kafka集群部署

1. 准备工作

名称	IP地址	Hostname	组织结构
zk1	192.168.247.101	zookeeper1	
zk2	192.168.247.102	zookeeper2	
zk3	192.168.247.103	zookeeper3	
kafka1	192.168.247.201	kafka1	
kafka2	192.168.247.202	kafka2	
kafka3	192.168.247.203	kafka3	
kafka4	192.168.247.204	kafka4	
orderer0	192.168.247.91	orderer0.test.com	
orderer1	192.168.247.92	orderer1.test.com	
orderer2	192.168.247.93	orderer2.test.com	
peer0	192.168.247.81	peer0.orggo.test.com	OrgGo
peer0	192.168.247.82	peer0.orgcpp.test.com	OrgCpp

为了保证整个集群的正常工作, 需要给集群中的各个节点设置工作目录, 我们要保证各个节点工作目录是相同的

```
# 在以上各个节点的家目录创建工作目录:
$ mkdir ~/kafka
```

2. 生成证书文件

2.1 编写配置文件

```
# crypto-config.yaml
OrdererOrgs:
  - Name: Orderer
    Domain: test.com
    Specs:
      - Hostname: orderer0  # 第1个排序节点: orderer0.test.com
      - Hostname: orderer1  # 第2个排序节点: orderer1.test.com
      - Hostname: orderer2  # 第3个排序节点: orderer2.test.com
```

```

PeerOrgs:
  - Name: OrgGo
    Domain: orggo.test.com
    Template:
      Count: 2  # 当前go组织两个peer节点
    Users:
      Count: 1

  - Name: OrgCpp
    Domain: orgcpp.test.com
    Template:
      Count: 2  # 当前cpp组织两个peer节点
    Users:
      Count: 1
```

2.2 生成证书

```
$ cryptogen generate --config=crypto-config.yaml
$ tree ./ -L 1
./
├── `crypto-config`   -> 证书文件目录
└── crypto-config.yaml
```

3. 生成创始块和通道文件

3.1 编写配置文件

> 配置文件名 `configtx.yaml` 这个名字是固定的, 不可修改的

```

---
################################################################################
#
#   Section: Organizations
#
#   - This section defines the different organizational identities which will
#   be referenced later in the configuration.
#
################################################################################
Organizations:
    - &OrdererOrg
        Name: OrdererOrg
        ID: OrdererMSP
        MSPDir: crypto-config/ordererOrganizations/test.com/msp

    - &go_org
        Name: OrgGoMSP
        ID: OrgGoMSP
```

```yaml
        MSPDir: crypto-config/peerOrganizations/orggo.test.com/msp
        AnchorPeers:
            - Host: peer0.orggo.test.com
              Port: 7051

    - &cpp_org
        Name: OrgCppMSP
        ID: OrgCppMSP
        MSPDir: crypto-config/peerOrganizations/orgcpp.test.com/msp
        AnchorPeers:
            - Host: peer0.orgcpp.test.com
              Port: 7051

################################################################################
#
#   SECTION: Capabilities
#
################################################################################
Capabilities:
    Global: &ChannelCapabilities
        V1_1: true
    Orderer: &OrdererCapabilities
        V1_1: true
    Application: &ApplicationCapabilities
        V1_2: true

################################################################################
#
#   SECTION: Application
#
################################################################################
Application: &ApplicationDefaults
    Organizations:

################################################################################
#
#   SECTION: Orderer
#
################################################################################
Orderer: &OrdererDefaults
    # Available types are "solo" and "kafka"
    OrdererType: kafka
    Addresses:
        # 排序节点服务器地址
        - orderer0.test.com:7050
        - orderer1.test.com:7050
        - orderer2.test.com:7050

    BatchTimeout: 2s
    BatchSize:
        MaxMessageCount: 10
        AbsoluteMaxBytes: 99 MB
        PreferredMaxBytes: 512 KB
    Kafka:
        Brokers:
            # kafka服务器地址
            - 192.168.247.201:9092
            - 192.168.247.202:9092
            - 192.168.247.203:9092
            - 192.168.247.204:9092
    Organizations:

################################################################################
#
#   Profile
#
################################################################################
Profiles:
    OrgsOrdererGenesis:
        Capabilities:
            <<: *ChannelCapabilities
        Orderer:
            <<: *OrdererDefaults
            Organizations:
                - *OrdererOrg
            Capabilities:
                <<: *OrdererCapabilities
        Consortiums:
            SampleConsortium:
                Organizations:
                    - *go_org
                    - *cpp_org
    OrgsChannel:
        Consortium: SampleConsortium
        Application:
            <<: *ApplicationDefaults
            Organizations:
                - *go_org
                - *cpp_org
            Capabilities:
                <<: *ApplicationCapabilities
```

3.2 生成通道和创始块文件

- 生成创始块文件

```
# 我们先创建一个目录 channel-artifacts 存储生成的文件, 目的是为了和后边的配置文件模板的配置项保持一致
$ mkdir channel-artifacts
# 生成通道文件
$ configtxgen -profile OrgsOrdererGenesis -outputBlock ./channel-artifacts/genesis.block
```

- 生成通道文件

```
# 生成创始块文件
$ configtxgen -profile OrgsChannel -outputCreateChannelTx ./channel-artifacts/channel.tx -channelID testchannel
```

4. Zookeeper设置

4.1 基本概念

> Zookeeper一种在分布式系统中被广泛用来作为分布式状态管理、分布式协调管理、分布式配置管理和分布式锁服务的集群。

- zookeeper 的运作流程

  > 在配置之前, 让我们先了解一下 `Zookeeper` 的基本运转流程:
  >
  > - 选举Leader
  >   - 选举Leader过程中算法有很多，但要达到的选举标准是一致的
  >   - Leader要具有最高的执行ID，类似root权限。
  >   - 集群中大多数的机器得到响应并跟随选出的Leader。
  > - 数据同步

- Zookeeper的集群数量

  > Zookeeper 集群的数量可以是 `3, 5, 7,` 它值需要是一个奇数以避免脑裂问题（split-brain）的情况。同时选择大于1的值是为了避免单点故障，如果集群的数量超过7个Zookeeper服务将会无法承受。

4.2 zookeeper配置文件模板

- 配置文件模板

  > 下面我们来看一个示例配置文件, 研究下zookeeper如何配置:

```
version: '2'
services:
  zookeeper1: # 服务器名，自己起
    container_name: zookeeper1 # 容器名，自己起
    hostname: zookeeper1    # 访问的主机名，自己起，需要和IP有对应关系
    image: hyperledger/fabric-zookeeper:latest
    restart: always # 指定为always
    environment:
      # ID在集合中必须是唯一的并且应该有一个值，在1和255之间。
      - ZOO_MY_ID=1
      # server.x=hostname:prot1:port2
      - ZOO_SERVERS=server.1=zookeeper1:2888:3888 server.2=zookeeper2:2888:3888 server.3=zookeeper3:2888:3888
    ports:
      - 2181:2181
      - 2888:2888
      - 3888:3888
    extra_hosts:
      - zookeeper1:192.168.24.201
      - zookeeper2:192.168.24.202
      - zookeeper3:192.168.24.203
      - kafka1:192.168.24.204
      - kafka2:192.168.24.205
      - kafka3:192.168.24.206
      - kafka4:192.168.24.207
```

- 相关配置项解释:

> 1. docker 的 `restart` 策略
>   - no – 容器退出时不要自动重启，这个是默认值。
>   - on-failure[:max-retries] – 只在容器以非0状态码退出时重启， 例如： `on-failure:10`
>   - **always** – 不管退出状态码是什么始终重启容器
>   - unless- **stopped** – 不管退出状态码是什么始终重启容器，不过当daemon启动时，如果容器之前已经为停止状态，不要尝试启动它。
> 2. 环境变量
>   - ZOO_MY_ID
>
>   zookeeper集群中的当前zookeeper服务器节点的ID, 在集群中这个只是唯一的, 范围: 1-255
>   - ZOO_SERVERS
>   - 组成zookeeper集群的服务器列表
>   - 列表中每个服务器的值都附带两个端口号
>   - 第一个: 追随者用来连接 Leader 使用的
>   - 第二个: 用户选举 Leader
> 3. zookeeper服务器中三个重要端口:
>   - 访问zookeeper的端口: 2181
>   - zookeeper集群中追随者连接 Leader 的端口: 2888
>   - zookeeper集群中选举 Leader 的端口: 3888
> 4. extra_hosts
>   - 设置服务器名和其指向的IP地址的对应关系
>   - `zookeeper1:192.168.24.201`
>   - 看到名字 `zookeeper1` 就会将其解析为IP地址: `192.168.24.201`

4.3 各个zookeeper节点的配置

zookeeper1 配置

```
# zookeeper1.yaml
version: '2'

services:

```

```yaml
  zookeeper1:
    container_name: zookeeper1
    hostname: zookeeper1
    image: hyperledger/fabric-zookeeper:latest
    restart: always
    environment:
      # ID在集合中必须是唯一的并且应该有一个值，在1和255之间。
      - ZOO_MY_ID=1
      # server.x=[hostname]:nnnnn[:nnnnn]
      - ZOO_SERVERS=server.1=zookeeper1:2888:3888 server.2=zookeeper2:2888:3888 server.3=zookeeper3:2888:3888
    ports:
      - 2181:2181
      - 2888:2888
      - 3888:3888
    extra_hosts:
      - zookeeper1:192.168.247.101
      - zookeeper2:192.168.247.102
      - zookeeper3:192.168.247.103
      - kafka1:192.168.247.201
      - kafka2:192.168.247.202
      - kafka3:192.168.247.203
      - kafka4:192.168.247.204
```

zookeeper2 配置

```yaml
# zookeeper2.yaml
version: '2'

services:

  zookeeper2:
    container_name: zookeeper2
    hostname: zookeeper2
    image: hyperledger/fabric-zookeeper:latest
    restart: always
    environment:
      # ID在集合中必须是唯一的并且应该有一个值，在1和255之间。
      - ZOO_MY_ID=2
      # server.x=[hostname]:nnnnn[:nnnnn]
      - ZOO_SERVERS=server.1=zookeeper1:2888:3888 server.2=zookeeper2:2888:3888 server.3=zookeeper3:2888:3888
    ports:
      - 2181:2181
      - 2888:2888
      - 3888:3888
    extra_hosts:
      - zookeeper1:192.168.247.101
      - zookeeper2:192.168.247.102
      - zookeeper3:192.168.247.103
      - kafka1:192.168.247.201
      - kafka2:192.168.247.202
      - kafka3:192.168.247.203
      - kafka4:192.168.247.204
```

zookeeper3 配置

```yaml
# zookeeper3.yaml
version: '2'

services:

  zookeeper3:
    container_name: zookeeper3
    hostname: zookeeper3
    image: hyperledger/fabric-zookeeper:latest
    restart: always
    environment:
      # ID在集合中必须是唯一的并且应该有一个值，在1和255之间。
      - ZOO_MY_ID=3
      # server.x=[hostname]:nnnnn[:nnnnn]
      - ZOO_SERVERS=server.1=zookeeper1:2888:3888 server.2=zookeeper2:2888:3888 server.3=zookeeper3:2888:3888
    ports:
      - 2181:2181
      - 2888:2888
      - 3888:3888
    extra_hosts:
      - zookeeper1:192.168.247.101
      - zookeeper2:192.168.247.102
      - zookeeper3:192.168.247.103
      - kafka1:192.168.247.201
      - kafka2:192.168.247.202
      - kafka3:192.168.247.203
      - kafka4:192.168.247.204
```

5. Kafka设置

5.1 基本概念

> Katka是一个分布式消息系统，由LinkedIn使用scala编写，用作LinkedIn的活动流（Activitystream)和运营数据处理管道（Pipeline）的基础。具有高水平扩展和高吞吐量。
>
> 在Fabric网络中，数据是由Peer节点提交到Orderer排序服务，而Orderer相对于Kafka来说相当于上游模块，且Orderer还兼具提供了对数据进行排序及生成符合配置规范及要求的区块。而使用上游模块的数据计算、统计、分析，这个时候就可以使用类似于Kafka这样的分布式消息系统来协助业务流程。
>
> 有人说Kafka是一种共识模式，也就是说平等信任，所有的HyperLedger Fabric网络加盟方都是可信方，因为消息总是均匀地分布在各处。但具体生产使用的时候是依赖于背书来做到确权，相对而言，Kafka应该只能是一种启动Fabric网络的模式或类型。

> Zookeeper一种在分布式系统中被广泛用来作为分布式状态管理、分布式协调管理、分布式配置管理和分布式锁服务的集群。Kafka增加和减少服务器都会在Zookeeper节点上触发相应的事件，Kafka系统会捕获这些事件，进行新一轮的负载均衡，客户端也会捕获这些事件来进行新一轮的处理。
>
> Orderer排序服务是Fablic网络事务流中的最重要的环节，也是所有请求的点，它并不会立刻对请求给予回馈，一是因为生成区块的条件所限，二是因为依托下游集群的消息处理需要等待结果。

5.2 kafka配置文件模板

- kafka配置文件模板

```yaml
version: '2'

services:
  kafka1:
    container_name: kafka1
    hostname: kafka1
    image: hyperledger/fabric-kafka:latest
    restart: always
    environment:
      # broker.id
      - KAFKA_BROKER_ID=1
      - KAFKA_MIN_INSYNC_REPLICAS=2
      - KAFKA_DEFAULT_REPLICATION_FACTOR=3
      - KAFKA_ZOOKEEPER_CONNECT=zookeeper1:2181,zookeeper2:2181,zookeeper3:2181
      # 99 * 1024 * 1024 B
      - KAFKA_MESSAGE_MAX_BYTES=103809024
      - KAFKA_REPLICA_FETCH_MAX_BYTES=103809024 # 99 * 1024 * 1024 B
      - KAFKA_UNCLEAN_LEADER_ELECTION_ENABLE=false
      - KAFKA_LOG_RETENTION_MS=-1
      - KAFKA_HEAP_OPTS=-Xmx256M -Xms128M
    ports:
      - 9092:9092
    extra_hosts:
      - "zookeeper1:192.168.24.201"
      - zookeeper2:192.168.24.202
      - zookeeper3:192.168.24.203
      - kafka1:192.168.24.204
      - kafka2:192.168.24.205
      - kafka3:192.168.24.206
      - kafka4:192.168.24.207
```

- 配置项解释

> 1. Kafka 默认端口为: 9092
> 2. 环境变量:
>   - KAFKA_BROKER_ID
>   - 是一个唯一的非负整数, 可以作为代理 `Broker` 的名字
>   - KAFKA_MIN_INSYNC_REPLICAS
>   - 最小同步备份

> - 该值要小于环境变量 `KAFKA_DEFAULT_REPLICATION_FACTOR` 的值
> - KAFKA_DEFAULT_REPLICATION_FACTOR
>   - 默认同步备份, 该值要小于kafka集群数量
> - KAFKA_ZOOKEEPER_CONNECT
>   - 指向zookeeper节点的集合
> - KAFKA_MESSAGE_MAX_BYTES
>   - 消息的最大字节数
>   - 和配置文件 `configtx.yaml` 中的 `Orderer.BatchSize.AbsoluteMaxBytes` 对应
>   - 由于消息都有头信息, 所以这个值要比计算出的值稍大, `多加1M就足够了`
> - KAFKA_REPLICA_FETCH_MAX_BYTES=103809024
>   - 副本最大字节数, 试图为每个channel获取的消息的字节数
>   - `AbsoluteMaxBytes` < `KAFKA_REPLICA_FETCH_MAX_BYTES` <= `KAFKA_MESSAGE_MAX_BYTES`
> - KAFKA_UNCLEAN_LEADER_ELECTION_ENABLE=false
>   - 非一致性的 Leader 选举
>   - 开启: true
>   - 关闭: false
> - KAFKA_LOG_RETENTION_MS=-1
>   - 对压缩日志保留的最长时间
>   - 这个选项在Kafka中已经默认关闭
> - KAFKA_HEAP_OPTS
>   - 设置堆内存大小, kafka默认为 1G
>   - -Xmx256M -> 允许分配的堆内存
>   - -Xms128M -> 初始分配的堆内存

5.3 各个kafka节点的配置

kafka1 配置

```yaml
# kafka1.yaml
version: '2'

services:

  kafka1:
    container_name: kafka1
    hostname: kafka1
    image: hyperledger/fabric-kafka:latest
    restart: always
    environment:
      # broker.id
      - KAFKA_BROKER_ID=1
      - KAFKA_MIN_INSYNC_REPLICAS=2
      - KAFKA_DEFAULT_REPLICATION_FACTOR=3
      - KAFKA_ZOOKEEPER_CONNECT=zookeeper1:2181,zookeeper2:2181,zookeeper3:2181
      # 100 * 1024 * 1024 B
      - KAFKA_MESSAGE_MAX_BYTES=104857600
      - KAFKA_REPLICA_FETCH_MAX_BYTES=104857600
      - KAFKA_UNCLEAN_LEADER_ELECTION_ENABLE=false
      - KAFKA_LOG_RETENTION_MS=-1
      - KAFKA_HEAP_OPTS=-Xmx512M -Xms256M
    ports:
      - 9092:9092
    extra_hosts:
      - zookeeper1:192.168.247.101
      - zookeeper2:192.168.247.102
      - zookeeper3:192.168.247.103
      - kafka1:192.168.247.201
      - kafka2:192.168.247.202
      - kafka3:192.168.247.203
      - kafka4:192.168.247.204
```

kafka2 配置

```yaml
# kafka2.yaml
version: '2'

services:

  kafka2:
    container_name: kafka2
    hostname: kafka2
    image: hyperledger/fabric-kafka:latest
    restart: always
    environment:
      # broker.id
      - KAFKA_BROKER_ID=2
      - KAFKA_MIN_INSYNC_REPLICAS=2
      - KAFKA_DEFAULT_REPLICATION_FACTOR=3
      - KAFKA_ZOOKEEPER_CONNECT=zookeeper1:2181,zookeeper2:2181,zookeeper3:2181
      # 100 * 1024 * 1024 B
      - KAFKA_MESSAGE_MAX_BYTES=104857600
      - KAFKA_REPLICA_FETCH_MAX_BYTES=104857600
      - KAFKA_UNCLEAN_LEADER_ELECTION_ENABLE=false
      - KAFKA_LOG_RETENTION_MS=-1
      - KAFKA_HEAP_OPTS=-Xmx512M -Xms256M
    ports:
      - 9092:9092
    extra_hosts:
      - zookeeper1:192.168.247.101
      - zookeeper2:192.168.247.102
      - zookeeper3:192.168.247.103
      - kafka1:192.168.247.201
      - kafka2:192.168.247.202
      - kafka3:192.168.247.203
      - kafka4:192.168.247.204
```

kafka3 配置

```yaml
# kafka3.yaml
version: '2'
```

```yaml

services:

  kafka3:
    container_name: kafka3
    hostname: kafka3
    image: hyperledger/fabric-kafka:latest
    restart: always
    environment:
      # broker.id
      - KAFKA_BROKER_ID=3
      - KAFKA_MIN_INSYNC_REPLICAS=2
      - KAFKA_DEFAULT_REPLICATION_FACTOR=3
      - KAFKA_ZOOKEEPER_CONNECT=zookeeper1:2181,zookeeper2:2181,zookeeper3:2181
      # 100 * 1024 * 1024 B
      - KAFKA_MESSAGE_MAX_BYTES=104857600
      - KAFKA_REPLICA_FETCH_MAX_BYTES=104857600
      - KAFKA_UNCLEAN_LEADER_ELECTION_ENABLE=false
      - KAFKA_LOG_RETENTION_MS=-1
      - KAFKA_HEAP_OPTS=-Xmx512M -Xms256M
    ports:
      - 9092:9092
    extra_hosts:
      - zookeeper1:192.168.247.101
      - zookeeper2:192.168.247.102
      - zookeeper3:192.168.247.103
      - kafka1:192.168.247.201
      - kafka2:192.168.247.202
      - kafka3:192.168.247.203
      - kafka4:192.168.247.204
```

kafka4 配置

```yaml
# kafka4.yaml
version: '2'
services:

  kafka4:
    container_name: kafka4
    hostname: kafka4
    image: hyperledger/fabric-kafka:latest
    restart: always
    environment:
      # broker.id
      - KAFKA_BROKER_ID=4
      - KAFKA_MIN_INSYNC_REPLICAS=2
      - KAFKA_DEFAULT_REPLICATION_FACTOR=3
      - KAFKA_ZOOKEEPER_CONNECT=zookeeper1:2181,zookeeper2:2181,zookeeper3:2181
      # 100 * 1024 * 1024 B
      - KAFKA_MESSAGE_MAX_BYTES=104857600
      - KAFKA_REPLICA_FETCH_MAX_BYTES=104857600
      - KAFKA_UNCLEAN_LEADER_ELECTION_ENABLE=false
      - KAFKA_LOG_RETENTION_MS=-1
      - KAFKA_HEAP_OPTS=-Xmx512M -Xms256M
    ports:
      - 9092:9092
    extra_hosts:
      - zookeeper1:192.168.247.101
      - zookeeper2:192.168.247.102
      - zookeeper3:192.168.247.103
      - kafka1:192.168.247.201
      - kafka2:192.168.247.202
      - kafka3:192.168.247.203
      - kafka4:192.168.247.204
```

6. orderer节点设置

6.1 orderer节点配置文件模板

- orderer节点配置文件模板

```
version: '2'

services:

  orderer0.example.com:
    container_name: orderer0.example.com
    image: hyperledger/fabric-orderer:latest
    environment:
      - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=aberic_default
      - ORDERER_GENERAL_LOGLEVEL=debug
      - ORDERER_GENERAL_LISTENADDRESS=0.0.0.0
      - ORDERER_GENERAL_LISTENPORT=7050
      - ORDERER_GENERAL_GENESISMETHOD=file
      - ORDERER_GENERAL_GENESISFILE=/var/hyperledger/orderer/orderer.genesis.block
      - ORDERER_GENERAL_LOCALMSPID=OrdererMSP
      - ORDERER_GENERAL_LOCALMSPDIR=/var/hyperledger/orderer/msp
      # enabled TLS
      - ORDERER_GENERAL_TLS_ENABLED=false
      - ORDERER_GENERAL_TLS_PRIVATEKEY=/var/hyperledger/orderer/tls/server.key
      - ORDERER_GENERAL_TLS_CERTIFICATE=/var/hyperledger/orderer/tls/server.crt
      - ORDERER_GENERAL_TLS_ROOTCAS=[/var/hyperledger/orderer/tls/ca.crt]

      - ORDERER_KAFKA_RETRY_LONGINTERVAL=10s
      - ORDERER_KAFKA_RETRY_LONGTOTAL=100s
      - ORDERER_KAFKA_RETRY_SHORTINTERVAL=1s
      - ORDERER_KAFKA_RETRY_SHORTTOTAL=30s
      - ORDERER_KAFKA_VERBOSE=true
      - ORDERER_KAFKA_BROKERS=
[192.168.24.204:9092,192.168.24.205:9092,192.168.24.206:9092,192.168.24.207:9092]
    working_dir: /opt/gopath/src/github.com/hyperledger/fabric
    command: orderer
    volumes:
        - ./channel-artifacts/genesis.block:/var/hyperledger/orderer/orderer.genesis.block
        - ./crypto-config/ordererOrganizations/example.com/orderers/orderer0.example.com/msp:/var/hyperledger/orderer/msp
        - ./crypto-config/ordererOrganizations/example.com/orderers/orderer0.example.com/tls/:/var/hyperledger/orderer/tls
    networks:
      default:
        aliases:
          - aberic
    ports:
      - 7050:7050
    extra_hosts:
      - kafka1:192.168.24.204
      - kafka2:192.168.24.205
      - kafka3:192.168.24.206
      - kafka4:192.168.24.207
```

- 细节解释

> 1. 环境变量
>   - ORDERER_KAFKA_RETRY_LONGINTERVAL
>   - 每隔多长时间进行一次重试, 单位:秒
>   - ORDERER_KAFKA_RETRY_LONGTOTAL
>   - 总共重试的时长, 单位: 秒
>   - ORDERER_KAFKA_RETRY_SHORTINTERVAL
>   - 每隔多长时间进行一次重试, 单位:秒
>   - ORDERER_KAFKA_RETRY_SHORTTOTAL
>   - 总共重试的时长, 单位: 秒
>   - ORDERER_KAFKA_VERBOSE
>   - 启用日志与kafka进行交互, 启用: true, 不启用: false
>   - ORDERER_KAFKA_BROKERS
>   - 指向kafka节点的集合
> 2. 关于重试的时长
>   - 先使用 `ORDERER_KAFKA_RETRY_SHORTINTERVAL` 进行重连, 重连的总时长为 `ORDERER_KAFKA_RETRY_SHORTTOTAL`
>   - 如果上述步骤没有重连成功, 使用 `ORDERER_KAFKA_RETRY_LONGINTERVAL` 进行重连, 重连的总时长为 `ORDERER_KAFKA_RETRY_LONGTOTAL`

6.3 orderer各节点的配置

orderer0配置

```
# orderer0.yaml
version: '2'
```

```yaml

services:

  orderer0.test.com:
    container_name: orderer0.test.com
    image: hyperledger/fabric-orderer:latest
    environment:
      - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=kafka_default
      - ORDERER_GENERAL_LOGLEVEL=debug
      - ORDERER_GENERAL_LISTENADDRESS=0.0.0.0
      - ORDERER_GENERAL_LISTENPORT=7050
      - ORDERER_GENERAL_GENESISMETHOD=file
      - ORDERER_GENERAL_GENESISFILE=/var/hyperledger/orderer/orderer.genesis.block
      - ORDERER_GENERAL_LOCALMSPID=OrdererMSP
      - ORDERER_GENERAL_LOCALMSPDIR=/var/hyperledger/orderer/msp
      # enabled TLS
      - ORDERER_GENERAL_TLS_ENABLED=false
      - ORDERER_GENERAL_TLS_PRIVATEKEY=/var/hyperledger/orderer/tls/server.key
      - ORDERER_GENERAL_TLS_CERTIFICATE=/var/hyperledger/orderer/tls/server.crt
      - ORDERER_GENERAL_TLS_ROOTCAS=[/var/hyperledger/orderer/tls/ca.crt]

      - ORDERER_KAFKA_RETRY_LONGINTERVAL=10s
      - ORDERER_KAFKA_RETRY_LONGTOTAL=100s
      - ORDERER_KAFKA_RETRY_SHORTINTERVAL=1s
      - ORDERER_KAFKA_RETRY_SHORTTOTAL=30s
      - ORDERER_KAFKA_VERBOSE=true
      - ORDERER_KAFKA_BROKERS=[192.168.247.201:9092,192.168.247.202:9092,192.168.247.203:9092,192.168.247.204:9092]
    working_dir: /opt/gopath/src/github.com/hyperledger/fabric
    command: orderer
    volumes:
      - ./channel-artifacts/genesis.block:/var/hyperledger/orderer/orderer.genesis.block
      - ./crypto-config/ordererOrganizations/test.com/orderers/orderer0.test.com/msp:/var/hyperledger/orderer/msp
      - ./crypto-config/ordererOrganizations/test.com/orderers/orderer0.test.com/tls/:/var/hyperledger/orderer/tls
    networks:
    default:
      aliases:
        - kafka
    ports:
      - 7050:7050
    extra_hosts:
      - kafka1:192.168.247.201
      - kafka2:192.168.247.202
      - kafka3:192.168.247.203
      - kafka4:192.168.247.204
```

orderer1配置

```yaml
# orderer1.yaml
```

```yaml
version: '2'

services:

  orderer1.test.com:
    container_name: orderer1.test.com
    image: hyperledger/fabric-orderer:latest
    environment:
      - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=kafka_default
      - ORDERER_GENERAL_LOGLEVEL=debug
      - ORDERER_GENERAL_LISTENADDRESS=0.0.0.0
      - ORDERER_GENERAL_LISTENPORT=7050
      - ORDERER_GENERAL_GENESISMETHOD=file
      - ORDERER_GENERAL_GENESISFILE=/var/hyperledger/orderer/orderer.genesis.block
      - ORDERER_GENERAL_LOCALMSPID=OrdererMSP
      - ORDERER_GENERAL_LOCALMSPDIR=/var/hyperledger/orderer/msp
      # enabled TLS
      - ORDERER_GENERAL_TLS_ENABLED=false
      - ORDERER_GENERAL_TLS_PRIVATEKEY=/var/hyperledger/orderer/tls/server.key
      - ORDERER_GENERAL_TLS_CERTIFICATE=/var/hyperledger/orderer/tls/server.crt
      - ORDERER_GENERAL_TLS_ROOTCAS=[/var/hyperledger/orderer/tls/ca.crt]

      - ORDERER_KAFKA_RETRY_LONGINTERVAL=10s
      - ORDERER_KAFKA_RETRY_LONGTOTAL=100s
      - ORDERER_KAFKA_RETRY_SHORTINTERVAL=1s
      - ORDERER_KAFKA_RETRY_SHORTTOTAL=30s
      - ORDERER_KAFKA_VERBOSE=true
      - ORDERER_KAFKA_BROKERS=[192.168.247.201:9092,192.168.247.202:9092,192.168.247.203:9092,192.168.247.204:9092]
    working_dir: /opt/gopath/src/github.com/hyperledger/fabric
    command: orderer
    volumes:
      - ./channel-artifacts/genesis.block:/var/hyperledger/orderer/orderer.genesis.block
      - ./crypto-config/ordererOrganizations/test.com/orderers/orderer1.test.com/msp:/var/hyperledger/orderer/msp
      - ./crypto-config/ordererOrganizations/test.com/orderers/orderer1.test.com/tls/:/var/hyperledger/orderer/tls
    networks:
    default:
      aliases:
        - kafka
    ports:
      - 7050:7050
    extra_hosts:
      - kafka1:192.168.247.201
      - kafka2:192.168.247.202
      - kafka3:192.168.247.203
      - kafka4:192.168.247.204
```

orderer2配置

```yaml
# orderer2.yaml
version: '2'

services:

  orderer2.test.com:
    container_name: orderer2.test.com
    image: hyperledger/fabric-orderer:latest
    environment:
      - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=kafka_default
      - ORDERER_GENERAL_LOGLEVEL=debug
      - ORDERER_GENERAL_LISTENADDRESS=0.0.0.0
      - ORDERER_GENERAL_LISTENPORT=7050
      - ORDERER_GENERAL_GENESISMETHOD=file
      - ORDERER_GENERAL_GENESISFILE=/var/hyperledger/orderer/orderer.genesis.block
      - ORDERER_GENERAL_LOCALMSPID=OrdererMSP
      - ORDERER_GENERAL_LOCALMSPDIR=/var/hyperledger/orderer/msp
      # enabled TLS
      - ORDERER_GENERAL_TLS_ENABLED=false
      - ORDERER_GENERAL_TLS_PRIVATEKEY=/var/hyperledger/orderer/tls/server.key
      - ORDERER_GENERAL_TLS_CERTIFICATE=/var/hyperledger/orderer/tls/server.crt
      - ORDERER_GENERAL_TLS_ROOTCAS=[/var/hyperledger/orderer/tls/ca.crt]

      - ORDERER_KAFKA_RETRY_LONGINTERVAL=10s
      - ORDERER_KAFKA_RETRY_LONGTOTAL=100s
      - ORDERER_KAFKA_RETRY_SHORTINTERVAL=1s
      - ORDERER_KAFKA_RETRY_SHORTTOTAL=30s
      - ORDERER_KAFKA_VERBOSE=true
      - ORDERER_KAFKA_BROKERS=
[192.168.247.201:9092,192.168.247.202:9092,192.168.247.203:9092,192.168.247.204:9092]
    working_dir: /opt/gopath/src/github.com/hyperledger/fabric
    command: orderer
    volumes:
      - ./channel-artifacts/genesis.block:/var/hyperledger/orderer/orderer.genesis.block
      - ./crypto-
config/ordererOrganizations/test.com/orderers/orderer2.test.com/msp:/var/hyperledger/order
er/msp
      - ./crypto-
config/ordererOrganizations/test.com/orderers/orderer2.test.com/tls/:/var/hyperledger/orde
rer/tls
    networks:
    default:
      aliases:
        - kafka
    ports:
      - 7050:7050
    extra_hosts:
      - kafka1:192.168.247.201
      - kafka2:192.168.247.202
      - kafka3:192.168.247.203
      - kafka4:192.168.247.204
```

7. 启动集群

> Kafka集群的启动顺序是这样的: 先启动 `Zookeeper` 集群, 随后启动 `Kafka` 集群, 最后启动 `Orderer` 排序服务器集群。由于peer节点只能和集群中 `orderer` 节点进行通信, 所以不管是使用solo集群还是kafka集群对peer都是没有影响的, 所以当我们的 `kafka` 集群顺利启动之后, 就可以启动对应的 `Peer` 节点了。

7.1 启动Zookeeper集群

- zookeeper1:192.168.247.101

```
$ cd ~/kafka
# 将写好的 zookeeper1.yaml 配置文件放到该目录下, 通过 docker-compose 启动容器
# 该命令可以不加 -d 参数, 这样就能看到当前 zookeeper 服务器启动的情况了
$ docker-compose -f zookeeper1.yaml up
```

- zookeeper2:192.168.247.102

```
$ cd ~/kafka
# 将写好的 zookeeper2.yaml 配置文件放到该目录下, 通过 docker-compose 启动容器
# 该命令可以不加 -d 参数, 这样就能看到当前 zookeeper 服务器启动的情况了
$ docker-compose -f zookeeper2.yaml up
```

- zookeeper3:192.168.247.103

```
$ cd ~/kafka
# 将写好的 zookeeper3.yaml 配置文件放到该目录下, 通过 docker-compose 启动容器
# 该命令可以不加 -d 参数, 这样就能看到当前 zookeeper 服务器启动的情况了
$ docker-compose -f zookeeper3.yaml up
```

7.2 启动Kafka集群

- kafka1:192.168.247.201

```
$ cd ~/kafka
# 将写好的 kafka1.yaml 配置文件放到该目录下, 通过 docker-compose 启动容器
# 该命令可以不加 -d 参数, 这样就能看到当前 kafka 服务器启动的情况了
$ docker-compose -f kafka1.yaml up
```

- kafka2:192.168.247.202

```
$ cd ~/kafka
# 将写好的 kafka2.yaml 配置文件放到该目录下, 通过 docker-compose 启动容器
$ docker-compose -f kafka2.yaml up -d
```

- kafka3:192.168.247.203

```
$ cd ~/kafka
# 将写好的 kafka3.yaml 配置文件放到该目录下, 通过 docker-compose 启动容器
$ docker-compose -f kafka3.yaml up -d
```

- kafka4:192.168.247.204

```
$ cd ~/kafka
# 将写好的 kafka4.yaml 配置文件放到该目录下，通过 docker-compose 启动容器
$ docker-compose -f kafka4.yaml up
```

7.3 启动Orderer集群

- orderer0:192.168.247.91

```
$ cd ~/kafka
# 假设生成证书和通道创始块文件操作是在当前 orderer0 上完成的，那么应该在当前 kafka 工作目录下
$ tree ./ -L 1
./
├── channel-artifacts
├── configtx.yaml
├── crypto-config
└── crypto-config.yaml
# 将写好的 orderer0.yaml 配置文件放到该目录下，通过 docker-compose 启动容器
$ docker-compose -f orderer0.yaml up -d
```

- orderer1:192.168.247.92

```
# 将生成的 证书文件目录 和 通道创始块 文件目录拷贝到当前主机的 ~/kafka目录中
$ cd ~/kafka
# 创建子目录 crypto-config
$ mkdir crypto-config
# 远程拷贝
$ scp -f itcast@192.168.247.91:/home/itcast/kafka/crypto-config/ordererOrganizations
./crypto-config
# # 将写好的 orderer1.yaml 配置文件放到该目录下，通过 docker-compose 启动容器
$ docker-compose -f orderer1.yaml up -d
```

- orderer2:192.168.247.93

```
# 将生成的 证书文件目录 和 通道创始块 文件目录拷贝到当前主机的 ~/kafka目录中
$ cd ~/kafka
# 创建子目录 crypto-config
$ mkdir crypto-config
# 远程拷贝
$ scp -f itcast@192.168.247.91:/home/itcast/kafka/crypto-config/ordererOrganizations
./crypto-config
# # 将写好的 orderer3.yaml 配置文件放到该目录下，通过 docker-compose 启动容器
$ docker-compose -f orderer3.yaml up -d
```

7.4 启动Peer集群

> 关于 Peer 节点的部署和操作和 Solo 多机多节点部署的方式是完全一样的, 在此不再阐述, 请翻阅相关文档。