Environment Variables with Docker-compose

Using environment variables in Docker

We'll cover the following

Currently, we have included our database credentials in the code itself. But in production systems and an ideal development workflow, it is not advised to add such critical information in the code.

> Clone the project using `git clone https://github.com/venky8283/Docker.git`

> Type `git checkout 6e24a37faa2e1bd3401c77b2951f995f410b3d35` to get to the code used in the lesson.

Environment variables are one of the options that are used in production systems to manage credentials. So, let's see how to work with the environment variables using Docker.

Accessing environment variables in Docker

Using the .env file

A file with only .env extension is used in the production systems to store all the variables. Since it is a hidden file, it is not normally pushed to the code.

We will create a .env file and move the variables to the .env file. To access the variables, we will use `${}` syntax in docker-compose.

The .env file should be in the same folder where the docker-compsoe.yml file is located.

You can try running `docker-compose up` once these changes are done.

Python

docker-compose.yml .env	All code files are copied to end of the page...

Using the env_file

Another method is to use the env_file keyword instead of the environment keyword in docker-compose.yml. In this method, it is not necessary that the .env file should be located in the same directory as the docker-compose file.

You will provide the location of the .env file like so:

```
env_file:
      - ./.env
```

This will fetch all the mentioned environment variables from the .env file.

```
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 defined variables.
    env_file:
      - ./.env
    # 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"
```

Priority of .env variables

There are multiple ways we can use the environment variables in Docker, which includes,

- ENV in Dockerfile
- `environment` keyword in docker-compsoe.yml file
- `-e` option from the command line

So, docker-compose has precedence levels to overcome the clash of variables. Let's see those.

When you set the same environment variable in multiple files, here's the priority used by Compose to choose which value to use:

- Compose file
- Shell environment variables
- Environment file
- Dockerfile
- Variable is not defined

So, if we define a variable in the Compose file in the `- environment` section and also in a `.env` file, Compose will consider the variable declared in the Compose file from `-environment` section.

Sometimes, this might be the solution to any unexpected behavior of the application when multiple environment variables are used.

With this, we have completed the docker-compose section. Try out the exercise and quiz as well.

From the next lesson, we will move towards advanced usage of Docker and look at an introduction to orchestration tools in Docker.

Code Files Content !!!

⤋ ⤋

```
-------------------------------------------------------------------------
|  docker-compose.yml [1]
-------------------------------------------------------------------------



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 rechability.
    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=${MYSQL_ROOT_PASSWORD}"
      - "MYSQL_USER=${MYSQL_USER}"
      - "MYSQL_PASSWORD=${MYSQL_PASSWORD}"
      - "MYSQL_DATABASE=${MYSQL_DATABASE}"
    # 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"
```

```
-------------------------------------------------------------------------
|  .env [1]
-------------------------------------------------------------------------


MYSQL_ROOT_PASSWORD=root
MYSQL_USER=testuser
MYSQL_PASSWORD=admin123
MYSQL_DATABASE=backend





**********************************************************************************
```