

# **Web Engineering 3**

## Vorlesung 1

Hochschule Zittau/Görlitz  
Christopher-Manuel Hilgner

# Agenda

- Organisatorische Sachen klären
- Klären und Wiederholen von Grundbegriffen
- Struktur einer Full-Stack-Webanwendung durchgehen
- Überblick über:
  - Database
  - Backend
  - Frontend
  - Containerization
- Kurz Git anschneiden bzw. wiederholen

# Organisatorisches

- Vorlesungen + Seminar
- Projektarbeit im Laufe des Semesters
- Gruppenarbeit möglich für das Projekt
- Crossbeleg mit anderen Modulen wird empfohlen
- Beleg mit Verteidigung am Ende des Semesters
- Zur Abgabe gehören:
  - Beleg als PDF
  - Folien der Verteidigung
  - Alle Projektdateien, in einem Git-Repository

# Beleg

- Keine festgelegte Seitenanzahl (bei mir waren es damals 10 bis 15 Seiten)
- Anforderungsanalyse
- Beschreibung der Umsetzung
- Gründe für Entscheidungen bei der Entwicklung darstellen
- Dokumentation der Software

# Verteidigung

Dauer: ca. 20 Minuten

- Anforderungen bzw. Ziele der Anwendung
- Demonstration des finalen Projekts
- Umsetzung Vorstellung
  - Technologien
  - Struktur
- Kurz ausgewählte Programmbestandteile vorstellen, die als wichtig angesehen werden

# Material

<https://github.com/XPromus/we3-hszg>

- Script, Folien, Beispielprojekte

# Inhalte

- Entwicklung von Full-Stack Web Anwendungen
- Spring für Backend-Entwicklung
- Docker/Podman zum Deployen von Anwendungen
- Frontend-Frameworks wie Svelte, Vue oder React...
- Debugging-Tools und Techniken
- Testing
- Authentication

# Grundbegriffe

## Client Side

- Alles, was mit dem Nutzer interagiert
- Gute UI und UX werden benötigt für einfache Interaktion

## Server Side

- Alles, was auf dem Server passiert
- Funktionen werden dem Frontend bereitgestellt

## API

- Das Frontend und Backend kommunizieren über APIs
- Erlauben es, Anwendungen in Schichten zu unterteilen
- Können der Middleware zugeordnet werden

# Grundbegriffe

## REST

- Architekturstil für Kommunikation in Websystemen
- Separation von Server und Client
- Komplett stateless: Server und Client brauchen keine Kontextinformationen über den jeweils anderen
- Kommunikation findet über Requests und Responses statt
- Alle CRUD-Operationen werden direkt auf HTTP-Methoden gemapped

# Grundbegriffe

## MVC - Model-View-Controller

- Design-Pattern zur Implementierung von User-Interfaces, Daten und Kontroll-Logik
- Fokus auf Separierung von Business-Logik und visueller Darstellung

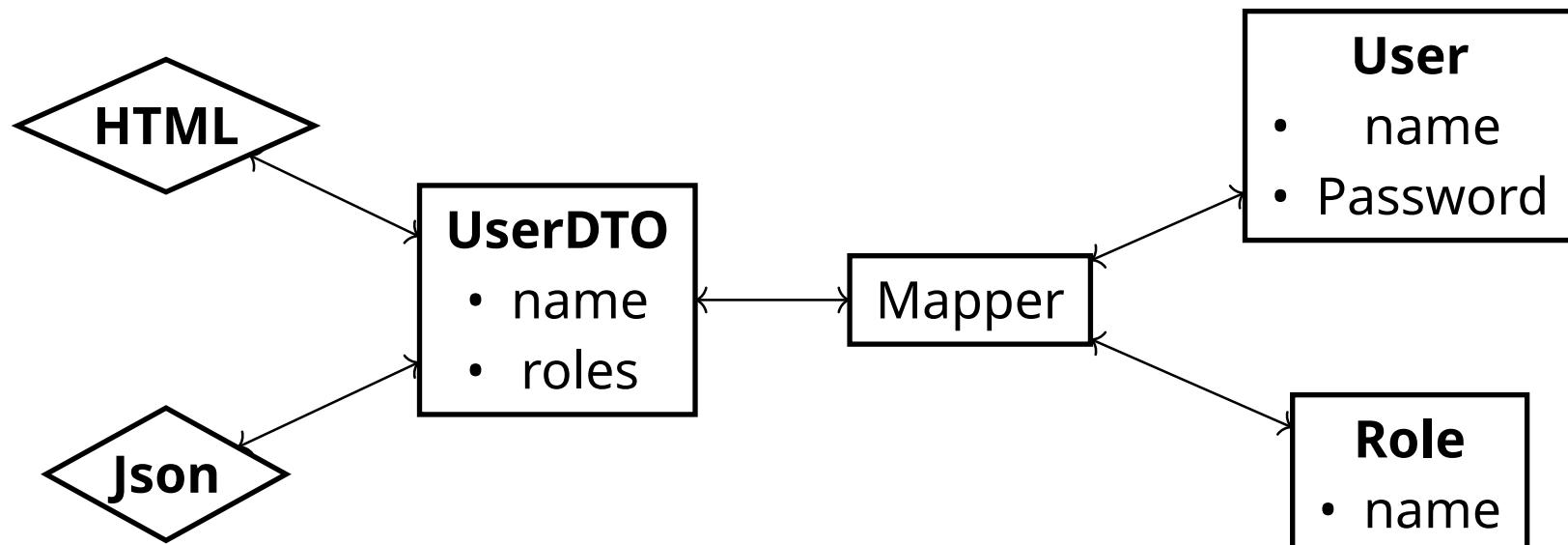
Komponenten:

- **Model:** Verwaltet Daten und Business-Logik
- **View:** Übernimmt Layout und Darstellung
- **Controller:** Leitet Befehle zum Model und View weiter

# Grundbegriffe

**DTO** - Data-Transfer-Object

- Bündelung von Datenfeldern in einem Objekt
- Einfachere Serialisierung und weniger Daten die verschickt werden müssen



# Grundbegriffe

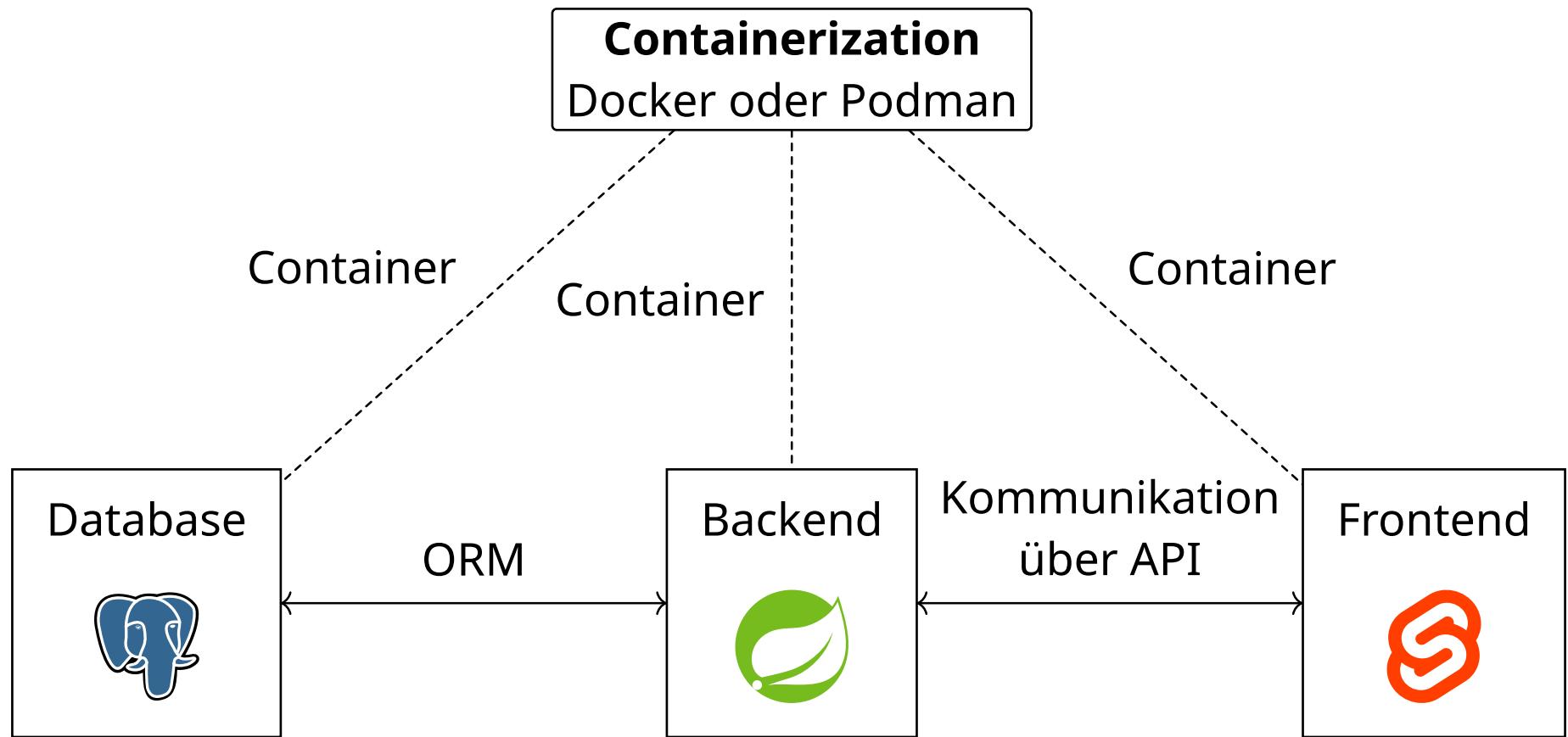
## CRUD

- Vier Grundoperationen, um mit persistenten Daten zu interagieren
- **Create:** Neue Dateneinträge werden gespeichert.
- **Read:** Existierende Daten werden ausgelesen
- **Update:** Existierende Daten werden aktualisiert
- **Delete:** Daten werden gelöscht

## ORM - Object-Relational-Mapping

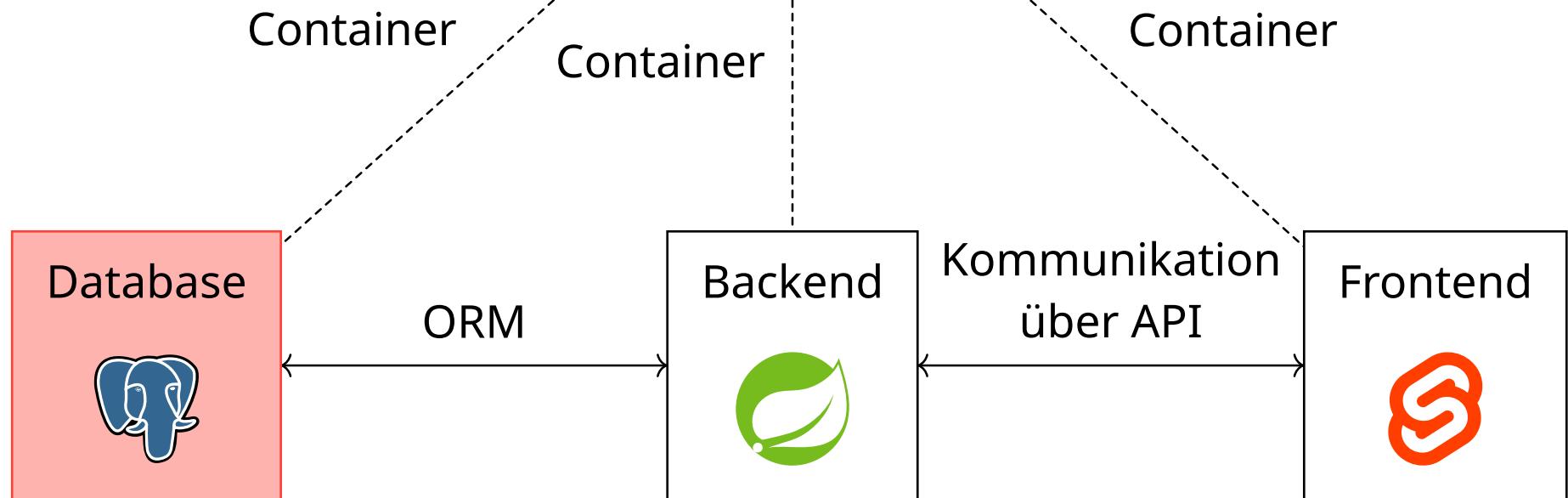
- Technik, um relationale Datenbank in Objekte einer Programmiersprache umzuwandeln
- Erstellung einer virtuellen Objekt Datenbank
- ORM Frameworks erlauben Durchführen von CRUD-Operationen
- SQL-Befehle müssen nicht per Hand geschrieben werden

# Struktur einer Full-Stack Web-Application



## Containerization

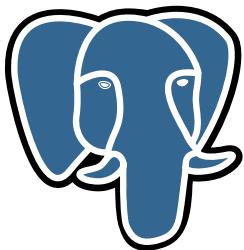
Docker oder Podman



# Database

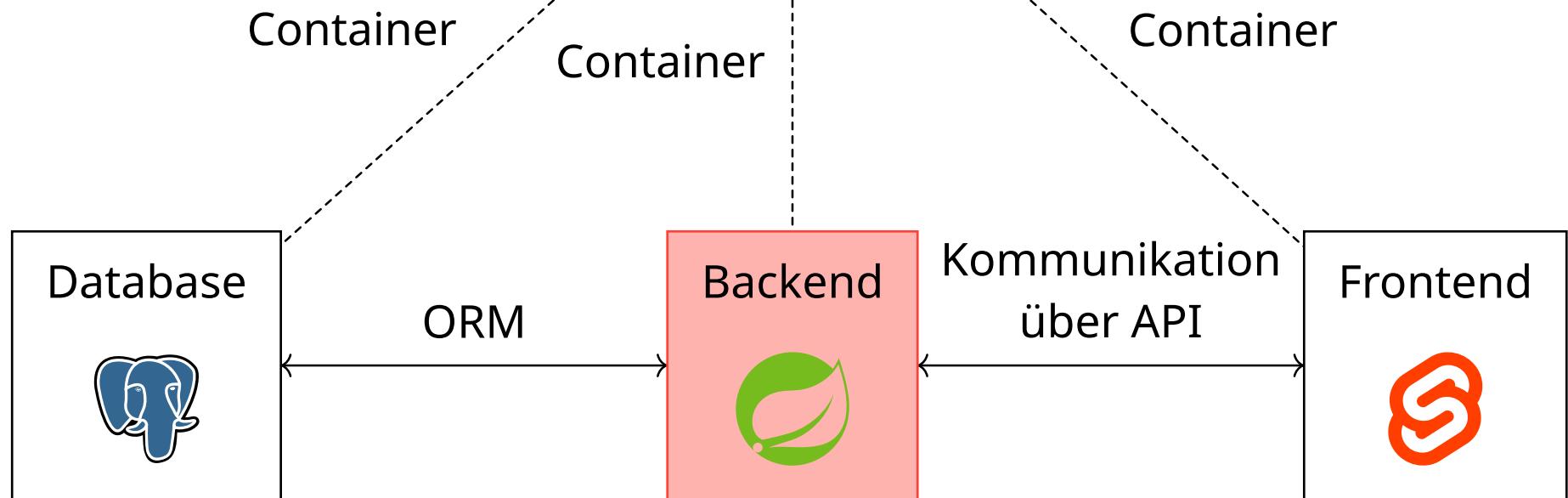
- Enthält alle zu speichernden Daten
- Manuelle Erstellung bzw. Konfigurierung nicht nötig
- Erstellung der Tabellen geschieht durch das Backend
- Manuelle Eingriffe nur dann in speziellen Situationen wie Migration nötig

In dieser Vorlesung kommt PostgreSQL zum Einsatz



## Containerization

Docker oder Podman



# Backend

- Definiert Datenstruktur mit Tabellen und Entities (ORM)
- Bevölkert Datenbank mit Daten und liest Daten aus (CRUD)
- Enthält Business-Logik der Anwendung
- Enthält REST-Endpunkte mit Mappings für HTTP-Funktionen

# Bestandteile

## Repositories

Ermöglichen Lesen und Schreiben auf die Datenbank.  
Sie stellen eine Menge an Methoden dafür bereit, je nachdem, welches Repository gewählt wird.

## Services

*mit DTOs und Mappern*

Verarbeitung von Daten und allgemeine Logik der Anwendung

## Controller

Erstellung der API-Endpunkte über Mappings

# Bestandteile

## Repositories

Ermöglichen Lesen und Schreiben auf die Datenbank.  
Sie stellen eine Menge an Methoden dafür bereit, je nachdem, welches Repository gewählt wird.

## Services

*mit DTOs und Mappern*

Verarbeitung von Daten und allgemeine Logik der Anwendung

## Controller

Erstellung der API-Endpunkte über Mappings

# Bestandteile

## Repositories

Ermöglichen Lesen und Schreiben auf die Datenbank.  
Sie stellen eine Menge an Methoden dafür bereit, je nachdem, welches Repository gewählt wird.

## Services

*mit DTOs und Mappern*

Verarbeitung von Daten und allgemeine Logik der Anwendung

## Controller

Erstellung der API-Endpunkte über Mappings



# Spring & Springboot

## Spring

- Stellt Infrastruktur bereit zur Erstellung von Enterprise-Anwendungen
- Kern ist dabei Dependency Injection mit dem IoC-Container
- Weitere Funktionalitäten können über Module hinzugefügt werden
- Durch Spring soll Entwicklungszeit stark verkürzt werden

## Springboot

- Erweiterung und Abstraktion des Spring-Frameworks
- Erstellung von Spring-Anwendungen mit minimaler Konfiguration
- *Starter Dependencies* ermöglichen schnelle Erstellung von Anwendungen

# Alternativen



QUARKUS

# Gradle



- Automatisierung von Building, Testing und Deployment
  - Verwaltet Abhängigkeiten
  - Reproduzierbare Builds
  - Erweiterbar durch Plugins
- 
- Build Vorgang wird in der `build.gradle(.kts)` beschrieben

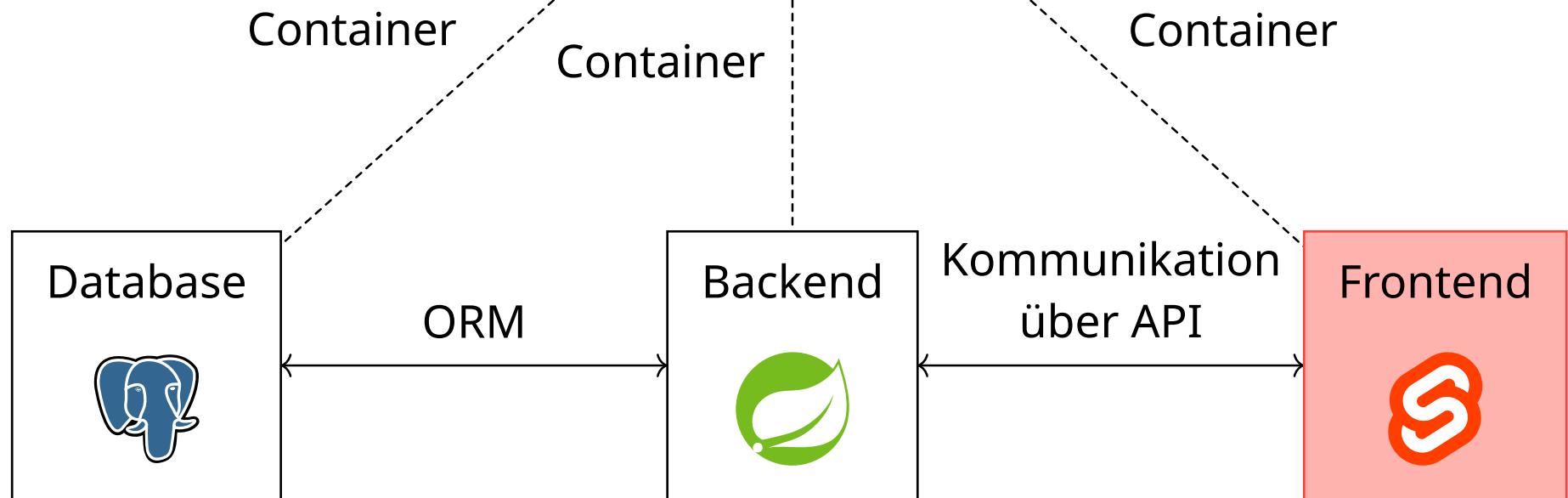
# Gradle Projekt Struktur

```
1  project
2    └── gradle
3    └── gradlew
4    └── gradlew.bat
5    └── settings.gradle(.kts)
6    └── subproject-a
7      └── build.gradle(.kts)
8      └── src/
9    └── subproject-b
10      └── build.gradle(.kts)
11      └── src/
```

 Markdown

## Containerization

Docker oder Podman



# Frontend

- Interaktion durch den Nutzer
- Darstellung der vom Backend erhaltenen Daten
- Input für neue Daten
- Durchdachtes User-Interface mit guter User-Experience





imgflip.com

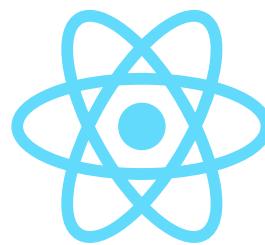
# Frameworks



Angular



VueJS



ReactJS



SvelteKit

# Frameworks

In dieser Vorlesung kommt SvelteKit mit TypeScript zum Einsatz. Andere Frameworks werden auch kurz beleuchtet.

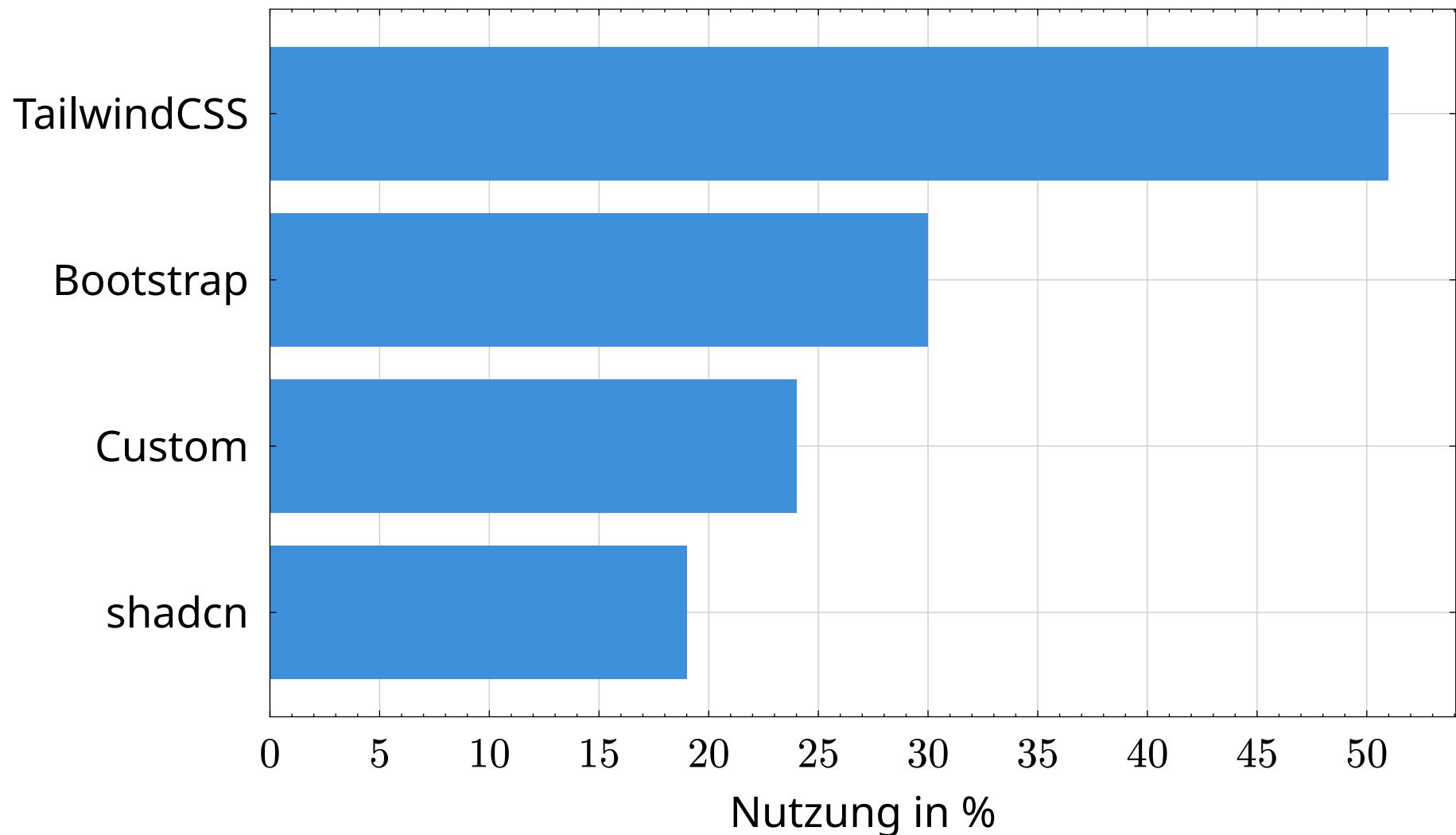


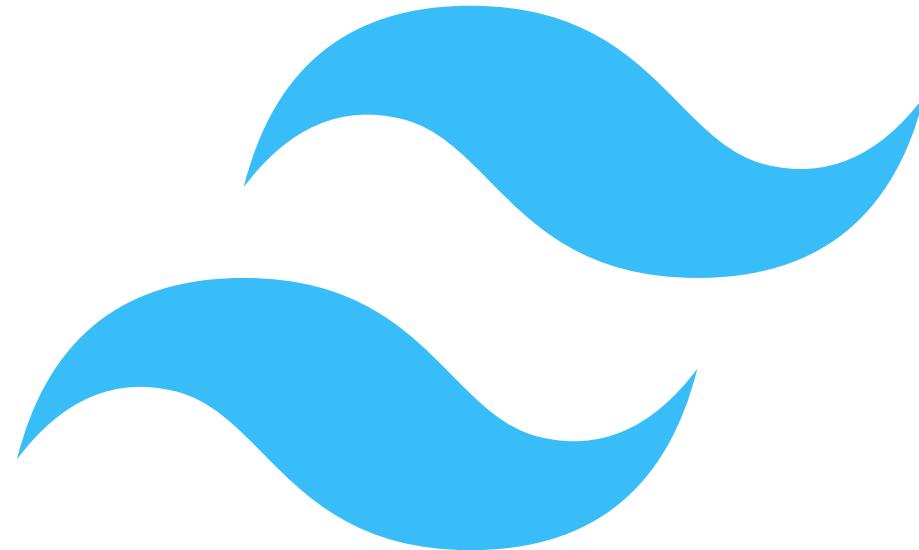
# Svelte

- Framework zum Bauen von User-Interfaces für das Web
- Deklarative Erstellung von Komponenten mit HTML, CSS und JavaScript
- Nutzen eines Compilers, um die Komponenten in imperatives JavaScript, für den Browser, zu überführen
- Erweiterung mit SvelteKit für:
  - Server-Side-Rendering
  - Routing
  - Code Splitting

# CSS Frameworks

Nutzungsdaten von CSS Frameworks nach dem State of CSS 2025





**TailwindCSS**

# TailwindCSS

- TailwindCSS als Framework für das Styling
- Es stellt keine Komponenten, sondern nur kleine Klassen bereit
- Näher an Vanilla-CSS als Frameworks wie Bootstrap

```
1  <div class="flex flex-col bg-slate-500">
2      <!-- Content -->
3  </div>
```

5 HTML

# Build-Tools

- Überführen den geschriebenen Code in eine ausführbare Version für den Browser
- Viele Features von Frameworks werden nicht von Browsern unterstützt
- Build-Tools überführen diesen Code in brauchbaren Code für den Browser

# Build-Tools

```
1  const App = () => <h1>Hello, World!</h1>;
```

jsx

wird zu

```
1  const App = () => React.createElement(  
2      'h1', null, 'Hello, World!'  
3  );
```

js JavaScript

# Build-Tools

## Bundling

- Zusammenführen von einzelnen Modulen für effizienteres Ausführen im Browser

## Transpilieren

- Kompatibilität zu älteren Browsern herstellen

# Build-Tools



**Vite**

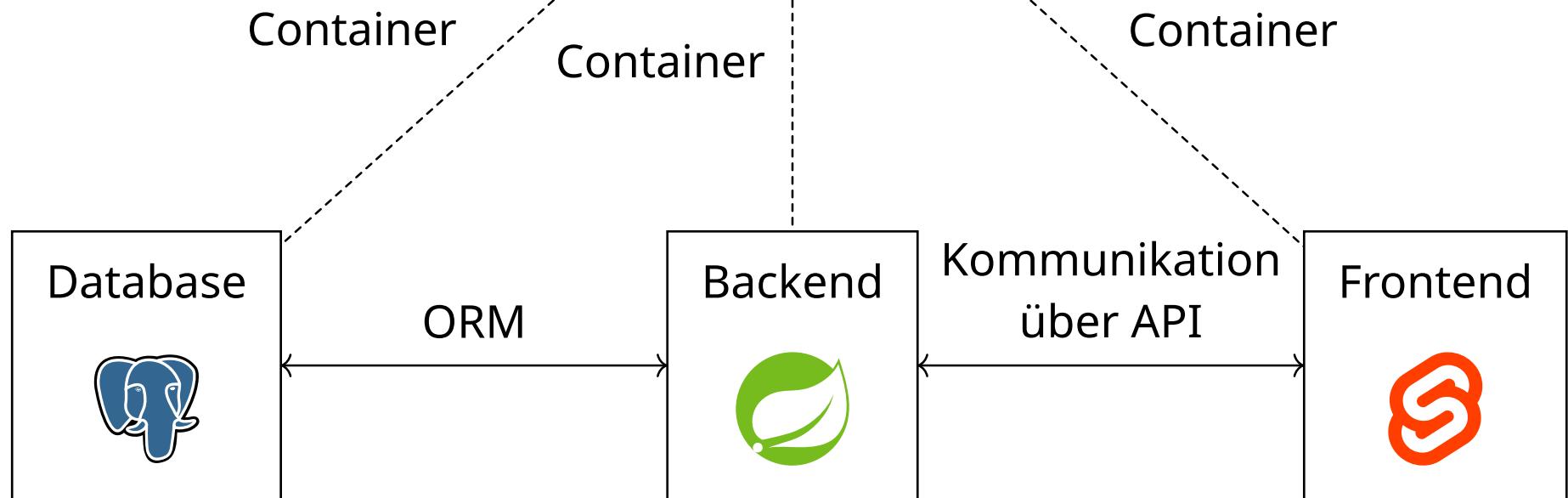
# Package Manager

- Definition der Abhängigkeiten durch eine JSON-Datei
- Verwaltung der Abhängigkeiten mit Versionierung
- NPM Registry stellt Packages bereit (> 3.100.000)
- Auswahl: npm, yarn, bun, deno usw.
- Alle können die npm-Registry nutzen, aber bieten unterschiedliche Vorteile in der Entwicklungserfahrung
- Vorteile bei npm-Alternativen liegen oft in der Performance

# **It works on my machine**

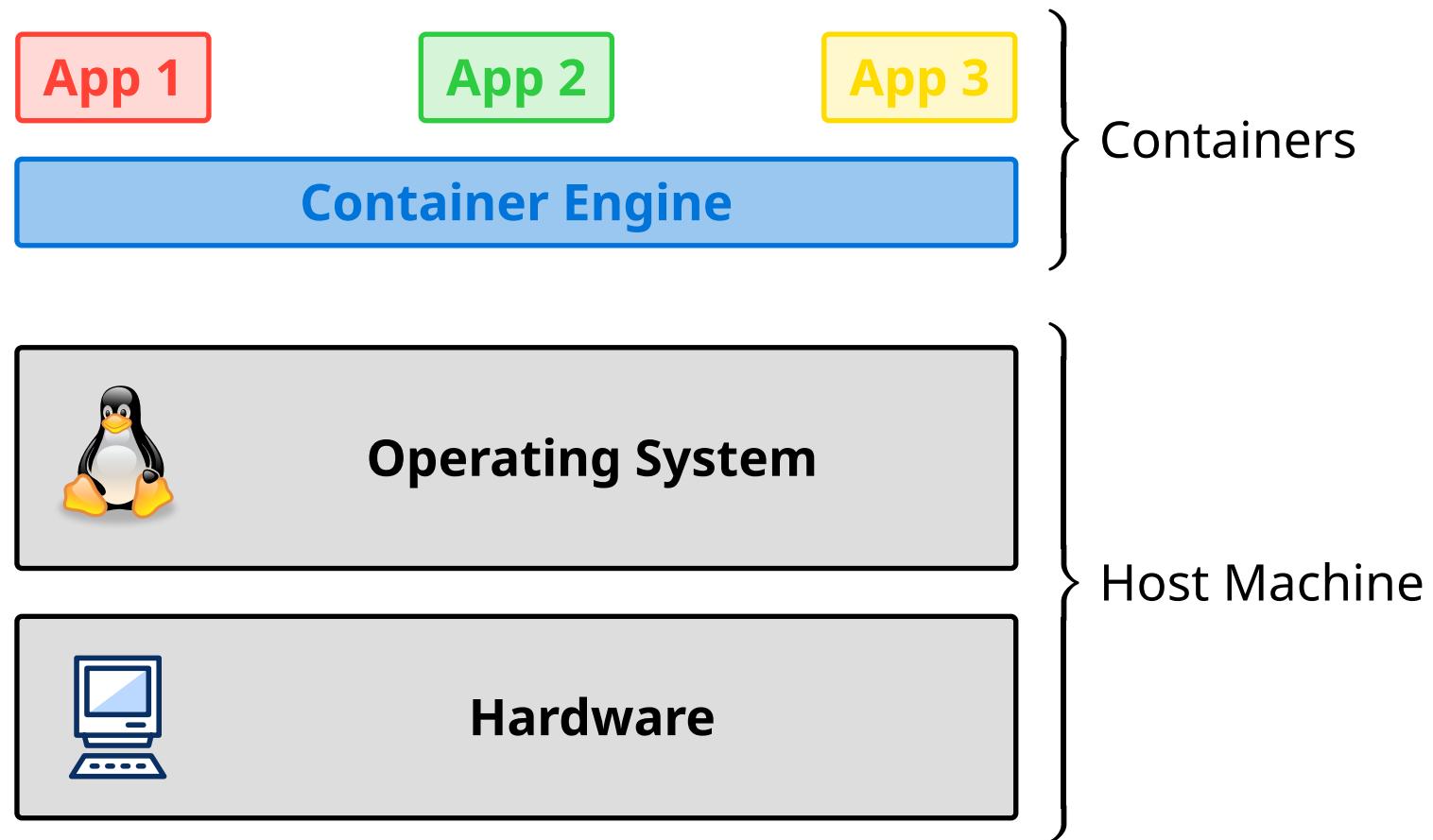
## Containerization

Docker oder Podman



# Containerization

- Erstellen von Umgebungen (Containern), die sich, unabhängig von ihrer Umgebung, immer gleich verhalten
- Ähnlich wie eine Virtuelle Maschine, allerdings ohne Virtualisierung der Hardware
- Alle Container laufen über den gleichen Kernel -> Mehr Performance als mit VMs
- Software wird in diesen Containern deployed



# Bestandteile

**Dockerfile**  
Beschreibt den  
Aufbau von einem  
Image

**Image**  
Eine Abbildung der  
Software, die  
eingesetzt werden  
soll, mit allen  
benötigten  
Abhängigkeiten.

**Container**  
Die laufende Software,  
basierend auf einem  
Image

# Dockerfile

```
1 # syntax=docker/dockerfile:1
2 FROM ubuntu:22.04
3
4 # install app dependencies
5 RUN apt-get update && apt-get install -y python3 python3-pip
6 RUN pip install flask==3.0.*
7
8 # install app
9 COPY hello.py /
10
11 # final configuration
12 ENV FLASK_APP=hello
13 EXPOSE 8000
14 CMD ["flask", "run", "--host", "0.0.0.0", "--port", "8000"]
```



# Bestandteile

**Dockerfile**  
Beschreibt den  
Aufbau von einem  
Image

**Image**  
Eine Abbildung der  
Software, die  
eingesetzt werden  
soll, mit allen  
benötigten  
Abhängigkeiten.

**Container**  
Die laufende Software,  
basierend auf einem  
Image

# Bestandteile

## Dockerfile

Beschreibt den Aufbau von einem Image

## Image

Eine Abbildung der Software, die eingesetzt werden soll, mit allen benötigten Abhängigkeiten.

## Container

Die laufende Software, basierend auf einem Image

# Containerization Tools

Zwei Optionen für Containerization:

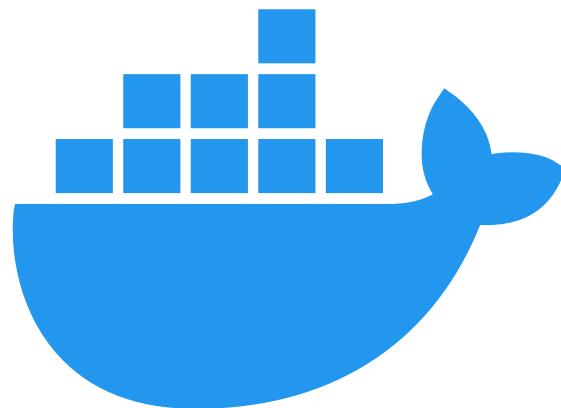


Figure 1: Docker

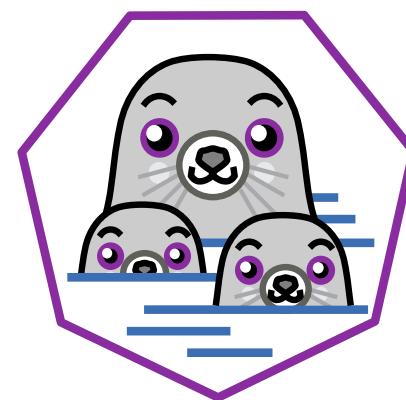


Figure 2: Podman

# Versionierung mit Git

- Versionierung aller Dateien in einem Projekt
- Mit jedem Commit wird ein Snapshot vom aktuellen Stand gemacht
- Wenn keine Änderungen an einer Datei vorgenommen wurden, wird auf den Snapshot mit der letzten Änderung verwiesen

# Zustände

## Modified

- Datei wurde geändert aber noch in die Datenbank committed

## Staged

- Modifizierte Datei wurde markiert, damit sie in den nächsten Commit-Snapshot kommt

## Committed

- Daten wurden erfolgreich in der Datenbank gespeichert

# Workflow

1. Modifizierung von Dateien
2. Nur Änderungen stagern, die Teil des nächsten Commits werden sollen
3. Ausführen eines Commits, welcher alle Änderungen in die Datenbank übernimmt, die als staged markiert wurden

# **Git - Online**

**GitHub**

**GitLab** von der TU Chemnitz: <https://gitlab.hrz.tu-chemnitz.de/>