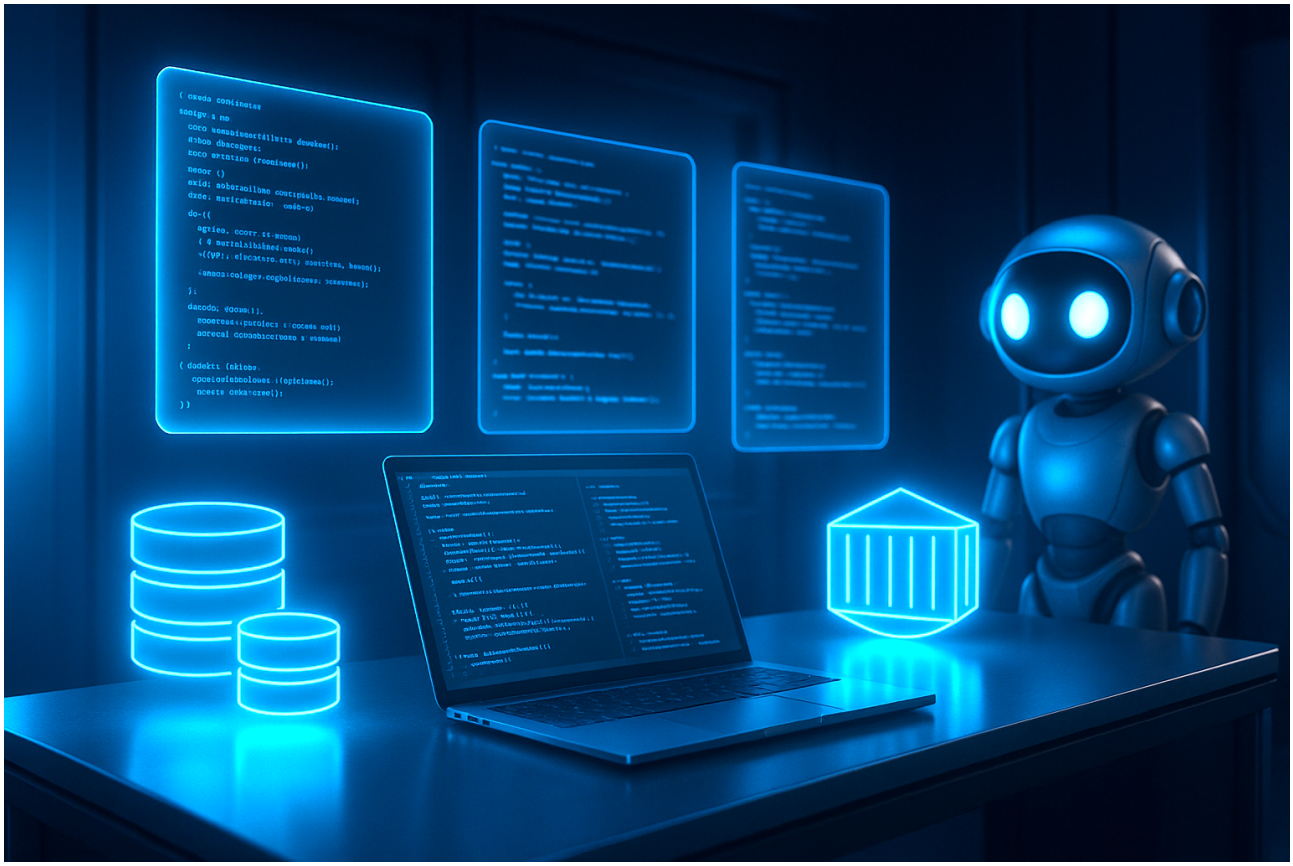
KI, Docker und MariaDB

Deine Zukunft als Entwickler

Michael Niedermair und andere, realisiert mit KI-Unterstützung



generiert mit Perplexity

1 Docker – Die Revolution in der Softwareentwicklung

Willkommen in der Welt von Docker! Wenn ihr eure Karriere als Anwendungsentwickler ernst nehmt, wird Docker nicht nur ein Werkzeug für euch sein, sondern eine Denkweise, die eure Effizienz, Kreativität und Professionalität auf ein neues Level hebt. Stellt euch eine Welt vor, in der die gefürchtete Aussage „Aber bei mir hat's funktioniert!" endgültig der Vergangenheit angehört. Genau diese Welt eröffnet euch Docker.

Was ist Docker und warum wird es eure Arbeit revolutionieren?

Im Kern ist Docker eine Plattform für die Containerisierung. Vergesst schwere, langsame virtuelle Maschinen. Docker verpackt eine Anwendung mit all ihren Abhängigkeiten – Bibliotheken, Konfigurationsdateien, Laufzeitumgebungen – in eine leichte, portable Einheit: einen Container. Dieser Container läuft isoliert vom Host-System und von anderen Containern. Das bedeutet, eine im Container laufende Applikation verhält sich auf eurem Entwicklungsnotebook exakt genauso wie auf dem Testserver oder in der finalen Produktionsumgebung in der Cloud.

Die unschlagbaren Vorteile für euch als Entwickler:

- **Absolute Konsistenz:**
  Entwickelt, testet und deployt in identischen Umgebungen. Docker eliminiert die „Matrix of Hell" aus unterschiedlichen Betriebssystem-Versionen, Patch-Levels oder installierten Bibliotheken im Team. Jeder arbeitet mit demselben Setup, was zu weniger Fehlern und massiv beschleunigten Entwicklungszyklen führt.
- **Maximale Portabilität:**
  Ein Docker-Container, der einmal erstellt wurde, läuft überall dort, wo Docker installiert ist – ohne Änderungen. Ob auf einem lokalen Windows-Rechner mit WSL 2, einem Linux-Server im Rechenzentrum oder bei einem beliebigen Cloud-Anbieter. Diese „Build once, run anywhere"-Philosophie gibt euch eine nie dagewesene Freiheit und Flexibilität.
- **Effizienz und Geschwindigkeit:**
  Container teilen sich den Kernel des Host-Betriebssystems und sind daher extrem ressourcenschonend und schnell. Das Hochfahren eines Containers dauert oft nur wenige Sekunden, im Gegensatz zu Minuten bei einer traditionellen VM. Ihr könnt komplexe Anwendungslandschaften mit mehreren Diensten (z. B. eine Webanwendung mit Datenbank und Cache) in Sekunden auf eurem eigenen Rechner starten.
- **Isolation und Sicherheit:**
  Jeder Container läuft in seiner eigenen, abgeschotteten Umgebung. Prozesse in einem Container können andere Prozesse auf dem Host nicht sehen oder beeinflussen. Dies verhindert Konflikte zwischen den Abhängigkeiten verschiedener Projekte und erhöht die Sicherheit eurer Anwendung.
- **Skalierbarkeit und Microservices:**
  Docker ist das Fundament moderner Anwendungsarchitekturen wie Microservices. Ihr könnt eure Anwendung in kleine, unabhängige Dienste zerlegen, die jeweils in einem eigenen Container leben. Diese Dienste könnt ihr unabhängig voneinander entwickeln, deployen und skalieren – ein entscheidender Vorteil für die Entwicklung komplexer und robuster Systeme.

Mit Docker werdet ihr nicht nur zu besseren Programmierern, sondern zu Architekten eurer eigenen, portablen und hocheffizienten Entwicklungswelten. Ihr übernehmt die Kontrolle über die gesamte Laufzeitumgebung und stellt sicher, dass euer Code genau so läuft, wie ihr es beabsichtigt habt – immer und überall.

2 Docker-Installationsmethoden im Vergleich

Die Wahl der richtigen Installationsmethode für Docker ist eine grundlegende Entscheidung, die den Arbeitsablauf, die Performance und die Systemstabilität maßgeblich beeinflusst. Die Ansätze unterscheiden sich je nach Host-Betriebssystem: **Windows** oder **Linux**.

2.1 Docker unter Windows

Unter Windows hat das **Windows Subsystem for Linux (WSL 2)** die Docker-Nutzung revolutioniert. Docker verwendet diese Technologie, um einen echten Linux-Kernel direkt in Windows auszuführen, was eine nahezu native Performance und Kompatibilität ermöglicht.

Methode 1: Docker Desktop für Windows (Der offizielle All-in-One-Weg)

Dies ist das offizielle, von Docker Inc. bereitgestellte „All-in-One-Sorglos-Paket“. Es handelt sich um eine Windows-Anwendung mit einer grafischen Benutzeroberfläche (dem Docker Dashboard), die im Hintergrund automatisch eine WSL 2-Umgebung verwaltet, um die Docker Engine auszuführen.

Vorteile	Nachteile
✅ **Sehr einfache Installation:** Ein einziger Download und wenige Klicks. ✅ **Grafische Benutzeroberfläche:** Das Docker Dashboard vereinfacht die Verwaltung von Containern, Images und Volumes erheblich. ✅ **Nahtlose Integration:** Perfekte Einbindung in Windows, den Explorer und IDEs wie VS Code. ✅ **Automatische Updates:** Hält sich und die Docker Engine selbstständig aktuell.	❌ **Hoher Ressourcenverbrauch:** Docker Desktop ist eine recht schwere Anwendung, die spürbar RAM und CPU beanspruchen kann. ❌ **Lizenzkosten für Unternehmen:** Für größere Unternehmen ist eine kostenpflichtige Subscription erforderlich. ❌ **Probleme mit Roaming Profiles:** Kann durch Speicherung im `AppData\Roaming`-Ordner zu extrem langen An- und Abmeldezeiten führen. ❌ **Weniger Kontrolle:** Man hat weniger direkten Einfluss auf die Konfiguration der Docker Engine. Es ist eine „Blackbox“.

Für wen ist das ideal?

Für Docker-Anfänger, Entwickler, die eine grafische Verwaltung bevorzugen, sowie für Einzelanwender und kleine Teams, die nicht von den Lizenzbedingungen betroffen sind und keine Roaming Profiles verwenden.

Methode 2: Docker Engine direkt in einer WSL~2-Distribution (Der manuelle Profi-Weg)

Bei diesem Ansatz wird die WSL~2-Distribution (z. B. Ubuntu) wie ein eigenständiger Linux-Rechner behandelt und die Docker Engine direkt dort mit dem Paketmanager `apt` installiert.

Vorteile	Nachteile
✅ **Schlank & performant:** Kein zusätzlicher GUI-Overhead, minimaler Ressourcenverbrauch. ✅ **Volle Kontrolle:** 100% Kontrolle über die Docker Engine und das zugrundeliegende Linux-System. ✅ **Keine Probleme mit Roaming Profiles:** Alle Daten bleiben innerhalb des WSL-Dateisystems. ✅ **Keine Lizenzkosten:** Verwendung der reinen Open-Source Engine. ✅ **Näher an der Produktion:** Die Arbeitsweise entspricht exakt der auf einem Linux-Server.	❌ **Keine grafische Benutzeroberfläche:** Alle Interaktionen finden auf der Kommandozeile statt. ❌ **Komplexere Einrichtung:** Erfordert mehrere manuelle Schritte (Repository, GPG-Keys, User-Gruppen, `sudoers`-Datei). ❌ **Manuelle Updates:** Man ist selbst für die Aktualisierung via `apt` verantwortlich. ❌ **Manuelles Starten des Dienstes:** Der Docker-Dienst muss oft manuell oder per Skript gestartet werden.

Für wen ist das ideal?

Für fortgeschrittene Anwender, Profis, alle Nutzer mit Roaming Profiles und Entwickler, die eine reine Linux-Terminal-Erfahrung mit maximaler Performance und Kontrolle wünschen.

2.2 Docker unter Linux

Auf einem „echten“ Linux-System ist Docker zu Hause. Hier läuft es nativ ohne zusätzliche Virtualisierungsschicht.

Methode 1: Docker Engine (Standalone / Nativ)

Dies ist der Standardweg und die ursprüngliche Form von Docker. Die Docker Engine wird als Systemdienst direkt auf dem Linux-Host installiert.

Vorteile	Nachteile
✅ **Maximale Performance:** Keine Virtualisierung, Docker läuft mit nativer Geschwindigkeit. ✅ **Extrem schlank:** Minimaler Ressourcen-Overhead. ✅ **Standard für die Produktion:** Nahezu alle Server-Installationen weltweit nutzen diese Methode. ✅ **Vollständig Open Source:** Keine Lizenzkosten.	❌ **Kommandozeilen-basiert:** Standardmäßig gibt es keine offizielle GUI. ❌ **Sicherheitsaspekt:** Der Docker Daemon läuft mit `root`-Rechten, was bei unachtsamer Konfiguration ein potenzielles Risiko darstellt. ❌ **Manuelle Einrichtung:** Erfordert mehrere Terminal-Befehle.

Für wen ist das ideal?

Für **alle** Linux-Benutzer, vom Desktop-Entwickler bis zum Server-Administrator. Dies ist der de-facto Standard und in 99% der Fälle die richtige Wahl unter Linux.

Methode 2: Docker Desktop für Linux

Dies ist eine Anwendung, die eine grafische Oberfläche bereitstellt und Docker in einer eigenen, leichtgewichtigen virtuellen Maschine ausführt, konzeptionell identisch zur Windows-Version.

Vorteile	Nachteile
✅ **Grafische Benutzeroberfläche:** Bietet das bekannte Docker Dashboard. ✅ **Konsistente Erfahrung:** Gleiche Oberfläche auf Windows, Mac und Linux. ✅ **Zusatzfunktionen:** Einfacher Zugriff auf Erweiterungen (Extensions).	❌ **Unnötige Abstraktion:** Fügt eine Virtualisierungsschicht hinzu, wo keine nötig wäre, was zu leichtem Performance-Overhead führt. ❌ **Höherer Ressourcenverbrauch:** Benötigt mehr RAM und CPU als die native Engine. ❌ **Lizenzkosten für Unternehmen:** Die gleichen Lizenzbedingungen wie bei der Windows-Version gelten auch hier.

Für wen ist das ideal?

Für Linux-Einsteiger, die von der Kommandozeile abgeschreckt sind, oder für Entwickler in heterogenen Teams, die eine auf allen Betriebssystemen identische Arbeitsumgebung wünschen.

Fazit und Empfehlung für den Unterricht

- **Für Windows-Nutzer:** Beginnt mit **Docker Desktop** für den einfachsten Einstieg. Bei Problemen (z. B. Roaming Profiles) oder für den nächsten Karriereschritt ist der Wechsel zur **manuellen Installation in WSL~2** der Weg zum Profi.
- **Für Linux-Nutzer:** Verwendet die **native Docker Engine**. Sie ist performanter, schlanker und der Industriestandard. Nur wer zwingend eine GUI benötigt, sollte Docker Desktop als Option in Betracht ziehen.

Meine Wahl

Ich verwende primär Linux, daher war meine Wahl „Docker direkt unter Linux“.
In der Schule arbeite ich mit Windows (Roaming Profiles), daher habe ich dort mit WSL~2 ein Ubuntu installiert und darin verwende ich Docker. Dadurch muss ich mich bei den zwei System nicht umstellen.

2.3 Installation – Der Startschuss für eure Docker-Reise

Für Windows

Unter Windows ist die empfohlene Methode die Installation von **Docker Desktop**, welches WSL~2 im Hintergrund nutzt.

- Docker Desktop für Windows (offizieller Download & Anleitung)
  - **Link:** https://docs.docker.com/desktop/install/windows-install/
  - **Beschreibung:**
    Dies ist die Hauptseite für die Installation. Ihr findet hier den direkten Download-Button für die ‘.exe‘-Datei sowie die Systemanforderungen und eine detaillierte Schritt-für-Schritt-Anleitung.

Für Linux

Unter Linux (und in Ihrer WSL-Umgebung) ist die empfohlene Methode die manuelle Installation der **Docker Engine** über den Paketmanager.

- Docker Engine für Ubuntu (offizielle Anleitung)
  - **Link:** small https://docs.docker.com/engine/install/ubuntu/
  - **Beschreibung:**
    Diese Seite enthält die vollständige Anleitung für die Kommandozeile. Sie führt Euch durch den Prozess, das offizielle Docker-Repository zu Eurem System hinzuzufügen und Docker dann mit ‘apt-get‘ zu installieren.
- Übersichtsseite für andere Linux-Distributionen
  - **Link:** https://docs.docker.com/engine/install/
  - **Beschreibung:**
    Falls Ihr eine andere Linux-Distribution als Ubuntu verwenden (z. B.. Debian, Fedora, CentOS), ist dies die perfekte Startseite. Ihr könnt hier eure spezifische Distribution auswählen, um die passende Installationsanleitung zu erhalten.

Diese Links sind die „Single Source of Truth“ und sollten immer die aktuellsten und sichersten Installationsmethoden enthalten.

3 Stateful Applications – MariaDB im Container-Alltag

Bisher haben wir uns mit der Theorie und der Installation beschäftigt. Jetzt wird es praktisch. Eine der häufigsten Aufgaben für Anwendungsentwickler ist die Interaktion mit einer Datenbank. Wir werden nun gemeinsam eine MariaDB-Datenbank – eine der populärsten Open-Source-Datenbanken der Welt – professionell als Docker-Container aufsetzen.

Dabei lernt ihr das wichtigste Konzept für den Betrieb von Anwendungen, die Daten speichern (sog. „stateful applications“): Datenpersistenz. Ein Container ist per Definition flüchtig. Wir müssen Docker anweisen, die wertvollen Daten unserer Datenbank an einem sicheren, dauerhaften Ort abzulegen.

3.1 Standardbefehle

Befehl	Beschreibung
`docker run`	Startet einen Container
`docker ps`	Zeigt laufende Container
`docker stop`	Stoppt einen Container
`docker rm`	Löscht einen Container
`docker logs`	Zeigt die Logs eines Containers
`docker images`	Listet vorhandene Images
`docker rmi`	Löscht ein Image
`docker exec`	Befehl im laufenden Container ausführen

3.2 Der Container-Start: Standard vs. Angepasst

Wir betrachten zwei gängige Szenarien: den einfachen Standardfall und den flexiblen, angepassten Fall für spezifische Anforderungen.

Szenario 1: Der Standardfall (Empfohlen für den Einstieg)

Dieser Ansatz verwendet den Standard-Port und ein von Docker verwaltetes Volume. Er ist sauber, plattformunabhängig und robust.

Installation

```
docker run -d
   --name meine-mariadb-standard
   -p 3306:3306
   -e MARIADB_ROOT_PASSWORD="mein-sicheres-passwort"
   -v mariadb-standard-data:/var/lib/mysql mariadb:latest
```

Wenn Ihr beim Ausführen Zeilenumbrücke verwenden wollt, dann wüsst Ihr diese „Markieren“. Unter Linux mit der Bash „ `\`“ und unter Windows bei CMD mit „ `^`“

- `--name meine-mariadb-standard`: Ein eindeutiger Name für diesen Container.
- `-p 3306:3306`: Das Standard-Mapping. Der Host lauscht auf Port 3306 und leitet an den Container-Port 3306 weiter.
- `-v mariadb-standard-data:/var/lib/mysql`: Die empfohlene Methode. `mariadb-standard-data` ist ein Docker-Managed Volume. Docker kümmert sich um die Speicherung. `/var/lib/mysql` ist der feste Datenpfad innerhalb des Containers.

Szenario 2: Der flexible Fall (Für Fortgeschrittene und spezielle Setups)

Manchmal müsst ihr von den Standards abweichen. Gründe dafür sind:

- Der Port 3306 ist auf eurem Host bereits belegt.
- Ihr wollt mehrere MariaDB-Instanzen parallel betreiben (z. B. um vertikale Lastverteilung zu simulieren oder um getrennte Umgebungen für verschiedene Projekte zu haben).
- Ihr wollt die Datenbankdateien an einem ganz bestimmten Ort auf eurer Festplatte ablegen (z. B. auf einem schnellen SSD-Laufwerk).

Hier ist ein Beispiel, das einen anderen Host-Port und ein festes Verzeichnis auf dem Host-System (ein Bind Mount) verwendet:

Installation

```
docker run -d
    --name meine-mariadb-flexibel
    -p 3307:3306
    -e MARIADB_ROOT_PASSWORD="ein-anderes-passwort"
    -v E:\daten\mariadb:/var/lib/mysql
    mariadb:latest
```

Analysieren wir die Unterschiede:

- `--name meine-mariadb-flexibel`: Ein anderer Name, um die Container klar zu trennen.
- `-p 3307:3306`: Das flexible Port-Mapping im Format HOST-PORT:CONTAINER-PORT. Euer Host-Rechner lauscht nun auf Port 3307 und leitet den Verkehr an den Port 3306 im Container weiter. Ihr könntet einen dritten Container mit `-p 3308:3306` starten und so weiter.
- `-v E:\ daten\ mariadb:/var/lib/mysql`: Dies ist ein Bind Mount. Ihr koppelt ein festes Verzeichnis eures Host-Systems direkt an das Datenverzeichnis im Container.
- Unter Windows: Gebt den vollen Pfad an, inklusive Laufwerksbuchstaben. Das Beispiel verwendet E:\ daten\ mariadb. Stellt sicher, dass dieses Verzeichnis existiert.
- Unter Linux: Die Syntax ist äquivalent, z. B. `-v /mnt/ssd/mariadb_data:/var/lib/mysql`.

3.3 Verbindung zur Datenbank: Zwei Wege zum Ziel

Frage: Muss ich einen Client, z. B. `mariadb` auf meinem Host installieren?
Antwort: Nicht zwingend!

Weg 1: Der „Docker-Way“ (ohne externen Client)

Ihr könnt eine Shell direkt im laufenden Container öffnen. Gebt einfach den Namen des gewünschten Containers an:

```
# Verbindung zum Standard-Container
docker exec -it meine-mariadb-standard mariadb -u root -p

# Verbindung zum flexiblen Container
docker exec -it meine-mariadb-flexibel mariadb -u root -p
```

Beispiel

Schauen wir uns zuerst an, welche Container laufen (verkürzte Ausgabe).

```
$ docker ps
CONTAINER ID   IMAGE            COMMAND                  PORTS                     NAMES
25f6ddbea830   ollama/ollama    "/bin/ollama serve"      0.0.0.0:11434->11434/tcp ollama
29499cd632fa   mariadb:latest   "docker-entrypoint.…s"   0.0.0.0:3306->3306/tcp   mymariadb
```

Ich habe zwei Container:

- **ollama**
  Mein „kleines“ KI-System.
- **mymariadb**
  Meine Datenbank für die KI-Anbindung.

Verbindung aufbauen mit WSL Ubuntu

```
$ docker exec -it mymariadb mariadb -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 3
Server version: 12.0.2-MariaDB-ubu2404 mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

Verbindung aufbauen in Windows mit CMD

```
c:\> wsl  docker exec -it mymariadb mariadb -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 4
Server version: 12.0.2-MariaDB-ubu2404 mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

Weg 2: Der klassische Weg (mit externem Client)

Wenn ihr einen grafischen Client (DBeaver, DataGrip, …) oder z. B. Java verwendet, müsst ihr nun die korrekten Verbindungsdaten für den jeweiligen Container nutzen:

	„meine-mariadb-standard“	„meine-mariadb-flexibel”
Host	127.0.0.1 (oder localhost)	Host: 127.0.0.1 (oder localhost)
Port	3306	3307
Benutzer	root	root
Passwort	mein-sicheres-passwort	ein-anderes-passwort

Verbindungsausbau mit Java:

```
db.url=jdbc:mariadb://localhost:3306/dbagent
db.user=dbagent
db.password=dbagent
```

3.4 ’localhost’ ist nicht gleich ’localhost’

Schauen wir uns an, wie wir den User `dbagent` in der Datenbank definiert haben.

Auszug: create_db.sql

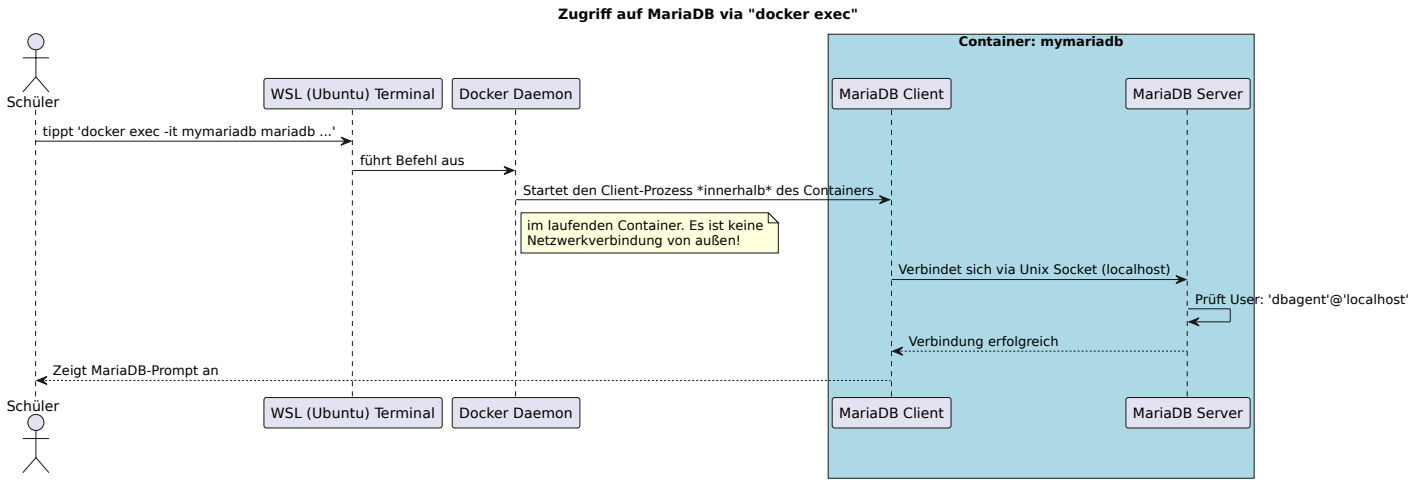
```
-- Erstellt einen neue User, falls diese noch existiert, vorher löschen.
-- User 'dbagent', PW ist leer
DROP USER IF EXISTS 'dbagent'@'localhost';
CREATE USER 'dbagent'@'localhost' IDENTIFIED BY '';

-- Rechte vergeben.
GRANT ALL PRIVILEGES ON dbagent.* TO 'dbagent'@'localhost';
```

Der User `dbagent` darf von `localhost` auf die Datenbank zugreifen. Von allen anderen „Seiten“ wird der Zugriff verweigert.

Schaubild 1: Direkter Zugriff über die WSL-Konsole

Dieses Diagramm zeigt, wie der Befehl `docker exec` funktioniert. Er agiert wie eine „besondere Hintertür“ direkt in den Container, weshalb der Zugriff als `localhost` gewertet wird.

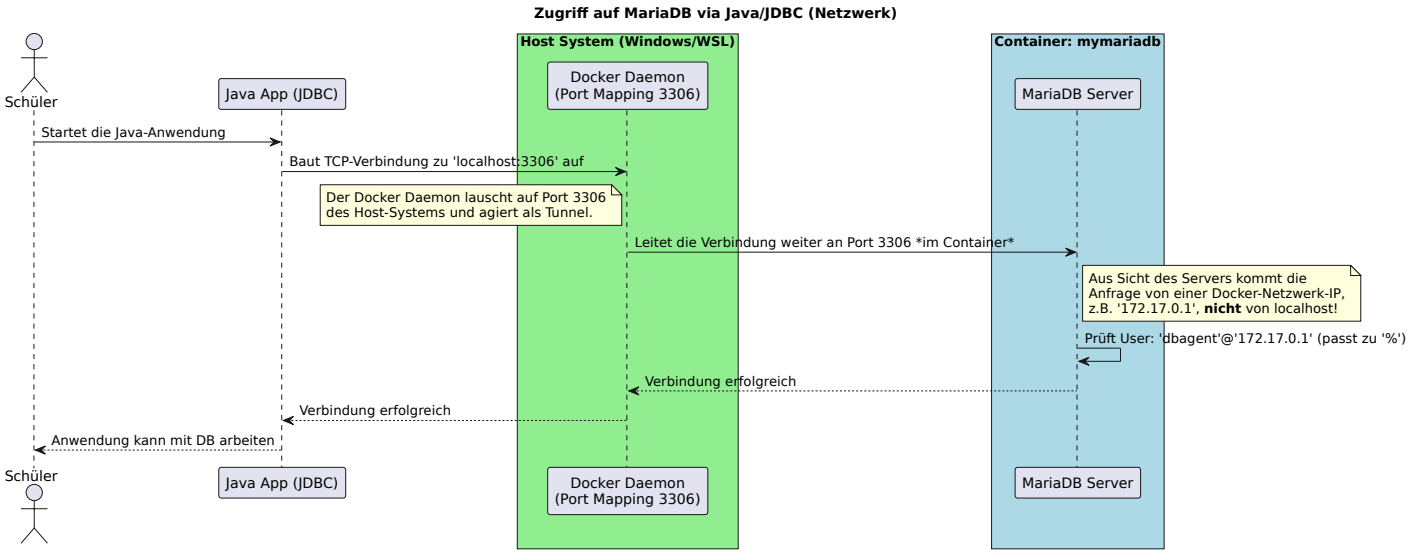


Erklärung:

- Schritt 1 & 2:
  Ihr als Entwickler gebt den Befehl `docker exec ...` in euer WSL-Terminal ein. Dieser Befehl geht an den Docker-Dienst (den Daemon).
- Schritt 3 (Der entscheidende Punkt):
  Der Docker Daemon startet den mariadb-Client nicht auf eurem Rechner, sondern direkt im abgeschlossenen Raum des mymariadb-Containers.
- Schritt 4 & 5:
  Weil der Client und der Server nun im selben „Raum“ (Container) sind, unterhalten sie sich über den direktesten Weg – einen sogenannten „Unix Socket“. Aus Sicht des Servers kommt diese Anfrage von `localhost`. Deshalb prüft er die Berechtigungen für den User `'dbagent'@'localhost'`.
- **Fazit:**
  Dieser Zugriffspfad testet nur, ob ein User sich innerhalb des Containers selbst anmelden darf.

Schaubild 2: Zugriff über Java/JDBC vom Host-System

Dieses Diagramm zeigt den typischen Anwendungsfall. Die Anwendung läuft „außen“ und greift über das Netzwerk auf den Container zu. Hier wird das Port-Mapping (`-p 3306:3306`) entscheidend.



Erklärung:

- Schritt 1 & 2:
  Eure Java-Anwendung startet auf dem Host-System (also außerhalb des Containers). Gemäß der Konfiguration (`jdbc:mariadb://localhost:3306/...`) versucht sie, eine Netzwerkverbindung zu `localhost` auf Port 3306 aufzubauen.
- Schritt 3 (Der entscheidende Punkt):
  Der Docker Daemon fängt diese Anfrage ab, weil wir beim Starten des Containers mit `-p 3306:3306` einen „Tunnel“ eingerichtet haben. Er leitet den gesamten Netzwerkverkehr von Port 3306 des Hosts an Port 3306 im Container weiter.
- Schritt 4 & 5:
  Der MariaDB-Server im Container empfängt die Anfrage. Für ihn kommt diese Verbindung aber nicht von `localhost`, sondern aus dem internen Docker-Netzwerk. Die Quell-IP ist die des Docker-Gateways (z. B. `172.17.0.1`). Deshalb muss der User die Berechtigung haben, sich von jedem Host (%) oder explizit von dieser IP aus zu verbinden. Er prüft also die Anmeldedaten für `'dbagent'@'%'`. Fazit: Dieser Zugriffspfad simuliert, wie eine reale Anwendung (z.B. ein Web-Backend) auf eine Datenbank zugreifen würde.

Lösung:

Die Zugriffsrechte müssen angepasst werden.

Auszug: create_db.sql

```
-- Erstellt einen neue User, falls diese noch existiert, vorher löschen.
-- User 'dbagent', PW ist 'dbagent'
DROP USER IF EXISTS 'dbagent'@'%';
CREATE USER 'dbagent'@'%' IDENTIFIED BY 'dbagent';

-- Rechte vergeben.
GRANT ALL PRIVILEGES ON dbagent.* TO 'dbagent'@'%';
```

3.5 Update-Prozess: Der Wegwerf-Container

In der Docker-Welt „patcht“ man keine laufenden Container. Man ersetzt sie. Dank persistenter Speicherung bleiben die Daten unangetastet. Der Prozess ist für beide Szenarien identisch, ihr müsst nur den korrekten Container-Namen und run-Befehl verwenden.

1. Neues Image holen: docker pull mariadb:latest
2. Alten Container stoppen und entfernen:
   `docker stop <container-name>` und `docker rm <container-name>`
3. Neuen Container mit dem alten Speicher starten: Führt exakt denselben docker run-Befehl, den ihr für die Erstellung verwendet habt, erneut aus. Docker verbindet den neuen Container automatisch mit dem bestehenden Volume oder Bind Mount.

3.6 Aufräumen: Die gezielte Löschung

- **Nur den Container löschen:**
  `docker stop <container-name>` und `docker rm <container-name>`
  Die Daten im Volume oder Bind-Mount-Verzeichnis bleiben erhalten.
- **Container UND Daten löschen (ALLES WEG!)**:
  1. Stoppt und löscht den Container wie oben.
  2. Bei einem Docker-Managed Volume:
     Löscht das Volume mit docker volume `rm mariadb-standard-data`.
  3. Bei einem Bind Mount:
     Löscht das Verzeichnis auf eurem Host-System manuell
     (z. B. den Ordner `E:\daten\mariadb`).

3.7 WSL und Docker „sauber“ beenden

Damit WSL und Docker sauber beendet werden, sollte man folgende Schritte vor dem „Herunterfahren“ machen. Packt man „das Ganze“ in ein Skript, so kann man auch den Rechner gleich ausschalten.

- Stopping all running Docker containers inside WSL Ubuntu…
  `wsl.exe docker stop $(docker ps -q)`
- Stopping all running Docker containers inside Windows…
  `FOR /f "tokens=*" %%i IN ('docker ps -q') DO docker stop %%i`
- Stopping the Docker service…
  `wsl.exe sudo service docker stop`
- Syncing filesystem buffers to disk…
  `wsl.exe sync`
- Shutting down the WSL system…
  `wsl.exe --shutdown`
- Shotdown Windows...
  `shutdown /s /f /t 0`

WSL Ubuntu

```
@echo off
echo Stopping all running Docker containers inside WSL Ubuntu
wsl.exe docker stop $(docker ps -q)

echo Stopping the Docker service
wsl.exe sudo service docker stop

echo Syncing filesystem buffers to disk
wsl.exe sync

echo Shutting down the WSL system
wsl.exe --shutdown

echo Shotdown Windows
shutdown /s /f /t 0
```

Windows

```
@echo off
echo Stopping all running Docker containers inside Windows
FOR /f "tokens=*" %%i IN ('docker ps -q') DO docker stop %%i

echo Stopping the Docker service
wsl.exe sudo service docker stop

echo Syncing filesystem buffers to disk
wsl.exe sync

echo Shutting down the WSL system
wsl.exe --shutdown

echo Shotdown Windows
shutdown /s /f /t 0
```