

Migration von Web-Entwicklungsumgebungen in virtuelle Container auf Basis von Docker

BACHELORARBEIT

für die Prüfung zum
Bachelor of Science

des Studiengangs Angewandte Informatik
an der Dualen Hochschule Baden-Württemberg Mosbach

von

Jonas Flührer

6. September 2021

Bearbeitungszeitraum 12 Wochen

Matrikelnummer, Kurs 9752762, INF18A

Ausbildungsfirma Hochwarth IT GmbH, 74924 Neckarbischofsheim

Betreuer der Ausbildungsfirma Anna-Lisa Helter

Gutachter der Dualen Hochschule Edward Riewe

Erklärung

Ich versichere hiermit, dass ich meine Bachelorarbeit mit dem Thema: „*Migration von Web-Entwicklungsumgebungen in virtuelle Container auf Basis von Docker*“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Ort, Datum

Unterschrift

Kurzfassung

Um an einer Webanwendung eines Kunden ein Feature hinzufügen oder eine Änderung vornehmen zu können, muss die Code-Basis dieses Projekts lokal auf dem Computer des Entwicklers vorliegen. Zu jedem Projekt gehören verschiedene Abhängigkeiten, wie z.B. ein Webserver und seine installierten Module, eine [PHP¹](#)-Instanz und seine erforderlichen Erweiterungen oder eine Datenbank-Engine und die zugehörigen globalen Einstellungen und noch viele weitere. Es kann sehr viel Zeit in Anspruch nehmen, all diese Abhängigkeiten lokal zu installieren und zu konfigurieren, um dann erst am Projekt arbeiten zu können.

Das Ziel dieser Arbeit ist die Erstellung eines „Template-Repositorys“, das bei der Migration von Kundenprojekten in sogenannte „Container“ mit der Container-Virtualisierungssoftware „Docker“ helfen soll. Dieses Template-Repository besteht aus Konfigurationsdateien der einzelnen Software-Komponenten mit sinnvollen Standardwerten, Skript-Dateien, die beim automatisierten Konfigurieren der Software helfen und einer Dokumentation für die Benutzung des Repositorys. Damit können viele Abhängigkeiten eines Kundenprojekts festgehalten und die Einrichtung solcher Projekte lokal auf dem Computer automatisiert werden.

Bevor das Template-Repository erstellt werden kann, müssen in einer Anforderungsanalyse zuerst die Prozesse gefunden werden, die beim Einrichten eines Kundenprojekts automatisiert werden können. Danach können die Konfigurationsdateien erstellt, die Skripte programmiert und getestet und die Dokumentation geschrieben werden. Zum Schluss folgt ein Leistungsvergleich einiger verwendeter Software-Komponenten und ein Kosten-Vergleich der alten und neuen Lösung.

Durch die Verwendung des Template-Repositorys ist eine beschleunigte Reproduzierbarkeit von Kundenprojekten möglich. Diese Zeiteinsparung mindert auf lange Sicht auch Kosten. Die Verwendung des Template-Repositorys erhöht auch die [DX²](#). Außerdem können durch die Angleichung der Entwicklungsumgebung an die Produktivumgebung Fehler im Kundenprojekt selbst vermieden werden.

¹ PHP: Hypertext Preprocessor

² Developer Experience

Abstract

In order to add a feature or make a change to a client's web application the code base of this project must be available locally on the developer's computer. Each project has various dependencies, such as a web server and its installed modules, a PHP³ instance and its required extensions, or a database engine and its global settings, and many more. It can take a lot of time to install and configure all these dependencies locally before actually being able to work on the project.

The goal of this work is to create a „template repository“ to help migrate customer projects into so-called „containers“ using the container virtualisation software „Docker“. This template repository consists of configuration files of the individual software components with reasonable default values, script files that help with the automated configuration of the software and documentation for the use of the repository. This can be used to capture many of the dependencies of a client project and automate the setup of such projects locally on the computer.

Before the template repository can be created, a requirements analysis must first find the processes that can be automated when setting up a client project. Then the configuration files can be created, the scripts programmed and tested, and the documentation written. Finally, there is a performance comparison of some of the software components used and a cost comparison of the old and new solution.

By using the template repository, an accelerated reproducibility of customer projects is possible. This time saving also reduces costs in the long run. The use of the template repository also increases the DX⁴. In addition, by aligning the development environment with the productive environment, errors in the customer project itself can be avoided.

³ PHP: Hypertext Preprocessor

⁴ Developer Experience

Inhaltsverzeichnis

Abkürzungsverzeichnis	vii
Abbildungsverzeichnis	ix
Tabellenverzeichnis	x
Code-Verzeichnis	xi
1 Einleitung	1
1.1 Rahmen der Arbeit	1
1.2 Aufgabenstellung	1
1.3 Vorgehensweise	2
1.4 Motivation	3
2 Technische Grundlagen	4
2.1 Docker	4
2.1.1 Virtualisierung in Containern	4
2.1.2 Architektur	6
2.1.3 Dockerfile	10
2.1.4 Volumes	12
2.1.5 Ports	14
2.1.6 Netzwerk	14
2.1.7 Docker unter Windows	16
2.2 Docker Compose	16
2.3 Git	19
2.4 Shell-Umgebungen und Bash	21
2.5 MariaDB Server	25
2.6 Apache Webserver	26
2.7 <code>mkcert</code> zur Generierung von Zertifikaten	28
2.8 Die <code>hosts</code> -Datei	28
2.9 PHP	29
2.9.1 Xdebug	30

2.9.2 Composer	31
2.10 npm und Grunt	33
2.11 MailHog	34
3 Praktische Lösung	36
3.1 Anforderungsanalyse	36
3.2 Funktionen des Template-Repositorys	40
3.2.1 Projekt in Container verschieben	42
3.2.2 Projekt in Containern nutzen	48
3.3 Dokumentation	58
3.4 Analyse der Performance	59
3.4.1 Docker Hyper-V- oder WSL2-Backend	59
3.4.2 Auswirkung von <code>opcache</code> auf die Performance	60
3.4.3 Vergleich XAMPP-Umgebung und Docker-Umgebung	61
3.5 Kosten-Analyse	62
4 Schlussbemerkungen	66
4.1 Kritische Reflexion	66
4.2 Ausblick	67
Anhänge	69
Anhang A: README.md	69
Literaturverzeichnis	81

Abkürzungsverzeichnis

API Application Programming Interface

APT Advanced Packaging Tool

CA Certificate Authority

CLI Command Line Interface

CPU Central Processing Unit

CSS Cascading Style Sheets

DBGp Debugger Protocol

DBMS Datenbankmanagementsystem

DNS Domain Name System

DX Developer Experience

FPM FastCGI Process Manager

GNU GNU's Not Unix

GUI graphical user interface

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

I/O Input/Output

IDE integrated development environment

IEEE Institute of Electrical and Electronics Engineers

IP Internetprotokoll

JSON JavaScript Object Notation

MB Megabyte

npm Node Package Manager

OCI Open Container Initiative

pcre Perl Compatible Regular Expression

PHP PHP: Hypertext Preprocessor

POSIX Portable Operating System Interface

RAM Random Access Memory

REST Representational State Transfer

SCSS Sassy Cascading Style Sheets

SMTP Simple Mail Transfer Protocol

SQL Structured Query Language

SSL Secure Sockets Layer

TLS Transport Layer Security

URL Uniform Resource Locator

VCS Version Control System

VM Virtuelle Maschine

WSL Windows Subsystem for Linux

YAML YAML Ain't Markup Language

Abbildungsverzeichnis

2.1	Vergleich VM und Container	5
2.2	Container in VMs	7
2.3	Docker-Architektur	7
2.4	Funktionsweise von Docker-Volumes	13
2.5	Docker Bridge-Netzwerk	15
2.6	Funktionsweise eines Webservers	26
2.7	Funktionsweise des PHP-Interpreter	29
2.8	Verbindlungsaufbau von Xdebug und IDE für Step Debugging	31
2.9	Web-Oberfläche von MailHog	35
3.1	Verzeichnisstruktur des Template-Repositorys	41
3.2	Projekt-Architektur der Entwicklungsumgebung für das Template-Repository .	41
3.3	Debugging-Schritte einer PHP-Webanwendung mit dem Template-Repository .	55
3.4	Vergleich der Ladezeit mit und ohne die PHP-Erweiterung <code>opcache</code>	61
3.5	Vergleich der Ladezeiten mit XAMPP-Umgebung und mit Docker-Umgebung .	62
3.6	Kosten-Vergleich eines Docker-Setups und eines lokalen Setups in den ersten zwei Jahren	64
4.1	Deployment-Setup auf kompatiblem Produktiv-Server	68

Tabellenverzeichnis

2.1	Regex-Zeichenfolgen und ihre Funktionen	25
3.1	Vergleich der Ladezeiten bei Docker Hyper-V- und WSL2-Backend	60
3.2	Kosten-Analyse Docker-Setup und altes Setup	64

Code-Verzeichnis

2.1	Befehlsabfolge zur Aktualisierung eines Datenbank-Containers	9
2.2	Beispielhafter Inhalt einer Dockerfile	11
2.3	Konsolenausgabe eines <code>docker build</code> -Befehls	12
2.4	Ausgabe eines Docker Compose-Befehls	17
2.5	Beispiel einer <code>docker-compose.yml</code> -Datei	18
2.6	Syntax für Umgebungsvariablen in einer <code>docker-compose.yml</code>	19
2.7	Beispiel einer <code>.gitignore</code> -Datei	21
2.8	Öffnen einer interaktiven Shell eines Containers	22
2.9	Initialisierung einer Variablen in Bash	22
2.10	<code>if</code> -Statement zur Variablenprüfung	22
2.11	<code>for</code> -Schleife in Bash	23
2.12	Beispiele der Funktionen von Parameter Expansion	24
2.13	<code>grep</code> in einem Bash <code>if</code> -Statement	25
2.14	Eine Virtual Host-Datei eines Apache Webservers	27
2.15	Beispielhafter Ausschnitt einer <code>hosts</code> -Datei	28
2.16	Beispiel einer <code>composer.json</code>	32
3.1	Festlegung der PHP-Version und Installation PHP-Erweiterungen	42
3.2	Docker Volume-Konfiguration des <code>web</code> -Containers	43
3.3	Kopieren der Datenbank-Konfigurationsdatei <code>my.cnf</code>	44
3.4	Konfiguration einer Vorlagendatei in der <code>docker-compose.yml</code>	45
3.5	Beispiel einer Konfigurations-Template-Datei	45
3.6	Hostname-Konfiguration des <code>db</code> -Containers	45
3.7	Auszug aus dem Bash-Skript <code>replace-templates.sh</code> mit Pseudocode	46
3.8	Build-Argumente zur dynamischen Installation verschiedener Komponenten	47
3.9	Build-Argument <code>INSTALL_COMPOSER</code> zur konditionellen Installation von Composer	47
3.10	Angabe von Verzeichnissen zur Automation von Installations-Prozessen	48
3.11	Skript zur automatisierten Installation von Bibliotheken	48
3.12	Beispielhafter Auszug einer <code>.env</code> -Datei	49
3.13	Freigabe der Ports im <code>web</code> -Container	49

3.14 Docker Volume für die <code>hosts</code> -Datei	50
3.15 Automatische DNS-Konfiguration durch das <code>set-hostname.sh</code> -Skript	51
3.16 Zuordnung von <code>.sql</code> -Datei und Datenbankname in der <code>.env</code> -Datei	52
3.17 <code>import-databases.sh</code> -Skript zum Importieren mehrerer Datenbank-Dateien . .	53
3.18 Installation von Xdebug abhängig von der genutzten PHP-Version	54
3.19 <code>xdebug.ini</code> zur automatischen Aktivierung von Xdebug	54
3.20 Installation von <code>mkcert</code> über die <code>Dockerfile</code> des <code>web</code> -Containers	55
3.21 Zuordnung des CA-Roots des Hosts in den Container über ein Docker Volume	56
3.22 Skript zur Generierung von lokal validen SSL-Zertifikaten	56
3.23 <code>iptables</code> -Konfiguration zur Weiterleitung von E-Mail-Verkehr	57

1 Einleitung

In folgendem Kapitel wird auf die Grenzen und die Problemstellung dieser Arbeit eingegangen. Außerdem sind verwendete Techniken und die Motivation der Lösung des Problems aufgeführt.

1.1 Rahmen der Arbeit

Diese Arbeit befasst sich mit der Erstellung und Analyse eines Repositorys¹, das als wieder-verwendbares Template für Web-Entwicklungsumgebungen verschiedener Kundenprojekte der Firma Hochwarth IT verwendet werden soll. Dabei soll die Software „Docker“ eingesetzt werden. Bei den Kundenprojekten handelt es sich ausschließlich um Web-Entwicklungsprojekte, die u.a. die Programmiersprachen [PHP](#) oder JavaScript umfassen. Andere Entwicklungsumgebungen für serverseitige Programmiersprachen wie Node.js, Go oder Java wären natürlich auch denkbar, aber müssen im Rahmen dieser Arbeit nicht unterstützt werden, da die Firma Hochwarth IT in Kundenprojekten ausschließlich [PHP](#) als serverseitige Programmiersprache verwendet.

Die Software Docker soll dabei nur zur Unterstützung bei der Einrichtung der lokalen Entwicklungsumgebung auf dem Computer des Entwicklers² dienen und nicht für das „Deployment“ in Live-Umgebungen auf einem Server eingesetzt werden. Dies liegt daran, dass die verwendeten Produktiv-Server der Kundenprojekte Docker nicht unterstützen. Mehr dazu in [Abschnitt 4.2](#).

1.2 Aufgabenstellung

Für diese Arbeit gibt es kein Lasten- oder Pflichtenheft, in dem die Aufgabenstellung genau festgelegt wäre. Die Aufgabenstellung ist auch nicht an einen Kunden gebunden. Es gibt also keine vordefinierten Anforderungen an die zu erstellende Software. Deshalb sollen die Funktionalitäten der Software mit einer Analyse festgelegt werden (siehe [Abschnitt 3.1](#)). Daraus

¹ Ein Repository ist eine Ansammlung von Dateien und Verzeichnissen, die in einem digitalen Archiv zusammengefasst sind. Der Begriff wird häufig bei Versionsverwaltungssystemen, wie z.B. Git verwendet (siehe [Abschnitt 2.3](#)).

² In dieser Arbeit wird aus Gründen der besseren Lesbarkeit das generische Maskulinum verwendet. Weibliche und anderweitige Geschlechteridentitäten werden dabei ausdrücklich mitgemeint, soweit es für die Aussage erforderlich ist.

ergeben sich dann die nötigen Aufgaben zur Erreichung der geforderten Funktionalitäten. Das Ziel dieser Arbeit kann aber dennoch wie folgt definiert werden: Erstellung eines Template-Repositories zur vereinfachten Migration von kundenspezifischen Software-Projekten in sogenannte „Docker-Container“. Damit das funktioniert, müssen so viele Projekt-Abhängigkeiten wie möglich festgehalten bzw. automatisiert werden (z.B. verwendete [PHP](#)-Module), um die lokale Einrichtung auf dem Computer zu vereinfachen und zu beschleunigen.

1.3 Vorgehensweise

Wie bereits aufgeführt, müssen zuerst sinnvolle Anforderungen an das Template-Repository gestellt werden, um die Migrationen der Web-Entwicklungsumgebungen in Container zu vereinfachen und zu automatisieren. Dies soll das Resultat der Analyse in [Abschnitt 3.1](#) sein. Die Erstellung des Repositorys umfasst u.a. das Schreiben von Bash-Skripten zur Automatisierung von Installationsprozessen und das Bearbeiten von Konfigurationsdateien. Bei der Erstellung des Repositorys muss darauf geachtet werden, dass es für viele Kundenprojekte von Hochwarth IT wiederverwendet werden kann. Sind alle zuvor aufgestellten Anforderungen erfüllt, kann das Repository zur Migration lokaler Entwicklungsumgebungen der Kundenprojekte verwendet werden. Weiterentwicklungen an diesem Repository sind zu einem späteren Zeitpunkt auf jeden Fall denkbar.

Als zugrunde liegende Technologie wird „Docker“ und „Docker Compose“ verwendet. Zum Entwickeln des Template-Repositorys kommt die [IDE](#)³ „PHPStorm“ der Firma „JetBrains“ zum Einsatz. Der Großteil der ca. 15 Entwickler des Unternehmens Hochwarth IT benutzt diese [IDE](#) für die Entwicklung der Kundenprojekte. Zusätzlich werden für dieses Projekt die PHPStorm-Plugins „.env files support“ [1] und „Batch Scripts Support“ [2] für besseres Syntax-Highlighting und Code-Vervollständigung der zu erstellenden Dateien verwendet.

Nach Erstellung des Repositorys kommen verschiedene Methoden zum Einsatz, um die Performance und die Kosteneinsparungen zu messen und zu vergleichen. So wird z.B. die Ladegeschwindigkeit der Webseite zwischen der Entwicklungsumgebung in Docker-Containern mit verschiedenen [PHP](#)-Erweiterungen und mit einer lokalen Entwicklungsumgebung ohne Docker-Container verglichen.

³ integrated development environment

1.4 Motivation

Um an einer Webanwendung eines Kunden z.B. ein Feature hinzufügen oder eine Änderung vornehmen zu können, muss die Code-Basis dieses Projekts lokal auf dem Computer des Entwicklers vorliegen. Zu jedem Projekt gehören verschiedene Abhängigkeiten, wie z.B. ein Webserver und seine installierten Module, eine [PHP](#)-Instanz und seine erforderlichen Erweiterungen oder eine Datenbank-Engine und die zugehörigen globalen Einstellungen. Es kann sehr viel Zeit in Anspruch nehmen, all diese Abhängigkeiten lokal zu installieren, um dann erst am Projekt arbeiten zu können. Das bedeutet, dass es starke Abhängigkeiten zwischen der Funktionalität einer Anwendung und der Umgebung des Entwickler-Computers gibt. Wünschenswert wäre, dass die Abhängigkeiten nicht am Entwickler-PC hängen, sondern an der Code-Basis des Kundenprojekts selbst. Somit könnten die Abhängigkeiten bequem über ein Versionierungssystem (z.B. Git) am Projekt gespeichert werden und wären somit losgelöst vom Computer des Entwicklers.

Manchmal kann es sehr frustrierend sein, Stunden damit zu verbringen, ein Projekt und alle zugehörigen Abhängigkeiten zu installieren und zu konfigurieren, damit das Projekt lokal einwandfrei läuft. Dahinter verbirgt sich z.B. der Wechsel von lokal installierten [PHP](#)-Versionen zwischen verschiedenen Kundenprojekten, die Pflege der *hosts*-Datei zur lokalen [DNS⁴](#)-Auflösung oder die Pflege der Konfigurationsdatei des Webservers. Dieser Aufwand ist direkt mit erhöhtem Zeitaufwand und damit auch Kosten verbunden. Nicht zu vernachlässigen ist dabei auch der potentiell entstehende Frust des Entwicklers.

Durch verschiedene Automatisierungsmechanismen im Template-Repository könnte dieser zusätzliche Installations- und Konfigurationsaufwand automatisiert und damit in Zukunft minimiert oder sogar komplett eliminiert werden. Dies würde nicht nur den Frust des Entwicklers bei der Projekteinrichtung senken, sondern auch noch Zeit und Geld sparen. Dabei sollte die Entwicklung an einem Kundenprojekt (Server-Start, Ladezeiten, etc.) genauso performant oder sogar performanter sein, als mit einem lokalen Web- und Datenbankserver oder in einer [VM⁵](#)-Umgebung. Außerdem sollte der Speicherplatz und die Ressourcen auf dem Computer des Entwicklers nicht allzu stark belastet werden.

⁴ Domain Name System

⁵ Virtuelle Maschine

2 Technische Grundlagen

In diesem Kapitel sind alle wichtigen Grundlagen und verwendete Technologien für die Lösung des Problems beschrieben. Es dient als Aufbau für das [Kapitel 3](#).

2.1 Docker

Docker ist eine quelloffene Plattform, um Software-Anwendungen zu entwickeln, sie auszuführen und bereitzustellen bzw. zu verteilen. Die Ausführung passiert dabei isoliert vom Host-System in sogenannten „Containern“. Als Host-System bezeichnet man den Computer, auf dem Docker die Software ausführt. Docker ist in der Programmiersprache „Go“ geschrieben und wird von der „Docker Inc.“ entwickelt. [3]

2.1.1 Virtualisierung in Containern

Im Gegensatz zu Hypervisor-basierter Virtualisierungs-Software wie z.B. „VirtualBox“ oder „VMWare ESXi“ simuliert Container-basierte keine komplette virtuelle Maschine, sondern verpackt nur einzelne Teile eines Systems in separate „Container“. [4]

Ein Container ist im Docker-Kontext nur ein Prozess auf dem Host-System, der von allen anderen Prozessen isoliert wurde [5]. An anderer Stelle beschreibt Docker Inc. einen Container als leichtgewichtiges, eigenständiges, ausführbares Softwarepaket, das alles enthält, was zum Ausführen einer Anwendung benötigt wird, also Code, Laufzeit, Systemtools, Systembibliotheken und Einstellungen [6].

In [Abbildung 2.1](#) ist eine Gegenüberstellung der Nutzung von Containern und virtuellen Maschinen zu sehen. Einen großen Unterschied macht dabei die zugrunde liegende Technologie. Virtuelle Maschinen verwenden als Virtualisierungstechnologie einen sogenannten „Hypervisor“.

Ein Hypervisor ist eine Software-Schicht, die es erlaubt, mehrere isolierte Umgebungen (Betriebssysteme) mit separaten virtualisierten Ressourcen zu simulieren und dabei die zugrunde liegende physikalische Hardware zu verteilen [7, S. 3]. Dabei kann der Hypervisor auf einem Betriebssystem oder direkt auf der Hardware laufen. Im Gegensatz zu Containern benötigen

virtuelle Maschinen ein Gast-Betriebssystem, um Anwendungen laufen zu lassen (vgl. Abbildung 2.1). Läuft der Hypervisor über Treiber direkt auf der Hardware ohne ein Betriebssystem dazwischen, spricht man von „Bare-Metal-Hypervisor“ oder Typ-1 Virtualisierung (Beispiel: VMWare ESXi). VirtualBox hingegen läuft auf einem zugrundeliegenden Betriebssystem, was auch Typ-2-Hypervisor genannt wird. [7, S. 10 ff.]

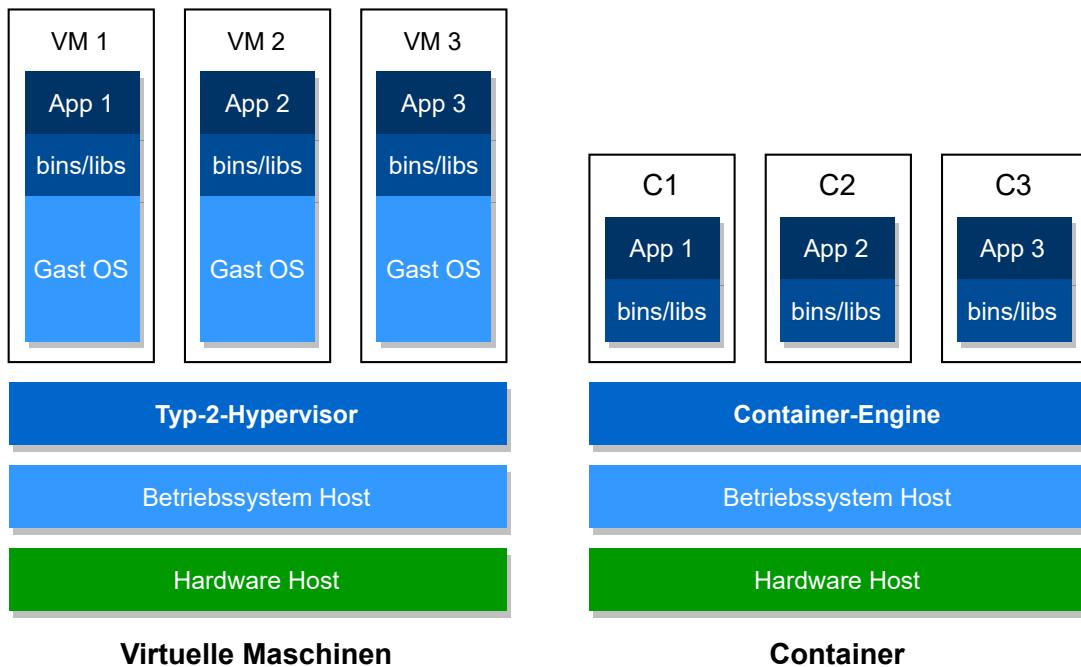


Abbildung 2.1: Vergleich Aufbau virtuelle Maschine und Container [4]

Bei der Container-basierten Virtualisierung gibt es auch eine Abstraktionsebene (hier „Container-Engine“), die ähnlich wie der Hypervisor die zugrunde liegenden Ressourcen an die Container verteilt und diese verwaltet. Hierbei werden die Ressourcen zwar wie bei virtuellen Maschinen beim Erstellen verteilt, können aber bei Container-basierten Systemen von der Container-Engine während der Laufzeit verändert werden. Außerdem nutzen Container-basierte Virtualisierungs-Systeme ein gemeinsames, virtualisiertes Betriebssystem-Image, das aus einem Root-Filesystem, sicher geteilten Systembibliotheken und ausführbaren Dateien besteht. [8, S. 278 f.]

Container-basierte Virtualisierungs-Systeme haben durch das effizientere Nutzen der unterliegenden Ressourcen eine bessere CPU¹-, I/O², RAM³- und Netzwerk-Performance im Vergleich zu virtuellen Maschinen [9, S. 236 f.]. Die Effizienz der verschiedenen Virtualisierungs-Systeme

¹ Central Processing Unit

² Input/Output

³ Random Access Memory

kann gemessen werden anhand von Gesamtleistung und Skalierbarkeit (gemessen an der Anzahl der gleichzeitig laufenden Container) [8, S. 277]. Laut Soltesz [8, S. 286] bietet Container-basierte Virtualisierung eine bis zu zwei Mal höhere Performance im Vergleich zu Hypervisor-basierten Systemen im Kontext von Server-ähnlichen Belastungen. Außerdem können Container dadurch besser skalieren, während die Leistung erhalten bleibt.

Genau wie bei Hypervisor-basierter Virtualisierung bleibt die Isolierung der Systeme (Container) dabei erhalten [8, S. 285]. Das Prinzip der Isolation zwischen verschiedenen Systemen ist ein zentrales Merkmal von Microservices. Die Isolierung einzelner Applikationen als Gegensatz zu *einem* großen System bringt Vorteile wie z.B. bessere Wartbarkeit oder einfachere Fehlersuche.

Doch wozu sollte man dann noch VMs nutzen, wenn Container-basierte Virtualisierung die schlankere und besser skalierbare Alternative ist? Die Nutzung von Hypervisor-basierten Virtualisierungssystemen erlaubt zum Beispiel die Virtualisierung von Windows und Unix-basierten Betriebssystemen auf einer gemeinsamen zugrunde liegenden Hardware. Container-basierte Virtualisierung hingegen kann keine Windows-Container und Linux-Container gleichzeitig laufen lassen [10, S. 76]. Der weiter verbreitete Anwendungsfall für Typ-1-Hypervisoren ist allerdings die Virtualisierung von mehreren Betriebssystemen auf einer leistungsstarken Server-Hardware z.B. in Rechenzentren. Wie bereits erwähnt, läuft der Hypervisor dabei direkt auf der Hardware und benötigt kein zugrunde liegendes Betriebssystem. Container hingegen benötigen ein zugrunde liegendes Betriebssystem für die Virtualisierung.

Ein denkbarer Anwendungsfall wäre z.B. die Kombination der beiden Technologien: Auf einer Server-Hardware läuft VMWare ESXi, um Betriebssysteme zu virtualisieren, worin wiederum eine Container-basierte Virtualisierung betrieben wird, um einzelne Microservices eines Systems zu kapseln (vgl. Abbildung 2.2).

2.1.2 Architektur

Im Folgenden wird die Abbildung 2.3 näher beschrieben.

Ein Container in Docker kann als eine Abstraktionsebene zum Host-System bezeichnet werden. Dabei ist die Interaktion zwischen Host-System und Container durch den „Docker Daemon“ namens `dockerd` vereinheitlicht, sodass ein Container getrennt und unabhängig vom Betriebssystem des Host-Systems (Windows, Linux oder MacOS) ausgeführt werden kann. `dockerd`

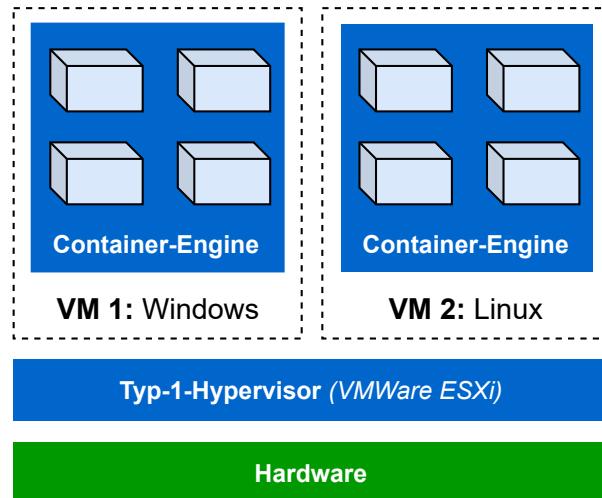


Abbildung 2.2: Container in virtualisierten Betriebssystemen auf Basis von Typ-1-Hypervisoren [10, S. 77]

wird in die zwei Services unterteilt: `runc` und `containerd`.

Ein Daemon ist ein Prozess, der im Hintergrund und nicht unter einem bestimmten Benutzer läuft [11, S. 21]. Unter Windows werden Daemons *Dienste* genannt.

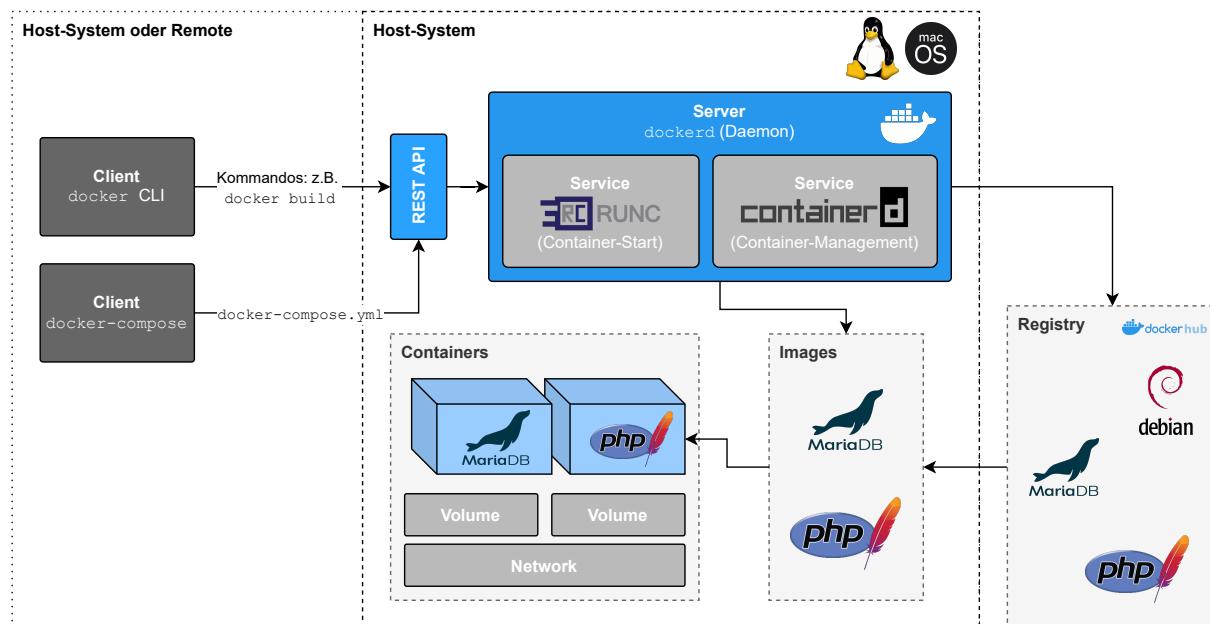


Abbildung 2.3: Die Architektur von Docker und die zugehörigen Beziehungen der Komponenten

runc und containerd

Das CLI⁴ `runc` sorgt dafür, dass das Ausführen von Containern gemäß der OCI⁵-Spezifikation abläuft. Die OCI entwickelt offene Industrie-Standards für Container-Formate und -Laufzeiten [12]. `runc` stellt die Container-Laufzeit zur Verfügung und ist zuständig für das Erstellen von

⁴ Command Line Interface

⁵ Open Container Initiative

„namespaces“ und „cgroups“. `runc` erwartet hierfür Eingaben von `containerd` [13].

Namespaces sind ein Feature des Linux-Kernel, das es erlaubt Systemressourcen über mehrere Prozesse hinweg so zu verteilen, dass es jedem Prozess so erscheint, Zugriff auf eine eigene Instanz dieser Ressource zu haben. Dabei werden die Systemressourcen in eine Abstraktionschicht verpackt. Daraus können Prozesse innerhalb eines Namespaces auf einem Host-System voneinander isoliert werden. Mit CGroups (Control Groups) können Ressourcen begrenzt und priorisiert werden. Sie werden z.B. dafür verwendet, um RAM-Beschränkungen in einem Container zu setzen, damit das Host-System nicht abstürzt, falls ein Container zu viel Platz im RAM verwenden sollte. [14]

Der Service `containerd` ist neben dem Download und Speichern von Docker-Images auch für die Ausführung der Container und deren Überwachung zuständig, also für die Verwaltung der Container. Er erwartet Eingaben über eine API vom `dockerd`-Daemon. [13]

Docker clients

Gesteuert wird der Daemon durch einen Client über eine standardisierte REST⁶ API⁷. Der CLI-Client `docker` umfasst verschiedene Befehle, wie z.B. `docker run` oder `docker build`, um einen Befehl an den Docker Deamon abzusetzen, damit ein Image über eine Dockerfile erstellt wird. Der Docker Daemon leitet dann die Kommandos an den entsprechenden Service weiter. Docker Clients müssen sich nicht unbedingt auf dem Host des Docker Daemons befinden, sondern können auch auf einen Docker Host auf einem Server zugreifen. [13]

Ein weiterer Client ist „Docker Compose“ [3]. Mehr dazu in [Abschnitt 2.2](#).

Images

Der Docker Daemon erstellt die Container aus Images. Ein Image ist ein schreibgeschütztes Template mit Anweisungen zur Erstellung eines Docker-Containers. Images können auf anderen Images basieren. So basiert z.B. das `php`-Image auf einem `debian`-Image (Debian = Linux-Distribution). Das `php`-Image hat also Debian als Basis und kann bei der Erstellung des Containers mit zusätzlichen Instruktionen weitere Software-Pakete installieren, wie z.B. PHP und nötige Konfigurationsdetails setzen. Mit sogenannten „Dockerfiles“ ist es möglich selbst Images schreiben. Mehr dazu im [Unterabschnitt 2.1.3.](#) [3]

⁶ Representational State Transfer

⁷ Application Programming Interface

Registry

Eine Registry kann Docker-Images speichern und stellt ein zentrales Repository für diese dar. Images werden automatisch bei Kommandos, wie `docker run` von der Registry heruntergeladen. Docker Inc. stellt eine öffentliche Registry namens „Docker Hub“ zur Verfügung [15]. Hier können eigene Images hochgeladen, mit der Community geteilt oder Images von anderen heruntergeladen und genutzt werden. Docker lädt standardmäßig die Images vom Docker Hub. Es ist sogar möglich, eine eigene Registry zu hosten. Eine Registry kann privat oder öffentlich sein. [3] [13]

Updates

Updates sind auf jedem System unverzichtlich, wenn es um das Thema Sicherheit, Bugfixing und neue Funktionen geht. Kommt es zum Einspielen von Patches/Updates bezüglich Docker, stellen sich ein paar Fragen. Wie führt man Updates innerhalb der Container durch? Haben Updates auf dem Host-System Einfluss auf die Umgebungen im Container? Wie aktualisiert man die Docker-Engine auf dem Host?

Um Software in einem Container zu aktualisieren, sollte immer das zugrunde liegende Basis-Image aktualisiert werden und nicht die Pakete manuell über z.B. `apt-get upgrade` innerhalb des Containers aktualisiert werden. Dadurch würde sonst das reproduzierbare Verhalten eines Containers verloren gehen. Diese Vorgehensweise könnte sogar als Anti-Pattern bezeichnet werden, also als nicht empfohlene Vorgehensweise zum Aktualisieren von Software innerhalb eines Containers. Da die persistenten Daten eines Containers (Volumes) unabhängig vom Basis-Image sind, kann das Image in einem Update-Prozess einfach ausgetauscht werden. Mit der Befehlsabfolge in [Code 2.1](#) kann ein einfacher Datenbank-Container über das Image `mysql` aktualisiert werden.

```
1 $ docker pull mysql # zieht neues Image vom Docker Hub
2 $ docker stop mysql-container # stoppt Container
3 $ docker rm mysql-container # löscht Container
4 $ docker run --name=mysqlContainer -e MYSQL_ROOT_PASSWORD=mypwd -v
   ↳ /my/data/dir:/var/lib/mysql -d mysql # startet Container mit neuem Image
```

Code 2.1: Befehlsabfolge zur Aktualisierung eines Datenbank-Containers

Patches von anderen Software-Paketen auf dem Host-System sollten weder die Docker-Engine

noch die Umgebungen innerhalb der Container beeinflussen, da diese ja vom Host-System über die bereits beschriebenen Technologien abgekapselt sind. Größere Updates vom Kernel des Host-Systems könnten natürlich zu Problemen führen mit den zugrunde liegenden Technologien, die Docker zur Container-Virtualisierung und Isolation nutzt.

Um die Docker-Engine auf dem Host-System zu aktualisieren, empfiehlt Docker Inc. die zugehörigen Pakete über den Paketmanager der jeweiligen Distribution zu updaten. Dabei bleibt das Verzeichnis `/var/lib/docker/` erhalten, da dieses die Images, Container, Volumes und konfigurierten Docker-Netzwerke enthält. [16]

Die Abfolge eines Updates der Docker-Engine auf dem Host ist trivial und kann grob wie folgt beschrieben werden: Docker-Engine herunterfahren, Software aktualisieren, (Host-System neu starten,) Docker-Engine wieder starten. [17]

2.1.3 Dockerfile

Um ein Docker-Image selbst zu generieren, kann eine sogenannte „Dockerfile“ erstellt werden. Dockerfiles haben eine einfache Syntax, die die Schritte zum Erstellen des Images beinhalten. Jede Anweisung dieser Dockerfile erzeugt bei der Erstellung des Images eine eigene virtuelle Schicht, die zwischengespeichert wird. Wird eine Zeile in der Dockerfile geändert, werden nur die betroffenen virtuellen Schichten aktualisiert, die sich dadurch ändern. Diese Vorgehensweise macht die Erstellung und Aktualisierung von Images über Dockerfiles signifikant schneller. [3]

Das heißt, in einem Dockerfile sind (implizit) alle Abhängigkeiten eines Containers festgelegt, also Betriebssystem, Systemtools, Systembibliotheken und Einstellungen. Somit kann unabhängig vom Host-System immer der gleiche Container mit einer Dockerfile erstellt und ausgeführt werden.

In [Code 2.2](#) ist eine beispielhafte Syntax einer Dockerfile zu sehen, auf die im Folgenden genauer eingegangen wird.

```

1 ARG PHP_VERSION=7.4
2 FROM php:${PHP_VERSION}-apache
3
4 # copy file into container
5 COPY ./sources.list /etc/apt/sources.list
6
7 # enable apache module rewrite
8 RUN a2enmod rewrite
9
10 # install nodejs for npm
11 RUN apt-get update && apt-get install -y curl \
12     && curl -sL https://deb.nodesource.com/setup_lts.x | bash - \
13     && apt-get install -y nodejs

```

Code 2.2: Beispielhafter Inhalt einer Dockerfile

Eine Dockerfile beginnt immer mit einer `FROM`-Anweisung. Diese stellt das Basis-Image bzw. „Parent-Image“ dar, auf dem aufgebaut werden soll (siehe auch „Images“ im [Abschnitt 2.1.2](#)). Das angegebene Image wird beim ersten Ausführen standardmäßig von Docker Hub heruntergeladen. Die `ARG`-Instruktion ist die einzige, die vor `FROM` stehen darf. Der Befehl `ARG` ist gut dafür geeignet, Dockerfiles dynamisch aufzubauen. Mit ihr können Variablen von außen für den „Build-Prozess“ in die Dockerfile einbezogen werden. [18]

Die wichtigste Anweisung ist `RUN`. Mit ihr können Shell-Kommandos auf Basis des aktuellen Images ausgeführt werden. Bei dem Basis-Image `ubuntu` werden also Linux Shell-Befehle (standardmäßig über `/bin/sh -c`) im Image als neue Schicht ausgeführt, selbst wenn der Docker Daemon auf Windows oder MacOS läuft. So können z.B. im Basis-Image `ubuntu` über das Paketmanagement-System `APT`⁸ mit dem `RUN`-Befehl Software-Pakete installiert werden. Natürlich können auch sämtliche andere Linux Shell-Befehle ausgeführt werden. [18]

Um eine Datei in einen Container zu kopieren, wird der Befehl `COPY <src> <dest>` verwendet. Die Anweisung kopiert dabei nicht nur lokale Dateien, sondern auch Verzeichnisse aus `<src>` (relativ zum Pfad der Dockerfile) und fügt diese dem Dateisystem des Containers unter dem Pfad `<dest>` hinzu. [18]

Wird danach der Befehl `docker build ./` im Verzeichnis der `Dockerfile` ausgeführt, wird zuerst das Basis-Image heruntergeladen. Wenn das Image (oder Teile davon) sich bereits auf

⁸ Advanced Packaging Tool

dem Host-System befindet, kann es aus dem Zwischenspeicher verwendet werden (vgl. Zeile 4 in [Code 2.3](#)). Danach werden die angegebenen Befehle der Dockerfile sequentiell ausgeführt. In den Zeilen 5-7 ist zu erkennen, dass Docker diese Befehle nicht nochmals ausführen musste, da bereits eine Schicht mit dem jeweiligen Befehl zwischengespeichert wurde (vgl. Ausgabe `CACHED`). Am Ende der Konsolenausgabe wird das fertige Image auf die Festplatte geschrieben. [19]

```
1 $ docker build ./
2 [+] Building 0.4s (9/9) FINISHED
3 => [...]
4 => [1/4] FROM docker.io/library/php:7.4-apache           0.0s
5 => CACHED [2/4] COPY ./sources.list /etc/apt/sources.list   0.0s
6 => CACHED [3/4] RUN a2enmod rewrite                      0.0s
7 => CACHED [4/4] RUN apt-get update && apt-get install -y curl && [...] 0.0s
8 => exporting to image                                     0.1s
9 => => exporting layers                                    0.0s
10=> => writing image sha256:b1e647bcd3e224843c2ce283b23901b5c031718[...] 0.0s
```

Code 2.3: Konsolenausgabe eines `docker build`-Befehls

2.1.4 Volumes

Docker speichert alle Daten in einem Verzeichnis auf dem Host-System. Darunter sind die Images, Container und sogenannte „named“ und „anonymous Volumes“ gespeichert [11, S. 27].

Bevor erklärt werden kann, was Volumes sind, muss zuerst darauf eingegangen werden, wie Dockers Dateisystem „Union File System“ funktioniert. Wenn ein Container gestartet wird, wird auf die zugrunde liegenden, schreibgeschützten Schichten („read-only“) des Images (siehe [Unterabschnitt 2.1.3](#)) eine „read-write“-Schicht (vgl. Runtime-Schicht in [Abbildung 2.4](#)) gelegt. Findet danach in einem laufenden Container eine (manuelle) Änderung einer Datei statt, wird diese aus der read-only-Schicht in die read-write-Schicht kopiert. Wird ein Container gelöscht und danach wieder erstellt, werden nur die read-only-Schichten aus dem Image durchlaufen. Dementsprechend sind alle Änderungen verloren, die zuvor in der read-write-Schicht waren. [20]

Volumes sind Dockers Mechanismus, um Daten außerhalb dieses Lebenszyklus eines Containers zu verwalten. Sie stellen also eine Möglichkeit dar, Daten in Persistenz zu halten, selbst

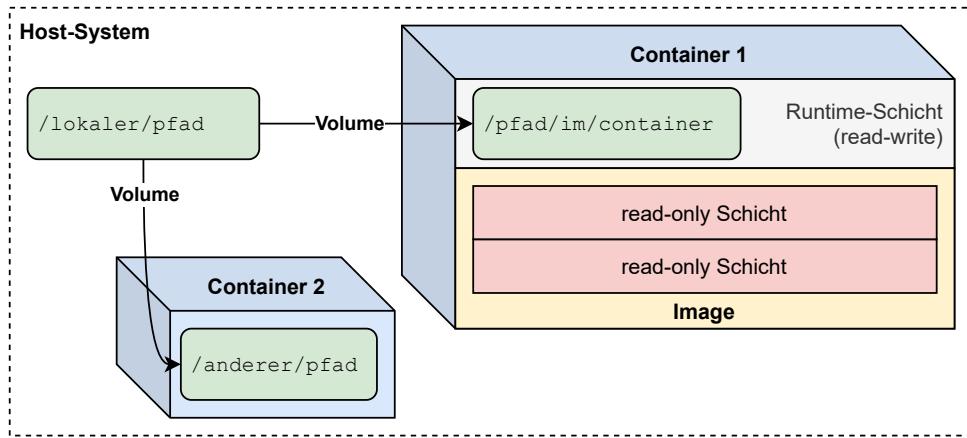


Abbildung 2.4: Funktionsweise von Docker-Volumes

wenn ein Container gelöscht oder neu gestartet wird. Volumes sind also Verzeichnisse bzw. Dateien, die außerhalb des Union File Systems existieren und dementsprechend keine Schicht-Architektur haben, sondern wie normale Dateien behandelt werden. [20]

Ein Volume funktioniert dabei im Prinzip wie „Mounting“. Das bedeutet, dass ein Verzeichnis oder eine Datei außerhalb des Containers einem Verzeichnis/Datei innerhalb des Containers zugeordnet wird und eine Änderung an einem von beiden Endpunkten in einer sofortigen Änderung am anderen Endpunkt resultiert.

Die Verwendung eines Volumes als Persistenz anstatt der read-write-Schicht hat den Vorteil, dass sich die Größe des Containers nicht verändert, da bei einem Volume die Dateien direkt vom Dateisystem des Host-Systems abgerufen und geschrieben werden [21].

Zudem können Volumes dazu genutzt werden, Daten zwischen Containern zu teilen (vgl. Abbildung 2.4). Ein klassisches Beispiel hierfür ist eine große, zentralisierte Datenbank, die von mehreren Containern oder dritten Clients (nicht Container) genutzt werden sollen. [11, S. 108]

Docker unterscheidet zwischen *anonymous*, *named* und *mount* Volumes. Volumes können mit dem `-v`-Argument des `docker`-Clients erstellt werden. Die Syntax dieses Arguments für den Befehl `docker run` sieht wie folgt aus:

```
$ docker run -v [./local/path:<name>:] /path/inside/container[:ro] # [...]
```

Anonymous und *named* Volumes werden von Docker verwaltet und sind auf dem Host-System unter `/var/lib/docker/volumes` zu finden. Durch *named* Volumes kann Docker ein Volume

mehreren Containern zur Verfügung stellen (vgl. `--volumes-from`-Flag). [21]

Die Spezifizierung eines *mount* Volumes hat die Syntax `./local/path:/path/in/container`. Damit werden explizit lokale Verzeichnisse oder Dateien einem Verzeichnis im Container zugeordnet und dementsprechend eingehängt (im Englischen: „mounting“). Dabei werden die Verzeichnisse jeweils angelegt, wenn diese nicht existieren. [11, S. 108]

Diese Art von Volume wird in dieser Arbeit am häufigsten verwendet.

2.1.5 Ports

Um eine Anwendung, die sich innerhalb eines Containers befindet, auf dem Host-System zur Verfügung zu stellen, muss diese über Ports auf das Host-System „exposed“ werden. Der dafür vorgesehene Flag `-p` des Docker CLI-Clients erlaubt unterschiedliche Syntaxen. Mit `-p 8080:80` ist z.B. eine Webanwendung, die im Container auf Port `80` läuft, auf dem Host-System unter `http://localhost:8080` verfügbar. Es kann allerdings auch der Host (mittels IP⁹-Adresse) angegeben werden, der Verbindungen von diesem Container akzeptieren soll: `-p 192.168.1.100:8080:80`. Diese Funktion wird im Rahmen dieser Arbeit verwendet, um Webanwendungen über einen definierbaren Hostnamen zur Verfügung zu stellen.

2.1.6 Netzwerk

Damit Docker-Container untereinander, nach außen zum Host-System und in das Internet kommunizieren können, benötigen die Container IP-Adressen. Außerdem muss ein virtuelles Netzwerk vorhanden sein. Mit dem Start des Docker Daemon wird eine neue Netzwerk-Schnittstelle auf dem Host-System namens `docker0` angelegt und standardmäßig eine virtuelle „Bridge“ eingerichtet. Die Bridge arbeitet dabei in der Regel mit privaten IP-Adressen aus dem 172er-Netz. [10, S. 237 f.]

Unter Windows ist die `docker0`-Schnittstelle nicht auf dem Host-System zu finden, sondern wird über WSL¹⁰ abstrahiert (WSL siehe in Unterabschnitt 2.1.7) [22].

Um ein neues Netzwerk über Docker zu erstellen, kann der Befehl `docker network create` genutzt werden. Über den Befehl `docker run` kann danach mit dem Argument `--network=<name>` der zu startende Container mit dem Netzwerk verknüpft werden. [23]

⁹ Internetprotokoll

¹⁰ Windows Subsystem for Linux

Ein beispielhafter Aufbau eines solchen Netzwerks auf einem Linux Host-System ist in [Abbildung 2.5](#) zu sehen. Dabei fungiert die Bridge (172.17.0.1) als Gateway für die unterliegenden Container. Beim Initialisieren eines Containers wird in der Bridge ein virtuelles Interface (*veth*) angelegt. Dieses Interface bildet die Schnittstelle zum jeweiligen Container (172.17.0.2 und 172.17.0.3). Will ein Container in das Internet kommunizieren, werden die Anfragen über das Bridge-Gateway 172.17.0.1 nach außen auf den Host geleitet und von dort aus ins Internet. [10, S. 240 f.]

Müssen die Container untereinander kommunizieren, ist dies u.a. über die IP-Adresse möglich. Beim Löschen eines Containers wird allerdings die IP-Adresse wieder freigegeben. Das bedeutet, dass die IP eines Containers nicht immer die gleiche ist. Deshalb wird die Kommunikation zwischen Containern bei Docker über DNS gelöst. Der Docker Daemon hat dafür einen eingebauten DNS-Server, der von den Containern standardmäßig genutzt wird, wenn ein Docker-Netzwerk über `docker network create` erstellt wurde. So kann z.B. der Container `web` ohne weitere Konfiguration den Container `db` über den Befehl `ping db` pingen, wenn beide Container sich im selben Docker Bridge-Netzwerk befinden. [24]

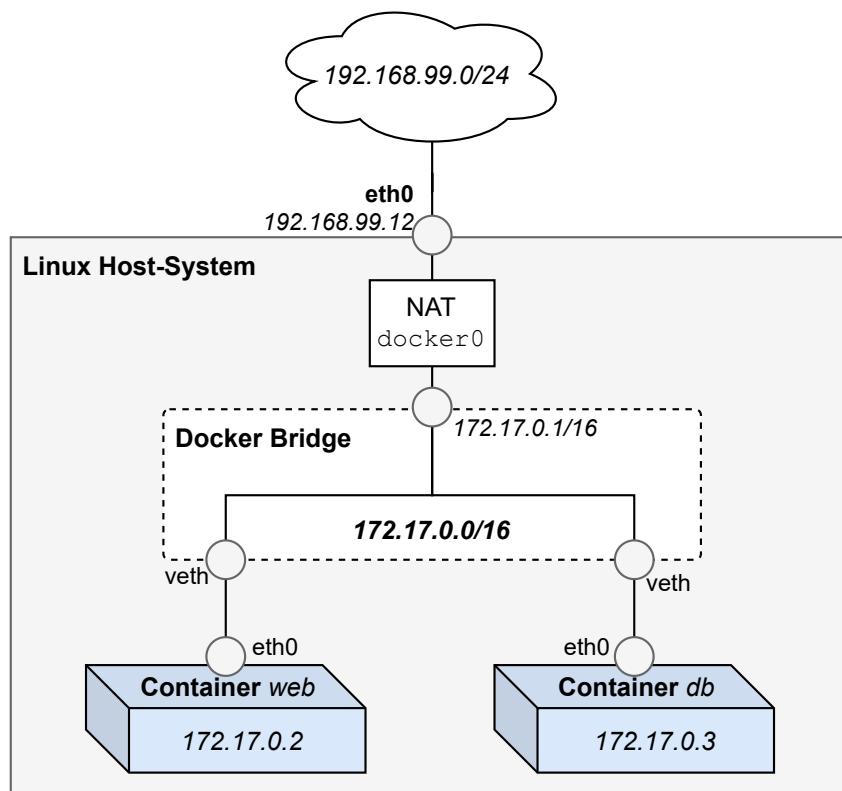


Abbildung 2.5: Beispiel eines automatisch konfigurierten Docker Bridge-Netzwerks [10, S. 240]

Außer der Bridge gibt es noch weitere Netzwerk-Treiber für Docker, z.B. ein Treiber für Host-

und ein Treiber für *Overlay*-Netzwerke. Host-Netzwerke sollten verwendet werden, wenn das Netzwerk nicht isoliert vom Docker-Host sein soll. Overlay-Netzwerke ermöglichen eine Kommunikation zwischen Container, die auf unterschiedlichen Docker-Hosts laufen.

2.1.7 Docker unter Windows

Da in dieser Arbeit die Docker-Engine hauptsächlich unter Windows genutzt wurde, sind hier noch ein paar Besonderheiten dazu aufgeführt. Docker wird unter Windows und MacOS mit „Docker Desktop“ installiert. Docker Desktop umfasst den `docker` CLI-Client, Docker Compose (siehe [Abschnitt 2.2](#)), die Docker-Engine bestehend aus dem `dockerd`-Daemon und der API und noch weitere Tools für die Sicherheit und für die Automatisierung von Container-Anwendungen. Außerdem hat Docker Desktop eine GUI¹¹, die es erlaubt, Container bzw. Containergruppen zu starten, stoppen, löschen oder deren Logs in Echtzeit anzuschauen. Es bietet außerdem Möglichkeiten zum Verwalten der lokalen Images und „remote“ Repositorys auf Docker Hub und hat noch viele weitere nützliche Funktionen. [17]

Docker kann unter Windows mit einem Hyper-V-Backend oder mit einem WSL2-Backend betrieben werden. WSL2 ist ein kompletter Linux-Kernel entwickelt von Microsoft, der es erlaubt, Linux-Distributionen wie z.B. Ubuntu unter Windows nativ und ohne Emulation laufen zu lassen. Im Vergleich zur Version 1 bringt WSL2 viele Verbesserungen bezüglich Dateisystem-Sharing, Container-Startzeiten und Ressourcenmanagement. Es wird mittlerweile auch von Docker Inc. empfohlen, WSL2 als Backend für Docker Desktop unter Windows zu verwenden anstatt das Hyper-V-Backend. [25]

2.2 Docker Compose

Die Befehle des `docker`-Client, um einen oder mehrere Container mit Dockerfiles, Volumes, Ports und zugehörigem Docker Netzwerk zu erstellen, können schnell umfangreich werden. Anstatt die Startbefehle für die Container in einem komplexen Shell-Skript festzulegen, ist die zentrale Idee von Docker Compose die Startkonfiguration der Container zu deklarieren, um danach die Anwendungen mit einem Befehl zu starten [11, S. 221]. Docker Compose ist also ein „Wrapper“-Client für den CLI-Client `docker`, um mehrere Container über eine

¹¹ graphical user interface

Konfigurationsdatei zu verwalten und die Abhängigkeiten, wie z.B. Dockerfiles, Volumes oder Ports festzuhalten.

Mit dem Befehl `docker-compose up` können mehrere Container gleichzeitig gestartet werden. Docker Compose übernimmt dabei das Erstellen bzw. Herunterladen der Images auf Basis der `docker-compose.yml`, die Konfiguration des Docker-Netzwerks und das Starten der Container. Ein Ausschnitt einer Beispielausgabe dieses Befehls ist in [Code 2.4](#) zu sehen.

```
1 $ docker-compose up
2 Creating network "docker-template_default" with the default driver
3 Creating test-db ... done
4 Creating test-web ... done
5 Attaching to test-db, test-web
6 test-db | [...] (log output)
7 test-web | [...] (log output)
8 [...]
```

Code 2.4: Ausgabe eines Docker Compose-Befehls

Die Konfiguration für die Container liest Docker Compose standardmäßig aus der Datei `docker-compose.yml` im aktuellen Verzeichnis. Der Inhalt dieser Datei ist im [YAML¹²](#)-Format. [YAML](#) ist ein Daten-Serialisierungs-Standard vergleichbar mit [JSON¹³](#), der sowohl menschenlesbar als auch schnell (de)serialisierbar ist [26].

Ein Beispiel einer solchen Datei ist in [Code 2.5](#) zu sehen. Unter `services` sind die Konfigurationen der zwei Container `web` und `db` zu finden.

¹² YAML Ain't Markup Language

¹³ JavaScript Object Notation

```

1 services:
2   web: # frontend-Container
3     build: ./web # in ./web lies a Dockerfile
4     command: /inside/container/startup.sh
5     volumes:
6       - ./web/scripts:/inside/container
7     ports:
8       - 8080:80
9     depends_on:
10      - db
11
12   db: # database-Container
13     image: mariadb:latest
14     environment:
15       MARIADB_ROOT_PASSWORD: test
16       MARIADB_DATABASE: db_name
17     ports:
18       - 3306:3306

```

Code 2.5: Beispiel einer `docker-compose.yml`-Datei

Während der `web`-Container über eine Dockerfile im Verzeichnis `./web` (relativ zum Verzeichnis der `docker-compose.yml`) erstellt wird, nutzt der `db`-Container das Docker-Image `mariadb:latest` und zieht dementsprechend dieses Image aus dem Docker Hub.

Mit dem Schlüsselwort `command` wird das `CMD`-Kommando aus der zugehörigen Dockerfile überschrieben und das angegebene Skript `/inside/container/startup.sh` am Ende des Starts des Containers ausgeführt [18]. Über ein Volume wird in [Code 2.5](#) (Zeile 6) dieses Skript dem `web`-Container hinzugefügt. Dieser Ablauf ist auch genauso über eine Dockerfile abbildungbar.

Über die Auflistung `ports` kann ein Container mehrere Ports nach außen auf das Host-System öffnen. Der `web`-Container ordnet z.B. den Port `80` innerhalb des Containers auf den Port `8080` außerhalb des Containers zu.

Das `depends_on`-Schlüsselwort gibt Abhängigkeiten zwischen Containern an und legt somit fest, in welcher Reihenfolge die Container gestartet und gestoppt werden. In diesem Beispiel hängt der Container `web` vom Container `db` ab. Dementsprechend wird bei einem `docker-compose up`-Befehl zuerst der `db`- und dann `web`-Container gestartet und bei dem Befehl `docker-compose stop` umgekehrt wieder gestoppt. [27]

Im Container `db` in [Code 2.5](#) werden mit `environment` Umgebungsvariablen an den Container übergeben. Die Werte dieser „Environment-Variablen“ können in „Startup-Skripten“ oder in `docker-compose.yml`-Dateien selbst benutzt werden und bieten somit eine einfache Möglichkeit, das Verhalten von Containern für den Benutzer konfigurierbar zu gestalten. Allerdings sind diese Variablen während des Build-Prozesses des Images nicht verfügbar, weshalb sie nicht in einer Dockerfile genutzt werden können. [\[27\]](#)

Docker Compose bietet auch die Möglichkeit die Umgebungsvariablen in einer separaten Datei anzugeben. Standardmäßig heißt diese Datei `.env` und befindet sich im selben Verzeichnis wie die `docker-compose.yml`. Mit dem Schlüsselwort `env_file` können in einer `docker-compose.yml` weitere Environment-Dateien angegeben werden [\[27\]](#).

Durch die Auslagerung der Umgebungsvariablen in andere Dateien ist es möglich, die `docker-compose.yml` mit all seinen Abhängigkeiten in ein Versionierungssystem (z.B. Git - siehe [Abschnitt 2.3](#)) zu übernehmen und die Datei für die Umgebungsvariablen mit Git zu ignorieren. Dadurch können alle projektspezifischen Abhängigkeiten in einem Git-Repository gespeichert, die dynamischen Umgebungsvariablen ignoriert und auf jedem Host-System individuell angepasst werden.

Um die Umgebungsvariablen in der `docker-compose.yml`-Datei nutzen zu können, gibt es eine spezielle Syntax, um zur Startzeit von Docker Compose die Werte der Environment-Variablen einzusetzen. [\[28\]](#)

In [Code 2.6](#) ist eine solche Syntax mit `${PROJECT_NAME}` dargestellt.

```
1 services:  
2   web:  
3     image: ${PROJECT_NAME}-web
```

[Code 2.6:](#) Syntax für Umgebungsvariablen in einer `docker-compose.yml`

2.3 Git

Linus Torvalds hat im Rahmen der Entwicklung des Linux Kernels nach einer quelloffenen Alternative für die Versionierung des Quellcodes gesucht. Nachdem er kein passendes Tool für seine Ansprüche gefunden hatte, hat er entschieden selbst ein Versionierungssystem ([VCS¹⁴](#)) zu entwickeln. Git wurde erstmals im April 2005 von Linus Torvalds veröffentlicht. [\[29, S. 5 f.\]](#)

¹⁴ Version Control System

Torvalds hat bei der Entwicklung von Git darauf geachtet, dass jeder Nutzer eines Repositorys eine komplette Historie jeder Datei offline verfügbar hat. Das ermöglicht paralleles und unabhangiges Entwickeln in dezentralen Kopien einer Codebasis, ohne ständig mit einem zentralen Repository synchronisieren zu müssen. Um eine Repository von einem zentralen Server lokal auf den Computer zu kopieren, wird der Befehl `git clone` verwendet. Natürlich ist es auch möglich, den Stand eines Projekts auf ein zentrales „remote Repository“ auf einem Server zu synchronisieren. Dies passiert mit dem Befehl `git push`. Um eine oder mehrere Dateien in Git im lokalen Repository zu speichern, wird der Befehl `git commit` verwendet. Die Änderungen werden in einem sogenannten „Commit“ gespeichert. Hierbei muss ein Änderungsprotokoll angegeben werden, genannt „commit message“. Das Setzen eines Projekts auf den Stand eines bestimmten Commits nennt man „auschecken“.

Wenn zwei Entwickler an unterschiedlichen Features einer Software arbeiten, kann es sinnvoll sein, für jedes Feature einen eigenen „Branch“ anzulegen, um diese danach wieder in einem Hauptbranch zusammenzuführen. Ein Branch ist eine Sequenz von Commits und muss mit einem Namen versehen werden. Um Branches zusammenzuführen und Konflikte gleicher Dateien in unterschiedlichen Branches aufzulösen, kann ein „Merge“-Prozess mit dem Befehl `git merge` eingeleitet werden. [29, S. 2 ff.]

Damit Git bestimmte Dateien oder Verzeichnisse nicht in die Versionsverwaltung mit aufnimmt, können diese in einer `.gitignore`-Datei festgelegt werden. Das ist z.B. bei temporären Dateien oder lokalen Konfigurationsdateien sinnvoll, die Zugangsdaten zu einer Datenbank enthalten, denn diese sollten aus sicherheitstechnischen Gründen nicht in einem (öffentlichen) Repository landen. Eine `.gitignore`-Datei kann in jedem Verzeichnis liegen. Die Einträge darin beziehen sich immer relativ zum Ordner, in dem sich die Datei befindet. In [Code 2.7](#) ist ein Beispiel einer solchen Datei zu finden. Dort sind auch die Sonderzeichen `*` und `!` zu finden.
`*` findet jegliches beliebige Zeichen beliebig oft und bietet so die Möglichkeit, mehrere Dateien mit einer Zeile von der Git-Versionierung auszuschließen. Das `!` negiert das Muster wieder und inkludiert so explizit Dateien, die zuvor exkludiert wurden (hier `config/global.ini`).
[29, S. 58 ff.]

```
1 relative/path/to/ignored-directory  
2 config/*.ini  
3 !config/global.ini
```

Code 2.7: Beispiel einer `.gitignore`-Datei

2.4 Shell-Umgebungen und Bash

Eine Shell ist im Prinzip ein Makroprozessor, der Befehle auf dem Betriebssystem ausführt und somit die Schnittstelle zwischen dem Benutzer und dem Betriebssystem bildet. Eine Unix-Shell kann dabei sowohl als Schnittstelle in einem Terminal über Befehle bedient werden als auch als Programmiersprache über ausführbare Skript-Dateien. [30]

Shells kommen bei Docker an mehreren Stellen zum Einsatz: Bei der Erstellung von Images durch `RUN`-Befehle in Dockerfiles, in einer interaktiven Shell innerhalb eines Containers oder in ausführbaren Shell-Skripten, die über den `CMD`-Befehl nach dem Start eines Containers ausgeführt werden.

Der `RUN`-Befehl in einer Dockerfile führt seine Befehle standardmäßig in der Shell `/bin/sh` in Linux-Containern und im Programm `cmd.exe` in Windows-Containern aus. Sollen die Befehle in der Dockerfile alle mit einer anderen Shell (z.B. `/bin/bash`) ausgeführt werden, so kann das mit dem `SHELL`-Befehl geändert werden. [18]

Um unter Linux explizit in einer ausführbaren Datei anzugeben, mit welchem Interpreter/Shell sie ausgeführt werden soll, kann mit der `#!`-Syntax am Anfang der Datei der absolute Pfad zur Shell-Binary angegeben werden, z.B. `#!/bin/bash`.

Um Shell-Befehle innerhalb eines laufenden Containers auszuführen, bietet Docker den Befehl `docker exec`, bzw. `docker-compose exec`. Damit kann jeder denkbare Befehl in der Shell des Containers ausgeführt werden. Außerdem ist es möglich mit den Argumenten `-i` und `-t` eine interaktive Shell zu allozieren, um Befehle direkt in der Container-Shell auszuführen. In [Code 2.8](#) wird ein Ubuntu-Container mit dem Namen `ubuntu_bash` im Hintergrund gestartet und danach mit dem `docker exec`-Befehl eine neue interaktive Shell im Container geöffnet.

```

1 usr:$ docker run --name ubuntu_bash -d -it ubuntu /bin/bash
2 usr:$ docker exec -it ubuntu_bash /bin/bash
3 root@e49aa604f585:$

```

Code 2.8: Öffnen einer interaktiven Shell eines Containers

Bash allgemein

Bash („Bourne-Again SHell“) ist die Standard-Shell im [GNU¹⁵](#) Betriebssystem und wurde 1989 entwickelt. Aktuell wird es von Chet Ramey verwaltet und weiter entwickelt. Sie ist kompatibel mit der Unix-Shell `sh` (`/bin/sh`) und eine Implementierung nach der „[IEEE¹⁶ POSIX¹⁷ Shell and Tools specification](#)“. Bash bietet für die interaktive als auch für die programmatische Verwendung viele Verbesserungen gegenüber `sh`. [30]

Die gesamte Dokumentation für Bash ist hier [31] zu finden. Im Folgenden wird auf triviale und auch komplexere Funktionen von Bash eingegangen, die in [Kapitel 3](#) für einige Skripte des Template-Repositorys relevant sind.

Variablen werden in Bash mit einem einfachen `=` initialisiert und mit einem vorangestellten `$` aufgerufen [31]. In [Code 2.9](#) wird mit dem `echo`-Befehl die Variable `myVar` ausgegeben.

```

1 myVar=World
2 echo "Hello $myVar"

```

Code 2.9: Initialisierung einer Variablen in Bash

Die `if`-Syntax in Bash kann sehr umfangreich werden. Es gibt viele Operatoren, um Boolean-, Integer-, String-Werte oder Dateien in einem `if`-Statement zu verwenden. Die Syntax zur Prüfung, ob eine Variable existiert, ist in [Code 2.10](#) zu sehen.

```

1 if [ -n "$var" ]; then
2     # $var ist nicht null
3 fi

```

Code 2.10: `if`-Statement zur Variablenprüfung

Um Schleifen in Bash zu nutzen, kann `for` oder `while` genutzt werden. Um über einen Be-

¹⁵ GNU's Not Unix

¹⁶ Institute of Electrical and Electronics Engineers

¹⁷ Portable Operating System Interface

reich zu iterieren, wird wie in [Code 2.11](#) gezeigt, die Syntax `for i in {START..END..STEP}` genutzt.

```
1 for i in {1..4}; do  
2     # aktuelle Nummer ist in $i  
3 done
```

Code 2.11: `for`-Schleife in Bash

Parameter Expansion

Eine sehr leistungsfähige Funktion in Bash ist die sogenannte „Parameter Expansion“. Sie erlaubt es, den Wert einer referenzierten Variable zu erhalten, diesen mit verschiedenen Funktionen zu verändern und auszugeben. Eine dieser Funktionen ist das globale Suchen von `PATTERN` und Ersetzen durch `STRING` in dem Muster `${VAR//PATTERN/STRING}` der Variablen `VAR`. Um einen Teil eines String-Wertes einer Variablen zu extrahieren, kann „Substring Expansion“ verwendet werden mit der Syntax `${VAR:OFFSET:LENGTH}`, wobei `LENGTH` optional ist. Eine kompliziertere Funktion ist die „Indirection“. Sie hat die Syntax `${!VARNAMEN}`. Angenommen die Variable `VARNAMEN` hat den Wert `test`, dann evaluierst die *Indirection* dynamisch den Wert der Variablen `$test`. Auch eine sehr nützliche Funktion ist die „Variable Name Expansion“. Ihre Syntax ist `${!PREFIX@}`. Diese Funktion expandiert zu einer Liste von allen Variablen-Namen, die mit `PREFIX` beginnen. [\[32\]](#)

In [Code 2.12](#) sind Beispiele der beschriebenen Parameter Expansion-Funktionen zu sehen.

```

1 STRING="Hello mom! Hello dad!"
2 echo ${STRING//Hello/Bye} # Bye mom! Bye dad!
3
4 STRING="This is a test123"
5 echo ${STRING:0:-3} # This is a test
6
7 PREFIX_1="wow"
8 PREFIX_2="cool"
9 for varName in "${!PREFIX_@}"; do # Variable Name Expansion
10   echo ${!varName} # Indirection
11   # first round 'wow', second round 'cool'
12 done

```

Code 2.12: Beispiele der Funktionen von Parameter Expansion

Command Substitution und Pipelines

Die sogenannte „Command Substitution“ in Bash (`$(command)`) erlaubt es, die Ausgabe eines Befehls direkt mit dem Befehl selbst zu ersetzen [33]. Ein Beispiel wäre das Umleiten der (Error)-Ausgabe des Kopieren-Befehls `cp` in eine Variable: `OUT=$(cp a.txt /path 2>&1)`.

Das Pipe-Zeichen `|` teilt mehrere Befehle in eine sogenannte „Pipeline“. Die Pipe sorgt dafür, dass die Standardausgabe eines Befehls an die Standardeingabe des nachfolgenden Befehls weitergeleitet wird. Ein Beispiel wäre die Weiterleitung des `printf`-Befehls an den `sort`-Befehl, um herauszufinden, welche Version höher ist: `printf '%s\n' 4.1 4.2 | sort -V`.

Regex und grep

Ein weiteres leistungsstarkes Tool in Bash ist das Programm `grep`, das u.a. reguläre Ausdrücke verwendet. Reguläre Ausdrücke (engl. „regular expressions“, kurz „regex“) werden verwendet, um Strings auf bestimmte Zeichen-Muster zu prüfen, oder diese durch andere zu ersetzen. Regex ist allerdings keine Sammlung von Funktionen, die von einer bestimmten Programmiersprache oder von einem Programm abhängig ist, sondern eine formale Sprache, bei der bestimmte Ausdrücke eine genaue Bedeutung haben, was es so leistungsstark und flexibel einsetzbar macht. [34, S. 439]

In Tabelle 2.1 findet sich eine Übersicht einiger dieser Ausdrücke beschrieben mit ihren Funktionen.

Zeichenfolge	Funktion
^	Beginn einer Zeile
\$	Ende einer Zeile
+	Quantifizierer: Vorangestellter Ausdruck muss mind. 1x vorkommen
?	Quantifizierer: Vorangestellter Ausdruck darf 0 oder 1x vorkommen
.	Findet 1 beliebiges Zeichen
[A-Za-z0-9_] oder \w	Findet ein Buchstabe, eine Zahl oder ein Unterstrich
\s	Findet ein beliebiges Leerzeichen
\	Escape-Zeichen, um Literale von Sonderzeichen zu finden, z.B. ein Dollar-Zeichen: \\$
(?!...)	„Negative Lookahead“: Stellt sicher, dass ein Ausdruck danach nicht vorkommt

Tabelle 2.1: Regex-Zeichenfolgen und ihre Funktionen [34]

In [Code 2.13](#) ist ein Beispiel eines `grep`-Befehls zu sehen, der einen regulären Ausdruck im Perl-Modus (`-P`) nutzt, um eine IP-Adresse in einer Datei zu finden, die nicht der Domain `webapp.local` zugeordnet ist. Die `pcre`¹⁸-Bibliothek gilt mittlerweile als die leistungsfähigste Regex-Implementierung, weshalb sich viele Programme darauf stützen, wenn Regex verwendet werden soll. Das Argument `--quiet` (oder `-q`) muss verwendet werden, um die normale Ausgabe des Befehls zu unterdrücken und einen „exit code“ zurückzugeben. Ist der exit code eines Befehls nicht 0, so wird er in einem `if`-Statement als `false` evaluiert.

```

1 IP=192.168.178.44
2 if grep --quiet -P "^(?!(webapp.local))" /etc/hosts; then
3     # IP in /etc/hosts gefunden, dem nicht webapp.local zugeordnet ist
4 fi

```

Code 2.13: `grep` in einem Bash `if`-Statement

2.5 MariaDB Server

MariaDB wurde von MySQL-Hauptentwickler „Michael Widenius“ von MySQL im Jahr 2008 abgespalten [35, S. 6 f.]. Es ist ein quelloffenes DBMS¹⁹-System für relationale Datenbanken,

¹⁸ Perl Compatible Regular Expression

¹⁹ Datenbankmanagementsystem

das robust und skalierbar ist und zusätzlich viele Speicher-Engines unterstützt, wodurch es für viele Anwendungsfälle geeignet ist [36].

2.6 Apache Webserver

Der Apache Webserver „`httpd`“ wurde 1995 von der „Apache Software Foundation“ eingeführt und ist heute einer der bekanntesten Webserver des Internets [37]. Ein Webserver ist ein Dienst auf einem physikalischen Server, der Dateien und Webseiten an Clients (z.B. Browser) bereitstellt.

In Abbildung 2.6 ist ein schematischer Prozess einer Anfrage an einen Webserver dargestellt. Ein Client (z.B. ein Browser) schickt eine [HTTP²⁰](#)-Anfrage an eine bestimmte Adresse im Internet ([IP](#) oder Hostname). Kommt diese Anfrage dann beim Webserver an, wird anhand von Informationen in dem [HTTP](#)-Paket geprüft, zu welchem „Virtual Host“ (vHost) der Webserver die Anfrage leiten soll. Danach wird die Anfrage zum Anwendungsserver umgeleitet. Das kann z.B. eine [PHP](#)-Anwendung oder eine einfache statische Webseite sein. Nachdem der Code für die Webseite erzeugt wurde (z.B. [HTML²¹](#)-Code), wird dieser Code vom Webserver in eine [HTTP](#)-Antwort verpackt und über das Internet bzw. das Netzwerk zurück an den Client geschickt. Dort nimmt der Browser die Anfrage entgegen und zeigt das angefragte Dokument bzw. die Webseite dem Benutzer auf dem Endgerät an.

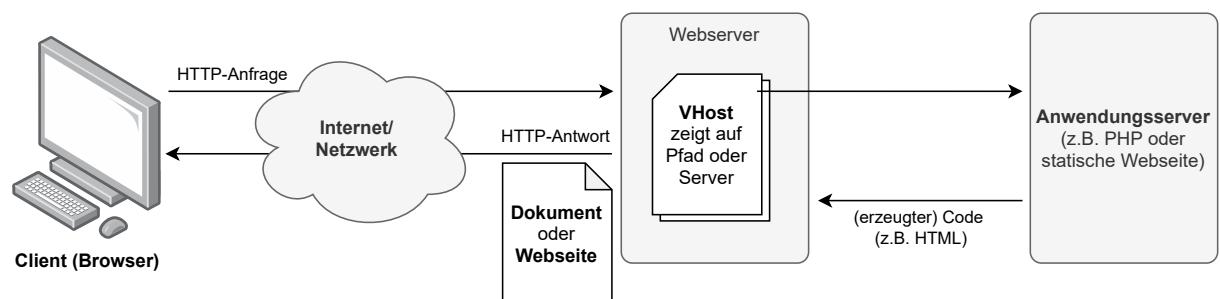


Abbildung 2.6: Funktionsweise eines Webservers

Eine vHost-Datei ist eine Konfigurationsdatei des Webservers, um eingehende [HTTP](#)-Anfragen an den richtigen Anwendungsserver weiterzuleiten und andere Einstellungen an einer Webseite vorzunehmen. In [Code 2.14](#) ist eine beispielhafte vHost-Datei eines Apache Webservers zu sehen.

²⁰ Hypertext Transfer Protocol

²¹ Hypertext Markup Language

```

1 <IfModule mod_ssl.c>
2   <VirtualHost *:443>
3     ServerName example.com
4     DocumentRoot /var/www/html
5
6     ErrorLog /var/log/apache/example.com.error.log
7     CustomLog /var/log/apache/example.com.access.log combined
8
9     SSLEngine on
10    SSLCertificateFile /etc/apache2/ssl/pub.pem
11    SSLCertificateKeyFile /etc/apache2/ssl/priv.pem
12  </VirtualHost>
13 </IfModule>
```

Code 2.14: Eine Virtual Host-Datei eines Apache Webservers

Die Zeile `<VirtualHost *:443>` gibt an, dass dieser vHost auf dem Standard-Port `443` für **HTTPS²²**-Verbindungen hört. Alle Anfragen, die an diesen Webserver gehen und nicht auf den Port `443` gehen, werden auch nicht von diesem vHost entgegen genommen. Mit dem Schlüsselwort `ServerName` kann die Anfrage einer Domain zugeordnet werden. Das bedeutet, dass alle Anfragen, die nicht auf `example.com` gehen, auch nicht von diesem vHost beantwortet werden. Die `DocumentRoot`-Direktive gibt ein Verzeichnis an, das als Einstiegspunkt der Webseite gilt und den Haupt-Dokumentenbaum darstellt, der im Internet verfügbar sein soll. Die zwei Schlüsselwörter `ErrorLog` und `CustomLog` geben den Pfad zu den Log-Dateien für die Fehler und die Anfragen des Webservers in Bezug auf diesen vHost an. [38]

Der Apache Webserver bietet verschiedene „Modules“ an, die über die Befehle `a2enmod` und `a2dismod` aktiviert bzw. deaktiviert werden können. Dadurch kann der Apache modular aufgebaut und nach Bedarf um Funktionen erweitert werden [39, S. 29]. Mit der Direktive `<IfModule mod_ssl.c>` wird geprüft, ob das `mod_ssl`-Module in der zugrunde liegenden Apache-Instanz aktiviert ist, nur dann wird dieser vHost aktiviert.

Das Modul `mod_ssl` bietet starke kryptographische Verschlüsselung über das **SSL²³**- und **TLS²⁴**-Protokoll. Es ermöglicht eine Webseite mit einem Zertifikat über **HTTPS** verschlüsselt bereitzustellen. Die `SSLEngine`-Direktive schaltet im vHost die Verschlüsselung der Verbindung aus oder ein. Mit `SSLCertificateFile` und `SSLCertificateKeyFile` werden die Pfade

²² Hypertext Transfer Protocol Secure

²³ Secure Sockets Layer

²⁴ Transport Layer Security

zu den „Private“- und „Public“-Keys angegeben. Diese sind notwendig für den „SSL Handshake“, der die verschlüsselte Kommunikation zwischen Client und Server sicherstellt [40]. [41]

2.7 mkcert zur Generierung von Zertifikaten

Um unter Linux ein SSL-Zertifikat zu erstellen, kann das Programm `openssl` verwendet werden. Wenn man keine Zertifizierungsstelle (engl. „Certificate Authority“, kurz CA²⁵) dazwischen geschaltet hat, die das Zertifikat verifiziert, dann nennt man das Zertifikat „self-signed“. Solche self-signed Zertifikate resultieren bei den meisten Browsern in Sicherheitswarnungen.

Um lokal ein valides Zertifikat auszustellen, das von einer Zertifizierungsstelle verifiziert wurde, kann das Programm `mkcert` verwendet werden. Nach der Installation dieses Tools wird mit dem Befehl `mkcert -install` eine lokale Certificate Authority erstellt und diese in dem sogenannten „Certificate Trust Store“ des Host-Systems registriert, damit das Betriebssystem `mkcert` als CA akzeptiert. Daraufhin kann mit `mkcert example.com` ein Public- und ein Private-Key-Paar generiert werden, das lokal durch die Zertifizierungsstelle für die Domain `example.com` validiert ist und somit nicht nur ein self-signed Zertifikat ist. Die Schlüssel für die Certificate Authority befinden sich nach dem Installations-Befehl von `mkcert` in einem bestimmten Verzeichnis auf dem Host. Dieses Verzeichnis wird `CAROOT` genannt. [42]

2.8 Die hosts-Datei

In vielen Betriebssystemen wie z.B. Linux, Windows oder MacOS gibt es eine `hosts`-Datei, die dafür sorgt, dass konfigurierte Domains auf bestimmte IP-Adressen aufgelöst werden. Sie fungiert also als lokaler DNS-Server. Einträge können dabei z.B. eine Adresse im lokalen Netzwerk auflösen, oder auch in einer öffentlichen IP-Adresse resultieren (vgl. [Code 2.15](#)). [43]

¹ 192.168.1.15 fileserver.local
² 212.15.4.69 some-webserver.com

Code 2.15: Beispielhafter Ausschnitt einer `hosts`-Datei

Bei einer Anfrage an die Adresse `some-webserver.com` in einem Browser mit einer konfigurierten `hosts`-Datei wie in [Code 2.15](#) wird zuerst diese Datei vom Betriebssystem durchsucht und dann auf die zugehörige IP-Adresse aufgelöst. Wenn dabei die Adresse nicht gefunden

²⁵ Certificate Authority

werden kann, dann wird die Auflösung der Adresse an den vom Host-System konfigurierten [DNS](#)-Server weitergeleitet.

Die Syntax der `hosts`-Datei ist unabhängig vom Betriebssystem und dementsprechend überall gleich. Die Speicherorte der Datei unterscheiden sich allerdings: Während sich die `hosts`-Datei sich bei Unix-Systemen unter `/etc/hosts` befindet, kann die Datei bei Windows 10 unter `C:\Windows\System32\drivers\etc\hosts` gefunden werden.

2.9 PHP

[PHP](#) ist eine bekannte Skript- und Programmiersprache, die für die Entwicklung von dynamischen Webanwendungen geeignet ist [44]. Sie wurde erstmals 1995 von Rasmus Lerdorf vorgestellt [45]. Zum Zeitpunkt dieser Arbeit ist die aktuelle Version 8.0, allerdings wird die Version 7.4 noch aktiv unterstützt [46]. [PHP](#) ist vor allem durch die breite Datenbankunterstützung und die große Anzahl an Erweiterungen so weit verbreitet.

[PHP](#) ist eine interpretierte Programmiersprache. Das bedeutet, dass der [PHP](#)-Code zunächst von einem Interpreter mitsamt Variablen auf dem Server geparsed und danach in einen Satz bestehender Opcodes („operation code“) übersetzt wird, also Maschinencode. Danach kann erst der daraus entstehende Bytecode auf dem Server ausgeführt werden. Das passiert für jede [PHP](#)-Datei bei jeder Anfrage. Dieser Ablauf ist schematisch in Abbildung 2.7 zu sehen.

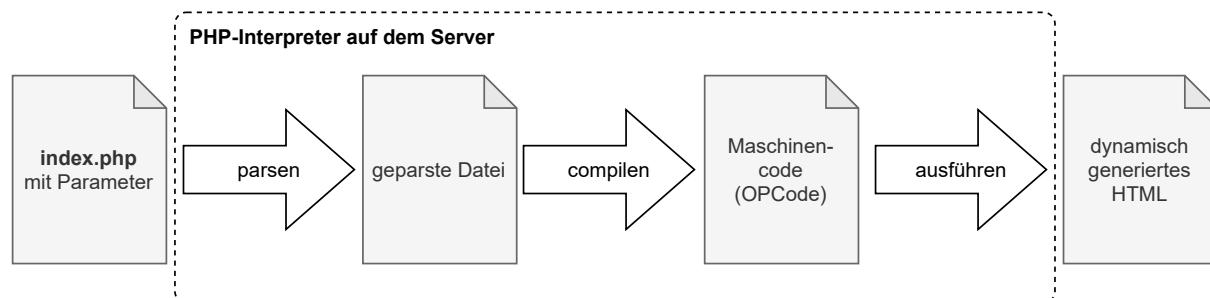


Abbildung 2.7: Funktionsweise des PHP-Interpreters

Für die Unterstützung von Transaktionen und Abfragen mit [SQL](#)²⁶-Servern, bietet [PHP](#) z.B. die Datenbanken-Erweiterung `mysqli` für MySQL-Server an. Damit können [SQL](#)-Statements über sogenannte „Prepared Statements“ ausgeführt werden ohne dabei Sicherheitslücken zu ermöglichen, wie z.B. [SQL](#)-Injections [47].

²⁶ Structured Query Language

Zur Bild-Manipulation mit [PHP](#) gibt es die Erweiterungen `gd` und `imagick`. Viele [PHP](#)-Frameworks, die in irgendeiner Form Funktionen rund um Bildbearbeitung eingebaut haben, stützen sich auf eine dieser beiden Erweiterungen. Ein Beispiel hierfür ist das Framework „Pimcore“ [48]. `imagick` ist eine Wrapper-Erweiterung der „ImageMagick“ API [49], ImageMagick wiederum eine quelloffene Bibliothek zur Erstellung, Bearbeitung und Zusammenstellung von Bitmap-Bildern [50].

Seit [PHP](#) 5.5.0 gibt es die Erweiterung `opcache`. Sie ist für die Verwaltung des sogenannten „Bytecode Cache“ zuständig. Ein Bytecode Cache speichert kompilierten [PHP](#)-Bytecode und verwendet diesen bei der Ausführung anstatt bei jeder Anfrage den [PHP](#)-Code zu lesen, parsen und compilieren. Der [PHP](#)-Interpreter kann dabei den [PHP](#)-Bytecode direkt aus dem [RAM](#) lesen und sofort ausführen, was die Performance einer komplexeren [PHP](#)-Anwendung drastisch verbessern kann. [51, S. 45 ff.]

2.9.1 Xdebug

Die [PHP](#)-Erweiterung „Xdebug“ wurde 2002 von Derick Rethans erstmals veröffentlicht. Xdebugs große Anwendungsbereiche sind „Step Debugging“ und „Profiling“ von [PHP](#)-Anwendungen. [52]

Der Profiler von Xdebug wird zur Performance-Analyse einer Anwendung genutzt. Er erlaubt eine genaue Einsicht der [RAM](#)- und [CPU](#)-Nutzung mit Zuordnung zu den Methodenaufrufen mitsamt ihren Argumenten und ihrer Laufzeit. Dadurch können Speicherlecks im Arbeitsspeicher („memory leaks“) und Performance-Engpässe („Bottlenecks“) in der Anwendung gefunden werden. [51, S. 260]

Step Debugging erlaubt das interaktive Durchlaufen eines Programmcodes und die Analyse von Variablen zur Laufzeit des Programms. Dabei kann zeilenweise der Code analysiert oder in bestimmte Funktionen des Programms gesprungen werden. Xdebug interagiert dafür mit einer [IDE](#), die zuvor konfiguriert werden und über das Protokoll [DBGp](#)²⁷ kommunizieren müssen. Danach kann in der [IDE](#) ein „Breakpoint“ gesetzt werden, um Xdebug mitzuteilen, an welchem Punkt im Programmcode der Step Debugger anhalten soll. Befinden sich [PHP](#)/Xdebug und die [IDE](#) auf dem selben Host, muss in der Konfigurationsdatei von [PHP](#) namens `php.ini` nur `xdebug.mode=debug` gesetzt werden. Befindet sich die Anwendung bzw.

²⁷ Debugger Protocol

der PHP-Interpreter auf einem anderen Host oder in einem Docker-Container, muss zusätzlich in der `php.ini` mit `xdebug.client_host` der Host spezifiziert werden, auf dem die IDE läuft [53]. Zur Auflösung des Hosts, auf dem Xdebug läuft, bietet Docker den speziellen DNS-Namen `host.docker.internal` an [54].

Nach Konfiguration von Xdebug muss nur noch die IDE für das Akzeptieren von Debug-Verbindungen und der Browser für das Senden der Umgebungsvariablen `XDEBUG_SESSION` konfiguriert werden. Dies geschieht am einfachsten über eine Browser-Erweiterung. Zudem muss der Debug-Port für eingehende Debug-Verbindungen in der IDE auf `9003` gesetzt werden, falls dies nicht schon die Standardeinstellung ist. Danach kann an beliebiger Stelle im Programmcode in der IDE ein Breakpoint gesetzt werden. Nach einer Anfrage des Browsers versucht die Xdebug-Erweiterung eine Verbindung zur IDE aufzubauen. In der IDE kann dann die lokale PHP-Datei der angefragten Datei des Servers zugeordnet werden, um die Ausführung des Codes zu stoppen, damit das Programm mittels Step Debugging analysiert werden kann. Dieser Ablauf ist in Abbildung 2.8 nochmals zu sehen. [53]

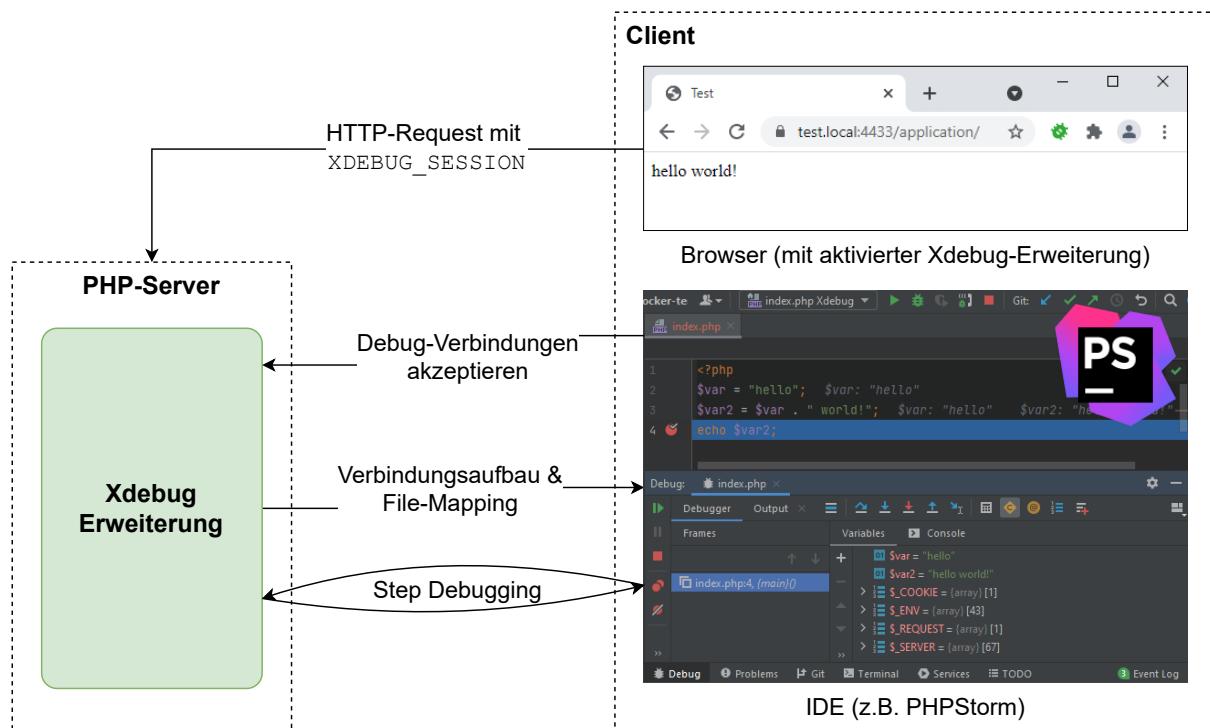


Abbildung 2.8: Verbindungsaufbau von Xdebug und IDE für Step Debugging

2.9.2 Composer

Composer ist ein Abhängigkeitsmanager (engl. „Dependency Manager“) für PHP-Anwendungen, der auf dem Terminal läuft. Die Aufgabe eines Dependency Managers ist es, die benötigten

Bibliotheken eines Projekts festzuhalten, herunterzuladen und zu verwalten. Er verwaltet dabei auch weitere geschachtelte Komponenten, die von den angegebenen Bibliotheken abhängen. Das Standard-Repository von Composer ist „Packagist“ [55]. Composer vereinfacht drastisch das Integrieren von Bibliotheken von Drittanbietern und das automatische Laden dieser Anwendungen (genannt „Autoloading“).

Unter dem Prozess des Autoloading versteht man im PHP-Kontext das automatische Laden von PHP-Klassen, ohne diese explizit mit einer „Require“- oder „Include“-Funktion zu laden (z.B. `require_once()`). Um zur Laufzeit Klassen zu laden, wenn diese zuvor noch nicht geladen wurden, bietet PHP stattdessen die flexiblere Funktion `spl_autoload_register()` an. Composer generiert automatisch eine `autoloader.php`, die diese Funktion verwendet und dann nur noch in die Anwendung eingebunden werden muss. Danach werden dynamisch die Klassen der Bibliotheken von Drittanbietern geladen, sofern diese in der Anwendung verwendet werden. [51, S. 88]

Composer speichert alle Abhängigkeiten in einer JSON-Datei, die sich im Wurzelverzeichnis des Projekts befindet: `composer.json`. Diese kann neben den abhängigen Komponenten u.a. den Namen und die Lizenz des Projekts beinhalten. In [Code 2.16](#) ist ein Beispiel einer `composer.json` zu finden.

```
1  {
2      "name": "hochwarth-it/some-project",
3      "license": "MIT",
4      "require": {
5          "monolog/monolog": "2.0.*"
6      }
7  }
```

Code 2.16: Beispiel einer `composer.json`

Unter dem `require`-Key sind alle abhängigen Komponenten mit Namen und einer speziellen Versionssyntax festgelegt („Semantic Versioning“). Die Syntax `MAJOR.MINOR.PATCH` besteht aus folgenden Grundregeln [56]:

- Erhöhe `MAJOR`, wenn inkompatible Änderungen vorgenommen wurden.
- Erhöhe `MINOR`, wenn rückwärts kompatible Funktionalitäten hinzugefügt wurden.

- Erhöhe `PATCH`, wenn Bugs gefixt wurden und die Software rückwärts kompatibel bleibt.

Composer bietet eine besondere Syntax, um bestimmte Versionsbereiche einer Komponente festzulegen. So kann z.B. die Version `2.0.*` mit dem „Wildcard“-Zeichen `*` angegeben werden, was äquivalent zu `>=2.0 <2.1` ist. Das bedeutet `2.0.*` aktualisiert bei Updates nur `PATCH`-Versionen dieser Komponente. Andere Operatoren sind z.B. `~` oder `^`. Mehr Infos dazu findet sich hier [57].

Soll z.B. die Komponente `monolog/monolog` als Abhängigkeit zu einem Projekt hinzugefügt werden, muss nur der Befehl `composer require monolog/monolog` eingegeben werden. Dieser Befehl sucht standardmäßig in dem öffentlichen Repository auf `packagist.org` nach der entsprechenden Komponente und lädt diese in das Unterverzeichnis `vendor`. Nach der Installation wird automatisch die Abhängigkeit der `composer.json` mit der speziellen Versionssyntax hinzugefügt und die exakte installierte Version in einer (neuen) Datei `composer.lock` festgehalten. Danach ist es empfehlenswert, die `composer.lock` in die Versionskontrolle einzufügen (siehe [Abschnitt 2.3](#)), damit andere Entwickler bei diesem Projekt exakt die gleichen Versionen der Abhängigkeiten installieren können. Das passiert nämlich über den Befehl `composer install`. Dieser Befehl überprüft, ob alle Komponenten im `vendor`-Verzeichnis mit den Versionen in der `composer.lock` übereinstimmen. Sollen alle oder einzelne Abhängigkeiten eines Projekts gemäß der angegebenen Versionssyntax aktualisiert werden, kann der Befehl `composer update` genutzt werden. [58]

Effektiv abstrahiert Composer dadurch Dependency Management und Autoloading in einer Software.

2.10 npm und Grunt

npm

[npm²⁸](#) ist ein Paket-Verwaltungssystem (engl. „Package Manager“) für Pakete in Node.js- und JavaScript-Laufzeitumgebungen. Es wurde 2009 von Isaac Z. Schlueter entwickelt und wird heute von der `npm, Inc.` verwaltet. Node.js ist eine JavaScript-Laufzeitumgebung, die auf der V8 JavaScript-Engine von Chrome aufbaut [59]. Ähnlich wie Composer (vgl. [Unterabschnitt 2.9.2](#)) hat `npm` eine öffentliche Anlaufstelle `npmjs.com` („`npm` Registry“), um Pakete öffentlich zu

²⁸ Node Package Manager

speichern und anderen Entwicklern zugänglich zu machen. Zudem hat es auch eine [CLI](#), mit dem Pakete installiert, aktualisiert und veröffentlicht werden können. [60]

Damit die [npm-CLI](#) die Abhängigkeiten festhalten kann, legt [npm](#) eine `package.json` an, ähnlich wie die `composer.json`. Die Versionssyntax bei [npm](#) ist die gleiche wie bei Composer [61]. Nach Ausführung des Befehls `npm install` im Verzeichnis der `package.json` liest [npm](#) alle abhängigen Softwarepakete ein und sucht das jeweilige Repository in der öffentlichen [npm](#)-Repository. Diese abhängigen Softwarepakete werden dann heruntergeladen und in dem Unterordner `node_modules` gespeichert. Außerdem wird ähnlich wie bei Composer beim Installationsprozess der Pakete eine `package-lock.json` angelegt, die die tatsächlich installierten Versionen der abhängigen Pakete festhält, um eine reproduzierbare Umgebung zu gewährleisten. Um ein Software-Paket einem Projekt hinzuzufügen, muss einfach nur `npm install <name>` ausgeführt werden. Dies fügt die Abhängigkeit der `package.json` hinzu, installiert das Paket in den `node_modules`-Ordner und hält die genaue Version des installierten Pakets mitsamt dessen Abhängigkeiten in der `package-lock.json` fest.

Grunt

Grunt ist ein JavaScript „Task Runner“, dessen Ziel es ist, sich wiederholende Aufgaben mit JavaScript zu automatisieren. Darunter z.B. die Kompilierung von [SCSS²⁹](#) in [CSS³⁰](#)-Dateien mit „Sass“, die Minimierung der Größe von JavaScript-Dateien mit „uglify“ oder das Entfernen ungenutzter [CSS](#)-Deklarationen mit „purgecss“. Diese Aufgaben und ihre Konfigurationen können mit einer `Gruntfile.js` festgelegt werden, welche von der Grunt-[CLI](#) z.B. mit dem einfachen Befehl `grunt` eingelesen wird, um danach die „Tasks“ auszuführen. Grunt ist sehr einfach mit einer Auswahl von hunderten Plugins erweiterbar, um so viele Aufgaben in der Frontend-Entwicklung zu automatisieren. [62]

2.11 MailHog

MailHog ist ein E-Mail-Tool für Entwicklungszwecke, geschrieben in der Sprache „Go“. Nach dem Starten von MailHog wird ein [SMTP³¹](#)-Server auf dem Port `1025` und ein [HTTP](#)-Server auf dem Port `8025` gestartet. Alle E-Mails, die auf den Port `1025` dieses Hosts geschickt

²⁹ Sassy Cascading Style Sheets

³⁰ Cascading Style Sheets

³¹ Simple Mail Transfer Protocol

werden, können nun von MailHog empfangen werden. MailHog speichert die E-Mails im **RAM** und bietet eine Web-Oberfläche zur Einsicht dieser E-Mails unter `http://localhost:8025` an. Ein Screenshot dieser Oberfläche ist in [Abbildung 2.9](#) zu sehen. [63]

MailHog kann so genutzt werden, um E-Mails zu empfangen, die während Test-Prozessen aus einer Entwicklungsumgebung gesendet werden.

The screenshot shows the MailHog web interface. At the top, there is a header with a MailHog icon, a search bar, and a GitHub link. On the left, a sidebar shows a green 'Connected' status, an 'Inbox (3)' notification, and a 'Delete all messages' option. Below the sidebar, a user profile for 'Jim' is displayed, including a bio: 'Jim is a chaos monkey.' and a link to 'Find out more at GitHub.', followed by a 'Enable Jim' button. The main area shows a list of three messages:

From	Subject	Date
Twitter Some Mailhogger	Follow Kevin Bridges, Andy Murray and STV News on Twitter!	a few seconds ago
Rhianna Jones Sophie Stiller	MailHog test message	a few seconds ago
Joe Jones Mark Thomas	MailHog test message	a few seconds ago

Abbildung 2.9: Web-Oberfläche von MailHog [63]

3 Praktische Lösung

Dieses Kapitel beginnt mit der Analyse der Anforderungen, woraufhin die Lösung des Problems mit einem Überblick über die verwendete Architektur und allen wichtigen zugehörigen technischen Details erörtert wird. Danach folgt eine Analyse zur Leistung und zum Kosten-Unterschied der entwickelten Anwendung im Vergleich zur alten Lösung.

3.1 Anforderungsanalyse

Um die Anforderungen an das Template-Repository aufstellen zu können, muss zuerst die Zielgruppe der Software festgelegt werden. Diese ist mit den 15 Entwicklern der Hochwarth IT sehr stark beschränkt. Ein weit verbreitetes Ziel von Software-Entwicklern ist es, so viel wie möglich im Entwicklungsprozess zu automatisieren, um so schnellstens Ergebnisse erzielen zu können. Ein weiteres Ziel der Entwickler ist eine gute „Developer Experience“ ([DX](#)). Darunter versteht man das Empfinden eines Entwicklers, während dieser mit einem bestimmten Framework oder Tool arbeitet und wie einfach es ist, damit zu arbeiten. Die Developer Experience ist also eine Benutzererfahrung („User Experience“) speziell für Entwickler [\[64\]](#).

Das Template-Repository soll helfen, ein Projekt bei der Einrichtung schneller zum Laufen zu bekommen und systembedingte Fehler durch eine einheitliche Entwicklungsumgebung auf jedem Entwickler-Rechner zu vermeiden.

Bevor also die Anforderungen an das Repository festgelegt werden können, muss der bisherige Ablauf zum Einrichten eines Projekts analysiert werden. Es folgt ein möglicher Ablauf zum Einrichten eines [PHP](#)-Projekts, wie es bei Hochwarth IT gängig ist. Das Projekt befindet sich dabei unter Windows 10 im Pfad `C:\Users\Name\Projects\sample` und soll unter dem lokalen Hostnamen `sample.local` mit [SSL](#)-Zertifikat laufen.

1. projektspezifische Einstellungen in der [DBMS](#) und im Webserver vornehmen, falls notwendig
2. lokalen Datenbank- und Webserver starten
3. Projekt klonen über den Git-Befehl `git clone`
4. `hosts`-Datei auf dem Host für `sample.local` konfigurieren (siehe [Abschnitt 2.8](#))

5. lokal valides [SSL](#)-Zertifikat mit `mkcert` ausstellen
6. vHost-Datei des lokalen Webservers konfigurieren: lokale Domain `sample.local` dem Pfad zum Webroot-Ordner des Projekts zuordnen und Pfade zu den Key-Dateien des [SSL](#)-Zertifikats angeben
7. projektspezifische [PHP](#)-Version einstellen und [PHP](#)-Erweiterungen installieren
8. projektspezifische Konfigurationsdatei anpassen für die Datenbankverbindung
9. Datenbank vom Live-System herunterladen und auf dem Datenbank-Server in eine neue Datenbank importieren
10. Abhängige Software-Komponenten des Projekts installieren (z.B. mit Composer: siehe [Unterabschnitt 2.9.2](#))
11. Projekt-[URL](#)¹ `https://sample.local` im Browser aufrufen

Notizen zu obigem Ablauf

Ein [SSL](#)-Zertifikat und die Bereitstellung über [HTTPS](#) ist nicht unbedingt notwendig für jedes lokale Webentwicklungs-Projekt. Für die Entwicklung mit einem lokalen Webserver an einem [PHP](#)-Projekt macht es keinen Unterschied, ob die Seite über [HTTP](#) oder [HTTPS](#) bereitgestellt wird. Soll allerdings in der Applikation ein Service-Worker nicht über den [DNS](#)-Namen `localhost` oder `127.x.y.z` verwendet werden, ist eine verschlüsselte Verbindung notwendig, weil sich der Service-Worker sonst nicht in einem „Secure Context“ befindet [65] [66]. Ein Service-Worker ist eine Technologie von Browsern, die in einem separaten Thread auf dem Endgerät im Hintergrund läuft, um z.B. Dateien offline zur Verfügung zu stellen. Dieser „Offline First“-Ansatz findet sich oft in mobilen Web-Applikationen [67]. Außerdem kommt die Bereitstellung des Projekts über [HTTPS](#) näher an die Produktiv-Umgebung auf dem Live-Server ran, was auch Ziel dieses Template-Repositorys ist. Dadurch treten weniger Fehler auf, die nur in der Entwicklungsumgebung bzw. nur auf der Produktiv-Umgebung vorkommen.

Theoretisch wäre das Projekt auch über `localhost` anstatt über Domain `sample.local` erreichbar. Eine spezielle Domain wie `sample.local` bietet allerdings

¹ Uniform Resource Locator

einen besseren Überblick und schnelleren Zugriff auf die lokalen Projekte. Außerdem bietet es die Möglichkeit, mehrere Projekte zeitgleich auf einem Host lokal laufen zu lassen, ohne auf einen Port pro Projekt ausweichen zu müssen.

Es gibt mehrere Möglichkeiten die [PHP](#)-Version für ein lokales Projekt einzustellen. Das ist sinnvoll und notwendig, weil verschiedene Projekte andere [PHP](#)-Versionen benötigen können. Das Problem dabei ist, dass man alle nötigen Versionen manuell lokal installieren und diese pro Projekt konfigurieren muss. Das ist ein sehr aufwändiger Prozess und kann bei Nichtbeachtung der gerade aktiven [PHP](#)-Version zu ungewünschten Seiteneffekten führen, die z.B. nur lokal auftreten und nicht auf dem Live-System.

Ein weiterer aufwändiger Prozess, der während der Entwicklung an einem Projekt auftreten kann, ist die Installation der korrekten Xdebug-Version und dessen Konfiguration, damit in der [PHP](#)-Anwendung vernünftig nach Programm-Fehler gesucht werden kann.

Um Webanwendungen auf mobilen Endgeräten zu testen oder Fehler zu reproduzieren, die z.B. nur unter iOS auftreten, muss die Anwendung im lokal angeschlossenen Netzwerk des Hosts verfügbar sein.

In komplexeren Webanwendungen müssen häufig automatisiert E-Mails an die Nutzer versandt werden. Damit in Entwicklungsumgebungen während des Testens der Anwendung nicht aus Versehen E-Mails an echte Nutzer verschickt werden, sollten diese vom Host-System automatisch abgefangen werden.

Viele Projekte von Hochwarth IT benötigen neben [PHP](#) auch weitere Software wie Composer, [npm](#) oder Grunt. Diese müssen manuell auf dem Host-System installiert werden.

Die Maßnahmen zur automatisierten Lösung der genannten Probleme sind in nachfolgendem Absatz zu finden.

Nach Klärung des üblichen Ablaufs, wie ein Projekt einzurichten ist, stellt sich nun die Frage, welche Prozesse mit dem zu erstellenden Repository basierend auf Docker automatisiert werden können, um den Entwicklern Zeit und dem Unternehmen Geld zu sparen. In folgender Aufzählung sind basierend auf dem oben beschriebenen Ablauf die Funktionen dargestellt, die das Template-Repository bieten soll:

- automatische Konfiguration der `hosts`-Datei für eine spezifische Domain
- automatisierte Ausstellung eines lokal validen [SSL](#)-Zertifikats
- Konfiguration der vHost-Datei voreinstellbarer Webroot-Ordner
- automatische Installation der projektabhängigen [PHP](#)-Version und [PHP](#)-Erweiterungen
- Import einer oder mehrerer Datenbank-Dateien in den Datenbank-Container beim initialen Container-Start
- automatisierte Konfiguration der projektspezifischen Konfigurationsdateien basierend auf Umgebungsvariablen
- automatische Installation der korrekten Xdebug-Version
- Integration von Xdebug mit der [IDE](#) „[PHPStorm](#)“ von JetBrains
- automatisierte, konfigurierbare Installation von Composer, npm und Grunt
- automatische Ausführung von `composer install` und `npm install` in gewünschten Verzeichnissen
- Verfügbarkeit der Anwendung innerhalb des physikalisch angeschlossenen Netzwerks des Hosts über die [IP](#)-Adresse des Hosts
- Abfangen ausgehender E-Mails eines Containers
- nach initialer Einrichtung („Dockerisierung“) des Projekts mit dem Template-Repository, soll das Projekt mit einem Befehl gestartet werden können
- Benutzer-Dokumentation der Nutzung und der Dockerisierung

Nach einmaliger Dockerisierung eines Projekts mit dem Template-Repository, das die beschriebenen Funktionen implementiert und nach Speicherung der Abhängigkeiten in Git kann das Projekt auf jedem Entwickler-PC ohne großen Aufwand in Docker-Container gestartet werden. Im Vergleich zu oben beschriebenem Ablauf, sieht der vereinfachte Ablauf zum lokalen Projektstart wesentlich kürzer aus, nämlich wie folgt:

1. Projekt klonen über den Git-Befehl `git clone`

2. Datenbank vom Live-System herunterladen und in `SQL`-Ordner (`./_docker/db/sql/`) des Projekts verschieben
3. `.env`-Datei mit den nötigen Umgebungsvariablen konfigurieren
4. `docker-compose up` im Projektverzeichnis ausführen zum Start der Anwendung
5. Projekt-[URL `https://sample.local`](https://sample.local) im Browser aufrufen

3.2 Funktionen des Template-Repositorys

Die Docker-Entwicklungsumgebung, die mit diesem Template-Repository erstellt werden kann, besteht aus zwei Containern: `db` und `web`. Der `db`-Container enthält die Datenbank mitsamt Datenbankeinstellungen und basiert auf einem kleinen MariaDB-Image, welches wiederum auf dem sehr kleinen Linux-Image „Alpine-Linux“ basiert [68]. Der `web`-Container besteht hauptsächlich aus einem Apache-Webserver und der `PHP`-Laufzeitumgebung und basiert auf dem `debian`-Image [69]. Die Trennung von Datenbank und Webserver in verschiedenen Containern bringt zum einen den Vorteil, dass die zugrundeliegende Software einfacher aktualisiert werden kann und zum anderen wird das Container-Setup dadurch dem Prinzip der Microservices gerecht, wie bereits beschrieben in [Unterabschnitt 2.1.1](#).

Die Konfigurationen der Container sind in der `docker-compose.yml` im Wurzelverzeichnis des Projekts festgehalten. Lokale Konfigurationen werden hauptsächlich in der `.env`-Datei im Wurzelverzeichnis festgelegt.

Beim `command`-Key ist in der `docker-compose.yml` ein `startup.sh`-Skript angegeben, das beim Start innerhalb des `web`-Containers ausgeführt wird und als Verteiler für andere Skripte dient, die beim Start ausgeführt werden sollen.

In [Abbildung 3.1](#) ist der Verzeichnisbaum des Template-Repositorys mit ergänzenden Kommentaren zu den Ordnern und Dateien zu sehen. Die Abbildung dient zur Orientierung für die nächsten Abschnitte.

In [Abbildung 3.2](#) ist eine schematische Übersicht der Architektur des Template-Repositorys und das Zusammenspiel der einzelnen Komponenten zu sehen.

```

/.          # Root-Verzeichnis
+-- docker/ # Docker-Template-Repository
|   +-- db/  # Datenbank-Container
|   |   +-- scripts/      # enthält Skript zum DBs importieren
|   |   +-- sql/         # zu importierende DBs
|   |   +-- my.cnf        # DB-Einstellungen
|   |   +-- Dockerfile
|   +-- web/  # Web-Container
|       +-- additional-inis/ # PHP-Einstellungen
|       +-- certs/        # generiertes Zertifikat
|       +-- scripts/      # Bash-Skripte für Container-Start
|       +-- sites-available/ # Apache VHost-Dateien
|       +-- tmpl/         # Vorlagendateien
|       +-- Dockerfile
+-- .env      # lokale Umgebungsvariablen
+-- .env.sample # Standardwerte für Umgebungsvariablen
+-- docker-compose.yml

```

Abbildung 3.1: Verzeichnisstruktur des Template-Repositorys

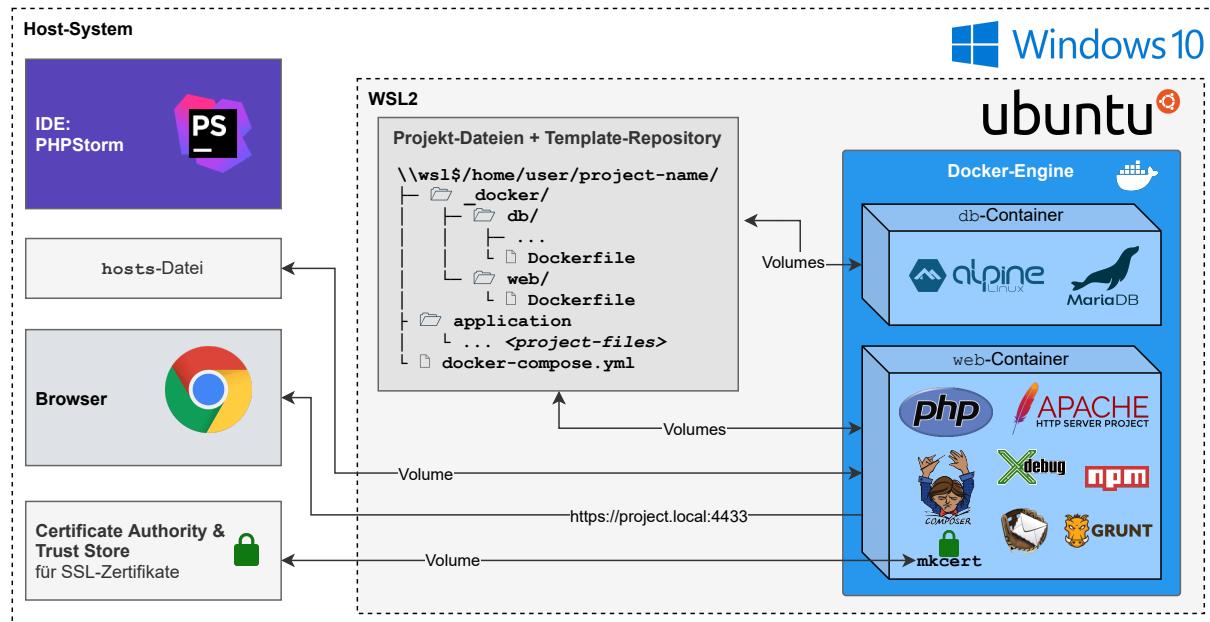


Abbildung 3.2: Projekt-Architektur der Entwicklungsumgebung für das Template-Repository

Die verwendete Entwicklungsumgebung zur Entwicklung und zum Testen des Template-Repositorys ist Windows 10 mit WSL2-Docker-Backend. Es gibt zwar auch Entwickler bei Hochwarth IT, die MacOS oder Linux als Host-System verwenden, aber dies soll nicht Teil der Betrachtung in dieser Arbeit sein. Zudem ist durch den in WSL2 befindlichen Linux-Kernel ein Großteil der Funktionalitäten dieses Template-Repositorys auch unter Linux und MacOS gegeben. Das Template-Repository befindet sich zusammen mit den projektspezifischen Dateien in einem Home-Verzeichnis im WSL Dateisystem. Diese Dateien sind über verschiedene Docker Volumes mit den Containern `db` und `web` verbunden. Die IDE PHPStorm, der Browser und die

Certificate Authority laufen auf dem Host-System. Die lokale `hosts`-Datei des Host-Systems ist ebenfalls über ein Volume mit dem `web`-Container verbunden.

3.2.1 Projekt in Container verschieben

Im Folgenden wird die bereits erwähnte Dockerisierung genauer aufgeführt. Alle Dateien, die dabei hinzugefügt werden, werden danach in das Versionierungssystem Git übernommen und über den Befehl `git push` auf das Remote-Repository hochgeladen, damit die Abhängigkeiten des Kundenprojekts gespeichert sind. Das umfasst alle Dateien, die direkt abhängig von projektspezifischen Gegebenheiten sind, also z.B. die `PHP`-Version oder die zu installierenden `PHP`-Erweiterungen. Das umfasst u.a. die `docker-compose.yml` oder die `Dockerfiles`.

Das Template-Repository ist auf einem Git-Server gespeichert, wo es heruntergeladen werden muss, bevor es genutzt werden kann. Um es in einem vorhandenen Kundenprojekt zu installieren, müssen zuerst alle Dateien in das Wurzelverzeichnis kopiert werden. Sollte sich dort bereits eine `.gitignore`-Datei befinden, muss die `.gitignore` aus dem Template-Repository mit der anderen zusammengeführt werden.

Konfiguration PHP

Die `PHP`-Version, die im `web`-Container installiert werden soll, wird in der `Dockerfile` unter `./_docker/web` gesetzt. In [Code 3.1](#) wird die `PHP`-Version über den Standardwert des Arguments `PHP_VERSION` festgelegt. In diesem Beispiel ist erkennbar, dass der `web`-Container auf dem `php`-Image mit dem Tag `5.6-apache` basiert.

```
1 ARG PHP_VERSION=5.6
2 FROM php:${PHP_VERSION}-apache
3
4 COPY --from=mlocati/php-extension-installer /usr/bin/install-php-extensions
   → /usr/local/bin/
5 #
6 RUN install-php-extensions opcache
7 RUN install-php-extensions gd
```

Code 3.1: Festlegung der PHP-Version und Installation PHP-Erweiterungen

In [Code 3.1](#) ist auch zu erkennen, dass mit der Binary-Datei `install-php-extensions` die nötigen `PHP`-Erweiterungen in dem Container installiert werden. `install-php-extensions`

wird vorher über das Docker Hub Repository `mlocati/php-extension-installer` [70] mit dem `COPY`-Befehl installiert.

Das Template-Repository bietet auch die Möglichkeit sämtliche Variablen der `php.ini` zu bearbeiten. Dazu muss nur eine weitere `.ini`-Datei in `./_docker/web/additional-inis` mit dem entsprechenden Inhalt abgelegt werden. In der `docker-compose.yml` ist im Service `web` ein Docker Volume festgelegt, das den Inhalt dieses Verzeichnisses einem speziellen Verzeichnis im Container zuordnet, das PHP beim Einlesen der `php.ini` berücksichtigt und entsprechend die Variablen der `.ini`-Dateien daraus überschreibt (vgl. [Code 3.2](#)).

```
1  web:  
2    volumes:  
3      - ./_docker/web/additional-inis:/usr/local/etc/php/conf.d/additional-inis  
4      - ./:/var/www/html  
5      - ./_docker/web/sites-available:/etc/apache2/sites-available
```

Code 3.2: Docker Volume-Konfiguration des `web`-Containers

Konfiguration Webserver

Wie in [Code 3.1](#) bereits zu sehen ist, wird der Apache Webserver durch das PHP-Image im `web`-Container mit installiert. Es wäre zwar mit [PHP-FPM](#)² möglich, den Webserver vom PHP-Container zu trennen, allerdings ist das nur notwendig, wenn in einem Projekt mehrere PHP-Versionen verwendet werden müssen [71]. Außerdem dient eine geringere Anzahl an Containern pro Projekt der Verständlichkeit für Entwickler, die noch nicht so viel mit Docker gearbeitet haben.

In der `docker-compose.yml` sind noch weitere Docker Volumes definiert. Das wichtigste Volume ist `./:/var/www/html`, welches das aktuelle Verzeichnis `./` dem Arbeitsverzeichnis `/var/www/html` des `web`-Containers zuordnet. Dadurch werden die Dateien des Kundenprojekts erst im Container verfügbar gemacht, wodurch der Webserver dann die Dateien bereitstellen kann. Standardmäßig wird durch dieses Volume das Wurzelverzeichnis des Projekts vom Apache Webserver bereitgestellt. Befindet sich allerdings das `DocumentRoot`-Verzeichnis des Projekts in einem anderen Unterordner, muss dies in den vHost-Dateien von Apache angepasst werden. Diese befinden sich in `./_docker/web/sites-available` und sind über ein Volume

² FastCGI Process Manager

in dem `web`-Container unter `/etc/apache2/sites-available` verfügbar, wie in [Code 3.2](#) zu sehen ist.

Konfiguration Datenbank-Server

Manche Projekte erfordern Anpassungen in den Datenbank-Einstellungen. Um im `db`-Container Einstellungen für MariaDB vorzunehmen, kann die Datei `/etc/my.cnf.d/my.cnf` bearbeitet werden. Diese Datei ist nicht über ein Docker Volume mit dem Container verbunden, sondern wird beim Build-Prozess mit der `Dockerfile` in den Container kopiert (vgl. [Code 3.3](#)). Das ist notwendig, weil MariaDB die Konfigurationsdatei bei falschen Berechtigungen ignoriert. Nach dem Kopierprozess wird nämlich die Berechtigung der Datei im Container mit `chmod` auf `0444` gesetzt. Das setzt die Berechtigung der Datei auf schreibgeschützt.

```
1 COPY ./my.cnf /etc/my.cnf.d/my.cnf  
2 RUN chmod 0444 /etc/my.cnf.d/my.cnf
```

Code 3.3: Kopieren der Datenbank-Konfigurationsdatei `my.cnf`

Vorlagendateien

Viele [PHP](#)-Frameworks haben Konfigurationsdateien, die standardmäßig über eine `.gitignore`-Datei von Git ignoriert werden. Sie dienen als lokale Konfiguration des Projekts, die nicht in Git gespeichert werden soll. Ein klassisches Beispiel hierfür ist die Verbindungs-Konfiguration zum lokalen Datenbank-Server. Es macht keinen Sinn, die Verbindungsinformationen wie Datenbank-Benutzer, Datenbank-Passwort und Datenbankname in Git zu speichern, da dies individuell auf dem lokalen Datenbank-Server festgelegt werden muss. Da allerdings das Template-Repository auch den Datenbank-Server in einem Container umfasst und diese Verbindungsinformationen zur Laufzeit bekannt sind, können die lokalen Konfigurationsdateien beim Start der Container automatisch generiert werden.

Dazu muss in der `docker-compose.yml` eine Umgebungsvariable mit dem Prefix `TEMPLATE_` angelegt werden, dessen Wert den Pfad der Vorlagendatei in `./_docker/web/tmp1/` dem Pfad der Datei in der Anwendung zuordnet (vgl. [Code 3.4](#) und Verzeichnisstruktur in [Abbildung 3.1](#)).

```

1 web:
2   environment:
3     TEMPLATE_DB_CONFIG: "template_db_config.php:./app/db_config.php"

```

Code 3.4: Konfiguration einer Vorlagendatei in der `docker-compose.yml`

Die Vorlagendatei `template_db_config.php` wird vom Entwickler in `./_docker/web/tmp1` angelegt und sollte von der originalen Konfigurationsdatei kopiert werden. Danach können mit der Bash „Parameter Expansion“-Syntax `${VARIABLE}` (siehe [Abschnitt 2.4](#)) dynamisch Umgebungsvariablen in die Template-Datei eingefügt werden (vgl. [Code 3.5](#)). Die relevanten Umgebungsvariablen werden in der `.env`-Datei definiert. Ist einer der Variablen nicht angegeben, können in der Template-Datei auch Standard-Werte angegeben werden. Die Syntax hierfür ist an die Bash-Syntax `${VAR:-default}` angelehnt.

```

1 return [
2   "database" => [
3     "host" => '${DOMAIN:-test.local}', // myapp.local, wenn DOMAIN gesetzt
4     "user" => '${DATABASE_USER_MAIN}', // root
5     "password" => '${DATABASE_PASS_MAIN}', // s3cr3tP4ssw0rd
6     "database_name" => '${DATABASE_NAME_MAIN}' // app_db
7   ]
8 ];

```

Code 3.5: Beispiel einer Konfigurations-Template-Datei

Die Umgebungsvariable `DOMAIN` wird in der `docker-compose.yml` im Container `db` als `hostname` angegeben (vgl. [Code 3.6](#)). Das bedeutet, dass der `db`-Container in allen Containern dieses Docker-Netzwerks verfügbar ist. Deshalb ist es möglich, eine Verbindung zum Datenbank-Server im `db`-Container ausgehend vom Web-Container über die konfigurierte `DOMAIN` herzustellen.

```

1 db:
2   hostname: ${DOMAIN}

```

Code 3.6: Hostname-Konfiguration des `db`-Containers

Das Skript `replace-templates.sh` ersetzt dann beim Start des Containers die Template-Dateien, wie es in der `docker-compose.yml` konfiguriert ist. In [Code 3.7](#) ist ein Auszug des Skripts zu sehen, teilweise mit Pseudocode zur einfacheren Darstellung. Für alle `TEMPLATE_`-

Umgebungsvariablen sucht das Skript in den angegebenen Dateien mit einem regulären Ausdruck (siehe [Abschnitt 2.4](#)) sämtliche Vorkommnisse von Variablen in der Form `${VAR}` oder `${VAR:-default}`. Danach werden diese mit dem entsprechenden Wert der Umgebungsvariable ersetzt und am Ende wird die konfigurierte Ziel-Datei geschrieben.

```

1  for tmplVarName in "${!TEMPLATE_@}"; do # für alle "TEMPLATE_*"-Variablen
2      tmplVarValue="${!tmplVarName}"
3      fc= # file content von ./app/db_config.php
4      # für alle Variablen ${VAR}/${VAR:-default} in template_db_config.php:
5      for toReplace in $(echo "$fc" | grep -P -o
6          '\$\{[A-Za-z0-9_]+(\}|\\:\\-.+?\})'); do
7          varName= # Name der Umgebungsvariable, z.B. DOMAIN
8          varValue=${!varName} # Wert der Umgebungsvariable, z.B. myapp.local
9          # wenn ${VAR} (Regex):
10         fc="${fc//\$\{$varName\}/$varValue}"
11         # ansonsten (wenn ${VAR:-default}):
12         default=${toReplace:${#varName}+4:-1}
13         # wenn $varValue gesetzt:
14         fc="${fc//\\\$\\{$varName\}/$varValue}"
15         # ansonsten:
16         fc="${fc//\$\\{$varName\}:\\-$default\\}/$varValue"
17     done
18     # file content in Konfigurationsdatei (Pfad dynamisch aus $tmplVarValue)
19     echo "$fc" > ./app/db_config.php
done

```

Code 3.7: Auszug aus dem Bash-Skript `replace-templates.sh` mit Pseudocode

Build-Argumente

Das Template-Repository erlaubt es, in der `docker-compose.yml` über Docker Build-Argumente `npm`, Composer und Grunt zu installieren. Hierfür müssen nur die entsprechenden Argumente auf `"true"` gesetzt werden, wie in [Code 3.8](#) gezeigt.

```

1 web:
2   build:
3     args:
4       INSTALL_COMPOSER: "true"
5       INSTALL_NPM: "true"
6       INSTALL_GRUNT: "false"

```

Code 3.8: Build-Argumente zur dynamischen Installation verschiedener Komponenten

Diese Docker Build-Argumente werden nur beim Build-Prozess des `web`-Containers in der zugehörigen `Dockerfile` evaluiert. Dementsprechend muss zur Anwendung der Änderung einer dieser Argumente der `web`-Container neu erstellt werden. Dies ist mit dem Befehl `docker-compose build web` möglich.

Ein Beispiel, wie die Build-Argumente in der `Dockerfile` genutzt werden, ist in [Code 3.9](#) zu sehen.

```

1 ARG INSTALL_COMPOSER="true" # wenn nicht gesetzt, ist es "true"
2 RUN if [ "$INSTALL_COMPOSER" = "true" ]; then \
3   # Installations-Befehle für Composer
4 fi

```

Code 3.9: Build-Argument `INSTALL_COMPOSER` zur konditionellen Installation von Composer

Docker Compose erlaubt keine Boolean-Werte für Umgebungsvariablen, weil der YAML-Parser Werte wie `yes` oder `true` nicht unverändert lässt, was zu Verwirrungen führen kann [\[72\]](#).

Composer befindet sich im `web`-Container, weil es eine starke Abhängigkeit von der `PHP`-Installation hat und nur damit funktionieren kann. `npm` hat zwar keine Abhängigkeit zu `PHP`, allerdings benötigt nicht jedes Kundenprojekt Node.js und `npm` und deshalb wäre ein zusätzlicher `npm`-Container, der gegebenenfalls nicht zum Einsatz kommt, nur verwirrend für unwissende Entwickler, die mit dem Template-Repository arbeiten. So ein zusätzlicher Container könnte nur die bereits in [Abschnitt 3.1](#) angesprochene Developer Experience verschlechtern. Da sich Node.js also im `web`-Container befindet, muss Grunt auch dort installiert werden, da es auf Node.js basiert.

Automatische Installation von abhängigen Bibliotheken

Zusätzlich bietet das Repository die Möglichkeit in festgelegten Verzeichnissen die Befehle `composer install` und `npm install` automatisch auszuführen. Auf diese Weise kön-

nen beim initialen Start des Containers abhängige Bibliotheken automatisiert über Composer und npm installiert werden. Wie in [Code 3.10](#) dargestellt, sind diese Verzeichnisse mit den Umgebungsvariablen `COMPOSER_INSTALL_PATHS` und `NPM_INSTALL_PATHS` in der `docker-compose.yml` konfigurierbar. Dabei können ein oder mehrere Verzeichnisse mit `:` getrennt angegeben werden.

```

1 web:
2   environment:
3     COMPOSER_INSTALL_PATHS: ./../app/other/component:/app/another/one
4     NPM_INSTALL_PATHS: ./
```

Code 3.10: Angabe von Verzeichnissen zur Automation von Installations-Prozessen

Das zugehörige Skript `composer-npm-install.sh` iteriert über alle Pfade in den Umgebungsvariablen und führt dort jeweils `composer install` bzw. `npm install` aus (vgl. [Code 3.11](#)). In der `for`-Schleife ist die Verwendung von Command Substitution und einer Pipeline zu sehen (siehe [Abschnitt 2.4](#)).

```

1 if [ -n "$COMPOSER_INSTALL_PATHS" ]; then
2   for path in $(echo "$COMPOSER_INSTALL_PATHS" | tr ':' ' '); do
3     cd "$path" || exit 1 # Error-Code, wenn es den Pfad nicht gibt
4     composer install
5   done
6 fi
7 # das Gleiche nochmal für $NPM_INSTALL_PATHS mit npm install
```

Code 3.11: Skript zur automatisierten Installation von Bibliotheken

3.2.2 Projekt in Containern nutzen

Um ein Kundenprojekt zu nutzen, das wie in [Unterabschnitt 3.2.1](#) beschrieben mit dem Template-Repository verbunden wurde, muss zuerst eine `.env`-Datei angelegt werden. Diese Datei ist in der `.gitignore` aufgeführt und wird dadurch nicht von Git in die Versionierung aufgenommen. Das bedeutet, die in der `.env`-Datei festgelegten Konfigurationen sind nur im lokalen Repository relevant. Um allerdings schnell eine `.env`-Datei mit sinnvollen Standards zu erstellen, kann die `.env.sample`-Datei im Wurzelverzeichnis des Template-Repositories kopiert werden. Diese Datei ist im Repository dem Versionierungssystem hinzugefügt.

Umgebungsvariablen, URL und Ports des Projekts

Die wichtigsten Umgebungsvariablen, die in der `.env`-Datei zuerst angepasst werden müssen, sind `PROJECT_NAME`, `LOOPBACK_IP` und `DOMAIN`. In [Code 3.12](#) ist ein Auszug einer Beispiel-Konfiguration dieser Umgebungsvariablen zu sehen. In der `docker-compose.yml` wird `PROJECT_NAME` als Prefix für den Namen der Container verwendet. Der Projektname sollte einzigartig auf dem Gerät sein, weil sonst Kollisionen von Containernamen in Docker entstehen können.

```
1 PROJECT_NAME=my-project
2 LOOPBACK_IP=127.0.0.3
3 DOMAIN=project.local
```

Code 3.12: Beispielhafter Auszug einer `.env`-Datei

Die `LOOPBACK_IP` ist notwendig, damit Docker den `web`-Container auf dem Host-System unter der angegebenen `DOMAIN` `http[s]://project.local` zur Verfügung stellen kann. Dies funktioniert durch die Öffnung der Ports 80 und 443 des `web`-Containers (siehe [Code 3.13](#)). `WEB_PORT` und `WEB_PORT_SSL` sind Umgebungsvariablen, die auch in der `.env`-Datei konfiguriert werden können. Diese sind über die `.env.sample` standardmäßig auf `8181` und `4433` gesetzt, um möglichen Kollisionen mit anderen Webservern auf dem Host-System aus dem Weg zu gehen, weil normalerweise ein Webserver auf die Ports `80` und `443` hören. Sind die Variablen `WEB_PORT` oder `WEB_PORT_SSL` nicht gesetzt, wird auf dem Host-System der Port 80 bzw. 443 verwendet. Die Port-Konfigurationen in [Code 3.13](#) verwenden die erweiterte Syntax `<IP>:<PORT>:<PORT>` mit einer zusätzlichen lokalen Loopback-IP-Adresse, um die belegten Ports auf dem Host-System auf eine Loopback-IP zu begrenzen. Dadurch können durch unterschiedliche `LOOPBACK_IP`-Konfigurationen verschiedener Kundenprojekte mehrere Projekte gleichzeitig in Docker laufen, ohne dass es zu Portkollisionen auf dem Host kommt.

```
1 web:
2   ports:
3     - "${LOOPBACK_IP}:${WEB_PORT:-80}:80"
4     - "${LOOPBACK_IP}:${WEB_PORT_SSL:-443}:443"
```

Code 3.13: Freigabe der Ports im `web`-Container

Die Webanwendung aus dem `web`-Container ist also durch die Port-Konfiguration unter `http://127.0.0.3:8181` bzw. `https://127.0.0.3:4433` erreichbar, wenn die `LOOPBACK_IP`

wie in [Code 3.12](#) konfiguriert ist. Doch damit das Projekt auch unter der angegebenen `DOMAIN` erreichbar ist, muss die `LOOPBACK_IP` in der `hosts`-Datei auf dem Host-System der konfigurierten Domain zugeordnet werden. Dabei muss die `LOOPBACK_IP` eine unbenutzte Loopback-IP des Host-Systems sein. Das bedeutet, dass die `IP` in der `hosts`-Datei noch keiner Domain zugeordnet sein darf. Ist die `LOOPBACK_IP` in der `hosts`-Datei der `DOMAIN` zugeordnet, dann ist die Webanwendung mit der Konfiguration aus [Code 3.12](#) unter `http://project.local:8181` bzw. `https://project.local:4433` aufrufbar, weil die `hosts`-Datei auf dem Host-System die `IP` in die Domain auflöst.

Automatische `hosts`-Konfiguration

Das Skript `set-hostname.sh` unter `./_docker/web/scripts` konfiguriert beim Container-Start automatisch die angegebenen Parameter in der `hosts`-Datei des Host-Systems. Auf diese Datei wird über ein Docker Volume zugegriffen, das die Datei direkt synchronisiert (vgl. Ausschnitt `docker-compose.yml` in [Code 3.14](#)). Im Container ist die `hosts`-Datei unter `/tmp/hostsfile` verfügbar. Der Pfad zur `hosts`-Datei auf dem Host-System muss dafür zuvor in der Variable `HOSTS_FILE` in der `.env`-Datei festgelegt werden, damit die Volume-Konfiguration funktioniert. In der `.env.sample` sind dafür sinnvolle Standardwerte angelegt.

```

1 web:
2   volumes:
3     - ${HOSTS_FILE}:/tmp/hostsfile

```

Code 3.14: Docker Volume für die `hosts`-Datei

In [Code 3.15](#) ist die Logik des Bash-Skripts `set-hostname.sh` zu sehen. Es werden die Werte aus [Code 3.12](#) angenommen. Zunächst wird mittels Regex mit dem Negative Lookahead (`(?![${DOMAIN}])`) überprüft, ob die angegebene `LOOPBACK_IP` bereits mit einer anderen Domain existiert. Ist dies der Fall, werden über alle möglichen Loopback-IPs iteriert, um die nächste nicht verwendete Loopback-IP zu finden. Nach Ermittlung der nächsten unbenutzten IP wird diese zusammen mit einem Error-Hinweis auf die Konsole ausgegeben.

Danach muss auch noch überprüft werden, ob die Domain `project.local` bereits mit einer anderen IP verwendet wird. Gibt es solche Einträge, werden diese über das Programm `sed` auskommentiert. Das `sed`-Dienstprogramm ist ein „Stream Editor“, der Dateien einlesen und Änderungen über Skript-Kommandos oder Reguläre Ausdrücke machen kann [73].

Schließlich wird nach dem genauen Eintrag `127.0.0.3 project.local` in der `hosts`-Datei gesucht und falls dieser nicht existiert, wird dieser an das Ende angehängt. Das bedeutet, dass keine zuvor eingetragenen Einträge in der `hosts`-Datei gelöscht werden, sondern nur neue Einträge ans Ende der Datei angehängt oder Konflikt-Einträge auskommentiert werden.

```

1 # IP '127.0.0.3' wird schon mit anderer Domain verwendet:
2 if grep -q -P "^\s*${LOOPBACK_IP}\s+(?!${DOMAIN})" "$tmpHostsFile"; then
3     # finde nächste freie Loopback-IP (Regex) und gib diese auf der Konsole aus
4     exit 1 # Error-Code, weil ${LOOPBACK_IP} bereits verwendet.
5 fi
6
7 # Domain 'project.local' wird bereits mit anderer IP verwendet:
8 if grep -q -P
9     "^\s*(?!${LOOPBACK_IP})([0-9]{1,3}[\.]){3}[0-9]{1,3}\s+${DOMAIN}\s*$"
10    /tmp/hostsfile; then
11    # Diese Zeile in der hosts-Datei mittels sed-Befehl auskommentieren.
12 fi
13
14 # Eintrag '127.0.0.3 project.local' existiert noch nicht, füge Eintrag hinzu:
15 if ! grep -q -P "^\s*${LOOPBACK_IP}\s+${DOMAIN}\s*$" /tmp/hostsfile; then
16     printf "%s %s\r\n" "$LOOPBACK_IP" "$DOMAIN" >> /tmp/hostsfile
17 fi

```

Code 3.15: Automatische DNS-Konfiguration durch das `set-hostname.sh`-Skript

Dadurch wird also die `hosts`-Datei des Host-Systems über ein Skript innerhalb eines Containers bearbeitet. Damit das Schreiben der Datei über das Docker Volume funktioniert, braucht allerdings der Benutzer, der Docker ausführt, Schreibrechte auf die `hosts`-Datei des Host-Systems. Damit der Entwickler diese Änderung der Berechtigung lokal auf dem Computer vornehmen kann, muss entweder der genannte Benutzer Administrator-Rechte haben oder Zugang auf einen Account haben, der Administrator-Rechte hat.

Datenbank-Import

Das Herunterladen der Datenbank vom Produktiv-System, wie in [Abschnitt 3.1](#) beschrieben, sollte nicht automatisiert werden, weil sonst Zugangsdaten zu Produktiv-Servern im Repository gespeichert werden müssten, was aus Sicherheitsgründen nicht denkbar ist. Dennoch kann der Import dieser Datenbank vereinfacht werden. Dazu muss nur die zu importierende `.sql`-Datei

(z.B. `db_export.sql`) in das Verzeichnis `./_docker/db/sql` gelegt und in der `.env`-Datei der Name der `.sql`-Datei einem Datenbanknamen zugeordnet werden (vgl. [Code 3.16](#)).

```
1 DATABASE_NAME_MAIN=app_db  
2 DATABASE_FILE_MAIN=db_export.sql
```

Code 3.16: Zuordnung von `.sql`-Datei und Datenbankname in der `.env`-Datei

Das MariaDB-Image hat die Konvention, dass beim ersten Start des Containers die `.sql`-Dateien automatisch importiert und die `.sh`-Skripte automatisch ausgeführt werden, die sich im Container in `/docker-entrypoint-initdb.d` befinden [68]. Es könnten also mehrere `.sql`-Dateien auf einmal über diese Konvention importiert werden. Darüber ist es allerdings nicht möglich, für jede Datenbank eigene Zugangsdaten zu konfigurieren, weshalb ein eigenes Import-Skript notwendig ist. Das Verzeichnis `/docker-entrypoint-initdb.d` ist über ein Docker Volume mit `./_docker/db/scripts` auf dem Host verbunden. Dort befindet sich nur die Datei `import-databases.sh`. Diese wird über die genannte Konvention automatisch beim Start des Containers ausgeführt, wenn bisher noch keine Datenbank im Container vorhanden ist [68]. Das Skript importiert mehrere `.sql`-Dateien auf einmal in die gewünschte Datenbank. Die `.sql`-Dateien kommen über das Volume `./_docker/db/sql:/sql-files` in den Container, wo sie erst von dem Skript importiert werden können.

In [Code 3.17](#) ist die Logik des Bash-Skripts zu sehen. Dort wird über alle Umgebungsvariablen iteriert, die mit `DATABASE_NAME_` beginnen, danach die zugehörige `.sql`-Datei, Datenbank-Name, Benutzer und Passwort ermittelt, um anschließend die Datenbank zu erstellen und die `.sql`-Datei mit dem `mysql`-Befehl zu importieren.

```

1  for dbNameVar in "${!DATABASE_NAME_@}"; do
2      dbName=${!dbName}
3      suffix= # aktuelles Suffix basierend auf $dbName, z.B. "MAIN"
4      varName="DATABASE_FILE_${suffix}"
5      sqlFile=/sql-files/${!varName} # z.B. /sql-files/db_export.sql
6      if [ -f "$sqlFile" ]; then # wenn .sql-Datei existiert
7          mysql -u root <<-EOF
8              CREATE DATABASE IF NOT EXISTS \`$dbName\`;
9              # User DATABASE_USER_MAIN mit Passwort DATABASE_PASS_MAIN setzen
10             EOF
11             mysql -u root "$dbName" < "$sqlFile" # import db_export.sql
12         fi
13     done

```

Code 3.17: `import-databases.sh` -Skript zum Importieren mehrerer Datenbank-Dateien

Xdebug

Da Xdebug eine PHP-Erweiterung ist und zum Funktionieren einen PHP-Interpreter benötigt, wird es beim Build-Prozess automatisch im `web`-Container installiert. Dafür wird das bereits in [Abschnitt 3.2.1](#) verwendete Binary `install-php-extensions` verwendet. Hierbei ist es wichtig, darauf zu achten, welche PHP-Version dem Container zugrunde liegt, weil nicht jede Xdebug-Version mit jeder PHP-Version kompatibel ist. In [Code 3.18](#) ist dargestellt, wie dieses Problem in der `Dockerfile` des `web`-Containers gelöst wird. Mehrere `if`-`elif`-Anweisungen überprüfen nämlich auf Basis der Kompatibilitätsliste von Xdebug [74], welches die aktuellste unterstützte Xdebug-Version der angegebenen PHP-Version ist. Die erste `if`-Bedingung prüft z.B., ob die Variable `$PHP` größer ist als `7.2` und installiert dann die aktuellste Xdebug3-Version. Um eine spezifische Xdebug-Version zu installieren, nutzt `install-php-extension` die „Caret“-Syntax mit vorangestelltem `~` [70].

```

1 RUN if [ $(printf '%s\n' 7.2 "$PHP" | sort -V | head -n1) = "7.2" ]; then \
2     install-php-extensions xdebug-~3; \
3     # ...
4     elif [ $(printf '%s\n' 5.4 "$PHP" | sort -V | head -n1) = "5.4" ]; then \
5         install-php-extensions xdebug-~2.4; \
6     else \
7         echo "[ERROR] while trying to install xdebug: invalid PHP version"; \
8     fi

```

Code 3.18: Installation von Xdebug abhängig von der genutzten PHP-Version

Xdebug ist standardmäßig über die [PHP](#)-Einstellungen aktiviert. Das passiert durch die Datei `xdebug.ini` im Unterordner `./_docker/web/additional-inis`, der über ein Docker Volume in den Container gelangt, wie bereits in [Abschnitt 3.2.1](#) beschrieben. Diese Datei enthält die Einstellungen für die Xdebug Version 3 und 2, sodass Xdebug aktiviert wird, unabhängig davon welche Version installiert ist. In [Code 3.19](#) sind die [PHP](#)-Einstellungen der `xdebug.ini` zu sehen, die beim Start des Containers von [PHP](#) eingelesen werden. Die Option `xdebug.remote_host` bzw. `xdebug.client_host` legt die IP-Adresse oder den Hostnamen fest, zu dem Xdebug eine Debugging-Verbindung herzustellen versucht. Sie sollte also die Adresse des Rechners sein, auf dem die [IDE](#) läuft [75]. Da allerdings Xdebug in einem Container mit einem Docker Bridge-Netzwerk läuft, muss hier die IP-Adresse des Host-Systems aus Sicht des Containers angegeben werden. Da diese nicht immer dieselbe ist, bietet Docker den speziellen [DNS](#)-Hostnamen `host.docker.internal` an, der die dynamische IP-Adresse auflöst.

```

1 ;für Xdebug 2:
2 xdebug.remote_enable=1 ;aktiviert Xdebug Step Debugging
3 xdebug.remote_host=host.docker.internal
4 ;für Xdebug 3:
5 xdebug.mode=debug      ;aktiviert Xdebug Step Debugging
6 xdebug.client_host=host.docker.internal

```

Code 3.19: `xdebug.ini` zur automatischen Aktivierung von Xdebug

Um Xdebug mit der [IDE](#) PHPStorm zu verwenden, müssen, wie in [Unterabschnitt 2.9.1](#) beschrieben, nur noch ein Breakpoint im Code gesetzt, Debug-Verbindungen eingeschaltet und die Browser-Erweiterung aktiviert werden (vgl. [Abbildung 3.3](#)). Danach stoppt der Debugger an der angegebenen Stelle und das Debugging der [PHP](#)-Anwendung kann gestartet werden.

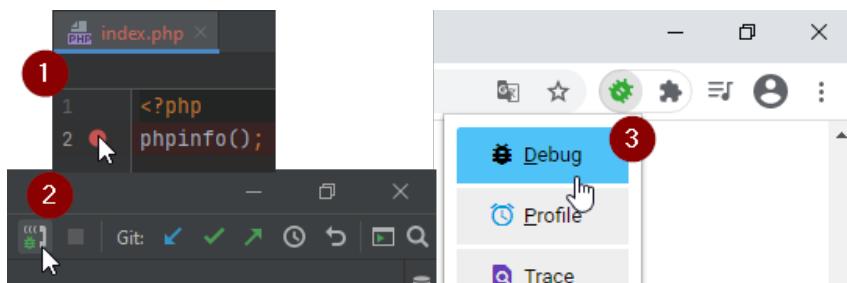


Abbildung 3.3: Debugging-Schritte einer PHP-Webanwendung mit dem Template-Repository

Es ist auch möglich [PHP](#)-Skripte oder Webanwendungen von einem entfernten Applikations-Server (z.B. der Produktiv-Server) lokal mit PHPStorm und Xdebug im Docker-Container zu debuggen. Wie das genau funktioniert, ist im [Anhang A](#) aufgeführt.

Erstellung eines Zertifikats für HTTPS

Ist beim Start der Container in der `.env`-Datei die Umgebungsvariable `HTTPS` auf `true` gesetzt, erzeugt das Skript `generate-certs.sh` automatisch ein valides Zertifikat für die festgelegte `DOMAIN` mit dem Tool `mkcert` (siehe [Abschnitt 2.7](#)). Hierfür muss über die `Dockerfile` `mkcert` im `web`-Container installiert werden. In [Code 3.20](#) ist zu erkennen, dass mit dem Programm `curl` die kompilierte Binary-Datei von GitHub heruntergeladen wird und in `/bin/mkcert` mit Berechtigungen zum Ausführen installiert wird.

```
1 RUN apt update && apt install -y curl \
2   && curl -Lo /bin/mkcert https://github.com/FiloSottile/mkcert/<binary> \
3   && chmod +x /bin/mkcert
```

Code 3.20: Installation von `mkcert` über die `Dockerfile` des `web`-Containers

Da das Skript `generate-certs.sh` innerhalb des Containers ausgeführt wird, braucht die `mkcert`-Installation im Container eine Möglichkeit mit der [CA](#) und dadurch mit dem Certificate Trust Store auf dem Host-System kommunizieren zu können. Das ist notwendig, damit die generierten Zertifikate valide sind, wenn sie von einem Browser auf dem Host-System über die Domain aufgerufen werden. Hierzu muss auf dem Host-System manuell einmalig `mkcert` installiert und `mkcert -install` ausgeführt werden, damit eine neue lokale [CA](#) erstellt wird. Der Pfad zu den Root-Zertifikaten der lokalen [CA](#) ist danach in der Umgebungsvariable `CAROOT` zu finden. Wie in [Code 3.21](#) zu sehen, wird diese Variable in der `docker-compose.yml` als Pfad für ein Docker Volume verwendet. Zusätzlich wird dort die Umgebungsvariable `CAROOT`

innerhalb des Containers auf den Ziel-Pfad des Volumes gesetzt. Auf diese Weise verwendet die `mkcert`-Installation innerhalb des Containers die **CA** des Host-Systems.

```

1 web:
2   environment:
3     CAROOT: /tmp/mkcert-caroot
4   volumes:
5     - ${CAROOT}:/tmp/mkcert-caroot

```

Code 3.21: Zuordnung des CA-Roots des Hosts in den Container über ein Docker Volume

Das Skript `generate-certs.sh` ist in [Code 3.22](#) zu sehen. Die Schlüssel-Dateien werden im Ordner `./_docker/web/certs` abgelegt. Wie in [Abschnitt 2.6](#) beschrieben, sind diese Pfade zu den Schlüssel-Dateien in der vHost-Datei des Webservers bei `SSLCertificateFile` und `SSLCertificateKeyFile` angegeben. Das Zertifikat wird für die Domain ausgestellt, die in der Umgebungsvariablen `DOMAIN` angegeben wurde.

```

1 certs=./_docker/web/certs
2 mkcert -install
3 mkcert -cert-file "$certs/pub.pem" -key-file "$certs/priv.pem" "$DOMAIN"

```

Code 3.22: Skript zur Generierung von lokal validen SSL-Zertifikaten

Das Skript `generate-certs.sh` könnte auch in einem separaten Container mit installiertem `mkcert` ausgeführt werden, da es für die Generierung des Zertifikats keinerlei Abhängigkeiten zu anderen Software-Komponenten im `web`-Container hat. Das wäre dann ein Container, der mit `docker-compose up` gestartet wird und nach Durchlaufen des Skripts und Generierung des Zertifikats wieder heruntergefahren wird. Das kann bei unerfahrenen Entwicklern zu Verwirrungen führen und so die Akzeptanz des Template-Repositorys senken. Deshalb ist es von Vorteil, wenn das Skript in einem bereits vorhandenen Container im Hintergrund ausgeführt wird.

Abfangen ausgehender E-Mails

Wie bereits in [Abschnitt 2.11](#) angedeutet, kann MailHog dazu genutzt werden, E-Mail-Verkehr zu Entwicklungszwecken aus einem Container zu empfangen. Die Idee ist, dass sämtlicher ausgehender **SMTP**-Verkehr innerhalb eines Containers abgefangen und danach zu MailHog weitergeleitet wird. Das ist mit dem Firewall-Programm `iptables` umsetzbar. Das Skript

`catch-mail.sh`, das mit dem Start des `web`-Containers ausgeführt wird, setzt dabei die nötigen Firewall-Regeln und startet danach MailHog im Hintergrund auf Port `1025` (vgl. [Code 3.23](#)). Dort wird über die gängigen **SMTP**-Ports `25`, `465` und `587` iteriert und für jede dieser Ports eine Firewall-Regel angelegt, welche den Verkehr, der auf diesen Ports ankommt, an MailHog weiterleitet. Somit verlassen versehentlich oder bewusst abgesendete E-Mails während der Entwicklung an einem Projekt niemals den `web`-Container, können aber trotzdem über die Weboberfläche von MailHog unter `http://<DOMAIN>:8025` eingesehen werden.

Das Skript `catch-mail.sh` wird nur ausgeführt, wenn die Umgebungsvariable `CATCH_MAIL` in der `.env`-Datei auf `true` gesetzt ist, was in der `startup.sh` realisiert ist.

```

1 portsToReroute=("25" "465" "587")
2 for port in "${portsToReroute[@]}"; do
3     iptables -t nat -A OUTPUT -p tcp --dport "$port" -j DNAT --to-destination
4         ↳ 127.0.0.1:1025
5 done
MailHog &>/dev/null & # MailHog im Hintergrund starten ohne Output

```

Code 3.23: `iptables`-Konfiguration zur Weiterleitung von E-Mail-Verkehr

Um dem Prinzip der Microservices gerecht zu werden, wäre es eigentlich sinnvoll, MailHog in einem separaten Container zu installieren und mit `iptables` den E-Mail-Verkehr des `web`-Containers zum `mailhog`-Container über den Hostnamen auf den Port `1025` weiterzuleiten. Allerdings hat `iptables` nicht die Möglichkeit Hostnamen in IP-Adressen aufzulösen, sondern kann nur Firewall-Regeln verarbeiten, die IP-Adressen verwenden. Aus diesem Grund wird MailHog über die `Dockerfile` in den `web`-Container installiert und der E-Mail-Verkehr auf die Loopback-IP `127.0.0.1` mit dem Port `1025` geleitet, wo MailHog läuft.

Container-Start und Container-Zugang

Wurde ein Kundenprojekt mit dem Template-Repository „dockerisiert“, können mit dem Befehl `docker-compose up` (siehe auch [Abschnitt 2.2](#)) alle beschriebenen Mechanismen zur Automatisierung ausgelöst und das Projekt gestartet werden. Wie in [Abschnitt 2.2](#) beschrieben, wird über das `depends_on`-Schlüsselwort in der `docker-compose.yml` die Start-Reihenfolge der Container festgelegt. Durch diese Konfiguration startet im Template-Repository zuerst der `db`-Container, weil der Entwickler sonst das Frontend der Anwendung öffnen könnte, obwohl

die Datenbank noch nicht importiert wurde. Das verhindert Verwirrung beim Entwickler und erhöht so die Developer Experience.

Um Fehler in einem Container zu suchen oder generell auf die Shell der Container `web` oder `db` zuzugreifen, kann der Befehl `docker-compose exec <container_name> /bin/bash` genutzt werden. Dort kann dann z.B. ein Update der PHP- oder JavaScript-Abhängigkeiten mit dem Befehl `composer update` bzw. `npm update` durchgeführt werden.

3.3 Dokumentation

Da das Template-Repository einen großen Funktionsumfang bietet und in der Benutzung nicht zwangsläufig intuitiv ist, ist es sinnvoll eine Dokumentation zu führen. Damit soll auch die Developer Experience gesteigert werden. Die Dokumentation ist im [Anhang A](#) in der englischen Sprache zu finden. Sie befindet sich in der Datei `README.md` im Wurzelverzeichnis des Repositorys und sollte, wie in [Unterabschnitt 3.2.1](#) beschrieben, bei jeder Dockerisierung in das Kundenprojekt kopiert werden. Diese Datei bietet sich an, da viele IDEs und Git-Server die Konvention nutzen, die Datei `README.md` oder ähnliche Dateinamen als Standard für Dokumentations-Dateien anzuzeigen. Die Dateiendung `md` steht für „Markdown“, was eine einfache Markup-Sprache für strukturierten Text ist [76].

Die Dokumentation ist in zwei Teile eingeteilt, die sich jeweils an unterschiedliche Benutzer des Template-Repositorys richten. Der erste Teil „Configure and run the dockerized application“ ist für Entwickler gedacht, die ein Projekt starten wollen, das bereits mit dem Template-Repository dockerisiert wurde. Dieser Teil der Dokumentation ist auch in [Unterabschnitt 3.2.1](#) enthalten. Der Inhalt des zweiten Teils „Dockerize the application“ ist für Entwickler, die die Abhängigkeiten eines Kundenprojekts mit dem Template-Repository in Docker-Container verpacken wollen und ist u.a. in [Unterabschnitt 3.2.2](#) zu finden.

Der erste Teil der Dokumentation ist so geschrieben, dass das Template-Repository auch von Entwicklern benutzt werden kann, die noch nie mit Docker gearbeitet haben oder nicht wissen, was es ist und wie es funktioniert. Das soll im Entwickler-Team die Akzeptanz des Template-Repositorys erhöhen. Der zweite Teil, also die Dokumentation der Dockerisierung kann zwar auch ohne Vorkenntnisse genutzt werden, allerdings sind Grundkenntnisse über Docker und Docker-Compose von Vorteil, wenn dies genutzt wird.

3.4 Analyse der Performance

In der folgenden Performance-Analyse werden die Ladezeiten der Startseite nach dem Login in einem Kundenprojekt verglichen. Die Basis des Kundenprojekts ist dabei bei jeder Messung die gleiche:

- Die installierte `PHP`-Version ist 7.4.
- Die Anwendung wird über eine lokal aufgelöste Domain von dem Webserver Apache in der Version 2.4 über `HTTP` bereitgestellt.
- Als `DBMS` wird MariaDB verwendet.
- Die verwendete DB ist ein Export der Produktiv-Umgebung und ca. 7,6 `MB`³ groß.
- Das Projekt basiert auf dem `PHP`-Framework „`CakePHP`“ in der Version 4 [77].
- Aufgerufen wurde die Webanwendung mit dem Chrome-Browser mit deaktiviertem Browser-Cache, sodass jede Anfrage vom Server neu angefragt wird.

3.4.1 Docker Hyper-V- oder WSL2-Backend

Im Juni 2019 hat Docker Inc. angekündigt, das Hyper-V-Backend von Docker mit einer `WSL2`-Integration zu ersetzen [78]. Dort ist die Rede von einer „*nahezu identischen I/O-Performance wie auf einem nativen Linux-Rechner*“ (Zitat übersetzt aus dem Englischen mit DeepL [79]). Außerdem soll die Build-Zeit eines Containers mit `WSL2` weitaus performanter sein im Vergleich zum Hyper-V-Backend.

Doch wie verhalten sich die Ladezeiten eines realen Kundenprojekts im Hyper-V-Backend verglichen mit einem `WSL2`-Backend? Ein Entwickler kann nämlich bei der Installation von Docker Desktop wählen (siehe [Unterabschnitt 2.1.7](#)), ob er Hyper-V oder `WSL2` als Backend verwenden will. Die [Tabelle 3.1](#) zeigt einen Ausschnitt der 20 aufgezeichneten Ladezeiten der Startseite eines Kundenprojekts einmal mit Docker Hyper-V-Backend und einmal mit `WSL2`-Backend. Während der Durchschnitt der Ladezeiten bei Hyper-V bei 506 Millisekunden liegt, hat das Laden der Startseite mit `WSL2`-Backend im Schnitt 17% weniger Zeit benötigt, nämlich

³ Megabyte

Docker-Backend	Hyper-V	WSL2
Ladezeit in ms	721	588
	513	423
	471	420
	483	404
	454	416
	:	:
Durchschnitt in ms	506	432

Tabelle 3.1: Vergleich der Ladezeiten bei Docker Hyper-V- und WSL2-Backend

432 Millisekunden. Dieser Unterschied von 74 Millisekunden ist allerdings bei der Benutzung kaum spürbar.

Bei der Messung der Daten lagen bei Hyper-V die Dateien des Kundenprojekts im Windows-Dateisystem und bei WSL2 im Dateisystem von WSL. Dieses Vorgehen ist nämlich aus Gründen der Performance bei Verwendung von Docker Volumes von Docker empfohlen [25].

Auch wenn der Performance-Unterschied der Webseite zwar messbar, aber kaum spürbar ist, empfiehlt es sich trotzdem, Docker unter Windows so zu konfigurieren, dass es WSL2 als Backend verwendet und das Kundenprojekt im WSL-Dateisystem liegt. Der Hauptgrund dafür ist, dass Docker damit die Ressourcen zum Betrieb der Docker-Engine (RAM und CPU-Nutzung) viel besser verteilen kann und so schnellere Build- und Start-Zeiten von Containern und der Docker-Engine selbst hat. [78]

3.4.2 Auswirkung von `opcache` auf die Performance

Wie bereits in Abschnitt 2.9 beschrieben, wird die PHP-Erweiterung `opcache` zur Verbesserung der Performance komplexer Webanwendungen verwendet. Wie verhalten sich also die Ladezeiten mit und ohne Verwendung von `opcache`?

In Abbildung 3.4 ist ein Diagramm zu den Ladezeiten der 20 Anfragen dargestellt, jeweils für ein Docker-WSL2-Setup mit und ohne installiertem `opcache`. Generell ist dort zu beobachten, dass die Ladezeiten mit installiertem `opcache` fast immer geringer sind, als ohne die Installation der PHP-Erweiterung. Zudem ist zu erkennen, dass nach dem ersten Laden der Webseite mit `opcache`, wie in Abschnitt 2.9 beschrieben, der Bytecode Cache auf dem Server

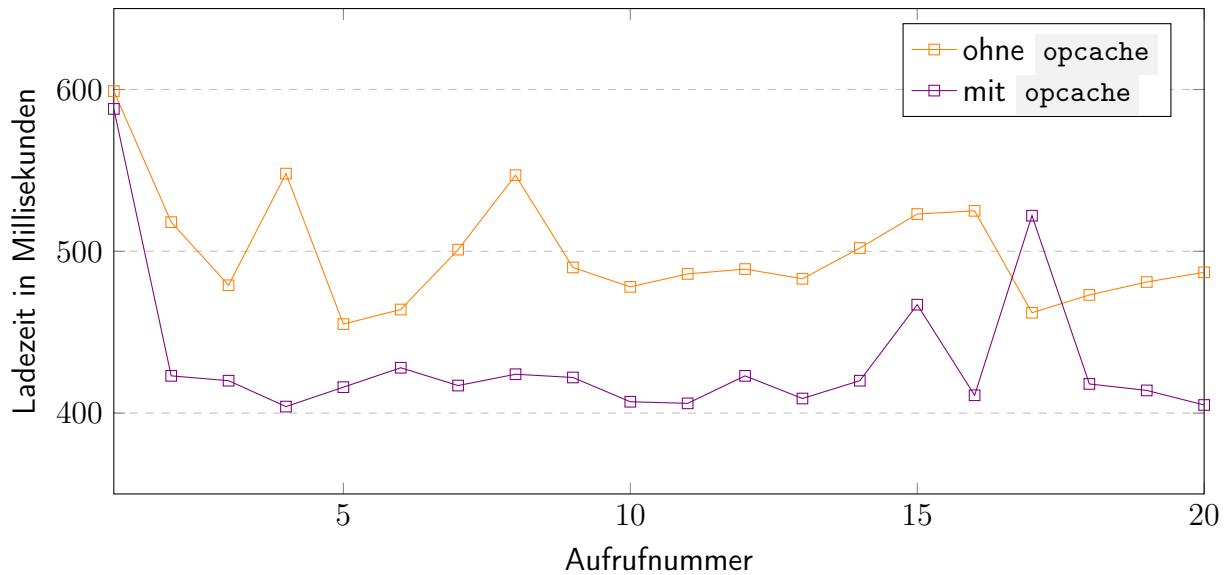


Abbildung 3.4: Vergleich der Ladezeit mit und ohne die PHP-Erweiterung `opcache`

geschrieben wird und alle nachfolgenden Anfragen dann eine weitaus geringere Ladezeit haben. Hier zeigt der Cache seine Wirkung. Aus diesem Grund wird bei dem Template-Repository standardmäßig `opcache` im `web`-Container mit installiert.

3.4.3 Vergleich XAMPP-Umgebung und Docker-Umgebung

In folgendem Abschnitt werden die Ladezeiten einer lokalen Web-Entwicklungsumgebung mit XAMPP und eine Web-Entwicklungsumgebung mit dem Template-Repository mit Docker verglichen. XAMPP ist eine kostenlose PHP-Umgebung mit Apache und MariaDB, die unter Windows, Linux und MacOS installiert werden kann [80]. In Abbildung 3.5 ist ein Balkendiagramm mit den Ladezeiten der ersten 20 Aufrufe der beiden Umgebungen zu sehen. Dabei wurde in beiden Umgebungen die PHP-Erweiterung `opcache` für das serverseitige Caching verwendet.

Beim ersten Aufruf der Seite ist deutlich zu sehen, dass die Ladezeit bei der XAMPP-Umgebung wesentlich höher ist, als die der Docker-Umgebung. In der XAMPP-Umgebung hat es fast 10 Sekunden gedauert, bis die Seite komplett geladen wurde. Fast alle Aufrufe in der XAMPP-Umgebung, die danach ausgeführt wurden, hatten eine geringere Ladezeit als die Aufrufe in der Docker-Umgebung. Im Durchschnitt war die XAMPP-Umgebung in den letzten 19 Aufrufen ca. 45 Millisekunden bzw. 12% schneller als die Docker-Umgebung.

Diese leicht erhöhten Ladezeiten in der Docker-Umgebung sind wahrscheinlich auf die zusätzli-

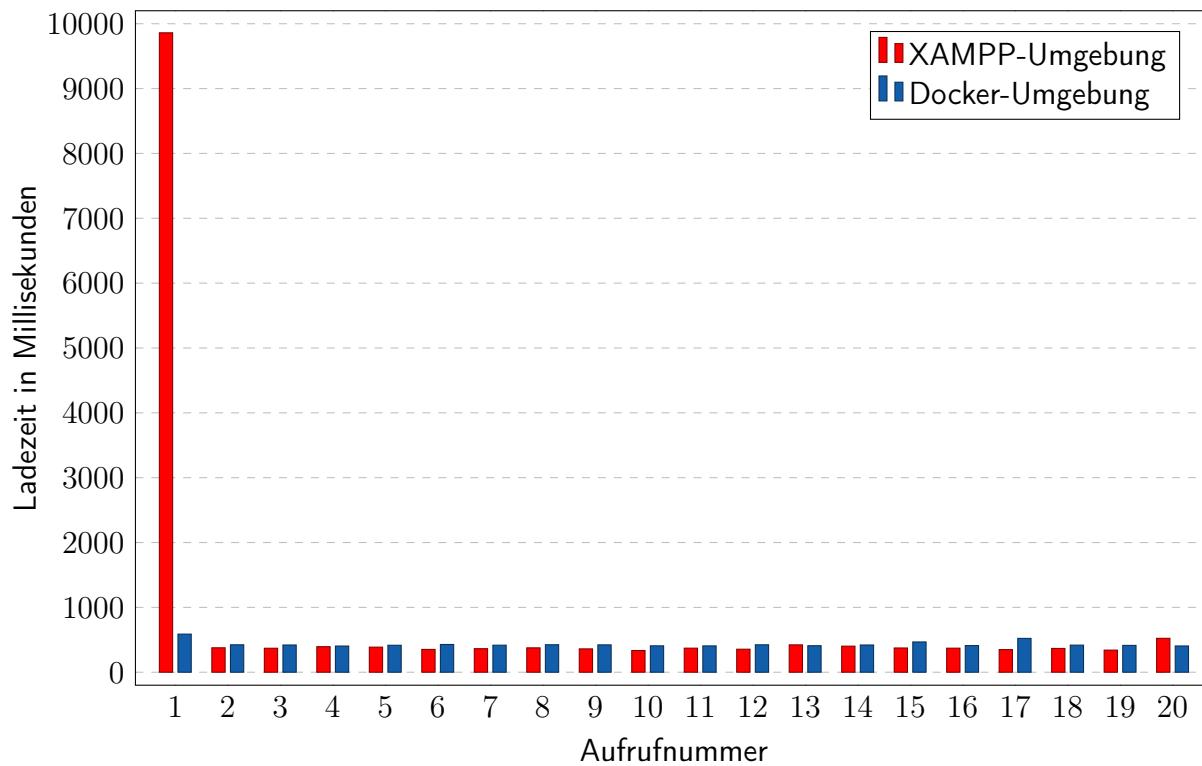


Abbildung 3.5: Vergleich der Ladezeiten mit XAMPP-Umgebung und mit Docker-Umgebung

chen Abstraktionsebenen [WSL](#) und Docker-Container zurückzuführen. Dennoch bringen diese Abstraktionsebenen zusätzliche Vorteile, welche die XAMPP-Umgebung nicht bieten kann, z.B. die Kapselung der Abhängigkeiten in Container und die daraus resultierende Reproduzierbarkeit von Projekten auf mehreren Computern. Außerdem kann das Template-Repository durch die Automationsmechanismen eine höhere Developer Experience bieten. Dementsprechend ist der geringe Performance-Nachteil zu vernachlässigen.

Unabhängig von der Performance der beiden Lösungen hat die Docker-Umgebung durch die Kapselung der Komponenten noch einen Vorteil. Im Vergleich zur alten Lösung, kann es bei projektspezifischer Software nicht mehr zu Software- bzw. Versionskonflikten auf dem Computer eines Entwicklers kommen. Das kann mit der alten Lösung z.B. bei Composer-Paketen vorkommen, die nicht mit der lokal installierten Composer-Version kompatibel sind.

3.5 Kosten-Analyse

Je geringer die Zeit zur lokalen Einrichtung eines Kundenprojekts ist, desto geringer sind die Kosten, die dafür entstehen und es kann mehr Zeit in die Entwicklung der Funktionen der Webanwendung gesteckt werden. Dementsprechend gilt es, die Zeit der Projekteinrichtung

zu verringern, um Kosten zu sparen. Dies soll mit dem entwickelten Template-Repository geschehen. In folgender Kosten-Analyse sollen die jährlichen Kosten der Projekteinrichtung mit dem Template-Repository mit den Kosten der Projekteinrichtung ohne Template-Repository, also ohne Docker-Container, verglichen werden. Dabei wurden auf Basis von Schätzungen im Gespräch mit Kollegen folgende Annahmen getroffen:

- 40 neue Projekte pro Jahr (*AnzahlProjekte*) mit je 3 Projekt-Einrichtungen (*AnzahlInstallationen*)
- Die Entwicklungskosten des Docker-Templates belaufen sich auf einmalige 973,69€ (errechnet durch die benötigte Zeit und den Azubi-Stundensatz).
- Die einmalige Dockerisierung eines Projekts dauert im Durchschnitt 0,75 Stunden.
- Die Zeit für die lokale Einrichtung eines dockerisierten Projekts beträgt 0,2 Stunden.
- Die Zeit, ein Projekt ohne Template-Repository lokal einzurichten, beträgt 1 Stunde.

Generell wird zur Kostenberechnung zwischen dem Gehalt eines Azubis und dem eines Angestellten unterschieden. Zur Berechnung der Entwicklungs- und Einrichtungskosten wurde bei der einen Hälfte der Projekteinrichtungen das Azubi-Gehalt und bei der anderen Hälfte das Gehalt eines Angestellten mit der jeweils geschätzten Dauer in Stunden multipliziert. Diese Hälfte der Dauer $T_{\frac{1}{2}}$ wird wie folgt berechnet:

$$T_{\frac{1}{2}} = \frac{\text{AnzahlProjekte} * \text{AnzahlInstallationen}}{2} * \text{Dauer} \quad (3.1)$$

Damit können danach die Einrichtungskosten K für ein Projekt berechnet werden:

$$K = \frac{T_{\frac{1}{2}} * \text{GehaltAzubi} + T_{\frac{1}{2}} * \text{GehaltAngestellter}}{\text{AnzahlProjekte}} \quad (3.2)$$

In Tabelle 3.2 ist die Berechnung der Gesamtkosten des ersten Jahres für das Docker-Setup und für das lokale Setup zu sehen. Dabei wurden obige Formeln für die Berechnung von K_{d1} und K_{l1} verwendet⁴.

⁴ Die Werte des Stundengehalts der Azubis (*GehaltAzubi*) und der Angestellten (*GehaltAngestellter*) sind vertrauliche Informationen und deshalb nicht angegeben.

	Docker-Setup	lokales Setup
einmalige Entwicklungskosten	973, 69€	
Dockerisierungskosten 1 Projekt K_{d1}	9, 74€	
Dockerisierungskosten alle Projekte	* 40 = 389, 40€	
lokale Einrichtungskosten 1 Projekt K_{l1}	5, 19€	38, 94€
lokale Einrichtungskosten alle Projekte	* 40 = 207, 68€	* 40 = 1557, 60€
Gesamtkosten 1. Jahr	1570, 77€	1557, 60€

Tabelle 3.2: Kosten-Analyse zur Einrichtung von Kundenprojekten mit Docker-Setup und lokalem Setup

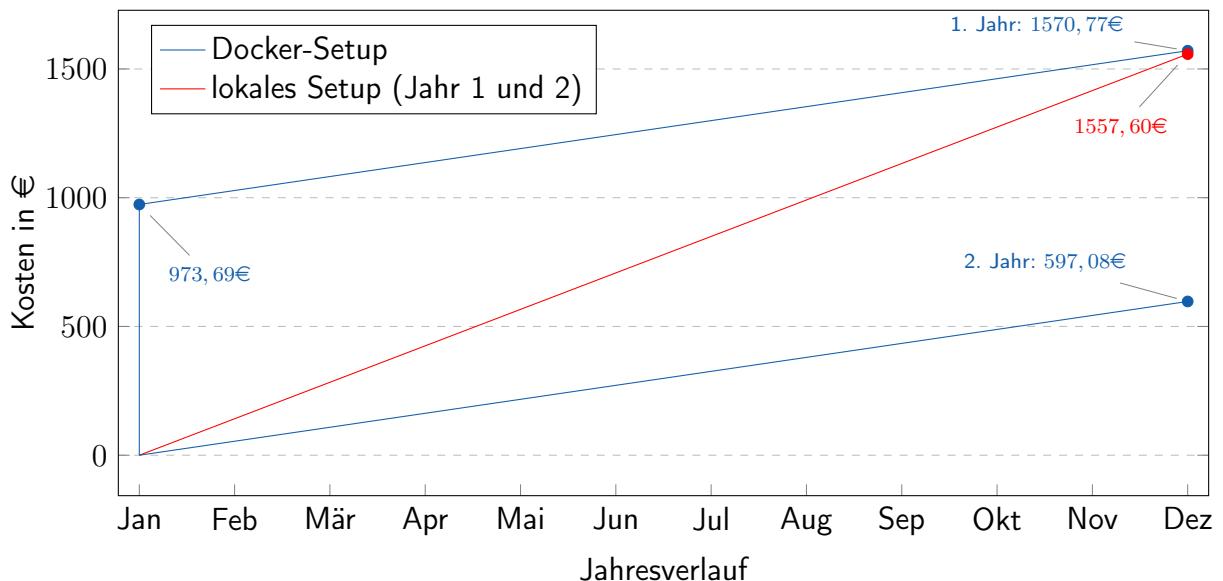


Abbildung 3.6: Kosten-Vergleich eines Docker-Setups und eines lokalen Setups in den ersten zwei Jahren

Dort ist zu sehen, dass im ersten Jahr beide Vorgehensweisen fast die gleichen Kosten verursachen. Im zweiten Jahr entfallen allerdings die initialen Entwicklungskosten von 973, 69€, weshalb ab dem zweiten Jahr die Docker-Lösung die günstigere Vorgehensweise zur Einrichtung der Kundenprojekte ist. Dann kostet die Docker-Lösung nämlich nur noch 597, 08€.

Verteilt man die Einrichtungs-Kosten der Projekte gleichmäßig auf ein Jahr, dann ergeben sich die Kurven, wie sie in Abbildung 3.6 zu sehen sind. Dort ist auch nochmals zu erkennen, dass die Steigung der Kurven, also die Einrichtungskosten pro Projekt beim Docker-Setup viel geringer sind als beim lokalen Setup. Das zahlt sich dann bereits im zweiten Jahr aus, wo die Gesamtkosten fast um ein Drittel geringer sind als im Vorjahr, weil die Entwicklungskosten des Template-Repositorys wegfallen.

Die Verwendung des Template-Repositorys ist also auf längere Sicht die günstigere Variante.

Zu diesem Vorteil kommt auch noch die bereits genannte bessere Developer Experience hinzu. Diese lässt sich nicht in die Kosten verrechnen, ist aber dennoch wichtig für die Einführung von Docker in den Entwicklungsprozess.

4 Schlussbemerkungen

Das fertige Template-Repository nutzt nun also verschiedene Konfigurationsdateien, Bash-Skripte und Docker Compose mit der Docker-Engine, um den Prozess der Projekt-Einrichtung zu automatisieren und mit Hilfe von Container-basierter Virtualisierung die Abhängigkeiten des Kundenprojekts zu kapseln. Mit den in [Abschnitt 3.2](#) beschriebenen Funktionen sind alle Anforderungen erfüllt, die in [Abschnitt 3.1](#) erarbeitet wurden.

Die wichtigsten Gründe und Vorteile für die Nutzung dieses Template-Repositorys, wie in [Abschnitt 1.4](#) und [Kapitel 3](#) genauer erörtert, sind in folgender Auflistung nochmals kurz zusammengefasst:

- Alle projektspezifischen Abhängigkeiten sind in Git enthalten, was die Reproduzierbarkeit des Projekts sicherstellt ([Unterabschnitt 3.2.1](#)).
- Die Kapselung des Projekts in Container verhindert Software-Konflikte über mehrere lokale Projekte hinweg ([Unterabschnitt 3.4.3](#)).
- Die Nutzung des Template-Repositorys spart auf lange Sicht Zeit und Geld und verbessert die Developer Experience ([Abschnitt 3.5](#)).
- Die Performance der Anwendung bleibt trotz Container-Abstraktion weitestgehend erhalten ([Abschnitt 3.4](#)).

Trotz aller Vorteile sollte die Lösung des Problems auch kritisch betrachtet werden, weil es potentiell weitere Verbesserungen oder Probleme bei der Verwendung des Template-Repository geben könnte. Dies ist in folgendem Abschnitt erläutert.

4.1 Kritische Reflexion

Der größte Kritikpunkt des Template-Repositorys ist wahrscheinlich die Benutzerfreundlichkeit, ein vorhandenes Projekt zu dockerisieren. Die komplexen und umfangreichen Funktionen sind zwar in der Dokumentation ([Abschnitt 3.3](#)) beschrieben, allerdings sinkt die Benutzerfreundlichkeit durch das Bearbeiten mehrerer Konfigurationsdateien (`Dockerfiles`, `docker-compose.yml`, `.env`-Datei, Datenbank- und Webserver-Konfigurationsdateien, ...).

Eine mögliche Lösung zur Verbesserung der Nutzbarkeit wäre, eine einzelne Konfigurationsdatei festzulegen, die ähnlich aufgebaut ist, wie die `.env`-Datei und dann eine „Wrapper-CLI-Anwendung“ für die Benutzung des Template-Repositorys programmiert. Diese Konsolen-Applikation sollte dann die Konfiguration der oben genannten Konfigurationsdateien auf Basis der Konfigurationsdatei automatisieren und so als Abstraktionsschicht zwischen Docker Compose und dem Benutzer dienen. Der Benutzer würde dann mit einem CLI-Befehl die Konfigurationsdatei mit sinnvollen Standardwerten generieren, diese für seine Bedürfnisse anpassen und mit einem weiteren Befehl die automatische Konfiguration und den Build- und Start-Prozess der Container starten. Dies würde zwar nicht die Dokumentation ersetzen, aber dennoch das Nutzen des Template-Repositorys auf festgelegte Befehle und die Konfigurationsdatei reduzieren. Der Entwickler müsste in diesem Fall auch gar nicht die grundsätzlichen Konzepte von Docker und Docker Compose kennen, weil alles über das Befehlszeilenprogramm abstrahiert wäre.

Ein weiterer Mangel des Template-Repositorys ist, dass es bisher nur unter Windows und nicht unter Linux und MacOS getestet wurde. Um alle Entwickler von Hochwarth IT für die Benutzung des Template-Repositorys einschließen zu können, müssen sämtliche Funktionen jeweils auf Linux und MacOS getestet und auf mögliche Fehler überprüft werden.

Damit einher geht auch die Dokumentation ([Abschnitt 3.3](#)), die bisher nur für Windows mit Docker WSL-Backend und für die IDE PHPStorm optimiert wurde. Hier müsste die Dokumentation für Linux, MacOS und gegebenenfalls für weitere IDEs angepasst werden.

4.2 Ausblick

Zu Beginn in [Abschnitt 1.1](#) ist begründet, dass das Template-Repository nicht für Deployment auf den Produktiv-Servern von Hochwarth IT ausgelegt ist, weil diese Docker nicht unterstützen. Wären zukünftig genutzte Produktiv-Server tatsächlich darauf ausgelegt, könnte das Template-Repository auch für das Deployment genutzt werden. Das hätte den Vorteil, dass sowohl die Entwicklungsumgebung als auch die Produktiv-Umgebung der Projekte sehr ähnlich wären, wodurch Fehler vermieden werden könnten.

Der Prozess des lokalen Einrichtens und Deployments würden sich ein wenig unterscheiden. Der Instanz des Docker-Containers müsste ein weiterer Webserver auf dem Server vorgeschal-

tet sein, weil sonst von außen nicht auf die Webanwendung, die über bestimmte Ports vom Container dem Host zur Verfügung gestellt wird, zugegriffen werden kann. Eine beispielhafte schematische Konfiguration eines solchen Produktiv-Servers ist in [Abbildung 4.1](#) zu sehen. Hierfür müsste in diesem Webserver ein vHost für die Domain (z.B. `example.com`) angelegt und dort ein sogenannter Reverse-Proxy mit dem Port konfiguriert werden, den der `web`-Container zur Verfügung stellt. In diesem Beispiel wäre das Port `8080`. Mit diesem Reverse-Proxy würde dann der Webserver den eingehenden Traffic von `example.com` auf die interne Adresse `localhost:8080` weiterleiten. Damit die Seite auch über [HTTPS](#) erreichbar wäre, müsste für diese Domain auch ein [SSL](#)-Zertifikat ausgestellt und hier konfiguriert werden.

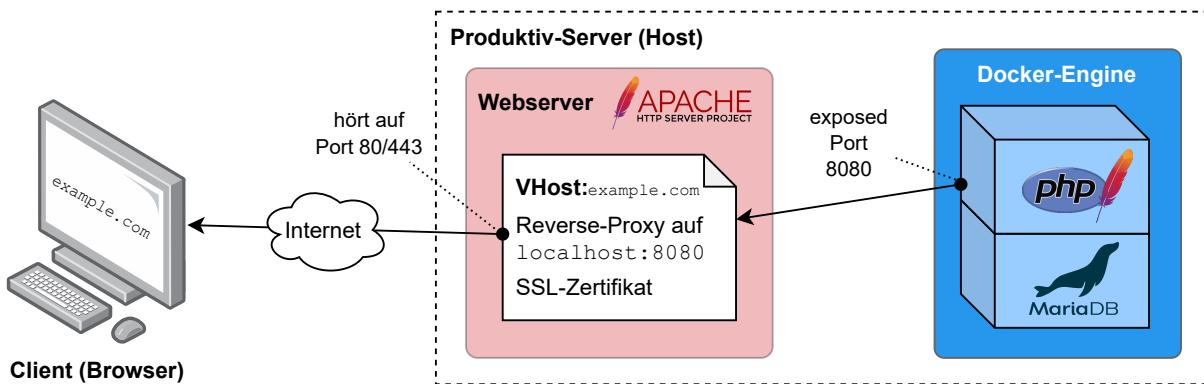


Abbildung 4.1: Deployment-Setup auf kompatiblem Produktiv-Server

Für den Start der Docker-Container wäre dann ein „Produktiv-Modus“ hilfreich, der über eine Umgebungsvariable in der `.env`-Datei festgelegt werden könnte. Dieser würde dann dafür sorgen, dass die automatische Erstellung des [SSL](#)-Zertifikats mit `mkcert` für den Docker-Container sowie die lokale Hostname-Konfiguration ausgesetzt wird.

Das Template-Repository befindet sich bezüglich oben aufgeführter Nachteile in ständiger Optimierung.

Anhänge

Anhang A: README.md

Docker Development Setup

Intro: What is this Docker setup?

This Docker development setup is mainly setup with the `docker-compose.yml` file. It creates two containers at startup (`web` and `db`).

The `web` container runs Debian (Linux) and is responsible for the web application. The current directory will be available in the `/var/www/html` directory of the web application.

The `db` container runs Alpine Linux (very small Linux). It holds the database server and is therefore responsible for managing database connection etc.

Configure and run the dockerized application:

Important Note: If you use this on Windows with WSL2 backend clone your repository inside a WSL directory (e.g. `~/projects/my-project`) for better performance.

1. Create `.env` file

copy `.env.sample` to `.env`

2. Set some important variables

Set your project name in the `.env` file like so:

```
PROJECT_NAME=my-project
```

This prevents container name collisions in the future.

Before we can configure a custom domain where your application will be available at, we need to set an **unused loopback IP** from the IP range `127.0.0.0/8` in your `.env` file.

For example:

```
LOOPBACK_IP=127.0.0.2
```

Note: If your configured `LOOPBACK_IP` is already in use, it will tell you the next free IP at container startup.

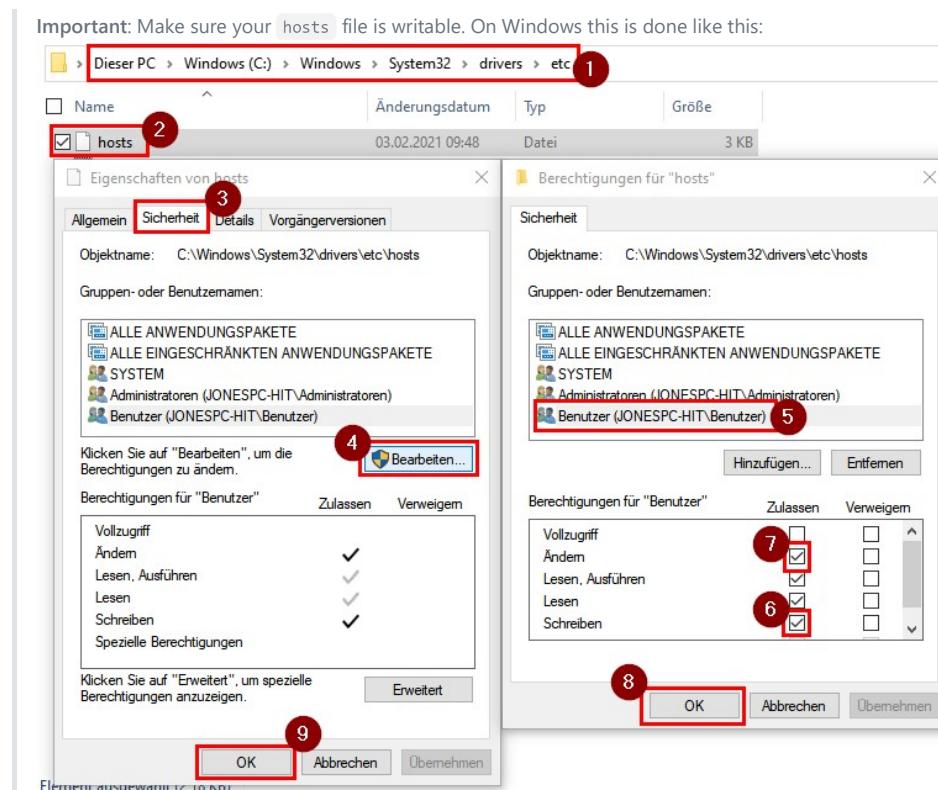
Set a custom domain where your application will be available at:

```
DOMAIN=test.local
```

Set the absolute path to your `hosts` file on your OS:

```
HOSTS_FILE=/mnt/c/Windows/System32/drivers/etc/hosts      # use this for Windows with WSL  
#HOSTS_FILE=/c/Windows/System32/drivers/etc/hosts          # use this for Windows  
#HOSTS_FILE=/etc/hosts                                     # use this for Linux  
#HOSTS_FILE=/private/etc/hosts                            # use this for MacOS
```

This is needed to automatically set your domain in your hosts file.



3. Database

Import at initial startup:

To import a database at initial docker startup move a `.sql` file to `./_docker/db/sql`

After that map this file to a database name of your choice in your `.env` file:

```
DATABASE_NAME_MAIN=test_name
DATABASE_FILE_MAIN=test.sql
```

This will import the file `./_docker/db/sql/test.sql` in the newly created database `test_name` at container startup within the `db` container.

To import multiple databases, add another `.sql` file and add more mapping variables with another suffix than `_MAIN` like above.

Important: If the container is already running, stop it, tear it down and start it again to trigger the import:

```
docker-compose down # Warning: this will delete your database inside the container
docker-compose up
```

4. Download and configure Docker Desktop

If you are on Windows or Mac download and install [Docker Desktop](#) if you haven't already.

For best performance on Windows install and configure Docker Desktop [with the WSL2 backend](#).

5. Start your containers

After configuration you can start your containers with executing the following command in the root directory of your project:

```
docker-compose up
```

Make sure to start the docker daemon first (Docker Desktop).

The first time executing this takes a few minutes.

| Tip: The Dashboard of Docker Desktop can be quite useful to manage your containers.

Port error

If you get an error like:

```
Ports are not available: listen tcp 127.55.0.1:3308: bind: Der Zugriff auf einen Socket war aufgrund der Zugriffsrechte des Socke
```

You have to change the corresponding port in your `.env` file to a port that is available on your local system. For example:

```
WEB_PORT: 8081
WEB_PORT_SSL: 4443
DB_PORT: 3309
```

6. Connect to database (client)

You can connect to the database (for example [DBeaver](#)) with the following credentials:

- host: the `DOMAIN` variable you specified in your `.env` file
- port: the `DB_PORT` variable you specified in your `.env` file
- user: `root`
- password: `root` (if not specified differently in `docker-compose.yml` with the environment variable `MYSQL_ROOT_PASSWORD` of the `db` container)
- database name: the `DATABASE_FILE_MAIN` or `DATABASE_FILE_<OTHER_SUFFIX>` you specified in your `.env` file

7. Open Application

Local

To open the application frontend open `http://<DOMAIN>:<WEB_PORT>` in your browser.

You can configure your `DOMAIN` in `.env` file. Make sure to restart the containers after changing it:

```
DOMAIN=test.local
```

https

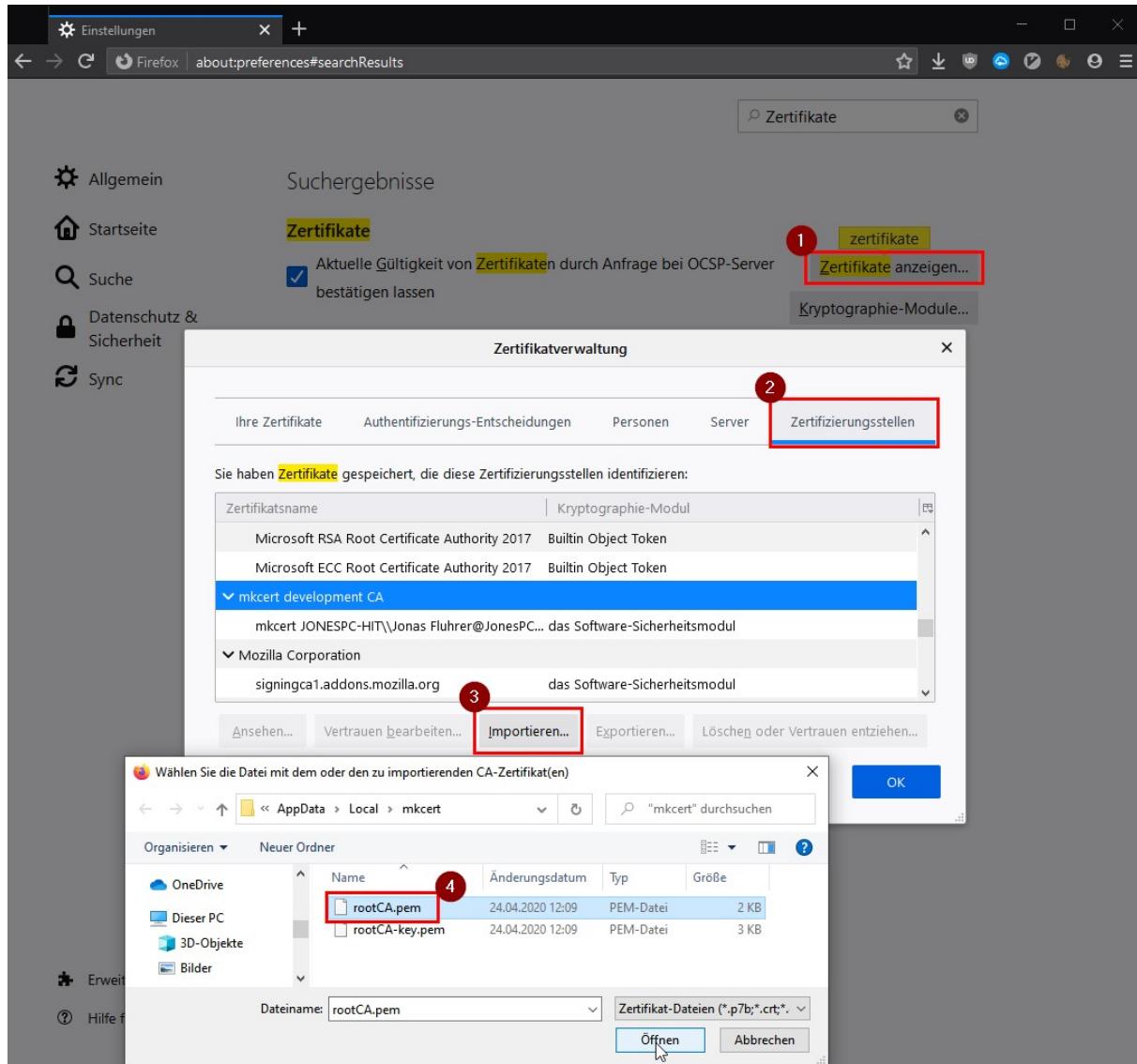
If you need the application available through https, you first need to install mkcert locally (see [GitHub mkcert for instructions](#)).

After installation run `mkcert -install` locally.

Then set `HTTPS=true` in your `.env` file and restart the container.

After that, it will automatically generate a valid ssl certificate and your application will be available at `https://<DOMAIN>:<WEB_PORT_SSL>`.

If you need to access it on Firefox, you first need to import the RootCA certificate in the certificate settings of Firefox (see screenshot). The `install-cert.cmd` tells you the location of this certificate file called `rootCA.pem`.



On the network

If you want to access your application from another device **on the same network**, you have to set up a new port mapping with the IP **your computer has on the corresponding network interface** (for example `192.168.178.54`).

Do this in a new file called `docker-compose.override.yml` in your project root:

```
services:
  web:
    ports:
      - "192.168.178.54:${WEB_PORT:-80}:80"
```

After you restarted your container, the application will be available at `http://192.168.178.54:<WEB_PORT>` on the network you are connected to.

8. Xdebug Configuration

Xdebug is **installed and enabled by default** and is ready to use.

To disable xdebug with PHP version < 7.2 change the file `./_docker/web/additional-inis/xdebug.ini` to:

```
xdebug.remote_enable=0
```

To disable xdebug with PHP version >= 7.2 change the file `./_docker/web/additional-inis/xdebug.ini` to:

```
xdebug.mode=off
```

To enable xdebug with PHP version >= 7.2 change the file `./_docker/web/additional-inis/xdebug.ini` to:

```
xdebug.mode=debug
```

After that you need to restart the container.

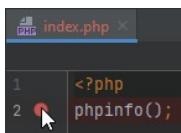
In this file you can also specify any other xdebug config you might need.

9. Xdebug usage:

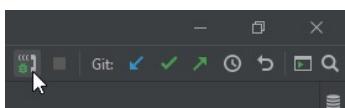
usage in browser

To use xdebug with PHPStorm do the following:

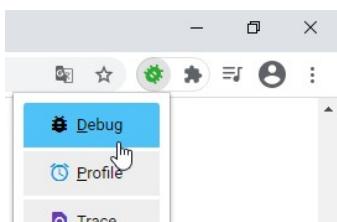
1. Set a breakpoint in your code:



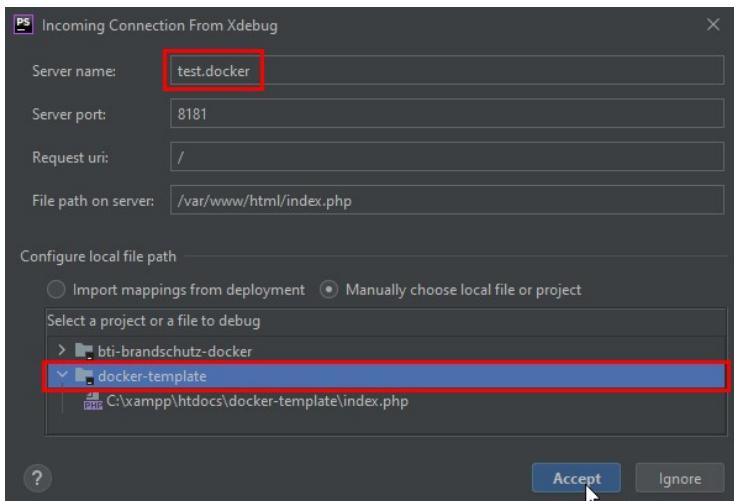
2. Listen to xdebug connections in your IDE:



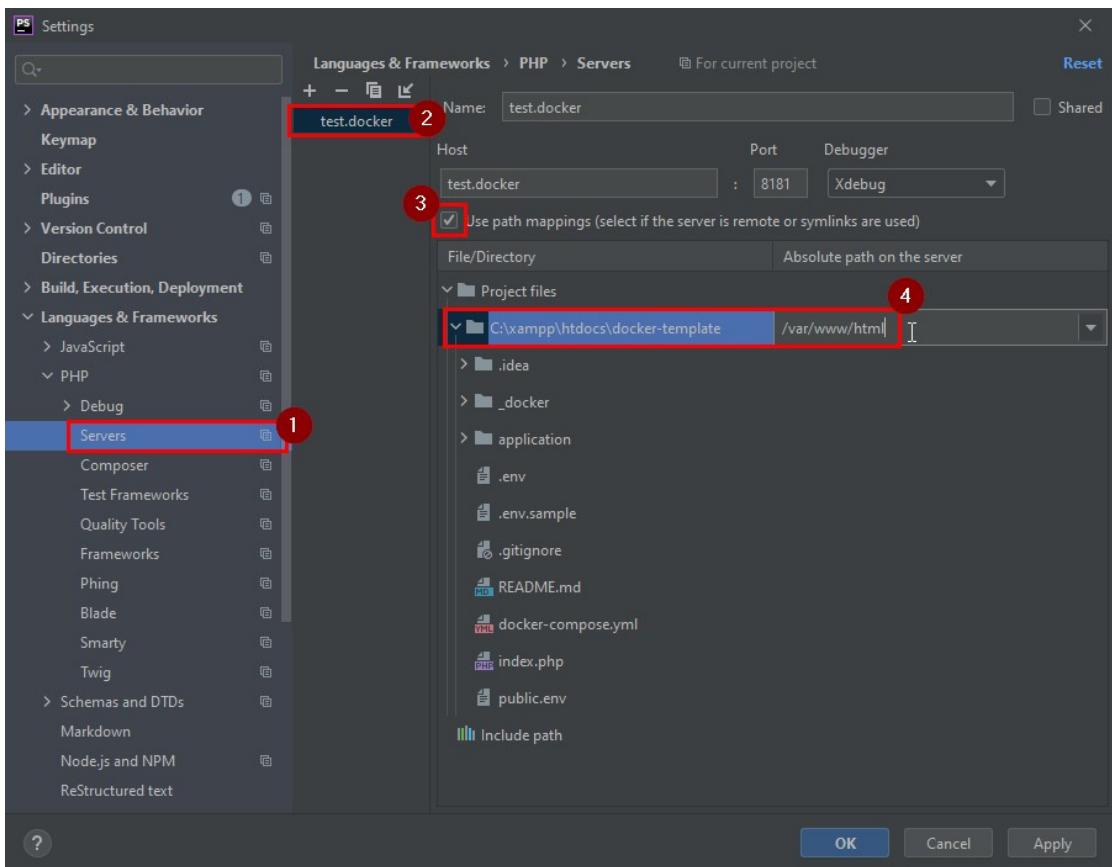
3. Turn on your browser debug extension and reload page:



4. Choose correct project and accept incoming connection:



5. Set path mappings in IDE settings: root of project should be set to /var/www/html

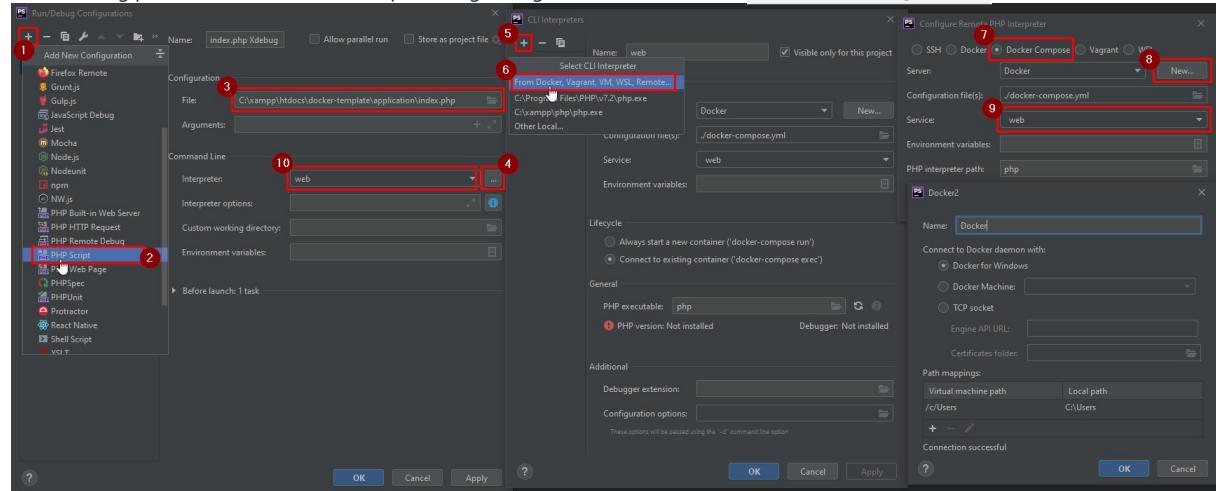


6. Reload page

If you screwed up somewhere in between, delete the server configuration in `Settings > Languages & Frameworks > PHP > Servers` and start over.

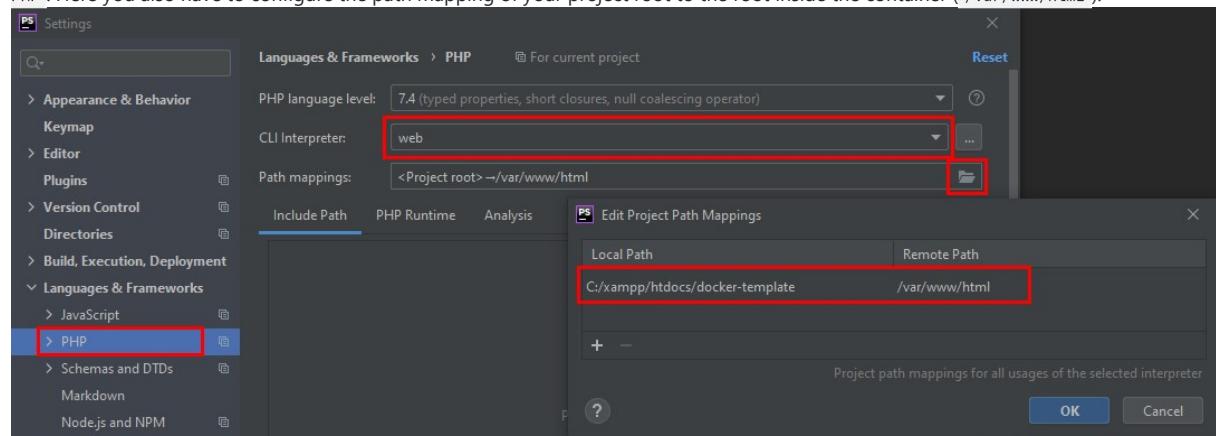
Debug a PHP script:

The following picture describes how to setup a Debug Configuration in PHPStorm in Run - Edit Configurations :



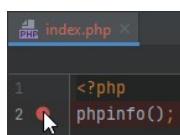
1. Click on the **+** to create a new configuration
2. Choose **PHP Script**
3. Choose your local file you want to debug
4. Click on **...** next to the CLI interpreter
5. Click on the **+** to create a new PHP CLI Interpreter
6. Choose **From Docker, Vagrant, VM, WSL, Remote...**
7. Choose **Docker Compose**
8. Click on **New...** next to **Server** if you haven't created a Docker Server Configuration yet. Confirm with **OK**.
9. Make sure to set the Interpreter to the created **web** CLI Interpreter.

After that make sure you **also** have the PHP CLI Interpreter configured to the docker container at **Settings - Languages & Frameworks - PHP**. Here you also have to configure the path mapping of your project root to the root inside the container (`/var/www/html`):

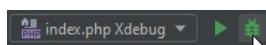


Now you can start debugging your PHP script with...

1. Setting a breakpoint in your PHP script:



2. Start running the script with the debug button:

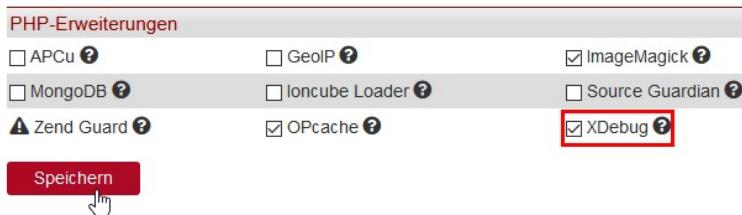


Xdebug remote connection to server:

To debug an application from a remote server in your local IDE do the following (on the example of project `bti-brandschutz-dev` available at <https://dev.bti-brandschutz.de>):

Warning: this is currently untested for server xdebug versions `>= 3.X`. You may need to adjust some more xdebug variables. In this case, this may help: https://xdebug.org/docs/upgrade_guide

1. Make sure your docker container are running
2. Enable xdebug extension on the server. Here with example from Hetzner:



3. Make sure the xdebug variables are set as follows:

```
xdebug.remote_enable=1
xdebug.remote_connect_back=0
xdebug.remote_host=localhost
xdebug.remote_port=9000      # change to port 9003 on xdebug version >= 3.X (run 'php --version' to find out)
```

These variables can also be set with an `.htaccess` file in the root directory on the server like:

```
php_flag xdebug.remote_enable On
php_flag xdebug.remote_connect_back Off
php_value xdebug.remote_host localhost
php_value xdebug.remote_port 9000      # change to port 9003 on xdebug version >= 3.X (run 'php --version' to find out)
```

4. Start an SSH tunnel from your machine to the server and enter the ssh password:

```
ssh -p <server-ssh-port> -R 9000:<DOMAIN>:9000 <server-ssh-user>@<server-ip>
```

Note 1: `DOMAIN` is the domain you configured in your `.env` file. The other parameters should be available from the provider, where the site is hosted.

Note 2: Set the xdebug `9000` to `9003` if the xdebug version of the server is `>= 3.X`. You can find out which xdebug version the server has by running `php --version` if the xdebug extension is enabled

5. Once the SSH tunnel connection is established, follow the steps from **Xdebug usage** above.

Note: At step 5 the root of your project is **NOT** `/var/www/html`. Instead, in the SSH session `cd` to the root path of the project and run `pwd` to get the full absolute path. Enter this path at step 5 as your project root.

10. Composer, npm, grunt and other commands

Composer, npm and grunt is pre-installed in the `web` container.

`composer install` and `npm install` is automatically executed at container startup if configured.

If you want to manually execute another command, it is best to execute it in the `web` container:

```
./shell.cmd # go into container on unix/linux
shell.cmd # go into container on windows

composer <any-composer-command>
npm <any-npm-command>
grunt
<any-other-command>
```

11. Catch outgoing mails

In a big application it is likely that there will be emails sent to a user at some point.

To avoid sending emails to users while developing or testing, the `web` container catches all emails sent to port `25`, `465` or `587` and sends them to a local mail server called *MailHog*.

The `web` container will catch these outgoing emails, if you enable it in your `.env` file:

```
CATCH_MAIL=true
```

You can look at the caught emails at `http://<DOMAIN>:<CATCH_MAIL_PORT>`. The default `CATCH_MAIL_PORT` is `8025`.

Dockerize the application:

Note 1:

If you make any changes to one of the following files:

- any `Dockerfile`

make sure to rebuild it:

```
docker-compose build
```

After that you can start it again with:

```
docker-compose up
```

Configuration:

1. PHP setup

To specify the **PHP version** change the `FROM` command in `./_docker/web/Dockerfile`

e.g. for PHP version 5.6:

```
ARG PHP_VERSION=5.6
FROM php:${PHP_VERSION}-apache
```

After that make sure to build this container again (see Note 1 above)

PHP Extensions:

To install and enable **PHP extensions** add them to `./_docker/web/Dockerfile`.

```
RUN install-php-extensions <extensionname>
```

If this did not work try this:

```
RUN docker-php-ext-install <extensionname>
```

All available extensions see here: <https://github.com/mlocati/docker-php-extension-installer#supported-php-extensions>.

More information on https://hub.docker.com/_/php/ at *How to install more PHP extensions.

Config:

To edit any `php.ini` config, just add another `.ini` file to `_docker/web/additional-inis/`

2. Webserver Setup

If you need to set the root directory of your web application to something different than `./` (for example `/webroot`) set it in `_docker/web/sites-available/000-default.conf`:

```
# ...
DocumentRoot /var/www/html/webroot
```

3. Database configuration

If you need to configure some database parameters (for example `innodb_file_format`), you can do that in the `_docker/db/my.cnf` file. After that you need to rebuild the `db` container with `docker-compose build db` and restart the container.

4. Custom application configuration files

If you have a git ignored file (for example a `database_config.php`) in your application that normally needs to be edited, you can define a template file, that reduces further configuration of the Docker user.

At container startup all environment variable occurrences within this file will be set to the corresponding value and copied to the defined location.

To do this, first set a file mapping in your `docker-compose.yml` as an environment variable with the prefix `TEMPLATE_`:

```
services:
  web:
    environment:
      TEMPLATE_DB_CONFIG: "database_config.php:./path/to/application/database_config.php"
```

In `_docker/web/templates/` define your configuration template file like this (in this case `database_config.php`):

```
<?php
return [
  "database" => [
    "host" => '${DOMAIN}',
    "user" => '${DATABASE_USER_MAIN}',
    "password" => '${DATABASE_PASS_MAIN}',
    "database_name" => '${DATABASE_NAME_MAIN}'
  ]
];
```

This also supports the bash default syntax with `:-` between variable name and default value:

```
 ${DOMAIN:-test.local}
```

If the environment variable `DOMAIN` is not set, this will default to `test.local`

The variables `DATABASE_USER_MAIN` and `DATABASE_PASS_MAIN` should be set in the `_docker/public.env`. If you have more databases set these variables with another suffix than `_MAIN`:

```
DATABASE_USER_MAIN="some_user"
DATABASE_PASS_MAIN="s3cr3tp4ssw0rd"
DATABASE_USER_SOMEOTHERSUFFIX="another_user"
```

```
DATABASE_PASS_SOMEOTHERSUFFIX="sauce"
```

In this case the literal environment variables in the file `_docker/web/templates/database_config.php` will be replaced with the corresponding environment value and then the file will be copied to `./path/to/application/database_config.php`.

The resulting file `./path/to/application/database_config.php` can be for example:

```
<?php
return [
    "database" => [
        "host" => 'test.local',
        "user" => 'some_user',
        "password" => 's3cr3tP4ssw0rd',
        "database_name" => 'some_db_name'
    ]
];
```

5. Install Composer

Composer is installed in its latest version if you define it as a build argument in the `docker-compose.yml`:

```
services:
  web:
    build:
      args:
        INSTALL_COMPOSER: "true"
```

It runs `composer install` at startup if you set the path where it is executed with the following environment variable in your `docker-compose.yml`:

```
services:
  web:
    # ...
    environment:
      COMPOSER_INSTALL_PATHS: ./
```

If you need to execute `composer install` in multiple paths you can do this by separating them by a `:`:

```
COMPOSER_INSTALL_PATHS: ./path:./another/path
```

If you need another composer version installed (for example `1.x`) you can change this in the `Dockerfile` under `_docker/web/`:

```
# install composer
RUN install-php-extensions @composer-1 && apt-get update && apt-get install -y unzip git
```

More info on this on <https://github.com/mlocati/docker-php-extension-installer#installing-composer>

6. Run `npm install` at startup

Node.js and `npm` is installed if you define it as a build argument in the `docker-compose.yml`:

```
services:
  web:
    build:
      args:
        INSTALL_NPM: "true"
```

The same goes for the installation of `grunt` with the build argument `INSTALL_GRUNT`.

If installed it automatically runs `npm install` in the directory you specified with:

```
services:
  web:
    # ...
    environment:
      NPM_INSTALL_PATHS: ./path/to/sub/dir
```

If you need to execute `npm install` in multiple paths you can do this by separating them by a `:`:

```
NPM_INSTALL_PATHS: ./path:/another/path
```

7. Access other applications from the container

If you need to access other applications from the outside that run inside the container, you need make the port available on the outside.

For example, if you run a Vue.js application with `npm run serve` and this application runs *inside* the container on `http://localhost:8080`, you can make this available on the outside under `http://<DOMAIN>:8080` if you add this to your `docker-compose.yml`:

```
services:
  web:
    ports:
      - "${LOOPBACK_IP:-127.255.255.254}:8080:8080"
```

8. Match your server setup

With Docker you want to match the environment of the server where the application will run later as close as possible. This helps prevent weird errors and bugs that only occur on the live system or only on your development system.

Here are some additional tips to prevent these discrepancies in the first place:

1. Put as many files as possible into git where it makes sense. This way other developers do not have to copy single files from the live system
 - i. put `composer.lock` into git, so other developers have the exact same versions when they install dependencies with `composer install`.
 - ii. put `.htaccess` to git if possible, so for example redirects are treated the same as on the live system.
 - iii. use the template files as explained above to automatically configure database configuration files that should not be in git.
2. Replicate the `php.ini` configuration from the server as close as possible. You can configure your own `.ini` file in `_docker/web/additional-inis`.
3. Install the same PHP version as used on the live system with the `FROM` command in the `_docker/web/Dockerfile`
4. Install the same Composer version as used on the live system. This can prevent dependency installation problems. More information on how to install a specific composer version in the `_docker/web/Dockerfile` on here <https://github.com/mlocati/docker-php-extension-installer#installing-composer>
5. Install all PHP extensions that are required by the application. This is also done in the `_docker/web/Dockerfile`

Literaturverzeichnis

- [1] Adel Faizrakhmanov. *.env files support - plugin for IntelliJ IDEs / JetBrains*. 16. Sep. 2019. URL: <https://plugins.jetbrains.com/plugin/9525--env-files-support> (besucht am 21.06.2021).
- [2] Alexey Efimov, Cezary Butler, Alexander Kriegisch, Johnny Boy, Alexandre Evstigneev, denis, Wi Wi. *Batch Scripts Support - IntelliJ IDEs / JetBrains*. 21. Apr. 2006. URL: <https://plugins.jetbrains.com/plugin/265-batch-scripts-support> (besucht am 21.06.2021).
- [3] Docker Inc. *Docker overview / Docker Documentation*. 6. Apr. 2020. URL: <https://docs.docker.com/get-started/overview/> (besucht am 23.06.2021).
- [4] Jonas Fluhrer. *Shopware 6 eCommerce: Einrichtung, Anpassung und Erweiterung von Shopware 6 Shopsystemen durch individuelle Bundles und Plugins*. T3_2000. DHBW Mosbach, 19. Aug. 2020.
- [5] Docker Inc. *Orientation and setup / Docker Documentation*. 6. Apr. 2020. URL: <https://docs.docker.com/get-started> (besucht am 22.06.2021).
- [6] Docker Inc. *What is a Container? / App Containerization / Docker*. 2014. URL: <https://www.docker.com/resources/what-container> (besucht am 23.06.2021).
- [7] Robert Larson, Janique Carbone. *Windows Server 2008 Hyper-V Resource Kit*. Microsoft Press, 2009.
- [8] Stephen Soltesz u. a. „Container-Based Operating System Virtualization: A Scalable, High-Performance Alternative to Hypervisors“. In: *SIGOPS Oper. Syst. Rev.* 41.3 (März 2007), S. 275–287. ISSN: 0163-5980. DOI: [10.1145/1272998.1273025](https://doi.org/10.1145/1272998.1273025). URL: https://www.win.tue.nl/~tozceleb/2IN05/essay_collection/7%20Container-based%20Operating%20System%20Virtualization.pdf.
- [9] Miguel G. Xavier u. a. „Performance Evaluation of Container-Based Virtualization for High Performance Computing Environments“. In: *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. 2013, S. 233–240. DOI: [10.1109/PDP.2013.41](https://doi.org/10.1109/PDP.2013.41). URL: [https://repository.pucrs.br/dspace/bitstream/10923/13239/2/Performance_Evaluation_of\(Container_Based_Virtualization_for_Hight_Performance_Computing_Environments.pdf](https://repository.pucrs.br/dspace/bitstream/10923/13239/2/Performance_Evaluation_of(Container_Based_Virtualization_for_Hight_Performance_Computing_Environments.pdf).
- [10] Oliver Liebel. *Skalierbare Container-Infrastrukturen: Das Handbuch für Administratoren*. Rheinwerk Verlag, 2017.

- [11] Ian Miell und Aidan Sayers. *Docker in practice*. Simon und Schuster, 2019.
- [12] The Linux Foundation. *Open Container Initiative*. Juni 2015. URL: <https://opencontainers.org/> (besucht am 22.06.2021).
- [13] DevopsCube. *What Is Docker? How Does It Work?* 13. Sep. 2020. URL: <https://devopscube.com/what-is-docker/> (besucht am 22.06.2021).
- [14] Sascha Grunert. *Demystifying Containers - Part I: Kernel Space*. 19. März 2019. URL: <https://medium.com/@saschagrunert/demystifying-containers-part-i-kernel-space-2c53d6979504> (besucht am 22.06.2021).
- [15] Docker Inc. *Docker Hub*. Juni 2014. URL: <https://hub.docker.com/> (besucht am 23.06.2021).
- [16] Docker Inc. *Install Docker Engine on Ubuntu | Docker Documentation*. Apr. 2020. URL: <https://docs.docker.com/engine/install/ubuntu> (besucht am 05.07.2021).
- [17] Docker Inc. *Install Docker Desktop on Windows | Docker Documentation*. Feb. 2017. URL: <https://docs.docker.com/docker-for-windows/install/> (besucht am 01.07.2021).
- [18] Docker Inc. *Dockerfile reference | Docker Documentation*. 2014. URL: <https://docs.docker.com/engine/reference/builder/> (besucht am 24.06.2021).
- [19] Docker Inc. *Best practices for writing Dockerfiles | Docker Documentation*. Juni 2018. URL: https://docs.docker.com/develop/develop-images/dockerfile_best-practices/#leveraging-build-cache (besucht am 24.06.2021).
- [20] Adrian Mouat. *Understanding Volumes in Docker*. 9. Dez. 2014. URL: <https://blog.container-solutions.com/understanding-volumes-docker> (besucht am 28.06.2021).
- [21] Docker Inc. *Use volumes | Docker Documentation*. Aug. 2017. URL: <https://docs.docker.com/storage/volumes/> (besucht am 28.06.2021).
- [22] Docker Inc. *Networking features in Docker Desktop for Windows | Docker Documentation*. März 2018. URL: <https://docs.docker.com/desktop/windows/networking/> (besucht am 18.08.2021).
- [23] Docker Inc. *docker network create | Docker Documentation*. Juni 2017. URL: https://docs.docker.com/engine/reference/commandline/network_create/ (besucht am 02.07.2021).
- [24] Docker Inc. *Container networking | Docker Documentation*. Feb. 2018. URL: <https://docs.docker.com/config/containers/container-networking/> (besucht am 02.07.2021).
- [25] Docker Inc. *Docker Desktop WSL 2 backend | Docker Documentation*. 13. Mai 2020. URL: <https://docs.docker.com/docker-for-windows/wsl/> (besucht am 01.07.2021).

- [26] YAML.org. *YAML Ain't Markup Language*. März 2001. URL: <https://yaml.org/about.html> (besucht am 01.07.2021).
- [27] Docker Inc. *Compose file version 3 reference / Docker Documentation*. Nov. 2020. URL: <https://docs.docker.com/compose/compose-file/compose-file-v3/> (besucht am 01.07.2021).
- [28] Docker Inc. *Environment variables in Compose / Docker Documentation*. Sep. 2016. URL: <https://docs.docker.com/compose/environment-variables/> (besucht am 01.07.2021).
- [29] Matthew McCullough Jon Loeliger. *Version Control with Git, 2nd Edition: Powerful tools and techniques for collaborative software development*. O'Reilly Media, 2012. ISBN: 978-1-4493-1638-9.
- [30] Chet Ramey. „Bash, the Bourne- Again Shell“. In: *Proceedings of The Romanian Open Systems Conference & Exhibition (ROSE 1994), The Romanian UNIX User's Group (GURU)*. 1994, S. 3–5. URL: <http://bashcookbook.com/bashinfo/source/bash-4.0/doc/rose94.pdf> (besucht am 05.07.2021).
- [31] Inc. Free Software Foundation. *bash(1): GNU Bourne-Again SHell - Linux man page*. 4. Juli 2018. URL: <https://linux.die.net/man/1/bash> (besucht am 05.07.2021).
- [32] Bash Hackers Wiki. *Parameter expansion [Bash Hackers Wiki]*. 21. Mai 2021. URL: <https://wiki.bash-hackers.org/syntax/pe> (besucht am 06.07.2021).
- [33] Inc. Free Software Foundation. *Command Substitution (Bash Reference Manual)*. 2007. URL: https://www.gnu.org/software/bash/manual/html_node/Command-Substitution.html (besucht am 07.07.2021).
- [34] Jürgen Wolf. *Shell-Programmierung*. Galileo Press, 2010. URL: http://gxmedia.galileo-press.de.s3.amazonaws.com/leseproben/2440/galileocomputing_shell_programmierung_3..pdf (besucht am 07.07.2021).
- [35] Linus Morten Nyman u. a. „Freedom and forking in open source software: the MariaDB story“. In: *Nordic Academy of Management 2013 (ISSN 2298-3112) Nordisk Företagsekonomisk Förening* (2013).
- [36] MariaDB Foundation. *About MariaDB Server - MariaDB.org*. 2011. URL: <https://mariadb.org/about/> (besucht am 11.08.2021).
- [37] Apache Software Foundation. *Welcome! - The Apache HTTP Server Project*. 2000. URL: <https://httpd.apache.org/> (besucht am 08.07.2021).
- [38] Apache Software Foundation. *core - Apache HTTP Server Version 2.4*. 22. Apr. 2021. URL: <https://httpd.apache.org/docs/2.4/mod/core.html> (besucht am 09.07.2021).

- [39] Lars Eilebrecht, Nikolaus Rath und Thomas Rohde. *Apache Webserver Installation, Konfiguration, Administration*. 4., erw. und überarb. Aufl. Open Source. MITP, 2002. ISBN: 9783826608292.
- [40] ABOUTSSL. *How SSL certificate Works, SSL encryption process*. Juli 2016. URL: <https://aboutssl.org/how-ssl-certificate-work/> (besucht am 09.07.2021).
- [41] Apache Software Foundation. *mod_ssl - Apache HTTP Server Version 2.4*. 19. März 2021. URL: https://httpd.apache.org/docs/2.4/mod/mod_ssl.html (besucht am 09.07.2021).
- [42] Filippo Valsorda. *FiloSottile/mkcert*. Juli 2018. URL: <https://github.com/FiloSottile/mkcert> (besucht am 03.08.2021).
- [43] OpenBSD. *hosts(5) - OpenBSD manual pages*. 28. März 2019. URL: <https://man.openbsd.org/hosts> (besucht am 03.08.2021).
- [44] The PHP Group. *PHP: Hypertext Preprocessor*. 1998. URL: <https://www.php.net/> (besucht am 09.07.2021).
- [45] The PHP Group. *PHP: History of PHP - Manual*. 2009. URL: <https://www.php.net/manual/en/history.php.php> (besucht am 09.07.2021).
- [46] The PHP Group. *PHP: Supported Versions*. 8. Juli 2021. URL: <https://web.archive.org/web/20210708022935/https://www.php.net/supported-versions.php> (besucht am 08.07.2021).
- [47] The PHP Group. *PHP: Prepared Statements - Manual*. 13. Juni 2021. URL: <https://www.php.net/manual/de/mysql.quickstart.prepared-statements.php> (besucht am 09.07.2021).
- [48] Pimcore GmbH. *System Requirements - Pimcore*. Apr. 2017. URL: https://pimcore.com/docs/pimcore/current/Development_Documentation/Installation_and_Upgrade/System_Requirements.html (besucht am 09.07.2021).
- [49] The PHP Group. *PHP: Imagick Einführung - Manual*. 13. Juni 2021. URL: <https://www.php.net/manual/de/intro.imagick.php> (besucht am 09.07.2021).
- [50] ImageMagick Studio LLC. *ImageMagick - Convert, Edit, or Compose Digital Images*. 2005. URL: <https://imagemagick.org/index.php> (besucht am 09.07.2021).
- [51] Josh Lockhart. *Modern PHP: New features and good practices*. O'Reilly Media, Inc., 2015.
- [52] Derick Rethans. *Xdebug - Debugger and Profiler Tool for PHP*. Mai 2002. URL: <https://xdebug.org/> (besucht am 19.07.2021).
- [53] Derick Rethans. *Xdebug: Documentation » Step Debugging*. Nov. 2020. URL: https://xdebug.org/docs/step_debug (besucht am 19.07.2021).

- [54] Docker Inc. *Networking features in Docker Desktop for Windows* / Docker Documentation. März 2018. URL: <https://docs.docker.com/docker-for-windows/networking/> (besucht am 19.07.2021).
- [55] Jordi Boggiano und Nils Adermann. *Packagist*. 2011. URL: <https://packagist.org/> (besucht am 19.07.2021).
- [56] Tom Preston-Werner. *Semantic Versioning 2.0.0* / *Semantic Versioning*. Dez. 2009. URL: <https://semver.org/> (besucht am 20.07.2021).
- [57] Jordi Boggiano, Nils Adermann und many community contributors. *Versions and constraints - Composer*. Juni 2015. URL: <https://getcomposer.org/doc/articles/versions.md> (besucht am 20.07.2021).
- [58] Jordi Boggiano, Nils Adermann und many community contributors. *Basic usage - Composer*. Feb. 2012. URL: <https://getcomposer.org/doc/01-basic-usage.md> (besucht am 20.07.2021).
- [59] OpenJS Foundation. *Node.js*. Sep. 2015. URL: <https://nodejs.org/en/> (besucht am 21.07.2021).
- [60] Inc. npm. *npm About*. Jan. 2015. URL: <https://www.npmjs.com/about> (besucht am 21.07.2021).
- [61] Inc. npm. *About semantic versioning / npm Docs*. Dez. 2018. URL: <https://docs.npmjs.com/about-semantic-versioning> (besucht am 21.07.2021).
- [62] OpenJS Foundation. *Grunt: The JavaScript Task Runner*. Apr. 2012. URL: <https://gruntjs.com/> (besucht am 21.07.2021).
- [63] Ian Kent. *mailhog/MailHog: Web and API based SMTP testing*. 2020. URL: <https://github.com/mailhog/MailHog> (besucht am 21.07.2021).
- [64] Bob Reselman und Matthew Broberg (Red Hat). *Developer experience: an essential aspect of enterprise architecture* / Enable Architect. 18. Nov. 2020. URL: <https://www.redhat.com/architect/developer-experience> (besucht am 23.07.2021).
- [65] Chromium Project. *Prefer Secure Origins For Powerful New Features - The Chromium Projects*. Jan. 2015. URL: <http://www.chromium.org/Home/chromium-security/prefer-secure-origins-for-powerful-new-features> (besucht am 02.08.2021).
- [66] World Wide Web Consortium (W3C). *Secure Contexts*. 30. Aug. 2016. URL: <https://www.w3.org/TR/powerful-features/> (besucht am 02.08.2021).

- [67] Mozilla Corporation. *Service-Worker benutzen - Web API Referenz / MDN*. Sep. 2020. URL: https://developer.mozilla.org/de/docs/Web/API/Service_Worker_API/Using_Service_Workers (besucht am 26.07.2021).
- [68] Johan Bergström. *jbergstroem/mariadb-alpine: A tiny MariaDB image*. Sep. 2017. URL: <https://github.com/jbergstroem/mariadb-alpine> (besucht am 27.07.2021).
- [69] Docker Community. *Php - Official Image | Docker Hub*. Sep. 2015. URL: https://hub.docker.com/_/php (besucht am 27.07.2021).
- [70] Michele Locati. *mlocati/php-extension-installer - Docker Image | Docker Hub*. Juli 2021. URL: <https://hub.docker.com/r/mlocati/php-extension-installer> (besucht am 28.07.2021).
- [71] John McCracken. *Isolating PHP with Docker Containers - DEV Community*. 1. Nov. 2018. URL: <https://dev.to/johnmccuk/isolating-php-with-docker-containers-4epn> (besucht am 05.08.2021).
- [72] Jason Rowland. *docker-compose does not pass environment variables true/false unchanged*. 29. Juli 2015. URL: <https://github.com/docker/compose/issues/1788> (besucht am 29.07.2021).
- [73] IEEE und The Open Group. *The Open Group Base Specifications Issue 7, 2018 edition*. 2018. URL: <https://pubs.opengroup.org/onlinepubs/9699919799/> (besucht am 30.07.2021).
- [74] Derick Rethans. *Xdebug: Documentation » Supported Versions and Compatibility*. Dez. 2020. URL: <https://xdebug.org/docs/compat> (besucht am 03.08.2021).
- [75] Derick Rethans. *Xdebug: Documentation » All settings*. Dez. 2020. URL: https://xdebug.org/docs/all_settings (besucht am 03.08.2021).
- [76] Aaron Swartz. *Markdown (Aaron Swartz: The Weblog)*. 2004. URL: <http://www.aaronsw.com/weblog/001189> (besucht am 09.08.2021).
- [77] Cake Software Foundation. *CakePHP - Build fast, grow solid | PHP Framework | Home*. 2019. URL: <https://cakephp.org/> (besucht am 05.08.2021).
- [78] Simon Ferquel. *Docker <3 WSL 2 - The Future of Docker Desktop for Windows - Docker Blog*. 16. Juni 2019. URL: <https://www.docker.com/blog/docker-hearts-wsl-2/> (besucht am 05.08.2021).
- [79] DeepL GmbH. *DeepL Translate – Der präziseste Übersetzer der Welt*. 2017. URL: <https://www.deepl.com> (besucht am 05.08.2021).
- [80] Apache Friends. *XAMPP Installers and Downloads for Apache Friends*. 3. Aug. 2021. URL: <https://www.apachefriends.org/de/index.html> (besucht am 09.08.2021).