Erkan Şirin Follow
Jul 26 · 8 min read · Listen

Multi-node Zookeeper-less Kafka Cluster Setup with Docker

gradually decrease and the use of Zookeeper-less Kafka will become widespread after Zookeeper-less Kafka is production ready. In this article, we will install Kafka clusters without Zookeeper with the help of Docker compose and thus we will have the chance to try 3-node Zookeeper-less Kafka.

What was the zookeeper doing?

Zookeeper is a coordination and configuration service. Distributed systems need constant coordination between nodes and instant sharing of up-to-date metadata. Distributed systems such as RabbitMQ, Elasticsearch, and Cassandra have solved this internally without Zookeeper. However, Kafka preferred Zookeeper at the beginning.

Zookeeper is primarily used to monitor node status in Kafka Cluster and to keep a list of topics and messages. We can list its five basic tasks as follows[1]:

1. Controller Selection. The controller is the broker that follows the leader/follower relationship for all partitions. Zookeeper ensures a Broker acting as a controller within the cluster.
2. Cluster membership: Maintains an up-to-date list of Brokers working in the Cluster. Admission of brokers to the cluster.
3. Topic configuration: Zookeeper functions such as a list of available topics,

4. Access Control List (ACLs): Zookeeper also maintains ACLs for all topics. This includes who or what is allowed to read/write each topic, the consumer group list, the group members, and the latest offset each consumer group gets from each partition.

5. Quotas: Zookeeper knows how much data each client is allowed to read/write.

Why did they remove Zookeeper?

You needed at least 3 Zookeeper nodes for a Kafka cluster. This is a serious cost in every aspect. Security, resources, disk, backup, maintenance, update, monitoring, etc etc. lots of extra trouble. Therefore, they have been removed both to relieve these burdens, to increase performance, and above all to be fully independent and self-sufficient.

Who is doing what Zookeeper did now?

Kafka does itself internally what Zookeeper does, using the KRaft consensus protocol with its own Controller nodes. For detailed information, you can refer here[2].

In the Zookeeper-less Kafka world, there are two different roles for nodes: Controller and Broker. Each node in the cluster can have one or both roles. All accept and process requests from clients (consumer/producer) just like Zookeeper before.

Is Zookeeper used in versions after Kafka 2.8?

Yes, it is used. Currently using the latest version 3.2.0 and Zookeeper is used. Zookeeper-less Kafka for production is not ready yet. 2.8.0 is the first version that Kafka can be used without Zookeeper. It doesn't mean Zookeeper has totally gone and can't be used.

Environment Information

Let's jump into practice part. I have conducted my work ina an environment like these:
Operating system: CentOS7

Docker: Docker Engine — Community Version: 20.10.15

Kafka: 3.2.0-Scala: 2.12

Number of nodes: 3 container

Kafka Client and Cluster UUID

commands from **/tmp/kafka/kafka_2.12–3.2.0**.

```
mkdir /tmp/kafka
curl “https://archive.apache.org/dist/kafka/3.2.0/kafka_2.12-
3.2.0.tgz" -o /tmp/kafka/kafka.tgz
cd /tmp/kafka
tar -xzf kafka.tgz
cd kafka_2.12-3.2.0
```

Now generate an uuid

```
./bin/kafka-storage.sh random-uuid
```

The result will be something like the following:

```
EP6hyiddQNW5FPrAvR9kWw
```

We will use this value in **start-kafka.sh**. It will be the cluster id number.

Project directory tree structure

```
tree -L 3
.
├── config
│   ├── kafka1
│   │   └── server.properties
│   ├── kafka2
│   │   └── server.properties
│   └── kafka3
│   └── server.properties
├── data
│   ├── kafka1
│   ├── kafka2
│   └── kafka3
├── docker-compose.yaml
├── Dockerfile
└── start-kafka.sh
```

Docker compose file

Please use the docker-compose code in repo.

```
version: "3.8"
networks:
 kafka-net:
 ipam:
 config:
```

```
build:
context: .
ports:
– "9092:9092"
networks:
kafka-net:
ipv4_address: 172.18.0.11
volumes:
– ./config/kafka1/server.properties:/kafka/config/server.properties
– ./data/kafka1/:/data/kafka/
kafka2:
container_name: kafka2
image: erkansirin78/kafka:3.2.0
build:
context: .
ports:
– "9292:9092"
networks:
kafka-net:
ipv4_address: 172.18.0.12
volumes:
– ./config/kafka2/server.properties:/kafka/config/server.properties
– ./data/kafka2/:/data/kafka/
kafka3:
container_name: kafka3
image: erkansirin78/kafka:3.2.0
build:
context: .
ports:
– "9392:9092"
networks:
kafka-net:
ipv4_address: 172.18.0.13
volumes:
– ./config/kafka3/server.properties:/kafka/config/server.properties
– ./data/kafka3/:/data/kafka/
```

In order to use the IP number in **server.properties**, we created a network and gave a static IP for each container. We will discuss the contents of the properties file below.

Dockerfile

```
FROM openjdk:11.0.10-jre-buster
RUN apt-get update && \
 apt-get install -y curl
ENV KAFKA_VERSION 3.2.0
ENV SCALA_VERSION 2.12
RUN mkdir /tmp/kafka && \
 curl
“https://archive.apache.org/dist/kafka/${KAFKA_VERSION}/kafka_${SCALA_
VERSION}-${KAFKA_VERSION}.tgz" \
 -o /tmp/kafka/kafka.tgz && \
 mkdir /kafka && cd /kafka && \
 tar -xvzf /tmp/kafka/kafka.tgz --strip 1
RUN mkdir -p /data/kafka
COPY start-kafka.sh /usr/bin
RUN chmod +x /usr/bin/start-kafka.sh
CMD [“start-kafka.sh”]
```

FROM openjdk:11.0.10-jre-buster: We are using the base image Java11.

RUN apt-get update …: We are updating the operating system packages in the

ENV instructions: Here we specify the Kafka and Scala versions we want to use.

RUN mkdir /tmp/kafka …: We download and open Kafka in accordance with the version we specified.

RUN mkdir -p /data/kafka: We are creating directory for Kafka commit log. Kafka will store its data here. Not the application log is usually confused.

COPY start-kafka.sh /usr/bin: Copy the script we created to start Kafka.

RUN chmod +x /usr/bin/start-kafka.sh: We give execute permission to the script.

CMD ["start-kafka.sh"]: Let Kafka work.

Kafka run script

```
#!/bin/bash

/kafka/bin/kafka-storage.sh format --config
/kafka/config/server.properties --cluster-id 'EP6hyiddQNW5FPrAvR9kWw'
```

/kafka/bin/kafka-storage.sh…: Formatting the directory where the data will be stored. The — ignore-formatted option ensures that if it has already been formatted, in subsequent runs, it will not get an error, it will be ignored.

/kafka/bin/kafka-server-start.sh…: Running Kafka.

server.properties configuration file

There is a different configuration file for each broker. Let's explain the important ones, out of default settings, and the different ones for each broker. Most of the configurations are the same as the ZK-Kafka setup. There are just a few differences.

```
process.roles=broker,controller

node.id=1

controller.quorum.voters=1@kafka1:9093,2@kafka2:9093,3@kafka3:9093

listeners=PLAINTEXT://:9192,CONTROLLER://:9093,LISTENER_DOCKER_EXTERNA
L://:9092

advertised.listeners=PLAINTEXT://kafka1:9092,LISTENER_DOCKER_EXTERNAL:
//172.18.0.11:9092

listener.security.protocol.map=CONTROLLER:PLAINTEXT,PLAINTEXT:PLAINTEX
T,SSL:SSL,SASL_PLAINTEXT:SASL_PLAINTEXT,SASL_SSL:SASL_SSL,LISTENER_DOC
KER_EXTERNAL:PLAINTEXT
```

process.roles: What will the node's role in the cluster? Controllers are needed running in KRaft mode without Zookeeper. We used both the broker and the controller value for all 3 nodes because we only have 3 nodes. For larger clusters, some nodes may be just controllers.

node.id=1: Each broker must have a different id number.

controller.quorum.voters: Addresses in the format nodeid@servername:port to vote for consensus. Here is port 9093 as controller jobs are returning from port 9093. This setting works like the old zookeeper.connect setting.

listeners: We added the LISTENER_DOCKER_EXTERNAL://:9092 record. Because we will use it in advertised.listeners.

advertised.listeners: We've added LISTENER_DOCKER_EXTERNAL://172.18.0.11:9092 here. The reason for this is that if there is client access outside of the Docker network such as our host machine(CentOS), it will not get an error. The server name kafka1 here only works for containers accessing each other within the docker network. We cannot access brokers with this name from outside the container.

it inside the listeners.

log.dirs: The directory where the data will be stored.

num.partitions: Default partitions value when the topic is created.

delete.topic.enable: If we want to delete the topic we can delete it.

kafka2 server.properties

```
process.roles=broker,controller
node.id=2
controller.quorum.voters=1@kafka1:9093,2@kafka2:9093,3@kafka3:9093
listeners=PLAINTEXT://:9192,CONTROLLER://:9093,LISTENER_DOCKER_EXTERNA
L://:9092
advertised.listeners=PLAINTEXT://kafka2:9192,LISTENER_DOCKER_EXTERNAL:
//172.18.0.12:9092
listener.security.protocol.map=CONTROLLER:PLAINTEXT,PLAINTEXT:PLAINTEX
T,SSL:SSL,SASL_PLAINTEXT:SASL_PLAINTEXT,SASL_SSL:SASL_SSL,LISTENER_DOC
KER_EXTERNAL:PLAINTEXT
log.dirs=/data/kafka
num.partitions=3
delete.topic.enable=true
```

kafka3 server.properties

```
controller.quorum.voters=1@kafka1:9093,2@kafka2:9093,3@kafka3:9093
listeners=PLAINTEXT://:9192,CONTROLLER://:9093,LISTENER_DOCKER_EXTERNA
L://:9092
advertised.listeners=PLAINTEXT://kafka3:9192,LISTENER_DOCKER_EXTERNAL:
//172.18.0.13:9092
listener.security.protocol.map=CONTROLLER:PLAINTEXT,PLAINTEXT:PLAINTEX
T,SSL:SSL,SASL_PLAINTEXT:SASL_PLAINTEXT,SASL_SSL:SASL_SSL,LISTENER_DOC
KER_EXTERNAL:PLAINTEXT
log.dirs=/data/kafka
num.partitions=3
delete.topic.enable=true
```

Start Cluster

After opening/extracting what you have downloaded from the repo and changing directory, if you created the data directories above, all you have to do is run docker-compose. If not, create it using the simple loop below. Because empty directories will not appear in Github repo.

```
for i in {1..3}; do mkdir data/kafka${i}; done
```

Docker-compose up

Get unlimited access

Then follow the log for one node. After a certain time, the logs will become stable.

```
docker logs -f kafka1
```

Expected output (just the ending part) will be lie this:

```
…
…
…
 zookeeper.ssl.ocsp.enable = false
 zookeeper.ssl.protocol = TLSv1.2
 zookeeper.ssl.truststore.location = null
 zookeeper.ssl.truststore.password = null
 zookeeper.ssl.truststore.type = null
 (kafka.server.KafkaConfig)
[2022-07-21 02:35:17,448] INFO [SocketServer listenerType=BROKER,
nodeId=1] Starting socket server acceptors and processors
(kafka.network.SocketServer)
[2022-07-21 02:35:17,460] INFO [SocketServer listenerType=BROKER,
nodeId=1] Started data-plane acceptor and processor(s) for endpoint :
ListenerName(PLAINTEXT) (kafka.network.SocketServer)
[2022-07-21 02:35:17,464] INFO [SocketServer listenerType=BROKER,
nodeId=1] Started data-plane acceptor and processor(s) for endpoint :
ListenerName(LISTENER_DOCKER_EXTERNAL) (kafka.network.SocketServer)
[2022-07-21 02:35:17,465] INFO [SocketServer listenerType=BROKER,
nodeId=1] Started socket server acceptors and processors
```

```
[2022-07-21 02:35:17,469] INFO Kafka commitId: 38103ffaa962ef50
(org.apache.kafka.common.utils.AppInfoParser)
[2022-07-21 02:35:17,469] INFO Kafka startTimeMs: 1658370917466
(org.apache.kafka.common.utils.AppInfoParser)
[2022-07-21 02:35:17,476] INFO Kafka Server started
(kafka.server.KafkaRaftServer)
[2022-07-21 02:35:17,479] INFO [BrokerLifecycleManager id=1] The
broker is in RECOVERY. (kafka.server.BrokerLifecycleManager)
[2022-07-21 02:35:17,643] INFO [Controller 1] Unfenced broker:
UnfenceBrokerRecord(id=3, epoch=0)
(org.apache.kafka.controller.ClusterControlManager)
[2022-07-21 02:35:19,531] INFO [BrokerLifecycleManager id=1] The
broker has been unfenced. Transitioning from RECOVERY to RUNNING.
(kafka.server.BrokerLifecycleManager)
[2022-07-21 02:35:20,043] INFO [Controller 1] Unfenced broker:
UnfenceBrokerRecord(id=2, epoch=2)
(org.apache.kafka.controller.ClusterControlManager)
[2022-07-21 02:35:20,044] INFO [Controller 1] Unfenced broker:
UnfenceBrokerRecord(id=1, epoch=3)
(org.apache.kafka.controller.ClusterControlManager)
```

List containers:

```
/kafka/bin/kafka-server-start.sh

docker-compose ps
NAME COMMAND SERVICE STATUS PORTS
kafka1 “start-kafka.sh” kafka1 running 0.0.0.0:9092->9092/tcp,
:::9092->9092/tcp
kafka2 “start-kafka.sh” kafka2 running 0.0.0.0:9292->9092/tcp,
:::9292->9092/tcp
```

Kafka Client Examples

Don't forget to change directory to `cd /tmp/kafka/kafka_2.12-3.2.0/`

List Nodes (Broker&Controller)

```
./bin/kafka-broker-api-versions.sh – bootstrap-server localhost:9092 |
awk '/id/{print $1}'

172.18.0.12:9092
172.18.0.13:9092
172.18.0.11:9092
```

Create Topic

```
./bin/kafka-topics.sh \
--bootstrap-server localhost:9092 \
--create --topic test1 \
--replication-factor 3 --partitions 5
```

Output: `Created topic test1.`

Output: `test1`

Produce message with console-producer

Open two different terminals and run the consumer in one and the producer in the other and send messages to test1 topic.

```
./bin/kafka-console-producer.sh — bootstrap-server localhost:9092 —
topic test1
>test producer
>Hello there
```

Consume message with console-consumer

```
./bin/kafka-console-consumer.sh — bootstrap-server localhost:9092 —
topic test1
test producer
Hello there
```

Reset

```
sudo rm -rf data/kafka1/*
sudo rm -rf data/kafka2/*
sudo rm -rf data/kafka3/*
```

To try Zookeeper-less Kafka, we have set up a Kafka cluster using docker. Until we meet in another post, bye…

References

1. https://dattell.com/data-architecture-blog/what-is-zookeeper-how-does-it-support-kafka
2. https://www.confluent.io/blog/kafka-without-zookeeper-a-sneak-peek/
3. https://kafka.apache.org/documentation/