GANZ EINFACH

Container-Praxis mit Docker

Container verstehen und anwenden
Entwickeln, absichern und zuverlässig betreiben

Ein unabhängiges Praxisbuch



Danilo Sieren

Danilo Sieren

Container-Praxis mit Docker Ganz Einfach

Container verstehen und anwenden
Entwickeln, absichern und zuverlässig betreiben

Überarbeitete und erweiterte
Neuauflage 2026

Verstehen · Anwenden · Überprüfen

Vorwort

Docker hilft dir, Anwendungen mit einer definierten Laufzeitumgebung bereitzustellen. Die Arbeit wird dadurch nachvollziehbarer, aber nicht automatisch sicher, ausfallsicher oder wartungsfrei. Dieses Buch zeigt dir, wie du Container bewusst einsetzt, Daten dauerhaft aufbewahrst und Änderungen mit überprüfbaren Ergebnissen durchführst. Du lernst die Bedienung und die technischen Zusammenhänge, die du für verlässliche Entscheidungen brauchst.

Die Anleitung richtet sich an interessierte Einsteiger, Selbstständige, kleine Teams und Administratoren, die Docker gründlich verstehen möchten. Du brauchst keine frühere Containerpraxis. Grundkenntnisse im Umgang mit Dateien und einem Terminal helfen dir. Bei fortgeschrittenen Themen werden Linux, Netzwerke und Datenbanken ausführlicher behandelt. Ein einzelner Rechner genügt für den überwiegenden Teil der Übungen. Die Clusterkapitel verlangen zusätzliche getrennte Linux-Testsysteme.

Ein durchgehendes Lernprojekt begleitet dich: Du stellst einen kleinen statischen Informationsdienst bereit, ergänzt eine getrennte PostgreSQL-Lerndatenbank und entwickelst einen kontrollierten Betriebsablauf. Die Datenbank ist bewusst kein versteckter Bestandteil des Webbeispiels. Du lernst beide Dienste zunächst unabhängig und verbindest Anwendungen erst, wenn ihre jeweiligen Schnittstellen bekannt sind. So lassen sich Fehler eingrenzen, statt mehrere unbekannte Komponenten gleichzeitig zu verändern.

Redaktionsstand ist der 1. November 2026. Die offiziellen Releaseverzeichnisse nennen zu diesem Zeitpunkt Docker Engine 29.9.2, Docker Desktop 4.97.0 und Compose 5.6.1. Diese Produkte haben getrennte Versionsnummern; eine

Desktopinstallation kann andere gebündelte Einzelversionen enthalten. Das Buch verwendet den aktuellen Aufruf docker compose. Die Bezeichnung Compose V2 in älteren Texten oder Modulnamen ist kein Beleg, dass die installierte Erweiterung noch die Hauptversion 2 trägt. [D01, D02, D03]

Die Beispiele beschreiben Linux-Container. Unter macOS und beim üblichen Linux-Containerbetrieb mit Docker Desktop unter Windows läuft dazu eine Linux-Umgebung in einer virtuellen Maschine. Windows-Container sind ein eigener Betriebsmodus und werden nicht mit Linux-Beispielen gleichgesetzt. Die Installation erfolgt passend zum jeweiligen System; die späteren Übungen verwenden überwiegend dieselben Docker-Befehle.

Fachliche Aussagen wurden anhand der im Quellenverzeichnis genannten Originaldokumentationen überprüft. Befehle und Konfigurationsbeispiele werden zusätzlich auf innere Konsistenz und, soweit die verfügbaren Werkzeuge es erlauben, auf Syntax geprüft. Eine erfolgreiche Syntaxprüfung ersetzt keinen praktischen Test auf deiner Hardware. Nicht alle Betriebssysteme, Netzwerkvarianten und Clusterkonfigurationen wurden für diese Ausgabe praktisch bereitgestellt. Das Versionsregister im Anhang trennt dokumentierte Funktionen, didaktische Beispiele und diese Prüfgrenzen.

So arbeitest du mit diesem Buch

Lies die ersten sechs Kapitel in ihrer Reihenfolge. Sie vermitteln die Begriffe, die Installation, den Containerlebenszyklus und den Aufbau eigener Images. Anschließend kannst du Speicher, Netzwerke und Compose zu einem vollständigen Lernprojekt zusammenführen. Sicherheit, Beobachtbarkeit, Wiederherstellung und kontrollierte Updates gehören zusammen: Eine Anwendung ist erst dann sinnvoll betreibbar, wenn du ihr Verhalten prüfen und ihre Daten zurückholen kannst.

Die Kapitel zu CI, Swarm und Infrastruktur als Code bauen darauf auf. Sie sollen dir eine begründete Auswahl ermöglichen. Ein kleines Projekt benötigt nicht allein deshalb einen Cluster, weil Docker ihn unterstützt. Zusätzliche Werkzeuge bringen zusätzliche Zustände, Rechte und Fehlerquellen mit. Du prüfst jeweils, welches Problem sie für dich lösen und welche Betriebsarbeit dadurch entsteht.

Befehlsblöcke enthalten keine vorangestellten Dollarzeichen als Eingabeaufforderung. Du kannst die Befehle dadurch leichter übernehmen. Ein Backslash am Zeilenende setzt einen Shellbefehl in der nächsten Zeile fort. Hinter diesem Zeichen darf kein Leerzeichen stehen. Die Shellbeispiele gelten für eine POSIX-kompatible Shell unter Linux, macOS oder WSL. Die PowerShell hat andere Fortsetzungs- und Anführungszeichenregeln. Unter Windows führst du solche Blöcke deshalb in deiner eingerichteten WSL-Shell aus.

Dateinamen und Projektnamen bleiben in den zusammengehörigen Übungen gleich. Ein neues Kapitel kann ein eigenes Verzeichnis verlangen; dies steht dann ausdrücklich dabei. Ein als Ausschnitt bezeichneter Block ergänzt eine

vorhandene Datei und ist keine eigenständig startbare Konfiguration. Alle übrigen angegebenen Dateien werden vollständig wiedergegeben oder auf klar benannte Teile verteilt, die du in derselben Datei zusammenfügst.

Verwende für die Übungen ausschließlich eine getrennte Lernumgebung ohne produktive Daten. Mehrere Übungen stoppen oder ersetzen genau benannte Testcontainer. Prüfe vor jedem solchen Schritt deinen Docker-Kontext. Ein lokales Terminal kann einen entfernten Server steuern. Das Arbeitsverzeichnis allein schützt dich nicht vor einer Verwechslung.

Die Praxisaufgaben enden mit einer fachlichen Abnahme: Du hältst fest, was du erwartet hast, was tatsächlich geschehen ist und welche Beobachtung das belegt. Ein grüner Status reicht beispielsweise nicht als Beleg, dass eine wiederhergestellte Datenbank alle benötigten Datensätze enthält. Diese Unterscheidung zwischen technischem Zustand und fachlichem Ergebnis zieht sich durch das gesamte Buch.

Inhalt

Vorwort 2

So arbeitest du mit diesem Buch 4

1 Container und ihre Grenzen verstehen 8

2 Die passende Docker Umgebung auswählen 14

3 Terminal Contexts und erste Container 20

4 Lebenszyklus Signale und Laufzeitkonfiguration 26

5 Eigene Images nachvollziehbar bauen 32

6 BuildKit Cache und mehrere Plattformen 38

7 Daten mit Volumes und Mounts erhalten 44

8 Netzwerke und Portfreigaben nachvollziehen 51

9 Compose Projekte sauber aufbauen 57

10 Compose Variablen Abhängigkeiten und Entwicklung 64

11 Datenbanken kontrolliert im Container betreiben 69

12 Veröffentlichungen Proxy und Hostfirewall 75

13 Rechte Rootless und Laufzeitsicherheit 80

14 Imageherkunft Signaturen und Lieferkette 86

15 Logs und systematische Fehleranalyse 91

16 Ressourcen Monitoring und Leistung 96

17 Sicherung Wiederherstellung und Speicherpflege 101

18 Updates Migrationen und Rückkehr planen 110

19 Tests CI und kontrollierte Auslieferung 116

20 Swarm und verteilte Dienste 123

21 Infrastruktur als Code und entfernte Verwaltung 130

22 Ein vollständiges Projekt fachlich abnehmen 136
Anhang A Vollständige Dateien für das Lernlabor 141
Anhang B Betriebsfälle ausführlich durchspielen 149
Anhang C Wiederholbare Prüfungen und Betriebsabläufe 156
Anhang D Begriffe und Entscheidungshilfen 164
Anhang E Versionsstand und Prüfgrenzen 170
Quellen und offizielle Dokumentation 173
Ausführliches Inhaltsverzeichnis 177
Stichwortverzeichnis 185
Impressum 187

1 Container und ihre Grenzen verstehen

1.1 Welche Probleme Container lösen

Eine Anwendung hängt von Bibliotheken, Laufzeitversionen, Dateien und Startparametern ab. Wenn du nur ihren Quelltext auf einen anderen Rechner kopierst, übernimmst du diese Umgebung nicht automatisch. Ein Image bündelt einen definierten Teil davon. Der Container startet aus diesem Image mit einer zusätzlich festgelegten Konfiguration. Dadurch kannst du dieselbe Anwendung unter vergleichbaren Voraussetzungen prüfen und verteilen.

Ein Beispiel: Du betreibst einen kleinen Informationsdienst, der HTML-Dateien ausliefert. Das Image legt den Webserver und seine Laufzeitdateien fest. Die Containerkonfiguration bestimmt, an welcher Adresse du den Dienst erreichst und ob die Dokumente aus dem Image oder aus einem eingebundenen Verzeichnis kommen. Diese beiden Ebenen solltest du von Anfang an unterscheiden. Ein neues Image ändert nicht automatisch eine extern eingebundene Datei.

Container beseitigen nicht alle Unterschiede zwischen Rechnern. Architektur, Kernel, Geräte, verfügbare Ressourcen, Netzwerkrouten und Dateisystemrechte bleiben relevant. Ein Test auf deinem Notebook beweist deshalb nicht, dass derselbe Dienst unter höherer Last auf einem Server zuverlässig läuft. Docker verbessert die Beschreibung der Umgebung; deine Abnahme prüft weiterhin die tatsächliche Verwendung.

1.2 Image und Container auseinanderhalten

Das Image ist ein adressierbares Softwarepaket. Ein Container ist eine daraus erzeugte Instanz mit eigenem Zustand. Aus einem

Image kannst du mehrere Container erzeugen. Sie können unterschiedliche Namen, Portfreigaben, Ressourcenlimits und Mounts besitzen. Ihre Änderungen landen nicht automatisch zurück im Image. [D43]

Wenn du einen Container stoppst, bleibt er zunächst als Objekt erhalten. Seine beschreibbare Schicht verschwindet dadurch nicht. Entfernst du ihn, geht diese Schicht verloren. Ein separat verwaltetes Volume besitzt einen anderen Lebenszyklus. Diese Unterscheidung erklärt, weshalb ein Neustart häufig scheinbar funktioniert, während das Ersetzen des Containers später ungesicherte Daten verschwinden lässt.

1.3 Prozesse und Kernelisolation

Ein Linux-Container führt Prozesse unter Kontrolle des Linux-Kernels aus. Namespaces begrenzen unter anderem die sichtbaren Prozesse, Mounts und Netzwerkschnittstellen. cgroups ermöglichen die Zuordnung und Begrenzung von Ressourcen. Zusätzliche Mechanismen wie Capabilities, Seccomp und Sicherheitsmodule können die Handlungsmöglichkeiten einschränken. Keiner dieser Bausteine ist für sich eine umfassende Sicherheitsgarantie. [D07]

Wichtig ist insbesondere der Benutzer-Namespace. Eine UID im Container wird nicht in jeder Standardinstallation auf eine unprivilegierte Host-UID abgebildet. Du musst unterscheiden, ob du normalen Root-Betrieb, userns-remap oder Rootless Docker verwendest. Die Aussage, Container-root sei grundsätzlich ein harmloser Benutzer auf dem Host, wäre falsch. Umgekehrt besitzt Container-root wegen weiterer Einschränkungen nicht automatisch alle Rechte eines uneingeschränkten Host-root.



1.4 Virtuelle Maschinen einordnen

Eine virtuelle Maschine besitzt gewöhnlich einen eigenen Gastkernel. Ein Linux-Container nutzt den Kernel der Umgebung, in der seine Laufzeit arbeitet. Das kann ein physischer Linux-Host oder der Linux-Gast von Docker Desktop sein. Auf einem Mac ist der Kernel deiner Linux-Container deshalb nicht der macOS-Kernel. Hardwarevirtualisierung spielt dort für die Linux-VM eine Rolle, während eine native Linux Engine grundsätzlich keine VM für gewöhnliche Linux-Container voraussetzt.

VMs und Container lassen sich kombinieren. Für getrennte Vertrauensbereiche können mehrere VMs sinnvoll sein, in denen jeweils Container laufen. Eine solche Aufteilung kostet zusätzliche Ressourcen, bietet aber eine andere Isolationsgrenze. Entscheide nach Schutzbedarf und Betriebsaufwand, statt eine Technik pauschal als überlegen darzustellen.

Tabelle 1.1 Die wichtigsten Ebenen

Ebene	Aufgabe	Bleibt zu prüfen
Image	Software und Vorgaben	Herkunft und Aktualität
Container	Laufende Instanz	Rechte und Ressourcen
Volume	Dauerhafte Daten	Sicherung und Besitz
Host oder VM	Kernel und Geräte	Patches und Kapazität

1.5 OCI und Portabilität

Standards für Containerformate und Laufzeitverhalten erleichtern die Nutzung von Images mit verschiedenen Werkzeugen. Daraus folgt keine vollständige Austauschbarkeit jeder Konfiguration. Ein Docker-Compose-Projekt ist

beispielsweise keine automatisch lauffähige Kubernetes-Beschreibung. Ein Linux-amd64-Image benötigt auf einem ARM-Rechner eine passende Alternative oder eine geeignete Emulationsumgebung.

Prüfe Portabilität daher in mehreren Schritten. Zunächst muss das Artefakt für das Ziel verfügbar sein. Dann müssen Startparameter, Sicherheitsfunktionen und Mounts unterstützt werden. Abschließend prüfst du das fachliche Verhalten. Auch ein erfolgreich gestarteter Prozess kann falsche Zeitzonen, fehlende Zertifikate oder ungeeignete Speicherrechte haben.

1.6 Monolithen und mehrere Dienste

Docker verlangt keine Microservicearchitektur. Du kannst eine zusammenhängende Anwendung in einem Container betreiben, wenn sie sich sinnvoll als Einheit starten, überwachen und aktualisieren lässt. Eine Datenbank und ein Webserver sind häufig getrennte Dienste, weil sie verschiedene Daten- und Wartungsanforderungen haben. Die Trennung allein erzeugt jedoch noch keine Ausfallsicherheit.

Jede zusätzliche Schnittstelle benötigt Fehlerbehandlung. Wenn ein Dienst auf eine Datenbank wartet, muss er mit zeitweisen Verbindungsfehlern umgehen. Ein vorgeschalteter Healthcheck reduziert Startprobleme, ersetzt diese Logik aber nicht. Für ein kleines Team kann eine einfachere Architektur mit gutem Restoreverfahren zuverlässiger sein als ein kompliziertes System ohne klare Zuständigkeiten.

1.7 Praxisprojekt Eine Anwendung beschreiben

Beschreibe einen Dienst, den du tatsächlich nutzen möchtest, zunächst ohne Installationsbefehle. Notiere seine Aufgabe, seine Benutzer und die Daten, die er dauerhaft benötigt. Ergänze alle eingehenden und ausgehenden Verbindungen. Trenne dabei die öffentliche Bedienoberfläche von Administrationszugängen. Ein

Verwaltungsport ist nicht automatisch Teil des normalen Nutzerzugangs.

Ordne anschließend jede Information einer Ebene zu: Software im Image, Konfiguration beim Start, dauerhafte Daten im Speicher und Voraussetzungen des Hosts. Wenn du eine Datei nicht sicher zuordnen kannst, ist das eine konkrete offene Frage für die Dokumentation des Anwendungsherstellers. Diese Vorarbeit verhindert später die verbreitete Annahme, ein einziges Containerimage enthalte bereits den gesamten Betriebszustand.

Die Aufgabe ist abgeschlossen, wenn du erklären kannst, welche Bestandteile du nach dem Verlust des Rechners erneut beschaffen könntest und welche du aus einer Sicherung zurückholen müsstest. Diese Beschreibung dient dir in den Backupkapiteln als erste Prüfliste.

1.8 Vertiefung Ein Ausfallmodell erstellen

Betrachte einen kleinen Dokumentendienst mit Weboberfläche und Datenbank. Für seine Benutzer zählt, dass sie ein Dokument öffnen und eine Änderung wiederfinden können. Der technische Containerzustand bildet nur einen Teil davon ab. Das Netz kann ausfallen, die Anmeldung kann fehlschlagen oder der Datenträger kann keine neuen Schreibvorgänge mehr annehmen, obwohl beide Prozesse weiterhin laufen.

Erstelle eine Liste mit den Ausfällen einzelner Komponenten. Beginne bei der Anwendung und arbeite nach außen: Prozess, Container, Engine, VM, Host, Stromversorgung und externer Zugang. Ergänze gemeinsam genutzte Dienste wie DNS und Zeitquelle. Für jeden Ausfall beschreibst du die sichtbare Folge und deinen Wiederanlaufweg. So erkennst du, welche vermeintlich unabhängigen Container tatsächlich denselben kritischen Unterbau teilen.

Die Liste zeigt auch, wo zusätzliche Replikate helfen. Zwei Webcontainer können den Ausfall eines Prozesses abfangen, benötigen aber weiterhin einen geeigneten Zugriffspfad und verfügbare Daten. Wenn beide auf derselben defekten SSD liegen, schützt ihre Anzahl nicht vor dem Verlust des Hosts. Eine räumliche Verteilung löst wiederum keine falsch replizierten Daten oder gelöschten Inhalte.

Lege abschließend fest, welche Ausfälle du im Labor praktisch prüfen kannst. Ein geordneter Prozessstop lässt sich einfach wiederholen. Eine echte Beschädigung produktiver Daten wird dagegen nicht als beiläufiger Test erzeugt. Dafür verwendest du separate Kopien und ein geplantes Szenario. Die Qualität des Modells liegt in seiner überprüfbaren Begrenzung, nicht in möglichst dramatischen Fehlerbezeichnungen.

2 Die passende Docker Umgebung auswählen

2.1 Engine Desktop und Erweiterungen

Docker Engine stellt die Laufzeit und ihre API bereit. Das CLI übermittelt Befehle an eine Engine. Docker Desktop bündelt eine Entwicklungsumgebung mit Oberfläche und verschiedenen Komponenten. Compose beschreibt mehrere zusammengehörige Dienste. Buildx steuert BuildKit-basierte Builds. Prüfe diese Teile getrennt, wenn du eine Anleitung oder Fehlermeldung auswertest.

Eine aktuelle Desktopversion beweist nicht, dass ein entfernter Server dieselbe Engineversion verwendet. Der Befehl docker version zeigt Client- und Serverangaben getrennt. docker compose version nennt die tatsächlich verwendete Compose-Erweiterung. Gerade bei einem Update solltest du diese Ausgaben zusammen aufbewahren, damit du später erkennst, welche Kombination sich verändert hat.

2.2 Einen Lernrechner planen

Für den Einstieg brauchst du freien Speicher für Images, Containerdaten, Buildcache und Sicherungen. Die Mindestanforderung eines Installers ist kein Kapazitätsplan. Ein Rechner mit mehreren Entwicklungswerkzeugen, einer Datenbank und einer Linux-VM benötigt mehr Reserven als ein einzelner Testcontainer. Beobachte nach den ersten Übungen die tatsächliche Belegung und lege ein nachvollziehbares Limit für deine Testdaten fest.

Der Rechner sollte keine ungesicherten wichtigen Daten als einzige Kopie enthalten. Du musst ihn während einer Übung neu starten oder Testobjekte löschen können, ohne deine tägliche

Arbeit zu gefährden. Für Linux-Serverübungen ist eine separate VM häufig praktisch. Ein Snapshot kann einen Versuch zurücksetzen, ersetzt aber keine externe Sicherung deiner Arbeit.

2.3 Installation auf dem Mac

Lade den offiziellen Installer passend zu Apple-Chip oder Intel-Prozessor. Prüfe die unterstützten macOS-Versionen und den Herausgeber, bevor du das Programm installierst. Docker nennt die aktuelle und die beiden vorherigen macOS-Hauptversionen als Unterstützungsrahmen; konkrete Anforderungen können sich mit einer Desktopausgabe ändern. Rosetta ist nicht pauschal Voraussetzung für alle ARM-nativen Container. [D05]

Starte Docker Desktop und warte, bis die Engine bereit ist. Öffne dann ein neues Terminalfenster und kontrolliere die Versionsbefehle aus Abschnitt 2.6. Über das Docker-Menü beziehungsweise das Einstellungssymbol gelangst du zu Settings. Unter Resources findest du die für dein Backend verfügbaren Ressourcen- und Dateifreigabeoptionen. Die sichtbaren Unterpunkte hängen von Plattform und Backend ab. [D38]

Eine von macOS freigegebene Datei liegt nicht automatisch auf einem entfernten Docker-Host. Auch Desktopfreigaben gelten nur für die konkrete Desktopumgebung. Wenn ein Mount fehlschlägt, prüfst du zuerst den Kontext und danach den freigegebenen Pfad. Das verhindert unnötige Rechteänderungen an deinem persönlichen Verzeichnis.

2.4 Installation unter Windows

Für die Linux-Beispiele dieses Buchs verwendest du eine unterstützte Docker-Desktop-Installation mit einem geeigneten Linux-Backend. WSL 2 ist ein verbreiteter Weg. Die aktuelle Installationsanleitung unterscheidet außerdem benutzerbezogene und systemweite Installationsarten. Windows-Container benötigen andere Voraussetzungen und werden nicht

von jeder Installationsart unterstützt. Prüfe deshalb die offizielle Tabelle für dein System, statt ältere Anleitungen ungeprüft zu übernehmen. [D06]

Richte die WSL-Integration für die gewünschte Distribution ein und prüfe Docker innerhalb genau dieser Shell. Bewahre deine Linux-Projektdateien möglichst an einem für diesen Arbeitsablauf geeigneten Ort auf. Ein Projekt über verschiedene Windows- und Linux-Dateisystemgrenzen hinweg kann andere Leistungs- und Rechtebedingungen besitzen. Erzeuge zunächst ein kleines Testverzeichnis und kontrolliere Lesen, Schreiben und Dateinamen, bevor du größere Projekte übernimmst.

2.5 Installation auf Linux

Verwende die Installationsanleitung für deine Distribution und Version. Das Docker-Repository und ein Distributionspaket sind unterschiedliche Bezugswege. Vermische ihre Pakete und Anleitungen nicht beliebig. Prüfe auf einem bereits genutzten Host zuerst, ob vorhandene Docker-, containerd- oder runc-Pakete von anderen Anwendungen benötigt werden. Die in einer Installationsanleitung aufgeführte Entfernung widersprechender Pakete ist kein Auftrag, sie auf jedem bestehenden Server ungeprüft zu löschen. [D04]

Für eine neue Ubuntu-Lernmaschine führt der dokumentierte Weg über das offizielle apt-Repository, dessen Signierschlüssel und die passenden Paketquellen. Installiert werden Engine, CLI, containerd sowie die Buildx- und Compose-Erweiterung. Übernimm die Repositoryanweisung unmittelbar aus der aktuellen Quelle, damit Distributionskennung, Schlüsseldatei und Paketarchitektur zusammenpassen. Dieses Buch liefert dafür keine veraltende Kopie eines langen Installationsskripts.

Wähle bewusst, wie du den Daemon administrierst. Die Mitgliedschaft in der docker-Gruppe ist bei einer normalen Root-Engine eine weitreichende Berechtigung. Sie macht den Benutzer

nicht zu einem eingeschränkten Containeranwender. Für deinen einzelnen Lernhost kannst du entweder gezielt mit administrativer Berechtigung arbeiten oder Rootless Docker einrichten. Die Sicherheitsfolgen werden in Kapitel 13 erläutert.

2.6 Installation prüfen

Führe nach der Einrichtung diese Bestandsaufnahme aus. Die Befehle verändern keine Containerkonfiguration:

```
docker version
docker info
docker compose version
docker buildx version
docker context ls
```

Client- und Serverabschnitt sollten erreichbar sein. Eine reine Clientausgabe mit Verbindungsfehler bedeutet, dass der Zugriff auf die Engine noch nicht funktioniert. Auf einem Desktop prüfst du, ob Docker Desktop gestartet ist. Unter Linux kontrollierst du den Dienst und den ausgewählten Socket. Ändere nicht vorschnell dessen Dateirechte; eine falsche Gruppe oder ein falscher Kontext lässt sich gezielt beheben.

2.7 Lizenz und Zuständigkeit klären

Engine und Docker Desktop sind hinsichtlich Bezug und Nutzungsbedingungen nicht dasselbe Produkt. Prüfe vor einem betrieblichen Einsatz die aktuellen Bedingungen der tatsächlich genutzten Desktopausgabe und gegebenenfalls weiterer Dienste. Verwechsle die technische Möglichkeit einer Installation nicht mit einer organisatorischen Freigabe. Dieses Buch nennt bewusst keine dauerhaften Preis- oder Tarifgarantien.

Notiere für deinen eigenen Rechner, wer Updates ausführt und wo deine Projektdaten liegen. Bei einem gemeinsam genutzten Server kommen Zugangskontrolle, Vertretung und Wartungsfenster hinzu. Selbst in einem kleinen Team verhindert

eine knappe Zuständigkeitsliste, dass mehrere Personen unabhängig dieselbe Engine umkonfigurieren.

2.8 Werkstatt Eine Bestandsaufnahme anlegen

Erstelle ein Textdokument mit Betriebssystem, CPU-Architektur, Installationsweg, Engineversion, Composeversion und Kontextname. Ergänze den Ort deiner Projektdateien sowie die verfügbaren Speicherreserven. Schreibe auch auf, ob die Engine in einer VM oder unmittelbar auf Linux läuft. Diese Angabe hilft dir später, Meldungen über Kernel oder Speicher richtig zuzuordnen.

Beende und starte deine Entwicklungsumgebung einmal kontrolliert. Prüfe danach erneut docker version. Diese einfache Wiederholung zeigt dir, ob dein System nur im momentanen Zustand funktioniert oder ob der Startablauf verstanden ist. Eine Fehlermeldung gehört mit Datum und Ursache in deine Notiz, sobald du sie gelöst hast. So wird aus einer einmaligen Installation eine nachvollziehbare Grundlage für alle weiteren Kapitel.

2.9 Vertiefung Kapazität vor der Installation rechnen

Nimm für eine erste Planung beispielhafte Werte: 8 GB für Betriebssystem und sonstige aktive Arbeit, 4 GB für die Linux-Umgebung und 3 GB für gleichzeitig laufende Lerndienste. Diese Werte sind ein Rechenbeispiel und keine Mindestanforderung. Bereits ihre Summe zeigt, dass die bloße Anzahl installierter Programme wenig über die benötigte Reserve aussagt. Du musst den gleichzeitig beanspruchten Speicher betrachten.

Beim Datenträger planst du getrennt Quelltexte, Imagecache, entpackte Images, Volumedaten und Backups. Ein Build kann vorübergehend mehr Platz benötigen als sein fertiges Ergebnis. Ein Restore benötigt außerdem einen zweiten Datenbestand. Wenn du für die Wiederherstellungsübung nur den Platz des

ursprünglichen Dienstes reservierst, kannst du den entscheidenden Test möglicherweise gar nicht ausführen.

Notiere für jede Position Ausgangswert, angenommene Zunahme und Prüftermin. Die Datenbank könnte im Labor nur wenige Megabyte enthalten, während ein späterer realer Import mehrere Gigabyte belegt. Ersetze die Annahmen nach den ersten Messungen durch beobachtete Werte. Halte trotzdem eine zusätzliche Reserve für Wartungsarbeiten und unerwartete Datenmengen vor.

Prüfe auf einem Notebook auch den Schlafzustand. Ein lokal funktionierender Dienst kann für andere Nutzer verschwinden, sobald der Rechner schläft oder das Netz wechselt. Docker macht einen persönlichen Arbeitsplatzrechner nicht automatisch zu einer dauerhaft verfügbaren Serverplattform. Wenn du einen solchen Betrieb beabsichtigst, gehören Energieversorgung, Netz und Wiederanlauf in die Planung.

3 Terminal Contexts und erste Container

3.1 Befehle bewusst lesen

Ein Docker-Befehl besteht aus dem Programm docker, einer Objektgruppe und einer Aktion. docker container ls listet Container; docker image ls listet Images. Kürzere Formen wie docker ps sind gebräuchliche Aliasse. Nutze zunächst die ausführlichen Objektgruppen, wenn dir dadurch klarer wird, woran ein Befehl arbeitet. Vor einer Löschaktion ist diese Klarheit wichtiger als wenige gesparte Zeichen.

Optionen vor dem Imagenamen gehören bei docker run zur Containerkonfiguration. Wörter nach dem Imagenamen ersetzen oder ergänzen den im Image vorgesehenen Startaufruf. Die genaue Wirkung hängt von ENTRYPOINT und CMD des Images ab. Wenn du einen Anwendungsparameter versehentlich vor den Imagenamen setzt, kann Docker ihn als eigene Option auswerten. Prüfe bei einer unerwarteten Meldung deshalb die Position der Argumente.

3.2 Den Zielkontext kontrollieren

Ein Kontext fasst Verbindungsinformationen zu einer Engine zusammen. Er kann auf deinen Rechner, auf Docker Desktop oder auf eine entfernte Engine zeigen. Zusätzliche Umgebungsvariablen und explizite Optionen können die Auswahl beeinflussen. Kontrolliere vor Eingriffen daher nicht nur den aktuellen Verzeichnisnamen, sondern auch das wirkliche Ziel. [D44]

```
docker context show
docker context ls
docker version
```