Mid-Term Replacement:

Docker Explained

Prepared for

Rajani Phadtare

Prepared by

Jean-Pierre Nde-Forgwang,

Operating Systems, Cohort D

School of Advance Digital Technology

SAIT

25 March 2024

Index

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 -

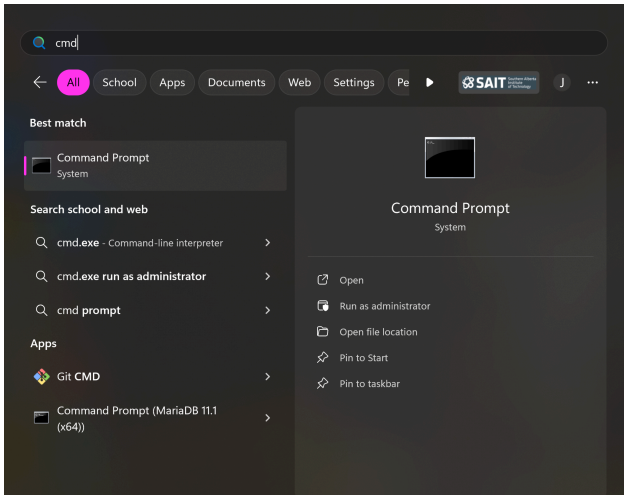
- Search "download docker" on your web browser
- Download the docker for your respective system (Windows/MacOS/Linux...)

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: Download Project

Clone a Repository

To download an existing project:

- Locate the Repository on GitHub: Find the project's GitHub page and locate the 'Code' button. Click it to reveal the cloning options.

- Copy the Repository URL: Choose HTTPS and copy the provided URL.

- Open Your Terminal/Command Prompt: Navigate to the directory where you want the project files to be placed. (ex. C:\Users\thefa>cd C:\Users\thefa\OneDrive\Desktop\Github Repo's)

- Clone the Repository: Enter git clone, followed by the URL you copied. (EX. git clone https://github.com/3mptySpac3/docker-challenge-template.git)

- You now have a local copy of the project, ready to be containerized with Docker.

Section 3: Challenge 1

index.html M

challenge1 > public > index.html > html

```
1   <!DOCTYPE html>
2   <html>
3     <head>
4       <title>Challenge 1</title>
5     </head>
6     <body>
7       <h1>Challenge 1</h1>
8       <p>Name: <span>Jean-Pierre Nde-Forgwang</span></p>
9       <p>ID: <span>000897309</span></p>
10    </body>
11
12
```

Dockerfile M

challenge1 > Dockerfile > ...

```
1   # Use the Nginx image from Docker Hub as the base image
2   FROM nginx:alpine
3
4   # Copy the 'public' folder contents into the Nginx server
5   COPY public /usr/share/nginx/html
6
7   # Expose port 80
8   EXPOSE 80
9
```

Index.html

- This file is the web page you want to serve. While this challenge uses a basic HTML file, Docker can serve any static website you place in the public directory.

Dockerfile

- **“`FROM nginx:alpine`”:** This line specifies the base image from which you are building your Docker image. Alpine Linux was chosen because it's a lightweight, security-oriented Linux distribution that's ideal for use with Docker containers.

- **“`COPY public /usr/share/nginx/html`”:** This instruction copies files from your local public directory (relative to the Dockerfile's location) into the container's file system at the specified path (/usr/share/nginx/html). This is the default directory where Nginx serves static files from. As a result, the contents of your public folder become the content that Nginx will serve over the web.

- **“`EXPOSE 80`”:** This tells Docker that the container listens on port 80 at runtime. This is standard for HTTP web traffic.

Run Dockerfile

In your text editors terminal:

- Build: “docker build -t name/webserver .”
- Run: “docker run --name mywebserver -d -p 8080:80 name/webserver”

Example:

- Build: “*docker build -t jpsdocker/webserver*”
- Run: “*docker run --name mywebserver -d -p 8080:80 jpsdocker/webserver*”

Explanation:

- *--name mywebserver:* Sets the name of the running container to mywebserver. This name can be used later if you want to stop or remove the container.

- *-d:* Runs the container in detached mode, meaning it runs in the background and doesn't block your terminal session.

- *-p 8080:80:* Maps port 8080 on your host machine to port 80 inside the container. This allows you to access the Nginx server running inside the container by visiting http://localhost:8080 in your web browser.

- *jpsdocker/webserver:* Specifies the image to create the container from, in this case, the image named jpsdocker/webserver that you previously built with the docker build command.

Section 4: Challenge 2

nginx.conf M

challenge2 > nginx.conf

```
events {}

http {
  server {
    listen 80;

    location / {
      proxy_pass http://webapp:8080;
      proxy_set_header Host $host;
      proxy_set_header X-Real-IP $remote_addr;
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
      proxy_set_header X-Forwarded-Proto $scheme;
    }
  }
}
```

Dockerfile M

challenge2 > Dockerfile > ...

```
# Choose the Node.js base image
FROM node:14

# Set the working directory inside the container
WORKDIR /usr/src/app

# Copy package.json and package-lock.json
COPY package*.json ./

# Install dependencies
RUN npm install

# Bundle app source
COPY . .

# Your app binds to port 8080, so you'll use the EXPOSE instruction to have it mapped by the docker daemon
EXPOSE 8080

CMD [ "node", "server.js" ]
```

docker-compose.yml M

challenge2 > docker-compose.yml

```
version: '3'
services:
  webapp:
    build: .
    environment:
      - PORT=8080
  nginx:
    image: nginx:alpine
    ports:
      - "8080:80"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf:ro
    depends_on:
      - webapp

```

Nginx.conf

```
events {}

http {
  server {
    listen 80;

    location / {
      proxy_pass http://webapp:8080;
      proxy_set_header Host $host;
      proxy_set_header X-Real-IP $remote_addr;
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
      proxy_set_header X-Forwarded-Proto $scheme;
    }
  }
}
```

- Configures Nginx to listen on port 80. It sets up a reverse proxy to the webapp service, redirecting traffic from Nginx to your Node.js app which listens on port 8080 within the Docker network. The proxy_set_header directives are used to pass on information about the original request to the proxied server (webapp).

Dockerfile

- `FROM node 14`: This line tells Docker to start with the official Node.js version 14 image. This image includes the Node.js runtime and npm package manager.

- `WORKDIR /usr/src/app`: Sets the working directory inside your Docker container to /usr/src/app. Future commands will run in this directory.

- `COPY package*.json ./`: Copies both package.json and package-lock.json (if available) to the working directory in the container. These files define your Node.js app's dependencies.

- `RUN npm install`: Installs your app's dependencies inside the container as defined in package*.json.

- `COPY . .`: Copies the rest of your app's source code into the container.

- `EXPOSE 8080`: Informs Docker that the container listens on port 8080 at runtime. This is the port that your Node.js application will use.

- `CMD [ "node", "server.js" ]`: The command that starts your Node.js app. It runs server.js using Node.
-

Docker-Compose.yml

- The docker-compose.yml file defines two services: the web application itself (webapp) and the Nginx reverse proxy (nginx).

```yaml
version: '3'
services:
  webapp:
    build: .
    environment:
      - PORT=8080
```

- This defines the webapp service and tells docker-compose to build it using the Dockerfile in the current directory. It sets an environment variable PORT to 8080, which is used by the Node.js application.

```yaml
  nginx:
    image: nginx:alpine
    ports:
      - "8080:80"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf:ro
    depends_on:
      - webapp
```

- Defines the nginx service, using the nginx:alpine image. It maps port 8080 on the host to port 80 on the container, allowing access to Nginx from the host machine on port 8080. It mounts the nginx.conf file as a read-only volume at the appropriate location for Nginx configuration. The depends_on setting indicates that nginx should only start after the webapp has started.

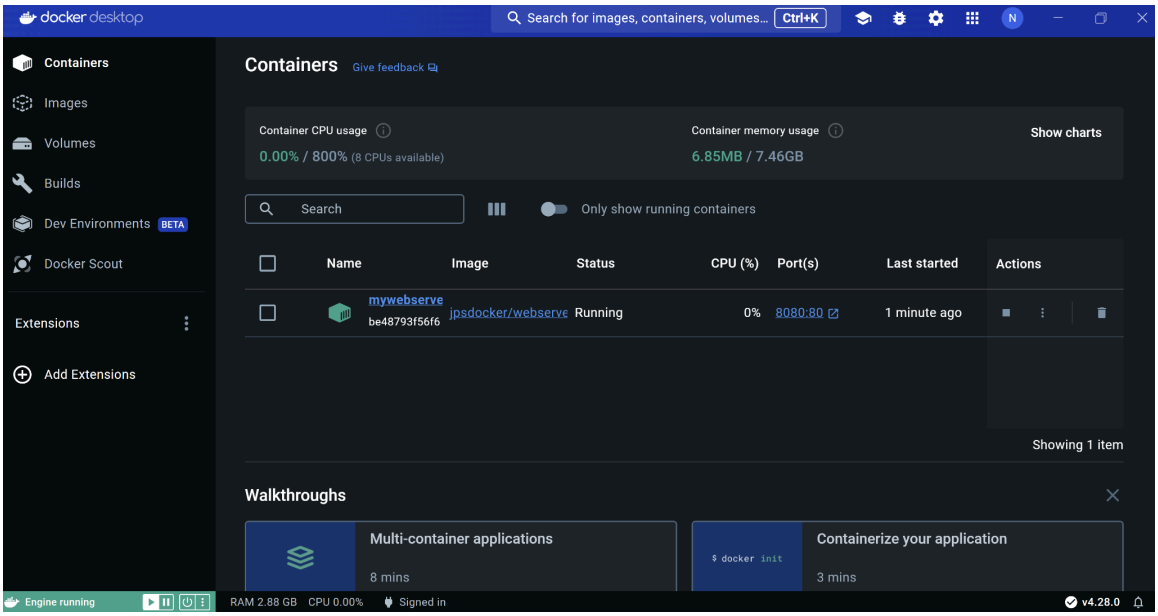
Commands

- *docker-compose up:* start all the services defined in your docker-compose.yml.
- *docker-compose up --build:* rebuild the images.
- *docker-compose up -d:* run the services in the background (in "detached" mode).
- *docker-compose down:* This command will stop and remove the containers, networks, volumes, and images created by 'up'.
- *docker-compose stop:* This will stop the running containers without removing them.
- *docker-compose start:* Start running the container.

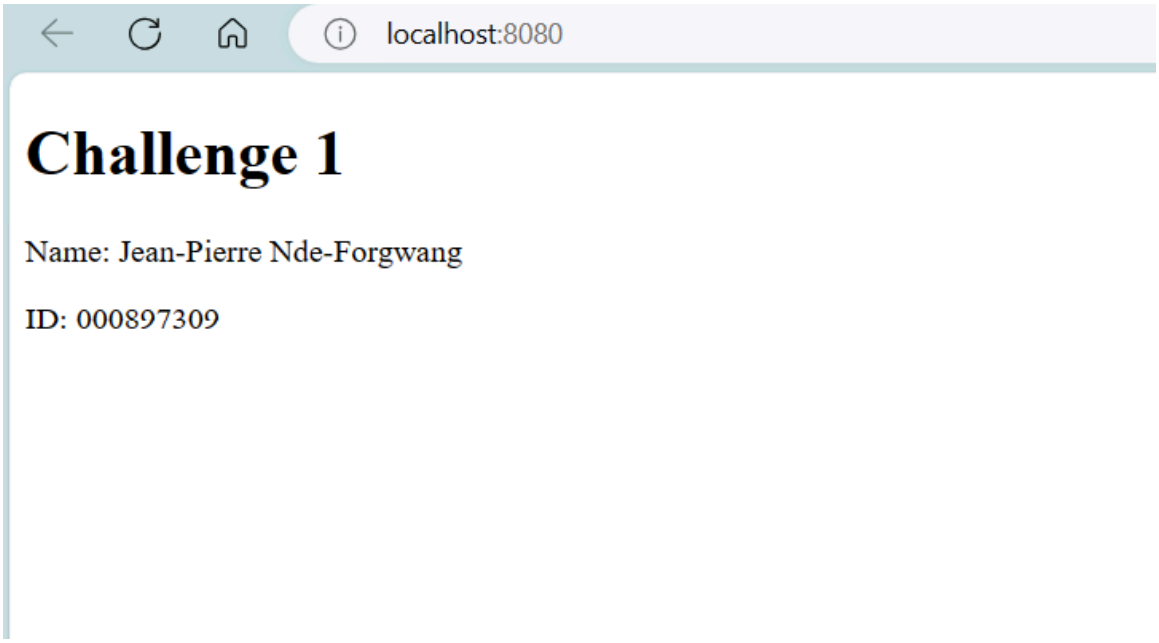
Conclusion:

- This guide has introduced you to Docker, showing you how to use it for developing, deploying, and scaling applications. You've learned the basics of Dockerfiles, Images, and Containers, and applied these concepts in practical challenges. By containerizing a simple web server and a dynamic Node.js application, you've seen firsthand how Docker ensures consistency across different environments.

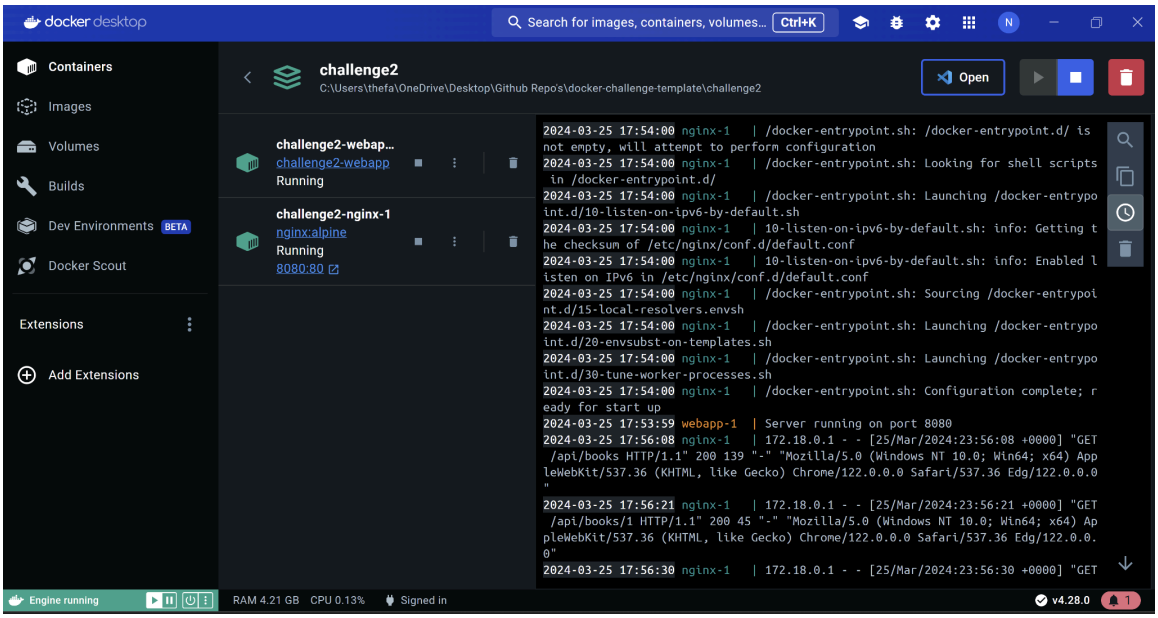
Challenge 1: http://localhost:8080/

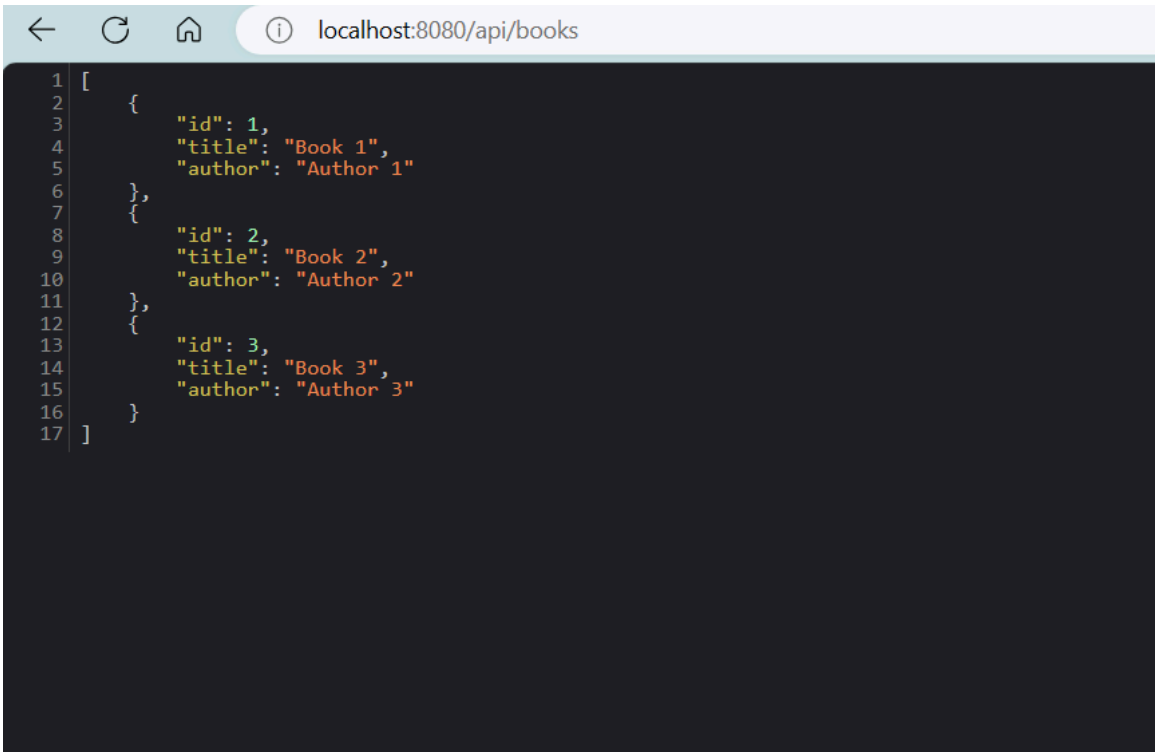


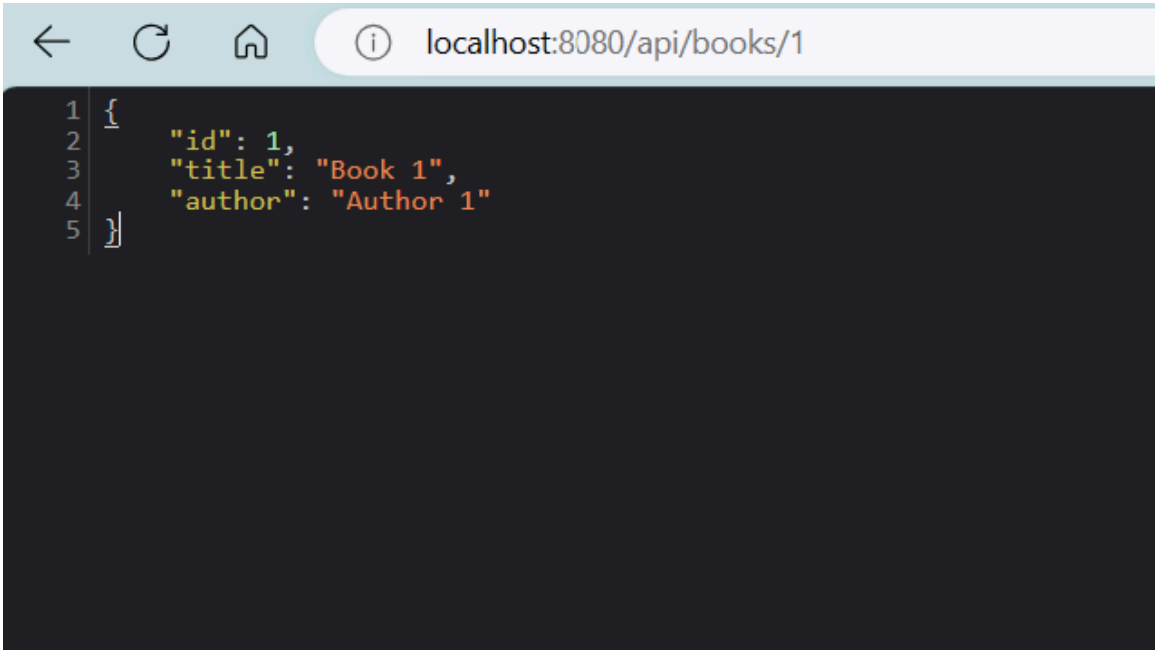
```
PS C:\Users\thefa\OneDrive\Desktop\Github Repo's\docker-challenge-template\challenge1> docker build -t jpsdocker/webserver .
[+] Building 9.3s (8/8) FINISHED                                                                                    docker:default
 => [internal] load build definition from Dockerfile                                                                         0.3s
 => => transferring dockerfile: 244B                                                                                         0.1s
 => [internal] load metadata for docker.io/library/nginx:alpine                                                              2.7s
 => [auth] library/nginx:pull token for registry-1.docker.io                                                                 0.0s
 => [internal] load .dockerignore                                                                                            0.1s
 => => transferring context: 2B                                                                                              0.0s
 => [internal] load build context                                                                                            0.1s
 => => transferring context: 278B                                                                                            0.0s
 => [1/2] FROM docker.io/library/nginx:alpine@sha256:31bad00311cb5eeb8a6648beadcf67277a175da89989f14727420a80e2e76742        5.8s
 => => resolve docker.io/library/nginx:alpine@sha256:31bad00311cb5eeb8a6648beadcf67277a175da89989f14727420a80e2e76742        0.1s
 => => sha256:ed3e62e73b33c9cfa4b253060771e4a9eebb751ab438052f197e847b4553a9ac 1.90MB / 1.90MB                               0.6s
 => => sha256:cb3218c8a053881bd00f4fa93e9f87e2c6204761e391b3aafc942f104362e538 2.50kB / 2.50kB                               0.0s
 => => sha256:e289a478ace02cd72f0a71a5b2ec0594495e1fae85faa10aae3b0da530812608 11.74kB / 11.74kB                             0.0s
 => => sha256:31bad00311cb5eeb8a6648beadcf67277a175da89989f14727420a80e2e76742 8.71kB / 8.71kB                               0.0s
 => => sha256:5126dce06df729f9a22956013e160f8b581d47095beec332d647a5c1119b2411 629B / 629B                                   0.5s
 => => sha256:619be1103602d98e1963557998c954c892b3872986c27365e9f651f5bc27cab8 3.40MB / 3.40MB                               2.5s
 => => sha256:1d0dd2dc2265a581798226f7c79d134ac797f42db3f934dd4af1d38a6b89ce5c 957B / 957B                                   0.9s
 => => sha256:2b1ab92f023179da00446365a60daa60d72a1edeb697fb81811e086eba2e0170 392B / 392B                                   0.9s
 => => sha256:6eba808ac059320c42179a6590b021f8695d3f12c2afa8745e219f635acf19d4 1.21kB / 1.21kB                               1.1s
 => => sha256:57038e85fbb88e96e34a84b125e568f540437561adb363fa791ff9e94e153dc1 1.40kB / 1.40kB                               1.2s
 => => sha256:eec94c9845c062c9f3495ce861d2bd9507bd7a13710de6b89a195a6b089ea8a7 12.65MB / 12.65MB                             3.8s
 => => extracting sha256:619be1103602d98e1963557998c954c892b3872986c27365e9f651f5bc27cab8                                    0.3s
 => => extracting sha256:ed3e62e73b33c9cfa4b253060771e4a9eebb751ab438052f197e847b4553a9ac                                    0.7s
 => => extracting sha256:5126dce06df729f9a22956013e160f8b581d47095beec332d647a5c1119b2411                                    0.0s
 => => extracting sha256:1d0dd2dc2265a581798226f7c79d134ac797f42db3f934dd4af1d38a6b89ce5c                                    0.0s
 => => extracting sha256:2b1ab92f023179da00446365a60daa60d72a1edeb697fb81811e086eba2e0170                                    0.0s
 => => extracting sha256:6eba808ac059320c42179a6590b021f8695d3f12c2afa8745e219f635acf19d4                                    0.0s
 => => extracting sha256:57038e85fbb88e96e34a84b125e568f540437561adb363fa791ff9e94e153dc1                                    0.0s
 => => extracting sha256:eec94c9845c062c9f3495ce861d2bd9507bd7a13710de6b89a195a6b089ea8a7                                    0.7s
 => [2/2] COPY public /usr/share/nginx/html                                                                                  0.1s
 => exporting to image                                                                                                       0.1s
 => => exporting layers                                                                                                      0.1s
 => => writing image sha256:0083706f3e11994824cf17e0a9467be136d976ea818e41d3d309b60bbdcfd575                                 0.0s
 => => naming to docker.io/jpsdocker/webserver                                                                               0.0s

What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout quickview
PS C:\Users\thefa\OneDrive\Desktop\Github Repo's\docker-challenge-template\challenge1>
```



Challenge 2: http://localhost:8080/api/books






localhost:8080/api/books/1
{
"id": 1,
"title": "Book 1",
"author": "Author 1"
}

Glossary: [6]

Compose	Compose is a tool for defining and running complex applications with Docker. With Compose, you define a multi-container application in a single file, then spin your application up in a single command which does everything that needs to be done to get it running. *Also known as Docker Compose*
Docker	The term Docker can refer to • The Docker project as a whole, which is a platform for developers and sysadmins to develop, ship, and run applications • The docker daemon process running on the host which manages images and containers (also called Docker Engine)

Docker Business	Docker Business is a Docker subscription. Docker Business offers centralized management and advanced security features for enterprises that use Docker at scale. It empowers leaders to manage their Docker development environments and accelerate their secure software supply chain initiatives.
Docker Desktop	Docker Desktop is an easy-to-install, lightweight Docker development environment. Docker Desktop is available for Mac, Windows, and Linux, providing developers a consistent experience across platforms. Docker Desktop includes Docker Engine, Docker CLI client, Docker Compose, Docker Content Trust, Kubernetes, and Credential Helper. Docker Desktop works with your choice of development tools and languages and gives you access to a vast library of certified images and templates in Docker Hub. This enables development teams to extend their environment to rapidly auto-build, continuously integrate, and collaborate using a secure repository.
Docker Desktop for Linux	Docker Desktop for Linux is an an easy-to-install, lightweight Docker development environment designed specifically for Linux machines. It's the best solution if you want to build, debug, test, package, and ship Dockerized applications on a Linux machine.
Docker Desktop for Mac	Docker Desktop for Mac is an easy-to-install, lightweight Docker development environment designed specifically for the Mac. A native Mac application, Docker Desktop for Mac uses the macOS Hypervisor framework, networking, and filesystem. It's the best solution if you want to build, debug, test, package, and ship Dockerized applications on a Mac.
Docker Desktop for Windows	Docker Desktop for Windows is an easy-to-install, lightweight Docker development environment designed specifically for Windows systems that support WSL 2 and Microsoft Hyper-V. Docker Desktop for Windows uses WSL 2 or Hyper-V for virtualization. Docker Desktop for Windows is the best solution if you want to build, debug, test, package, and ship Dockerized applications from Windows machines.

Docker Hub	Docker Hub is a centralized resource for working with Docker and its components. It provides the following services: • A registry to host Docker images • User authentication • Automated image builds and workflow tools such as build triggers and web hooks • Integration with GitHub and Bitbucket • Security vulnerability scanning
Docker ID	Your free Docker ID grants you access to Docker Hub repositories and some beta programs. All you need is an email address.
Docker Official Images	The Docker Official Images are a curated set of Docker repositories hosted on Docker Hub. Docker, Inc. sponsors a dedicated team that is responsible for reviewing and publishing all content in the Docker Official Images. This team works in collaboration with upstream software maintainers, security experts, and the broader Docker community.
Docker Open Source Images	Docker Open Source Images are published and maintained by organizations that are a member of the Docker Open Source Program.
Docker Personal	Docker Personal is a Docker subscription. With its focus on the open-source communities, individual developers, education, and small businesses, Docker Personal will continue to allow free use of Docker components - including the Docker CLI, Docker Compose, Docker Engine, Docker Desktop, Docker Hub, Kubernetes, Docker Build and Docker BuildKit, Docker Official Images, Docker Scan, and more.
Docker Pro	Docker Pro is a Docker subscription. Docker Pro enables individual developers to get more control of their development environment and provides an integrated and reliable developer experience. It reduces the amount of time developers spend on mundane and repetitive tasks and empowers developers to spend more time creating value for their customers.
Docker Team	Docker Team is a Docker subscription. Docker Team offers capabilities for collaboration, productivity, and security across organizations. It enables groups of developers to unlock the full power of collaboration and sharing combined with essential security features and team management capabilities.
Docker Trusted Content Program	The Docker Trusted Content Program verifies content through four programs, Docker Official Images, Docker Verified Publisher Images, Docker Open Source Images, and Custom Official Images.
Docker Verified Publisher Images	Docker Verified Publisher Images are confirmed by Docker to be from a trusted software publishers that are partners in the Verified Publisher program. Docker Verified Publisher Images are identified by the Verified Publisher badge included on the Docker Hub repositories.
Docker subscription	Docker subscription tiers, sometimes referred to as plans, include Personal, Pro, Team, and Business. For more details, see Docker subscription overview.
Dockerfile	A Dockerfile is a text document that contains all the commands you would normally execute manually in order to build a Docker image. Docker can build images automatically by reading the instructions from a Dockerfile.

ENTRYPOINT	In a Dockerfile, an `ENTRYPOINT` is an optional definition for the first part of the command to be run. If you want your Dockerfile to be runnable without specifying additional arguments to the `docker run` command, you must specify either `ENTRYPOINT`, `CMD`, or both.
	
	• If `ENTRYPOINT` is specified, it is set to a single command. Most official Docker images have an `ENTRYPOINT` of `/bin/sh` or `/bin/bash`. Even if you do not specify `ENTRYPOINT`, you may inherit it from the base image that you specify using the `FROM` keyword in your Dockerfile. To override the `ENTRYPOINT` at runtime, you can use `--entrypoint`. The following example overrides the entrypoint to be `/bin/ls` and sets the `CMD` to `-l /tmp`.
	
	`$ docker run --entrypoint=/bin/ls ubuntu -l /tmp`
	
	• `CMD` is appended to the `ENTRYPOINT`. The `CMD` can be any arbitrary string that is valid in terms of the `ENTRYPOINT`, which allows you to pass multiple commands or flags at once. To override the `CMD` at runtime, just add it after the container name or ID. In the following example, the `CMD` is overridden to be `/bin/ls -l /tmp`.
	
	`$ docker run ubuntu /bin/ls -l /tmp`
	
	In practice, `ENTRYPOINT` is not often overridden. However, specifying the `ENTRYPOINT` can make your images more flexible and easier to reuse.

SSH	SSH (secure shell) is a secure protocol for accessing remote machines and applications. It provides authentication and encrypts data communication over insecure networks such as the Internet. SSH uses public/private key pairs to authenticate logins.

Union file system	Union file systems implement a union mount and operate by creating layers. Docker uses union file systems in conjunction with copy-on-write techniques to provide the building blocks for containers, making them very lightweight and fast. For more on Docker and union file systems, see Docker and OverlayFS in practice. Example implementations of union file systems are UnionFS and OverlayFS.
amd64	AMD64 is AMD's 64-bit extension of Intel's x86 architecture, and is also referred to as x86_64 (or x86-64).
arm64	ARM64 is the 64-bit extension of the ARM CPU architecture. arm64 architecture is used in Apple silicon machines.
base image	A base image has no parent image specified in its Dockerfile. It is created using a Dockerfile with the `FROM scratch` directive.
btrfs	btrfs (B-tree file system) is a Linux filesystem that Docker supports as a storage backend. It is a copy-on-write filesystem.
build	Build is the process of building Docker images using a Dockerfile. The build uses a Dockerfile and a "context". The context is the set of files in the directory in which the image is built.
cgroups	cgroups is a Linux kernel feature that limits, accounts for, and isolates the resource usage (CPU, memory, disk I/O, network, etc.) of a collection of processes. Docker relies on cgroups to control and isolate resource limits. *Also known as control groups*

cluster	A cluster is a group of machines that work together to run workloads and provide high availability.
container	A container is a runtime instance of a docker image. A Docker container consists of • A Docker image • An execution environment • A standard set of instructions The concept is borrowed from shipping containers, which define a standard to ship goods globally. Docker defines a standard to ship software.
container image	Docker images are the basis of containers. An image is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime. An image typically contains a union of layered filesystems stacked on top of each other.

copy-on-write	Docker uses a copy-on-write technique and a union file system for both images and containers to optimize resources and speed performance. Multiple copies of an entity share the same instance and each one makes only specific changes to its unique layer. Multiple containers can share access to the same image, and make container-specific changes on a writable layer which is deleted when the container is removed. This speeds up container start times and performance. Images are essentially layers of filesystems typically predicated on a base image under a writable layer, and built up with layers of differences from the base image. This minimizes the footprint of the image and enables shared development. For more about copy-on-write in the context of Docker, see Understand images, containers, and storage drivers.
filesystem	A file system is the method an operating system uses to name files and assign them locations for efficient storage and retrieval. Examples : • Linux : overlay2, extfs, btrfs, zfs • Windows : NTFS • macOS : APFS

image	Docker images are the basis of containers. An Image is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime. An image typically contains a union of layered filesystems stacked on top of each other. An image does not have state and it never changes.
invitee	People who have been invited to join an organization, but have not yet accepted their invitation.
layer	In an image, a layer is modification to the image, represented by an instruction in the Dockerfile. Layers are applied in sequence to the base image to create the final image. When an image is updated or rebuilt, only layers that change need to be updated, and unchanged layers are cached locally. This is part of why Docker images are so fast and lightweight. The sizes of each layer add up to equal the size of the final image.
libcontainer	libcontainer provides a native Go implementation for creating containers with namespaces, cgroups, capabilities, and filesystem access controls. It allows you to manage the lifecycle of the container performing additional operations after the container is created.
libnetwork	libnetwork provides a native Go implementation for creating and managing container network namespaces and other network resources. It manages the networking lifecycle of the container performing additional operations after the container is created.
member	The people who have received and accepted invitations to join an organization. Member can also refer to members of a team within an organization.
namespace	A Linux namespace is a Linux kernel feature that isolates and virtualizes system resources. Processes which are restricted to a namespace can only interact with resources or processes that are part of the same namespace. Namespaces are an important part of Docker's isolation model. Namespaces exist for each type of resource, including `net` (networking), `mnt` (storage), `pid` (processes), `uts` (hostname control), and `user` (UID mapping). For more information about namespaces, see Docker run reference and Isolate containers with a user namespace.

node	A node is a physical or virtual machine running an instance of the Docker Engine in swarm mode. Manager nodes perform swarm management and orchestration duties. By default manager nodes are also worker nodes. Worker nodes execute tasks.
organization	An organization is a collection of teams and repositories that can be managed together. Docker users become members of an organization when they are assigned to at least one team in the organization.
organization name	The organization name, sometimes referred to as the organization namespace or the org ID, is the unique identifier of a Docker organization.
overlay network driver	Overlay network driver provides out of the box multi-host network connectivity for Docker containers in a cluster.
overlay storage driver	OverlayFS is a filesystem service for Linux which implements a union mount for other file systems. It is supported by the Docker daemon as a storage driver.
parent image	An image's parent image is the image designated in the `FROM` directive in the image's Dockerfile. All subsequent commands are based on this parent image. A Dockerfile with the `FROM scratch` directive uses no parent image, and creates a base image.
persistent storage	Persistent storage or volume storage provides a way for a user to add a persistent layer to the running container's file system. This persistent layer could live on the container host or an external device. The lifecycle of this persistent layer is not connected to the lifecycle of the container, allowing a user to retain state.

registry	A Registry is a hosted service containing repositories of images which responds to the Registry API. The default registry can be accessed using a browser at Docker Hub or using the `docker search` command.
repository	A repository is a set of Docker images. A repository can be shared by pushing it to a registry server. The different images in the repository can be labeled using tags. Here is an example of the shared nginx repository and its tags.
seats	The number of seats refers to the number of planned members within an organization.
service	A service is the definition of how you want to run your application containers in a swarm. At the most basic level, a service defines which container image to run in the swarm and which commands to run in the container. For orchestration purposes, the service defines the "desired state", meaning how many containers to run as tasks and constraints for deploying the containers. Frequently a service is a microservice within the context of some larger application. Examples of services might include an HTTP server, a database, or any other type of executable program that you wish to run in a distributed environment.
service account	A service account is a Docker ID used for automated management of container images or containerized applications. Service accounts are typically used in automated workflows, and do not share Docker IDs with the members in a Docker Team or Docker Business subscription plan.

service discovery	Swarm mode container discovery is a DNS component internal to the swarm that automatically assigns each service on an overlay network in the swarm a VIP and DNS entry. Containers on the network share DNS mappings for the service through gossip so any container on the network can access the service through its service name. You don't need to expose service-specific ports to make the service available to other services on the same overlay network. The swarm's internal load balancer automatically distributes requests to the service VIP among the active tasks.
swarm	A swarm is a cluster of one or more Docker Engines running in swarm mode.
swarm mode	Swarm mode refers to cluster management and orchestration features embedded in Docker Engine. When you initialize a new swarm (cluster) or join nodes to a swarm, the Docker Engine runs in swarm mode.
tag	A tag is a label applied to a Docker image in a repository. Tags are how various images in a repository are distinguished from each other.
task	A task is the atomic unit of scheduling within a swarm. A task carries a Docker container and the commands to run inside the container. Manager nodes assign tasks to worker nodes according to the number of replicas set in the service scale.
team	A team is a group of Docker users that belong to an organization. An organization can have multiple teams.

virtual machine	A virtual machine is a program that emulates a complete computer and imitates dedicated hardware. It shares physical hardware resources with other users but isolates the operating system. The end user has the same experience on a Virtual Machine as they would have on dedicated hardware. Compared to containers, a virtual machine is heavier to run, provides more isolation, gets its own set of resources and does minimal sharing. *Also known as VM*
volume	A volume is a specially-designated directory within one or more containers that bypasses the Union File System. Volumes are designed to persist data, independent of the container's life cycle. Docker therefore never automatically deletes volumes when you remove a container, nor will it "garbage collect" volumes that are no longer referenced by a container. *Also known as: data volume* There are three types of volumes: *host, anonymous, and named*: • A **host volume** lives on the Docker host's filesystem and can be accessed from within the container. • A **named volume** is a volume which Docker manages where on disk the volume is created, but it is given a name. • An **anonymous volume** is similar to a named volume, however, it can be difficult to refer to the same volume over time when it is an anonymous volume. Docker handles where the files are stored.
x86_64	x86_64 (or x86-64) refers to a 64-bit instruction set invented by AMD as an extension of Intel's x86 architecture. AMD calls its x86_64 architecture, AMD64, and Intel calls its implementation, Intel 64.

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