NEWS · AUGUST 21, 2026

Docker Desktop installieren und den ersten Container starten (1/5)

By Dr. Holger Flick



Stellen Sie sich den Moment vor. Sie stecken mitten in einem Delphi-Client/Server-Feature und brauchen endlich ein echtes Backend zum Entwickeln — eine Postgres-Datenbank etwa, einen Redis-Cache oder einen Message Broker. Auf Ihrem eigenen Windows-Entwicklungsrechner. Jetzt sofort.

Also besorgen Sie sich den Installer. Sie führen ihn aus. Er registriert einen Windows-Dienst, verändert Ihren `PATH`, verteilt Konfigurationsdateien an drei Stellen, öffnet einen Port und verlangt ein Superuser-Passwort, das Sie bis Donnerstag vergessen haben werden. Es funktioniert — bis Sie für ein Projekt Version 15 und für ein anderes Version 17 brauchen und sich plötzlich zwei Dienste um denselben Port streiten. Und wenn das Projekt endet, hinterlässt die Deinstallation verwaiste Dienste, herrenlose Verzeichnisse und eine Registry, die Sie lieber nicht anfassen. Jeder erfahrene Desktop-Entwickler kennt genau dieses mulmige Gefühl.

Es gibt einen besseren Weg, und er ist der ganze Grund für diese Serie. Mit einem einzigen Befehl haben Sie einen echten, laufenden Postgres-Server. Mit einem weiteren ist er wieder weg — sauber, vollständig, als wäre er nie da gewesen. Keine Dienste, keine `PATH`-Operationen, kein Deinstallations-Chaos. Das ist das Versprechen von **Containern**, und dieser Beitrag rüstet Sie aus, um es einzulösen.

Wenn Sie gerade die Serie Netzwerktechnik für Delphi-Entwickler abgeschlossen haben: Dies ist die Fortsetzung, die ich Ihnen an deren Ende versprochen habe. Diese Serie hat Sie fit gemacht in IP-Adressen, Ports, DNS, TCP, HTTP und Web Services. All das werden Sie hier brauchen — aber zuerst kommen das Tooling und drei Ideen, die den Rest der Serie leicht machen.

> **Diese Serie: Docker für Delphi-Entwickler**
> 1. **Docker Desktop installieren und den ersten Container starten — dieser Beitrag** — was ein Container, ein Image und eine Registry wirklich sind 2. Postgres im Container betreiben und aus Delphi ansprechen — eine echte Datenbank mit einem Befehl, FireDAC erstellt und fragt eine Tabelle ab 3. Images, Tags, Layer und Registries — woher Images kommen und wie sie aufgebaut sind 4. Docker Compose: Datenbank und Backend in einer Datei — Multi-Service-Anwendungen und Netzwerkzugriff über Service-Namen 5. Isolationsarchitektur: Web-Client ’ Backend ’ Datenbank— private Netzwerke und wer die Daten erreichen darf

Drei Ideen vor dem ersten Befehl

Bevor wir irgendetwas installieren, gebe ich Ihnen das mentale Modell an die Hand. Docker hat genau drei Konzepte, die Sie im Kopf behalten müssen, und jedes davon lässt sich sauber auf etwas abbilden, das Sie als Delphi-Entwickler bereits kennen. Verstehen Sie diese drei, und die Befehle werden offensichtlich statt magisch.

Die drei sind **Image**, **Container** und **Registry**. Wir nehmen sie der Reihe nach durch.

Ein Image ist eine Klasse

Ein Image ist eine schreibgeschützte Vorlage: ein verpacktes, eingefrorenes Rezept, das eine Anwendung und alles enthält, was sie zum Laufen braucht — das Programm, seine Bibliotheken, seine Standardkonfiguration, bis hinunter zu einem minimalen Ausschnitt eines Linux-Dateisystems. Es *tut* von sich aus nichts. Es liegt einfach da, ein Bauplan, der darauf wartet, zum Leben erweckt zu werden.

Wenn Ihnen das bekannt vorkommt, soll es das auch: **Ein Image ist eine Klasse.** Sie deklarieren es einmal, es hält keinen lebenden Zustand, und es existiert, um instanziiert zu werden.

Ein Container ist ein Objekt

Ein Container ist das, was Sie bekommen, wenn Sie ein Image *ausführen*: eine lebende, laufende Instanz mit eigenem Zustand, eigenem Speicher und einer eigenen, isolierten Sicht auf Dateisystem und Netzwerk. Sie können viele Container aus einem Image starten, genau so, wie Sie viele Objekte aus einer Klasse konstruieren. Jeder ist unabhängig; Sie können einen wegwerfen, ohne das Image oder die anderen anzutasten.

Die Analogie, die jeder Delphi-Entwickler bereits besitzt, fasst also das Ganze in einer Zeile zusammen.



Die eine Analogie, die Docker erschließt: Image verhält sich zu Container wie Klasse zu Objekt

Lesen Sie es von links nach rechts: ein Image links, mehrere lebende Container in der Mitte, und rechts das Delphi-Vokabular. `docker run` ist der Konstruktor. Sobald das klick macht, ist der Rest von Docker nur noch eine Frage der Methodennamen.

Eine Registry ist der Ort, von dem Images kommen

Das Postgres-Image haben nicht Sie geschrieben, und das werden Sie auch nie. Jemand hat es veröffentlicht, und Sie *ziehen* es mit einem Pull. Der Ort, von dem Sie es beziehen, ist eine Registry — ein Server, der Images speichert und ausliefert. Die öffentliche Standard-Registry ist Docker Hub, eine riesige Bibliothek fertiger Images: Postgres, Redis, nginx und Tausende mehr, gepflegt und versioniert, damit Sie es nicht tun müssen.

Denken Sie an eine Registry wie an ein Paket-Repository für ganze Anwendungen statt für Quellcode-Bibliotheken — dem Geist nach näher an Delphis GetIt oder einem NuGet-Feed, aber ausgeliefert wird ein kompletter, lauffähiger Server. So fügen sich die drei Ideen zusammen.



Ein Image aus einer Registry pullen, dann in einen Container starten — der komplette Kreislauf

Das Image wird einmal aus der Registry auf Ihren Rechner gezogen; von dort starten Sie es in Container, wann immer Sie sie brauchen. Behalten Sie diesen Kreislauf im Kopf — genau das tut der nächste Befehl.

Der eine Vergleich, der sich lohnt: Ein Container ist keine VM

Vielleicht fragen Sie sich, worin der Unterschied zu einer virtuellen Maschine besteht — der Technologie, die ein komplettes Gastbetriebssystem samt Kernel auf Ihrem eigenen laufen lässt. Die ehrliche Ein-Zeilen-Antwort: Ein Container ist eine **leichtgewichtige, isolierte Box, die sich den Kernel des Hosts teilt**, statt einen eigenen zu booten. Er startet daher in Sekundenbruchteilen und trägt nur die Anwendung mit sich, nicht ein ganzes Betriebssystem. Eine VM virtualisiert einen kompletten Computer; ein Container isoliert eine einzelne Anwendung. Das ist der einzige VM-Vergleich, den diese Serie braucht — wir sind nicht hier, um Virtualisierung zu lehren, sondern nur, um Container neben etwas zu stellen, das Sie bereits kennen.

Docker auf Windows installieren

Das Werkzeug, das wir installieren, ist Docker Desktop — die offizielle Anwendung, die die Docker Engine, die Kommandozeilenwerkzeuge und ein Verwaltungs-Dashboard in einem Windows-Installer bündelt. Unter Windows führt sie ihre Linux-Container mit WSL 2 aus, dem Windows Subsystem for Linux Version 2 — einem echten, von Microsoft gelieferten Linux-Kernel, den Windows für Sie im Hintergrund betreibt. (Docker Desktop kann alternativ das Hyper-V-Backend verwenden, aber WSL 2 ist der empfohlene Standard und das, was wir einrichten.)

> **Prüfen Sie zuerst die Systemvoraussetzungen**
> Laut Dockers Installationsdokumentation für Windows benötigt das WSL-2-Backend 64-Bit **Windows 10 22H2** (Build 19045) oder **Windows 11 23H2** (Build 22631) oder höher — in den Editionen Enterprise, Pro oder Education —, einen 64-Bit-Prozessor mit SLAT, mindestens **8 GB RAM**, im BIOS/UEFI aktivierte Hardware-Virtualisierung und **WSL Version 2.1.5 oder neuer**. Prüfen Sie die aktuellen Anforderungen unter dem Link, bevor Sie beginnen, denn sie ändern sich mit der Zeit.

Die Einrichtung ist eine kurze, geordnete Prozedur. Arbeiten Sie sie von oben nach unten ab.

1. WSL 2 aktivieren

Öffnen Sie PowerShell **als Administrator** (Rechtsklick ’ Als Administrator ausführen), führen Sie den einen Installationsbefehl aus und starten Sie neu, wenn Sie dazu aufgefordert werden. Laut Microsofts WSL-Installationsanleitung aktiviert dieser eine Befehl die erforderlichen Windows-Features und installiert eine Standard-Linux-Distribution:

```
wsl --install
```

Auf diesem Weg installierte Distributionen sind standardmäßig auf WSL 2 eingestellt. Falls WSL aus einer älteren Einrichtung bereits vorhanden war, machen Sie WSL 2 explizit zum Standard:

```
wsl --set-default-version 2
```

2. Docker Desktop installieren

Laden Sie den Installer von der offiziellen Docker-Desktop-Seite herunter, führen Sie `Docker Desktop Installer.exe` aus und stellen Sie auf der Konfigurationsseite sicher, dass die Option **WSL 2 backend** ausgewählt ist. Schließen Sie den Assistenten ab und akzeptieren Sie die Subscription-Vereinbarung.

3. Starten und prüfen, ob die Engine läuft

Starten Sie Docker Desktop über das Startmenü und warten Sie, bis das Wal-Symbol im Infobereich aufhört, sich zu bewegen — das bedeutet, dass die Engine läuft. Öffnen Sie dann ein Terminal (PowerShell oder Windows Terminal) und prüfen Sie, ob die CLI sie erreicht:

```
docker version
```

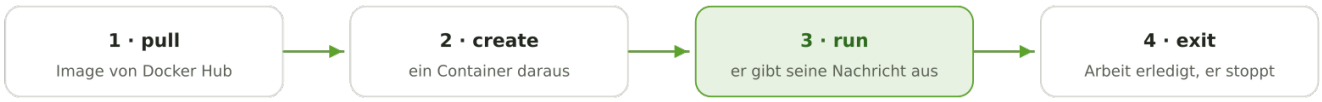
Sie sollten sowohl einen Abschnitt **Client** als auch einen Abschnitt **Server** sehen. Ist der Server-Abschnitt vorhanden, läuft die Engine, und Sie sind startklar.

Ihr erster Container: `hello-world`

Jetzt kommt der Moment, für den die ganze Installation da war. Führen Sie Dockers winziges offizielles Test-Image aus, das genau für diesen Zweck existiert:

```
docker run hello-world
```

Sie sehen eine kurze Nachricht, die mit "Hello from Docker!" beginnt — aber der interessante Teil ist, was Docker getan hat, um sie zu erzeugen. Es ist der Drei-Ideen-Kreislauf, live und in dieser Reihenfolge.



"Unable to find image locally" beim ersten Mal ist normal — das ist der Pull

Was ein einziges docker run tut: aus der Registry pullen, erstellen, ausführen, beenden

Der Ablauf im Klartext: Weil Sie `hello-world` noch nie ausgeführt hatten, konnte Docker das Image lokal nicht finden und hat es deshalb per **Pull** von Docker Hub geholt (Idee drei). Dann hat es aus diesem Image einen Container **erstellt** (Idee zwei, das Objekt aus der Klasse) und ihn **ausgeführt**. Die einzige Aufgabe des Containers war, seinen Gruß auszugeben — also hat er sich danach **beendet**. Die Zeile "Unable to find image locally", die Sie beim ersten Mal sehen, ist kein Fehler — sie ist der Pull selbst, und sie wird sich nicht wiederholen, denn das Image liegt jetzt auf Ihrem Rechner.

Eine Handvoll Befehle für mehr Sicherheit

Der Container hat sich beendet, aber er ist nicht verschwunden — und das Image auch nicht. Mit ein paar kurzen Befehlen sehen Sie, was `docker run` hinterlassen hat, und räumen es auf. Jeder tut genau eine klare Sache.

Die Tabelle nennt jeden Befehl und genau das, was er anzeigt oder tut.

Befehl	Was er tut
`docker ps`	Listet nur **laufende** Container. Direkt nach `hello-world` ist die Liste leer — er hat sich bereits beendet.
`docker ps -a`	Listet **alle** Container, auch gestoppte. Hier finden Sie Ihren `hello-world`-Container im Zustand `Exited`.
`docker images`	Listet die Images auf Ihrem Rechner — das gerade gezogene `hello-world`-Image ist eines davon.
`docker rm <name-or-id>`	

	Entfernt einen gestoppten Container über seinen Namen oder seine ID (beides finden Sie in `docker ps -a`). Das ist die "sauber aufräumen"-Hälfte des Versprechens.

Probieren Sie sie der Reihe nach aus: `docker ps` (leer), dann `docker ps -a` (da ist Ihr beendeter Container, mit einem automatisch vergebenen Namen wie `sleepy_bell`), dann `docker images` (da ist das gezogene Image), dann `docker rm` mit dem Namen des Containers, um ihn zu entfernen. Dieser letzte Befehl ist der Sinn der ganzen Übung — der Container lässt sich sauber entsorgen, und nach Teil 2 werden Sie dasselbe mit einem kompletten Postgres-Server tun.

> **Zwei Flags, die man früh kennen sollte**
> Sie brauchen sie noch nicht, aber es sind die beiden, nach denen Sie ständig greifen werden. `--rm` bei `docker run` weist Docker an, den Container automatisch zu löschen, sobald er sich beendet — kein manuelles `docker rm` mehr nötig. Und `-d` (detached) führt einen Container im Hintergrund aus, statt Ihr Terminal zu blockieren — genau so werden Sie im nächsten Beitrag einen langlebigen Server wie Postgres starten.

Klartext: Lizenzierung und freie Alternativen

Auf eines bestehe ich, bevor Sie irgendetwas auf einem Werkzeug aufbauen: Wissen Sie, was es kostet und was es sonst noch gibt. Docker Desktop ist für viele Menschen kostenlos — für manche Organisationen aber nicht —, und es gibt exzellente freie Open-Source-Alternativen. Hier ist das faire, vollständige Bild.

Laut Dockers Subscription-Bedingungen ist **Docker Desktop kostenlos** für den persönlichen Gebrauch, für Bildungszwecke, nichtkommerzielle Open-Source-Projekte und kleine Unternehmen — definiert als **weniger als 250 Mitarbeiter* und *weniger als 10 Millionen US-Dollar Jahresumsatz**. Größere Organisationen und Behörden benötigen ein kostenpflichtiges Abonnement (Pro, Team oder Business). Das ist ein wirklich großzügiges Gratis-Angebot, das die meisten einzelnen Entwickler und kleinen Firmen vollständig abdeckt — und wenn Sie in einem größeren Unternehmen arbeiten, finanzieren die Bezahlpläne die Pflege des Toolings und der riesigen öffentlichen Image-Bibliothek, auf die Sie sich die ganze Serie über stützen werden.

Es lohnt sich auch, präzise zu sein, *was* hier lizenziert wird. Die Bezahlpflicht gilt für **Docker Desktop, die komfortable gebündelte Anwendung**. Die zugrunde liegende Docker Engine und die `docker`-CLI sind Open Source (Apache 2.0), und das Container-Format selbst ist ein offener Standard unter der Führung der Open Container Initiative (OCI) — genau deshalb können die unten genannten Alternativen exakt dieselben Images ausführen.

Freie und Open-Source-Alternativen

Wenn die Lizenz von Docker Desktop nicht zu Ihrer Situation passt: Zwei ausgereifte Open-Source-Werkzeuge führen dieselben OCI-Container und -Images aus — exakt die von Docker Hub —, sodass alles in dieser Serie weiterhin gilt. Rancher Desktop (Open Source, von SUSE) bietet Container- und Kubernetes-Tooling auf Windows, macOS und Linux und kann sogar die Docker/Moby-Engine verwenden, sodass der vertraute `docker`-Befehl unverändert funktioniert. Podman Desktop (Open Source, ein CNCF-Projekt) ist eine herstellerneutrale GUI zum Bauen und Ausführen von OCI-Containern, ebenfalls plattformübergreifend. Wählen Sie nach Passung: Docker Desktop als polierter Alles-in-einem-Standard mit kommerziellem Support, Rancher oder Podman Desktop, wenn Sie einen vollständig quelloffenen Stack wollen. Keines davon ist ein Trostpreis — alle drei sind erstklassige Werkzeuge auf demselben offenen Standard.

Fazit

Sie haben das Tooling installiert und — wichtiger noch — Sie besitzen jetzt die drei Ideen, auf denen der gesamte Rest dieser Serie steht. Ein **Image** ist eine schreibgeschützte Vorlage — eine Klasse. Ein **Container** ist eine laufende Instanz davon — ein Objekt, geboren aus `docker run`. Eine **Registry** wie Docker Hub ist der Ort, von dem Images kommen — einmal pullen, beliebig oft starten. Und ein Container ist keine virtuelle Maschine: Er ist eine leichtgewichtige Box, die sich den Kernel Ihres Hosts teilt, deshalb startet er schnell und lässt sich sauber entfernen.

> Ein Container ist ein Objekt; ein Image ist seine Klasse; `docker run` ist der Konstruktor. Sobald das klick macht, hört Docker auf, ein Mysterium zu sein, und wird zum Werkzeug.

Diese Eigenschaft, sich "sauber entfernen" zu lassen, ist das Versprechen vom Anfang dieses Beitrags — und beim nächsten Mal lösen wir es richtig ein. In Teil 2: eine echte Postgres-Datenbank mit einem Befehl, laufend auf Ihrem Windows-Rechner, ohne dass nativ irgendetwas installiert wird — und eine Delphi-Anwendung, die mit FireDAC über die moderne RTL eine Tabelle erstellt, Zeilen einfügt und sie wieder ausliest. Bis dahin.

INDEPENDENT · AD-FREE · READER-SUPPORTED

Free to read. Not free to make.

Every tutorial, article, and line of example code on Holger's Code is published without a paywall and without ads. This page is about what keeps it that way.

Professional work, published free

The content here is held to the same standard as professional client work: every tutorial is built as a real project, tested against current versions, documented step by step, and revisited when the frameworks move on. A single in-depth tutorial routinely takes days — and that's before the bills behind it: servers, storage, build infrastructure, licenses, and the hardware everything is verified on.

If an article or tutorial here has saved you an afternoon of debugging, taught you a technique, or helped you ship, it created real value for you. Supporting it isn't charity — it's paying for professional work you found useful, at whatever price you choose. And if you can't, keep reading anyway. That's what the free part means.

Ways to support

KO-FI

Buy me a coffee

A one-time tip — no account, no subscription, any amount.

BOOKS & COURSES

Buy the paid work

The paid tier of everything published free here — and you get something lasting in return.

SPREAD THE WORD

Share an article

Send a tutorial to a colleague, cite an article, subscribe to the RSS feed.

Nothing here goes behind a paywall, whether you chip in or not. But if something on this site earned its keep today, this is where to say so.

Support on Ko-fi —