# Challenge 3

## Getting Started

This tutorial assumes you have installed docker from <https://docs.docker.com/engine/install/> and used the default parameters.

To get started unzip the challenge3 file a location for use. It contains a folder named docker which contains the files you will need. This docker folder will be the root for this application.

## Getting the Full Stack Application running

1. Your docker folder should resemble something like the below. A screenshot of a computer

   Description automatically generated
2. To start create a .env file in the docker folder.
3. You’ll need to put the following enviroment variables into it.
   1. PORT
   2. DB\_HOST
   3. DB\_USERNAME
   4. A screenshot of a computer program

      Description automatically generatedDB\_PASSWORD
   5. DB\_DATABASE
   6. MYSQL\_ROOT\_PASSWORD
   7. MYSQL\_DATABASE
4. They will be assigned their respective values from the list below:
   1. 3000
   2. Db
   3. Root
   4. Password
   5. Books
   6. Password
   7. A screenshot of a computer

      Description automatically generatedBooks

The file should resemble this after.

1. Next create a file named docker-compose.yaml in the root folder.
2. We’ll be using version 3 so put in “version: '3'” at the top of the file.
3. Next create services for the three parts of this application:
   1. api
   2. nginx
   3. database(db)
4. Additionally a volume will be need at the bottom of the file.

A screenshot of a computer

Description automatically generated

1. We will build all three services with ./ in front of their respective names eg. for api build:./api
2. The api will depend on the db add a

depends\_on:

- db

and will use enviomental variables so add a

env\_file:

- .env

A screen shot of a computer

Description automatically generated

1. The database will also use environment variables but will use a volume

db\_data:/var/lib/mysql

A screen shot of a computer

Description automatically generated

1. The last service is nginx which depends on the api and uses ports 8080 for the nginx and 80 for the container. A screen shot of a computer program

   Description automatically generated
2. The last thing in this file is to create a named volume to persist data across container restarts. It will be db\_data A black background with blue text

   Description automatically generated.
3. At this point the docker-componse.yaml should resemble.A screen shot of a computer program

   Description automatically generated
4. First the docker container needs to be built, use the following command in a terminal in the docker folder: docker-compose build if making a fresh instance or docker-compose build –no-cache if needing to rebuild all parts of the container. If changing things in modules or other parts of the container use the latter. To open the terminal in the folder use either VScode and right click on the docker file to open a menu, select the option to “Open in the Integerated Terminal”. Alternitvely use cd “file path to docker folder here” to change directory to docker folder on your computer.A screen shot of a computer

   Description automatically generated
5. To run the container run “docker-compose up” in the same terminal as the one above. Adding a -d to run in detached mode will allow the container to run in the background. A black screen with white text

   Description automatically generated
6. Open the docker desktop app to confirm the container is running like so. A screenshot of a computer

   Description automatically generated
7. If the container is running open a browser and input “localhost:8080/api/books” to confirm the application is working. It should look like so.A screenshot of a computer

   Description automatically generated

# Challenge 4

https://www.appsdeveloperblog.com/scaling-with-docker-compose-a-beginners-guide/  
was used for this section.

## Benefits of scaling up an application with docker

Scaling up an application provides 6 benefits.

1. Ease of use: Docker provides an easy means to scale up.
2. Environment consistency: Each instance is a exact replica enabling consistent behavior.
3. Development and testing: Easy to see how apps react to high loads.
4. Load distribution: Distribution of load across multiple instances improves performance and responsiveness.
5. Resilience/redundancy: Multiple instances enables the app to continue to function even if one instance goes down.
6. Cost: Docker enables an easy way to scale based on demand enabling efficient use of resources.

For general use having multiple instances of an application running provides the benefits of load distribution enabling more responsive and better performing apps, resilience and redundancy enabling the app to keep functioning even if some of the instances need to be taken down enabling more uptime and reliability and lastly enabling an easy way to test how an app reacts to variables like high load by running multiple client.

## Hostname

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

Even if refreshing or making the request on multiple tabs they all have the same hostname.

## Scaling up

As we need to scale up node-service we first need to enable it to bind to multiple ports simply not explicitly assigning a port in the docker-compose.yaml is sufficient for this purpose. After this change using the command “docker-compose up –-scale node-services=3” will create 3 instances of the node-services api. A screenshot of a computer

Description automatically generated When using this method making multiple requests will have the webpage access different api instances resulting in different hostnames like so. A screenshot of a computer

Description automatically generated

Using “docker-compose ps” will show the running services, here we show that we have three node-service services running.

A screen shot of a computer code

Description automatically generated