Solution

The solution to exercise 3

We'll cover the following

- Web
- Database

Congratulations!! You've made it this far. You can pat your back, because as you have done a wonderful job. Solving this exercise might have squeezed all your developer plus Docker knowledge, but you did not give up and that's a good thing.

Even if you were not able to solve this exercise, but you've tried your best, kudos to you because this exercise was a little tricky one. So, let's go through the solution and see what you might have missed.

```
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/:/exercise_3"

    # Link database container to app container
    # for rechability.
    links:
      - "database:exercisedb"

    command: ["flask", "run"]

    environment:
      - "FLASK_RUN_HOST=0.0.0.0"

    depends_on:
      - database
```

```
database:

  # image to fetch from docker hub

  build:
    context: ./db
    dockerfile: Dockerfile-db
  #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
  # 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"
```

Here, we will only see the new changes in docker-compose.yml file. Since we have covered the rest of the commands in previous lessons, we don't need to repeat them but you can go back and revise them in case you don't understand all of them.

Web

- The very first thing is to set up the basic skeleton using `build` and `ports` keyword for our web service.
- Next, we check what the working directory is in Dockerfile and mount the host volume accordingly.
- Then we cross-check the DB credentials used in the app.py file. If it is changed, match those in the links section, because `links` will decide what should be the reachable hostname of the database service.
- `command`: Here comes the tricky one. This will pass the command to the container to execute. If you cross-check the Dockerfile, you will notice there is no `CMD` or `ENTRYPOINT` at the end to make a container executable. So, we are passing the command to run the Flask app from the docker-compose.yml file itself.
- `environment`: This is not a very tricky one, but if you do not set the

`FLASK_RUN_HOST` variable to 0.0.0.0, you won't be able to access it outside of the container.

Database

There is nothing new in this section. But you have to focus on the environment variables passed to it because this will be used by the application and any spelling mistake in these will result in a connection failure between the app and the database.

Writing down password and DB credentials in code files is not good practice. So at the end of this course, you will learn how to pass those to the application as well as to the container.

In the next section, you will learn some advanced topics of the Docker system which will strengthen your Docker skills. After solving this tricky exercise, you must be eager to learn some advanced topics.

So, jump to the section whenever you are ready.