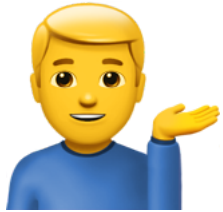
Docker: The Efficient

Introduction



Who am I?

- My name is Jannis Seemann
- I'm German 🇩🇪, with a degree in computer science (TU München)
- **Important work experience:**
  - With 18, first internship at Google in London 🇬🇧
  - At age 20, another one in Mountain View, California 🇺🇸
  - Nowadays, I'm a freelance web developer (and server admin)
  - And one of the most popular instructors on the German speaking market on Udemy (300.000+ students)
- **In this course:**
  - You benefit from my work experience
  - ... and from my online teaching experience

Day 1

- Understand container lifecycles
- Apply essential CLI commands for containers and images
- Build custom images using `Dockerfiles`

Day 2

- Manage storage with volumes and bind mounts
- Connect and isolate containers through custom networks
- Orchestrate multi-container apps with Docker Compose

Day 2.5

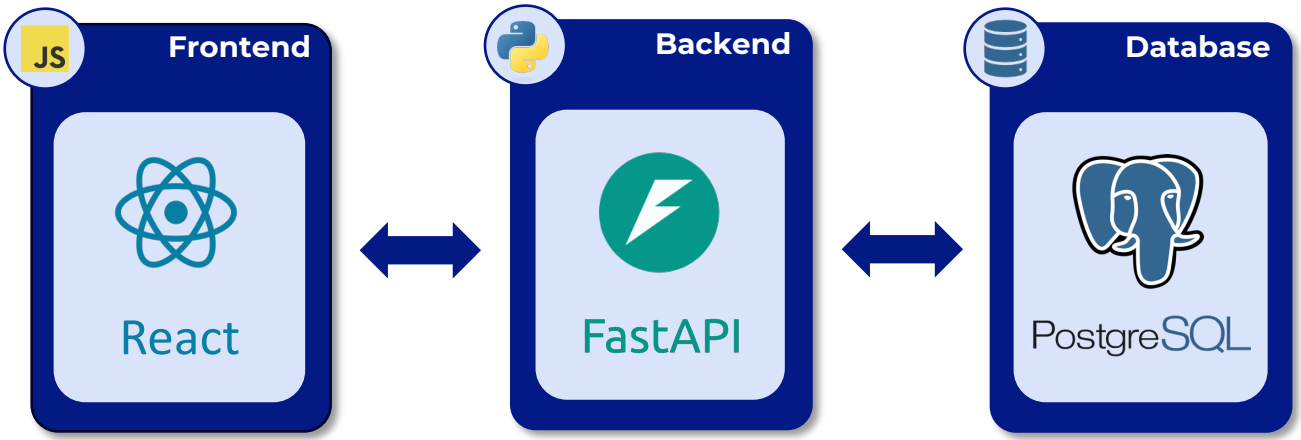
- Set up and manage private docker registries
- Automate everything: From GitHub to automatic deployment
- Create a fully-fledged deployment- & dev setup

Thank you

- Thank you for subscribing / purchasing this course
- This really means a lot to us...
- ... as this allows us to dedicate so much time to our courses

Docker: The Efficient

See what Docker does!


JS
Frontend
React
Backend
FastAPI
Database
PostgreSQL

Which pain points does Docker solve?

Simplified setup & portability

No manual installations or compatibility issues

Share and run the app anywhere with just a `compose.yml` file without extra setup

Cross-platform consistency

Works seamlessly on Windows, macOS, and Linux

Guarantees consistent dependencies and avoids version mismatches

Scalability & reproducibility

Easily scale services by simple configuration changes

Ensures identical environments across development, testing, and production

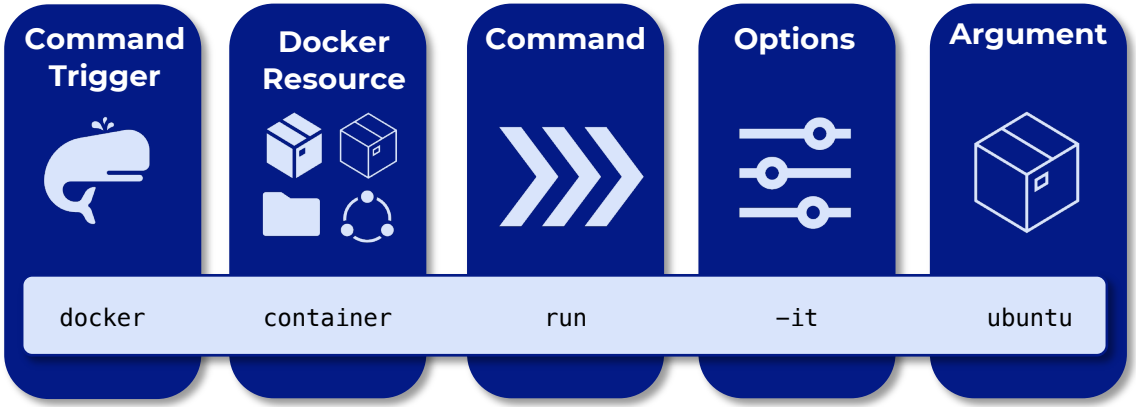
Docker replaces manual setups with a single command that works consistently everywhere!

Docker: The Efficient Guide

Run your first Docker containers!

Jannis Seemann

The structure of a Docker command



Challenge

Run a `node` container and verify it by executing the following commands:

```
console.log(0.1+0.2===0.3);
```

```
console.log([...Array(5).keys()]);
```

```
console.log(typeof NaN);
```

Docker: The Efficient

What is a container?

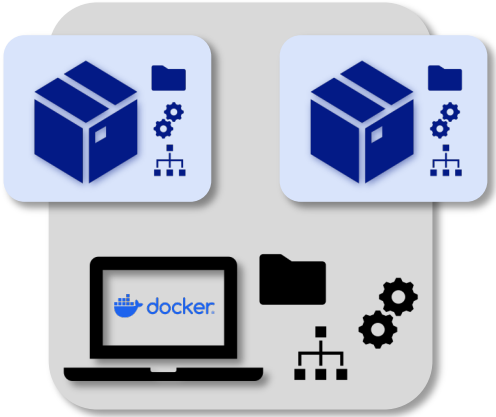
What is a Container?

Efficient runtime environment

- Provides everything an application needs to run
- Uses fewer resources than virtual machines

Isolated and self-sufficient

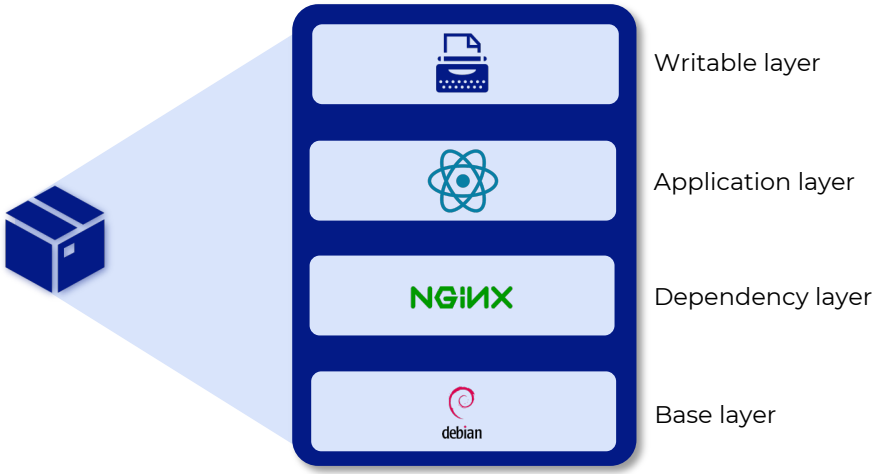
- Isolated from the rest of your system and other containers
- Runs independently, with its own filesystem, processes, and network stack

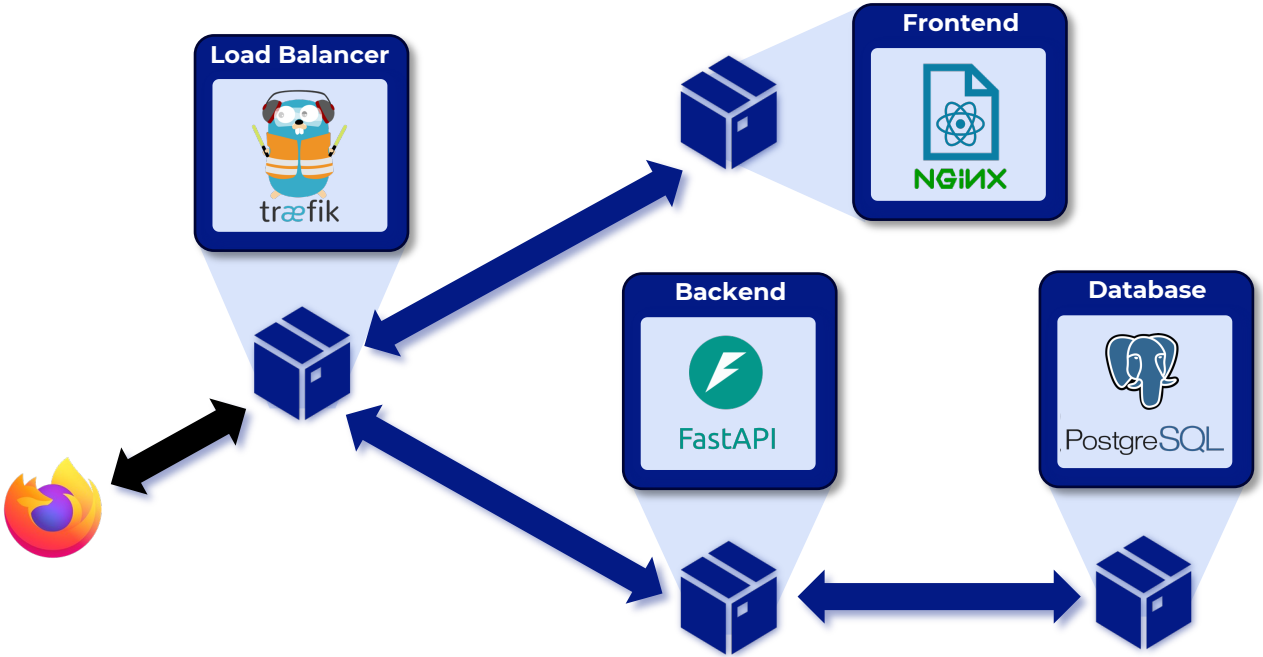


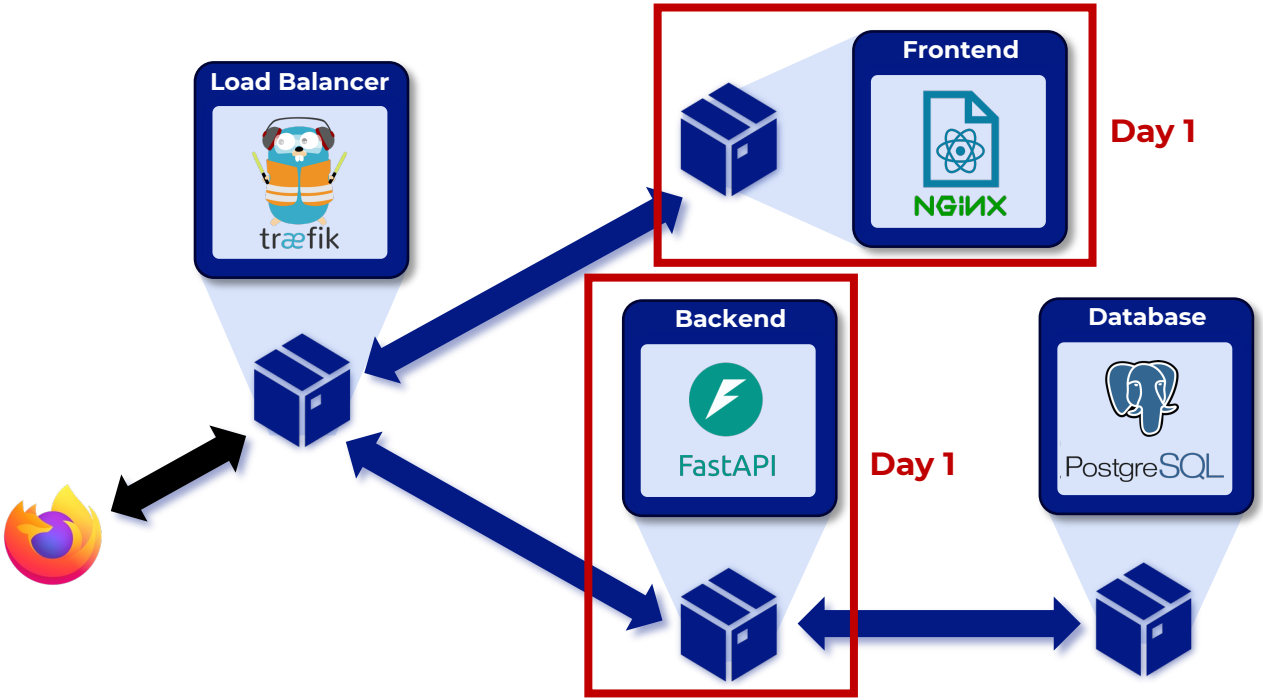
The structure of a container

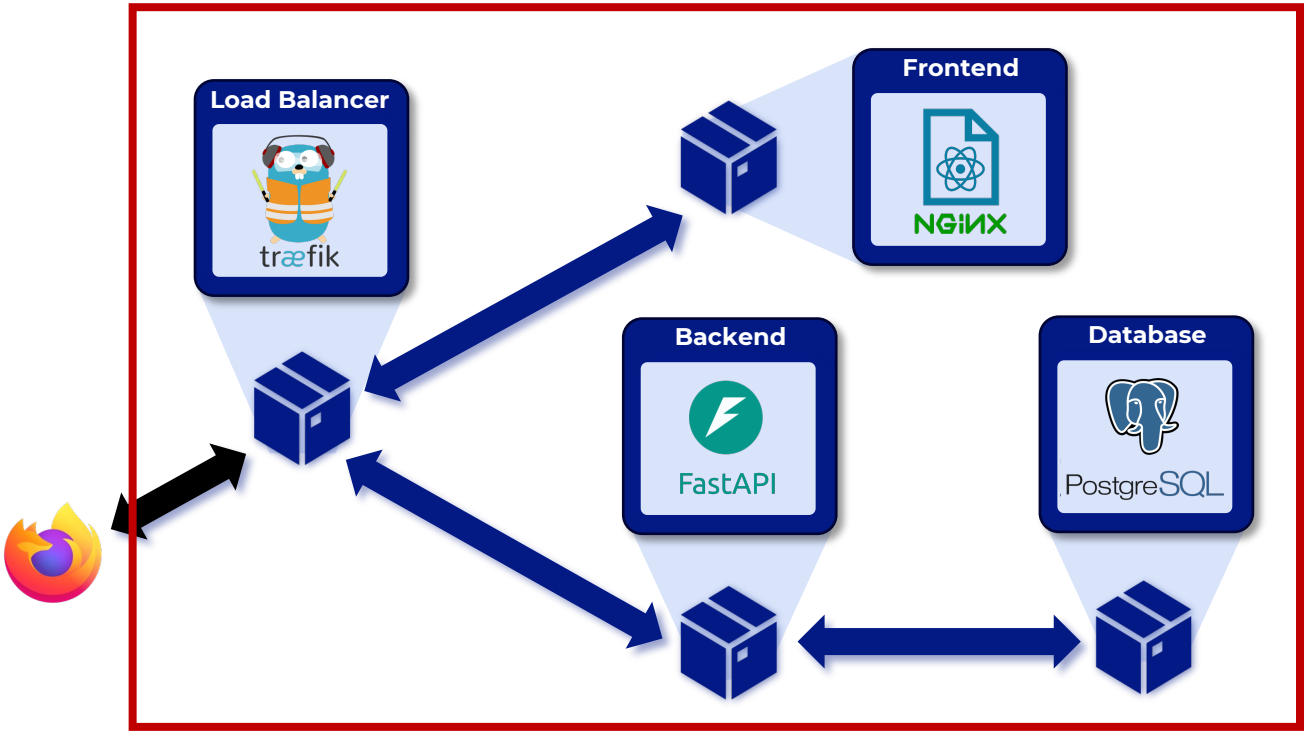
A container packages the entire code base and all dependencies of an application

Enables it to run consistently across different systems




Load Balancer
træfik
Frontend
NGINX
Backend
FastAPI
Database
PostgreSQL


Load Balancer
træfik
Frontend
NGINX
Day 1
Backend
FastAPI
Day 1
Database
PostgreSQL

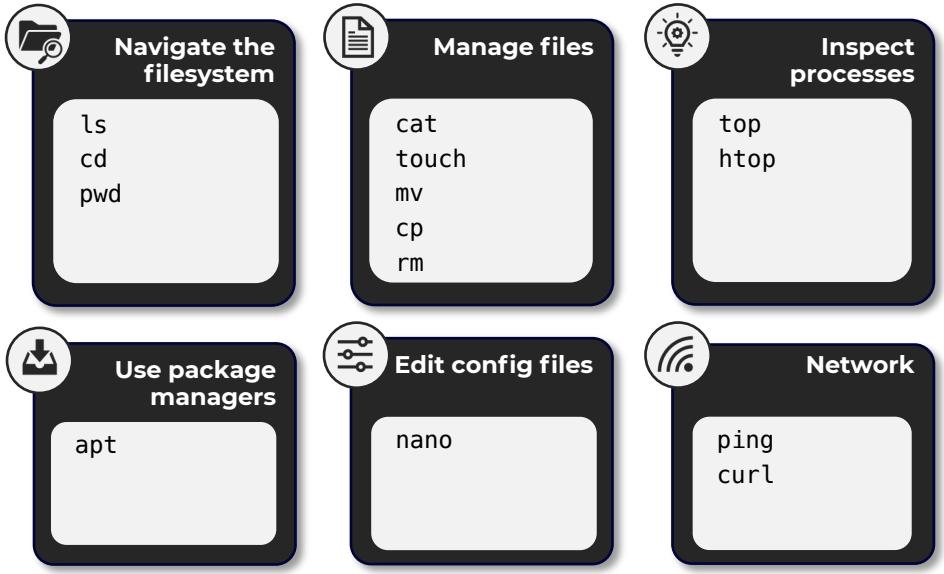

Load Balancer
træfik
Frontend
NGINX
Backend
FastAPI
Database
PostgreSQL
Day 2

Docker: The Efficient

Crash course: Linux shell

Jannis Seemann

What do you have to know?



Docker: The Efficient

How to install Docker?

Open-source

Docker Daemon:
Creates and runs containers

Docker CLI:
Provides the docker terminal command

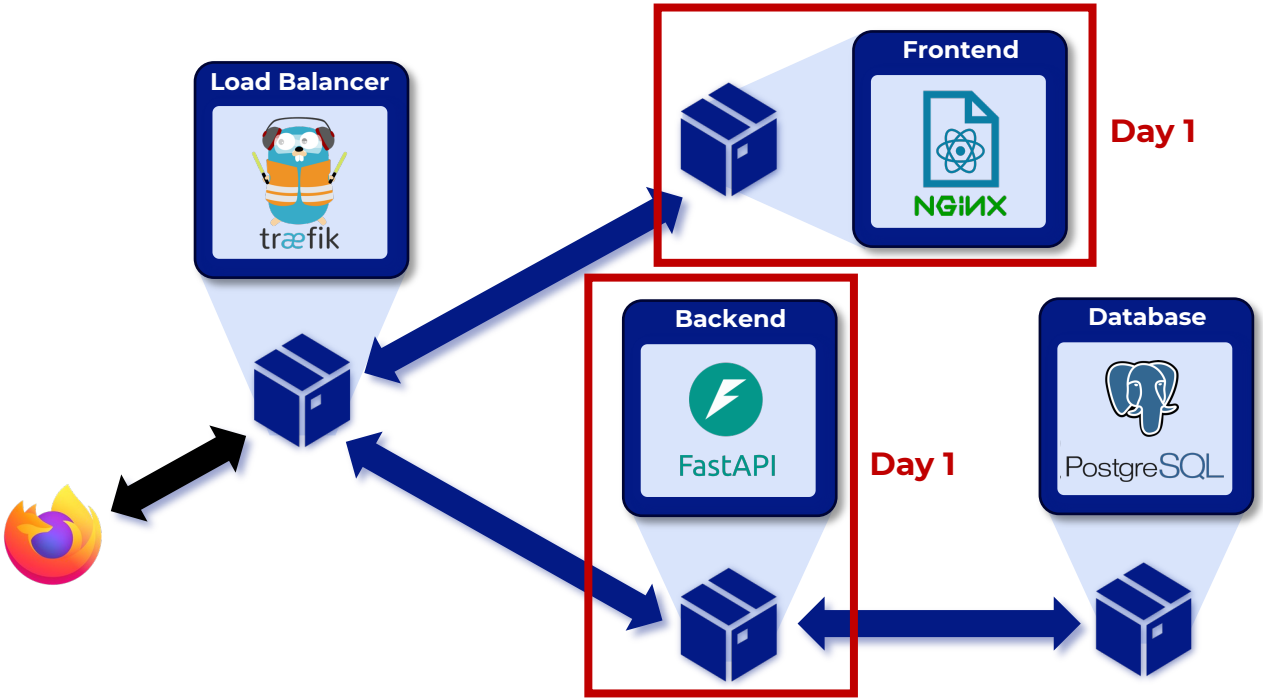
Docker Inc.

Docker Inc. leads the development of the open-source docker components

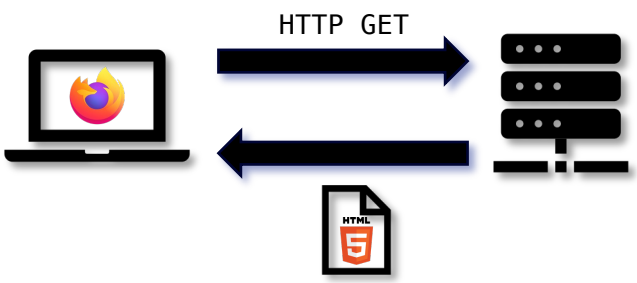
And offers an easy-to-use GUI (proprietary):
Docker Desktop

Docker: The Efficient

Making containers accessible (networking)


Load Balancer
træfik
Frontend
NGINX
Day 1
Backend
FastAPI
Day 1
Database
PostgreSQL

What are web servers?



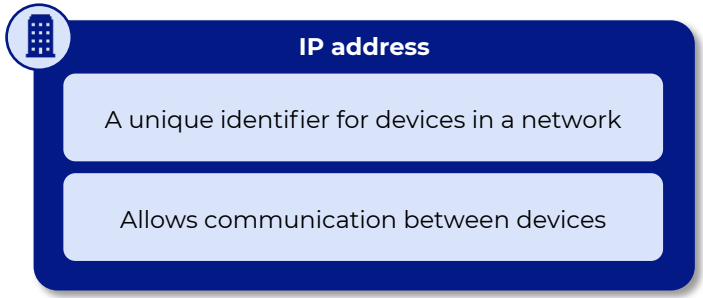
NGINX — Market share: ~34%

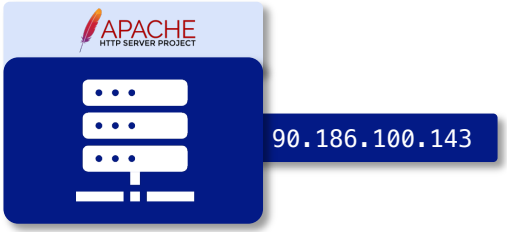
Lightweight, high-performance server, ideal for static content and load balancing

APACHE HTTP SERVER PROJECT — Market share: ~27%

Flexible server for dynamic content, ideal for hosting setups and legacy applications

IP addresses & port numbers





IP addresses & port numbers

IP address

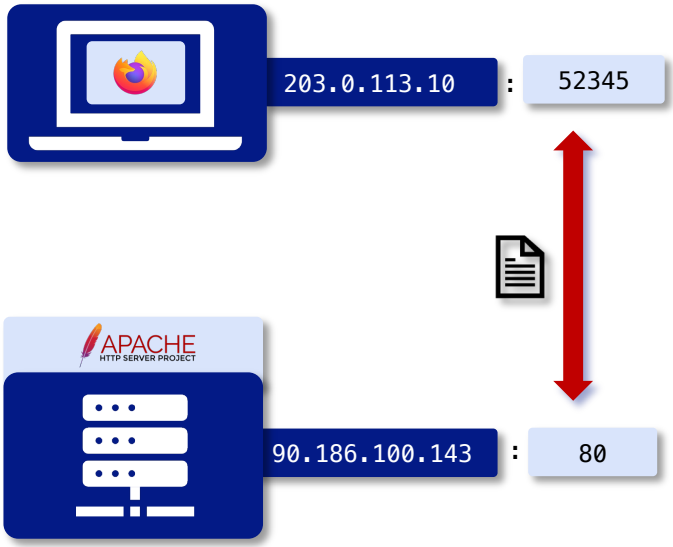
A unique identifier for devices in a network

Allows communication between devices

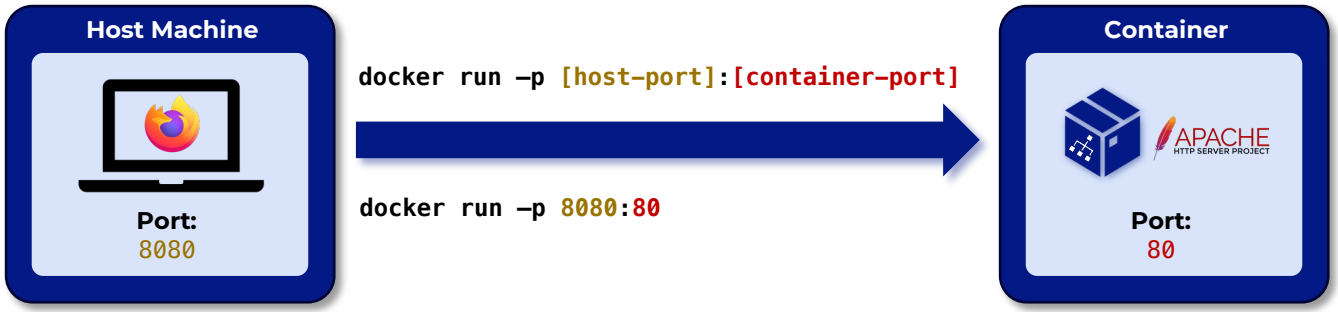
Port number

Logical endpoint on a device

Ensures data is delivered to the correct application or service on that device



Port forwarding: –p



Docker: The Efficient

Running containers in the background: Interactive vs. detached

Interactive mode (`-it`)

Runs the container in the foreground and attaches a terminal (`-t`)

Allows direct interaction with the container (`-i`)

Detached mode (`-d`)

Runs the container in the background

This allows us to continue using our terminal

Access containers (`docker exec`)

We can use `docker exec` to execute commands

```
docker [container]
exec -it
<container>
<command>
```

Challenge

Change the default HTML file in an NGINX container

Run an `nginx` container in the background, map its port to your local port `8000`

Access the running container and start a `bash` shell

Edit the default HTML file `index.html` in `/usr/share/nginx/html` and change `<html>` to `<html style="background: plum">`

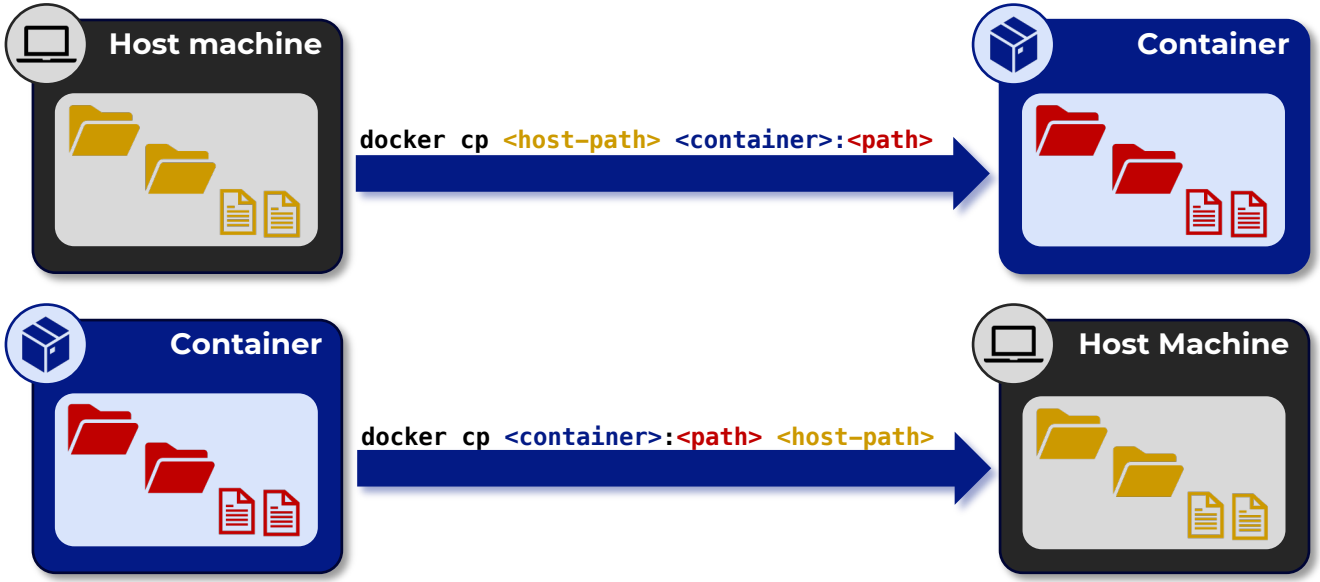
Verify the change in your browser – what has changed?

Make sure the port is not already in use; if it is, either free it or choose a different one.

You may want to install a text editor like `nano` (through `apt`)

Docker: The Efficient

Copying data into / from containers


Host machine
docker cp <host-path> <container>:<path>
Container
Container
docker cp <container>:<path> <host-path>
Host Machine

Docker: The Efficient

Challenge: Copying data into a container

Challenge

The goal: Deploy the expense tracker frontend with nginx

Start an `nginx` container

Copy the `/usr/share/nginx/html/index.html` file from the container to a local directory (as a backup)

Replace the contents of the `/usr/share/nginx/html` directory with the contents of the `dist` directory

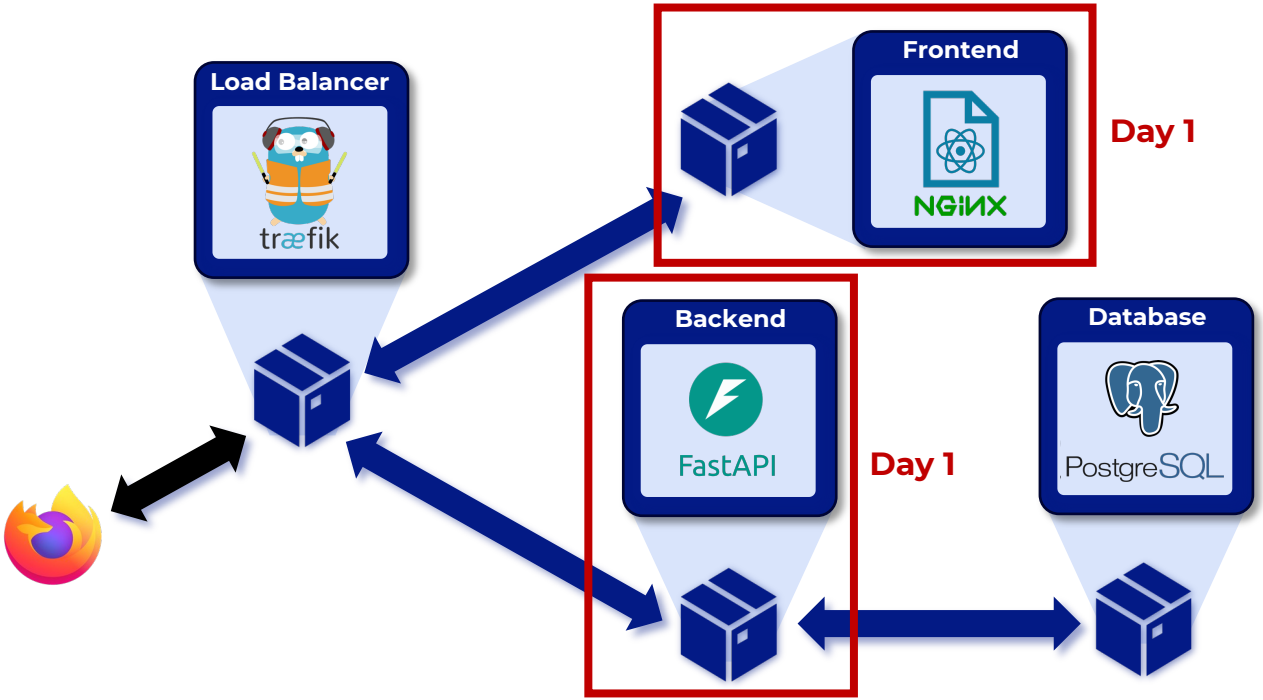
Verify the change in your browser

You can also specify a name, running mode, and port

The `dist` directory contains the production build of our react app. You can find it in the next lecture.

Docker: The Efficient

Overview:
What are images?


Load Balancer
træfik
Frontend
NGINX
Day 1
Backend
FastAPI
Day 1
Database
PostgreSQL

Docker: The Efficient

What are images?

What is an Image?

Blueprint for containers

Provides everything needed to run an application

Includes code, runtime, libraries, environment variables, and other configurations

Immutable and read-only

The image itself cannot be changed

Changes occur only in the container's writable layer

Layered architecture

Images consist out of multiple layers to optimize storage

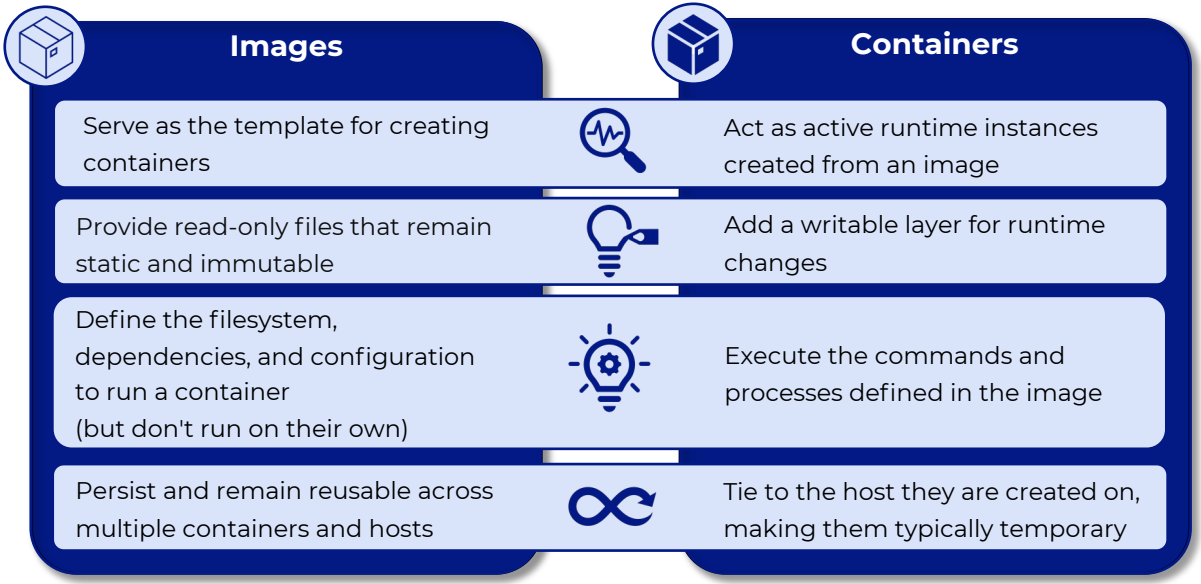
Layers are cached for faster builds and efficient reuse

Containers are the running instances created from images

Docker: The Efficient

Containers vs. images

Jannis Seemann


Images
Containers
Serve as the template for creating containers
Act as active runtime instances created from an image
Provide read-only files that remain static and immutable
Add a writable layer for runtime changes
Define the filesystem, dependencies, and configuration to run a container (but don't run on their own)
Execute the commands and processes defined in the image
Persist and remain reusable across multiple containers and hosts
Tie to the host they are created on, making them typically temporary

Docker: The Efficient

Finding images on Docker Hub

Jannis Seemann

What is Docker Hub?

Centralized image registry

A platform to store, manage, and share Docker images

Hosts both public and private repositories

Collaboration and discovery

Enables teams to share images easily

Provides access to a vast library of official and community-contributed images

Integration with docker tools

Seamlessly integrates with the Docker CLI for pulling, pushing, and managing images

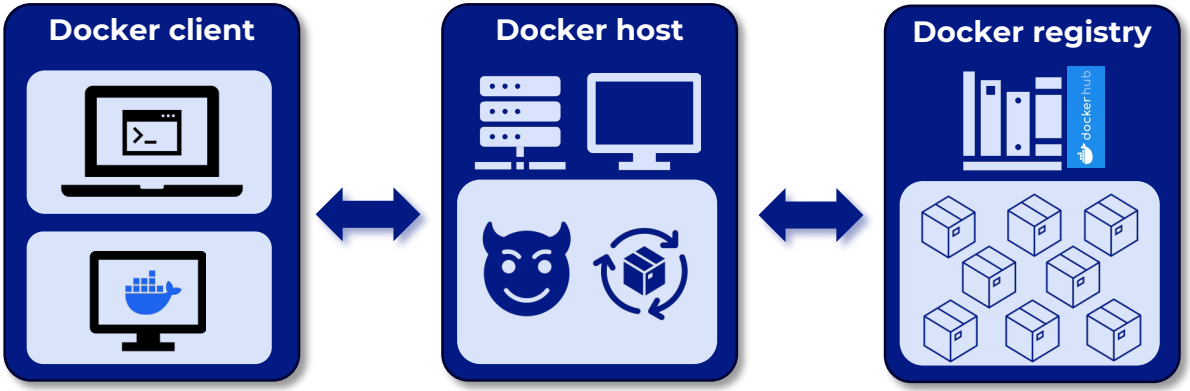
Acts as the default registry for Docker

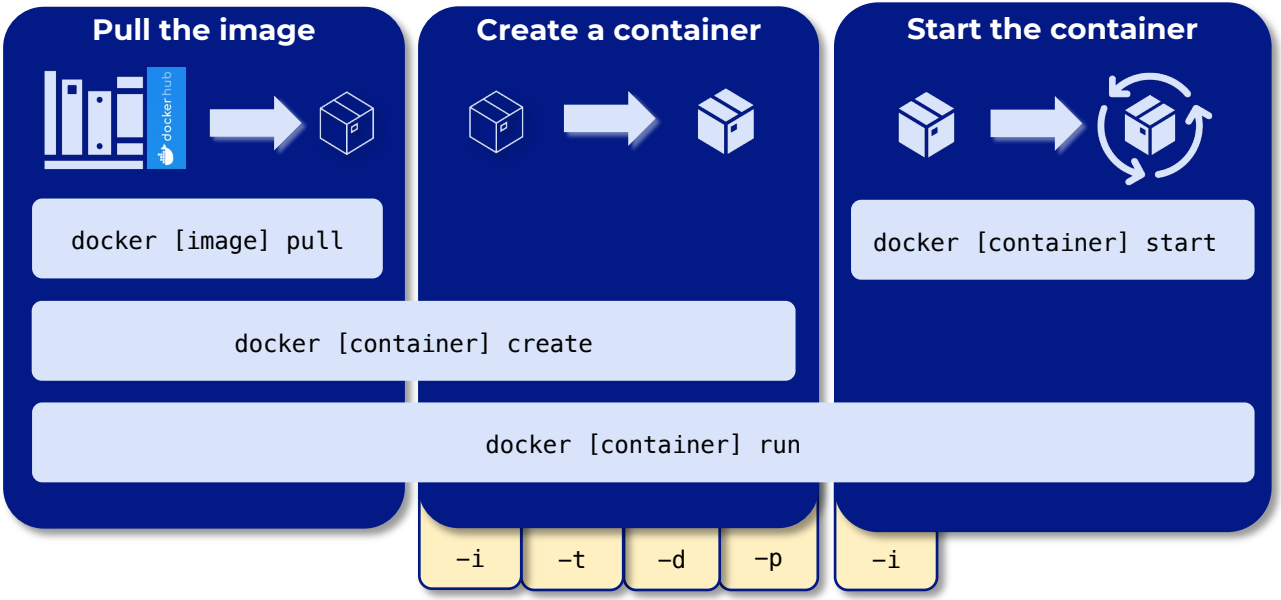
Docker Hub ensures consistency and accessibility for deployments

Docker: The Efficient

Dive deep into the container creation process!

Jannis Seemann


Docker client
Docker host
Docker registry
docker hub


Pull the image
docker hub
docker [image] pull
Create a container
Start the container
docker [container] start
docker [container] create
docker [container] run
-i
-t
-d
-p
-i

Docker: The Efficient

Overview: Dockerfiles

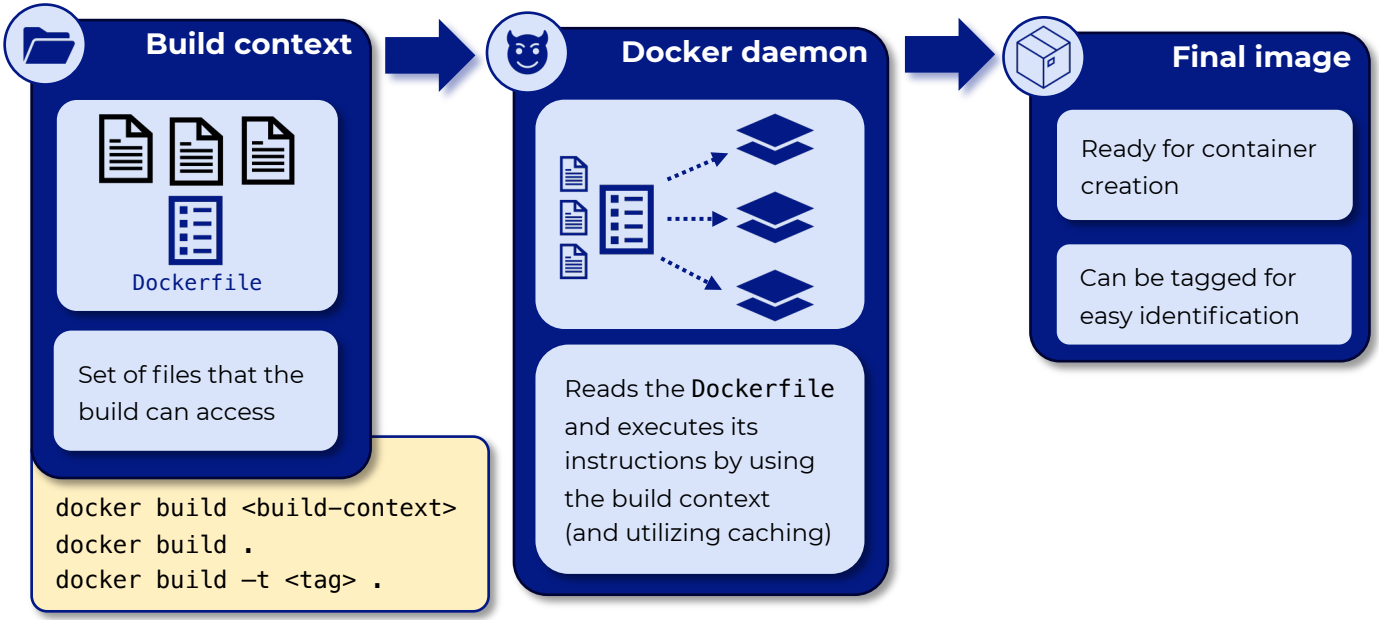
Overview: Dockerfiles

- **In this chapter:**
  - We'll explore Dockerfiles
  - They allow you to create your own images
  - ... which we can then use to execute our own application
- **The structure:**
  - **In the videos:**
    - We'll create a Dockerfile for the backend
  - **After that:**
    - You will create a first Dockerfile for the frontend (by using pre-compiled files)
    - And we'll enhance this frontend together (and enabling compiling of .js files)
- **We recommend:**
  - Using a text editor like Visual Studio Code:
    - https://code.visualstudio.com/
  - This will make it way easier to follow along

Docker: The Efficient

Build your first own image with a Dockerfile

Jannis Seemann


Build context
Dockerfile
Set of files that the build can access
docker build <build-context>
docker build .
docker build –t <tag> .
Docker daemon
Reads the Dockerfile and executes its instructions by using the build context (and utilizing caching)
Final image
Ready for container creation
Can be tagged for easy identification

The FROM Instruction

Definition

Specifies the base image for your custom Docker image

```
FROM <image>[:<tag>]
```

```
FROM python:3.12
```

Important notes

Defines the operating system or runtime environment for your app

Every `Dockerfile` must start with a `FROM` instruction

For now, a Dockerfile only uses a single `FROM` instruction

Best practices

Use official images from trusted sources

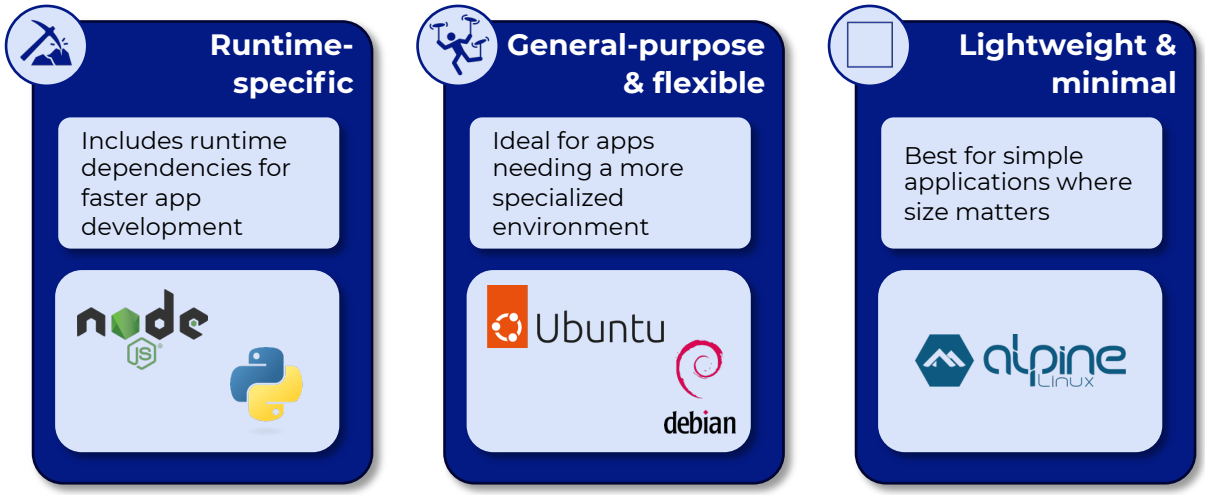
Choose lightweight base images for smaller builds when possible

Specify a tag to avoid unexpected changes

Docker: The Efficient

Choosing a suitable base image

How to choose a good base image?



Beware of Alpine

Advantages

A lightweight Linux distribution (~5 MB), that reduces image size

Minimal dependencies for faster builds

Limitations

Musl C Library

Smaller but not fully compatible with all software

Slower for apps with complex I/O or memory workloads (e.g., Python)

APK Package Manager

Limited Ecosystem

Less optimized packages

Best Practices

Ideal for lightweight apps like microservices or static binaries

Test thoroughly for third-party library compatibility

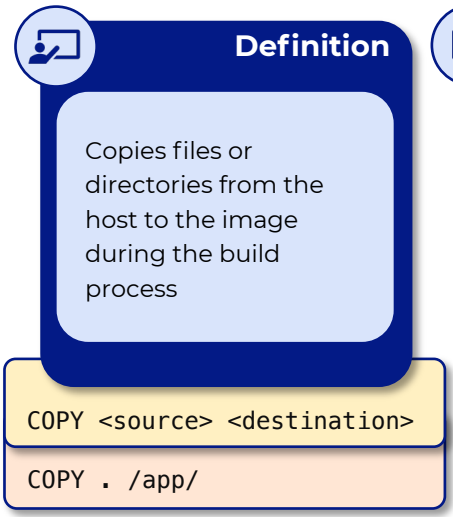
Consider `debian:slim` for better compatibility

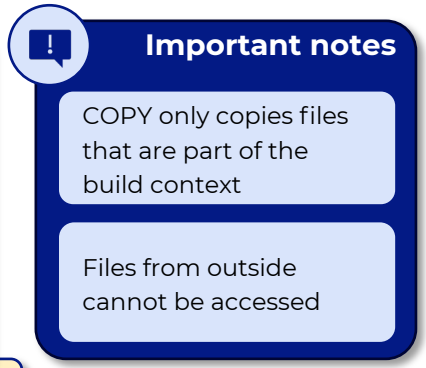
Docker: The Efficient

The COPY & WORDKIR instructions

Jannis Seemann

The COPY instruction





The `WORKDIR` instruction

Definition

Sets the working directory within the container's filesystem for subsequent instructions

```
WORKDIR <path>
```

```
WORKDIR /app
COPY . .
```

Important notes

`WORKDIR` persists in the container (and will be the default directory)

Automatically creates the directory if it doesn't exist

Best practices

Use `WORKDIR` to simplify commands and avoid repetitive paths

Use logical and consistent directories to enhance readability

Docker: The Efficient

The RUN instruction

The RUN Instruction

Definition

Executes a command during the image build process to install software, configure the environment, or prepare the image

Important notes

Creates a new layer for each `RUN` instruction

Layers are cached to speed up future builds if unchanged

Best practices

Combine multiple commands into a single `RUN` instruction to minimize layers

Try not to create files that are not needed – for example, by disabling caching

```
RUN ["executable", "param1", "param2"]
```

Exec format

```
RUN executable param1 param2
```

Shell format

Runs the command via `/bin/sh -c`, enabling shell features like pipes, environment variable expansion,...

Docker: The Efficient

The CMD instruction

The CMD instruction

Definition

Specifies the default command that is executed when the container launches

Important notes

Later `CMD` instructions overwrite earlier ones

The `CMD` is overridden if a command is specified during `docker run`

Best practices

Use the exec format whenever possible (it's not always)

```
CMD ["executable", "param1", "param2"]
```

Exec format

```
CMD executable param1 param2
```

Shell format

Runs the command via `/bin/sh -c`, enabling shell features like pipes, environment variable expansion,...

Docker: The Efficient

The EXP0SE Instruction

Jannis Seemann

The EXPOSE instruction

Definition

Declares the port(s) that the container will listen on at runtime

```
EXPOSE <port>[/protocol]
```

```
EXPOSE 80
EXPOSE 80/tcp
```

Important Notes

Does **not publish the port** to the host automatically

To makaccessible on the host, use the `-p` or `--publish` flag with `docker run`e the port

Best Practices

Use `EXPOSE` to document the container's intended ports for clarity and maintainability

Docker: The Efficient

Multi-stage builds

Jannis Seemann

Problems with single-stage builds

Large image sizes

Build tools like `npm` or compilers remain in the final image

Unnecessary dependencies are deployed

Security risks

Build tools and extra layers expand the attack surface

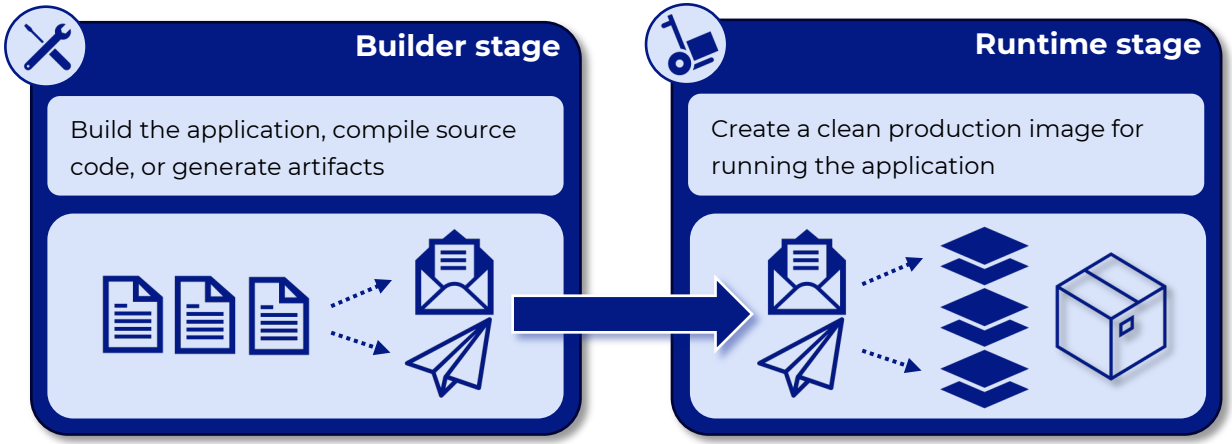
Additional files may introduce vulnerabilities

Inefficient builds

Minor source changes invalidate the cache, forcing full rebuilds

Managing separate `Dockerfiles` or scripts for building and serving increases complexity unnecessarily

Multi-stage builds



Docker: The Efficient

The ARG & ENV instructions

Jannis Seemann

The ARG instruction

Definition

Defines build-time variables for the image build process

Important notes

Available during the build process only (not at runtime)

Pass values using the `--build-arg` flag

Best practices

Use `ARG` for build-specific configurations

Avoid using `ARG` for sensitive data

```
ARG <name>[=<default_value>]
```

The ENV instruction

Definition

Sets environment variables that persist across all subsequent instructions and into the running container

Important notes

Unlike `ARG`, `ENV` variables remain available at runtime after the container starts

Best practices

Use for runtime configuration

Avoid storing sensitive data directly in `ENV`

```
ENV <key>=<value>
```

Docker: The Efficient

The LABEL instruction

The LABEL instruction

Definition

Adds metadata to an image (key-value pairs)

```
LABEL <key>=<value>
```

```
LABEL version="1.0.0"
```

Important notes

An image can have more than one label

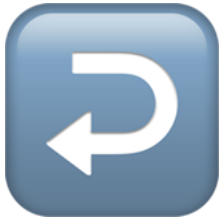
Labels included in base images are inherited by derived images

Enables searching and filtering images using `--filter label=<key>`

Best practices

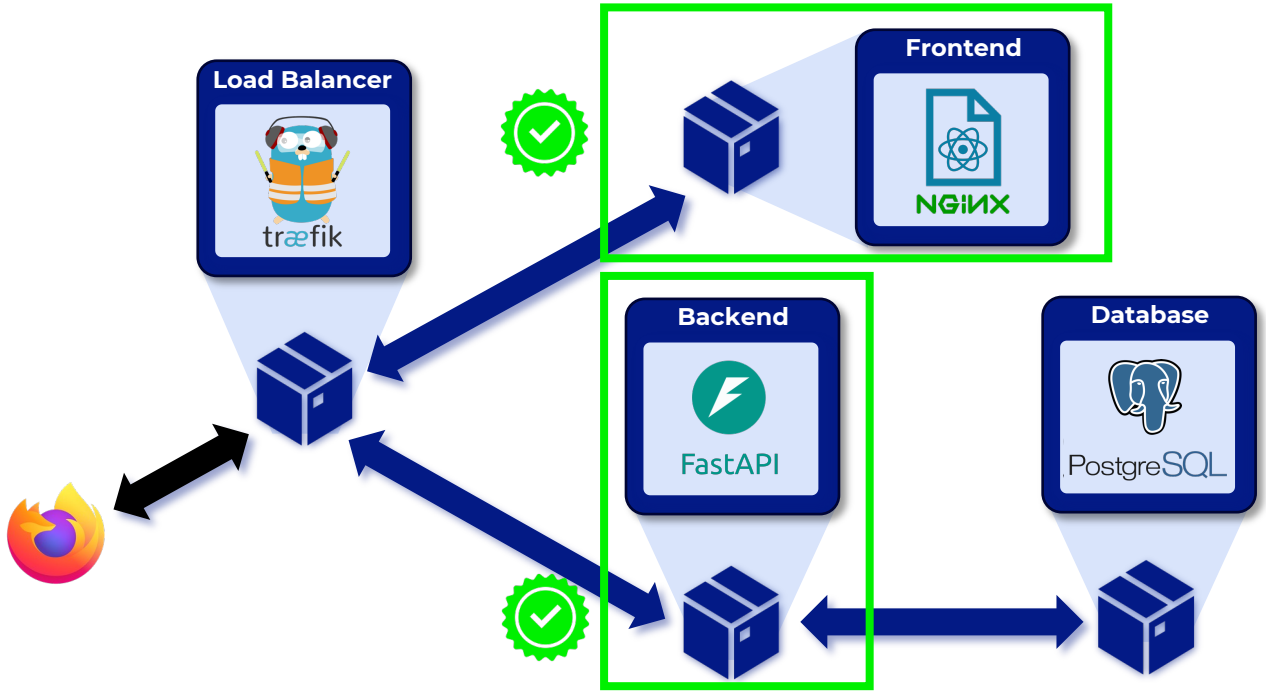
Follow consistent naming conventions

Keys should only contain lowercase alphanumeric characters and: `.`, `_`, `/`, `-`



Docker: The Efficient

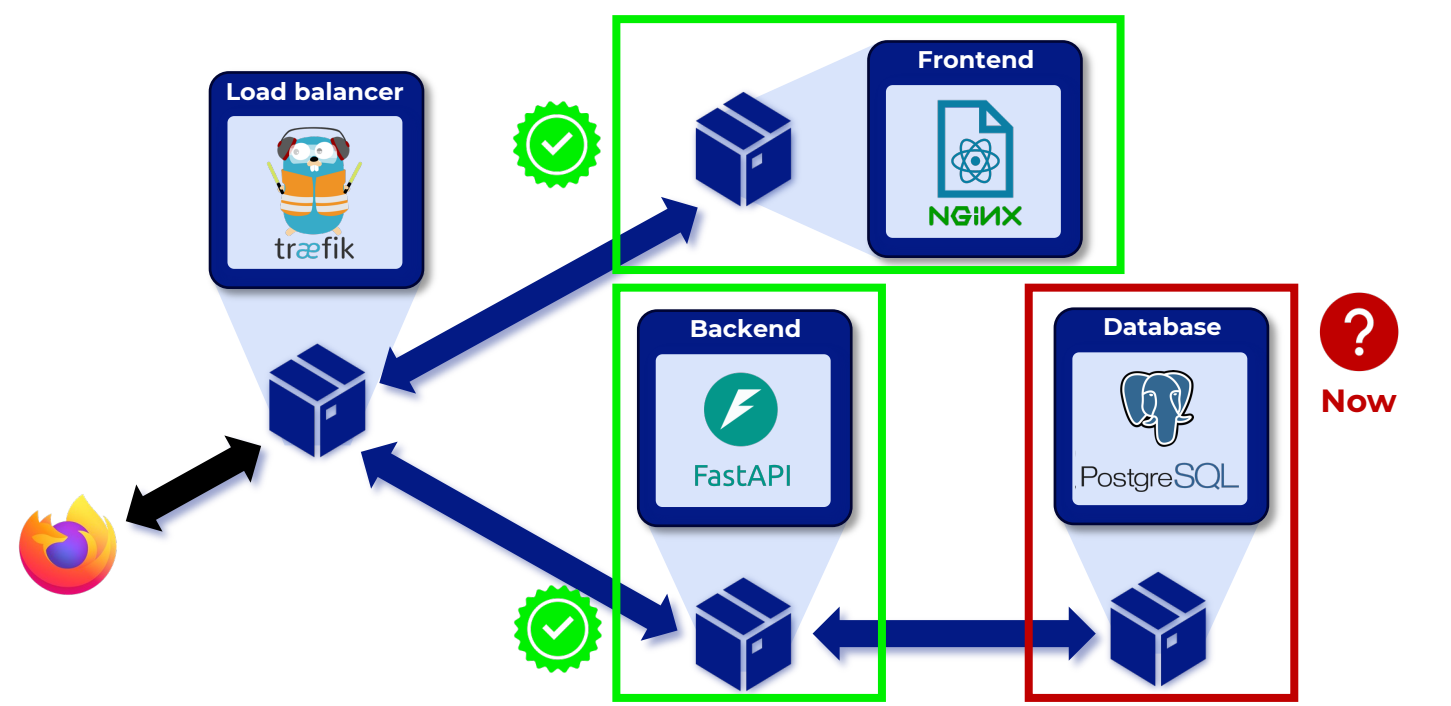
Day 1: Conclusion


Load Balancer
træfik
Frontend
NGINX
Backend
FastAPI
Database
PostgreSQL

Docker: The Efficient

Persistent storage for docker containers

Jannis Seemann


Load balancer
træfik
Frontend
NGINX
Backend
FastAPI
Database
PostgreSQL
Now

Data loss

Writable layers are ephemeral

When the container is removed: all stored data is lost

Limited scalability

Data inside the writable layer is tied to a single container

Cannot be shared between containers or accessed outside the container directly

Difficult to run multiple or new instances that share the same data

Docker: The Efficient

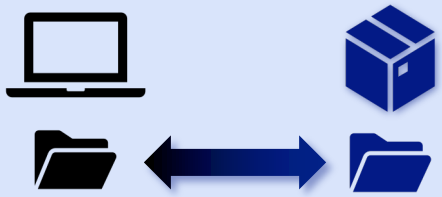
Storage mechanisms

Jannis Seemann

Bind mount

Link a specific host directory or file to a container path

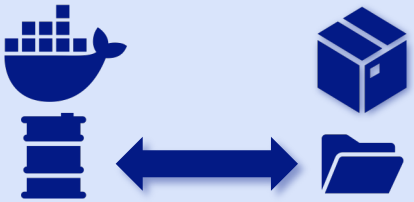
Directly reflects changes between host and container

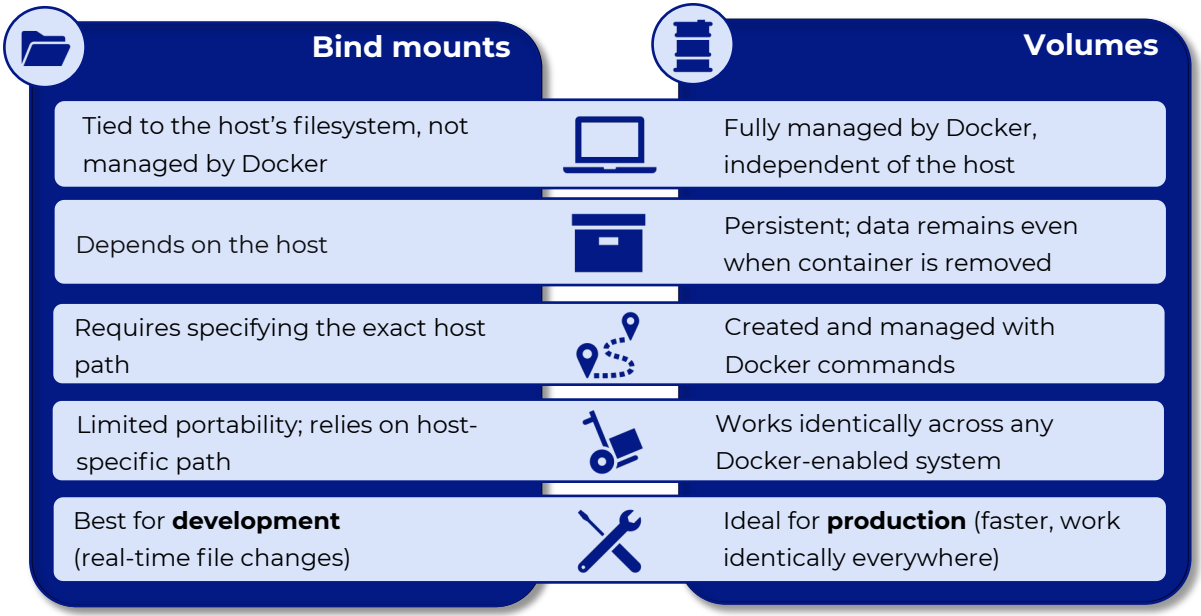


Volume

Provide a dedicated storage space independent of the host's filesystem for persistent container data

Managed by Docker




Bind mounts
Volumes
Tied to the host's filesystem, not managed by Docker
Fully managed by Docker, independent of the host
Depends on the host
Persistent; data remains even when container is removed
Requires specifying the exact host path
Created and managed with Docker commands
Limited portability; relies on host-specific path
Works identically across any Docker-enabled system
Best for development (real-time file changes)
Ideal for production (faster, work identically everywhere)

Docker: The Efficient

Bind mounts

Definition

A mechanism to link a host directory or file to a container

Changes made on the host or in the container are reflected in real-time

Important notes

Docker itself does not offer any means to manage bind mounts

Files depend on the host's lifecycle

Best practices

During development, or for infrequently accessed files (config files)

Configure the container's access (e.g., read-only: `ro`)

Use absolute paths for the path on the host

```
docker run -v /host/path:/container/path[:access] <image>
```

```
docker run --mount
type=bind,source=/host/path,target=/container/path[,access]
<image>
```

Docker: The Efficient

Persistent storage through volumes

Jannis Seemann

Definition

A Docker-specific resource to provide persistent storage beyond a container's lifecycle

Must be attached to a container during creation

Important notes

Some images automatically create volumes for you

You can manage volumes using `docker volume …`

Best practices

Use in production for reliable and managed data storage

Explicitly name volumes for easier management

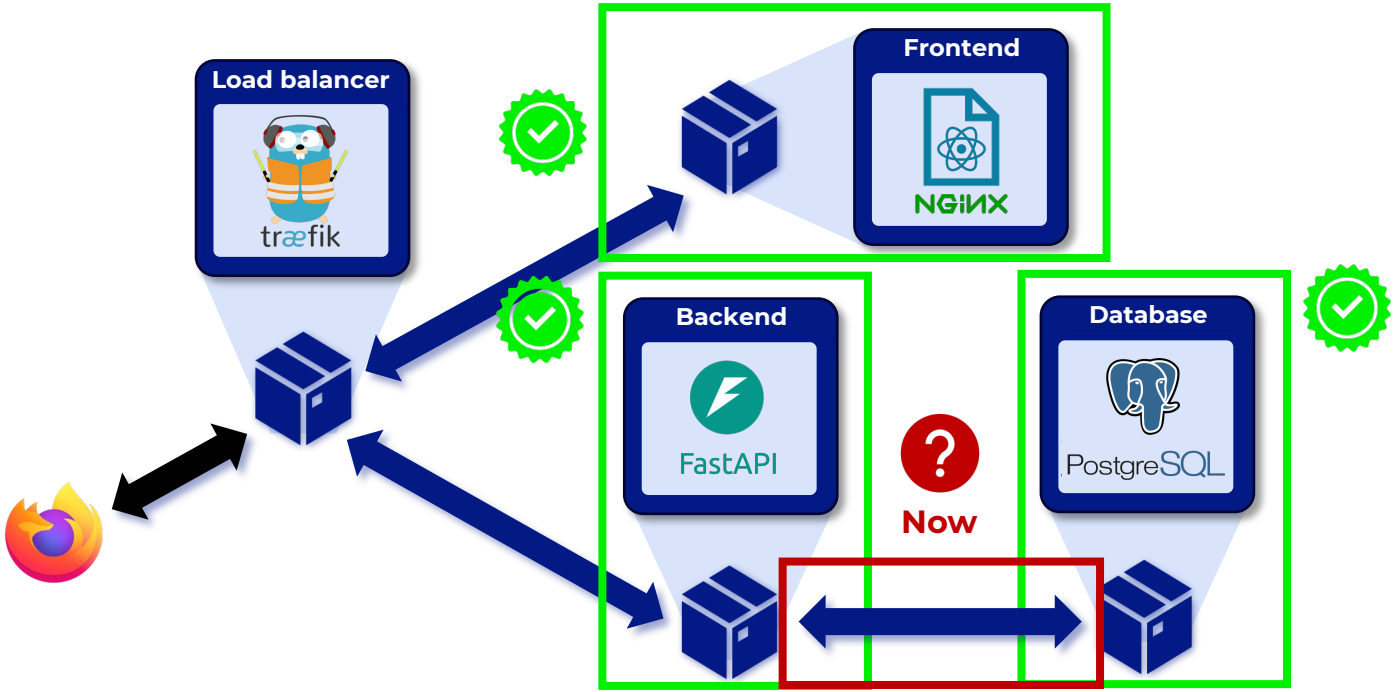
```
docker run -v <volume-name>:/container/path <image>
```

```
docker run --mount type=volume,source=<volume-name>,
target=/container/path <image>
```

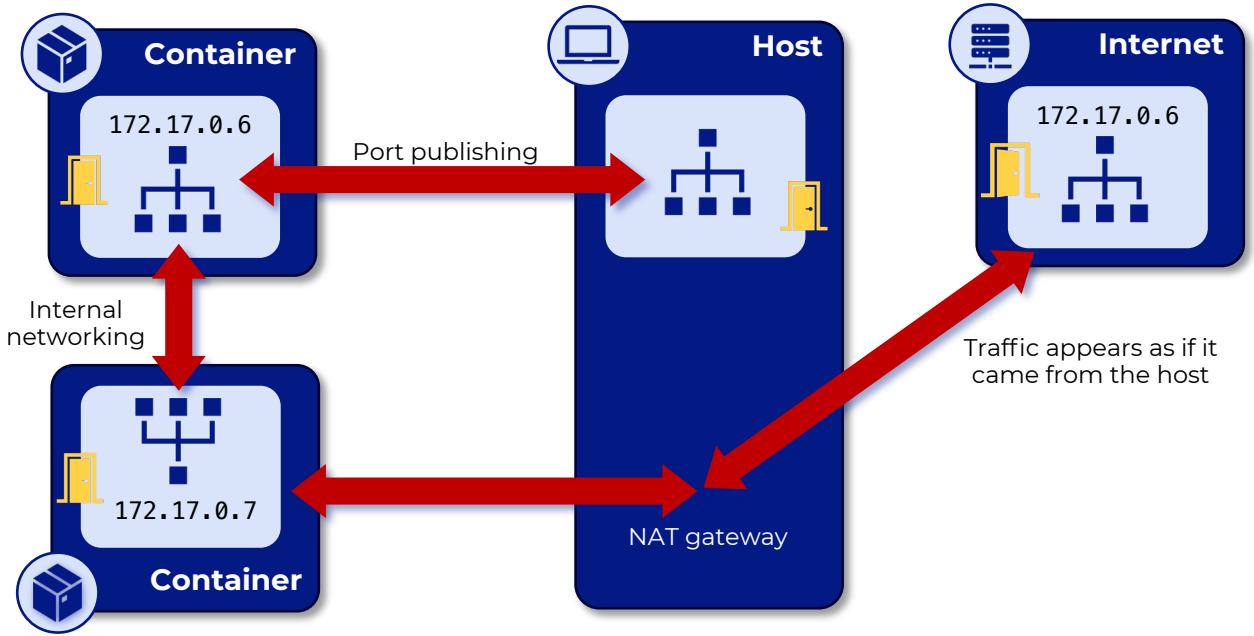
Docker: The Efficient

Container networking [Day 2]

Jannis Seemann


Load balancer
træfik
Frontend
NGINX
Backend
FastAPI
Database
PostgreSQL
?
Now

Container communication



Docker network modes

Bridge

- Each container gets its own isolated network stack, including an IP address
- Containers can communicate with each other using their IP
- Default mode

Host

- Removes any network isolation from the host
- The container shares the host's IP address and uses its ports directly
- Leads to low latency
- Only available on Linux

None

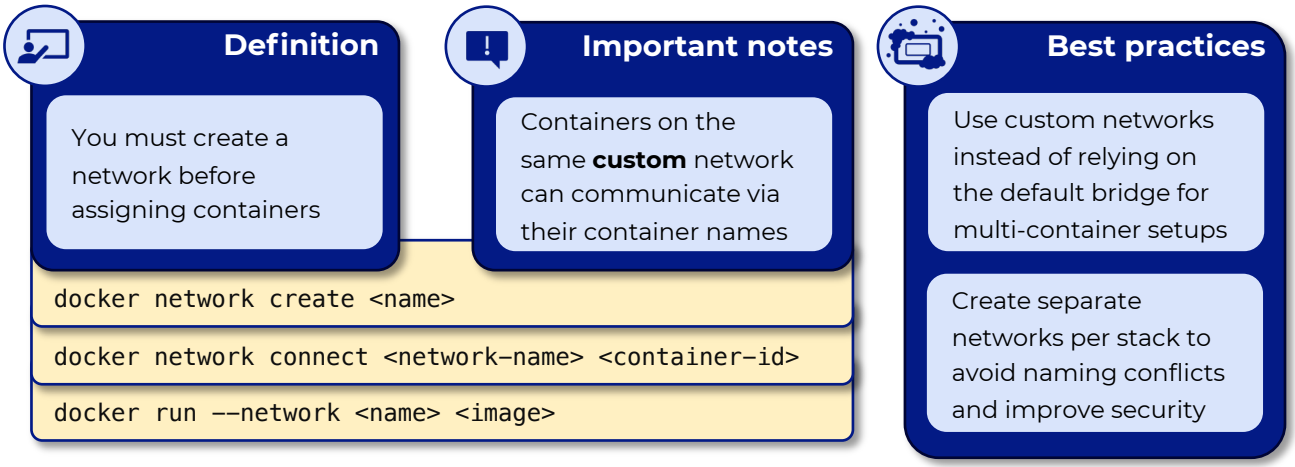
- The container has no network connectivity or IP address
- Completely isolated from any network
- Use for highly secure setups or manual configuration

Docker: The Efficient

Creating a custom network

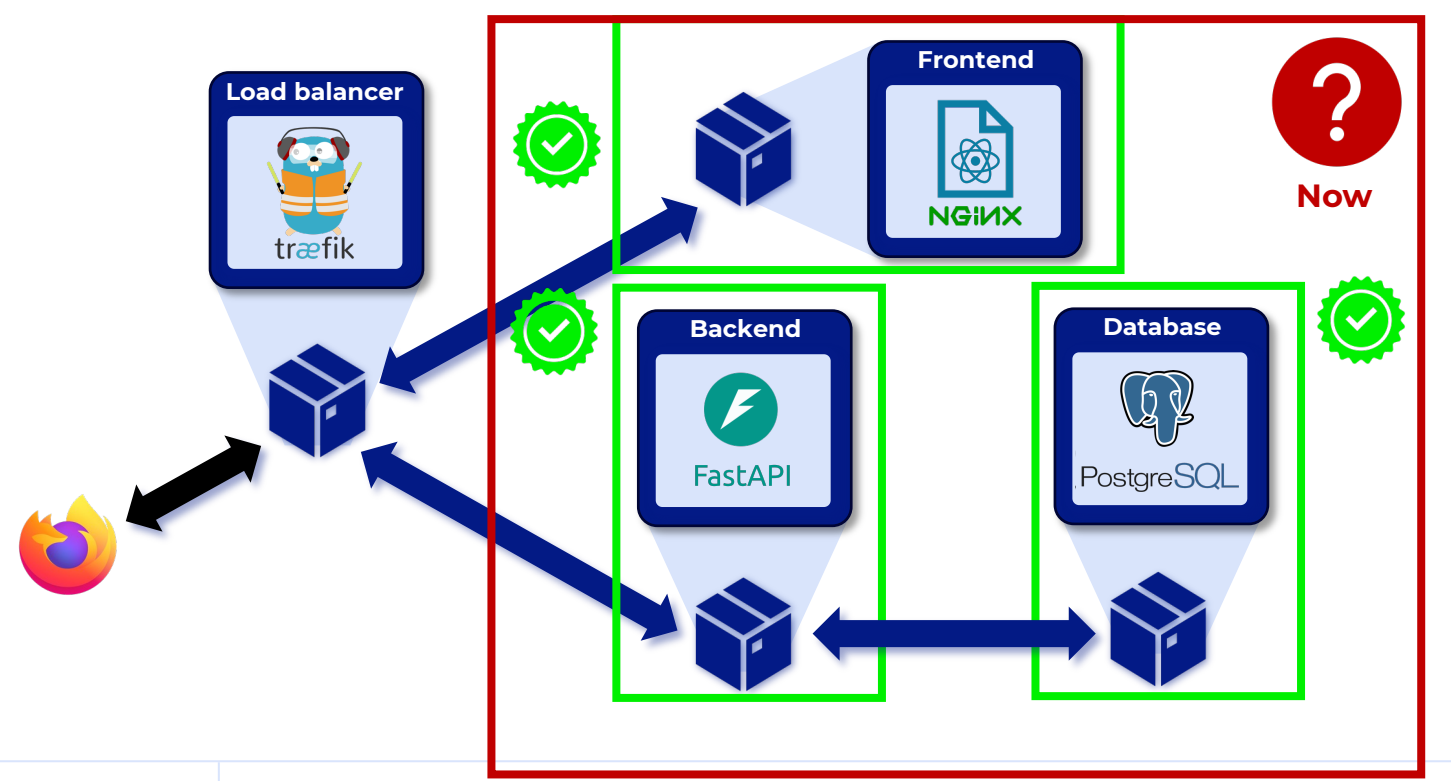
Jannis Seemann

Custom networks



Docker: The Efficient

Launching the application


Load balancer
træfik
Frontend
NGINX
Backend
FastAPI
Database
PostgreSQL
Now

Docker: The Efficient

The tool docker compose [Day 2]

Jannis Seemann

Manual setup problems

Configuration complexity

Each container must be started individually with long docker run commands

Requires manually setting environment variables, volumes, and networks

Reproducibility issues

Difficult to ensure the same configuration across multiple machines

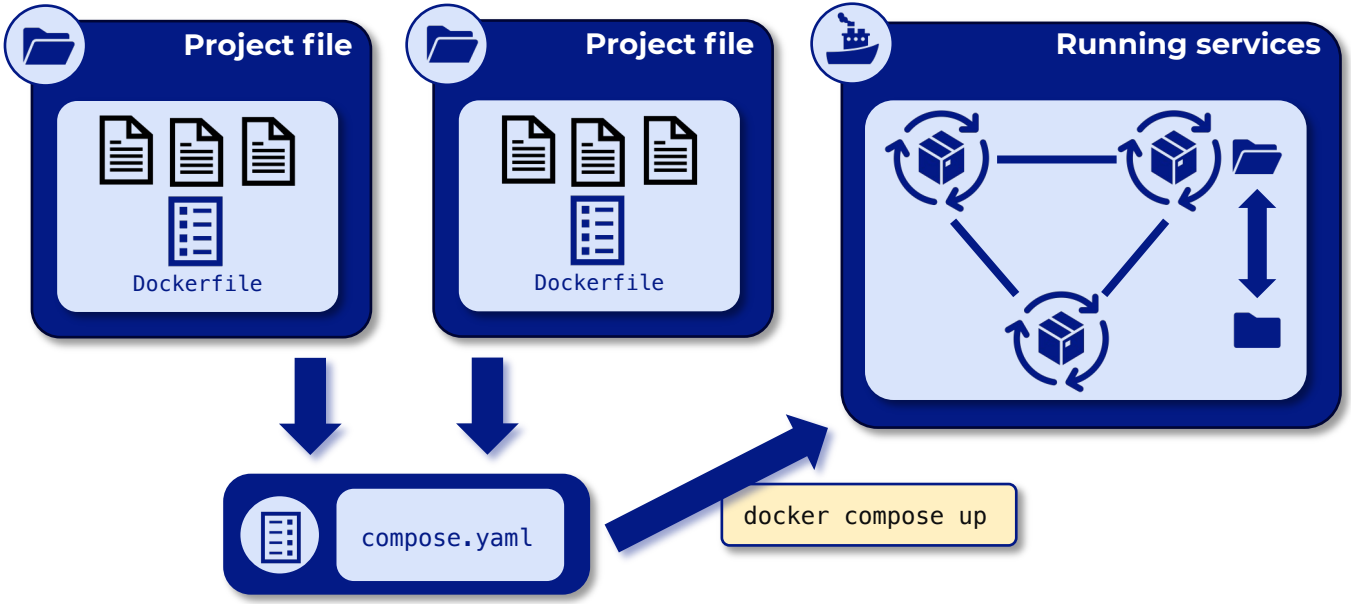
Requires extensive documentation for setup and dependencies

Scalability limitations

Scaling services requires manually running and managing multiple containers

No built-in way to define and maintain multi-container relationships

How docker compose works



Docker: The Efficient

Writing a compose.yaml file

Jannis Seemann

YAML Format

Human-readable configuration format

Relies on indentation and key-value pairs

Key Sections

`services:`

Specifies functional units, grouping containers by their roles (e.g., `app`, `db`)

`networks:`

Defines custom networks, lists all networks used in `services`

`volumes:`

Defines persistent storage, lists all named volumes used in `services`

Benefits

Centralizes configuration for multi-container setups

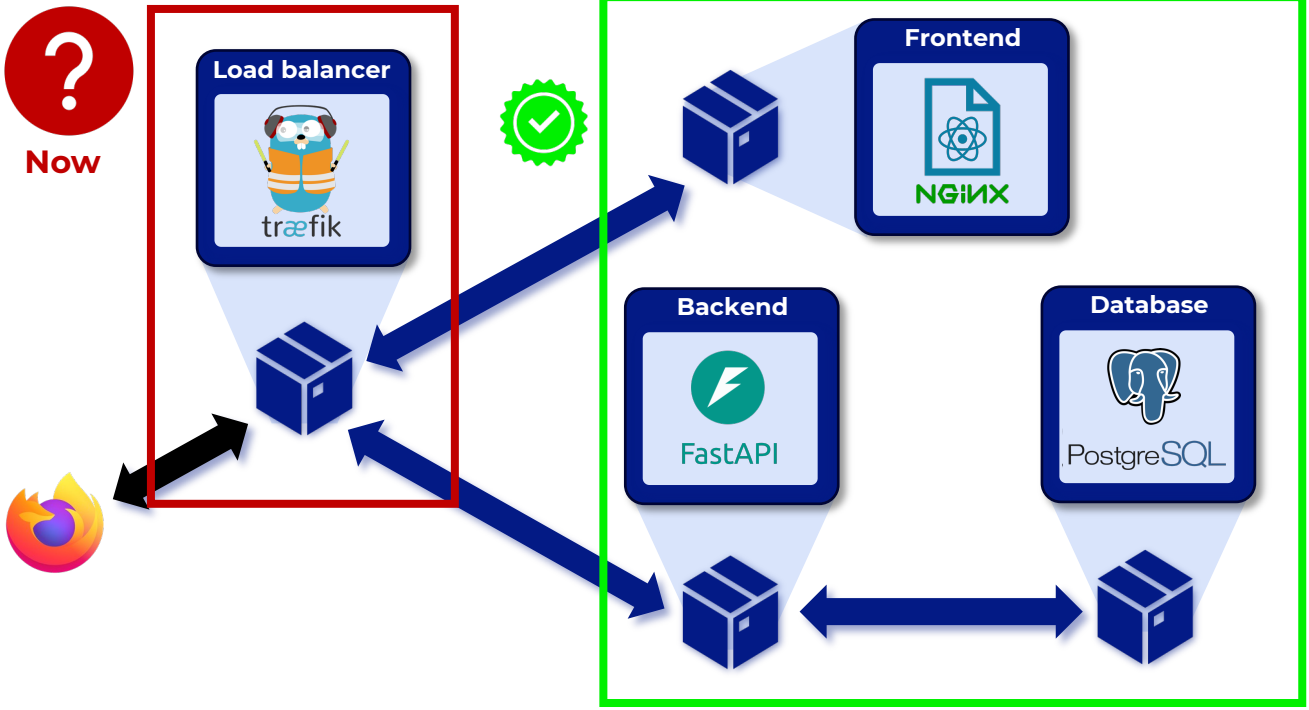
Ensures reproducibility across different environments

Simplifies deployment with a single command

Docker: The Efficient

Traefik: Load balancer

Jannis Seemann


Now
Load balancer
træfik
Frontend
NGINX
Backend
FastAPI
Database
PostgreSQL

Docker: The Efficient

Day 3: Using docker in daily workflows

Jannis Seemann

Overview

- Create a combined development / deployment setup in docker compose
- **Private registries & repositories:**
  - Where can we store our images?
  - Can we configure an automated deployment setup?
  - GitHub -> private registry -> server
- **How to deploy to a real server:**
  - How to connect to a remote server (ssh)
  - How to enable https?

Docker: The Efficient

Overview: How to distribute docker images

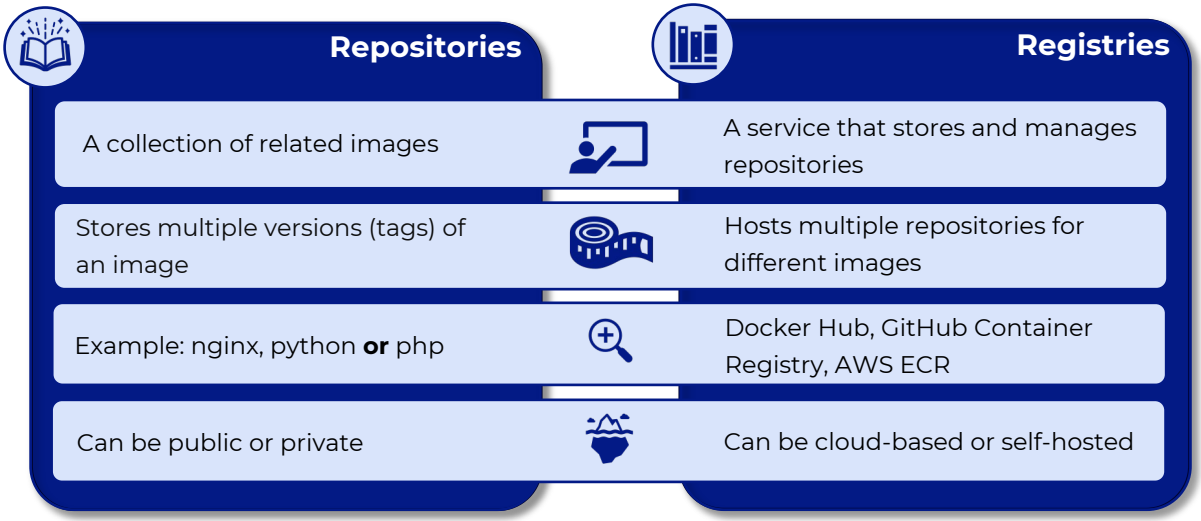
Jannis Seemann

Overview

- **You will learn:**
  - How to store an image in a private docker repository
  - This is fundamental when deploying to production
  - You will learn how to handle multi-architecture builds (arm64, amd64,...)
  - **This will lay the groundwork for the next section:**
    - Automating the entire deployment workflow

Docker: The Efficient

Storing an image in a private repository


Repositories
Registries
A collection of related images
A service that stores and manages repositories
Stores multiple versions (tags) of an image
Hosts multiple repositories for different images
Example: nginx, python **or** php
Docker Hub, GitHub Container Registry, AWS ECR
Can be public or private
Can be cloud-based or self-hosted