

# One to run them all



Marcos Maia



Jun 16 '19 Updated on Aug 31, 2019 8 min read

[#docker](#) [#kafka](#) [#zookeeper](#) [#kafkamanager](#)

In this Post we're going to cover how to run Kafka for local development of our applications, this will be useful to run future examples and can be used as a basic reference, I will try to keep consistency with this approach.

There are many possible ways to run Kafka locally for development, among those the most common are probably:

- downloading Kafka and Zookeeper manually and running them locally,
- directly from the command line using Docker,
- using a docker-compose file,
- using the Confluent CLI commands,
- running them on the cloud.

From all the possibilities the one I find more intuitive and easy to maintain in development during projects is using docker-compose. I have used different approaches, including VM based solutions in the past but using docker-compose brings a better developer experience. So I will stick to that in my posts here.

If you don't have docker and docker-compose installed please check out [my previous post](#) where I point out directions to where

to find the proper documentation to install it in your own environment.

Docker compose uses a simple YAML file and can also build from Dockerfiles for some more advanced setups if you need more control of the images you're building in your projects. Compose also enables you to set configuration parameters and share those changes within a repository with other team members and reuse it to run as a [Swarm Cluster](#) if you so desire, so we're going to cover mostly docker-compose here as it's my preference and I will provide some hints and links to documentation on how to do it using other approaches.

I think important to clarify that I am not advocating to run all your application in docker during development, it should be dependencies and mocks that you might need. During development, it's very important that you can run your unit tests and application directly from the IDE of your choice to have a better developer experience with faster feedback cycles.

## Running Kafka from a docker-compose file

There are many available docker images of Kafka and Zookeeper, you can also use their binaries to pack your own docker images. Here I will share configurations for a couple of existent options from [docker-hub](#).

- [wurstmeister Kafka and Zookeeper docker images](#).
- [Confluent Kafka and Zookeeper images](#).

The source code with the compose files used in this post is available, you can clone the repository using:

```
git clone git@github.com:stockgeeks/docker-compose.git
```

Open the cloned project in your favorite IDE. The source code for this post is under the folder `one-to-run-them-all`. Navigate to this folder in a command prompt to run the docker-compose commands presented next. If you have problems running the commands make sure to have docker and docker-compose installed as explained in the link shared above and check [this compatibility matrix](#).

When running kafka in this examples notice the configurations on the project specifically the `listener` and `advertised.listener` and `advertised.hostname` as they're key to understand later how we can connect our application running from the IDE to the local kafka development broker we are running here in this post.

## Kakfa wurstmeister image

For this initial example, we're going to use the latest Kafka docker image from [wurstmeister](#) which it's available in [docker hub here](#).

We will not cover components like the schema registry in this post, the idea is to keep it simple and focus on running Kafka while learning some ways to interact with it using docker and docker-compose.

Let's first make it run it in the background, from a shell inside the `one-to-run-them-all` folder in the project:

```
docker-compose up -d
```

you can then use `docker ps` to check the running containers. If you're comfortable with multiple command line shells you can [watch](#) the docker containers continuously in one of them while developing, I usually do that: `watch docker ps` and hit the keyboard with `CTRL + C` when you want to stop watching.

If you're using Mac or Windows it's also possible to install `watch` command and use it. You can alternatively if you have other docker containers running also watch only those in the compose file, from the same directory where the compose file is: `watch docker-compose ps`

To check and follow the container logs in another terminal window, always in the same folder level as the docker-compose file:

```
docker-compose logs -f
```

Now that we have Kafka running locally let's issue some commands to test our setup, first enter the running Kafka docker container:

```
docker exec -it kafka /bin/bash
```

Now you're inside the running Kafka container and can access the Kafka support scripts available in the command line under

/opt/kafka/bin , let's start creating a topic called client with 1 partition and replication factor 1.

```
./kafka-topics.sh --bootstrap-server kafka:9092 --create --topic client
```

let's then list the topics:

```
./kafka-topics.sh --bootstrap-server kafka:9092 --list
```

You should see `client` as this is the topic we just created, but let's get some more details with `describe` :

```
./kafka-topics.sh --bootstrap-server kafka:9092 --topic client --describe
```

Now you should get some more details like Partition, leader, replicas and in sync replicas which in this case are all the same as we've set all to 1 when creating the topic.

With a small variation of docker command you can execute the same kafka commands from the host directly, all you need to do is prefix the commands with `docker exec -it kafka` so from the host machine and NOT inside the kafka running container, you can run: `docker exec -it kafka`  
`/opt/kafka/bin/kafka-topics.sh --bootstrap-server kafka:9092 --topic client --describe` and should have exactly the same output as before.

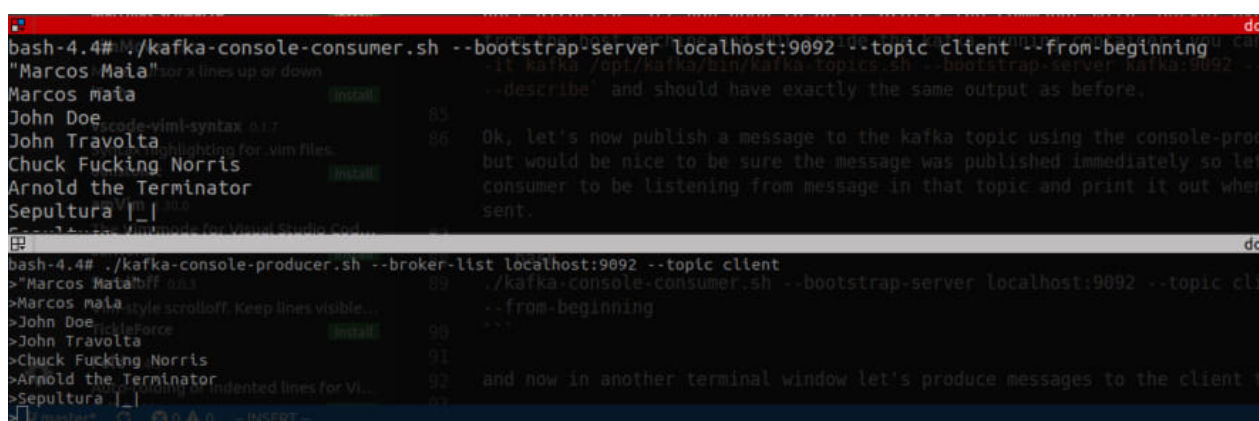
Ok, let's now publish a message to the kafka topic using the console-producer from kafka, but would be nice to be sure the message was published immediately so let's start a consumer to be listening from message in that topic and print it out when a new message is sent.

```
./kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic
```

and now in another terminal window let's produce messages to the client topic:

```
./kafka-console-producer.sh --broker-list localhost:9092 --topic clier
```

Your terminal will be in waiting status, see screenshot below, with a `>`, type in the line and press enter, the message will be produced to Kafka and received by the client in the consumer terminal.

The image shows two terminal windows. The top window is a Kafka console consumer, running the command `./kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic client --from-beginning`. It displays a list of messages: "Marcos Maia", John Doe, John Travolta, Chuck Fucking Norris, Arnold the Terminator, and Sepultura. The bottom window is a Kafka console producer, running the command `./kafka-console-producer.sh --broker-list localhost:9092 --topic client`. It shows the same list of messages being typed into the prompt, with a cursor at the end of the last line.

Now that we have kafka running and have produced and consumed a few messages, let's check the docker-compose file:

```
version: '3.2'
```

```
services:
```

```
# https://github.com/wurstmeister/zookeeper-docker
```

```
zookeeper:
```

```
  container_name: zookeeper
```

```
  image: wurstmeister/zookeeper:latest
```

```
  environment:
```

```
    ZOOKEEPER_CLIENT_PORT: 2181
```

```
  ports:
```

```
    - "2181:2181"
```

```
# https://hub.docker.com/r/confluentinc/cp-kafka/
```

```
kafka:
```

```
  container_name: kafka
```

```
  image: wurstmeister/kafka:2.12-2.2.1
```

```
  environment:
```

```
    ## the >- used below infers a value which is a string and proper
```

```
    ## ignore the multiple lines resulting in one long string:
```

```
    ## https://yaml.org/spec/1.2/spec.html
```

```
    KAFKA_ADVERTISED_LISTENERS: >-
```

```
      LISTENER_DOCKER_INTERNAL://kafka:19092,
```

```
      LISTENER_DOCKER_EXTERNAL://${DOCKER_HOST_IP:-kafka}:9092
```

```
    KAFKA_LISTENERS: >-
```

```
      LISTENER_DOCKER_INTERNAL://:19092,
```

```
      LISTENER_DOCKER_EXTERNAL://:9092
```

```
    KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: >-
```

```
      LISTENER_DOCKER_INTERNAL:PLAINTEXT,
```

```
      LISTENER_DOCKER_EXTERNAL:PLAINTEXT
```

```
    KAFKA_INTER_BROKER_LISTENER_NAME: LISTENER_DOCKER_INTERNAL
```

```
    KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
```

```
    KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
```

```
    KAFKA_LOG4J_LOGGERS: >-
```

```
      kafka.controller=INFO,
```

```
      kafka.producer.async.DefaultEventHandler=INFO,
```

```
      state.change.logger=INFO
```

```
  ports:
```

```
    - 9092:9092
```

```
  depends_on:
```

```
- zookeeper
volumes:
- /var/run/docker.sock:/var/run/docker.sock
```

To stop your Kafka and Zookeeper, from the same folder where the compose file is, using the command line: `docker-compose down -v` which will also clean the mounted volume resetting the kafka topics, if you want to keep existing messages you can remove the `-v` from the command.

## Confluent Kafka and Zookeeper images

Let's now run kafka [confluent docker image](#).

In the project under the folder `one-to-run-them-all` there's a [second docker-compose](#) file called `docker-compose-confluent.yml` to run it we can issue the same command as before specifying the file name when want to run using the option `-f`, but before we do that please make sure the previous containers are stopped, if not, run from the command line: `docker-compose down -v` which will stop the containers we started previously.

```
docker-compose -f docker-compose-confluent.yml up -d
```

Check if the containers are running and put a `watch docker ps` as before to be sure the confluent containers with zookeeper and kafka are running. If yes, let's execute the same commands as before, the main difference is that Confluent Kafka has it's own installation standards so the kafka scripts are under `/usr/bin` which basically gives access to them from everywhere in the shell inside the container, so go ahead and repeat the same commands issued for the



previous example to enter the container shell `docker exec -it`  
`docker /bin/bash` and the commands to list, publish, consume are almost the same:

Enter the kafka container:

```
docker exec -it kafka /bin/bash
```

Start the consumer

```
kafka-console-consumer --bootstrap-server localhost:9092 --topic client
```

Start the producer

```
kafka-console-producer --broker-list kafka:9092 --topic client
```

By default for development which this images are meant for, the topic creation happens automatically when a client tries to consume or produce to it, this should be disabled in a production environment.

## Command line using docker images

This option only requires that you have Docker installed, not docker-compose, you'll run the docker images for Kafka and zookeeper from the command line and that's it. Make sure you have

Docker installed. The tricky part is to pick the docker image, there are many available. The most populars in my perception(would need further research to confirm) are:

- wurstmeister
- Confluent Kafka Images

Please leave a comment if you prefer any other kafka image for development.

## Command line using binaries

This approach is well documented with all required links to download the binaries in the official Apache Kafka Documentation [Quick Start](#) section.

## Confluent cli tools

Confluent provides a very user friendly way of running local kafka for development together if all it's platform which includes many extra features including management and monitoring tools and much more.

If you're using Confluent stack this can be a great option. They have very detailed documentation on their pages about it:

- [Confluent CLI](#)
- [Docker Developer Guide](#)

## Managed Kafka Offers In the Cloud

If you have plans to run your kafka Cluster in the Cloud fully automated, there are a few options, the most natural being [Confluent Cloud](#) as it's managed by a team with many Kafka core contributors and it's creators, currently you can deploy Confluent managed Kafka clusters to Google Cloud or AWS.

Other possible options include:

[IBM Event Streams for IBM Cloud](#) is a full, automated Kafka managed service provided by IBM.

For Azure check their [Quickstart article](#) and more [details in this post](#) with extra links and detailed information.

For AWS there's the AWS official offer [MSK](#) but you will also find different options and companies that offer Kafka managed cluster services over AWS.

For Google Cloud it's recommended to use the [Confluent offer](#) as mentioned above.

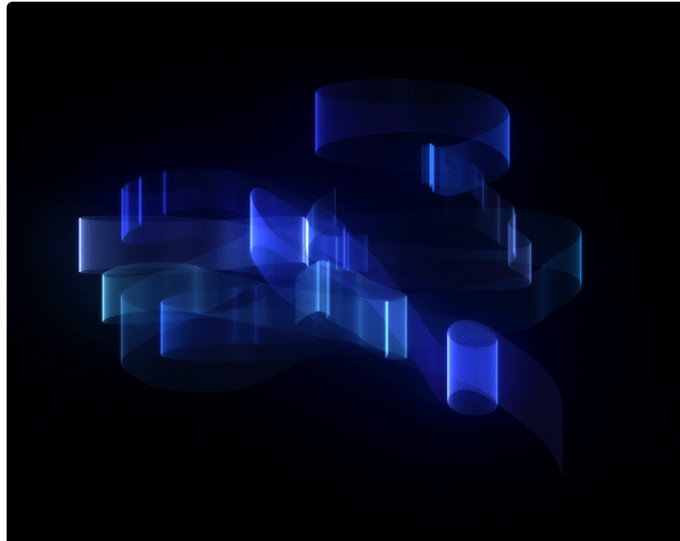
## Kubernetes or Swarm versions

I don't have any experience here, but there are some talks available specially for running Kafka in Kubernetes. Some links:

- [Running Kafka in Kubernetes practical guide](#)
- [Running production Kafka clusters in Kubernetes](#)

## What's next

So in this post I've tried to give some introduction on how to spin up a local kafka infrastructure locally for development. This is the basis to the upcoming posts where we will see a bit more of developing Kafka Applications with Springboot.



[dev.to](#) now has dark mode.  
Select **night theme** in the "misc"  
section of [your settings](#) ❤



**Marcos Maia** + FOLLOW

I am an experienced Developer, Trainer and Speaker. Eager to learn and share knowledge. A bit introspective. Keep coding. Be humble. Help others.

@thegroo  thegroo  mmaia  [gitlab.com/marcosmaia](https://gitlab.com/marcosmaia)

## Discussion

Add to the discussion



PREVIEW


SUBMIT

[code of conduct](#) - [report abuse](#)

---

---

[Home](#) [About](#) [Privacy Policy](#) [Terms of Use](#) [Contact](#) [Code of Conduct](#)

DEV Community copyright 2016 - 2020 

---