Containerizing Applications with Dockerfiles

Pasquale Domenico Colaianni
DTU Biosustain, Research Software Engineering

2025-11-26

Challenges of sharing software with others

POV: Reviewer

1. Obtain your software
   - git clone or download a .zip
2. Scan for files and instructions
   - Is this a Python codebase or ... ?
   - Do I have all tools installed? Interpreters, compilers, Poetry, pipenv
   - Do I have the *correct* version of each?
3. Is my OS (Linux/Mac/Win) supported?
4. Will the software affect the integrity of my system?
   - A bug might cause the software to write in the wrong path

POV: You

1. Hope the reviewer can execute the software
   - Create an environment as close as possible to yours
2. Hope the reviewer is allowed (and inclined) to install any missing parts

Goals

Formally:

- Environment consistency
  - Ability to recreate the same software environment
- Dependency management
  - Better than relying on documentation only
- Isolation
  - The packaged app is not affected by the rest of the system
  - Example: permissions in Android and iOS

Informally:

- Plan so that the reviewer must execute as few actions as possible to reproduce your results
  - Reduce the amount of requirements

What is Docker?

Docker is a software for packaging and running your application in an isolated environment.

The result can be easily shared with others.

Docker comes with a command line tool.

Advantages

Docker is widely used in bioinformatics and software development because it addresses several challenges that are common in these fields:

- Reproducibility
- Portability
- Software dependencies management
- Scalability
- Collaboration
- Version control of environments
- Integration with other bioinformatics tools (e.g. Nextflow)

The only solution? [skip]

Virtual machines (images, snapshots of)

- The hardware must support emulation:
  - CPU and IO, including GPU, storage, networking, etc. (VT-x, AMD-V, VT-d, AMD-Vi)
- They emulate an entire operating system
  - and not just your application: more memory is required

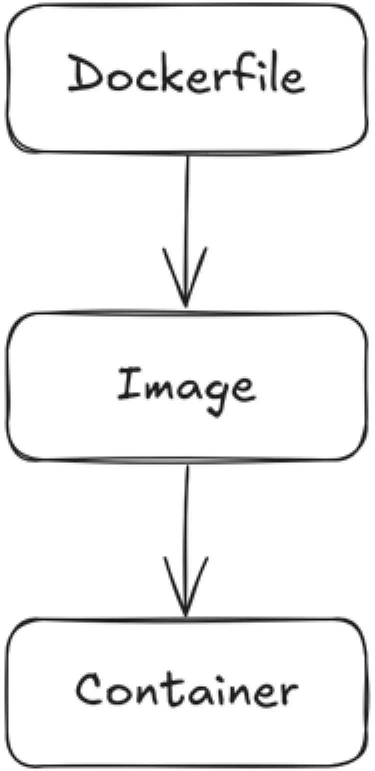
Docker containers

- They share the host's operating system
  - more memory available to your app
- Easier to package and share your work
- Faster to create, destroy, start, stop, etc.

Both solutions provide a consistent and isolated environment for your application.

Docker terminology

- Dockerfile
  - A text file that describes how to package and run your software
- Image
  - Created from a Dockerfile, with `docker build`
  - A package of your code and its dependencies
  - Is immutable: ideal for sharing with others
- Container
  - Created from an image, with `docker run`
  - A running instance of an image
  - Is mutable
    - Best if ephemeral, stateless



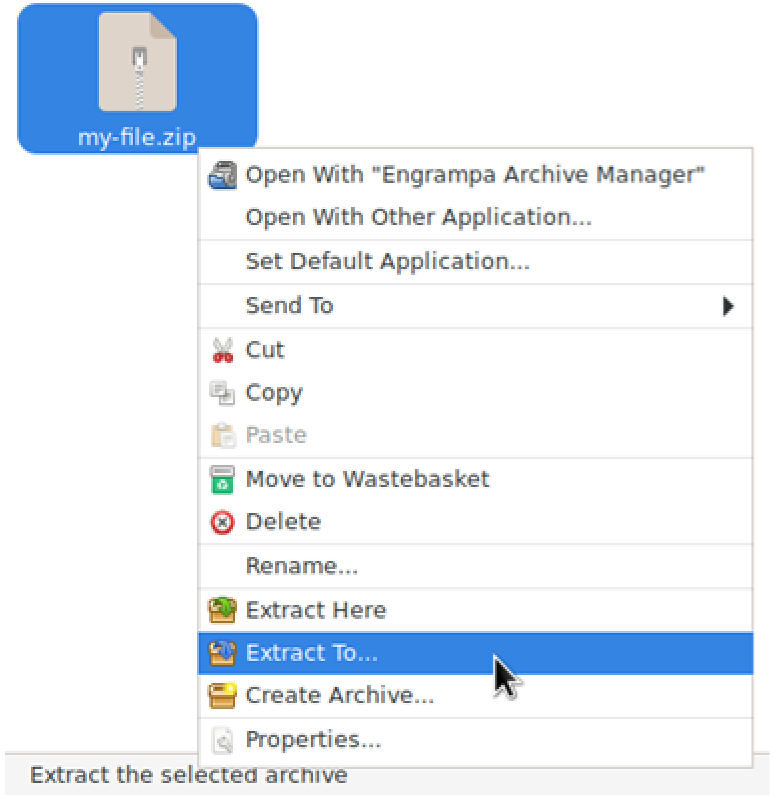
Glossary: https://docs.docker.com/reference/glossary/

Think of it like…

- Docker images as .zip files
- Docker containers as the extracted content

Point being: a .zip file can be extracted multiple times, to different destination folders. The content of those folders is the same.

People do not work directly on a .zip file. They work on the extracted content.


my-file.zip
Open With "Engrampa Archive Manager"
Open With Other Application...
Set Default Application...
Send To
Cut
Copy
Paste
Move to Wastebasket
Delete
Rename...
Extract Here
Extract To...
Create Archive...
Properties...
Extract the selected archive

Bonus term: Registry

A storage and distribution system of Docker images. Docker Hub is the default registry (docker.io).

Registry terms:

- repo name: related images
- tag: label identifying a specific version of an image

Example:

docker pull ~~docker.io/library/~~debian:12

docker pull debian:12

Dockerfile

It is a text file. It describes how to package and run your application.

```
FROM python:3.14
COPY main.py /app/main.py
CMD ["python", "/app/main.py"]
```

Find base images on Docker Hub:

- https://hub.docker.com/_/debian
- https://hub.docker.com/_/python
- https://hub.docker.com/_/ubuntu

No dependencies in this app.

How is the base image built? Where does it come from?

More instructions for Dockerfiles

- ADD - Add local or remote files and directories
- ENTRYPOINT - Specify default executable
- ENV - Set environment variables
- RUN - Execute build commands
- WORKDIR - Change working directory

All instructions: https://docs.docker.com/reference/dockerfile/

Images and containers - Commands

Given a Dockerfile, create an image:

```
docker build -t name[:tag] .
```

Given an image, create a container and start it:

```
docker run name[:tag]
```

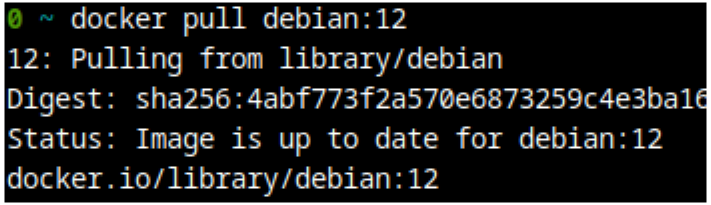
Examples

https://github.com/biosustain/dsp_docker_training

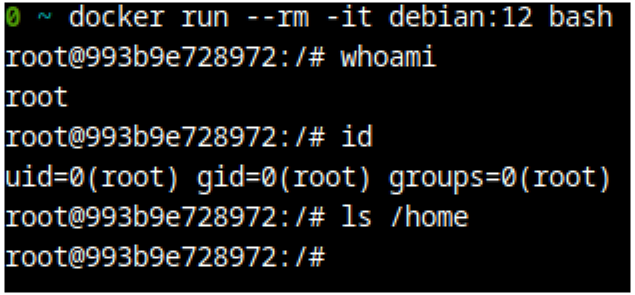
Subfolder: *course_contents/Dockerfile_examples*

Exploring a container

```
docker pull debian:12
docker run --rm -it debian:12 bash
```



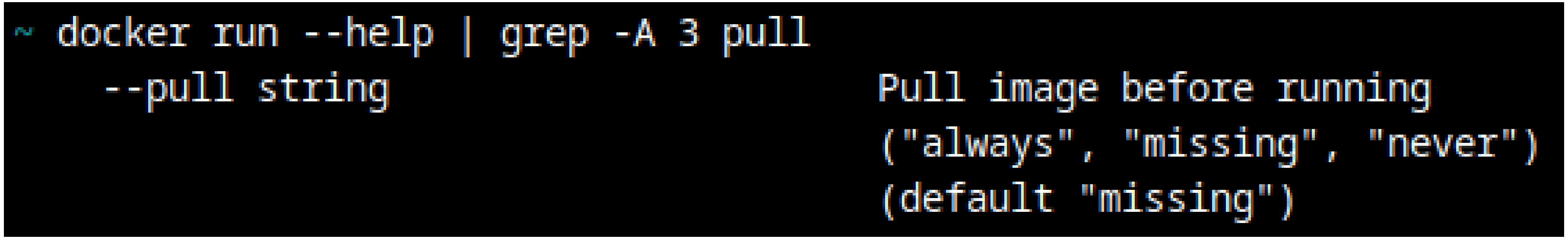
```
0 ~ docker pull debian:12
12: Pulling from library/debian
Digest: sha256:4abf773f2a570e6873259c4e3ba16
Status: Image is up to date for debian:12
docker.io/library/debian:12
```



```
0 ~ docker run --rm -it debian:12 bash
root@993b9e728972:/# whoami
root
root@993b9e728972:/# id
uid=0(root) gid=0(root) groups=0(root)
root@993b9e728972:/# ls /home
root@993b9e728972:/#
```

Enter `exit` to get out of the container. Option `--rm` removes the container.

Is docker pull needed?



```
~ docker run --help | grep -A 3 pull
    --pull string                                Pull image before running
                                                 ("always", "missing", "never")
                                                 (default "missing")
```

Running some commands…

… on the host:

```
+ hostname
work-laptop

+ ls -ahl /home
total 12K
drwxr-xr-x  3 root root 4.0K Nov 26  2023 .
drwxr-xr-x 19 root root 4.0K Feb  8 22:46 ..
drwx------ 42 user user 4.0K Feb 17 15:09 user

+ ip -o link show
lo
eth0
docker0
br-123dfb4a82c1
```

… within the container:

```
+ hostname
06dbdcf97766

+ ls -ahl /home
total 8.0K
drwxr-xr-x 2 root root 4.0K Dec 31 10:25 .
drwxr-xr-x 1 root root 4.0K Feb 17 14:14 ..


+ ip -o link show
ip: command not found
```

Try yourself with `docker run -it python:3.14 bash`

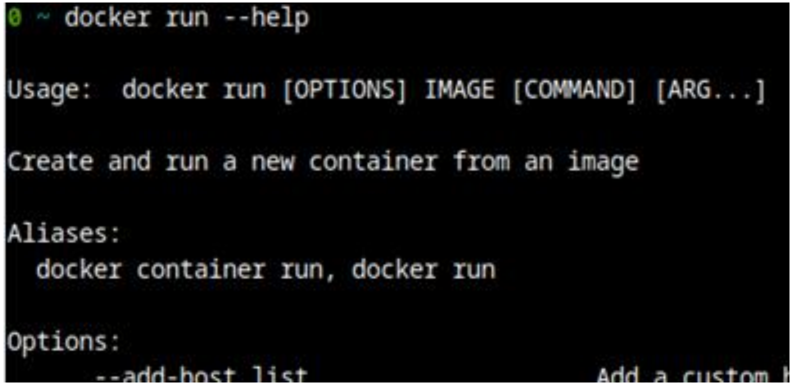
This underscores the isolation point made on slide 3.

Ask for help

```
docker build --help

docker run --help

docker ps --help
```



```
0 ~ docker run --help

Usage:  docker run [OPTIONS] IMAGE [COMMAND] [ARG...]

Create and run a new container from an image

Aliases:
  docker container run, docker run

Options:
      --add-host list                    Add a custom h
```

Each help page provides a list of Aliases and Options.

Why Reproducibility Matters

- Scientific results must be verifiable by others
  - Inconsistent environments make that impossible
- Many bioinformatics tools behave differently depending on dependency versions
  - Which can lead to irreproducible or incorrect biological conclusions
- Regulatory, clinical, and high-stakes research settings require traceable and auditable computational pipelines
- Long-term studies need to re-run analyses years later
  - Without reproducible environments, workflows often break as software ecosystems change

Reproducibility

Tracking the versions of your dependencies is very important.

- requirements.txt
    - Basic reproducibility: it tracks direct dependencies versions
- lock file (uv.lock, environment.lock, Pipfile.lock, poetry.lock)
    - Good reproducibility: it tracks direct and indirects dependencies versions

Pick one:

- https://docs.astral.sh/uv/ (strongly recommended)
- https://docs.conda.io/projects/conda/en/latest/commands/index.html
- https://pipenv.pypa.io/en/latest/commands.html
- https://python-poetry.org/docs/cli/

Reproducibility with conda [skip]

```
conda create --name app-with-conda python=3.12
conda activate app-with-conda
conda install conda-lock numpy
conda env export > environment.yml
conda-lock -f environment.yml -p linux-64
# To recreate the environment:
conda-lock install -n another-env -f conda-lock.yml
```

https://www.anaconda.com/blog/8-levels-of-reproducibility

Reproducibility with pipenv [skip]

```
pyenv local 3.13
python -m venv venv
source venv/bin/activate
pip install pipenv
pipenv install numpy
# It generated two files: Pipfile and
Pipfile.lock
```

```
FROM python:3.13
WORKDIR /app
COPY Pipfile.lock ./
RUN pip install pipenv && pipenv sync
COPY app.py ./
ENTRYPOINT ["pipenv", "run", "python", "app.py"]
```

- pyenv: https://github.com/pyenv/pyenv
  - Tool for installing and selecting versions of Python
- venv: https://docs.python.org/3/library/venv.html
  - Module from the standard library for creating virtual environments

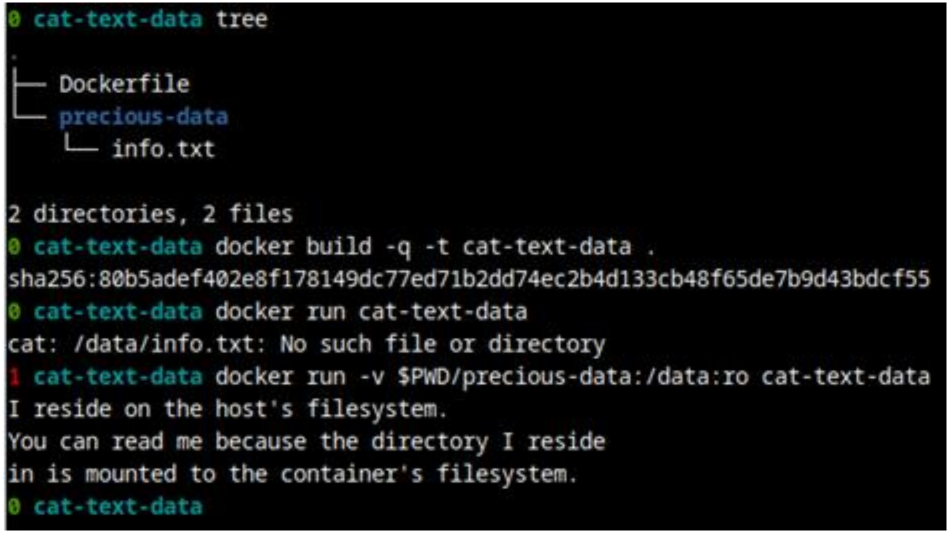
Reproducibility in practice

- Track the lock file in your git repo
- Do not share your venv or secrets: use .dockerignore and .gitignore
- Keep your dependencies fresh
  - Once in a while (e.g. every 6 months) try upgrading your dependencies
    - If all works as expected, update the lock file in git
  - Avoid surprises due to breaking changes
    - pandas: https://pandas.pydata.org/pandas-docs/stable/whatsnew/index.html
    - numpy: https://numpy.org/doc/stable/release.html
- docker build --pull --no-cache
  - Use a fresh base image and no cache for your "production" images (images meant to be shared with others)

Exposing data to/from a container

- Via bind mounts
  - https://docs.docker.com/engine/storage/bind-mounts/
  - Alternative: volumes https://docs.docker.com/engine/storage/volumes/

```
FROM debian:12-slim
CMD ["cat", "/data/info.txt"]
```



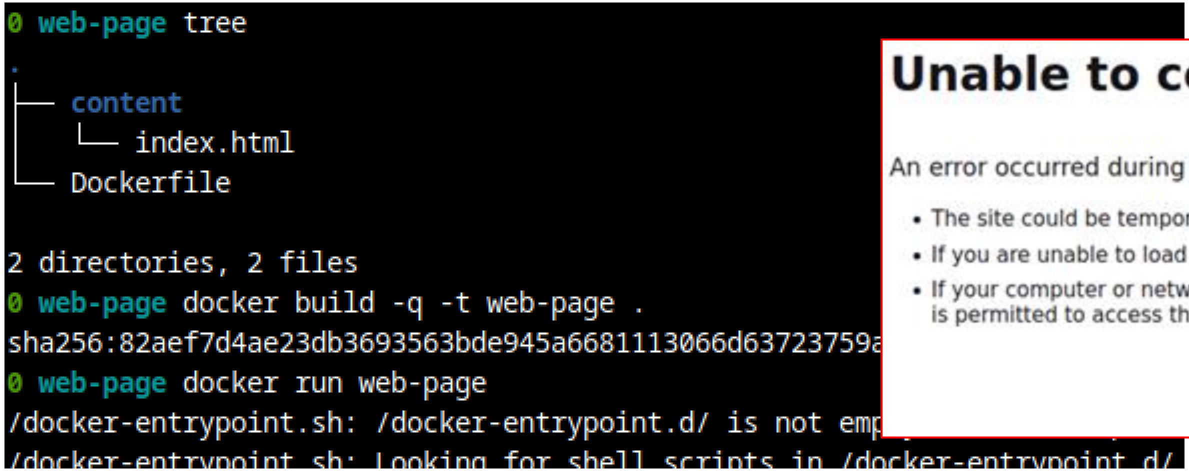
```
0 cat-text-data tree
.
├── Dockerfile
└── precious-data
    └── info.txt

2 directories, 2 files
0 cat-text-data docker build -q -t cat-text-data .
sha256:80b5adef402e8f178149dc77ed71b2dd74ec2b4d133cb48f65de7b9d43bdcf55
0 cat-text-data docker run cat-text-data
cat: /data/info.txt: No such file or directory
1 cat-text-data docker run -v $PWD/precious-data:/data:ro cat-text-data
I reside on the host's filesystem.
You can read me because the directory I reside
in is mounted to the container's filesystem.
0 cat-text-data
```

Network communication 1/3 [skip]

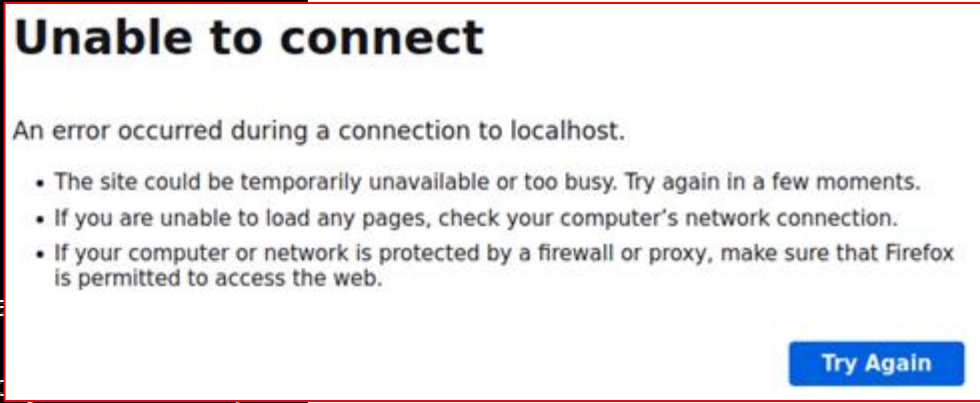
- Communication between containers
  - enabled by default for containers in the same bridge network
- Communication host <-> container
  - blocked by default; ports must be published

```
FROM nginx:stable
WORKDIR /usr/share/nginx/html
COPY content/index.html ./
```



```
0 web-page tree
.
├── content
│   └── index.html
└── Dockerfile

2 directories, 2 files
0 web-page docker build -q -t web-page .
sha256:82aef7d4ae23db3693563bde945a6681113066d63723759a
0 web-page docker run web-page
/docker-entrypoint.sh: /docker-entrypoint.d/ is not emp
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
```



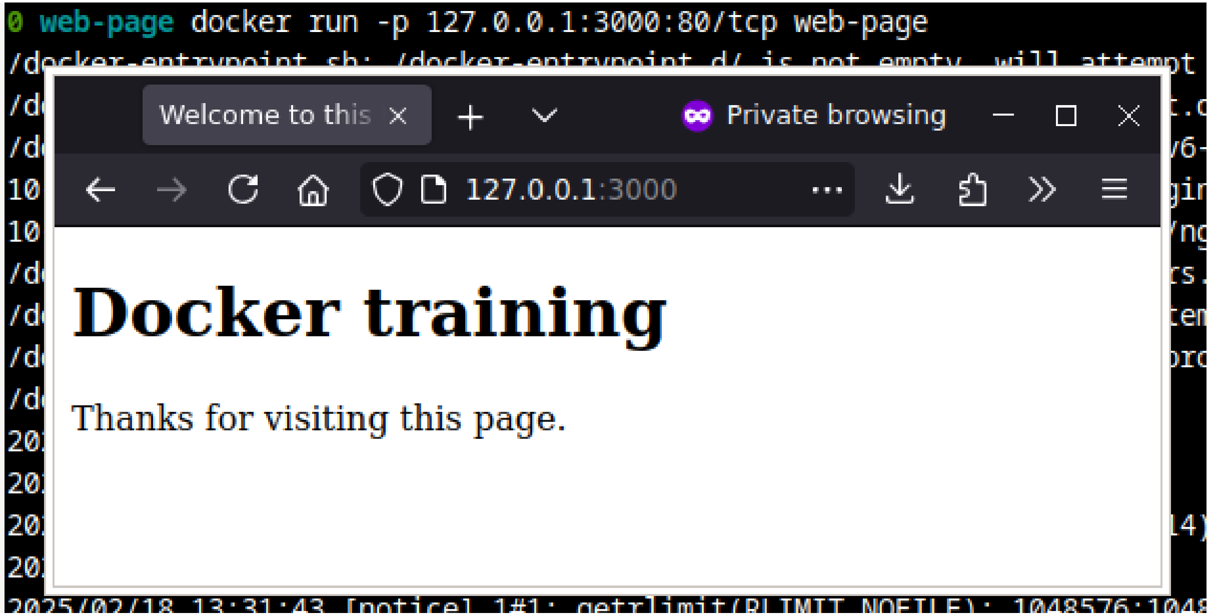
Unable to connect

An error occurred during a connection to localhost.

- The site could be temporarily unavailable or too busy. Try again in a few moments.
- If you are unable to load any pages, check your computer's network connection.
- If your computer or network is protected by a firewall or proxy, make sure that Firefox is permitted to access the web.

Network communication 2/3 [skip]

docker run -p host_ip:host_port:container_port/protocol



```
0 web-page docker run -p 127.0.0.1:3000:80/tcp web-page
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt
2025/02/18 13:31:43 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048
```

Docker training

Thanks for visiting this page.

Challenge:
Can you achieve the same result, without copying index.html into the image?

https://docs.docker.com/reference/cli/docker/container/run/#publish

Network communication 3/3 [skip]

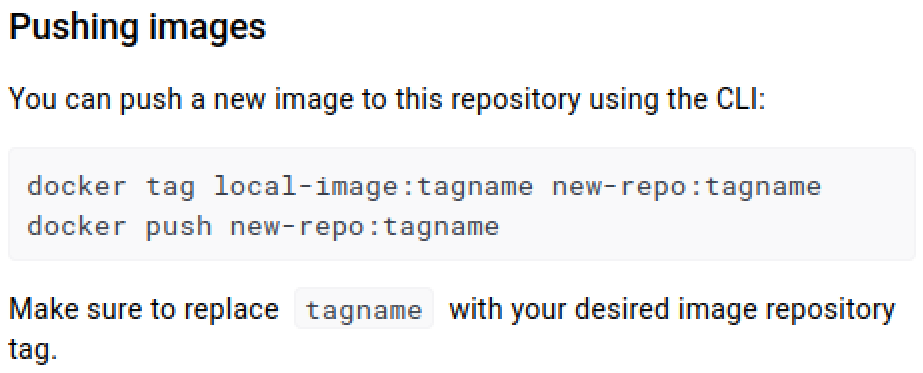
Challenge:
Can you achieve the same result, without copying index.html into the image?

Solution:
We could bind mount the folder containing the index.html file into the container.

```
docker run \
    -v $PWD/content:/usr/share/nginx/html:ro \
    -p 127.0.0.1:3000:80 \
    nginx:stable
```

Share Docker images

- Create an account on https://hub.docker.com and sign in
- Navigate to *Repositories*, create one
    - https://hub.docker.com/repository/create
- In your terminal, remember to execute `*docker login*`



Pushing images

You can push a new image to this repository using the CLI:

```
docker tag local-image:tagname new-repo:tagname
docker push new-repo:tagname
```

Make sure to replace `tagname` with your desired image repository tag.

Multi-platform images

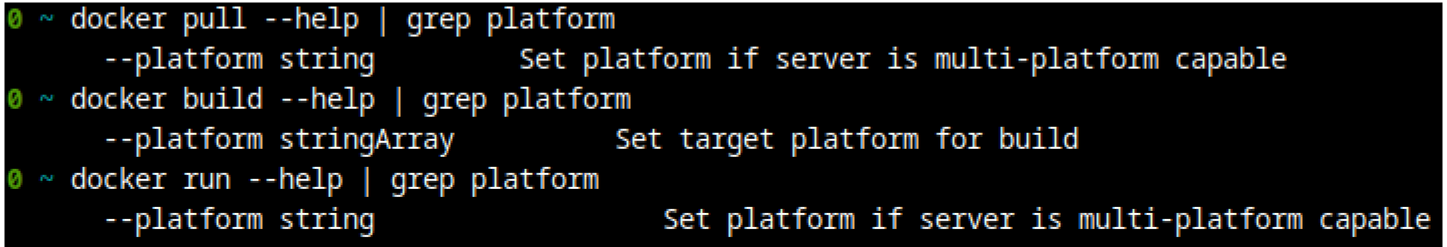
- Images can be built with support for multiple platforms
    - docker build --platform linux/amd64,linux/arm64 .
        - Provides support for the two most common architectures

https://docs.docker.com/build/building/multi-platform/

Authors might forget to build an image for your platform.

Solution: tell Docker which image to use!

```
docker run --platform linux/amd64 image-name:tag
```



```
0 ~ docker pull --help | grep platform
      --platform string         Set platform if server is multi-platform capable
0 ~ docker build --help | grep platform
      --platform stringArray          Set target platform for build
0 ~ docker run --help | grep platform
      --platform string                   Set platform if server is multi-platform capable
```

Install instructions and best practices

- How to install: https://docs.docker.com/engine/install/
- Building images: https://docs.docker.com/build/building/best-practices/
    - Very important paragraphs:
        - Choose the right base image
        - Rebuild your images often
        - Exclude with .dockerignore
        - Create ephemeral containers
        - Don't install unnecessary packages
        - Sort multi-line arguments
        - Dockerfile instructions
    - Reference: https://docs.docker.com/reference/dockerfile/
- Code samples: https://docs.docker.com/reference/samples/
- Compose: https://docs.docker.com/compose/how-tos/environment-variables/best-practices/
- WSL: https://docs.docker.com/desktop/features/wsl/best-practices/

Some best practices covered today

- docker run --rm
- Read-only bind mounts
- Separate input and output bind mounts
- Changes within `docker exec` must be reflected in the Dockerfile
- Consider using docker compose

Further support

For Docker and software engineering: pasdom@dtu.dk

You're welcome to share feedback and ask for help.

Idea: software engineering hour, e.g. once a month, for topics such as git, programming languages, conventions, best practices, etc.