消息队列 CKafka 版

高可用

腾讯云

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

文档目录

高可用
多可用区部署

最近更新时间：2025-03-17 14:34:24

概述

CKafka 支持跨可用区部署，在拥有3个或3个以上可用区的地域购买 CKafka 实例时，专业版可以最多选择四个可用区，高级版最多可以选择两个可用区进行部署。该实例分区副本会强制分布在各个可用区节点上，这种部署方式能够让您的实例在单个可用区不可用情况下仍能正常提供服务。

部署架构

CKafka 的跨可用区部署分为网络层、数据层和控制层。

网络层

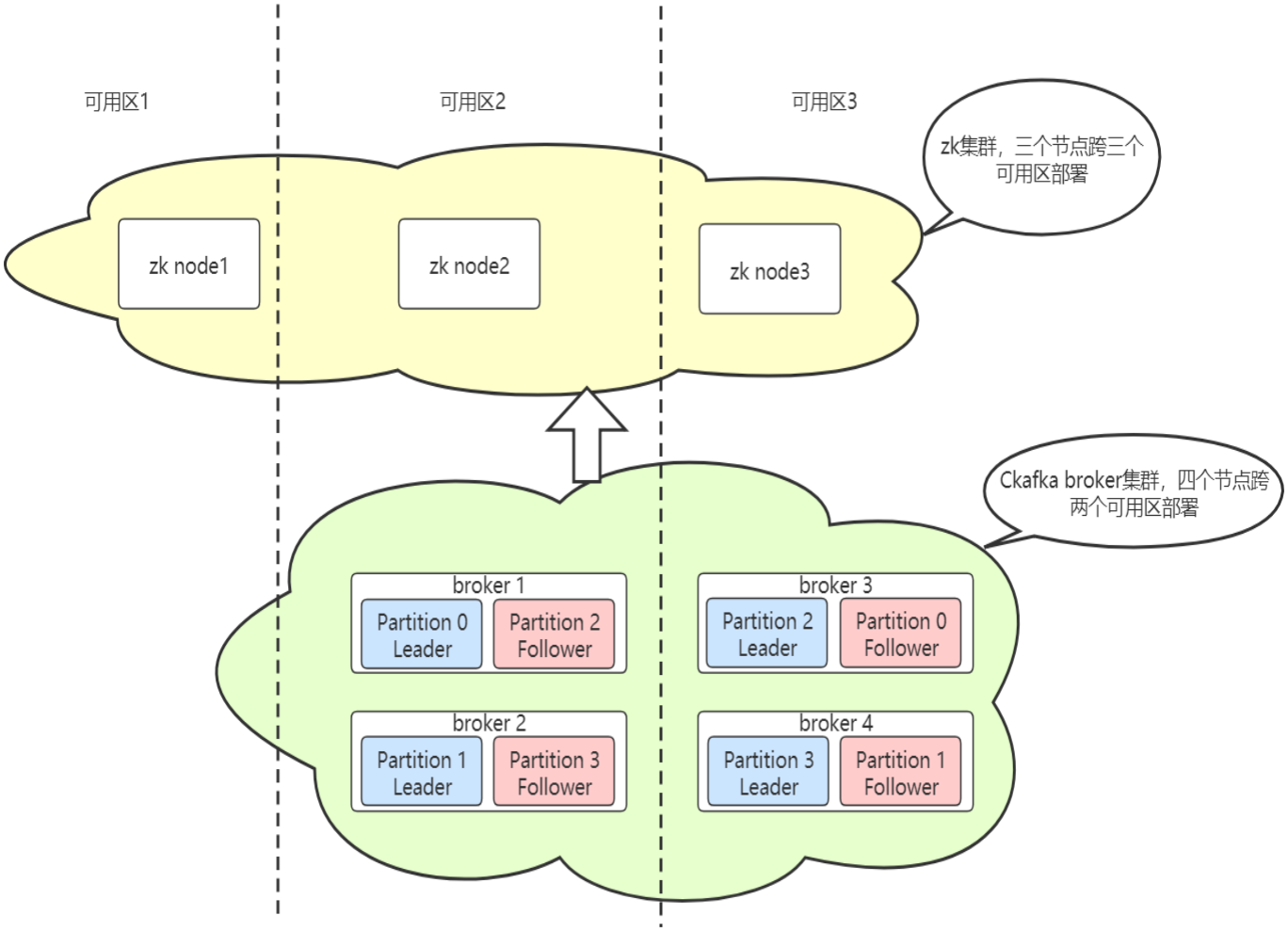
CKafka 会为客户端暴露一个 VIP，客户端在连接到 VIP 后，会拿到主题分区的元数据信息（该元数据通常是地址会通过同一个 VIP 的不同 port 进行一一映射）。这是一个可以随时 failover 到另一个可用区的 VIP，当某个可用区不可用时，该 VIP 会自动漂移到该地域另一个可用的节点，从而实现跨可用区容灾。

数据层

CKafka 数据层和原生 Kafka 采用相同的分布式部署方式，即多个数据副本分布在不同 broker 节点，不同节点会部署在不同可用区。在处理某个分区时，不同的节点之间会有 leader-follower 的关系，当 leader 发生异常不在线时，集群控制节点（Controller）会选举出新的分区 leader 来承接这个分区的请求。
对于客户端来说，当某个可用区出现异常不可用后，如果某个主题分区的 leader 位于不可用区 broker 节点上，则原先建立的相关链接会出现超时或者链接被关闭的情况，当该分区 leader 节点异常之后，Controller（若 Controller 节点异常，剩余节点会竞选出新的 Controller 节点）会选举出新的leader节点提供服务，leader 切换 时间在秒级（具体切换时间与集群节点个数，元数据大小成正比）。客户端会定时刷新主题分区元数据信息，链接新的 leader 节点进行生产消费。

控制层

CKafka 的控制层和原生 Kafka 采用相同的技术方案，依赖 zookeeper 对 broker 节点进行服务发现和集群 Controller 选举。**支持跨可用区部署的 CKafka 实例，其中 zookeeper 集群 zk 节点（以下简称 zk 节点）部署在三个可用区（或机房）。**当其中任意一个可用区的 zk 节点出现故障断连，整个 zk 集群仍可以正常提供服务。



跨可用区部署优劣势

优势

可以大幅度提升集群的容灾能力，当单个可用区出现意外的网络不稳定、断电重启等不可抗力风险时，仍能保证客户端在短时间等待重连后恢复消息的生产和消费。

劣势

如果采取跨可用区部署，由于分区副本分布在多个可用区上，故消息复制相比单个可用区存在额外的跨区网络时延，该时延会直接影响到生产（客户端 Ack 参数大于1，或者等于-1, all）的客户端写入耗时。目前广州、上海、北京几个主要地域跨可用区的时延一般10ms~40ms。

跨可用区部署场景解析

单 AZ 不可用

单个 AZ 不可用后，如前文对原理的解析，客户端会出现断连重连，重连后服务仍能正常提供。

由于管控 API 服务目前不支持跨可用区部署，所以在单个 AZ 不可用之后，可能出现无法通过控制台创建 Topic，配置 ACL 策略，查看监控等现象，但不会影响存量业务的生产消费。

两个 AZ 网络隔离

如果两个 AZ 之间出现网络隔离，即无法相互通信，则可能会出现集群脑裂的现象，即两个可用区的节点都提供服务，但其中一个可用区的数据写入会在集群恢复后视为脏数据。
考虑如下场景，当集群 Controller 节点和 zk 集群一个 zk 节点与其他节点发生网络隔离。此时，其他节点会重新竞选产生新的 Controller（因为 zk 集群多数节点网络通信正常，则 Controller 可以竞选成功）但是发生网络隔离的 Controller 仍然认为自己是 Controller 节点，这时候集群会出现脑裂的情况。
此时客户端的写入需要分情况考虑，这里举个例子，当客户端的 Ack 策略等于 -1 或者 all，副本数为2时，假设集群是3节点，脑裂之后会2:1分布，原先 leader 在1节点地域的分区写入会报错，另一边则会正常写入。而一旦是3副本且配置了 Ack = -1 或者 all，则两边都不会写入成功。这时就需要根据具体参数配置来确定进一步的处理方案。
在集群网络恢复后，客户端无需做操作即可恢复生产消费，但是由于服务端会重新对数据进行归一化，其中一个分裂节点的数据会被直接截断，但对于多副本跨区的数据存储方式来说，这种截断也并不会带来数据丢失。

多 AZ 容灾限制

容量限制

CKafka 是通过底层资源多 AZ 分布实现多 AZ 容灾。AZ 故障情况下，CKafka 会将 Partition Leader 切换到可用 AZ 的 Broker 节点中，所以承载流量的底层资源会变少，因此会有容量问题。以下是单 AZ 故障资源可用情况：

- 2 AZ 容灾实例，可使用容量 = 实例 / 2，为保证正常使用，客户需冗余 100% 的资源。
- 3 AZ 容灾实例，可使用容量 = 实例 / 3 * 2，为保证正常使用，客户需冗余 50% 的资源。
- 4 AZ 容灾实例，可使用容量 = 实例 / 4 * 3，为保证正常使用，客户需冗余 33.3% 的资源。
- n AZ 容灾实例，可使用容量 = 实例 / n * (n - 1)，为保证正常使用，客户需冗余 1 / (n -1) 的资源

参数及配置限制

CKafka 支持在控制台修改 topic 级别的配置 `min.insync.replicas`，该配置在客户端设置 ack=-1 时生效，可以确保一条消息被 `min.insync.replicas` 个副本同时同步才返回生产成功(如：topic 副本 = 3，min.insync.replicas=2，则需要生产消息至少被 2 个副本同步才算成功)。
所以 topic 的配置 (`min.insync.replicas` < AZ 数)，才能保证 AZ 故障情况下生产可用性；topic 的配置 `min.insync.replicas` < topic副本数才能保证单节点故障下生产可用性。设置规则为：
`min.insync.replicas` < topic 副本数 <= AZ 数

- 2 AZ 实例，topic 副本数 = 2，min.insync.replicas = 1
- 3 AZ 实例，topic 副本数 = 2，min.insync.replicas = 1
- 3 AZ 实例，topic 副本数 = 3，min.insync.replicas <= 2

多副本及选举机制

最近更新时间：2025-03-17 14:34:25

多副本

多副本设计可增强系统可用性、可靠性。
Replica 均匀分布到整个集群，Replica 的算法如下：

1. 将所有 Broker（假设共 n 个 Broker）和待分配的 Partition 排序。
2. 将第 i 个 Partition 分配到第（i mod n）个 Broker 上。
3. 将第 i 个 Partition 的第 j 个 Replica 分配到第（(i + j) mod n）个 Broker 上。

Leader Election 选举机制

消息队列 CKafka 版在 ZooKeeper 中动态维护了一个 ISR（in-sync replicas），ISR 里的所有 Replica 都跟上了 Leader。只有 ISR 里的成员才有被选为 Leader 的可能。

- 假设 ISR 中 f + 1个 Replica，一个 Partition 能在保证不丢失已 commit 的消息的前提下
- 容忍 f 个 Replica 的失败。
- 假设共有 2f + 1个 Replica（包含 Leader 和 Follower），commit 之前必须保证有 f + 1个
- Replica 复制完消息，为了保证正确选出新的 Leader，fail 的 Replica 不能超过 f 个。

数据高可靠

最近更新时间：2025-03-17 14:34:24

本文将分别通过生产端、服务端（CKafka）和消费端介绍影响消息队列 CKafka 版可靠性的因素，并提供对应的解决方法。

生产端数据丢失如何处理?

数据丢失原因

生产者将数据发送到消息队列 CKafka 版时，数据可能因为网络抖动而丢失，此时消息队列 CKafka 版未收到该数据。可能情况：

- 网络负载高或者磁盘繁忙时，生产者又没有重试机制。
- 磁盘超过购买规格的限制，例如实例磁盘规格为 9000GB，在磁盘写满后未及时扩容，会导致数据无法写入到消息队列 CKafka 版。
- 突发或持续增长峰值流量超过购买规格的限制，例如实例峰值吞吐规格为 100MB/s，在长时间峰值吞吐超过限制后未及时扩容，会导致数据写入消息队列 CKafka 版变慢，生产者有排队超时机制时，导致数据无法写入到消息队列 CKafka 版。

解决方法

- 生产者对自己重要的数据，开启失败重试机制。
- 针对磁盘使用，在配置实例时设置好监控和 告警策略 ，可以做到事先预防。
  遇到磁盘写满时，可以在控制台及时升配（消息队列 CKafka 版非独占实例间升配为平滑升配不停机且也可以单独升配磁盘）或者通过修改消息保留时间降低磁盘存储。
- 为了尽可能减少生产端消息丢失，您可以通过 buffer.memory 和 batch.size（以字节为单位）调优缓冲区的大小。缓冲区并非越大越好，如果由于某种原因生产者宕机了，那么缓冲区存在的数据越多，需要回收的垃圾越多，恢复就会越慢。应该**时刻注意生产者的生产消息数情况、平均消息大小**等（消息队列 CKafka 版监控中有丰富的监控指标）。
- 配置生产端 ACK
  当 producer 向 leader 发送数据时，可以通过 request.required.acks 参数以及 min.insync.replicas 设置数据可靠性的级别。
- 当 acks = 1时（默认值），生产者在 ISR 中的 leader 已成功收到数据可以继续发送下一条数据。如果 leader 宕机，由于数据可能还未来得及同步给其 follower，则会丢失数据。
- 当 acks = 0时，生产者不等待来自 broker 的确认就发送下一条消息。这种情况下数据传输效率最高，但数据可靠性最低。

> ⚠ **注意：**

> 当生产端配置 acks = 0 时，如果当前实例被限流，为了保护服务端能正常提供服务，服务端会主动关闭与客户端的连接。

- 当 acks = -1或者 all 时，生产者需要等待 ISR 中的所有 follower 都确认接收到消息后才能发送下一条消息，可靠性最高。
  即使按照上述配置 ACK，也不能保证数据不丢，例如，当 ISR 中只有 leader 时（ISR 中的成员由于某些情况会增加也会减少，最少时只剩一个 leader），此时会变成 acks = 1的情况。所以需要同时在配合 min.insync.replicas 参数（此参数可以在消息队列 CKafka 版控制台 Topic 配置开启高级配置中进行配置），min.insync.replicas 表示在 ISR 中最小副本的个数，默认值是1，当且仅当 acks = -1或者 all 时生效。

建议配置的参数值

此参数值仅供参考，实际数值需要依业务实际情况而定。

- 重试机制：`message.send.max.retries=3;retry.backoff.ms=10000;`
- 高可靠的保证：`request.required.acks=-1;min.insync.replicas=2;`
- 高性能的保证：`request.required.acks=0;`
- 可靠性+性能：`request.required.acks=1;`

服务端（CKafka）数据丢失如何处理?

数据丢失原因

- partition 的 leader 在未完成副本数 followers 的备份时就宕机，即使选举出了新的 leader 但是数据因为未来得及备份就丢失。
- 开源 Kafka 的落盘机制为异步落盘，也就是数据是先存在 PageCache 中的，当还没有正式落盘时，broker 出现断开连接或者重启或者故障时，PageCache 上的数据由于没有来得及落盘进而丢失。
- 磁盘故障导致已经落盘的数据丢失。

解决方法

- 开源 Kafka 是多副本的，官方推荐通过副本来保证数据的完整性，此时如果是多副本，同时出现多副本多 broker 同时挂掉才会丢数据，比单副本数据的可靠性高很多，所以消息队列 CKafka 版强制 Topic 是双副本，可配置3副本。
- 消息队列 CKafka 版服务配置了更合理的参数 log.flush.interval.messages 和 log.flush.interval.ms，对数据进行刷盘。
- 消息队列 CKafka 版对磁盘做了特殊处理，保证部分磁盘损坏时也不会影响数据的可靠性。

建议配置的参数值

非同步状态的副本可以选举为 leader：`unclean.leader.election.enable=false // 关闭`

消费端数据丢失如何处理?

数据丢失原因

- 还未真正消费到数据就提交 commit 了 offset，若过程中消费者挂掉，但 offset 已经刷新，消费者错过了一条数据，需要消费分组重新设置 offset 才能找回数据。
- 消费速度和生产速度相差太久，而消息保存时间太短，导致消息还未及时消费就被过期删除。

解决方法

- 合理配置参数 auto.commit.enable，等于 true 时表示自动提交。建议使用定时提交，避免频繁 commit offset。
- 监控消费者的情况，正确调整数据的保留时间。监控当前消费 offset 以及未消费的消息条数，并配置告警，防止由于消费速度过慢导致消息过期删除。

数据丢失排查方案

在本地打印分区 partition 和偏移量 offset 进行排查

打印信息代码如下：

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

- 如果能够打印出 partition 和 offset，则表示当前发送的消息在服务端已经被正确保存。此时可以通过消息查询的工具去查询相关位点的消息即可。
- 如果打印不出 partition 和 offset，则表示消息没有被服务端保存，客户端需要重试。

迁移可用区

最近更新时间：2025-03-17 14:34:22

操作场景

您可以将**消息队列 CKafka 版专业版实例**迁移至同一地域内的其它可用区。迁移可用区后，实例的所有属性、配置和连接地址都不会改变。迁移所需时间跟实例的数据量有关。
例如在如下场景中，您可以选择迁移可用区：

- 假设您正在尝试修改实例的实例类型，但我们无法在当前可用区中启动新实例类型的实例。在这种情况下，您可以将实例迁移到能够启动该实例类型的可用区。
- 当前可用区已无资源进行扩容的情况下，您也可以将实例迁移至同地域内其他资源充足的可用区，以满足业务需要。

前提条件

- 实例状态为运行中。
- 实例所在的地域需要有多个可用区，才支持迁移可用区功能。

费用说明

本功能免费。即使将实例从单可用区迁移至多个可用区，也不收取费用。

功能说明

- 当原实例是单可用区部署时，支持切换可用区，也可以升级成多可用区部署。关于多可用区部署详情请参见 跨可用区部署 。
- 当原实例是多可用区部署时，支持切换可用区，不支持切换回单可用区部署。

迁移类型及场景说明

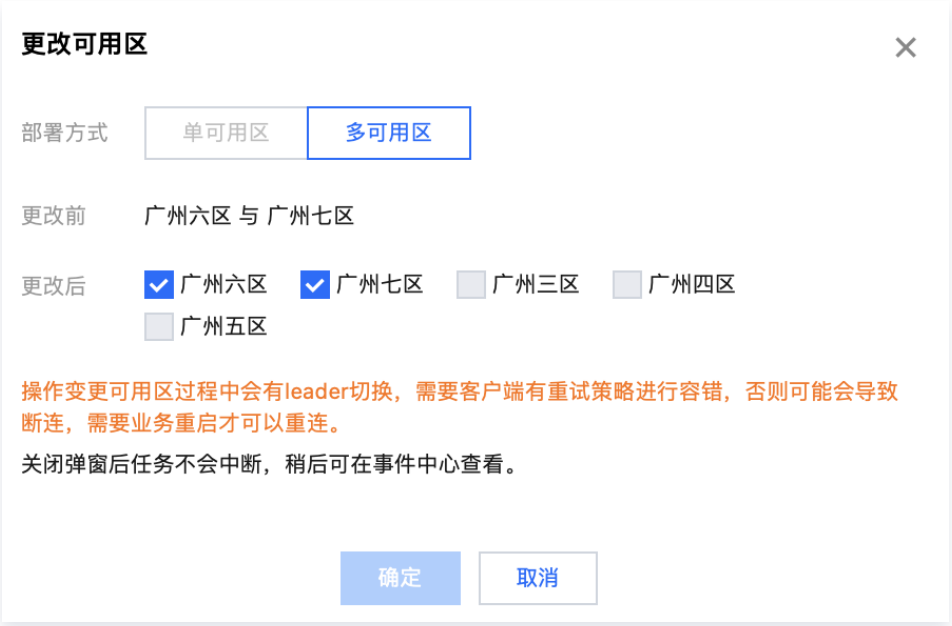
迁移类型	场景
从一个可用区迁移至另一个可用区	实例所在可用区出现满负载或者其它影响实例性能的情况。
从一个可用区迁移至多个可用区	提高实例的容灾能力，实现跨机房容灾。主备实例分别位于不同的可用区。相对于单可用区实例，多可用区实例可以承受更高级别的灾难。例如，单可用区实例可以承受服务器和机架级别的故障，而多可用区实例可以承受机房级别的故障。

操作步骤

> ⚠ **注意：**

> 操作变更可用区过程中会有 leader 切换，需要客户端有重试策略进行容错，否则可能会导致断连，需要业务重启才可以重连。

1. 登录 CKafka 控制台 。
2. 在左侧导航栏单击**实例列表**，单击目标实例的“ID/名称”，进入基本信息页。
3. 在基本信息模块，单击可用区右边的编辑按钮，选择您要选择的切换的可用区。



4. 单击**确认**，预计等待5-10分钟完成变配，在实例列表的状态栏可以查看变配进度。