启用基于 Kafka 的排序服务

郑重声明

本文档假设读者知道怎么配置 Kafka 和 ZooKeeper 集群，并阻止了非授权访问以保证它们在使用过程中的安全性。本指南的唯一目的是说明如何配置你的 Hyperledger Fabric 排序服务节点（OSN，ordering service node）使用 Kafka 集群为你的区块链网络提供排序服务的步骤。

关于排序节点的角色以及它在网络和交易流程中的作用的信息，请参考 排序服务 。

关于如何设置排序节点的信息，请参考 设置排序节点 。

关于配置 Raft 排序服务的信息，请参考 配置并使用 Raft 排序服务 。

概览

每个通道映射到 Kafka 中一个单独的单分区主题（topic）。当一个 OSN 通过 `Broadcast` RPC 接收到交易时，它会进行检查以确认广播的客户端有写入通道的权限，然后将交易转发（或者说是生产）到 Kafka 中合适的分区中。这个分区被 OSN 消费，将接受到的交易打包到本地区块，持久化保存在他们的本地账本，然后通过 `Deliver` RPC 将他们发送给接收客户端。底层的细节请参考 the document that describes how we came to this design ，图表 8 阐述了上述过程。

步骤

使用 `K` 和 `Z` 表示 Kafka 和 ZooKeeperer 集群中的节点数量：

1. `K` 的最小值为4。（我们将在第4步中解释，这是崩溃错误容忍的最小节点数量，比如，有4个 broker，当有1个 broker 宕机的时候，仍然可以继续读写和创建新通道。）
2. `Z` 可以是3、5或者7。它应该是奇数个以防止脑裂的发生，并且多余1个以防止单点故障。多余7个 ZooKeeper 服务器就没有必要了。

具体过程如下：

3. Orderers： **Kafka 相关的信息编码在网络的创世区块中。** 如果你使用了 `configtxgen`，编辑 `configtx.yaml`，或者使用一个系统通道创世区块的预配置文件，然后：

   - `Orderer.OrdererType` 设置为 `kafka`。
   - `Orderer.Kafka.Brokers` 包含 *至少两个* 你集群中的 Kafka broker 的 `IP:port` 。列表中不需要是所有的节点。（这些是你的引导 broker 。）

4. Orderers: **设置最小区块大小。** 每个区块最大为 *Orderer.AbsoluteMaxBytes* 个字节（不包含头部），这个值你可以在 `configtx.yaml` 中设置。我们使用 `A` 来代表它，它将影响到我们在第6步对 Kafka broker 的配置。

5. Orderers： **创建创世区块。** 使用 `configtxgen` 。在第3步和第4步的设置是系统层级的设计，将影响到网络中所有的 OSN。记下创世区块的位置。
6. Kafka 集群： **合理的配置你的 Kafka brokers。** 确保每一个 Kafka broker 配置了这些关键项：

- `unclean.leader.election.enable = false` — 数据持久化是区块链环境中的重要环节。我们不能在同步复制集合之外选择一个领导通道，或者我们冒着覆盖上一个领队产生的偏移量的风险，并因此重写排序节点产生的区块。

    - `min.insync.replicas = M` — 这里的值 `M` 设为 `1 < M < N` （查看下边的 `default.replication.factor`）。 数据写入至少 `M` 个副本之后才认为被提交。其他情况下，写操作返回一个错误。然后：
        - 如果写入的 `N` 个副本中有 `N-M` 个不可用，操作仍可正常运行。
        - 如果有更多的副本不可用，Kafka 就不能维护 ISR 集合中的 `M` 个，所以它就会停止接受写入。读取是没有问题的。当重新同步到 `M` 个副本的时候，通道可以恢复写的功能。

- `default.replication.factor = N` — 这里的值 `N` 设为 `N < K` 。 `N` 个副本意味着每个通道都会将它的数据备份到 `N` 个 broker。 这些是一个通道 ISR 集合的备份。就像我们在上边提到的 `min.insync.replicas section` 不是所有的节点一直都是可用的。 `N` 的值要设置的小于 `K` ，因为当少于 `N` 个 broker 运行的时候就不能创建通道了。所以，如果你设置为 `N = K` ，那么只要有一个 broker 宕机了，就意味着区块链网络就不能创建通道了，也就是说排序服务的崩溃容错就不存在了。

  基于我们上边所说的， `M` 和 `N` 的最小值分别为2和3。这样的配置可以保证新通道的创建，并且所有的通道都持续可写。
- `message.max.bytes` 和 `replica.fetch.max.bytes` 的值应该设 `A` 的值大，在上边你在 `Orderer.AbsoluteMaxBytes` 中将 `A` 的值设为了4。再为头部数据增加一些空间（多余 1 MiB 就够了）。以下条件适用：

```
  Orderer.AbsoluteMaxBytes < replica.fetch.max.bytes <= message.max.bytes
(为了完备性的考虑，我们要求 ``message.max.bytes`` 的值小于 ``socket.request.max.bytes`` ， ``sock
```

- `log.retention.ms = -1`. Until the ordering service adds support for pruning of the Kafka logs, you should disable time-based retention and prevent segments from expiring. (Size-based retention — see `log.retention.bytes` — is disabled by default in Kafka at the time of this writing, so there's no need to set it explicitly.)

7. Orderers： **将每一个 OSN 指向创世区块。** 编辑 `orderer.yaml` 中的 `General.BootstrapFile` 来指定 Orderer 指向步骤5中创建的创世区块。（同时，要确保 YAML 文件中的其他键合理的配置。）
8. Orderers： **调整轮询间隔和超时。** （可选步骤。）

- `orderer.yaml` 文件中的 `Kafka.Retry` 部分可以让你调整 metadata/producer/consumer 请求的频率和 socket 超时时间。（这里有你希望看到的 Kafka 生产者和消费者的全部信息。）
- 另外，当创建一个新通道时，或者重新加载一个存在的通道时（比如重启一个排序节点），排序节点和 Kafka 集群的交互过程如下：

    - 排序节点为该通道相关的 Kafka 分区创建一个 Kafka 生产者（写入者）。
    - 排序节点使用生产者向分区发送一个无操作的 `CONNECT` 消息。

- 排序节点为分区创建一个 Kafka 消费者（读取者）。
- 即使任意一个步骤失败了，你也可以通过调整重试的频率重复上边的步骤。他们将会每隔 `Kafka.Retry.ShortInterval` 所设置的时间进行 `Kafka.Retry.ShortTotal` 次尝试，和每隔 `Kafka.Retry.LongInterval` 所设置的时间进行 `Kafka.Retry.LongTotal` 次尝试，直到成功为止。注意，排序节点只有在上述步骤成功完成后才可以进行读写。

9. **设置 OSN 和 Kafka 之间的 SSL 通信。** （可选步骤，但是强烈建议。）参考 the Confluent guide 配置 Kafka 集群的设置，然后在每一个相关的 OSN 中设置 `orderer.yaml` 中 `Kafka.TLS` 的键值。
10. **以如下顺序启动节点：ZooKeeper 集群，Kafka 集群，排序服务节点。**

其他注意事项

1. **首选消息容量。** 在上边第4步中（查看 `Steps`_ 部分）你可以通过设置 `Orderer.Batchsize.PreferredMaxBytes` 来设定默认区块大小。Kafka 对于相对较小的消息有较高的吞吐量；所以该值不要大于1 MiB。
2. **使用环境变量覆盖设置。** 当使用 Fabric 提供的示例 Kafka 和 ZooKeeper Docker 镜像时（请查看 `images/kafka` 和 `images/zookeeper` 相关信息），你可以通过环境变量来覆盖 Kafka broker 或者 ZooKeeper 服务器的设置。将配置文件中的点替换为下划线，例如 `KAFKA_UNCLEAN_LEADER_ELECTION_ENABLE=false` 将覆盖 `unclean.leader.election.enable` 的值。这将和 OSN *本地* 配置文件的效果是一样的，例如在 `orderer.yaml` 中的设置。例如 `ORDERER_KAFKA_RETRY_SHORTINTERVAL=1s` 将覆盖 `Orderer.Kafka.Retry.ShortInterval` 所设置的值。

Kafka 协议版本兼容性

Fabric 使用 sarama client library 支持 Kafka 0.10 到 1.0 的版本，同样还支持较老的版本。

使用 `orderer.yaml` 中的 `Kafka.Version` 键，你可以配置你使用哪个 Kafka 协议版本和 Kafka 集群的 brokers 通信。使用老协议版本的 Kafka 代理向后兼容。因为 Kafka 代理对老协议版本的向后兼容性，升级你的 Kafka 代理版本时不需要升级 `Kafka.Version` 的键值，但是 Kafka 集群使用老协议版本可能会出现 性能损失 。

调试

将环境变量 `FABRIC_LOGGING_SPEC` 设置为 `DEBUG` 和 `orderer.yaml` 中的 *Kafka.Verbose`* 设置为 `true` 。