of members in ISR may increase or decrease in certain circumstances, and in some cases, only one leader is left), `acks` will be 1. Therefore, you also need to use the `min.insync.replicas` parameter (which can be configured in CKafka Console > Topic Management > Advanced Configuration). It represents the minimum number of replicas in ISR, which is 1 by default and takes effect when and only when acks = -1 or all.

### Recommended Parameter Values

These parameter values are for reference only, and actual values depend on the actual conditions of your business.

- Retry mechanism: `message.send.max.retries=3;retry.backoff.ms=10000;`
- Guarantee for high reliability: `request.required.acks=-1;min.insync.replicas=2;`
- Guarantee for high performance: `request.required.acks=0;`
- Reliability + performance: `request.required.acks=1;`

## Service (CKafka)

### Causes of Data Loss

- The partition's leader is down before the backup of followers is completed. Even if a new leader is elected, data will be lost because it has not been backed up yet.
- Open-source Kafka features async storage to disk, that is, data is first stored in PageCache. If the broker breaks, restarts, or fails, the data stored in PageCache will be lost because it has not been stored to the disk yet.

- Stored data may be lost due to disk failure.

### Solutions

- Open-source Kafka has a multi-replica feature. You are recommended to use replicas to ensure data integrity. Data will be lost only if multiple replicas and multiple brokers fail at the same time, so data reliability is higher than that in the single-replica case. Therefore, CKafka requires at least 2 replicas for a topic and supports configuring 3 replicas.
- More reasonable values are configured for the `log.flush.interval.messages` and `log.flush.interval.ms` parameters in CKafka for data flushing.
- In CKafka, the disk is specially designed to ensure that data reliability will not be compromised even if the disk is partially damaged.

### Recommended Parameter Values

A replica which is not in syncing status can be elected as a leader:
`unclean.leader.election.enable=false // Disabled`

## Consumer

### Causes of Data Loss

- The offset is committed before data is consumed. If the consumer is down in the process but the offset has been updated, the consumer will miss a data entry, and the consumer group will have to reset the offset in order to retrieve it.
- The consumption speed differs significantly from the production speed, but the message retention period is too short; therefore, the message will be deleted upon expiration before it is consumed.

### Solutions

- Configure the `auto.commit.enable` parameter appropriately. When it is `true`, commit is performed automatically. You are recommended to use scheduled commit so as to avoid frequent commit offset.
- Monitor the consumer and correctly adjust the data retention period. Monitor the consumption offset and the number of messages that are not consumed, and configure an alarm to prevent messages from being deleted upon expiration due to slow consumption.

# Configuration Guide for Common Parameters in CKafka

Last updated : 2020-07-08 14:30:56

## Broker Configuration Parameter Description

The following are the configurations of a CKafka Broker for your reference:

```
# Maximum message length in bytes.
message.max.bytes=1000012

# Whether to allow automatic creation of topics. The default value is false. Currently, topics ca
n be created in the console or through the Tencent Cloud API.
auto.create.topics.enable=false

# Whether to allow topic deletion by calling the API.
delete.topic.enable=true

# The maximum request length allowed for a Broker is 16 MB.
socket.request.max.bytes=16777216

# Each IP can establish up to 5,000 connections with a Broker.
max.connections.per.ip=5000

# Offset retention period. The default value is 7 days.
offsets.retention.minutes=10080

# Everyone is allowed to access when there is no ACL configuration.
allow.everyone.if.no.acl.found=true

# The log segment size is 1 GB.
log.segment.bytes=1073741824

# The log rolling check interval is 5 minutes. If the retention period is set to less than 5 minu
tes, it may still take 5 minutes to clear logs.
log.retention.check.interval.ms=300000
```

For configurations not listed here, see the open-source Kafka default configurations.

# Topic Configuration Parameter Description

**1. Number of partitions**

From the producer's point of view, writes to different partitions are completely in parallel; from the consumer's point of view, the number of concurrencies depends entirely on the number of partitions (if there are more consumers than partitions, there will definitely be idle consumers). It is important to select an appropriate number of partitions to fully play the performance of the CKafka instance. The number of partitions should be determined based on the throughput of production and consumption, ideally through the following formula:

```
Num = max(T/PT, T/CT) = T/min(PT, CT)
```

Num represents the number of partitions, T the target throughput, PT the maximum production throughput by the producer to a single partition, and CT the maximum consumption throughput by the consumer from a single partition. The number of partitions is equal to T/PT or T/CT, whichever is larger.

In practice, the actual PT is determined by batch size, compression algorithm, acknowledgement mechanism, number of replicas, and so on, while the actual CT is subject to business logic, which varies according to the actual conditions.

**We recommend that the number of partitions be greater than or equal to that of consumers to achieve maximum concurrency. For example, if there are 5 consumers, there should be 5 or more partitions.** However, having too many partitions will lower production throughput and increase time consumed by elections and thus need to be avoided. See the following for reference:

- A single partition can implement sequential writes of messages.
- A single partition can only be consumed by a single consumer process in the same consumer group.
- A single consumer process can consume multiple partitions simultaneously, so partition limits the concurrency of consumers.
- The more partitions there are, the longer it takes to elect a leader upon failure.
- Offset can be down to the partition level. The more partitions there are, the more time the offset query consumes.
- The number of partitions can be dynamically increased but not reduced. However, an increase will result in message rebalance.

### 2. Number of replicas

At present, the number of replicas must be at least 2 to ensure availability. To ensure high reliability, we recommend maintaining at least 3 replicas.

> The number of replicas will affect the production/consumption traffic; for example, if there are 3 replicas, the actual traffic will be 3 times the production traffic.

### 3. Log retention period

The `log.retention.ms` configuration of a topic is set through the retention period of the instance in the console.

### 4. Other topic-level configurations

```
# Maximum message length at the topic level.
max.message.bytes=1000012

# Messages in the 0.10.2 version are in the V1 format.
message.format.version=0.10.2-IV0

# Replica not in ISR can be selected as a leader; in this case, availability is higher than relia
bility, and there is a risk of data loss.
unclean.leader.election.enable=true

# Minimum number of replicas for producer requests submitted by ISR. If the number of replicas in
synchronization is below this value, the server will no longer accept write requests where reques
t.required.acks is set to -1 or all.
min.insync.replicas=1
```

## Producer Configuration Guide

The following describes common parameter settings for the Producer client. We recommend adjusting them based on your actual business scenarios:

```
# The producer will attempt to bundle and send the messages sent to the same partition by the bus
iness to the Broker. The batch.size parameter sets the maximum size of the bundle, which is 16 KB
by default. If batch.size is too small, the throughput will drop; if it is too large, too much me
mory will be used.
batch.size=16384

# The following describes the 3 ACK mechanisms supported by a Kafka producer:
```

```
# -1 or all: the Broker responds to the producer and continues to send the next message or next batch of messages only after the leader receives the data and syncs it to followers in all ISRs. This configuration provides the highest data reliability, as messages will never be lost as long as one synced replica survives. Note: this configuration cannot ensure that replicas will be returned after the data is written to all of them. It can be used together with the min.insync.replicas parameter at the topic level.
# 0: the producer continues to send the next message or next batch of messages without waiting for acknowledgement of synchronization completion from the Broker. This configuration provides the highest production performance but lowest data reliability (data loss may occur when the server fails. If the leader is dead but the producer is unaware of that, the Broker cannot receive messages).
# 1: the producer sends the next message or next batch of messages after it receives an acknowledgement that the leader has successfully received the data. This configuration is a balance between production throughput and data reliability (messages may be lost if the leader is dead but has not been replicated yet).

# If users do not configure this, the default value will be 1. Users can customize this according to their business conditions.
acks=1

# Control the maximum time a production request waits in the Broker for replica synchronization to meet conditions set through acks.
timeout.ms=30000

# Configure the memory that the producer uses to cache messages to be sent to the Broker. Users must adjust this according to the total memory size of the process where the producer resides.
buffer.memory=33554432

# If messages are produced faster than they are sent by the sender thread to the Broker, the memory configured by buffer.memory will be used up and thus block the send operation by the producer. This parameter sets the maximum blocking time.
max.block.ms=60000

# Set the time to send the scheduled message, so that more messages can be sent in batches. This parameter is 0 by default, indicating immediate send. When the messages to be sent reach the size set by batch.size, the request will be sent immediately regardless of whether the time set by linger.ms is reached.
linger.ms=0

# Maximum size of the request packet that the producer can send, which defaults to 1 MB. Note: this value cannot exceed the maximum packet size of 16 MB configured for the Broker.
max.request.size=1048576

# Compression format configuration. Currently, version 0.9 and earlier do not support compression, and version 0.10 and later do not support gzip compression.
compression.type=[none, snappy, lz4]

# Timeout period for the client to send a request to the Broker, which cannot be smaller than the
```

```
value of replica.lag.time.max.ms configured for the Broker. The current value is 10,000 ms.
request.timeout.ms=30000

# Maximum number of unacknowledged requests that the client can send on each connection. If this parameter is greater than 1 and retries is greater than 0, data may be out of order. If strict ordering is necessary, set this value to 1.
max.in.flight.requests.per.connection=5


# Number of retries upon request error. It is recommended that you set the parameter to a value greater than 0 to enable retries and ensure to the greatest extent that messages do not get lost.
retries=0

# Retry interval upon request failure.
retry.backoff.ms=100
```

## Consumer Configuration Guide

The following describes common parameter settings for the Consumer client. We recommend adjusting them based on your actual business scenarios:

```
# Whether to sync the offset to the Broker after a message is consumed, so the latest offset can be obtained from the Broker when the consumer fails.
auto.commit.enable=true

# Interval for the automatic submission of offset when auto.commit.enable=true is configured. We recommend setting this value to at least 1,000.
auto.commit.interval.ms=5000

# Mode to initialize the offset when no offset is configured for the Broker (such as upon initial consumption or when the offset expired for more than 7 days), or mode to reset the offset when error OFFSET_OUT_OF_RANGE occurs.
# earliest: reset to the minimum offset in the partition.
# latest: reset to the maximum offset in the partition. This is the default value.
# none: throw an OffsetOutOfRangeException exception without resetting the offset.
auto.offset.reset=latest

# Identify the consumer group to which the consumer belongs.
group.id=""

# Consumer timeout period when the Kafka consumer groups are used. If the Broker does not receive the heartbeat of the consumer within this period, the consumer will be considered to have failed and the Broker will initiate rebalance. Currently, this value must be configured in the Broker between 6000 (value of group.min.session.timeout.ms) and 300000 (value of group.max.session.timeou
```

```
t.ms).
session.timeout.ms=10000

# Interval at which the consumer sends a heartbeat when the Kafka consumer groups are used. This value must be smaller than the session.timeout.ms value, and is recommended to be smaller than one third of it.
heartbeat.interval.ms=3000

# Maximum interval allowed for calling the poll again when the Kafka consumer groups are used. If poll is not called within this time period, the consumer will be considered to have failed and the Broker will re-initiate rebalance to assign the partitions to other consumers.
max.poll.interval.ms=300000

# Minimum data size returned by a fetch request. The default value is 1 B, indicating that the request can be returned as soon as possible. Increasing this value will increase throughput and latency.
fetch.min.bytes=1

# Maximum data size returned by a fetch request. The default value is 50 MB.
fetch.max.bytes=52428800

# Fetch request wait time.
fetch.max.wait.ms=500

# Maximum data size returned by each partition in a fetch request. The default value is 10 MB.
max.partition.fetch.bytes=1048576

# Number of records returned in one poll call.
max.poll.records=500

# Client request timeout period. If no response is received after this time elapses, the request will time out and fail.
request.timeout.ms=305000
```

# Descriptions of Common Exceptions with CKafka Client

Last updated : 2020-07-08 14:30:56

### Kafka Java Client Exceptions

**The following describes client configuration or service exceptions. If any of the following exceptions occurs, the client will not automatically retry.**

| Exception | Description | Analysis |
| --- | --- | --- |
| UnknownServerException | An unknown error occurred when the server processed the request. | The error will be returned during traffic throttling in the legacy version. If it occurs in the new version, it may be caused by a server bug. |
| RecordTooLargeException | Message is too large. | The current configuration is `message.max.bytes=1000012` . |
| InvalidRequiredAcksException | The `acks` parameter configured for the producer is invalid. | - |
| InconsistentGroupProtocolException | Group protocol is inconsistent with the client protocol. | Check whether the same `group.id` is configured for the consumer and the connector. They cannot join the same group if different protocols are used. |
| InvalidGroupIdException | Consumer group ID is invalid. | Use no more than 128 characters such as a-z, A-Z, 0-9, and ._-. |
| InvalidTopicException | Topic is invalid. | When topic auto-creation is enabled, an exception will be returned if the client uses an invalid topic. Check whether the topic name contains invalid characters or exceeds the length limit. |

| Exception | Description | Analysis |
| --- | --- | --- |
| InvalidSessionTimeoutException | Session.timeout.ms configured for the consumer is invalid. | Current minimum and maximum values allowed by the server are 6000 (value of `group.min.session.timeout.ms`) and 300000 (value of `group.max.session.timeout.ms`). |
| InvalidCommitOffsetSizeException | Submitted offset information is too large and exceeds the maximum message size, so `__consumer_offsets` cannot be written. | Current configuration is `message.max.bytes=1000012`. |
| OffsetMetadataTooLarge | Metadata contained in the request to submit the offset is too large. | `offset.metadata.max.bytes=4096` is configured for the server. |
| UnsupportedVersionException | Broker does not support requests in this version. | We recommend using a v0.10.2.x client. |

**The following exceptions may occur briefly during normal program operation, and the client will automatically retry. If an exception persists, the service is running improperly.**

| Exception | Description | Analysis |
| --- | --- | --- |
| TimeoutException | Request timeout. | If the initial connection reports a request timeout, check whether the address is correct and run telnet to confirm whether the network works properly. If this exception occurs only occasionally during program execution, it may be caused by network jitters. |

| Exception | Description | Analysis |
| --- | --- | --- |
| CorruptRecordException | Message is invalid. | Possible causes include CRC error or invalid data size. This exception may also occur if the compression method used is **gzip** or **the version is below 0.9**. |
| UnknownTopicOrPartitionException | Topic or partition does not exist. | Go to the console to check whether the corresponding topic has been created. Note: the client produces and consumes through `TopicName` rather than `TopicId`. This exception will also occur if the client does not have permission to access the topic. |
| LeaderNotAvailableException | Partition has no leader. | When the topic has just been created, the server has not selected the appropriate leader. An error will be returned to the client and the client will automatically retry to get the leader information. **Only the legacy version has this exception, which has been removed from 0.10.2.1.** |
| NotLeaderForPartitionException | Partition leader is unavailable. | As the client caches the metadata of the topic, when the partition leader changes, production or consumption requests may still be sent to the original leader. An error will be returned to the client and the client will automatically update the metadata. |

| Exception | Description | Analysis |
| --- | --- | --- |
| NetworkException | Client connection is closed by the server. | Network exception or the number of connections exceeds the limit. |
| NotEnoughReplicasException | Number of ISRs is insufficient. | The number of ISRs in the partition when data is written is smaller than the value of `min.insync.replicas` configured for the topic, which may be caused by ISR jitters. |
| NotEnoughReplicasAfterAppendException | ISR jitters occur after data is written to the local Broker, making the configuration of `min.insync.replicas` unsatisfied. | - |
| BrokerNotAvailableError | Partition leader is not found. | As the client caches the metadata of the topic, when the partition leader changes, the production or consumption requests may still be sent to the original leader. An error will be returned to the client and the client will automatically update the metadata. After the leader changes, new production requests sent to the original leader will be automatically forwarded to the new leader upon this error reporting. Theoretically, the integrity of consumption data written will not be affected. |

| Exception | Description | Analysis |
| --- | --- | --- |
| NotLeaderForPartitionError | Partition leader is not found. | As the client caches the metadata of the topic, when the partition leader changes, production or consumption requests may still be sent to the original leader. An error will be returned to the client and the client will automatically update the metadata. After the leader changes, new production requests sent to the original leader will be automatically forwarded to the new leader upon this error reporting. Theoretically, the integrity of consumption data written will not be affected. |

**The following exceptions will occur if the log level is configured as DEBUG and will be automatically processed by the client.**

| Exception | Description | Analysis |
| --- | --- | --- |
| OffsetOutOfRangeException | Offset passed in was out of range for consumer pulled messages. | If an offset reset policy (earliest or latest) is configured for the client, the client will reset the offset according to the policy; otherwise, the user application needs to handle this exception. |
| GroupLoadInProgressException | Coordinator of the consumer group is being loaded. | This may occur briefly when the server is being upgraded. The client will automatically retry. |

| Exception | Description | Analysis |
| --- | --- | --- |
| GroupCoordinatorNotAvailableException | Coordinator is not available. | This may occur briefly when the server is being upgraded. The client will automatically retry. |
| NotCoordinatorForGroupException | The current node is not the coordinator of the consumer group, and the coordinator has been migrated to another node. | This may occur briefly when the server is being upgraded. The client will automatically retry. |
| IllegalGenerationException | Generation of the consumer group is invalid. | Possible causes include heartbeat timing out or a new consumer joining the group. The consumer will automatically retry joining the consumer group. |
| RebalanceInProgressException | Consumer group is rebalancing. | Possible causes include heartbeat timing out or a new consumer joining the group. The consumer will automatically retry joining the consumer group. |