# Deployment

**Web Engineering**

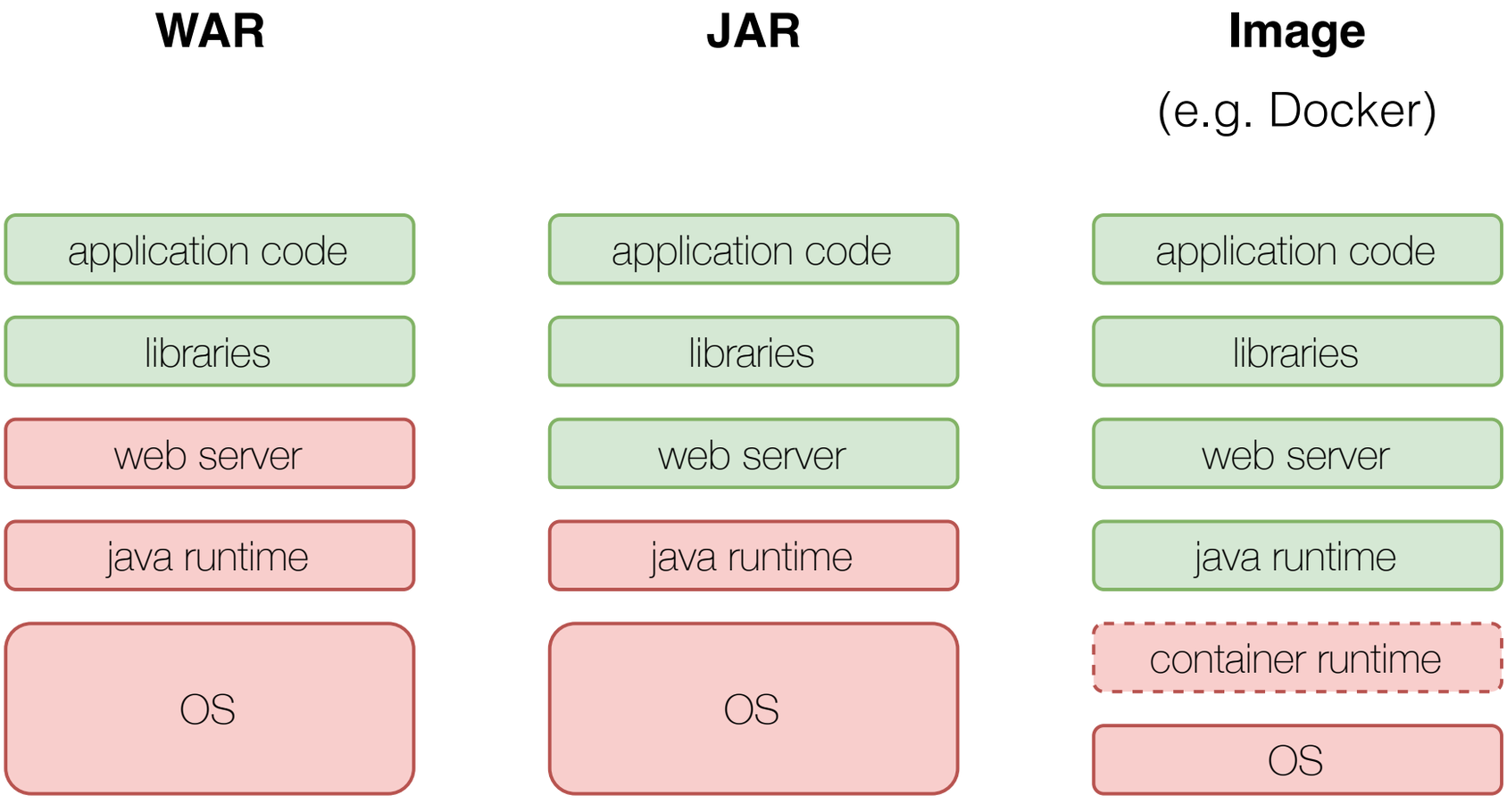
Markus Mächler

# It's Quiz Time!

https://api.socrative.com/rc/PnNn27

# Packaging

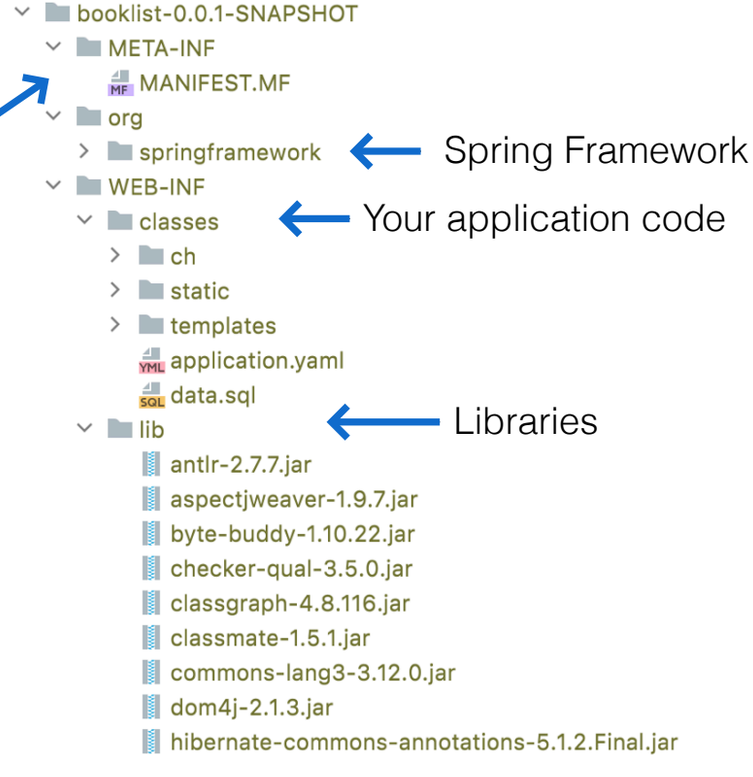
# Packaging of Java web applications



# WAR - Web Application Archive

- Contains application code and libraries
- Can be created with gradle **war** plugin
  - `./gradlew bootWar`
- Needs a web server (e.g. Tomcat) to run
- A **WAR** file is just a **ZIP** archive

```
1 plugins {
2     // ...
3     id 'war'
4 }
```

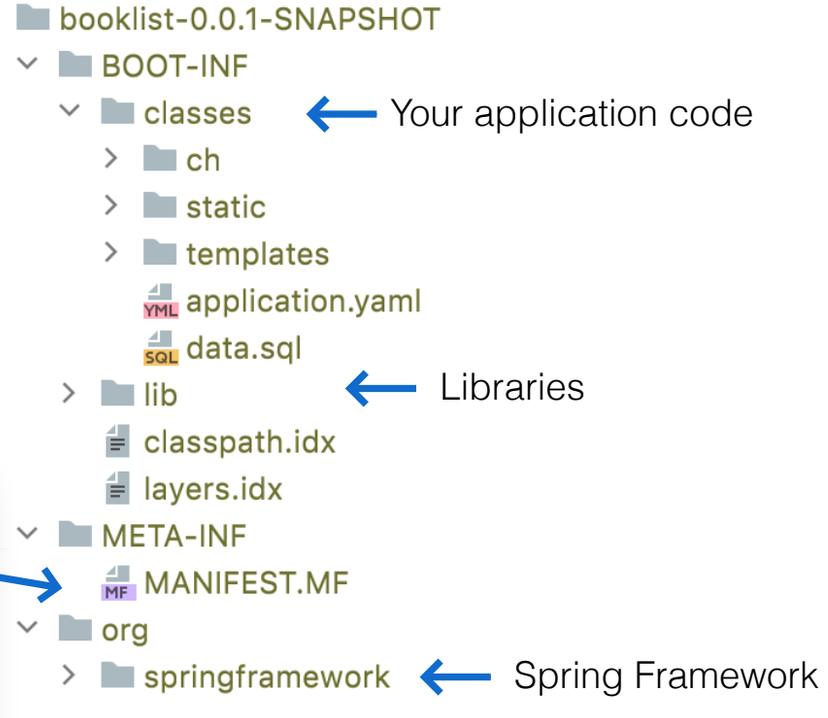



```
1  Manifest-Version: 1.0
2  Main-Class: org.springframework.boot.loader.WarLauncher
3  Start-Class: ch.fhnw.webec.booklist.BooklistApplication
4  Spring-Boot-Version: 3.1.4
5  Spring-Boot-Classes: WEB-INF/classes/
6  Spring-Boot-Lib: WEB-INF/lib/
7  Spring-Boot-Classpath-Index: WEB-INF/classpath.idx
8  Spring-Boot-Layers-Index: WEB-INF/layers.idx
9  Build-Jdk-Spec: 21
10 Implementation-Title: booklist
11 Implementation-Version: 0.0.1-SNAPSHOT
```

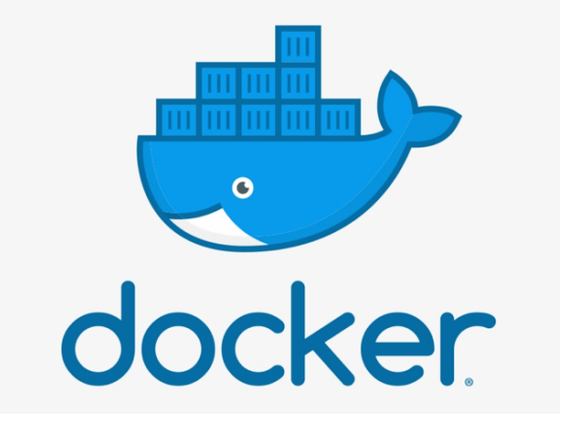
# JAR - Java Archive

- Contains application code, libraries and an embedded web server
- Can be created with gradle
  - `./gradlew bootJar`
- Needs a java runtime
  - `java -jar booklist.jar`
- A **JAR** file is just a **ZIP** archive

```
1  Manifest-Version: 1.0
2  Main-Class: org.springframework.boot.loader.JarLauncher
3  Start-Class: ch.fhnw.webec.booklist.BooklistApplication
4  Spring-Boot-Version: 3.1.4
5  Spring-Boot-Classes: BOOT-INF/classes/
6  Spring-Boot-Lib: BOOT-INF/lib/
7  Spring-Boot-Classpath-Index: BOOT-INF/classpath.idx
8  Spring-Boot-Layers-Index: BOOT-INF/layers.idx
9  Build-Jdk-Spec: 21
10 Implementation-Title: booklist
11 Implementation-Version: 0.0.1-SNAPSHOT
```



# Image

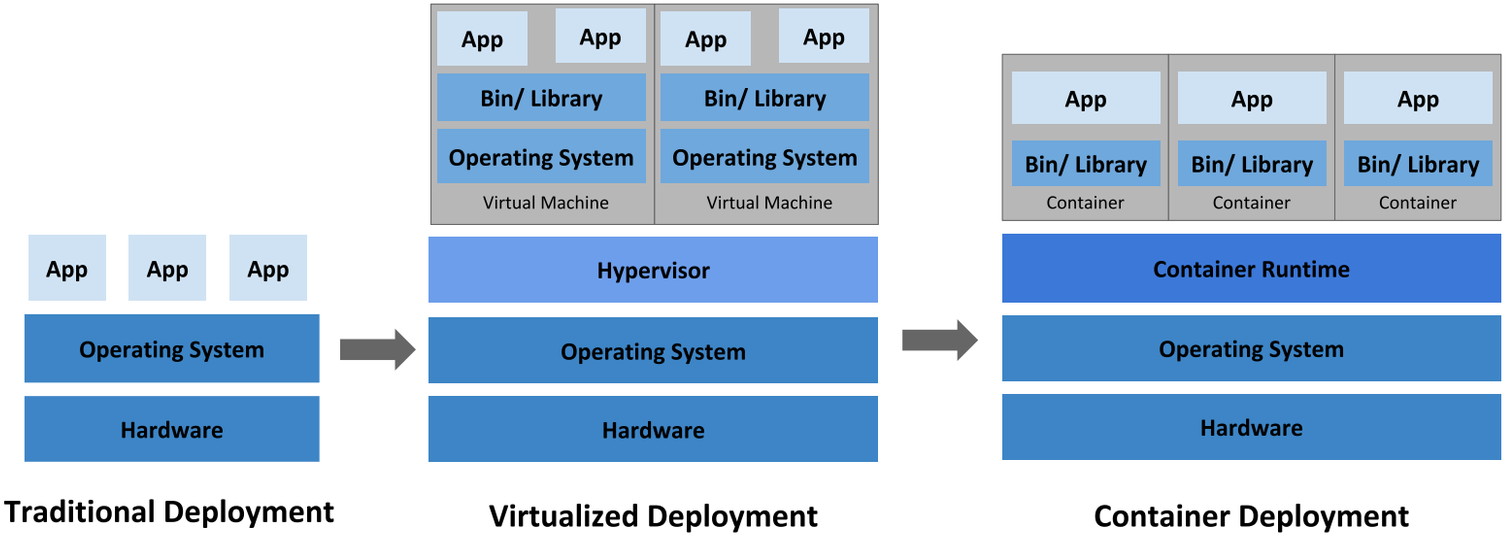



- Contains application code, libraries, web server and a java runtime
- Can be created with gradle
  - `./gradlew bootBuildImage --imageName=webec/booklist`
- Needs a container runtime (e.g. Docker Engine)
  - `docker run webec/booklist`

# Docker

# Docker

- Docker is a software that enables the creation of Linux or Windows containers
- Containers have their own **network interface**, **process-** and **disk-space**
- Docker uses features of the kernel (core of the OS) to segregate processes
  - Behind the scenes → Container from scratch [Kevin Boone]




Going back in time [kubernetes.io]

# Dockerfile

- A Dockerfile is the "recipe" for building a docker image
- It contains instructions such as:
  - **FROM** → inherit from an existing image or "scratch"
  - **RUN** → executes a command on top of the current image (e.g. apt-get)
  - **EXPOSE** → tells docker that the container listens on that port
  - **COPY** → copies files from the host to the image
  - **ENTRYPOINT** → defines the executable to run when the container starts

```
1 FROM eclipse-temurin:21-jre-alpine
2 EXPOSE 8080
3 COPY build/libs/booklist-0.0.1-SNAPSHOT.jar booklist.jar
4 ENTRYPOINT ["java","-jar","/booklist.jar"]
```

# Image

- An **image** *(cf. Java class)* is the template for creating a **container** *(cf. Java object)*
- Images are immutable, once built the files do not change anymore
- Images are built in layers
  - Each layer receives an ID (hash value of its contents)
  - FROM, RUN, COPY and ADD create new layers
- Images get an ID (hash value of layer hashes)
- Images can be tagged → assigning a label to the hash value to make it human readable (e.g. **latest**, **v1**, **v1.1**, etc.)

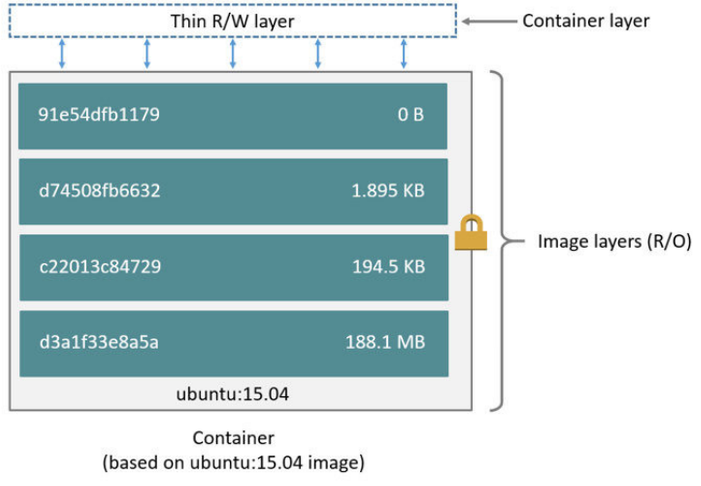



Image layers [docs.docker.com]

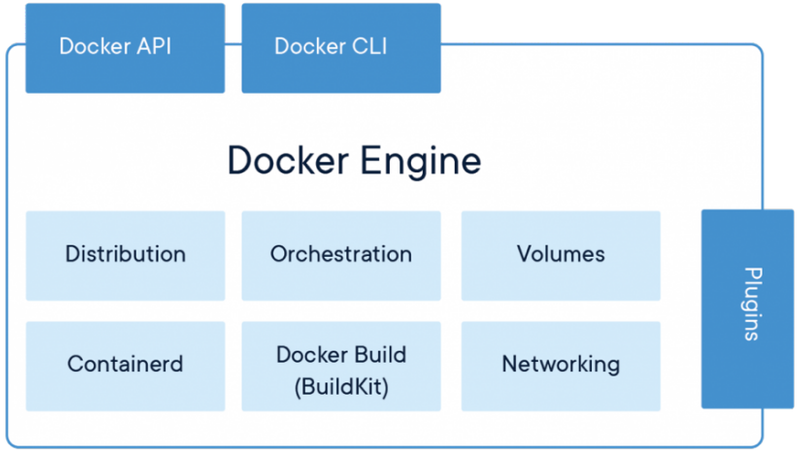
# Container

- A container represents an instance of a running image
- Containers have their own **network interface**, **process-** and **disk-space**
- When a container is stopped all filesystem changes are discarded

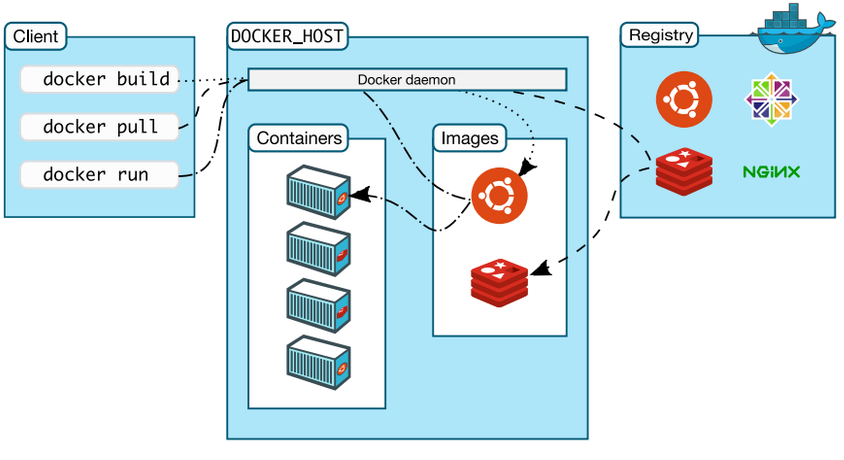
  → use volumes to persist changes
- OS Support
  - **Linux**: Linux containers implemented by the kernel
  - **Windows**:
    - Windows containers implemented by the kernel
    - Linux containers running on a linux distribution on Hyper-V
  - **macOS**: Linux containers running on a linux distribution on HyperKit
- The Open Container Initiative creates standards around containers

# Runtime

- The **Docker daemon** manages images, containers, networks, volumes, etc.
- The **Docker Engine** provides a **REST API** and **CLI** to interact with the docker daemon (e.g. docker build, docker pull, ...)



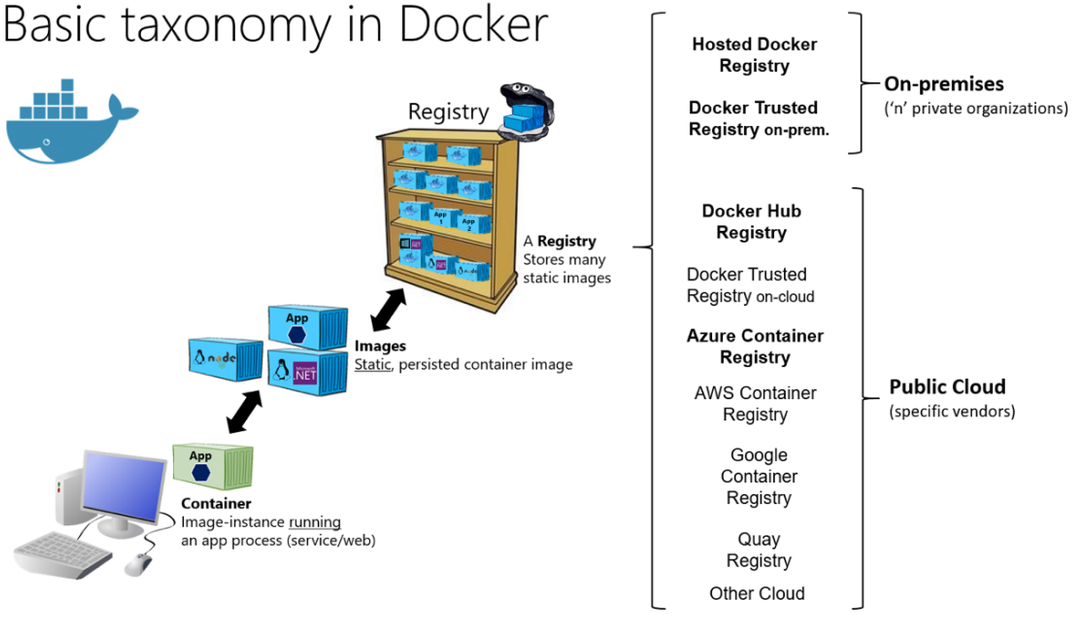

Container Runtime with Docker Engine [www.docker.com]




Docker architecture [docs.docker.com]

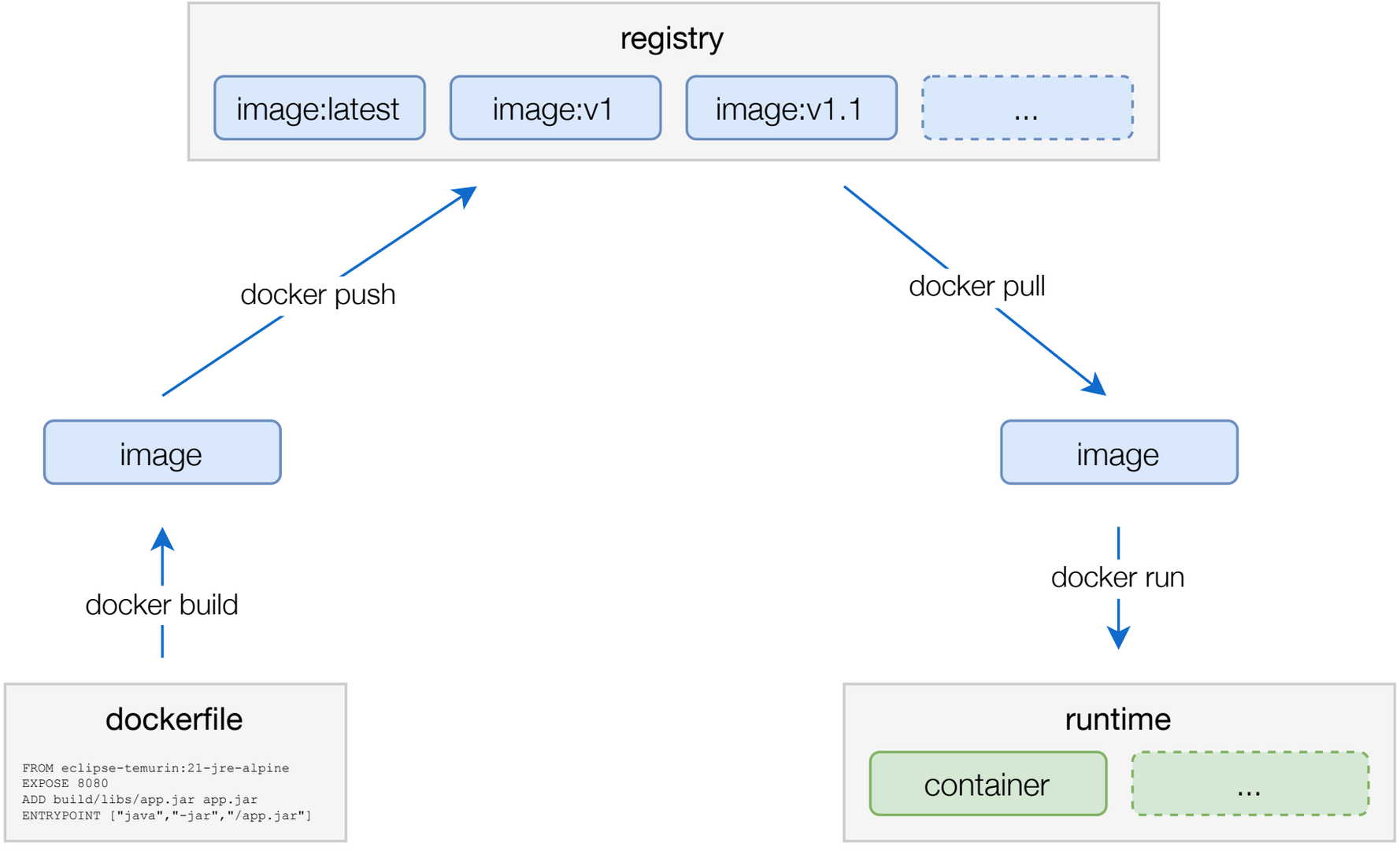
# Registry

- A registry is used to store and retrieve Docker images
- Docker Hub is a public registry provided and maintained by docker
  - Alternatives are GitHub Packages, Azure Container Registry, etc.




Taxonomy of Docker terms and concepts [docs.microsoft.com]

# Docker overview



# Infrastructure & Scaling

# Accessing a website through a browser


client

What's the IP for example.ch?

217.26.52.52

name server

client

Open a TCP connection

web server

217.26.52.52:80

client

HTTP Request

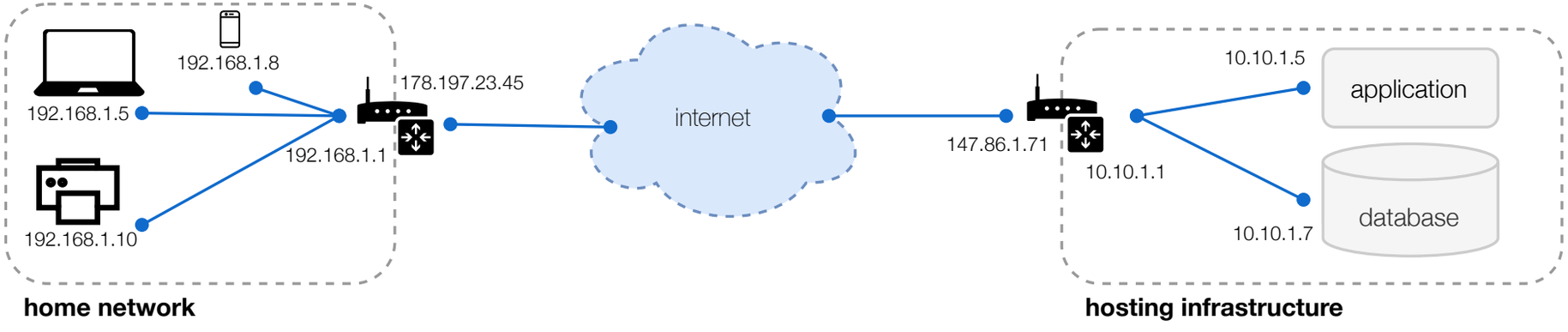
```
GET / HTTP/1.1
Host: example.ch
```

web server


HTTP Response

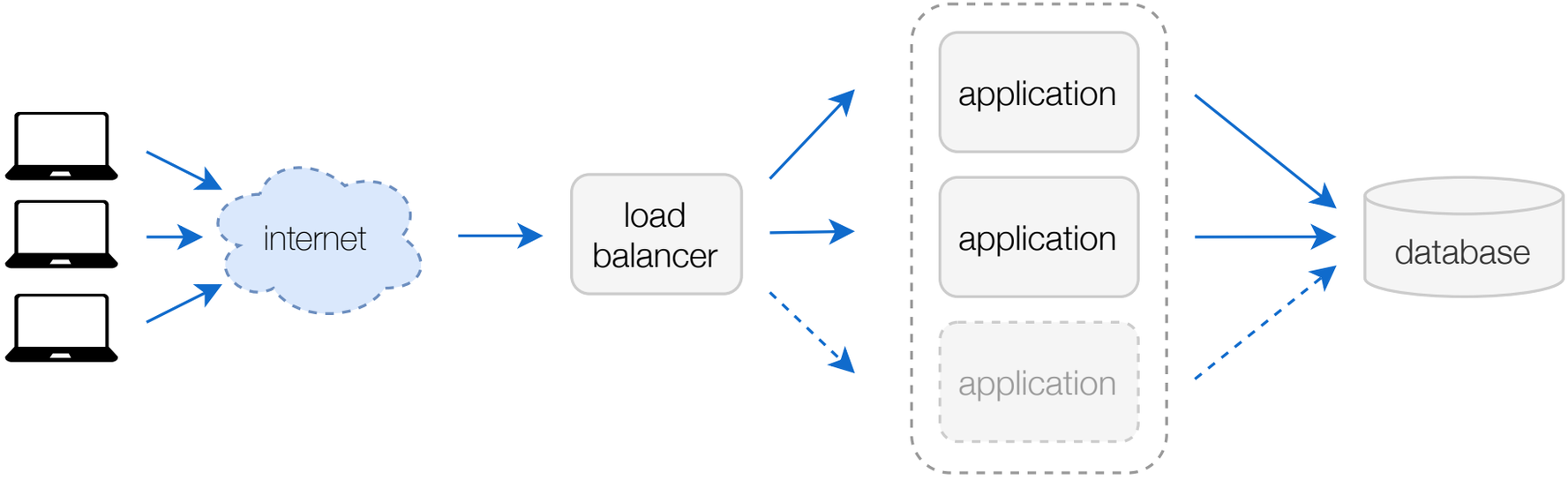
```
HTTP/1.1 200 OK
Content-Type: text/plain

Hello World
```

# Basic Setup



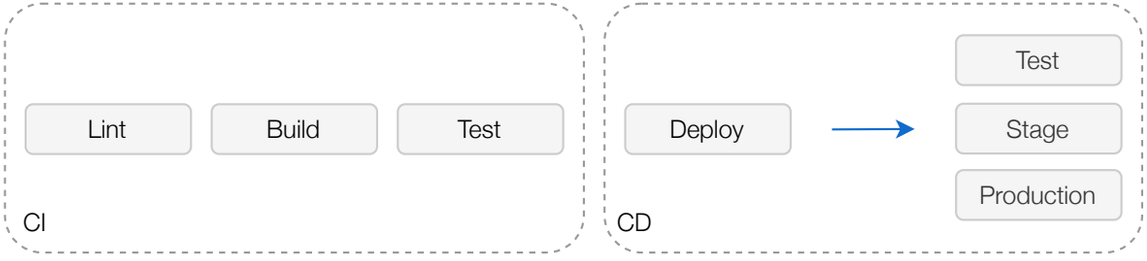

- What you need is:
  - A connection to the **Internet** (**public IP** address)
  - A **computer** to run your application (which has a web server included)
  - A **domain** and a **DNS** name server with an A-Record (domain → IP)
  - (e.g. your computer, home router with port forwarding and dynamic DNS)
- Application and database are usually separated, communicating over TCP

# Vertical and Horizontal Scaling




- **Vertical scaling** → adding more resources (RAM, CPU, ...) to a machine
  - Rather easy and cheap, but only limited scaling possible
- **Horizontal scaling** → adding more "machines" (e.g. containers)
  - A load balancer distributes the requests
  - More difficult (the application must be scalable → concurrency issues!), but potentially (almost) unlimited scaling

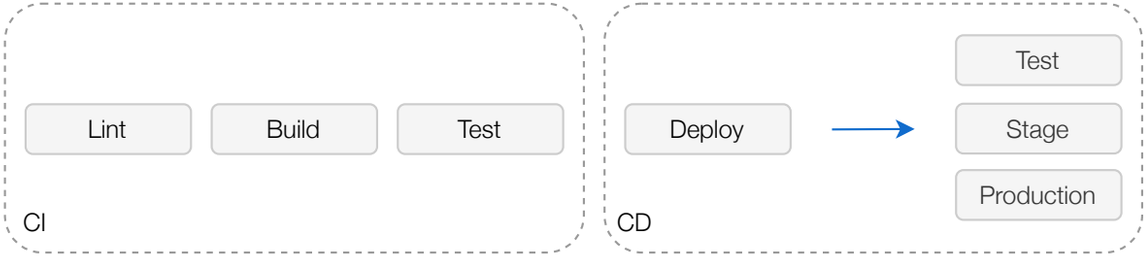
# CI/CD




- Continuous Integration
  - Use a git branching strategy like Gitflow or trunk-based development
  - Merge changes back to develop or main branch often

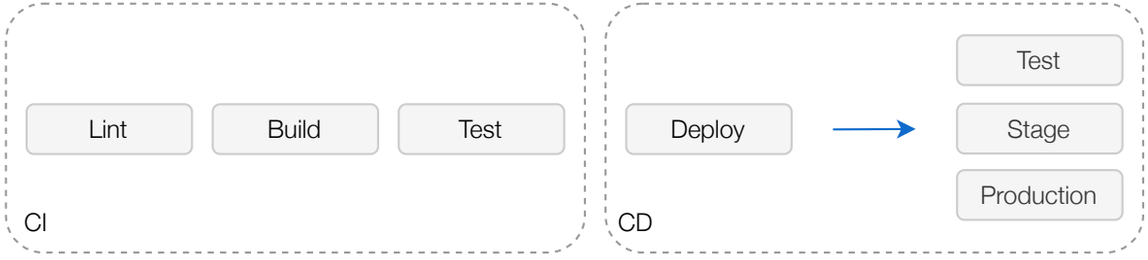
    → avoid integration challenges (merge conflicts)
  - CI pipeline
    - **Lint:** check code style, copy paste detector, etc.
    - **Build:** compile Java classes, build Angular application, etc.
    - **Test:** run Unit, Integration and E2E tests (in this order)
    - Triggered by e.g. pushing a commit or a merge request
      - Only allow to merge when CI pipeline runs through successfully

# CI/CD




- Continuous Delivery
  - Automating deployment (often to one of three environments)
    - **Test:** Current development (e.g. current develop branch)
    - **Stage:** "Exactly" resembles production, used for testing
    - **Production:** Productive environment users interact with
  - Continuous Deployment → automatically deploy each change that ran successfully through the pipeline

# CI/CD




```yaml
image: gradle:alpine
build:
    stage: build
    script: gradle --build-cache assemble
    cache:
        key: "$CI_COMMIT_REF_NAME"
        policy: push
        paths:
            - build
            - .gradle
test:
    stage: test
    script: gradle check
    cache:
        key: "$CI_COMMIT_REF_NAME"
        policy: pull
        paths:
            - build
            - .gradle
deploy:
    image: ruby:latest
    stage: deploy
    script:
        - gem install dpl
        - dpl --provider=heroku --app=$HEROKU_APP_NAME --api_key=$HEROKU_API_KEY
    only:
        - main
```

# What's more?

# What's more?

- Backup & Recovery
- Database migration (e.g. Flyway, Liquibase)
- Monitoring & Alerting (e.g. metrics from Spring Boot actuator)
- Caching (e.g. Varnish, Cloudflare, etc.)
- Security (Web Application Firewall, DDoS prevention)
- Microservices
- Container orchestration (Kubernetes, Docker Swarm, Apache Mesos, etc.)

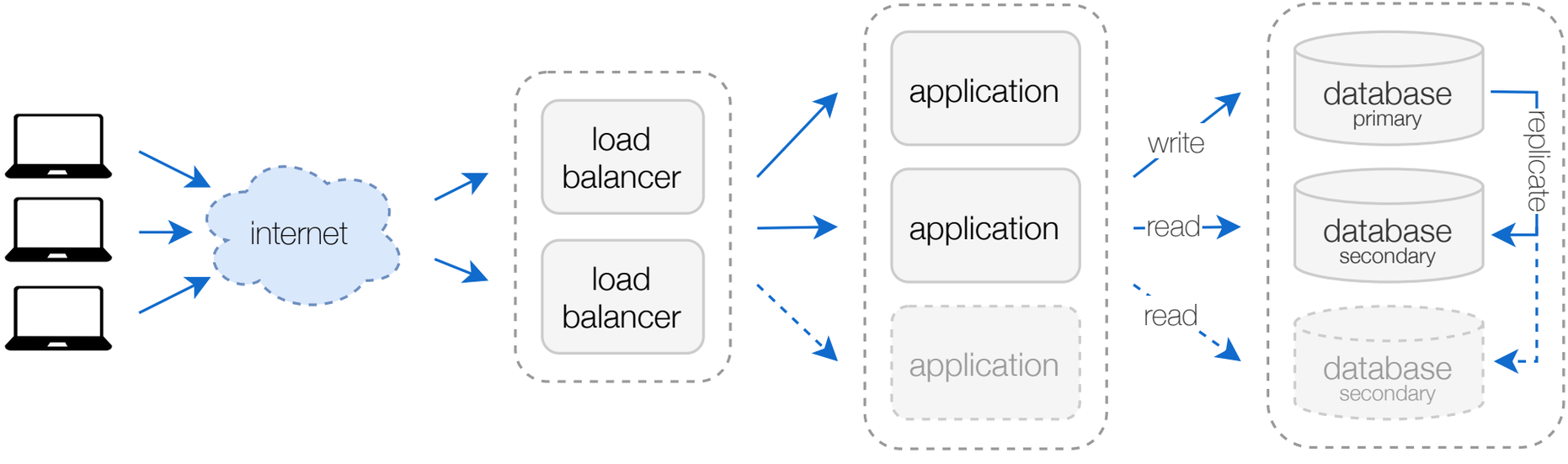
# Lessons learned

- You know the differences between **WAR**, **JAR** and **Image** packaging
- You know what a **Dockerfile**, **Docker Image**, **Container** and **Registry** is and how they are related to each other
- You know what is needed to make an application accessible over the Internet (computer, domain, public IP, DNS)
- You know the difference between **vertical** and **horizontal scaling**
- You know what **CI/CD** is


# Bonus Material

bonus material

# Even more Horizontal Scaling




- Scaling a load balancer
  - Round Robin DNS → assigning multiple IPs to a single domain
  - Anycast → assigning the same IP to multiple machines
- Scaling a database by separating **read** from **write** queries
- You probably do not need this → avoid premature optimisation