Terminalmanagementsystem (TMS)

Docker-Installationsanleitung

Version 1.0

Cherry Digital Health GmbH
Cherrystraße 2
D-91275 Auerbach/OPf.

Historie

Version	Datum	Beschreibung
0.1	23.04.2024	Initiale Version
0.2	16.05.2025	Neue Kapitel hinzugefügt, Formulierungen ausgebessert
0.3	20.05.2025	Kleine Korrekturen und Layoutanpassungen
0.4	01.08.2025	Updateroutine hinzugefügt
0.5	04.11.2025	Überarbeitete Fassung nach Korrekturlesen
1.0	22.12.2025	Kapitel 3.2 wurde korrigiert Redundante Inhalte (doppelte Virtualisierung) wurden entfernt

Inhaltsverzeichnis

1. Einleitung

Diese Anleitung beschreibt die Installation und Inbetriebnahme der Applikation Terminalmanagementsystem (TMS) von CHERRY zur Verwaltung und Administration von Kartenterminals mittels Docker. Sie richtet sich an Administratoren, die alle Kartenterminals (KT) wie das CHERRY eHealth Terminal ST-1506 zentral verwalten möchten.

Durch den Einsatz von Docker wird die Installation des TMS deutlich vereinfacht, da alle notwendigen Komponenten in Containern bereitgestellt werden. Dadurch entfällt die aufwendige manuelle Einrichtung.

Die für die Installation benötigte Datei docker-compose.yml kann direkt über das Download-Center von CHERRY heruntergeladen werden:

https://download.cherry-service.de/downloadcenter/

Diese Datei enthält bereits alle notwendigen Konfigurationswerte, um die Anwendung schnell und unkompliziert starten zu können.

Ziel dieser Anleitung ist es, eine verständliche und nachvollziehbare Schritt-für-Schritt-Beschreibung bereitzustellen, die es jedem ermöglicht, die Anwendung unabhängig von technischen Vorkenntnissen erfolgreich zu installieren. Die Anleitung führt systematisch durch alle erforderlichen Installations- und ggf. Konfigurationsschritte.

Die Entscheidung für Docker als Plattform basiert auf mehreren Vorteilen:

- Docker ermöglicht schnelle Software-Deployments, erleichtert die Installation und sorgt für eine vereinfachte Wartbarkeit der Anwendung. Zudem ist eine klare Nachvollziehbarkeit von Updates und Fehlern gewährleistet, da jede Komponente in einem isolierten Container läuft.
- Die Container-Technologie erlaubt es, Anwendungen plattformunabhängig zu betreiben, da alle Abhängigkeiten, Bibliotheken und Konfigurationen direkt im Container integriert sind. Über die Docker-Compose-Datei können Umgebungsvariablen für die Anwendung definiert und genutzt werden. Fehler lassen sich dadurch präzise nachvollziehen und beheben, ohne dass es zu Konflikten mit der Host-Umgebung kommt.
- Docker ermöglicht die Ausführung von Anwendungen in Containern, die alle notwendigen Komponenten beinhalten. Diese Container sind isolierte Umgebungen, in denen die Software läuft, ohne dass umfangreiche manuelle Konfigurationen erforderlich sind.

Docker kann unter verschiedenen Betriebssystemen genutzt werden:

1. **Windows**: Ab Version 4.41.0.
   Hierbei ist wichtig, dass Docker Desktop die Virtualisierungstechnologie WSL2 (Windows Subsystem for Linux 2) nutzt. WSL2 wird im Rahmen der Docker-Desktop-Installation automatisch vorgeschlagen und kann direkt mitinstalliert werden.
   Eine manuelle Installation ist in der Regel nicht notwendig. Hyper-V hingegen wird nicht unterstützt, da es nachweislich zu Problemen in der UDP-Kommunikation mit den Kartenterminals und der Kommunikation aus Containern führt. Um eine stabile und zuverlässige Verbindung zu gewährleisten, wird in der Konstellation mit Docker Desktop ab Version 4.41.0 ausschließlich WSL2 unterstützt. Ältere Versionen von Docker Desktop werden ebenfalls nicht mehr unterstützt.

2. **Linux**: Ab Version 4.38.0.
   Docker läuft nativ auf Linux-Systemen, was eine direkte Kommunikation mit dem Kernel erlaubt.

Hinweis zur doppelten Virtualisierung:
Eine doppelte Virtualisierung entsteht, wenn Docker innerhalb einer bereits virtualisierten Umgebung (z. B. VMware oder VirtualBox) betrieben wird. In diesem Fall läuft Docker nicht direkt auf dem Host-Betriebssystem, sondern in einer virtuellen Maschine (VM). Innerhalb dieser VM wird nochmals eine Docker-Umgebung gestartet. Diese Konstellation führt häufig zu Problemen mit der Netzwerkkommunikation. Aus diesem Grund wird die Nutzung des TMS in Docker innerhalb einer doppelten Virtualisierung vorerst nicht unterstützt.

Zusammenfassung:

- Docker läuft stabil auf Windows und Linux in den angegebenen Versionen.
- In der Konstellation mit Docker Desktop ab 4.41.0 wird ausschließlich WSL2 unterstützt.
- Docker Desktop läuft nicht mit Hyper-V, da es zu UDP-Problemen mit den Kartenterminals und der Container-Kommunikation führt.
- Doppelte Virtualisierung (VM innerhalb einer VM) wird nicht unterstützt.

Ein weiterer zentraler Bestandteil der Docker-Integration ist die Datenbankanbindung. Das System unterstützt sowohl Microsoft SQL Server 2022 als auch PostgreSQL 15. Welche Datenbank genutzt wird, entscheidet sich anhand des definierten Connection-Strings in der Datei docker-compose.yml. Die Anwendung erkennt automatisch, ob es sich um einen SQL- Server oder eine PostgreSQL-Datenbank handelt und stellt die Verbindungen entsprechend her. Eine Anpassung der Verbindungsparameter kann jederzeit über die docker-compose.yml erfolgen.

Grundsätzlich werden mit Docker folgende zentrale Komponenten als eigenständige Container gestartet:

- **TMS:**
  Das zentrale Verwaltungssystem für die Kartenterminals und somit Herzstück der Anwendung. Dieser Container umfasst noch folgende Komponenten:
  - **ASP.NET 9 Umgebung**
  - **integrierten Kestrel-Webserver**

- **postgres_db**:
  Ein eigenständiger PostgreSQL 15-Datenbankcontainer zur Speicherung der Applikationsdaten
- **pgadmin4**:
  Ein grafisches Oberflächentool (pgAdmin4), um sich mit der PostgreSQL-Datenbank zu verbinden
- **sql_server2022**:
  Ein eigenständiger Microsoft SQL Server 2022-Datenbankcontainer. Der Zugriff ist von außen über Microsoft SQL Server Management Studio möglich.

In den folgenden Kapiteln wird detailliert beschrieben, wie die Vorbereitung, Installation und Inbetriebnahme des TMS mit Docker erfolgt.

2. Vorbereitung

2.1. Entscheidung für Betriebssystem und Umgebung

Damit Docker Desktop problemlos funktioniert, müssen bestimmte Systemvoraussetzungen erfüllt sein. Dies betrifft vor allem die benötige Hardware sowie spezielle Einstellungen.

Allgemeine Rahmenbedingungen und Hardware-Anforderungen:

- **Internetverbindung:** Die Software wird aus dem Docker Hub-Repo geladen
- **Adminrechte:** Sind zwingend notwendig auf dem Host-PC für die Installation und Inbetriebnahme
- **CPU**: Mindestens 4 Kerne
- **Arbeitsspeicher (RAM)**: Mindestens 2 GB, besser 4 GB
- **Speicherplatz**: Mindestens 4 GB freier Speicher
- **Unterstützte Betriebssysteme und Docker-Versionen:**

Betriebssystem	Docker-Desktop-Version	Bemerkungen
Windows	>= 4.41.0	i. O.
Linux (Ubuntu >=22.04)	>= 4.38.0	i. O.
Doppelte Virtualisierung VM + Linux	>= 4.38.0	noch nicht vollständig getestet und geprüft
Doppelte Virtualisierung VM + Windows	>= 4.41.0	noch nicht vollständig getestet und geprüft

Spezielle Hinweise für Windows:

Docker Desktop ab der Version 4.41.0 verwendet als Virtualisierungsumgebung ausschließlich WSL2 (Windows Subsystem for Linux 2). Diese Umgebung ist notwendig, um die Anwendung TMS stabil und performant zu betreiben.

Hyper-V wird in dieser Version nicht unterstützt, da es zu UDP-Kommunikationsproblemen mit den Kartenterminals und der Netzwerkkommunikation innerhalb der Container geführt hat. Diese Probleme verursachten Verbindungsabbrüche, weshalb WSL2 als stabile Alternative implementiert wurde.

Die Installation von WSL2 wird während der Docker-Desktop-Installation ab Version 4.41.0 automatisch vorgeschlagen. Eine manuelle Installation ist in der Regel nicht notwendig, kann jedoch über die Windows-Features aktiviert werden. Docker Desktop erkennt nach der Installation WSL2 und bindet es automatisch ein. Hyper-V bleibt hierbei vollständig deaktiviert. Die TMS-Anwendung läuft somit ausschließlich über WSL2, da nur hier eine stabile und zuverlässige Verbindung gewährleistet ist.

2.2. Auswahl einer Datenbank

Dieser Abschnitt beschreibt, welche Datenbanken für das Terminalmanagementsystem verwendet werden können, wie diese innerhalb der Docker-Container integriert sind und wie der Zugriff auf die Datenbanken erfolgt. Die Auswahl der Datenbank erfolgt direkt über die Konfiguration in der docker-compose.yml. Abhängig vom eingestellten Connection-String erkennt die Anwendung, ob eine Verbindung zu PostgreSQL 15 oder zu Microsoft SQL Server 2022 hergestellt werden soll.

Bevor Sie sich für PostgreSQL und SQL Server entscheiden, sollten Sie anhand der nachfolgenden Datentabelle die Unterschiede vergleichen. Diese Tabelle soll Ihnen als Entscheidungshilfe dienen, um die für ihre Anwendung passende Datenbank auszuwählen.

2.2.1. Verfügbare Datenbanken im Vergleich

Containername	postgres-db	SQL Server
Softwareversion	PostgreSQL 15	Microsoft SQL Server 2022
Verwaltungstool	pgAdmin4 (im Container pgadmin4)	Microsoft SQL Server Management Studio (SSMS) auf dem Host-PC
Kosten	Open Source, kostenlos nutzbar	Wir nutzen ausschließlich die Express-Edition. Die anderen Ausprägungen sind kostenpflichtig.
Administration	Nur über pgAdmin4 im Container pgadmin4	Direkt über SSMS auf dem Host-PC erreichbar
Komplexität	Einfach zu konfigurieren	Erfordert mehr Ressourcen als PostgreSQL, bietet aber umfassendere Verwaltungsfunktionen
Backup und Restore	Über pgAdmin oder Kommandozeile (komplizierter)	Integriert in SSMS mit grafischer Oberfläche, komfortable Backup-Strategien

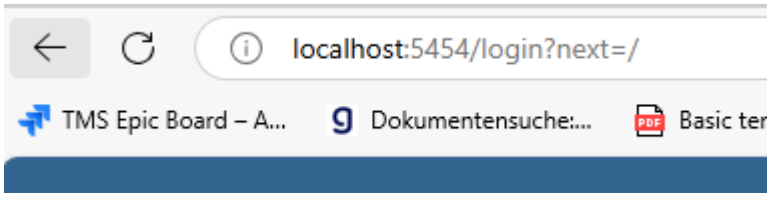
2.2.2. Zugriffsmöglichkeiten der Datenbanken

2.2.2.1. Zugriff über Container pgadmin4

PostgreSQL benötigt für die Verwaltung zwingend pgAdmin4, welches als eigener Container (pgadmin4) läuft. Hierüber können Datenbankstrukturen eingesehen, Abfragen erstellt und Daten verwaltet werden. Nachfolgend finden Sie eine Schritt-für-Schritt-Anleitung, wie Sie sich in die PostgreSQL-Datenbank über den Container pgadmin4 einloggen und eine Verbindung herstellen können.

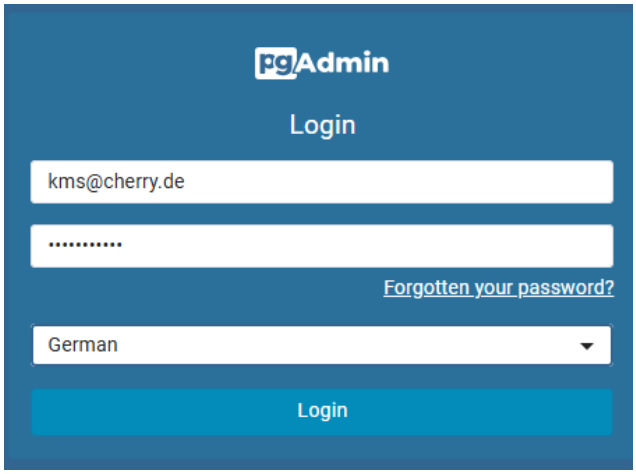
Voraussetzung: Die entsprechenden Container pgAdmin4 und postgres sind vorab erstellt und gestartet worden.

1. Öffnen Sie ihren Webbrowser und geben Sie die folgende Adresse ein: http://localhost:5454:



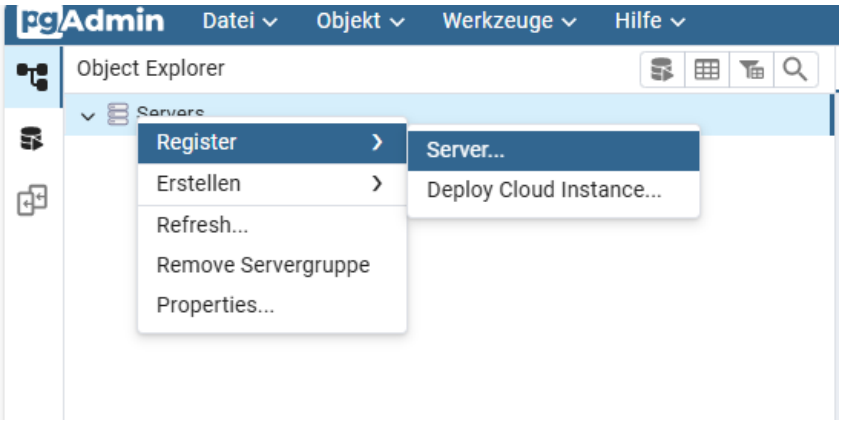
2. Melden Sie sich mit den Zugangsdaten an, die in der docker-composer.yml unter der Rubrik „pgadmin" definiert sind:

```
pgadmin:
 image: dpage/pgadmin4
 container_name: pgadmin4
 ports:
   - 5454:5454/tcp
 hostname: pgadmin
 environment:
   - POSTGRES_HOST_AUTH_METHOD=trust
   - PGADMIN_DEFAULT_EMAIL=kms@cherry.de
   - PGADMIN_DEFAULT_PASSWORD=Cherry2025!
```

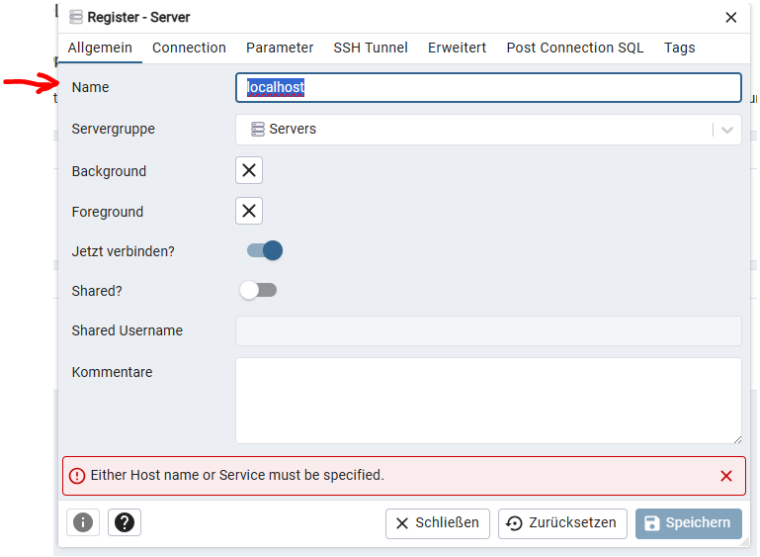


3. Klicken Sie nach erfolgreichem Login im linken Menü auf die Auswahl „Servers".

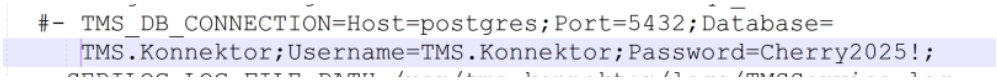
4. Machen Sie einen Rechtsklick auf „Servers“ und wählen Sie „Register“ und dann „Server“ aus:



5. Tragen Sie im Reiter „Allgemein” unter „Name“ die Bezeichnung „localhost“ ein:

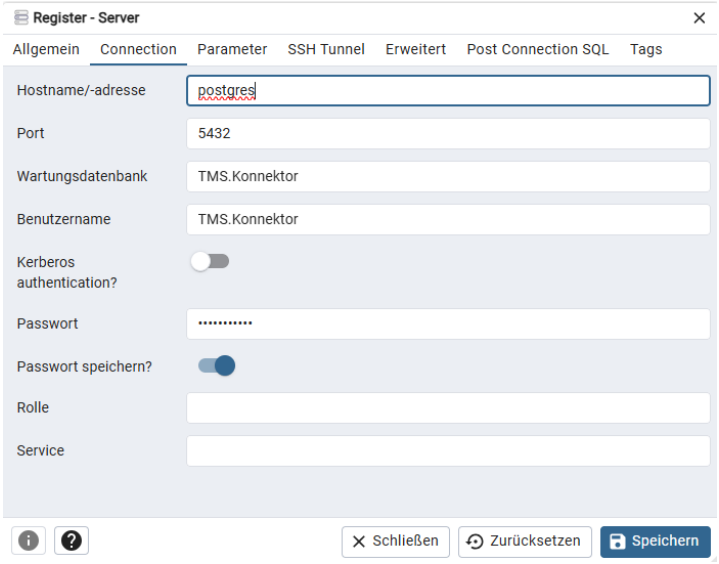


6. Wechseln Sie zum Reiter „Connection“.
7. Übernehmen Sie die Verbindungsdaten aus der docker-composer.yml:

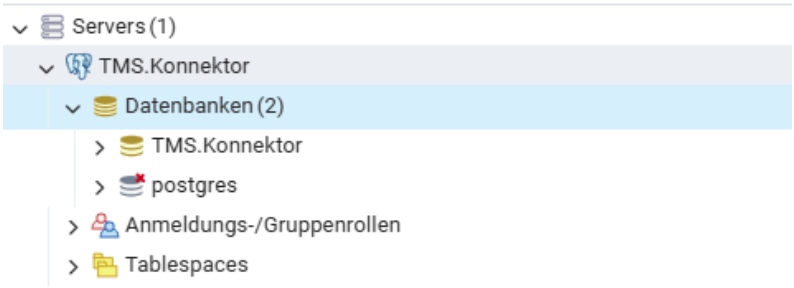


Alternativ finden Sie die Verbindungsdaten ebenfalls aus dem Abschnitt postgres in der Rubrik „environment“.

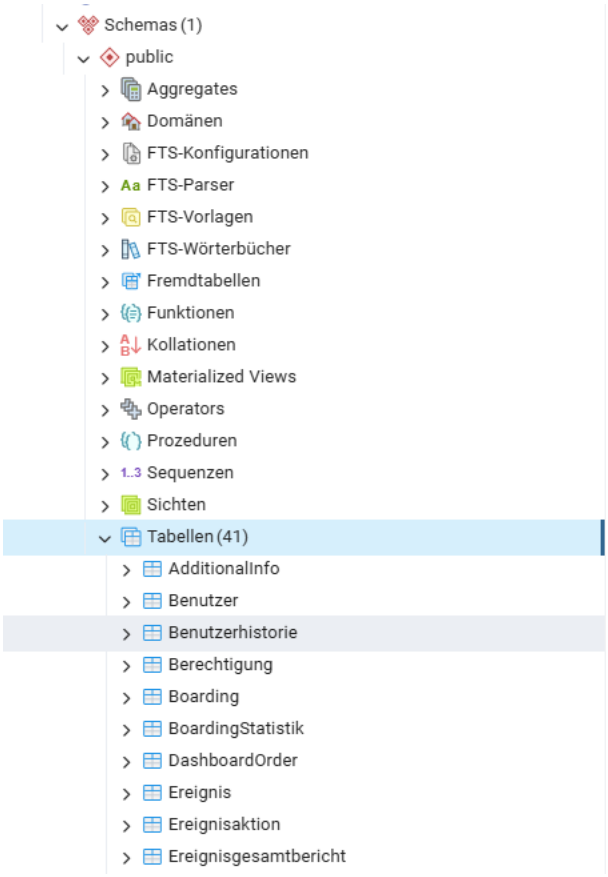
8. Tragen Sie die Werte für Hostname und Hostadresse, Port, Wartungsdatenbank, Benutzername und Passwort ein:



9. Klicken Sie auf „Speichern“.
   Nach erfolgreicher Verbindung sehen Sie Ihre Datenbankeinträge unter „Datenbanken“:



10. Navigieren Sie unter „Schemas“ zu „Tabellen“, um die Tabellenstruktur anzuzeigen:



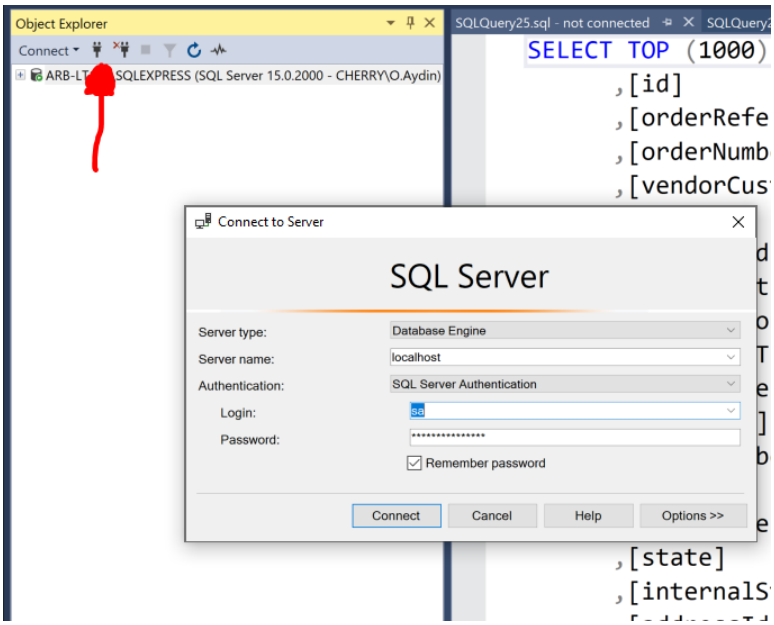
2.2.2.2. Zugriff über SQL Server

SQL Server 2022 kann direkt über das Microsoft Management Studio (SSMS) auf dem Host-PC verwaltet werden. Über die im Container „SQL Server“ freigegebenen Ports ist der Zugriff über das Netzwerk möglich. Nachfolgend finden Sie eine Schritt-für-Schritt-Anleitung, wie Sie sich über SQL Server in die SQL Server-Datenbank einloggen und eine Verbindung erstellen können.

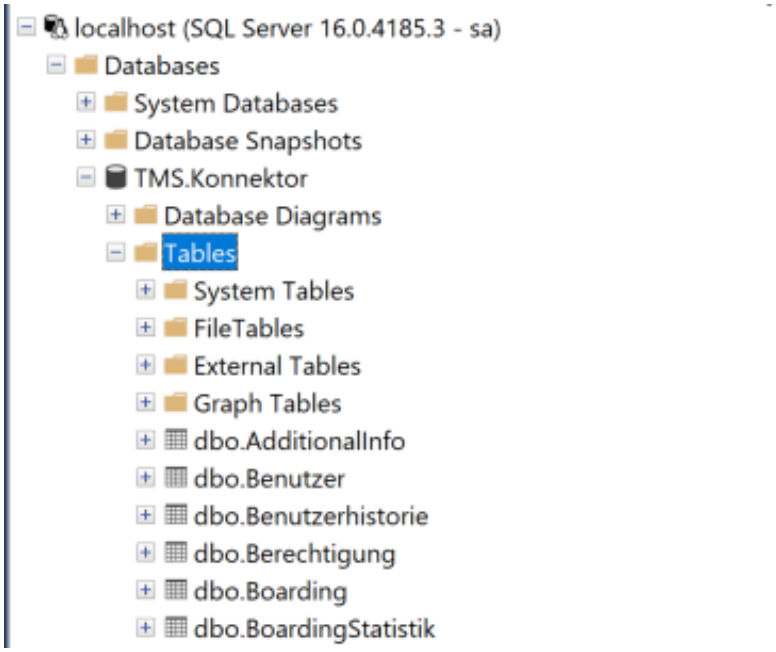
Voraussetzung: Die entsprechenden Container sind vorab erstellt und gestartet worden.

1. Öffnen Sie Microsoft SQL Server Management Studio (SSMS) auf dem Host-PC.
2. Klicken Sie auf „Verbinden“.
3. Geben Sie im Feld „Server name“ localhost ein.
4. Wählen Sie als Authentifizierungsmethode „SQL Server Authentication“ aus.

5. Tragen Sie im Feld „Login:“ den Benutzernamen „sa“ ein:



6. Tragen Sie das Passwort ein.
   Das Passwort finden Sie in der docker-compose.yml im Abschnitt „environment” des SQL-Containers „sql“, Variable: „MSSQL_SA_PASSWORD“.
   Alternativ kann das Passwort auch im Connection-String des TMS-Containers eingesehen werden.
7. Klicken Sie auf „Connect“, um die Verbindung herzustellen.
   Nach erfolgreicher Verbindung sehen Sie im Objekt-Explorer die verfügbaren Datenbanken unter „Databases“:



2.3. Installation von Docker Desktop

In diesem Kapitel wird die Installation von Docker Desktop beschrieben, um die Grundlage für die Ausführung der TMS-Anwendung bereitzustellen.

2.3.1. Begriffsklärungen

Docker	Docker Desktop
• Docker ist eine Software, mit der Anwendungen in sogenannten Containern ausführt werden. Ein Container ist eine Art „Box“, die alles enthält, was die Anwendung zum Laufen braucht – also die Anwendung selbst, Abhängigkeiten und diverse Einstellungen. • Das Besondere an Docker ist, dass die Container überall gleich funktionieren – egal ob auf einem Entwickler-PC, einem Server oder in der Cloud. • Docker macht die Installation und Verwaltung von Anwendungen einfacher, weil man nicht mehr manuell alle benötigten Programme und Einstellungen einrichten muss. Stattdessen startet man einfach einen Container mit einem einzigen Befehl.	• Um Docker effizient nutzen zu können, empfehlen wir Docker Desktop. Docker Desktop ist eine Anwendung, die Docker auf einem Windows oder Linux-Computer nutzbar macht. Sie stellt eine einfache Benutzeroberfläche bereit und sorgt dafür, dass Docker-Container lokal ausgeführt werden können. • Ein zentraler Bestandteil von Docker Desktop ist die integrierte Konsole (Terminal), über die direkt Befehle ausgeführt werden können. Zudem stellt Docker Desktop eine Dateisystem-Ansicht bereit, mit der man das Dateisystem innerhalb eines Containers erkunden kann.

2.3.2. Docker unter Windows installieren

1. Laden Sie die Installationsdatei für Docker Desktop direkt von der offiziellen Webseite herunter:

   Docker Desktop für Windows

2. Führen Sie die Installationsdatei aus und folgen Sie den Anweisungen des Installers.

3. Installieren Sie während der Installation WSL2 für Windows mit. Docker Desktop erkennt dies automatisch und bietet die Einrichtung an:

2.3.3. Docker unter Linux (Ubuntu) installieren

1. Laden Sie die Installationsdatei für Docker Desktop direkt von der offiziellen Webseite herunter:

   Docker Desktop für Linux (Ubuntu/Debian)

2. Wechseln Sie in das Verzeichnis, in dem sich die DEB-Datei befindet.
3. Vergeben Sie die notwendigen Rechte, damit die Datei ausgeführt werden kann:

```
chmod +x Docker Desktop-amd64.deb
```

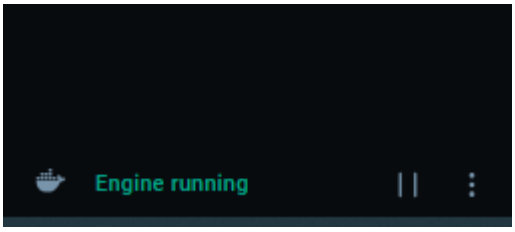
4. Führen Sie die Datei aus:

```
./Docker Desktop-amd64
```

5. Folgen Sie den Anweisungen auf dem Bildschirm, um die Installation abzuschließen.

2.3.4. Installation überprüfen

1. Starten Sie nach der Installation die Anwendung Docker Desktop und prüfen Sie, ob die Docker-Engine läuft. Dies erkennen Sie an der Statusanzeige „Engine running“ unten links:



Die folgenden Befehle zeigen die Versionen der installierten Docker-Komponenten an und helfen Ihnen dabei zu prüfen, ob Docker korrekt installiert wurde. Führen Sie die Befehle an einer der folgenden Stellen aus:

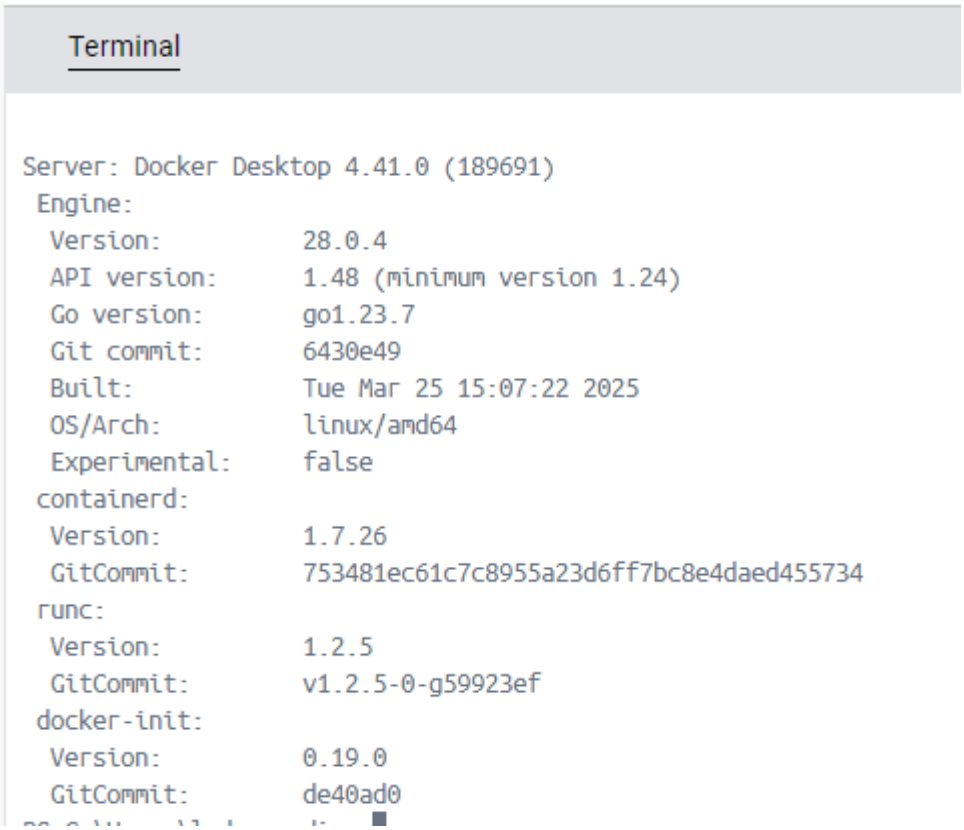
- Windows: Verwenden Sie entweder die integrierte PowerShell oder alternativ das Docker-Desktop-Terminal:
  → Dashboard → Container → Unten rechts „Terminal” anklicken
- Linux: Öffnen Sie das native Terminal.

2. Geben Sie anschließend folgende Befehle ein:

```
# Docker Version anzeigen
docker –version

# Docker Compose Version anzeigen
docker-compose –version
```

Wenn beide Versionen angezeigt werden, ist Docker Desktop erfolgreich installiert und einsatzbereit:



2.3.5. Docker Desktop konfigurieren

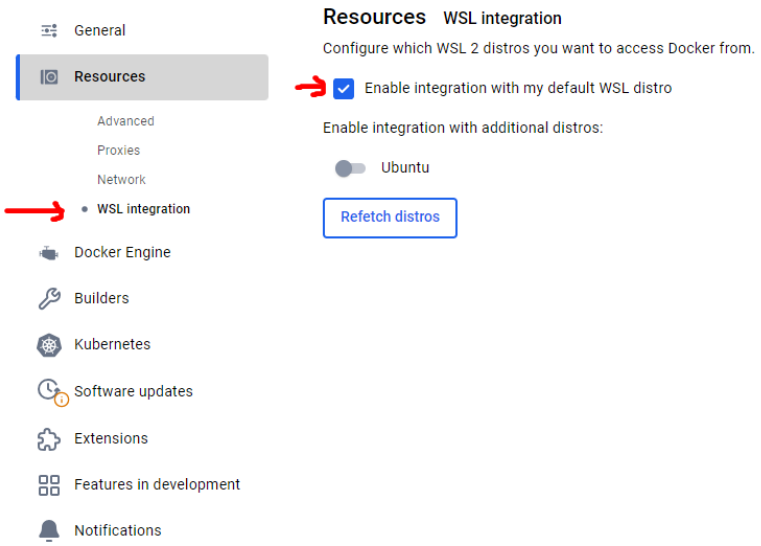
Dieser Abschnitt beschreibt die Konfiguration von Docker Desktop, um eine optimale Umgebung für die TMS-Anwendung bereitzustellen.

1. Docker Desktop öffnen: Starten Sie Docker Desktop über das Desktop-Icon oder über das Startmenü.
2. Einstellungen aufrufen: Klicken Sie oben rechts auf das Zahnradsymbol (Settings), um die Einstellungen zu öffnen.
3. Stellen Sie sicher, dass WSL2 aktiviert ist:
    a. Navigieren Sie zum Menüpunkt „General“.

b. Überprüfen Sie, ob „Use the WSL 2 based engine“ aktiviert ist:

c. Navigieren Sie zum Menüpunkt „Resources“ → „WSL Integration“.

d. Stellen Sie sicher, dass die Option „Enable integration with my default WSL distro“ aktiviert ist:



4. Docker Desktop neu starten: Starten Sie Docker Desktop nach den Anpassungen neu, damit die Änderungen übernommen werden.

2.4. Anpassung von Docker Compose

2.4.1. Begriffserklärungen

Die Datei docker-compose.yml definiert die gesamte Konfiguration der Anwendung. Sie legt fest, welche Container gestartet werden, wie diese miteinander verbunden sind, welche Umgebungsvariablen gesetzt werden und welche Ports und Volumes verwendet werden.

Diese Datei enthält bereits vorkonfigurierte Werte, die den Betrieb ermöglichen. Die Datei kann auch bei Bedarf nach Kundenwünschen angepasst werden, z. B. durch die Änderung von Datenbankverbindungen, E-Mail-Einstellungen oder Netzwerkadressen.

2.4.2. Standardlizenz verwenden und bei Bedarf anpassen

Die Standardlizenz der Docker-Umgebung enthält bereits eine vorkonfigurierte Lizenz mit den Standardwerten:

- Kundenkennung: Cherry
- Installation: TMS

Wenn Sie die Standardinstallation nutzen, überspringen Sie diesen Abschnitt. Sollten jedoch Anpassungen notwendig sein, wie z. B. der Einsatz eines eigenen E-Mail-Servers oder die Anpassung der Datenbank-Passwörter, bietet es sich an, diese Anpassungen in der docker-compose.yml vorzunehmen.

2.4.3. Notwendige und empfohlene Anpassungen an docker-compose.yml

Die Konfiguration in der docker-compose.yml unterscheidet zwischen folgenden Anpassungen:

- Notwendige Anpassungen (N*): erforderlich für den Betrieb des Systems
- Empfohlenen Anpassungen (E*): optional

Die Übersichtstabelle gibt Ihnen einen schnellen Überblick über die notwendigen und empfohlenen Anpassungen. Details zu den jeweiligen Anpassungen finden Sie in den nachfolgenden Kapiteln, die Schritt für Schritt die einzelnen Parameter und deren Bedeutung erklären:

Container	Bereich	Kategorie	Parameter	N*	E*	Bemerkung
tms	volumes	Zertifikat	- ./cert:/root/.aspnet/https	x		Prüfen Sie, ob ein Verzeichnis "cert" existiert und ob darin das entsprechende Zertifikat abgelegt ist.
tms	environment		ASPNETCORE_Kestrel__Certificates__Default__Path ASPNETCORE_Kestrel__Certificates__Default__Password		x	
tms		Lizenz	Installation Kundenkennung Kundenschlüssel	x		Die Werte für Installation, Kundenkennung und Kundenschlüssel erhalten Sie aus der Lizenzdatei. Diese Werte müssen zwingend eingetragen werden, damit die Anwendung korrekt startet. Siehe Kapitel 2.4.4 für detaillierte Erklärungen.
tms		E-Mail	EmailSender (FromName bis Password)		x	Siehe Kapitel 2.4.5 für detaillierte Erklärungen.
tms		Daten-bank	TMS_DB_CONNECTION=…		x	Siehe Kapitel 2.4.3 für detaillierte Erklärungen.
postgres			Alle Einträge aus environment		x	
pgadmin			Alle Einträge aus environment		x	
sql			Alle Einträge aus environment außer Listenport		x	

2.4.4. Datenbank: Passwörter und Zugangsdaten festlegen

Hinweis: Zu den Passwörtern und Zugangsdaten müssen bestimmte Umgebungsvariablen in der Datei docker-compose.yml gesetzt werden. Nach dem Ändern der Passwörter für die Datenbanken muss der Parameter "TMS_DB_CONNECTION" angepasst werden.

Container	tms	
Bereich	**Variable**	**Beispielwert**
Environment	TMS_DB_CONNECTION=	postgres;Port=5432; Database=TMS.Konnektor; Username=TMS.Konnektor; Password=Cherry2025!;
		sql_server2022; Database=TMS.Konnektor; User Id=sa; Password=Cherry2025!; MultipleActiveResultSets=true; trustServerCertificate=true;

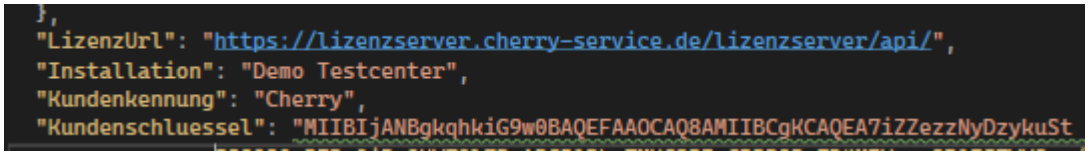
Container	sql	
Bereich	**Variable**	**Beispielwert**
SQL Server	SA_PASSWORD	Cherry2025!
	MSSQL_DATABASE	TMS.Konnektor

Container	postgres	
Bereich	**Variable**	**Beispielwert**
PostgreSQL	POSTGRES_USER	TMS.Konnektor
	POSTGRES_PASSWORD	Cherry2025!
	POSTGRES_DB	TMS.Konnektor

2.4.5. Lizenz prüfen oder hinterlegen

Variable	Beispielwert
Installation	Demo Testcenter (Wert unter „Installation in der Lizenzdatei“)
Kundenkennung	Cherry (Wert unter „Kundenkennung“ in der Lizenzdatei“)
Kundenschlüssel	MII... (Wert unter „Kundenschluessel“ in der Lizenzdatei)
volumes → cert	./cert:/root/.aspnet/https
ASPNETCORE_Kestrel__Certificates__Default__Path	/root/.aspnet/https/certificate.pfx
ASPNETCORE_Kestrel__Certificates__Default__Password	Cherry2025!

Auszug aus der Lizenzdatei:



Hinweis:

Optional können Sie das vorgefertigte Zertifikat im Ordner „cert“ durch ein eigenes Zertifikat ersetzen. Hierzu müssen Sie in der docker-compose.yml sowohl den **Pfad** (volumes) als auch die **Zertifikatsparameter** (Path und Password) anpassen.

2.4.6. E-Mail-Einstellungen konfigurieren

Standardmäßig wird der Standard-Mailserver der Anwendung verwendet. Wenn Sie einen eigenen Mail-Server verwenden möchten, können Sie die entsprechenden Einträge in den Zeilen 23 bis 29 eintragen und mit Ihren eigenen Werten versehen.

Weitere Umgebungsvariablen wie z. B. E-Mail-Server-Einstellungen oder Zertifikate können Sie in der docker-compose.yml eintragen.

Beispiel für E-Mail-Konfiguration:

Bereich	Variable	Beispielwert
E-Mail	EmailSender.FromName	TMS
	EmailSender.FromEmail	admin@cherry.de
	EmailSender.Host	smtp.cherry.de
	EmailSender.Port	587
	EmailSender.EnableSSL	true
	EmailSender.UserName	admin@cherry.de
	EmailSender.Password	examplepassword

2.5. Docker-Hub-Login

Bevor Sie die Anwendung starten, müssen Sie die benötigten Docker-Images aus dem Docker Hub herunterladen. Hierfür stellt CHERRY einen zeitlich begrenzten Access Token bereit, der für den Login erforderlich ist.

1. Öffnen Sie PowerShell oder ein Terminal.
    - Unter Windows: PowerShell oder Docker-Desktop-Terminal
    - Unter Linux: Terminal
2. Melden Sie sich an.
   Verwenden Sie für die Anmeldung den folgenden Befehl:
   Ersetzen Sie $Token$ und $Nutzer$ durch die Zugangsdaten, die Sie von CHERRY erhalten haben:

```
echo $Token$ | docker login -u $Nutzer$ --password-stdin
```

Siehe E-Mail:

anbei ihre Daten für das Docker-Login:

Token: andreash163
User: dckr_pat_XM1d2FZSrU3QdTFnjCYlm2ackHE

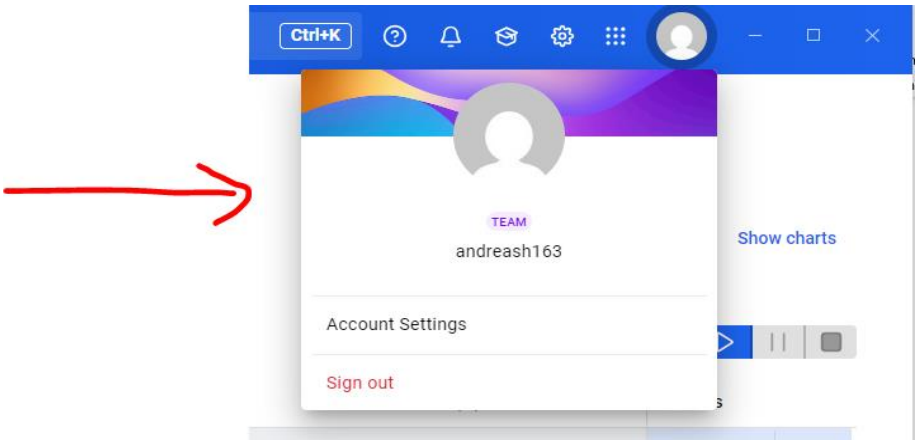
Geben Sie diesen Befehl in die Konsole /Terminal ein :
echo dckr_pat_XM1d2FZSrU3QdTFnjCYlm2ackHE | docker login -u andreash163 --password-stdin

Dieser Token ist nur 14-Tage gültig.

Mit freundlichen Grüßen,

ihr TMS-Team

3. Überprüfen Sie den Login-Status auf Docker Desktop:



Der Benutzername sollte hier sichtbar sein. Falls dies nicht der Fall ist, überprüfen Sie Ihre Zugangsdaten.

3. Installation

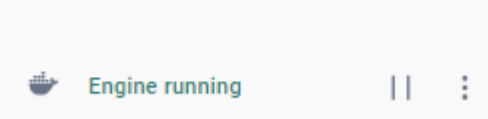
3.1. TMS-Software installieren

Dieser Abschnitt beschreibt, wie Sie die TMS-Software mit Docker Compose installieren, starten und überprüfen.

1. Öffnen Sie die Eingabeaufforderung bzw. die Eingabekonsole.
2. Wechseln Sie in das Verzeichnis, in dem sich die docker-compose.yml befindet:

```
# Ersetze Sie bitte diesen Pfad
cd /pfad/zur/docker-compose.yml
```

3. Überprüfen Sie, ob Docker Desktop gestartet wurde:
   Öffnen Sie Docker Desktop und stellen Sie sicher, dass die Docker-Engine läuft (siehe unten links):



4. Nun kann die Installation und Inbetriebnahme der Container erfolgen. Hierfür gibt es zwei Optionen:

   - 1. Option: Docker Compose im Hintergrund starten:

```
docker-compose up -d
```

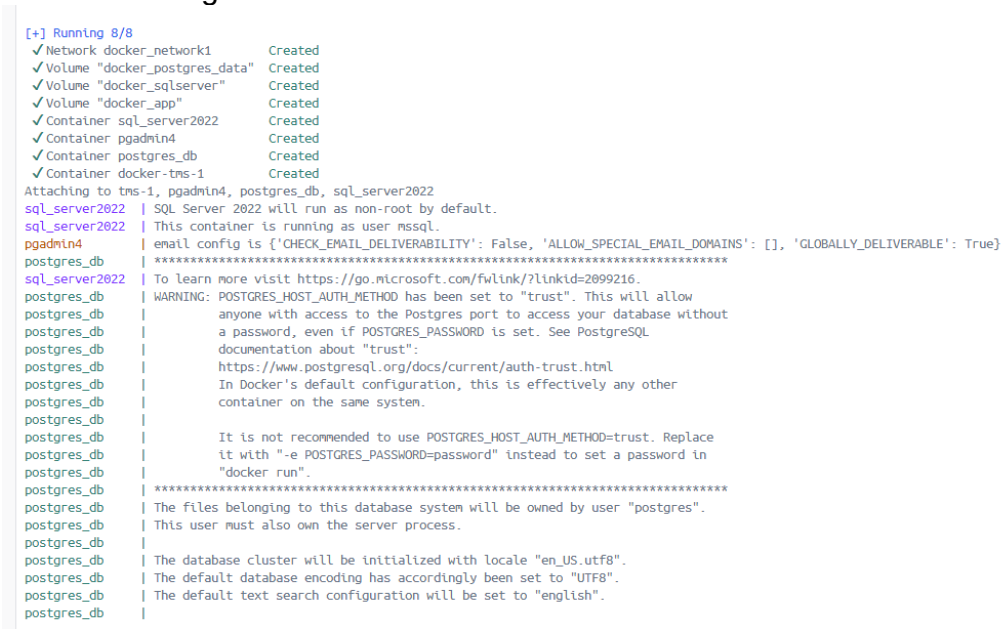
   Erklärung, was dieser Befehl macht:

   - Die benötigten Images werden aus dem privaten Repository geladen, falls diese lokal nicht vorhanden sind.
   - Die definierten Container werden erstellt und gestartet.
   - Alle Dienste, die in der `docker-compose.yml` definiert sind, werden aktiv.
   - Die Container laufen bei der Angabe „-d“ im Hintergrund und das Terminal bleibt frei für weitere Befehle.

- 2. Option: Docker Compose im Vordergrund starten (Debug-Modus):

```
docker-compose up
```

  - Sichtbare Ausgabe:



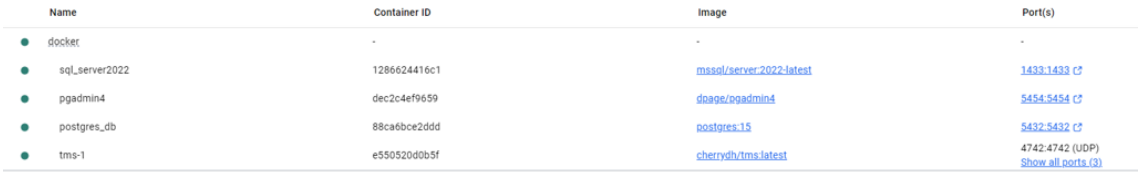
  - Die Logs aller Container werden direkt im Terminal angezeigt.
  - Ideal für Debugging und Fehleranalyse.

Hinweis: Wenn die Container im Vordergrund gestartet wurden, bleiben die Logs so lange sichtbar, bis der Prozess mit STRG + C beendet wird. Die Container laufen dann weiter im Hintergrund.

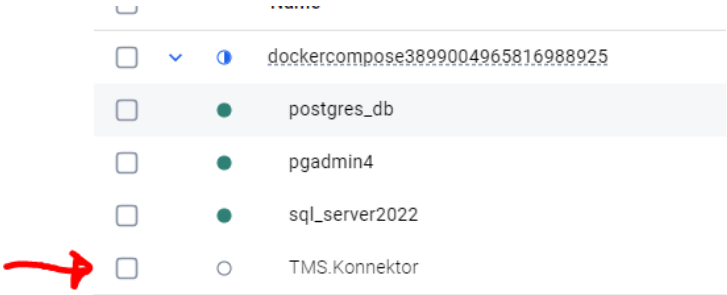
3.2. Container überprüfen

Nach dem Start der Container kann überprüft werden, ob diese ordnungsgemäß laufen. Zur Überprüfung gehen Sie wie folgt vor:

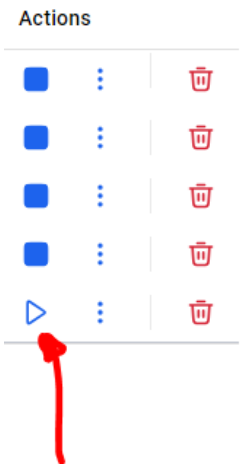
1. Navigieren Sie zu „Container“.
   Jeder gestartete Container wird hier mit einem grünen Ampelzeichen angezeigt:



   Wurde ein Container nicht gestartet, ist er ausgegraut:



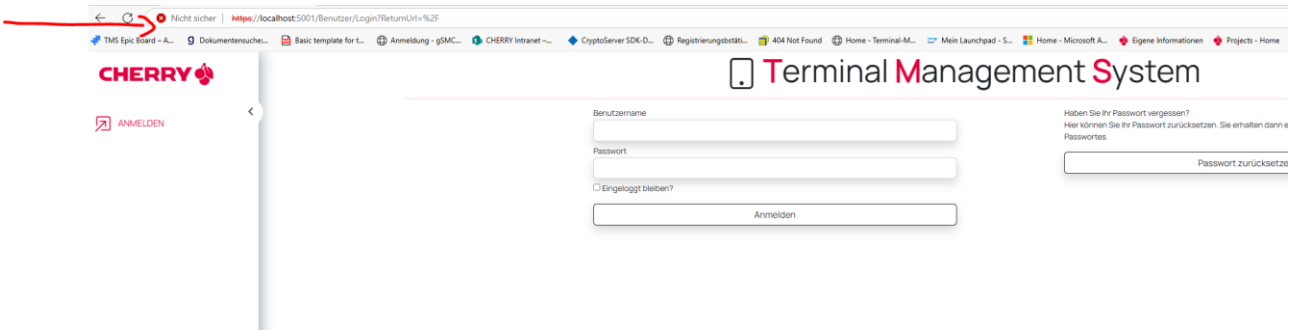
   Tipp: Ein grünes Ampelzeichen bedeutet, dass der Container vollständig läuft. Ist dies nicht der Fall, können Sie über den Neustart versuchen, den Container neu zu starten:



In der folgenden Tabelle finden Sie eine Übersicht aller relevanten Docker-Container, die für die TMS-Anwendung gestartet werden:

Containername	Beschreibung	Verwaltungstool	Netzwerkzugriff
tms	TMS-Software basiert auf ASP.NET 9 und Kestrel	Docker Desktop Dashboard, docker ps	Über Port 5000 (HTTP) und 5001 (HTTPS)
postgres_db	PostgreSQL 15 Datenbank zur Speicherung der Anwendungsdaten	pgAdmin4 (im Container pg4-admin)	Über Port 5432
pgadmin4	Grafische Oberfläche zur Verwaltung der PostgreSQL-Datenbank	Webzugriff über Browser (http://localhost:5454)	Über Port 5454
sql_server2022	Microsoft SQL Server 2022-Datenbank mit Zugriff über SQL Server Management Studio (SSMS) auf dem Host-PC	SQL Server Microsoft Management Studio (SSMS)	Über Port 1433

2. Die TMS-Applikation ist über https://localhost:5001 (nur lokal) oder über die IP-Adresse des Hosts erreichbar:



3.3. Container stoppen

Dieser Abschnitt beschreibt, wie die Anwendung und ihre Container sauber gestoppt werden.

Geben Sie folgenden Befehl ein, um die Anwendung zu stoppen (Terminaleingabe):

```
docker-compose down
```

Erklärung:

- Dieser Befehl beendet alle laufenden Container, die in der docker-compose.yml definiert sind.
- Die Konfiguration der Container bleibt erhalten.

Wichtig: Die Images und die Volumes (Datenbanken, persistente Daten) bleiben weiterhin bestehen.

3.4. Logs untersuchen

Dieser Abschnitt beschreibt, wie die Logs der Docker-Container über das Docker Desktop Dashboard eingesehen und analysiert werden. Die Log-Ausgabe ermöglicht es, Fehler, Warnungen und allgemeine Informationen der Anwendung zu überwachen.

3.4.1. Logs über Docker Desktop anzeigen

1. Öffnen Sie Docker Desktop.
2. Navigieren Sie zu „Container“.
3. Klicken Sie auf den entsprechenden Container (**tms**, **postgres_db**, **sql_server2022** oder **pgadmin4**).
4. Wählen Sie im Menü den Punkt „Logs“ aus.

Hier werden die Logs des Containers in Echtzeit angezeigt. Im Logfenster sehen Sie:

- **Informationen:** Allgemeine Betriebsinformationen der Anwendung.
- **Warnungen:** Hinweise auf mögliche Probleme.
- **Fehler:** Kritische Probleme, die behoben werden müssen.

3.4.2. Log-Level: Informationen durchsuchen und bewerten

Die meisten Logs sind auf dem Log-Level Information und zeigen den Ablauf der Anwendung. Hauptsächlich kann über diese Logs das Boarding überprüft werden.

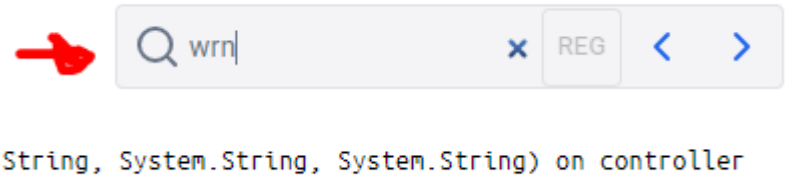
Im nachfolgenden Beispiel wird dargestellt, dass vier Kartenterminals gefunden wurden:

Suchbegriff: BoardingBackgroundController

```
[15:19:52 INF] BoardingBackgroundController => Kartenterminal ORGA6100-0142000002150C updated
[15:19:53 INF] BoardingBackgroundController => Kartenterminal ST-1506-A00128323 updated
[15:19:54 INF] BoardingBackgroundController => Kartenterminal ST-1506-A00128331 updated
[15:19:55 INF] BoardingBackgroundController => Kartenterminal ST-1506-A00128314 updated
[15:19:55 INF] BoardingBackgroundController => 4 Kartenterminals found via Network-Boarding!
```

3.4.3. Logs nach Fehlern und Warnungen filtern und durchsuchen

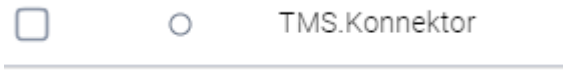
Sie können im Docker Desktop Dashboard die Logs nach bestimmten Begriffen durchsuchen. Nutzen Sie das Suchfeld im Logbereich, um gezielt nach Fehlern (err) oder Warnungen (wrn) zu suchen:



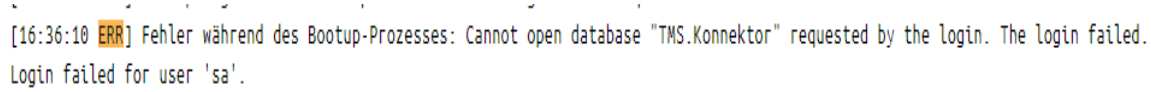
Hinweise:

- Die Suche ist nicht casesensitiv, d. h. Groß- und Kleinschreibung spielt keine Rolle.
- Die Logs bleiben im Docker Desktop erhalten, solange der Container aktiv ist. Beim Neustart des Containers werden die Logs neu geschrieben.

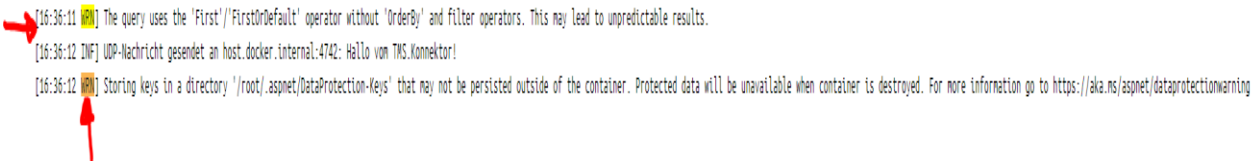
1. Nach Fehlern suchen:
   Container ist zum Beispiel ausgegraut:

   Geben Sie „err“ ein, um nach allen Fehlermeldungen zu filtern:



2. Nach Warnungen suchen:
   Geben Sie "warn" ein, um nach allen Warnungen zu filtern:



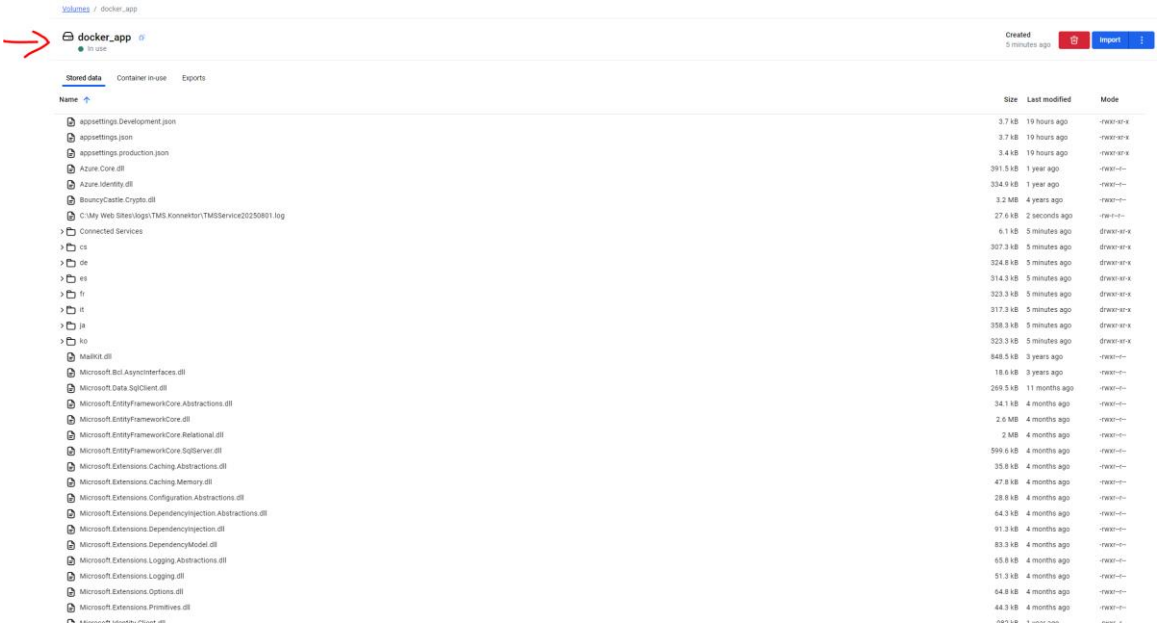
3.5. TMS-Software aktualisieren (Update durchführen)

Hinweis: Nur das Volume der Hauptanwendung TMS darf gelöscht werden, nicht die Volumes der Datenbanken, da sonst alle Daten verloren gehen.

1. Stoppen Sie den Container:

   Befehl:
   docker-compose down

2. Löschen Sie gezielt das Volume der Anwendung “docker_app“:



   Befehl:
   docker volume rm docker_app

3. Bauen Sie oder laden Sie die Container neu. In diesem Fall wird nur das TMS-Image neugeladen, während die bestehenden Container (Datenbanken) unverändert bleiben, um ihre Daten weiterhin aus dem vorhandenen Volume beziehen zu können:

   Befehl:
   docker-compose up -d

4. Anhang

4.1. Beschreibungstabelle docker-composer.yml

Contrainer	tms	
Eintrag	**Inhalt**	**Erklärung**
„image“	image: cherry/tms:latest	Definiert das Image für das TMS. Standardgemäß wird immer das Tag „latest“ verwendet. Dadurch wird beim Starten der Container immer automatisch die aktuell verfügbare Version des Images aus dem Docker-Repository geladen. Wenn Sie stattdessen gezielt mit einer bestimmten Version arbeiten möchten (z. B**. 2.02** oder **2.03**), können Sie das Tag „latest“ einfach durch die gewünschte Versionsnummer ersetzen. Auf diese Weise können Sie bei Bedarf auch auf ältere Versionen zugreifen.
„ports“	5000:8080	Externer Port 5000 wird auf internen Port 8080 des Containers gemappt (HTTP).
	5001:8081	Zuordnung von externen Port 5001 auf internen Port 8081 (HTTPS)
	4742:4742/udp	Port 4742 für den Service-Discovery zum Boarden der Kartenterminals
„volumes“		Mit volumes werden Verzeichnisse vom Host-PC in den Container eingebunden (gemountet). Damit können Daten persistiert oder bestimmte Ressourcen (z. B. Zertifikate) bereitgestellt werden, die der Container benötigt.
	./cert:/root/.aspnet/https	Bindet den lokalen Ordner cert (auf dem Host) in das Verzeichnis /root/.aspnet/https im Container ein. Der Ordner muss sich auf der gleichen Ebene wie die Datei docker-compose.yml befinden. Der Ordner enthält ein selbstsigniertes HTTPS-Zertifikat, das von der Anwendung benötigt wird. Alternativ kann auch ein Zertifikat einer offiziellen Zertifizierungsstelle verwendet werden.

Contrainer	tms	
Eintrag	**Inhalt**	**Erklärung**
	/app:/app	Dieses Volume bindet den lokalen Ordner app in den gleichnamigen Ordner im Container ein. Wird verwendet, um die Applikationsdateien, Bibliotheken und Skripte bereitzustellen, die im Container benötigt werden.
	C:/Program Files/WireGuard:/host_wireguard_all:rw	Dieses Volume bindet das Verzeichnis C:\Program Files\WireGuard vom Windows-Host in den Container (mit Lese- und Schreibrechten). Ermöglicht dem Container Zugriff auf die lokale WireGuard-Installation auf dem Windows-Host. Es können die WireGuard-Hub-Funktionen auch über Docker über das Einbinden dieses Volumenes durchgeführt werden. Falls WireGuard an einem anderen Pfad installiert wurde, muss dieser Pfad im Volume-Eintrag entsprechend angepasst werden.
„networks“		Mit „networks“ wird definiert, wie die einzelnen Container miteinander kommunizieren können. Docker erstellt dabei ein eigenes virtuelles Netzwerk, über das die einzelnen Container (TMS, Datenbank) miteinander verbunden sind.
	network1	Das Netzwerk network1 stellt sicher, dass alle definierten Container (z. B. tms, postgres, sql) untereinander kommunizieren können, ohne dass sie direkt mit dem restlichen Netzwerk des Hosts verbunden sind. Die Container sind in einer isolierten Umgebung und erhalten dabei ein eigenes Subnetz.
„Environment“		„environment“ wird verwendet, um die Umgebungsvariablen an die Container zu übergeben. Die Umgebungsvariablen sind Konfigurationswerte, die beim Start des Containers übergeben werden. Sie steuern Pfade, Datenbankverbindungsparameter oder andere Einstellungen innerhalb der Anwendung. Vorteil: Die Anwendung bleibt damit flexibel konfigurierbar, ohne dass der Anwendungscode verändert werden muss.
	ASPNETCORE_URLS=https://+:8081;http://+:8080	ASPNETCORE_URLS=https://+:8081;http://+:8080: Die Anwendung nutzt den Kestrel Web-Server, der im Container die eingehenden HTTP/-HTTPS-Anfragen verarbeitet. Kestrel lauscht über diese Ports 8081, 8080 auf Anfragen.

Contrainer	tms	
Eintrag	**Inhalt**	**Erklärung**
		Über das Portmapping (siehe Rubrik „ports“) wird der Zugriff von außen über 5000 (HTTP) und 5001 (HTTPS) ermöglicht. Dadurch können Clients über die Host-Ports 5000/5001 auf die Anwendung zugreifen, obwohl Kestrel im Container auf anderen Posts läuft.
	ASPNETCORE_HTTPS_PORT=5001	HTTPS-Port nach außen, sollte nur im Einklang mit dem Port-Mapping angepasst werden.
	ASPNETCORE_Kestrel__Certificates__Default__Path=/root/.aspnet/https/certificate.pfx	Gibt den Pfad zum Zertifikat des Containers an, dass für die HTTPs-Kommunikation zwingend notwendig ist. Dieser Pfad verweist auf den Ordner /root/.aspnet/https, der über das Volume /cert:/root/.aspnet/https vom Host (./cert:/) eingebunden wird (siehe Abschnitt „volumes“ oben). Stellen Sie sicher, dass die Zertifikatsdatei certificate.pfx im Ordner cert liegt. Falls Sie eigene Zertifikate verwenden oder eigene Ordnerstruktur nutzen, überprüfen Sie, ob der Ordner-Eintrag im „volumes“ zum Zertifikatspfad in der Umgebungsvariable (ASPNETCORE_Kestrel__Certificates__Default__Path=/) passt.
	ASPNETCORE_Kestrel__Certificates__Default__Password=Cherry2025!	Beinhaltet das Passwort des Zertifikats. Überprüfen Sie, ob das Passwort korrekt ist.
	TMS_DB_CONNECTION=Server=sql_server2022;Database=TMS.Konnektor;User Id=sa;Password=Cherry2025!;MultipleActiveResultSets=true;trustServerCertificate=true;	Definiert die Verbindung zur SQL-Server-Datenbank. Diese Variable steuert, ob die Anwendung mit SQL Server oder ProstgreSQL kommuniziert. Wichtig: Es sind zwei Verbindungszeichenfolgen für TMS_DB_Connection vorhanden: • Eine für SQL Server • Eine für PostreSQL Nur eine dieser Verbindungen darf aktiv sein. Per Default ist der SQL Server aktiv. Die jeweils nicht verwendete Verbindung muss durch ein Rautezeichen (#) am Zeilenanfang auskommentiert werden.

Contrainer	tms	
Eintrag	**Inhalt**	**Erklärung**
		Das System erkennt anhand des Connections-Strings, ob es sich um SQL Server oder PostgreSQL handelt!
	TMS_DB_CONNECTION=Host=postgres;Port=5432;Database=TMS.Konnektor;Username=TMS.Konnektor;Password=Cherry2025!;	Das ist der PostgeSQL-Datenbankverbindungsstring. Sofern PostgreSQL genutzt werden soll, kommentieren Sie den SQL-Server-Verbindungsstring aus.
	SERILOG_LOG_FILE_PATH=/usr/tms.konnektor/logs/TMSService.log	Ablageort für die Logs im Container. Kann beliebig angepasst werden.
	EmailSender.FromName=TMS EmailSender.FromEmail=yourmail@address.de EmailSender.Host=smtp-mail.outlook.com EmailSender.Port=587 EmailSender.EnableSSL=3 EmailSender.UserName=yourmail@address.de EmailSender.Password=yourPassword	Dienen zur Einrichtung des eigenen E-Mail-Servers, über den das TMS E-Mails verschicken kann. Diese Zeilen sind standardmäßig mit # auskommentiert, weil das TMS dann den Standard-Mailserver nutzt. Aus Datenschutzgründen wird empfohlen, einen eigenen SMTP-Server zu verwenden. Dann werden diese Zeilen auskommentiert und die Daten hier eingetragen. Parameter: FromName: Anzeigename des Absenders FromMail: Absenderadresse Host: SMTP-Serveradresse Port: SMTP-Port EnableSSL: Verschlüsselung (3 steht für STARTTLS) Username / Password: Zugangsdaten für den SMTP-Server
	Installation=Demo Testcenter Kundenkennung=Cherry Kundenschluessel=MII…	Diese Parameter steuern die Lizenzierung und Zuordnung der Installation. Installation: Gibt den Namen der jeweiligen Installation an und muss mit der Installation in der Lizenz übereinstimmen. Kundenkennung: Die eindeutige Kennung des Kunden gemäß Lizenz. Kundenschlüssel: Der Lizenzschlüssel. Anpassungen erfolgen nur in Abstimmung mit CHERRY, da diese Werte zur Lizenzprüfung anhand der Lizenzdatei verwendet werden.

Contrainer	tms	
Eintrag	**Inhalt**	**Erklärung**
		Diese Parameter steuern die Datenbankmigrationen, also das automatische Ausführen von SQL-Skripten, um die Datenbankstruktur auf den aktuellen Stand zu bringen.
	Migrations_Aktiv=Ja	Migrations_Aktiv: Gibt an, ob die Migrationen aktiviert sind. Wenn auf Ja gesetzt, werden bei jedem Start des Containers die definierten Skripte ausgeführt.
	Migrations_Dateien=1.8.6-V1_DatenbankErzeugen.sql;1.8.6-V2_TabellenErzeugen.sql;1.8.6-V3-ErzeugeBenutzer.sql	Migrations_Dateien: Listet die auszuführenden Skripte auf, die die Datenbankstruktur aktualisieren. Dieser Wert wird von CHERRY verwaltet und sollte nicht angepasst werden.
„depends_on“	postgres: condition: service_healthy sql: condition: service_healthy	Mit depends_on wird festgelegt, dass der TMS-Container gestartet wird, wenn die angegebenen abhängigen Container (Datenbanken) bereit sind. In diesem Fall sind das die PostgreSQL-Datenbank (postgres) und der SQL Server (sql). Das TMS wartet so lange, bis die Datenbanken vollständig hochgefahren und einsatzbereit (healthy) sind. Dies wird durch die Bedingung condition service_healthy sichergestellt, die prüft, ob der jeweilige Container einen erfolgreichen HealthCheck meldet.

Container	sqlserver_2022	
Eintrag	**Inhalt**	**Erklärung**
Service-Name (Containerbezeichnung)	sql	Dieser Service stellt SQL Server 2022 von Microsoft als Datenbank zur Verfügung.
„image“	"mcr.microsoft.com/mssql/server:2022-latest"	Das Image lädt die aktuelle Version des SQL Servers 2022 aus dem offiziellen Microsoft-Repository herunter.
„container_name“	sql_server2022	Sollte nicht editiert werden
„ports“	1433:1433	1433 ist der Standardport für SQL Server Der Port wird 1:1 vom Host in den Container gemappt, so dass der SQL Server sowohl intern als auch extern (Host) über diesen Port erreichbar ist.
„enviroment“	ACCEPT_EULA=Y	Bestätigt automatisch die EULA von Microsoft
	MSSQL_SA_PASSWORD=Cherry2025!	MSSQL_SA_Password: Legt das Passwort für den SA-User im SQL Server fest. In diesem Fall: Cherry2025! Sofern Sie diesen Wert anpassen, müssen Sie auch im TMS-Container den Verbindungsstring anpassen.
„healthcheck“	test: ["CMD-SHELL", "/opt/mssql-tools18/bin/sqlcmd -U sa -P Cherry2025! -Q 'SELECT 1' -C || exit 1"] interval: 10 s timeout: 5 s retries: 5	Prüft, ob der SQL Server einsatzbereit ist Der Healthcheck läuft: Alle 10 Sekunden (interval) Timeout nach 5 Sekunden (timeout) Bis zu 5 Wiederholungen (retries)
„volumes“	sqlserver:/var/opt/mssql	Das Volume sorgt dafür, dass die Datenbank des SQL Server außerhalb des Container gespeichert wird, um die Datenbank persistent zu halten. Dadurch bleiben die Daten erhalten, auch wenn der Container neu gestartet wird.
„networks“	network1	SQL Server ist im gleichen Network wie die anderen Services (tms, etc), damit die Kommunikation untereinander funktioniert.

Container	postgres	
Eintrag	**Wert**	**Erklärung**
Service-Name (Containerbezeichnung)	postgres	Dieser Service als Datenbank zur Verfügung.
„image“	postgres:15	Die Image-Bezeichnung für den PostgreSQL-Server
„environment“	POSTGRES_USER: TMS.Konnektor POSTGRES_PASSWORD: Cherry2025! POSTGRES_DB: TMS.Konnektor POSTGRES_HOST_AUTH_METHOD: trust	Definiert den Benutzer, das Passwort und die Datenbank, die beim Start automatisch erstellt werden. AUTH_METHOD trust: Vereinfacht die Authentifizierung im Container-Network
„ports“	5432:5432	Standardport für Postgres Port wird direkt durchgereicht
„volumes“	postgres_data:/var/lib/postresql/data	Speichert die Daten persistent, auch bei Containerneustart
„healthcheck“	healthcheck: test: ["CMD-SHELL", "pg_isready"] interval: 5s timeout: 5s retries: 5	Prüft mit pg_ready, ob der Server bereit ist
„networks“	network1	PostgreSQL-Server ist im gleichen Network wie die anderen Services (tms, etc), damit die Kommunikation untereinander funktioniert.

Container	pgadmin	
Eintrag	**Wert**	**Erklärung**
Service-Name (Containerbezeichnung)	pgadmin	Stellt PGAdmin4 bereit, eine grafische Weboberfläche zur Verwaltung von PostgreSQL-Datenbanken
„image“	dpage/pgadmin4	Image aus dem Docker-Repo
„container_name“	pgadmin4	Kann beliebig gesetzt werden
„ports“	- 5454/5454/tcp	Der Web-Zugang zu PG-Admin ist über den Hostport 5454 erreichbar: http://localhost:5454
„environment“	POSTGRES_HOST_AUTH_METHOD=trust PGADMIN_DEFAULT_EMAIL=kms@cherry.de PGADMIN_DEFAULT_PASSWORD=Cherry2025! PGADMIN_LISTEN_PORT=5454	PGADMIN_DEFAULT_EMAIL / PASSWORD: Zugangsdaten für den Login in die PG-Admin Weboberfläche PGAdmin_Listen_Port: Setzt den internen Port von PGAdmin auf 5454, muss mit dem externen Port übereinstimmen
„networks“	network1	PGAdmin ist im gleichen Network wie die anderen Services (tms, postgres, etc), damit die Kommunikation untereinander funktioniert.

4.2. Standard-Datei docker-composer.yml

```
services:
  tms:
    image: cherrydh/tms:latest
    ports:
      - "5000:8080"
      - "5001:8081"
      - "4742:4742/udp"
    volumes:
      - ./cert:/root/.aspnet/https
      - app:/app
      - C:/Program Files/WireGuard:/host_wireguard_all:rw
    networks:
      - network1
    environment:
      - ASPNETCORE_URLS=https://+:8081;http://+:8080
      - ASPNETCORE_HTTPS_PORT=5001
      - ASPNETCORE_Kestrel__Certificates__Default__Path=/root/.aspnet/https/
        certificate.pfx
      - ASPNETCORE_Kestrel__Certificates__Default__Password=Cherry2025!
      - TMS_DB_CONNECTION=Server=sql_server2022;Database=TMS.Konnektor;User
        Id=sa;Password=Cherry2025!;MultipleActiveResultSets=true;
        trustServerCertificate=true;
      # Postgres-DB lediglich auskommentieren und sql_server2022 einkommentieren
      #- TMS_DB_CONNECTION=Host=postgres;Port=5432;Database=TMS.Konnektor;
         Username=TMS.Konnektor;Password=Cherry2025!;
      #- SERILOG_LOG_FILE_PATH=/usr/tms.konnektor/logs/TMSService.log
      #- EmailSender.FromName=TMS
      #- EmailSender.FromEmail=yourmail@address.de
      #- EmailSender.Host=smtp-mail.outlook.com
      #- EmailSender.Port=587
      #- EmailSender.EnableSSL=3
      #- EmailSender.UserName=yourmail@address.de
      #- EmailSender.Password=yourPassword
      - Installation=Demo Testcenter
      - Kundenkennung=Cherry
      - Kundenschluessel=MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA7iZZezz
NyDzykuSt5S8O20oBZDu0iBwQUWFQlFPwAPGP1PLp7MUC3PIcCPBB9PcED4MZV+gy2R1Z5TWYSxv/Q
GlErd8YgoIrPo2RXd/kfLXGlp2cOufX/k2GPcS1nQYkAY8OjKMGvLu7pG8RpJdSHNP1szzT5xCYYWEXVby+
MljoFqTsezrMlEjBPnHIsg3mjVgXDgqI2RJ5nc5N7fZX3hjQOxFhbn/F6jzGdrKqKmj6ayeMUhwCMG42IcBKj32
```

```
FIMsmyxOuiEHpBXfiQQpPoCxfiejWd1VEtKKEu7wxmEsGNLYJUGCdw57i+Je4d+vBMePabx9nct8l7odFTpk
MvQIDAQAB
      - Migrations_Aktiv=Ja
      - Migrations_Dateien=1.8.6-V1_DatenbankErzeugen.sql;
        1.8.6-2_TabellenErzeugen.sql;1.8.6-V3-ErzeugeBenutzer.sql
    depends_on:
      postgres:
        condition: service_healthy
      sql:
        condition: service_healthy

  postgres:
   image: postgres:15
   container_name: postgres_db
   environment:
     POSTGRES_USER: TMS.Konnektor
     POSTGRES_PASSWORD: Cherry2025!
     POSTGRES_DB: TMS.Konnektor
     POSTGRES_HOST_AUTH_METHOD: trust
   expose:
     - "5432"
   ports:
     - 5432:5432
   volumes:
     - postgres_data:/var/lib/postgresql/data
   hostname: postgres
   healthcheck:
      test: ["CMD-SHELL", "pg_isready"]
      interval: 5s
      timeout: 5s
      retries: 5
   networks:
     - network1

  pgadmin:
   image: dpage/pgadmin4
   container_name: pgadmin4
   ports:
     - 5454:5454/tcp
```

```
    hostname: pgadmin
    environment:
      - POSTGRES_HOST_AUTH_METHOD=trust
      - PGADMIN_DEFAULT_EMAIL=kms@cherry.de
      - PGADMIN_DEFAULT_PASSWORD=Cherry2025!
      - PGADMIN_LISTEN_PORT=5454
    networks:
      - network1

  sql:
    image: "mcr.microsoft.com/mssql/server:2022-latest"
    container_name: sql_server2022
    ports:
      - "1433:1433"
    environment:
      - ACCEPT_EULA=y
      - MSSQL_SA_PASSWORD=Cherry2025!
    healthcheck:
      test: ["CMD-SHELL", "/opt/mssql-tools18/bin/sqlcmd -U sa -P Cherry2025! -Q 'SELECT 1' -C || exit 1"]
      interval: 10s
      timeout: 5s
      retries: 5
    volumes:
      - sqlserver:/var/opt/mssql
    networks:
      - network1

networks:
    network1:

volumes:
  sqlserver:
    driver: local
  app:
    driver: local
  postgres_data:
    driver: local
```