Final Replacement:

Docker Explained

Prepared for

Rajani Phadtare

Prepared by

Jean-Pierre Nde-Forgwang,

Operating Systems, Cohort D

School of Advance Digital Technology

SAIT

22 April 2024

Introduction:

Docker Explained: It is essentially a tool and a method in which you can reliably package software into containers, which you can reliably run in another environment. There are three main components in relation to docker; Dockerfile, Image, and container.

- *Dockerfile*: "[C]ode that tells docker how to build an image." [1] It is like a recipe with a list of instructions that Docker will follow to assemble an image; think blueprint.

- *Image*: "[S]napshot of your software along with all of its dependencies down to the operating system level the image is immutable." [1] The image is a static snapshot of the blueprint, it will handle everything needed to run the application. The immutability means it cannot be changed once it's created. This allows for consistency in every environment. (Should the software change, a new Image must be built to incorporate those changes.)

- *Container*: "[S]oftware running in the real world." [1] When you start an image, it becomes a container, active, and capable of performing tasks in real time. Containers are running instances of the image and can be started, stopped, moved, and deleted.

By utilizing Docker, you're effectively sealing your project within a container. This container can then be transported to any other machine or environment. Upon arrival, the Dockerfile dictates how the local system should set up the Image. Subsequently, this Image springs to life as a Container—a live instance of your project ready to work on a new device.

Prerequisites:

1. Basic Understanding of Command-Line Interface (CLI) Usage
2. Access to a Computer with an Internet Connection
3. Root Access to Install Software
4. Modern Web Browser (Chrome, Firefox, Edge...)
5. A Text editor (Visual Studio Code, Notepad++, Sublime...)

6. Basic Git Knowledge

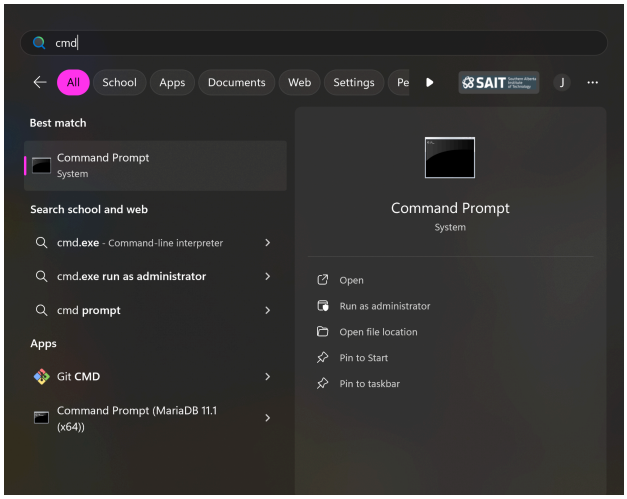
Section 1: Installing Docker

Download Docker -

- Search "download docker" on your web browser
- Download the docker for your respective system (Windows/MacOS/Linux...)

Installation Process -

- Allow Docker to make changes to your device.
- Wait



Verify Installation -

- Open the start menu
- Search "cmd" command prompt
- Type "docker –version" to verify docker installation

```
C:\Users\thefa>docker --version
Docker version 25.0.3, build 4debf41
```

Section 2: Challenge 3

Docker-compose.yml -

- The `docker-compose.yml` orchestrates a full-stack application consisting of a MariaDB database, a Node.js application, and a `Nginx` web server.

- `db service`: Utilizes the official `MariaDB` image. Environment variables are set from a .env file and passed to the container. Volumes persist data and initialize the database.

- `node-service`: Built from the ./api directory. Uses an .env file for environment variables. It has a dependency on the db service, ensuring the database is ready before starting.

- `nginx service`: Built from the ./nginx directory. Exposes `port 8080` on the host, routing traffic to port 80 in the container. It uses a custom `nginx.conf` for configuration and depends on the node service, ensuring it starts afterward.

.env File -

- The .env file contains environment variables for both the database and Node.js application. This file ensures sensitive information is not hard-coded into image builds or docker-compose.yml, adhering to best practices for security and maintainability.

Running the Stack

To start the stack, the following command is used:

```
docker-compose up -d
```

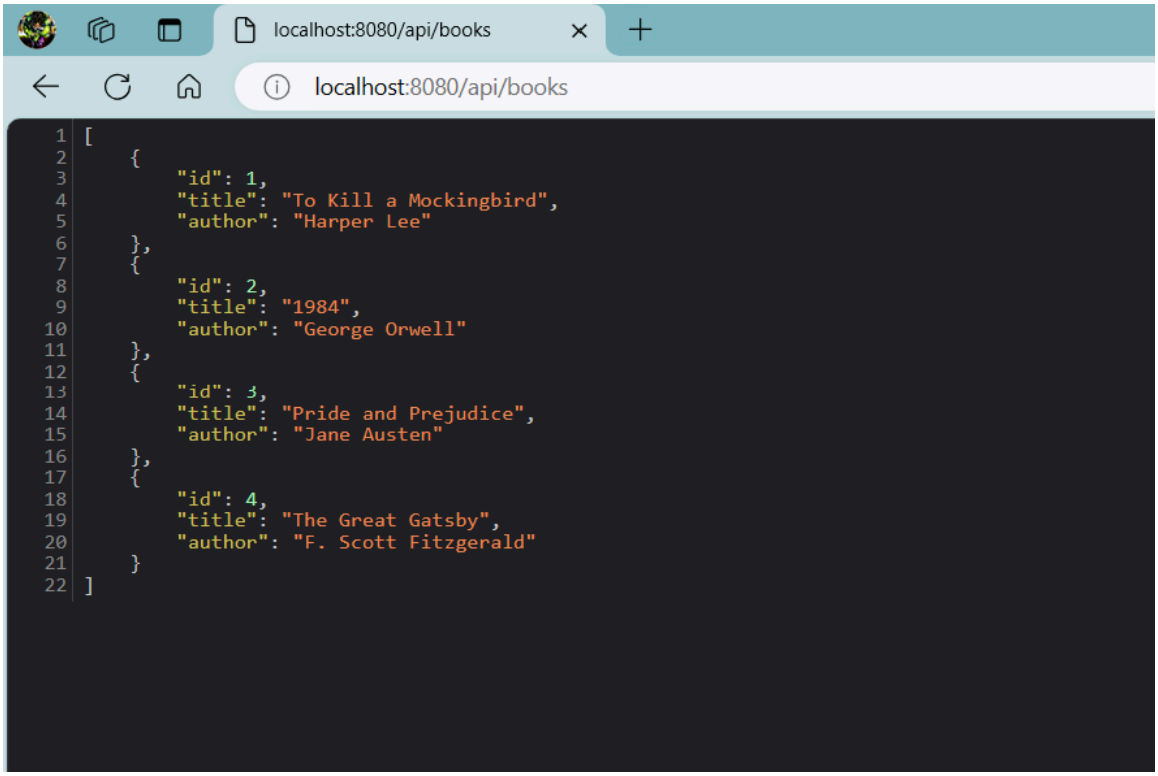
This command launches all services in detached mode, allowing them to run in the background.

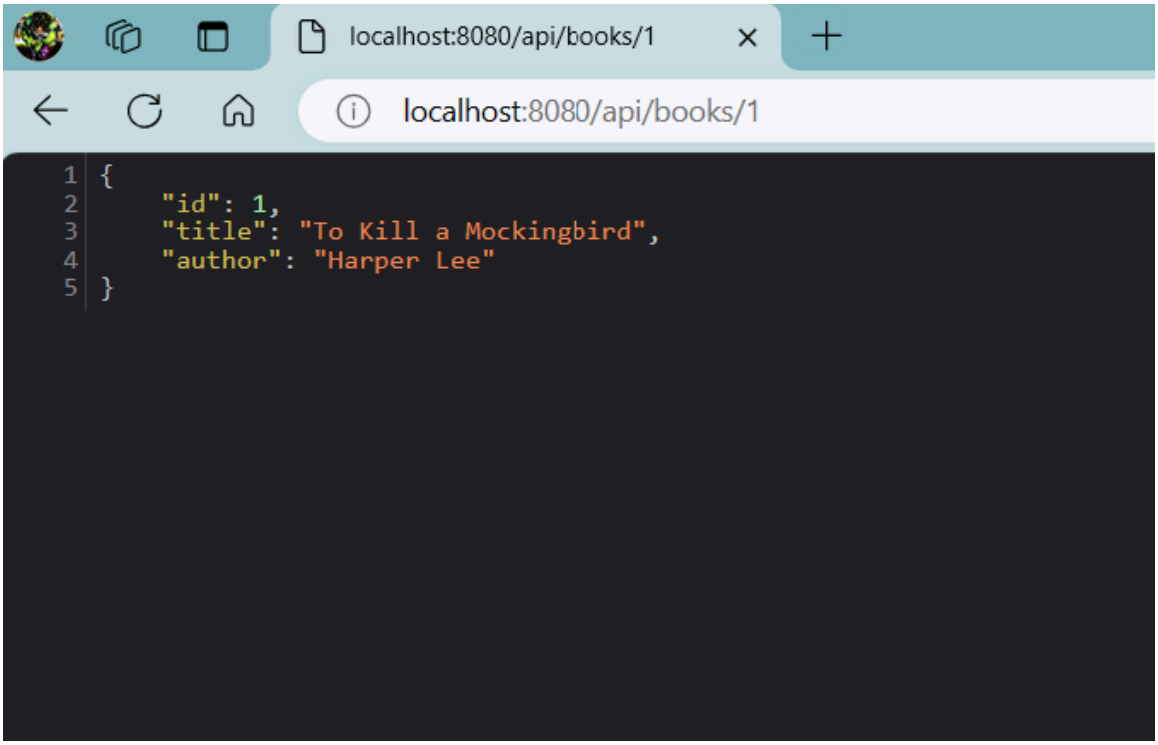
Testing Endpoints

After the stack is running, you can test the endpoints:

- http://localhost:8080/api/books should list all books.
- http://localhost:8080/api/books/1 should detail the book with ID 1.

If the results are not as expected, you would check the logs using `docker-compose logs` **`[service-name]`** and make necessary adjustments.


localhost:8080/api/books
localhost:8080/api/books
[
{
"id": 1,
"title": "To Kill a Mockingbird",
"author": "Harper Lee"
},
{
"id": 2,
"title": "1984",
"author": "George Orwell"
},
{
"id": 3,
"title": "Pride and Prejudice",
"author": "Jane Austen"
},
{
"id": 4,
"title": "The Great Gatsby",
"author": "F. Scott Fitzgerald"
}
]


localhost:8080/api/books/1
localhost:8080/api/books/1
{
"id": 1,
"title": "To Kill a Mockingbird",
"author": "Harper Lee"
}

```
MYSQL_ROOT_PASSWORD=JPspassword
MYSQL_USER=JPsusername
MYSQL_PASSWORD=JPspassword
MYSQL_DATABASE=mydatabase

DB_ROOT_PASSWORD=JPspassword
DB_HOST=db
DB_USERNAME=JPsusername
DB_PASSWORD=JPspassword
DB_DATABASE=mydatabase
```

```yaml
version: '3'  # Specifies the version of the Docker Compose file format

services:  # Defines the services that make up your application

  db:  # This service is for the MariaDB database
    image: mariadb  # Uses the official MariaDB image from Docker Hub
    env_file:  # Specifies the file from which to read environment variables
      - .env
    volumes:  # Mounts paths on the host machine to paths in the container
      - db_data:/var/lib/mysql  # Persists the database data in a volume
      - ./db/init/init.sql:/docker-entrypoint-initdb.d/init.sql  # Initializes the database with an SQL script

  node-service:  # This service is for the Node.js application
    build: ./api  # Specifies the directory where the Dockerfile for this service is located, which Docker Compose will use to build the image
    env_file:  # Uses the same .env file to set environment variables for this service
      - .env
    depends_on:  # Specifies that this service depends on the db service, ensuring the database is ready before the node-service starts
      - db

  nginx:  # This service is for the Nginx web server
    build: ./nginx  # Points to the directory containing the Dockerfile to build the Nginx image
    ports:  # Maps port 8080 on the host to port 80 in the container, making the Nginx server accessible from the host
      - "8080:80"
    volumes:  # Mounts the custom nginx configuration into the container
      - ./nginx/nginx.conf:/etc/nginx/conf.d/default.conf
    depends_on:  # Specifies that nginx should start after node-service is available
      - node-service

volumes:  # Declares named volumes that can be used by the services
  db_data:  # Named volume for persisting database data
```

Section 3: Challenge 4

Scaling with Docker Compose

The goal of Challenge 4 was to scale the node-service to have 3 running instances instead of just one. The docker-compose.yml file did not need any changes for this challenge because the scale option is a command-line argument, not a file configuration.

Scaling Command

To scale the node-service, the following command was used:

```
docker-compose up -d --scale node-service=3
```

This command starts the services and scales the node-service to 3 instances.

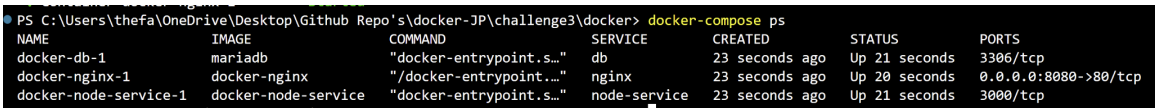
Testing Load Balancing

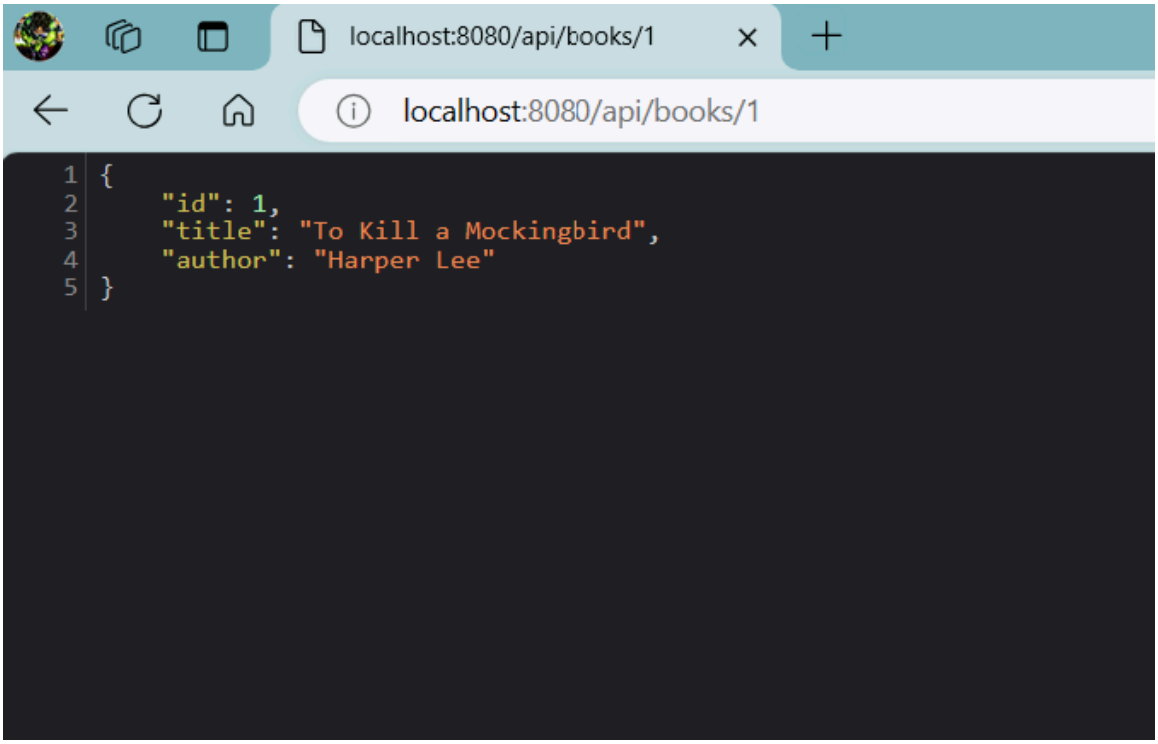
- To verify the service scaling, the `http://localhost:8080/api/stats` endpoint is hit multiple times. The expectation is to see different hostname values in the JSON response, which indicate requests being served by different instances of node-service.

Command Summary

- `docker-compose ps`: Lists the status of services, including scaled instances.
- `docker-compose down`: Stops and removes all containers and networks.
- `docker-compose logs`: Shows the logs of a service, useful for troubleshooting.

By observing different container IDs (hostname values) in the API response, it was confirmed that load balancing was effective, and the node-service instances were scaled correctly.






```
PS C:\Users\thefa\OneDrive\Desktop\Github Repo's\docker-JP\challenge3\docker> docker-compose up -d --scale node-service=3
s
 ✓ Container docker-node-service-3  Started
 ✓ Container docker-node-service-2  Started
 ✓ Container docker-nginx-1         Running
PS C:\Users\thefa\OneDrive\Desktop\Github Repo's\docker-JP\challenge3\docker> docker-compose ps
NAME                    IMAGE                COMMAND                  SERVICE         CREATED              STATUS              PORTS
docker-db-1             mariadb              "docker-entrypoint.s…"   db              About a minute ago   Up About a minute   3306/tcp
docker-nginx-1          docker-nginx         "/docker-entrypoint.…"   nginx           About a minute ago   Up About a minute   0.0.0.0:8080->80/tcp
docker-node-service-1   docker-node-service  "docker-entrypoint.s…"   node-service    About a minute ago   Up About a minute   3000/tcp
docker-node-service-2   docker-node-service  "docker-entrypoint.s…"   node-service    12 seconds ago       Up 10 seconds       3000/tcp
docker-node-service-3   docker-node-service  "docker-entrypoint.s…"   node-service    12 seconds ago       Up 9 seconds        3000/tcp
PS C:\Users\thefa\OneDrive\Desktop\Github Repo's\docker-JP\challenge3\docker>
```


```
{
    "status": "success",
    "contents": {
        "MemFree": 4565804,
        "MemAvailable": 6373404
    },
    "pid": 1,
    "hostname": "6522a242063b",
    "counter": 0
}
```

Section 4: Summary

Challenge 3 -

- Challenge 3 focused on the creation of a Docker Compose environment to support a full-stack application. This involved setting up a `MariaDB` database, a `Node.js` application server, and a `Nginx` webserver to handle incoming requests. The configuration was managed through a `docker-compose.yml` file, which defined the relationships and dependencies between services, and used an .env file for secure variable management. Proper functioning was confirmed through endpoint testing, ensuring that the application could serve data from the database through the `Node.js` layer and present it via `Nginx`.

Challenge 4 -

- In Challenge 4, the task was to scale the application's `Node.js` service. By applying Docker Compose's scaling capabilities, the service was expanded from one to three instances, which were then managed by `Nginx` for load balancing. The scaling process was validated by repeated requests to an API endpoint, which returned different container hostnames, indicating successful load distribution across the instances. This challenge provided practical insights into the scalability features of Docker and its role in enhancing application availability and load management.

Reference:

[1] Docker. "Docker for Beginners: Full Course," YouTube, uploaded by Fireship, July 21, 2020. [Online]. Available: https://www.youtube.com/watch?v=Gjnup-PuquQ. [Accessed: Mar. 28, 2024].

[2] Docker. "Docker Compose in 12 Minutes," YouTube, uploaded by Fireship, Mar. 2, 2017. [Online]. Available: https://www.youtube.com/watch?v=rIrNIzy6U_g. [Accessed: Mar. 28, 2024].

[3] Docker. "Overview of Docker Compose," Docker Documentation, Docker. [Online]. Available: https://docs.docker.com/get-started/overview/. [Accessed: Mar. 28, 2024].

[4] Docker. "Dockerfile reference," Docker Documentation, Docker. [Online]. Available: https://docs.docker.com/reference/dockerfile/. [Accessed: Mar. 28, 2024].

[5] Docker. "Get started with Docker Compose," Docker Documentation, Docker. [Online]. Available: https://docs.docker.com/get-started/08_using_compose/. [Accessed: Mar. 28, 2024].

[6] Docker. "Glossary," Docker Documentation, Docker. [Online]. Available: https://docs.docker.com/glossary/. [Accessed: Mar. 28, 2024].

Top