# Assignment: Creating a Full-Stack App with Docker

## Introduction

This project is the second part of a series of Docker challenges and aims to stimulate students to develop their technical skills using containerization tools like Docker, relating to some content covered during the Operating Systems and Cloud Computing course.

## Dependencies

To perform these challenges, each student must

1. Have **git** installed in their machine.
2. Have a **GitHub account**.
3. Have **Docker installed** on their computer (or in their Linux VM if you consider it more convenient).

### Getting started

The first step is to clone the repository and **keep it private**:

<https://github.com/eduluz1976/docker-challenge-template>

The repository folders should follow this structure:

- README.md

- student.cfg

+ challenge3

+ challenge4

The file **student.cfg** will contain the student information, and the README.md some information about your work.

## Resources

* <https://www.youtube.com/watch?v=1je3VxDF67o>
* <https://docs.docker.com/compose/environment-variables/>
* <https://docs.docker.com/reference/cli/docker/compose/up/>
* <https://www.appsdeveloperblog.com/scaling-with-docker-compose-a-beginners-guide/>

### Goals

Each challenge has a set of goals. Aside from each challenge goal, this laboratory aims to stimulate all students to learn more about Docker, and how it relates to all Operating Systems and Cloud Computing content covered in the course. Moreover, these challenges will bring some real case scenarios from the industry, which will be useful for job interviews, software development activities, and open space for new learnings in related areas, such as DevOps, Architecture, and Cybersecurity.

Remember: much more than grades, these challenges will bring you knowledge and experience valid for your professional life. This is a great opportunity to boost your knowledge in a very required subject nowadays.

### Guidelines

Here are some basic rules for these challenges:

1. This is an **individual assignment**.
2. At the end of each challenge, we expect that each student will deliver:
   1. Their personal, public repository on GitHub, with all the necessary files to make the produced applications work.
   2. A report with all the steps that they went through, to achieve the results. See more details about this journal below.
   3. Answer to eventual questions.
3. **Copies (or shared work) will not be tolerated**.

#### Report guidelines

Think of your report as a tutorial for someone who has never used Docker before and needs some guidance.

Submitted on D2L:

* Your report as a single file in PDF format:
  + You can add links to web pages used as references.
  + Add screenshots of your terminal, and your browser, showing the commands you used, and the results.

# Challenges

## Challenge 3 – Full-stack application

This challenge aims to use the same idea as the previous one, where a Docker Compose was used to serve an application in 2 layers (web server and Node app). Now, the goal is to add a database to it and make them work together.

### Goals

* Understand how it works the connection among different components (services).
* Present the environment variables in the application’s configuration.
* Produce a docker-compose.yml with a web server, application and database working together.

### Steps

* Use the folder challenge3.
* Extract the files present on challenge3.zip to the challenge’s root folder.
* Create a .env file with the appropriate values (refer to the configuration variables described below).
* Create the docker-compose.yml with all 3 services:
  + nginx
  + node-service
  + db
* Check if all services are running properly.
  + Using your browser, access the address <http://localhost:8080/api/books>
    - If the result is not expected, then return and fix it.
  + Now open the URL: <http://localhost:8080/api/books/1>
    - If the result is not expected, then return and fix it.
* Commit all files and push them to the remote repository.
* Submit the repository's URL in D2L and answer the questions.

### Expected outcomes

* When you access the following URLs:
  + “http://localhost:8080/api/books” you will get a JSON message with all books.
  + “http://localhost:8080/api/books/1” you will get a JSON message with just one book.
* When executing the command “docker-compose ps”, you should see something similar to:

|  |
| --- |
| NAME IMAGE COMMAND SERVICE CREATED STATUS PORTS  challenge3-db-1 challenge3-db "docker-entrypoint.s…" db 2 hours ago Up 2 hours 3306/tcp  challenge3-nginx-1 challenge3-nginx "/docker-entrypoint.…" nginx 2 hours ago Up 2 hours 0.0.0.0:8888->80/tcp  challenge3-node-service-1 challenge3-node-service "docker-entrypoint.s…" node-service About an hour ago Up About an hour 3000/tcp |

* On D2L you will submit the document with all steps necessary to achieve the results.

### Configuration Variales

For this challenge, you will need to provide some environmental variables to configure your services. Refer to this page for more details: <https://hub.docker.com/_/mariadb>

Application config:

* DB\_ROOT\_PASSWORD
* DB\_DATABASE
* DB\_USERNAME
* DB\_PASSWORD
* DB\_HOST

Database config:

* MYSQL\_ROOT\_PASSWORD
* MYSQL\_DATABASE
* MYSQL\_USER
* MYSQL\_PASSWORD
* MYSQL\_HOST

## Challenge 4 – Scaling up an application

This last project aims to allow the students to learn how to scale up a service. Using the very same code from the previous challenge, the goal is to scale the node service in such a way that instead of having only 1 instance, we have 3 running.

**This challenge does not involve any extra implementation**. Only research the commands necessary to scale service and document the steps and the evidence that you achieve the proposed goals.

### Goals

* Scale up the node service from 1 to 3 instances.
* Understand the benefits of having more instances running instead of only one.

### Steps

* Use the challenge3 application.
* In your browser or REST client, make a GET request to your application:
  + <http://localhost:8080/api/stats>
* Record the field “hostname”.
* Repeat the same operation a couple of times.
* Now, research how to scale up a particular service.
  + Perform it on “node-service”, scaling up from 1 to 3 instances.
* Now repeat the operation, performing requests on:
  + <http://localhost:8080/api/stats>, multiple times,
  + Record the hostnames.
* Did you see any difference?
* Execute the command “docker-compose ps” and record the output.

### Expected outcomes

* When you make requests to the URL “http://localhost:8080/api/stats” before scaling up your node-service, you should always see the same hostname after repeated requests.
* After scaling up your node-service, you should note that now each request will bring a different hostname (up to 3). Remember the Round-Robin?
* On D2L you will submit the document with all steps necessary to achieve the results.