

# Docker Challenge Notes (Tutorial)

## Preface

This article is to introduce and document the Docker Challenge assignment on the Operating Systems course. Although they are the most basic parts, these two challenges cover common scenarios where Docker is used in practice.

## Background

**What is docker and why** do you use it? For the official description, see [here](#). I'll use my understanding of the analogy here:

Imagine you are a chef preparing a big meal for your guests. You have your recipe (application code), and you need a kitchen (runtime environment) to prepare your food. In different places, the kitchen may be configured differently (different operating systems and dependencies), which may affect the way you prepare your food, and even the final taste (consistency issues with the application runtime).

## Problems with the traditional approach

If we don't use Docker, it's like you need to relearn and adapt every time you prepare food based on the kitchen at your location. For example, you can prepare a dish perfectly in your home kitchen (your development environment), but when you try to prepare the same dish in your friend's kitchen (production environment), it may end up tasting very different because of different tools, different heat in the stove, etc.

## Docker's solution

With Docker, it's like you have a mobile kitchen that you can take anywhere. No matter where you are, with this mobile kitchen, you can prepare your food the

same way, making sure it tastes the same every time you make it. This mobile kitchen is the Docker container, which ensures that your application (food) can run the same way anywhere (different computing environments).

## Why Docker?

**Consistency:** Just like our mobile kitchen ensures that food tastes the same no matter where it is made, Docker ensures that your apps run the same way in any environment.

**Portability:** You can take your mobile kitchen (Docker containers) anywhere, whether it's a new server or a cloud environment, without making any changes.

**Isolation:** In your big kitchen, there may be multiple mobile kitchens working at the same time, but they don't interfere with each other. In Docker, each container is independent, ensuring isolation between applications.

**Rapid deployment:** It's much faster to prepare a new mobile kitchen than it is to build a new physical kitchen. Similarly, deploying a new application or service with Docker is much faster than the traditional way.

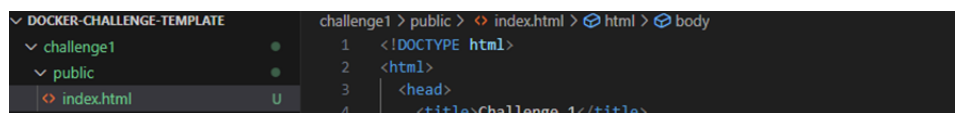
(We'll move on to the challenges below)

## Challenge 1 - Simple web server for static web pages

This is an introduction of how to use Docker to serve some static pages.

### Steps and solutions:

**1. Create static files:** Follow the instructions to create the required web page file `index.html` and put it in the public folder.



The screenshot shows a file explorer on the left with the following structure:

- DOCKER-CHALLENGE-TEMPLATE
  - challenge1
    - public
      - index.html

On the right, the code editor shows the content of `index.html`:

```
challenge1 > public > index.html > html > body
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>Challenge 1</title>
```

**2. Create a Docker file:** Based on the official image, copy the external static file into the image, set the parameters, and define the instructions for running the service.

1. # Use the official nginx image
2. FROM nginx:alpine
- 3.
4. # Copy the contents of the public directory to the nginx root directory

```

5. COPY public /usr/share/nginx/html
6.
7. # expose port 80
8. EXPOSE 80
9.
10. # Run nginx
11. CMD ["nginx", "-g", "daemon off;"]

```

(Tips: The Dockerfile defines the image, and the image can't be changed after it's been created, so in our fully static example we're creating a mirror of our content by copying the external index.html to the docker's internal filesystem. Similarly, we can delete the default Nginx server configuration file from within docker and copy the Nginx server configuration file we defined in the challenge1 folder into the docker image to change the Nginx server configuration).

**3. Create images:** Create images by docker build with parameters, where -t or --tag is the build command to create an image, we use most often.

*docker build -t challenge-1 .*

Usage: docker build -t [Image name]:[Tag/version] [Dockerfile path]

The screenshot shows a VS Code editor with a file explorer on the left displaying a project structure with folders 'challenge1' and 'challenge2', and files like 'index.html', '.gitignore', 'Dockerfile', 'README.md', and 'student.cfg'. The main editor shows the content of the 'Dockerfile' for 'challenge1', which matches the code in the first block. Below the editor, the 'TERMINAL' tab shows the output of the command 'docker build -t challenge-1 .'. The output indicates a successful build of version 2.5s (8/8), showing steps for loading the build definition, transferring the Dockerfile, loading metadata for 'nginx:alpine', authenticating with the Docker registry, and finally transferring the content to the image.

**4. Create the docker container:** Using the image challenge-1 you just created

*docker create --name challenge-1-container -p 8080:80 challenge-1*

A hash will be returned if the creation is successful.

```

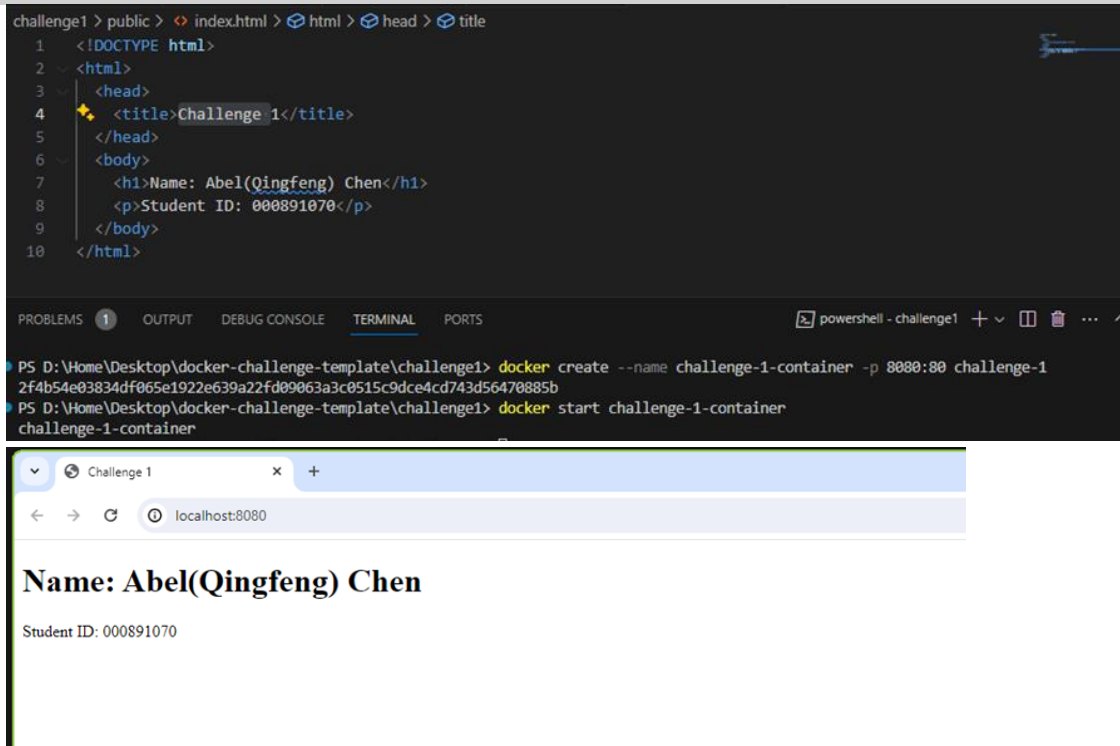
PS D:\Home\Desktop\docker-challenge-template\challenge1> docker create --name challenge-1-container -p 8080:80 challenge-1
2f4b54e03834df065e1922e639a22fd09063a3c0515c9dce4cd743d56470885b

```

Usage: create --name [container name] -p [host port]:[container port] [Image name]

**5. Run the docker container**

docker start challenge-1-container



```
challenge1 > public > index.html > html > head > title
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>Challenge 1</title>
5 </head>
6 <body>
7 <h1>Name: Abel(Qingfeng) Chen</h1>
8 <p>Student ID: 000891070</p>
9 </body>
10 </html>
```

```
PS D:\Home\Desktop\docker-challenge-template\challenge1> docker create --name challenge-1-container -p 8080:80 challenge-1
2f4b54e03834df065e1922e639a22fd09063a3c0515c9dce4cd743d56470885b
PS D:\Home\Desktop\docker-challenge-template\challenge1> docker start challenge-1-container
challenge-1-container
```

Challenge 1

localhost:8080

**Name: Abel(Qingfeng) Chen**

Student ID: 000891070

We see that the corresponding service starts and the page is displayed as expected.

docker stop challenge-1-container

can be used to shut down the container.

## Challenge 2 – Creating a dynamic application

Given a simple NodeJS application, create a Dockerfile that allows to expose the endpoints to external clients, and the “docker-compose.yml”, responsible for the webserver and the application orchestration.

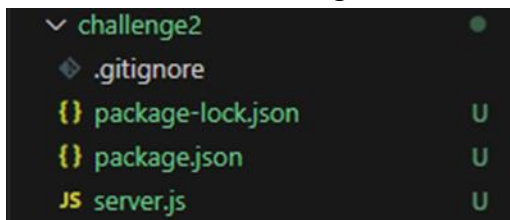
# Background

docker compose, the official description <https://docs.docker.com/compose/>, my own understanding is that this is the combination of some series of docker containers. They work together through the configuration defined in the docker-compose.yml file. Note that the definition of each image is still using the Dockerfile definition, docker-compose.yml is more similar to a coordination script, responsible for the creation of the corresponding docker containers, set the parameters, according to the specified is the order of startup, one step to complete the series of actions, startup and shutdown, very convenient.

## Steps and solutions:

### 1. app definition:

Extract service.js (main program), package.json (dependency package definition) to the challenge2 folder.



Create the app's Dockerfile, there is one more step than challenge1 "RUN npm install", which is to generate the corresponding dependency files in the created image.

```
1. # Use the official node image
2. FROM node:14
3.
4. # Set the working directory
5. WORKDIR /usr/src/app
6.
7. # copy package.json and package-lock.json
8. COPY package*.json ./
9.
10. # Install dependencies
11. RUN npm install
12.
13. # copy app files
14. COPY . .
15.
16. # expose port 3000
17. EXPOSE 3000
```

```

18.
19. # define environment variable
20. ENV PORT=3000
21.
22. # define the command to run
23. CMD ["npm", "start"]

```

(Tips: RUN denotes code that is to be run only when the image is created. CMD defines the commands that are run when the container is created, but not when the image is created.)

## 2. Nginx Definition:

We use Nginx as a proxy server to forward requests from port 8080 to port 3000 of the **app**, configured using the docker-compose feature, using the name defined in docker-compose to represent it.

```

1. events {}
2.
3. http {
4.     server {
5.         listen 80;
6.
7.         location / {
8.             proxy_pass http://app:3000; # app is the name of the service in the
               docker-compose file
9.             proxy_http_version 1.1;
10.             proxy_set_header Upgrade $http_upgrade;
11.             proxy_set_header Connection 'upgrade';
12.             proxy_set_header Host $host;
13.             proxy_cache_bypass $http_upgrade;
14.         }
15.     }
16. }

```

(Tips: Since our project has only one app to define the image (nginx uses the official image), all of our files can be placed in the challenge2 root folder, if there are multiple mirrors that need to be defined separately, you can set up the app subfolder, nginx subfolder to place the corresponding Dockerfile, and then the root file is placed in the (docker-compose.yml)

## 3 create docker-compose.yml file

```

1. version: '3.8'

```

```

2. services:
3.   app: # name of the service
4.     build: . # a container image is built from the Dockerfile in the current directory
5.     volumes:
6.       - ./usr/src/app # bind mount the current directory to /usr/src/app
7.       - /usr/src/app/node_modules # exclude the node_modules directory from the bind
        mount
8.     environment:
9.       - PORT=3000 # set the PORT environment variable to 3000
10.
11.   nginx: # name of the service
12.     image: nginx:latest # use the latest version of the nginx image
13.     ports:
14.       - "8080:80" # bind port 8080 on the host to port 80 on the container
15.     volumes:
16.       - ./nginx.conf:/etc/nginx/nginx.conf # bind mount the nginx.conf file to
        /etc/nginx/nginx.conf
17.     depends_on:
18.       - app # start the app service before the nginx service

```

\* "- /usr/src/app/node\_modules" is an anonymous mapping which means we don't change the contents of the folder in the image, because I don't want to run npm install in my local environment, and if I don't keep the contents of the image, the program won't run because it doesn't have any dependencies.

## 4. Create an image and run the corresponding docker container and network according to the settings

```
docker-compose up --build
```

```

PS D:\Home\Desktop\docker-challenge-template\challenge2> docker-compose up --build
[+] Running 8/8
  ✓ nginx 7 layers [#####] 0B/0B Pulled 4.0s
  ✓ 8a1e25ce7c4f Pull complete 1.3s
  ✓ e78b137be355 Pull complete 2.6s
  ✓ 39fc875bd2b2 Pull complete 0.4s
  ✓ 035788421403 Pull complete 0.7s
  ✓ 87c3fb37cbf2 Pull complete 1.1s
  ✓ c5cdd1ce752d Pull complete 1.4s
  ✓ 33952c599532 Pull complete 1.6s
2024/03/23 18:05:07 http2: server: error reading preface from client //./pipe/docker_engine: file has already been cl
osed
[+] Building 0.0s (0/0) docker:default
2024/03/23 18:05:07 http2: server: error reading preface from client //./pipe/docker_engine: file has already been cl
[+] Building 0.7s (11/11) FINISHED docker:default
=> [app internal] load build definition from Dockerfile 0.0s
=> => transferring dockerfile: 414B 0.0s
=> [app internal] load metadata for docker.io/library/node:14 0.6s
=> [app auth] library/node:pull token for registry-1.docker.io 0.0s
=> [app internal] load .dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> [app 1/5] FROM docker.io/library/node:14@sha256:a158d3b9b4e3fa813fa6c8c59eb8f0a860e015ad4e59bbce5744d2f6fd 0.0s
=> [app internal] load build context 0.0s
=> => transferring context: 258B 0.0s
=> CACHED [app 2/5] WORKDIR /usr/src/app 0.0s
=> CACHED [app 3/5] COPY package*.json ./ 0.0s
=> CACHED [app 4/5] RUN npm install 0.0s
=> CACHED [app 5/5] COPY . . 0.0s
=> [app] exporting to image 0.0s
=> => exporting layers 0.0s
=> => writing image sha256:99e05cd42528525d9d758370a8e05f410a2c296e6731fe67b54d9b08bba1a2dc 0.0s
=> => naming to docker.io/library/challenge2-app 0.0s
[+] Running 3/2
  ✓ Network challenge2_default Created 0.0s
  ✓ Container challenge2-app-1 Created 0.1s
  ✓ Container challenge2-nginx-1 Created 0.0s
Attaching to app-1, nginx-1
app-1 | > srv@1.0.0 start /usr/src/app
app-1 | > node server.js
app-1 |
app-1 | Server running on port 3000
nginx-1 | /docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
nginx-1 | /docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
nginx-1 | /docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
nginx-1 | 10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
nginx-1 | 10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
nginx-1 | /docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
nginx-1 | /docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
nginx-1 | /docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
nginx-1 | /docker-entrypoint.sh: Configuration complete; ready for start up

```

At this point the docker desktop interface can also see the docker-compose running:

Containers [Give feedback](#)

Container CPU usage ⓘ

0.00% / 2800% (28 CPUs available)

Container memory usage ⓘ

29.78MB / 15.14GB

Show charts

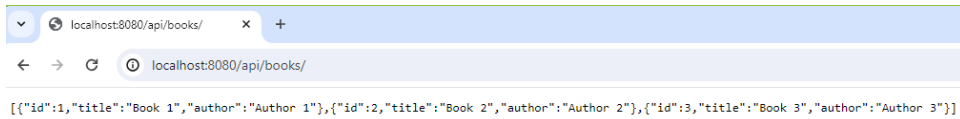
☰

Only show running containers

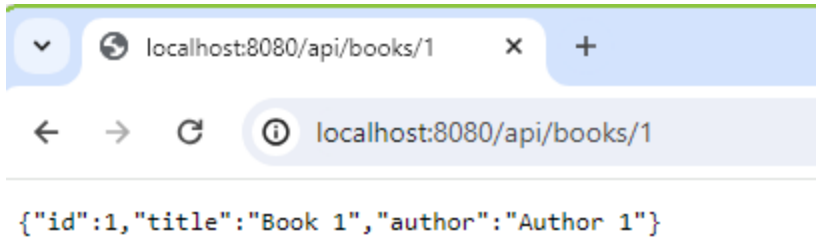
<input type="checkbox"/>	Name	Image	Status	CPU (%)	Port(s)	Last started	Actions
<input type="checkbox"/>	challenge-1-containe 2f4b54e03834	challenge-1	Exited (137)	0%	8080:80	2 days ago	
<input type="checkbox"/>	challenge2		Running (2/2)	0%		2 days ago	
<input type="checkbox"/>	app-1 abfd1cf682be	challenge2-app	Running	0%		2 days ago	
<input type="checkbox"/>	nginx-1 b3f204335b9a	nginx:latest	Running	0%	8080:80	2 days ago	

We can test if the service is running properly :





```
[{"id":1,"title":"Book 1","author":"Author 1"}, {"id":2,"title":"Book 2","author":"Author 2"}, {"id":3,"title":"Book 3","author":"Author 3"}]
```

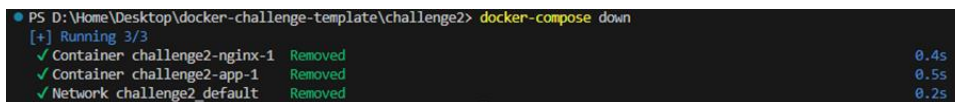


```
{"id":1,"title":"Book 1","author":"Author 1"}
```

We can see that the output of `http://localhost:8080/api/books/` and `http://localhost:8080/api/books/1` is as expected.

`docker-compose down`

Can shut it down.



```
PS D:\Home\Desktop\docker-challenge-template\challenge2> docker-compose down
[+] Running 3/3
✔ Container challenge2-nginx-1 Removed 0.4s
✔ Container challenge2-app-1 Removed 0.5s
✔ Network challenge2_default Removed 0.2s
```

Thanks for reading, and please contact me at [abo.chen@gmail.com](mailto:abo.chen@gmail.com) if you have any questions about the task.