Docker-Compose Volumes, Environments

Getting started with docker-compose

We'll cover the following

With an automated multi-container workflow using docker-compose, you must have started feeling a little enthusiastic about it.

So, without further ado, we will look at the meaning of the docker-compose file line by line.

```
version: '3'
services:
  web:
    # Path to dockerfile.
    # '.' represents the current directory in which
    # docker-compose.yml is present.
    build: .

    # Mapping of container port to host

    ports:
      - "5000:5000"
    # Mount volume
    volumes:
      - "/usercode/:/code"

    # Link database container to app container
    # for reachability.
    links:
      - "database:backenddb"

  database:

    # image to fetch from docker hub
    image: mysql/mysql-server:5.7

    # Environment variables for startup script
    # container will use these variables
    # to start the container with these define variables.
    environment:
      - "MYSQL_ROOT_PASSWORD=root"
      - "MYSQL_USER=testuser"
      - "MYSQL_PASSWORD=admin123"
      - "MYSQL_DATABASE=backend"
    # Mount init.sql file to automatically run
    # mount init.sql file to automatically run
    # and create tables for us.
    # everything in docker-entrypoint-initdb.d folder
    # is executed as soon as container is up nd running.
    volumes:
      - "/usercode/db/init.sql:/docker-entrypoint-initdb.d/init.sql"
```

Docker Compose file

Docker file explanation

The Compose file is in YAML format. It's a very simple language. You can validate your YAML file using any of the online validator tools. Here is one for the reference.

- **version '3'**: Like other software, docker-compose also started with a version 1.0. At the time of writing this course, the current latest version of Compose file is 3.7.

We have specified what version of Compose file we will be using and Docker will provide the features accordingly.

Compose versions are backwards compatible, hence I suggest you use the latest version.

- **services**: The services section defines all the different containers to be created. We have two services namely, web and database. In Compose version 3, we can have multiple containers of the same service as well.

We will see that in the next section, but if you are curious, you can check here under the deploy section in the compose file.

- **web**: This is the name of our Flask app service. It can be anything. Docker Compose will create containers with this name.

- **build**: This clause specifies the Dockerfile location. '.' represents the current directory where the docker-compose.yml file is located and Dockerfile is used to build an image and run the container from it. We can also enter a path to Dockerfile there.

- **ports**: The ports clause is used to map or expose the container ports to the host machine. If you remember, we used `-p 5000:5000` while running the container using Docker. This will do the same work for us.

- **volumes**: This is the same as the `-v` option used to mount disks in Docker. Here, we are attaching our code files directory to the containers, ./code

  directory so that we don't have to rebuild the images for every change in the files.

This will also help in auto-reloading the server when running in debug mode.

- **links**: Links literally links one service to another. In the bridge network, we have to specify which container should be accessible to which container using a link to respective containers.

Here, we are linking the database container to the web container, so that our web container can reach the database in the bridge network.

- **image**: If we don't have a Dockerfile and want to run a service directly using an already built image, we can specify the image location using the 'image' clause. Compose will pull the image and fork a container from it.

- **environment**: Any environment variable that needs to be set up in the container can be created using the 'environment' clause. This does the same work as the `-e` argument in Docker while running a container.

All the clauses or keywords above are commonly used keywords, which are enough to start a development workflow. However, there are some advanced-level keywords that are used in production.

As we move forward in the course, we will make this docker-compose file production-ready.

In the next lesson, we will play around with docker-compose using commands so that you can get used to it.