Duale Hochschule Baden-Württemberg Mannheim

Projektberichtarbeit

Drei-Schicht Anwendung

Studiengang Informatik

Studienrichtung Angewandte Informatik

Verfasser(in):	Moritz Werr, Phil Richter, Max Stege
Matrikelnummer:	5401527
Matrikelnummer:	4164342
Matrikelnummer:	7285772
Kurs:	TINF22AI2
Dozent:	Jochen Kluger
Bearbeitungszeitraum:	01.10.2024 – 20.12.2024

Ehrenwörtliche Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit mit dem Titel "*Drei-Schicht Anwendung*" 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.

Quelltextverzeichnis

1 Einleitung (Moritz Werr)

Im Rahmen dieses Projektes wurde eine Anwendung in drei Schichten aufgeteilt mithilfe von Docker und Docker-Compose. Diese wurde im Rahmen eines vorherigen Semesters implementiert als eine monolithische Serveranwendung, obwohl diese jedoch eigentlich aus mehreren Komponenten besteht. Generell besteht diese aus drei Teilen: dem Webserver und der Programmierlogik für die Anwendung, einem Datenbankserver für die persistente Speicherung der Daten und einer TLS-Verschlüsselung für die Sicherheit der Datenübertragung. Vorher liefen diese alle auf einem Server, es war eine monolithische Architektur. Nun wird diese in ihren einzelnen Funktionalitäten aufgespalten, hin zu einer Mikroservicearchitektur. Jede einzelne Funktionalität wird in einen eigenen Softwarecontainer übertragen und laufen isoliert auf der gleichen Maschine, indem sie sich den Betriebsystemkernel teilen [20, Vgl. S. 152444]. Dieser Prozess nennt sich Containerisierung.

Durch die Umwandlung in eine Mikroservicearchitektur wird ein hoher Grad an Flexibilität gewonnen. Die einzelnen Komponenten arbeiten unabhängig voneinander und ergeben gemeinsam ein funktionierendes System [6, Vgl. S.1f]. Dafür muss die Anwendung jedoch in manchen Fällen umgeschrieben werden. Durch den Umbau zu einer Mikroservicearchitektur, wird oftmals die Komplexität der Anwendung erhöht [18, Vgl. S.10], weswegen noch weitere Werkzeuge zur Verwaltung der Anwendung benötigt werden. Hilfswerkzeuge zur Verwaltung und Konfiguration der Anwendung werden benötigt.

Weiterhin sind nicht nur die einzelnen Komponenten austauschbar, sondern können nun auch weitere Funktionalitäten durch die Containerisierung erreicht werden. Es können sehr einfach Healthchecks implementiert werden, die überprüfen, ob einzelne Komponenten noch richtig funktionieren [17]. Wenn dies nicht mehr der Fall ist, können die einzelnen Komponenten der drei Schichten neugestartet werden [17]. Dafür muss definiert werden wie diese miteinander zusammenarbeiten. Wenn nun ein einzelner Container Probleme hat, wird dies erkannt und er kann automatisch neugestartet werden. Weiterhin sorgt dies auch dafür, dass Container in der richtigen Reihenfolge gestartet werden [17].

Im Rahmen dieses Projektes wurden Docker und Docker-Compose verwendet aufgrund ihrer guten Dokumentation und großen Anzahl an Features. Kernbestandteil war Docker zur Containerisierung und Docker-Compose zur Verwaltung der Container. Es musste ein neues Dockerfile für den Webserver erstellt werden, zwei vorhandene Containerimages wurden für die TLS-Verschlüsselung (traefik) bzw. die Datenspeicherung (mariadb) verwendet und ein Initialisierungsskript für die Datenbank wurde erstellt.

2 Docker als Softwarecontainer (Moritz Werr)

Für die Containerisierung wurde Docker verwendet. Softwarecontainer bieten viele Vorteile gegenüber traditioneller Virtualisierung. Traditionelle Virtualisierung wird verwendet um Ressourcen effizienter zu nutzen. Dabei werden viele schwache Maschinen in einer großen Maschine konsolidiert [20, Vgl. S.152443]. Diese teilen sich alle die gleichen Hardwareressourcen, welche über den Hypervisor auf die einzelnen virtuellen Maschinen verteilt wird. Allerdings teilen sich virtuelle Maschinen nicht ihre Softwarekomponenten. Dadurch wird oftmals der gleiche Programmcode öfters im Arbeitsspeicher vorkommen. Softwarecontainer funktionieren in dieser Hinsicht anders. Sie teilen sich normalerweise den Kernel mit dem Containerhost [20, Vgl. S.152444]. Diese sind dadurch um einiges leichtgewichtiger und schneller als traditionelle Virtualisierung [3, Vgl. S.137]. Aufgrund dieser Leichtgewichtigkeit und Geschwindigkeit lassen sich Docker-Anwendungen mit Orchestrierungswerkzeugen wie Kubernetes auch sehr leicht horizontal und vertikal skalieren [20, Vgl. S.152458].

Weiterhin sind Anwendungen innerhalb von Docker-Containern auch sehr portabel, da ein Container nicht vom Hostsystem abhängig ist, sondern von dessen Docker-Image. Alle Abhängigkeiten (Dependencies) sind in einem einzelnen Image gebündelt [3, Vgl. S.137]. Dadurch sind containerisierte Anwendungen hoch portabel. Sie können auf jeder Maschine laufen, die eine kompatible Docker-Engine hat. Nicht nur läuft die Anwendung auf sehr vielen Maschinen, sondern es läuft auch überall die gleiche Version. Das Image sorgt für einen hohen Grad an Konsistenz und Wiederholbarkeit.

Außerhalb der hohen Effizienz und Portabilität von Softwarecontainern, bieten diese auch noch einen gewissen Grad an Sicherheit. Obwohl virtuelle Maschinen einen höheren Grad an Sicherheit bieten durch getrennte Kernel [20, Vgl. S.152455], bieten Container einen guten Schutz durch Prozessisolierung, Dateisystemisolierung, Ressourcenlimitierung und Netzwerkisolation [20, Vgl. S.152444ff]. Dafür werden Funktionalitäten des Linux Kernels verwendet. Primär wären dies CGroups, Namespaces und MAC-Systemen (Mandatory Access Control) wie SELinux oder Apparmor. Cgroup sorgen dabei für die Resourcenlimitierung, Namespaces für die Prozess- und Netzwerkisolation und Apparmor für die Dateisystemisolierung.

Durch die Nutzung von Docker ist die Anwendung sehr performant, portabel und auch noch gut abgesichert.

3 Docker Compose (Max Stege)

Docker Compose ist ein essentielles Werkzeug zur Verwaltung von Multi-Container-Anwendungen. Es bietet eine Vielzahl von Vorteilen, die den Entwicklungs- und Betriebssprozess erheblich vereinfachen:

1. **Multi-Container-Verwaltung**: Docker Compose erlaubt es, mehrere Container mit einer einzigen YAML-Datei zu orchestrieren. Dies ist besonders nützlich in komplexen Anwendungen wie der hier entwickelten Drei-Schichten-Anwendung, wo Backend, Frontend und Datenbank in getrennten Containern laufen [11].
2. **Einfachere Konfiguration**: Anstatt Container manuell mit zahlreichen `docker run`-Befehlen zu starten, wird die gesamte Konfiguration in einer zentralen Datei (z. B. `docker-compose.yml`) definiert. Dies umfasst Details wie Netzwerkverbindungen, Umgebungsvariablen, Abhängigkeiten und Healthchecks [2].
3. **Übersichtlichkeit**: Docker Compose bietet eine klare Struktur für komplexe Anwendungen. In der YAML-Datei können Dienste wie der Webserver, die Datenbank und der TLS-Proxy logisch voneinander getrennt definiert werden, was die Wartbarkeit erleichtert [19].
4. **Wartbarkeit**: Durch die zentrale Definition aller Dienste wird der Wartungsaufwand minimiert. Änderungen, wie das Hinzufügen neuer Dienste oder das Anpassen von Konfigurationen, können unkompliziert in der YAML-Datei vorgenommen werden [2].
5. **Bessere Bedienbarkeit**: Mit einfachen Befehlen wie `docker-compose up` oder `docker-compose down` können alle Container gleichzeitig gestartet oder gestoppt werden. Zusätzlich ermöglicht die Option `-build`, neue Container-Builds direkt aus dem Quellcode zu erstellen [11].

In der entwickelten Webanwendung wurde Docker Compose eingesetzt, um die Komponenten Webserver, Datenbank und TLS-Proxy zu koordinieren:

- **Webserver**: Das Docker Compose-Setup definiert das Image, die Volumes für Bilder, Labels für die TLS-Integration über den Traefik-Proxy und wichtige Umgebungsvariablen, die die Verbindung zur Datenbank regeln. Healthchecks stellen sicher, dass der Webserver ordnungsgemäß funktioniert [12].
- **Datenbank**: Die MariaDB-Datenbank wird ebenfalls als eigenständiger Dienst mit spezifischen Umgebungsvariablen und Volumes betrieben. Ein Initialisierungsskript wird beim Start des Containers ausgeführt, um die Datenbankstruktur zu erstellen [19].

- **TLS-Proxy**: Der Traefik-Container dient als Proxy, der sowohl HTTP- als auch HTTPS-Anfragen handhabt. Hierfür werden spezifische Ports und Konfigurationsdateien genutzt, um eine sichere Datenübertragung zu gewährleisten [11].

Die Verwendung von Docker Compose bietet nicht nur Effizienz und Konsistenz bei der Entwicklung, sondern verbessert auch die Skalierbarkeit und Stabilität der Anwendung im Produktivbetrieb.

4 Die Webanwendung (Phil Richter)

Die Webanwendung besteht aus einem Internetforum, in dem Benutzer Beiträge erstellen, kommentieren und liken können. Um am Forum teilzunehmen, muss sich der Benutzer mit einer Emailadresse und einem Passwort registrieren und einloggen. Falls der Benutzer sein Passwort vergisst, kann er dieses mit Hilfer einer Emailadresse zurücksetzen und ein neues vergeben. Des Weiteren existiert ein Admin Account mit erweiterten Rechten. Dieser kann Beiträge von allen Nutzern löschen, während normale Benutzer nur ihre eigenen Beiträge löschen können. Zudem kann dieser sowohl die Emailadresse als auch das Passwort eines Benutzers ändern. Falls notwendig kann ein Admin auch einen Account löschen.

Die Anwendung besteht im Frontend aus HTML [7], CSS [1] und JavaScript [13]. Das Backend wurde mit PHP [16] geschrieben. Zusätzlich existiert noch eine Datenbank, in der die Benutzerdaten und Beiträge gespeichert werden. Die Datenbank wurde mit MariaDB [14] realisiert. Die Mails werden mit msmtp [15] über einen SMTP-Server von Gmail versendet.

5 Webserver

Für den Webserver wurde ein Dockerfile implementiert welches wiefolgt aussieht:

```
1  FROM php:8.1-apache
2  COPY src/ /var/www/html
3  RUN apt-get update && apt-get install -y msmtp && rm -rf /var/
       cache/apt/lists
4  COPY msmtprc /etc/msmtprc
5  RUN chmod 600 /etc/msmtprc && chown www-data: /etc/msmtprc
6  RUN mkdir /var/www/html/pictures
7  RUN sed -i 's/AllowOverride None/AllowOverride All/g' /etc/
       apache2/apache2.conf
8  RUN printf "AddType application/x-httpd-php .html\nAddType
       application/x-httpd-php .htm\n" >> /etc/apache2/apache2.
       conf
9  RUN a2enmod rewrite
10 RUN chown www-data: -R /var/www/html/
11 RUN docker-php-ext-install mysqli && docker-php-ext-enable
       mysqli
12 RUN echo "sendmail_path = /usr/bin/msmtp -t" >> /usr/local/etc
       /php/php.ini-development
13 RUN cp /usr/local/etc/php/php.ini-development /usr/local/etc/
       php/php.ini
14 CMD apachectl -DFOREGROUND
```

Quelltext 5.1: dockerfile

Im ersten Schritt wird in Zeile 1 das Basisimage `php: 8.1-apache` gesetzt. Dies ist ein bereis vorkonfiguriertes Image mit PHP 8.1 und Apache Webserver. In Zeile 2 wird der Quellcode der Webanwendung, welche im Ordner `src/` liegt, in den Ordner `/var/www/html` des Containers kopiert. Damit die Webanwendung erfolgreich ausgefürt wird, müssen verschiedene Konfigurationen von Apache vorgenommen werden. Dies passiert in Zeile 7 bis 9. Dabei wird unteranderem das Apache Modul `rewrite` aktiviert, sowie angepasst, dass die Dateiendungen `.html` und `.htm` als PHP-Dateien interpretiert werden.

Zusätzlich werden weitere, für die Webanwendung notwendige Komponenten installiert. Dazu gehört für den Mailversand `msmtp` in Zeile 3 und für die Datenbank `mysqli` in Zeile 11. Damit msmtp verwendet wird, wird in Zeile 12 der `sendmail_path` in der PHP-Konfiguration korrekt gesetzt. Zudem wird in Zeile 4 die Konfigurationsdatei `msmtprc` in den Container

kopiert. Diese Datei enthält die Konfigurationen wie unter anderem Host und Port für den Mailversand.

Des Weitren wird in Zeile 6 ein Ordner `pictures` erstellt, in welchem später die von den Nutzern hochgelandenen Bilder gespeichert werden. In Zeile 5 und 10 wird der Besitzer der Dateien und Ordner auf `www-data` gesetzt. Dadurch wird sichergestellt, dass der Apache Webserver auf die Dateien zugreifen kann. Zum Schluss wird in Zeile 14 der Apache Webserver gestartet.

Um den Webserver als eigenständigen Container zu betreiben wird folgender Code in der `docker-compose.yml` Datei definiert:

```
1     webserver:
2     image: webserver
3     build: .
4     volumes:
5       - pictures:/var/www/html/pictures
6     labels:
7       - "traefik.enable=true"
8       - "traefik.http.routers.webserver.rule=Host(‘localhost‘)"
9       - "traefik.http.services.webserver.loadbalancer.server.port=80"
10      - "traefik.http.routers.webserver.entrypoints=web,websecure"
11      - "traefik.http.routers.webserver.tls=true"
12    environment:
13      DATABASE_USER: web_eng_user
14      DATABASE_HOST: mariadb
15      DATABASE_PASS: linux
16      DATABASE_NAME: web_eng
17      GMAIL_APP_PASSWORD: ${GMAIL_APP_PASSWORD}
18    depends_on:
19      db:
20        condition: service_healthy
21        restart: true
22    healthcheck:
23      test: ["CMD", "curl", "-f", "http://localhost"]
24      interval: 10s
25      timeout: 5s
26      retries: 3
27      start_period: 10s
```

Quelltext 5.2: `docker-compose.yml`

Dabei wird in Zeile 2 das Image `webserver` definiert, welches in Zeile 3 aus dem aktuellen Verzeichnis, dem dockerfile, gebaut wird. In Zeile 5 wird ein Volume `pictures` definiert. Dies

ist dazu notwendig, dass die hochgeladenen Bilder auch nach einem Neustart des Containers noch vorhanden sind. Die in Zeile 6 und folgenden definierten Labels sind dazu notwendig, dass der Webserver über den TLS-Proxy `traefik` erreichbar ist. Außerdem wird festgelegt, dass der Webserver auf Localhost erreichbar ist und auf Port 80 lauscht. Zusätzlich wird tls aktiviert.

Um auf die Datenbank zugreifen zu können und Mails über Gmail versenden zu können, werden in Zeile 12 und folgende die notwendigen Umgebungsvariablen definiert und gesetzt. Dabei wird das Passwort für den Gmail Account aus einer Lokalen .env Datei gelesen. Somit wird sichergestellt, dass das Passwort nicht im Klartext vorliegt.

Da der Webserver auf die Datenbank angewiesen ist, wird in Zeile 18 bis 21 eine Abhängigkeit definiert. Dabei wird überprüft, ob die Datenbank erreichbar ist. Falls nicht, wird diese neu gestartet. Des Weiteren wird in Zeile 22 bis 27 ein Healthcheck definiert. Dabei wird alle 10 Sekunden mit Hilfe eines Curl-Befehls überprüft, ob der Webserver erreichbar ist. Falls der Healthcheck dreimal fehlschlägt, gilt der Container als "unhealthy".

6 Datenbank (Max Stege)

Die Datenbank ist eine der zentralen Komponenten der Drei-Schichten-Architektur. In diesem Projekt wurde MariaDB als Datenbankserver eingesetzt, der mithilfe von Docker Compose als eigenständiger Container betrieben wird. Die Konfiguration des Datenbankdienstes in der Docker-Compose-Datei sieht wie folgt aus:

```
db:
    image: mariadb
    hostname: mariadb
    restart: always
    volumes:
      - db:/var/lib/mysql
      - ./database.sql:/docker-entrypoint-initdb.d/database.
        sql
    environment:
      MARIADB_USER: web_eng_user
      MARIADB_PASSWORD: linux
      MYSQL_ROOT_PASSWORD: example
    healthcheck:
      test: ["CMD", "healthcheck.sh", "--connect", "--
        innodb_initialized"]
      start_period: 10s
      interval: 10s
      timeout: 5s
      retries: 3
```

Quelltext 6.1: `docker-compose.yml`

Der Datenbankcontainer verwendet das MariaDB-Image, ein populäres und performantes Open-Source-Datenbankmanagementsystem [5]. Der Hostname des Containers wird explizit auf `mariadb` gesetzt, um eine klare Identifikation im Docker-Netzwerk zu ermöglichen. Das Volumemapping `db:/var/lib/mysql` sorgt dafür, dass die Datenbank persistent bleibt und bei einem Neustart des Containers erhalten bleibt [10]. Zusätzlich wird ein SQL-Skript `database.sql` eingebunden, das die initiale Struktur der Datenbank definiert.

Um die Datenbank zu konfigurieren, werden verschiedene Umgebungsvariablen definiert:

- `MARIADB_USER`: Der Benutzername für den Datenbankzugriff.
- `MARIADB_PASSWORD`: Das Passwort für den definierten Benutzer.

- `MYSQL_ROOT_PASSWORD`: Das Passwort für den Root-Benutzer der Datenbank.

Diese Variablen ermöglichen eine flexible Anpassung der Datenbankkonfiguration, ohne Änderungen am Containerimage vorzunehmen [9].

Ein Healthcheck wird definiert, um sicherzustellen, dass die Datenbank ordnungsgemäß funktioniert. Der Test prüft, ob die Datenbank erreichbar ist und ob das InnoDB-Subsystem korrekt initialisiert wurde [8]. Die Parameter `start_period`, `interval`, `timeout` und `retries` legen fest, wie oft und in welchen Zeitabständen der Healthcheck durchgeführt wird.

Die Datenbank ist essentiell für die persistente Speicherung der Anwendungsdaten. Der Webserver kommuniziert direkt mit der Datenbank, um Benutzerinformationen, Forenbeiträge und andere relevante Daten zu speichern und abzurufen [4]. Dank der Containerisierung kann die Datenbank isoliert und unabhängig von anderen Komponenten betrieben werden, was die Wartung und Skalierung erleichtert [19].

7 TLS-Proxy (Moritz Werr)

Im der Docker-Compose-Datei `docker-compose.yml` definiert folgender Abschnitt den Traefik-Container:

```
1
2  services:
3    traefik:
4      image: traefik:v2.10
5      command:
6        - "--api.insecure=true"
7        - "--providers.docker=true"
8        - "--entrypoints.web.address=:80"
9        - "--entrypoints.websecure.address=:443"
10       - "--providers.file.filename=/dynamic/traefik_dynamic.
           yml"
11     ports:
12       - "80:80"
13       - "443:443"
14       - "8080:8080"
15     volumes:
16       - "/var/run/docker.sock:/var/run/docker.sock"
17     depends_on:
18       db:
19         condition: service_healthy
20         restart: true
21       webserver:
22         condition: service_healthy
23         restart: true
```

Quelltext 7.1: `docker-compose.yml`

Traefik übernimmt hier die TLS-Verschlüsselung für die Webanwendung. Dafür gibt es bereits ein existierendes Containerimage, weswegen dieses nur noch konfiguriert werden muss. Es wird ebenfalls in der gleichen Docker-Compose-Datei definiert.

Zuerst wird das Image ausgewählt für den Service mit dem Namen `traefik` genannt. Danach werden noch weitere Kommandos erwähnt, damit Traefik richtig konfiguriert ist . Primär muss TLS aktiviert werden und die entsprechenden HTTP- und HTTPS-Ports freigegeben werden.

In unserem Fall wurde das Standardzertifikat von Traefik verwendet, weil selbstsignierte Zertifikate hinterlegen durchaus schwierig ist. Bei uns handelt es um einen Prototypen und nicht um ein produktionsreifes Endergebnis.

Als nächstes werden Port 80 und 443 für die Kommunikation freigegeben. Weiterhin wird der Port 8080 für das Managen des Containers freigegeben. Der normale Webservercontainer ist nicht außerhalb des Docker-Netzwerkes erreichbar, weswegen über den Traefik-Container kommuniziert werden muss.

Weiterhin wird der Docker-Socket vom Hostsystem in den Container gereicht, damit der Traefik-Container den Webservercontainer finden kann (Service Discovery).

Als letztes werden die Healthchecks für den Traefik-Container definiert. Da eine TLS-Verschlüsselung nicht ohne Applikationsserver funktioniert, mus dieser funktionieren. Weiterhin kann der Applikationsserver nicht ohne die Datenbank funktionieren. Der Traefik-Container ist nur indirekt von diesem abhängig, trotzdem wird dieser hier erwähnt, um die Abhängigkeit zu verdeutlichen.

8 Bedienung der Anwendung (Moritz Werr)

Bevor die Anwendung gestartet werden kann, muss zuerst das Image des Webservers gebaut werden. Dies wird normalerweise mit dem Befehl `docker build` gemacht. Der folgende Befehl baut das Image, wenn man sich im Wurzelverzeichnis dieses Projektes befindet:

```
docker build . -t webserver
```

bzw.

```
docker compose build
```

Nachdem das Image gebaut wurde, kann die Anwendung gestartet werden. Im Rahmen unseres Projektes haben wir die Emailauthentifizierung größtenteils hartkodiert. Die Datei `.env` beinhaltet das Passwort für den Emailversand und ist der Einfachheit in der ZIP-Datei enthalten, jedoch nicht auf Github. Wenn ein eigener Emailprovider verwendet werden soll, muss die Datei `msmtp.rc` im Projektordner angepasst werden.

Die Anwendung kann durch individuelle `docker run`-Befehle gestartet werden, wird aber wegen der großen Anzahl an Parametern nicht bevorzugt. Stattdessen wird mit Docker-Compose die bereits konfigurierte Anwendung gestartet:

```
docker compose up -d
```

Damit der Befehl funktioniert muss im Wurzelverzeichnis des Projektes der Befehl ausgeführt werden. Sonst muss der Pfad zu Datei explizit angegeben werden:

```
docker compose -f /path/to/project/folder/docker-compose.yml up -d
```

Das Bauen des Images kann hier auch vollführt werden:

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

Damit wird die Anwendung im Hintergrund gestartet und erstellt auch noch automatisch die definierten Docker-Volumes. Weiterhin wird ein eigenes Docker-Netzwerk für die Anwendungscontainern erstellt. Dieses kann über die Hostports 80, 443 und 8080 erreicht werden. Die Anwendung sollte nun über folgenden Link: **https://localhost/** erreichbar sein.

Dort können Sie sich mit den Anmeldeinformationen aus der SQL-Datei anmelden (Standardmäßig: *admin* bzw. *user* mit dem Passwort *Test12345!*).

Anschließend kann die Anwendung mit dem folgenden Befehl wieder beendet werden:

```
docker compose down
```

Bibliography

[1] *Cascading Style Sheets*. URL: https://www.w3.org/Style/CSS/Overview.en.html (visited on 12/20/2024).

[2] Felipe Bedinotto Fava et al. "Assessing the Performance of Docker in Docker Containers for Microservice-Based Architectures". In: *32nd Euromicro International Conference on Parallel, Distributed and Network-Based Processing* (2024), pp. 137–142. DOI: 10.1109/PDP62718.2024.00026.

[3] Felipe Bedinotto Fava et al. "Assessing the Performance of Docker in Docker Containers for Microservice-Based Architectures". In: *2024 32nd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. 2024 32nd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP). ISSN: 2377-5750. Mar. 2024, pp. 137–142. DOI: 10.1109/PDP62718.2024.00026. URL: https://ieeexplore.ieee.org/document/10495554 (visited on 12/20/2024).

[4] MariaDB Foundation. *MariaDB Connectivity*. https://mariadb.com/kb/en/connect/. Accessed: 2024-12-20. 2024.

[5] MariaDB Foundation. *MariaDB Documentation*. https://mariadb.com/kb/en/documentation/. Accessed: 2024-12-20. 2024.

[6] Martin Fowler and James Lewis. "Microservices: Nur ein weiteres Konzept in der Softwarearchitektur oder mehr". In: *Objektspektrum* 1.2015 (2015), pp. 14–20.

[7] *HTML Standard*. URL: https://html.spec.whatwg.org/multipage/ (visited on 12/20/2024).

[8] Docker Inc. *Docker Healthcheck Documentation*. https://docs.docker.com/engine/reference/builder/#healthcheck. Accessed: 2024-12-20. 2024.

[9] Docker Inc. *Environment Variables in Compose*. https://docs.docker.com/compose/environment-variables/. Accessed: 2024-12-20. 2024.

[10] Docker Inc. *Managing Data in Docker*. https://docs.docker.com/storage/. Accessed: 2024-12-20. 2024.

[11] Docker Inc. *Overview of Docker Compose*. https://docs.docker.com/compose/overview/. Accessed: 2024-12-20. 2024.

[12] Docker Inc. *Services Top-Level Elements*. https://docs.docker.com/reference/compose-file/services/. Accessed: 2024-12-20. 2024.

[13] *Learn JavaScript Online - Courses for Beginners - javascript.com*. URL: https://www.javascript.com/ (visited on 12/20/2024).

[14] *MariaDB Foundation*. MariaDB.org. URL: https://mariadb.org/ (visited on 12/20/2024).

[15] *msmtp - about*. URL: https://marlam.de/msmtp/ (visited on 12/20/2024).

[16] *PHP: Hypertext Preprocessor*. Dec. 19, 2024. URL: https://www.php.net/index.php (visited on 12/20/2024).

[17] *Services top-level elements*. Docker Documentation. 100. URL: https://docs.docker.com/reference/compose-file/services/ (visited on 12/20/2024).

[18] Ruoyu Su and Xiaozhou Li. "Modular Monolith: Is This the Trend in Software Architecture?" In: *2024 IEEE/ACM International Workshop New Trends in Software Architecture (SATrends)*. 2024 IEEE/ACM International Workshop New Trends in Software Architecture (SATrends). Apr. 2024, pp. 10–13. URL: https://ieeexplore.ieee.org/document/10669865 (visited on 12/20/2024).

[19] Junzo Watada et al. "Emerging Trends, Techniques and Open Issues of Containerization: A Review". In: *IEEE Access* 7 (2019), pp. 152443–152472. DOI: 10.1109/ACCESS.2019.2945930.

[20] Junzo Watada et al. "Emerging Trends, Techniques and Open Issues of Containerization: A Review". In: *IEEE Access* 7 (2019). Conference Name: IEEE Access, pp. 152443–152472. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2945930. URL: https://ieeexplore.ieee.org/abstract/document/8861307 (visited on 12/20/2024).