Kafka Docker: Run Multiple Kafka Brokers and ZooKeeper Services in Docker

To handle loads more easily, set up a multi-node Kafka cluster on Docker

DLT Labs Follow
Nov 25, 2019 · 5 min read ★

Photo by JJ Ying on Unsplash

Apache Kafka is a distributed streaming platform which has the capability to publish and subscribe to streams of records, store streams of records for fault-tolerant handling,

and process streams of records.

In general, there are two broad uses of Kafka:

- Building real-time streaming data pipelines that reliably get data between systems or applications
- Building real-time streaming applications that transform or react to the streams of data.

In this piece, we set up a multi-node Kafka cluster on Docker because multi-node Kafka clusters handle loads very easily. This piece should be very helpful for those users who want to process lots of data.

To learn more about Kafka go to the link: http://kafka.apache.org/

Take a look at the following illustration. It shows the cluster diagram of Kafka.

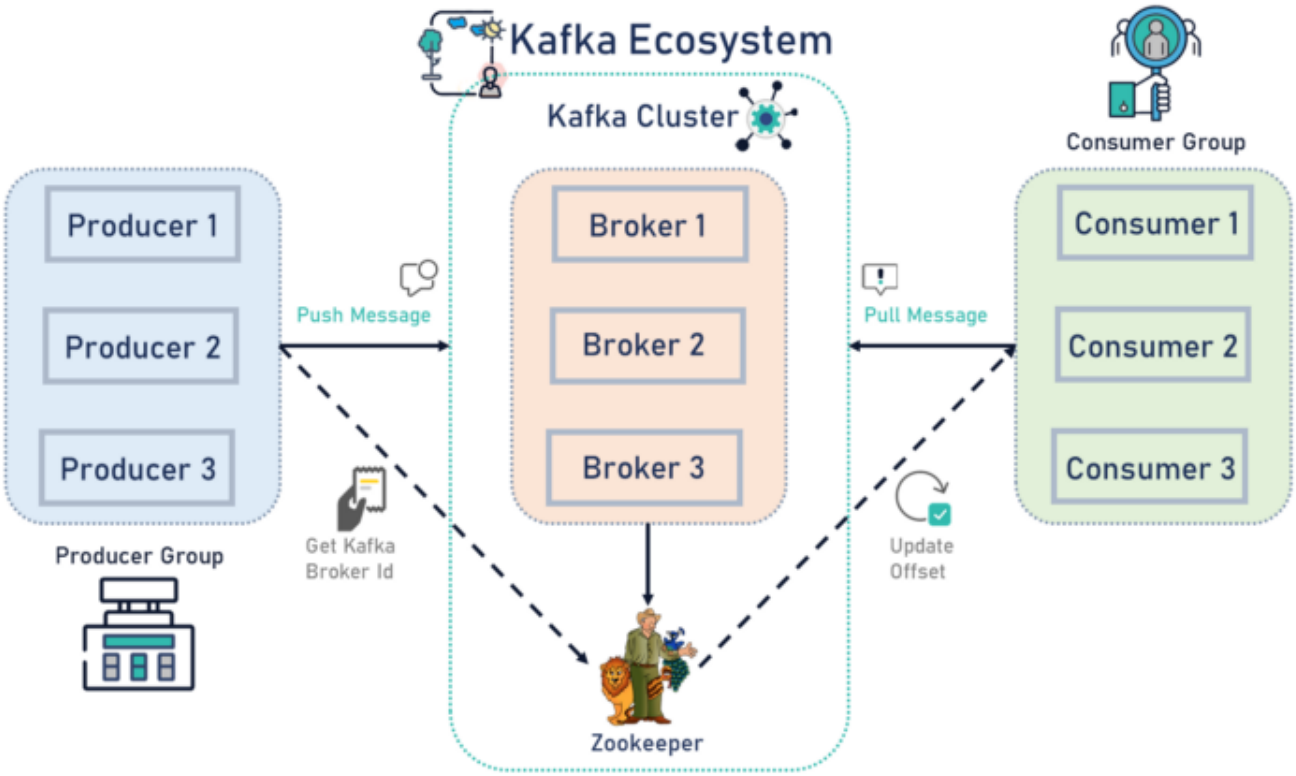

Image source: DLT Labs

Pre-Requisites

1. Install Docker: https://www.docker.com/get-started#h_installation
2. Install Docker Compose: https://docs.docker.com/compose/install/

Set up a three-node Kafka cluster

In a three-node Kafka cluster, we will run three Kafka brokers with three Apache ZooKeeper services and test our setup in multiple steps.

docker-compose file content

The code below is responsible for setting up a Kafka cluster with three Kafka brokers and three ZooKeeper services. Save this code in a `.yml` file where the user gets access from CLI. Then follow these further steps:

```
version: '2.1'
services:
 zookeeper-1:
   image: confluentinc/cp-zookeeper:latest
   environment:
     ZOOKEEPER_SERVER_ID: 1
     ZOOKEEPER_CLIENT_PORT: 22181
     ZOOKEEPER_TICK_TIME: 2000
     ZOOKEEPER_INIT_LIMIT: 5
     ZOOKEEPER_SYNC_LIMIT: 2
     ZOOKEEPER_SERVERS:
localhost:22888:23888;localhost:32888:33888;localhost:42888:43888
   network_mode: host
   extra_hosts:
     - "moby:127.0.0.1"

 zookeeper-2:
   image: confluentinc/cp-zookeeper:latest
   environment:
     ZOOKEEPER_SERVER_ID: 2
     ZOOKEEPER_CLIENT_PORT: 32181
     ZOOKEEPER_TICK_TIME: 2000
     ZOOKEEPER_INIT_LIMIT: 5
     ZOOKEEPER_SYNC_LIMIT: 2
     ZOOKEEPER_SERVERS:
localhost:22888:23888;localhost:32888:33888;localhost:42888:43888
   network_mode: host
   extra_hosts:
     - "moby:127.0.0.1"
```

```
 zookeeper-3:
   image: confluentinc/cp-zookeeper:latest
   environment:
     ZOOKEEPER_SERVER_ID: 3
     ZOOKEEPER_CLIENT_PORT: 42181
     ZOOKEEPER_TICK_TIME: 2000
     ZOOKEEPER_INIT_LIMIT: 5
     ZOOKEEPER_SYNC_LIMIT: 2
     ZOOKEEPER_SERVERS:
localhost:22888:23888;localhost:32888:33888;localhost:42888:43888
   network_mode: host
   extra_hosts:
     — “moby:127.0.0.1”

 kafka-1:
   image: confluentinc/cp-kafka:latest
   network_mode: host
   depends_on:
     — zookeeper-1
     — zookeeper-2
     — zookeeper-3
   environment:
     KAFKA_BROKER_ID: 1
     KAFKA_ZOOKEEPER_CONNECT:
localhost:22181,localhost:32181,localhost:42181
     KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://localhost:19092
   ports:
     — “19092:19092”
   extra_hosts:
     — “moby:127.0.0.1”

 kafka-2:
   image: confluentinc/cp-kafka:latest
   network_mode: host
   depends_on:
     — zookeeper-1
     — zookeeper-2
     — zookeeper-3
   environment:
     KAFKA_BROKER_ID: 2
     KAFKA_ZOOKEEPER_CONNECT:
localhost:22181,localhost:32181,localhost:42181
     KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://localhost:29092
   ports:
     — “29092:29092”
   extra_hosts:
     — “moby:127.0.0.1”

 kafka-3:
   image: confluentinc/cp-kafka:latest
   network_mode: host
   depends_on:
```

```
      - zookeeper-1
      - zookeeper-2
      - zookeeper-3
    environment:
      KAFKA_BROKER_ID: 3
      KAFKA_ZOOKEEPER_CONNECT:
localhost:22181,localhost:32181,localhost:42181
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://localhost:39092
    ports:
      - "39092:39092"
    extra_hosts:
      - "moby:127.0.0.1"
```

In the above `docker-compose` file, we have three ZooKeeper services and three Kafka brokers with different IDs and their environment setup. In ZooKeeper we use several environments, which are as follows:

- `ZOOKEEPER_SERVER_ID` — Unique server ID for all ZooKeeper services
- `ZOOKEEPER_CLIENT_PORT` — The port to listen for client connections
- `ZOOKEEPER_TICK_TIME` — The basic time unit in milliseconds used by ZooKeeper. This is used to do heartbeats, and the minimum session timeout will be twice the `tickTime`.
- `ZOOKEEPER_INIT_LIMIT` — `InitLimit` is the timeout duration ZooKeeper uses to limit the length of time the ZooKeeper servers in quorum have to connect to a leader. The entry `syncLimit` limits how far out-of-date a server can be from a leader.

With both of these timeouts, you specify the unit of time using `tickTime`. In this example, the timeout for `initLimit` is five ticks at 2000 milliseconds a tick, or ten seconds.

In multi-node Kafka cluster setup, when a message comes in, ZooKeeper will decide which Kafka broker handles the message*;* because of this, every Kafka broker depends upon a ZooKeeper service, which is a nine-step process:

Step 1.

Start ZooKeeper and Kafka using the Docker Compose Up command with detached mode.

```
docker-compose -f <docker-compose_file_name> up -d
```

Step 2.

In another terminal window, go to the same directory. Before we move on, let's make sure the services are up and running:

```
docker ps
```

Step 3.

Check the ZooKeeper logs to verify that ZooKeeper is healthy. For example, for service `zookeeper-1`:

```
docker logs <zookeeper-1_containerId>
```

Repeat this step to verify the rest of the ZooKeeper containers.

Step 4.

Verify that the ZooKeeper ensemble is ready:

```
docker run --net=host --rm confluentinc/cp-zookeeper:latest bash -c
“echo stat | nc localhost <ZOOKEEPER_CLIENT_PORT> | grep Mode”
```

Repeat this step to verify the rest of the ZooKeeper containers.

Output: You should see one leader and two followers:

```
Mode: follower

Mode: leader

Mode: follower
```

Step 5.

Check the logs to see if the Kafka brokers have booted up successfully.

```
docker logs <kafka-1_containerId>

docker logs <kafka-2_containerId>

docker logs <kafka-3_containerId>
```

Step 6.

Test that the broker is working as expected. Now that the brokers are up, we will test that they are working as expected by creating a topic. And make sure that the minimum partitions is one and the replication factor not more than the number of ZooKeeper services.

```
docker run --net=host --rm confluentinc/cp-kafka:latest kafka-topics
--create --topic <topic_name> --partitions <Number_of_partitions>
--replication-factor <number_of_replication_factor> --if-not-exists
--zookeeper localhost:32181
```

You should see the following output:

```
Created topic “testTopic”
```

Step 7.

Now verify that the topic is created successfully by describing the topic.

```
docker run --net=host --rm confluentinc/cp-kafka:latest kafka-topics
--describe --topic testTopic --zookeeper localhost:32181
```

Step 8.

Next, we'll try generating some data to the `testTopic` we just created.

```
docker run --net=host --rm confluentinc/cp-kafka:latest bash -c "seq
42 | kafka-console-producer --broker-list localhost:29092 --topic
testTopic && echo 'Producer 42 message.'"
```

The command above will pass 42 integers using the console producer that is shipped with Kafka.

As a result, you should see something like this in your terminal:

```
Producer 42 message.
```

Step 9.

It looked like things were successfully written, but let's try reading messages back using the console consumer and make sure they are all accounted for.

```
docker run --net=host --rm confluentinc/cp-kafka:latest kafka-
console-consumer --bootstrap-server localhost:29092 --topic testTopic
--new-consumer --from-beginning --max-message 42
```

It might take some time for this command to return data. Kafka has to create the consumer offset topic behind the scenes when you consume data for the first time.

Now that the Kafka cluster setup is ready to use, we can access all Kafka brokers with the below host:

```
Kafka_host = “0.0.0.0:19092,0.0.0.0:29092,0.0.0.0:39092”
```

For access to the Kafka cluster with Node.js, you can learn about enabling the npm package at Kafka-node npm https://www.npmjs.com/package/kafka-node