Docker Compose

In this lesson, we'll discuss Docker Compose.

WE'LL COVER THE FOLLOWING

Overview

A typical microservice system contains more than a single Docker container. As explained in chapter 2, microservices are modules of a system.

It would be good to have a way to start and run several containers together for starting all the modules that the system consists of in one go. This can be done with Docker Compose.

Service discovery with Docker Compose links

Coordinating a system of multiple Docker containers requires more than just starting multiple Docker containers. It also requires **configurations for the virtual network** with which the Docker containers communicate with each other. In particular, **containers must be able to find each other in order to communicate**.

In a Docker Compose environment, **a service can simply contact another**

service via a Docker Compose link and then use the service name as the

hostname. So it could use a URL like `http://order/` to contact the order microservice.

Docker Compose links offer some kind of service discovery, that is, a way for microservices to find other microservices. Synchronous microservices require a form of service discovery.

Docker Compose links extend Docker links. Docker links only allow communication. Docker Compose links also implement **load balancing** and set the start order so that the dependent Docker containers start first.

Ports

In addition, Docker Compose can bind ports from the containers to the ports of the Docker host where the Docker containers run.

Volumes

Docker Compose can also provide volumes. These are file systems that can be shared by multiple containers. This allows containers to communicate by exchanging files.

YAML configuration

Docker Compose configures the interaction of the Docker containers with a YAML configuration file `docker-compose.yml`.

The following file comes from a project which implements Edge Side Includes as a way to compose websites from different sources. For this purpose, three containers must be coordinated.

- `common` is a web application that is supposed to deliver common artifacts.
- `order` is a web application for processing orders.
- `varnish` is a web cache to coordinate the two web applications.

```
version: '3'
services:
  common:
    build: ../scs-demo-esi-common/
  order:
    build: ../scs-demo-esi-order
  varnish:

    build: varnish
    links:
     - common
     - order
    ports:
     - "8080:8080"
```

- The first line defines the used version of Docker Compose – in this case three.
- The second line starts the definition of the services.
- Line three defines the service `common`. The directory specified in line four contains a `Dockerfile` with which the service can be built. An alternative to `build` would be `image` to use a Docker image from a Docker registry.
- The definition of the service `order` also specifies a directory with a `Dockerfile`. No other settings are required for this service (lines 5/6).
- The service `varnish` is also defined by a directory with a `Dockerfile` (lines 7/8).
- The service `varnish` must have Docker Compose links to the services `common` and `order`. Therefore, it has entries under `links`. The `varnish` service can therefore reach the other services using the host names `common` and `order` (lines 9-11).
- Finally, port 8080 of the service `varnish` is bound to port 8080 of the Docker host, on which Docker containers run (lines 12-13).

Additional options

Further elements of the YAML configuration are described in the reference documentation. For example, Docker Compose supports volumes shared by multiple Docker containers. Docker Compose can also configure the Docker containers using environment variables.

Docker Compose live environment!

You can try out Docker compose commands in the following environment. It consists of 3 Docker images running together. The final app should be available at the generated link such as https://x6jr4kg.educative.run

Docker Compose commands

Docker Compose is controlled by the command line tool `docker-compose`. It must be started in the directory where the file `docker-compose.yml` is stored. The reference documentation lists all command line options for this tool. This lesson in the Appendix shows an overview of the Docker Compose commands. The most important ones are:

- `docker-compose build` builds the images for the services. **Try this command in the Docker compose coding environment above**!

- With `docker-compose up`, all services are started. The command returns the combined standard output of all services. This is rarely helpful, so `docker-compose up -d` is often the better choice. In this case, the standard output is not returned. **Try this command next**! Notice all the services will be up and accessible at the given link. With `docker log` the output of individual containers can be viewed.

- With `docker-compose up --scale <service>=<number>`, a larger number of containers for a service can be started. In the example from the listing, `docker-compose up --scale common=2` could ensure that two containers for the service `common` are started. **Try this next**.

- `docker-compose down` shuts down all services and deletes the containers. **Try this to shut down all services**.

Since the examples often require the interaction of several Docker containers, most examples have a `docker-compose.yml` file to run the containers together.

QUIZ

1

What is the purpose of Docker Compose?

COMPLETED 0%

1 of 3

In the next lesson, we’ll discuss some variations to what we have already learned.