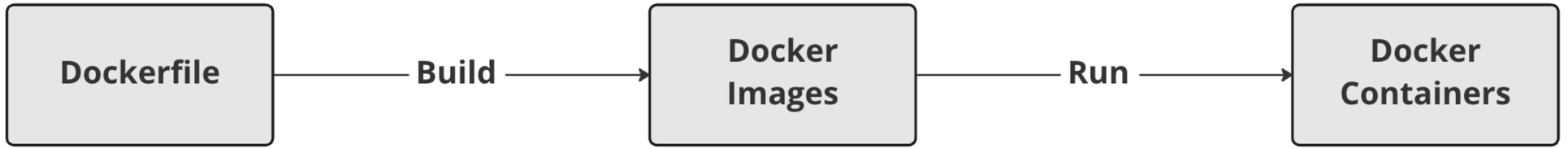
Creating your own Docker images

INTRODUCTION TO DOCKER

Tim Sangster
Software Engineer @ DataCamp

datacamp

Creating images with Dockerfiles



Text file with instructions to create an image

Blueprint to create a container

Running Image

Starting a Dockerfile

A Dockerfile always start from another image, specified using the FROM instruction.

```
FROM postgres
FROM ubuntu
FROM hello-world
FROM my-custom-data-pipeline
```

```
FROM postgres:15.0
FROM ubuntu:22.04
FROM hello-world:latest
FROM my-custom-data-pipeline:v1
```

Building a Dockerfile

Building a Dockerfile creates an image.

```
docker build /location/to/Dockerfile
docker build .
```

```
[+] Building 0.1s (5/5) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 54B
...
 => CACHED [1/1] FROM docker.io/library/ubuntu
 => exporting to image
 => => exporting layers
 => => writing image sha256:a67f41b1d127160a7647b6709b3789b1e954710d96df39ccaa21..
```

Naming our image

In practice we almost always give our images a name using the -t flag:

```
docker build -t first_image .
```

```
...
=> => writing image sha256:a67f41b1d127160a7647b6709b3789b1e954710d96df39ccaa21..
=> => naming to docker.io/library/first_image
```

```
docker build -t first_image:v0 .
```

```
=> => writing image sha256:a67f41b1d127160a7647b6709b3789b1e954710d96df39ccaa21..
=> => naming to docker.io/library/first_image:v0
```

Customizing images

```
RUN <valid-shell-command>
```

```
FROM ubuntu
RUN apt-get update
RUN apt-get install -y python3
```

Use the -y flag to avoid any prompts:

```
...
After this operation, 22.8 MB of additional disk space will be used.
Do you want to continue? [Y/n]
```

Building a non-trivial Dockerfile

When building an image Docker actually runs commands after RUN

Docker running `RUN apt-get update` takes the same amount of time as us running it!

```
root@host:/# apt-get update
Get:1 http://ports.ubuntu.com/ubuntu-ports jammy InRelease [270 kB]
...
Get:17 http://ports.ubuntu.com/ubuntu-ports jammy-security/restricted arm64 Pack..
Fetched 23.0 MB in 2s (12.3 MB/s)
Reading package lists... Done
```

Summary

Usage	Dockerfile Instruction
Start a Dockerfile from an image	FROM <image-name>
Add a shell command to image	RUN <valid-shell-command>
Make sure no user input is needed for the shell-command.	RUN apt-get install **-y** python3

Usage	Shell Command
Build image from Dockerfile	docker build /location/to/Dockerfile
Build image in current working directory	docker build **.**
Choose a name when building an image	docker build -t first_image .

Let's practice!

INTRODUCTION TO DOCKER

Managing files in your image

INTRODUCTION TO DOCKER

Tim Sangster
Software Engineer @ DataCamp

datacamp

COPYing files into an image

The COPY instruction copies files from our local machine into the image we're building:

```
COPY <src-path-on-host> <dest-path-on-image>
COPY /projects/pipeline_v3/pipeline.py /app/pipeline.py
```

```
docker build -t pipeline:v3 .
...
[4/4] COPY ./projects/pipeline_v3/pipeline.py /app/pipeline.py
```

If the destination path does not have a filename, the original filename is used:

```
COPY /projects/pipeline_v3/pipeline.py /app/
```

COPYing folders

Not specifying a filename in the src-path will copy all the file contents.

```
COPY <src-folder> <dest-folder>
COPY /projects/pipeline_v3/ /app/
```

`COPY /projects/pipeline_v3/ /app/` will copy everything under `pipeline_v3/`:

```
/projects/
    pipeline_v3/
        pipeline.py
        requirements.txt
        tests/
            test_pipeline.py
```

Copy files from a parent directory

```
/init.py
/projects/
    Dockerfile
    pipeline_v3/
        pipeline.py
```

If our current working directory is in the `projects/` folder.

We can't copy `init.py` into an image.

```
docker build -t pipeline:v3 .
 => ERROR [4/4] COPY ../init.py /     0.0s
failed to compute cache key: "../init.py" not found: not found
```

Downloading files

Instead of copying files from a local directory, files are often downloaded in the image build:

- Download a file

```
RUN curl <file-url> -o <destination>
```

- Unzip the file

```
RUN unzip <dest-folder>/<filename>.zip
```

- Remove the original zip file

```
RUN rm <copy_directory>/<filename>.zip
```

Downloading files efficiently

- Each instruction that downloads files adds to the total size of the image.
- Even if the files are later deleted.
- The solution is to download, unpack and remove files in a single instruction.

```
RUN curl <file_download_url> -o <destination_directory>/<filename>.zip \
&& unzip <destination_directory>/<filename>.zip -d <unzipped-directory> \
&& rm <destination_directory>/<filename>.zip
```

Summary

Usage	Dockerfile Instruction
Copy files from host to the image	COPY <src-path-on-host> <dest-path-on-image>
Copy a folder from host to the image	COPY <src-folder> <dest-folder>
We can't copy from a parent directory where we build a Dockerfile	~~COPY ../<file-in-parent-directory> /~~

Keep images small by downloading, unzipping, and cleaning up in a single RUN instruction:

```
RUN curl <file_download_url> -O <destination_directory> \
&& unzip <destination_directory>/<filename>.zip -d <unzipped-directory> \
&& rm <destination_directory>/<filename>.zip
```

Let's practice!

INTRODUCTION TO DOCKER

Choosing a start command for your Docker image

INTRODUCTION TO DOCKER

Tim Sangster
Software Engineer @ DataCamp

datacamp

What is a start command?

The hello-world image prints text and then stops.

```
docker run hello-world
```

```
Hello from Docker!

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon created a new container from the hello-world image which runs
    executable that produces the output you are currently reading.
 3. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.
```

What is a start command?

An image with python could start python on startup.

```
docker run python3-sandbox
```

```
Python 3.10.6 (main, Nov  2 2022, 18:53:38) [GCC 11.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
...
```

```
....
>>> exit()
repl@host:/#
```

Running a shell command at startup

```
CMD <shell-command>
```

The CMD instruction:

- Runs when the image is started.
- Does not increase the size of the image .
- Does not add any time to the build.
- If multiple exist, only the last will have an effect.

Typical usage

Starting an application to run a workflow or that accepts outside connections.

```
CMD python3 my_pipeline.py
```

```
CMD postgres
```

Starting a script that, in turn, starts multiple applications

```
CMD start.sh
```

```
CMD python3 start_pipeline.py
```

When will it stop?

- hello-world image -> After printing text
- A database image -> When the database exits

A more general image needs a more general start command.

- An Ubuntu image -> When the shell is closed

Overriding the default start command

Starting an image

```
docker run <image>
```

Starting an image with a custom start command

```
docker run <image> <shell-command>
```

Starting an image interactively with a custom start command

```
docker run -it <image> <shell-command>
```

```
docker run -it ubuntu bash
```

Summary

Usage	Dockerfile Instruction
Add a shell command run when a container is started from the image.	CMD <shell-command>

Usage	Shell Command
Override the CMD set in the image	docker run <image> <shell-command>
Override the CMD set in the image and run interactively	docker run -it <image> <shell-command>

Let's practice!

INTRODUCTION TO DOCKER

Introduction to Docker layers and caching

INTRODUCTION TO DOCKER

Tim Sangster
Software Engineer @ DataCamp

datacamp

Docker build

Downloading and unzipping a file using the Docker instructions.

```
RUN curl http://example.com/example_folder.zip
RUN unzip example_folder.zip
```

Will change the file system and add:

```
/example_folder.zip
/example_folder/
    example_file1
    example_file2
```

It is these changes that are stored in the image.

Docker instructions are linked to File system changes

Each instruction in the Dockerfile is linked to the changes it made in the image file system.

```
FROM docker.io/library/ubuntu
 => Gives us a file system to start from with all files needed to run Ubuntu
```

```
COPY /pipeline/ /pipeline/
 => Creates the /pipeline/ folder
 => Copies multiple files in the /pipeline/ folder
```

```
RUN apt-get install -y python3
 => Add python3 to /var/lib/
```

Docker layers

- Docker layer: All changes caused by a single Dockerfile instruction.
- Docker image: All layers created during a build

--> Docker image: All changes to the file system by all Dockerfile instructions.

While building a Dockerfile, Docker tells us which layer it is working on:

```
=> [1/3] FROM docker.io/library/ubuntu
=> [2/3] RUN apt-get update
=> [3/3] RUN apt-get install -y python3
```

Docker caching

Consecutive builds are much faster because Docker re-uses layers that haven't changed.

Re-running a build:

```
=> [1/3] FROM docker.io/library/ubuntu
=> CACHED [2/3] RUN apt-get update
=> CACHED [3/3] RUN apt-get install -y python3
```

Re-running a build but with changes:

```
=> [1/3] FROM docker.io/library/ubuntu
=> CACHED [2/3] RUN apt-get update
=> [3/3] RUN apt-get install -y R
```

Understanding Docker caching

When layers are cached helps us understand why sometimes images don't change after a rebuild.

- Docker can't know when a new version of python3 is released.
- Docker will use cached layers because the instructions are identical to previous builds.

```
=> [1/3] FROM docker.io/library/ubuntu
=> CACHED [2/3] RUN apt-get update
=> CACHED [3/3] RUN apt-get install -y python3
```

Understanding Docker caching

Helps us write Dockerfiles that build faster because not all layers need to be rebuilt.

In the following Dockerfile all instructions need to be rebuild if the pipeline.py file is changed:

```
FROM ubuntu
COPY /app/pipeline.py /app/pipeline.py
RUN apt-get update
RUN apt-get install -y python3
```

```
 => [1/4] FROM docker.io/library/ubuntu
 => [2/4] COPY /app/pipeline.py /app/pipeline.py
 => [3/4] RUN apt-get update
 => [4/4] RUN apt-get install -y python3
```

Understanding Docker caching

Helps us write Dockerfiles that build faster because not all layers need to be rebuilt.

In the following Dockerfile, only the COPY instruction will need to be re-run.

```
FROM ubuntu
RUN apt-get update
RUN apt-get install -y python3
COPY /app/pipeline.py /app/pipeline.py
```

```
 => [1/4] FROM docker.io/library/ubuntu
 => CACHED [2/4] RUN apt-get update
 => CACHED [3/4] RUN apt-get install -y python3
 => [4/4] COPY /app/pipeline.py /app/pipeline.py
```

Let's practice!

INTRODUCTION TO DOCKER