Hochschule Bremerhaven

Fachbereich II
Management und Informationssysteme
Wirtschaftsinformatik B.Sc.

Modul
Vernetzte Systeme

Analyse verschiedener Dockerkonstellationen

Vorgelegt von:	Junior Ekane	MatNr. 40128
	Florian Quaas	MatNr. 39952
	Herman Tsago	MatNr. 00000
	Steve Aguiwo II	MatNr. 40088

Vorgelegt am: 28. Februar 2024

Dozent: Prof. Dr. Oliver Radfelder
Dozentin: Prof. Dr. Karin Vosseberg

Inhaltsverzeichnis

1 Einleitung 4
1.1 Vorstellung des Moduls Vernetzte Systeme 4

2 Projektablauf 4
2.1 Vorstellung der Projektidee 4
2.2 Projektorganisation und Teamarbeit 4

3 Werkzeuge 4
3.1 Verwendete Werkzeuge 4

4 Docker 7
4.1 Was ist ein Docker? 7
4.1.1 Wie funktioniert Docker ? 8
4.2 Was ist ein Netzwerk? 8
4.3 Was ist ein Image? 8
4.3.1 Wie erstellt man ein Docker-Image ? 9
4.4 Was ist ein Container ? 9
4.4.1 Wie erstellt und startet man einen Container? 9
4.4.2 Überprüfung des Container-Status 10
4.5 Wie stoppt und entfernt man einen Container? 10
4.6 Was sind die Vor - und Nachteile von Docker ? 11
4.6.1 Vorteile 11
4.6.2 Nachteile 11
4.7 Wie haben wir Docker verwendet? 12

5 Ergebnis 13
5.1 Analyse mit zwei laufenden Containern 13
5.1.1 Aufbau der Image: `fancy` 14
5.1.2 Aufbau unseres lokalen Netzwerkes: mynet 14
5.1.3 Aufbau der Container: `work und service` 15
5.1.4 Lasttest mit zwei Containern 16
5.1.5 Hoffnung und Erwartungen 17
5.2 Analyse mit drei laufenden Containern 18
5.3 Analyse mit vier laufenden Containern 18

6 Lasttest 18
6.1 Was haben wir analysiert? 18
6.2 Wie haben wir das analysiert? 18
6.3 Endergebnis 18

7 Reflexion **18**

8 Fazit **18**

Literaturverzeichnis **19**

Selbstständigkeitserklärung **19**

1 Einleitung

1.1 Vorstellung des Moduls Vernetzte Systeme

2 Projektablauf

2.1 Vorstellung der Projektidee

2.2 Projektorganisation und Teamarbeit

Für unsere Auseinandersetzung zum Thema ”Analyse verschiedener Dockerkonstellationen” im Rahmen der Veranstaltung ”Vernetzte Systeme” haben wir uns zegielt für einen strukturierten Projektplan, der sich an agilen Methoden orientiert. Zum Entwerfen und Fertigstellung haben wir einen Sprint von drei Wochen gemacht inklusiv das Verfassen vom Bericht. Regelmäßig haben wir uns alle zwei Tage getroffen und da wurde immer über die Fortschritte gesprochen und neue Aufgabe verteilt. Die ersten Tage habe wir erstmal dürber ausgetauscht, was wir machen wollen und wie wir es tun möchten?

Die Implementierung haben wir in mehreren iterativen Phasen unterteilt, wobei das Kanban-System zur Visualisierung und Verwaltung des Arbeitsflusses eingesetzt wurde. Hierfür haben wir die Webanwendung Trello(Verlinkung fehlt) verwendet. In der App haben wir über einen Projekt-Dashboard verfügt, auf dem unsere Aufgaben in drei Kategorien aufgelistet waren wie:

- Zu erledigen
- In Arbeit
- Erledigt

Dies ermöglichte eine flexible Anpassung an wechselnde Anforderungen und eine kontinuierliche Verbesserung der Implementierung. Die agilen Prinzipien unterstützten uns dabei, schnell auf Herausforderungen zu reagieren und eine effektive Implementierung sicherzustellen.

3 Werkzeuge

3.1 Verwendete Werkzeuge

Bei so einem Projekt, da wo es draum geht mit mehreren entfernten Systemen zu arbeiten, braucht man genauso wi ebei anderen Projekten auch Werkzeuge, mit denen man seinen Entwurf

auf die Beine bringen kann.

Hierfür hatten wir schon aus der Vorlesung einige Vorgaben bekommen wie zum Beispiel die Verwendung einer Docker-Umgebung, woran wir uns auch entsprechend eingehalten haben. aber damit unser Entwurf unserer Vorstellung entspricht haben wir auch noch andere Werkzeuge gebraucht, damit wir zum Beispiel unsere mit der Programmiersprache Java entwickelte Mini-Anwendung laufen lassen können, Dateneingaben speichern können, das Einkommen von mehreren Anfragen zu gleicher Zeit verwalten können und auch dann das Verhalten unserer Mini-Anwendung beobachten können. Hier sind unsere Werkzeuge aufgelistet:

- **Git**: Es ist tatsächlich so, dass man oft alleine entwickelt, wenn es sich um solche minimale Anwendungen wie bei uns handelt, doch ist es aber auch nicht ungewöhnlich, dass man mit anderer Leuten kooperieren muss und wenn dies der Fall ist, muss auf irgendeine Art sichergestellt werden, dass alle Leute auf einem gemeinsamen Ordner arbeiten, dessen Inhalt jeder verwalten und auch nach Bedarf zurücksetzen kann. Hierzu ist Git das perfekte Werkzeuge, denn es ermöglicht eine Versionsverwaltung von Dateien, sodass man auch auf einen vergangenen Inhalt zugreifen kann, wenn gemerkt wird, dass etwas schiefgelaufen ist oder dass man etwas gelöscht hat, dass im code von großem Belang war.
- **Apache Tomcat**: Es handelt sich hierbei um eine Open-Source-Implemtierung verschiedener Spezifikationen wie etwa der **jakarta Servlet**, **jakarta Anmerkungen**, **jakarta Authentifizierung** und noch mehr. Es gehört die Apache Software Foundation. Mit einem Tomcat-Server wird möglich eine Anwendung, die ursprünglich in der Programmiersprache Java geschrieben wurde, als Webanwendung laufen zu lassen, was bei uns tatsächlich erfoderlich war.
- **Docker**: Es ist ein revolutionäres Tool, das die Art und Weise, wie Anwendungen entwickelt, ausgeliefert und ausgeführt werden, verändert hat. Es ermöglicht Entwicklern, ihre Anwendungen samt Abhängigkeiten in sogenannten Containern zu verpacken. Diese Containerisierung erleichtert die Bereitstellung und den Betrieb von Anwendungen, da sie auf jedem System laufen können, das Docker unterstützt, unabhängig von der Umgebung. Dies sorgt für Konsistenz über Entwicklung, Test und Produktion hinweg und vermeidet das berüchtigte Äber bei mir lokal funktioniert es" Problem. Für unser Projekt ist Docker essentiell, da es uns ermöglicht, verschiedene Konstellationen mit MariaDB, Redis und anderen Tools effizient zu orchestrieren und zu analysieren.
- **MariaDB**: Dieses Werkzeug muss keiner mehr vorgestellt werden, der sich schon zumindest einmal mit Datenbanken auseinander gesetzt hat. MariaDB ist eine sehr berühmte Open-Source-Datenbank, die als Abspaltung von MySQL entstanden ist, mit dem Ziel, vollständige Open-Source-Freiheit zu gewährleisten. Sie bietet eine hohe Kompatibilität mit MySQL, was bedeutet, dass Anwendungen, die für MySQL geschrieben sind, in der Regel ohne Änderungen mit MariaDB funktionieren. In unserem Projekt nutzen wir

MariaDB zur Speicherung von Daten, die von unserer Servlet-Anwendung verwendet werden.

- **Redis**: Es geht hierbei um ein in-memory Datenstrukturspeicher also eine NoSQL-Datenbank, der als Datenbank, Cache und Message Broker verwendet werden kann. Es unterstützt Datentypen wie Strings, Hashes, Listen, Sets und mehr, was es sehr flexibel macht. Redis ist bekannt für seine hohe Geschwindigkeit und Effizienz bei Operationen mit Daten im Speicher. Bei uns hatten wir die Idee gehabt, dass wir mit Redis eine Warteschlange bauen können. redis kennen wir schon zusammen mit MariaDB und K6 aus dem zweiten Semester, in dem wir eine kurze Einführung in jeden Themen gehabt haben.
- **Tcpdump**: Aus der vorletzten veranstaltung kennengelernt ist Tcpdump ein mächtiges Kommandozeilen-Tool zur Netzwerküberwachung und -analyse. Es ermöglicht es, den Datenverkehr auf einem Netzwerk zu erfassen und zu analysieren, was für die Diagnose von Netzwerkproblemen oder für Sicherheitsanalysen unerlässlich ist. In unserem Projekt verwenden wir tcpdump, um den Datenverkehr zwischen den verschiedenen Containern zu analysieren und zu verstehen, wie unsere Anwendungen kommunizieren und wie sich die Netzwerkleistung auf die Gesamtleistung der Anwendung auswirkt.
- **Ping**: Sehr beghert ist Ping ein einfaches, aber äußerst nützliches Netzwerk-Utility, das verwendet wird, um die Verfügbarkeit und die Latenz (Ping-Zeit) zwischen zwei Netzwerkknoten zu testen. Es sendet ICMP Ëcho Request"Nachrichten an das Ziel und erwartet Ëcho Reply"Nachrichten zurück. In unserem Docker-basierten Setup nutzen wir Ping, um die Netzwerkverbindung zwischen den Containern sowie die Verbindung zur Außenwelt zu überprüfen, was für die Fehlersuche und Leistungsoptimierung unerlässlich ist.
- **K6**: Es ist ein modernes, Open-Source-Performance-Testing-Tool, das für Entwickler konzipiert wurde, um die Leistung und Skalierbarkeit von Anwendungen in einer DevOps-Umgebung zu testen. Es ermöglicht das Schreiben von Tests in JavaScript, was die Erstellung realistischer Test-Szenarien vereinfacht. In unserem Projekt verwenden wir K6, um die Belastbarkeit und Performance unserer Dockerkonstellationen unter verschiedenen Bedingungen zu testen, um sicherzustellen, dass unsere Anwendungen auch unter Last zuverlässig und effizient laufen. Effektiv wurde K6 für die Testphase bei uns verwendet. Mehr dazu unten bei Lasttest.

Mit diesen Werkzeugen, die uns entweder sehr vertraut oder mit denen wir noch einstiegen, haben wir unsere eigene Wege gefunden, wie die mit einander kombiniert werden, um einsere Projektidee umsetzen zu können und Erfolg zu erzielen.

4 Docker

Die Verwaltung komplexer Datenbanken, Programmiersprachen, Frameworks und anderer Komponenten beim Erstellen von Anwendungen kann eine Herausforderung darstellen. Es besteht auch die Gefahr von Kompatibilitätsproblemen, insbesondere bei der Arbeit mit verschiedenen Betriebssystemen. Diese Faktoren können die Effizienz und den Erfolg Ihrer Arbeit beeinträchtigen. Glücklicherweise bietet Docker eine elegante Lösung für diese Probleme. Durch die Nutzung von Docker können Anwendungen in Container-Umgebungen erstellt und verwaltet werden. Dies ermöglicht nicht nur eine saubere Trennung von Ressourcen, sondern erleichtert auch die Portabilität zwischen verschiedenen Umgebungen erheblich. Docker vereinfacht die Entwicklung und macht sie effektiver, indem es viele zeitaufwändige Konfigurationsaufgaben automatisiert. Als Open-Source-Plattform bietet es eine flexible Lösung für Entwickler, um Anwendungen in einer sicheren Sandbox zu entwickeln. In einer Welt, in der Effizienz und Skalierbarkeit von größter Bedeutung sind, ist Docker ein unverzichtbares Werkzeug für Entwickler. Durch die Nutzung von Containertechnologie können Sie sich auf das konzentrieren, was wirklich zählt: die Entwicklung großartiger Anwendungen.

lol

4.1 Was ist ein Docker?

Der Begriff „Docker”wird vielfältig verwendet und kann sich auf das Open-Source-Community-Projekt, Tools aus dem Open Source-Projekt oder direkt auf das Unternehmen Docker Inc. beziehen

Docker ist eine revolutionäre Open-Source-Plattform, die die Art und Weise verändert hat, wie wir Anwendungen entwickeln, testen und bereitstellen. Sie bietet eine Sandbox-Umgebung, in der Entwickler Anwendungen in Containern erstellen können ,leichte, virtualisierte Umgebungen, die eine effiziente Nutzung von Ressourcen ermöglichen. Obwohl Container als Konzept schon seit 1979 existieren, hat Docker den Zugang zu ihnen drastisch vereinfacht. Durch Docker können Entwickler Anwendungen lokal oder auf Produktionsservern erstellen, testen und implementieren, ohne sich um Kompatibilitätsprobleme oder Abhängigkeiten zu sorgen. Seit der Veröffentlichung von Docker 1.0 im Jahr 2014 hat sich die Nutzung von Containern als Standardpraxis für Einzelpersonen und Unternehmen etabliert. Große Unternehmen wie Netflix, Target und Adobe setzen Docker erfolgreich ein, und die Plattform verzeichnet mittlerweile mehr als 13 Millionen Nutzer weltweit. Docker hat die Art und Weise, wie wir Software entwickeln und bereitstellen, revolutioniert und ermöglicht es Entwicklern, effizienter und agiler zu arbeiten. Mit Docker können Anwendungen schnell und konsistent zwischen verschiedenen Umgebungen bereitgestellt werden, was zu einer beschleunigten Entwicklungszeit und einer verbesserten Skalierbarkeit führt.

4.1.1 Wie funktioniert Docker ?

Die Docker-Technologie verwendet den Linux-Kernel und seine Funktionen, einschließlich Cgroups und Namespaces, um Prozesse zu unterscheiden, damit sie separat ausgeführt werden können. Der Zweck des Containers ist die Unabhängigkeit – die Fähigkeit, mehrere Prozesse und Apps unabhängig voneinander auszuführen. Dadurch wird die Nutzung Ihrer Infrastruktur optimiert und gleichzeitig die Sicherheit gewährleistet, die durch die Arbeit mit verschiedenen Systemen entsteht. Container-Tools wie Docker verwenden ein imagebasiertes Deployment-Modell. Dies macht es einfacher, eine Anwendung oder eine Reihe von Services mit allen Abhängigkeiten in verschiedenen Umgebungen zu nutzen. Darüber hinaus automatisiert Docker das Deployment der Anwendung (oder einer Kombination von Prozessen, aus denen eine Anwendung besteht) innerhalb dieser Containerumgebung. Diese Programme verwenden Linux-Container, was Docker unglaublich einfach und einzigartig macht. Wie funktioniert Docker ?

Die Docker-Technologie verwendet den Linux-Kernel und seine Funktionen, einschließlich Cgroups und Namespaces, um Prozesse zu unterscheiden, damit sie separat ausgeführt werden können. Der Zweck des Containers ist die Unabhängigkeit – die Fähigkeit, mehrere Prozesse und Apps unabhängig voneinander auszuführen. Dadurch wird die Nutzung Ihrer Infrastruktur optimiert und gleichzeitig die Sicherheit gewährleistet, die durch die Arbeit mit verschiedenen Systemen entsteht. Container-Tools wie Docker verwenden ein imagebasiertes Deployment-Modell. Dies macht es einfacher, eine Anwendung oder eine Reihe von Services mit allen Abhängigkeiten in verschiedenen Umgebungen zu nutzen. Darüber hinaus automatisiert Docker das Deployment der Anwendung (oder einer Kombination von Prozessen, aus denen eine Anwendung besteht) innerhalb dieser Containerumgebung. Diese Programme verwenden Linux-Container, was Docker unglaublich einfach und einzigartig macht.

4.2 Was ist ein Netzwerk?

Ein Netzwerk in Docker ist eine virtuelle Umgebung, die es Containern ermöglicht, miteinander zu kommunizieren. Es ermöglicht die Verbindung von Containern und ermöglicht es ihnen, Ressourcen und Informationen auszutauschen. Netzwerke in Docker können verwendet werden, um verschiedene Container miteinander zu verbinden und sie in einer isolierten Umgebung zu betreiben.

4.3 Was ist ein Image?

Images sind schreibgeschützte Templates mit Anweisungen zum Erstellen eines Containers. Ein docker Image erstellt Container zur Ausführung auf der Docker-Plattform. ein Docker-Image

führt Code in einem Docker-Container aus und es ist möglich mehrere Container aus demselben Image zu erstellen.

4.3.1 Wie erstellt man ein Docker-Image ?

Um ein Docker-Image zu erstellen, verwendet man den `docker image build` Befehl zusammen mit weiteren Optionen zur Spezifikation des Build-Kontexts und des Tags des Images. Zum Beispiel:

```
docker image build --progress=plain -t image-fancy context
```

Dieser Befehl erstellt ein Image mit dem Namen `image-fancy` aus dem Build-Kontext `context`, wobei `--progress=plain` eine einfachere Ausgabe während des Build-Prozesses erzeugt.

4.4 Was ist ein Container ?

Ein Container ist eine Abstraktion aud fer App-Ebene, die Code und Abhängigkeiten zusammenfasst. Mehrere Container könnenauf demselben Computer ausgeführt werden und den Betriebssystemkernel mit anderen Containern teilen, die jeweils als isolierte Prozesse im Benutzerbereich ausgeführt weden. Container nehmen weniger Platz ein als VMs und können mehr Anwendungen verarbeiten.

4.4.1 Wie erstellt und startet man einen Container?

Ein Container wird basierend auf einem Docker-Image erstellt und gestartet. Das folgende Beispiel zeigt, wie ein Container mit dem Namen `services` erstellt und konfiguriert wird:

```
docker container create \
    --name "services" \
    --hostname "services" \
    --network mynet \
    --volume "$PWD/www/:/var/www/html" \
    --volume "$PWD/log/:/log" \
    --publish 80:80 \
    --publish 8080:8080 \
    "image-fancy"
```

Dieses Kommando erstellt einen Container, der zum `mynet` Netzwerk gehört, Volumes für Webinhalte und Logdateien einbindet und Ports 80 sowie 8080 veröffentlicht.

Um einen Docker-Container zu starten, verwendet man den Befehl `docker container start` gefolgt vom Namen oder der ID des Containers, den man starten möchte. Dieser Befehl ändert den Zustand des Containers von gestoppt zu laufend.

Wir haben einen Docker-Container mit dem Namen `services` erstellt. Um diesen Container zu starten, der Befehl dazu lautet:

```
docker container start services
```

Dieser Befehl sucht nach einem Docker-Container, der `services` heißt, und startet ihn. Sobald der Container gestartet ist, läuft er im Hintergrund weiter, und die in ihm enthaltene Anwendung beginnt mit ihrer Ausführung basierend auf den beim Erstellen des Containers festgelegten Konfigurationen und Einstellungen.

4.4.2 Überprüfung des Container-Status

Um zu überprüfen, ob der Container erfolgreich gestartet wurde und läuft, kann man den Befehl `docker container ls` verwenden, der alle laufenden Container auflistet. Wenn Sie auch gestoppte Container sehen möchten, können Sie `docker container ls --all` hinzufügen, um eine vollständige Liste zu erhalten.

4.5 Wie stoppt und entfernt man einen Container?

Zum Stoppen und Entfernen eines Containers verwendet man die Befehle `docker container stop` und `docker container rm`. Das folgende Skript demonstriert diesen Prozess für einen Container namens `services`:

```
if docker container ls | grep -q "\bservices$"; then
    docker container stop services
fi

if docker container ls --all | grep -q "\bservices$"; then
    docker container rm services
fi
```

Zusätzlich entfernt das Skript veraltete SSH-Konfigurationen für den Container aus *~*`.ssh/docker_config`.

4.6 Was sind die Vor - und Nachteile von Docker ?

4.6.1 Vorteile

Docker bietet eine Vielzahl von Vorteilen, die es zu einem beliebten Werkzeug in der Softwareentwicklung und im IT-Betrieb gemacht haben:

- **Portabilität:** Einmal erstellte Docker-Images können über verschiedene Umgebungen hinweg ohne Änderungen ausgeführt werden. Dies erleichtert die Bereitstellung und Migration von Anwendungen erheblich.
- **Konsistenz:** Docker-Container bieten eine konsistente Umgebung für die Anwendung, unabhängig davon, wo der Container ausgeführt wird. Dies reduziert das Problem ës funktioniert auf meinem Rechner nicht", indem Diskrepanzen zwischen Entwicklung, Test und Produktion minimiert werden.
- **Isolation:** Container sind voneinander isoliert und verpacken ihre eigene Software, Bibliotheken und Konfigurationen. Dies verbessert die Sicherheit und vereinfacht Abhängigkeitsmanagement.
- **Ressourceneffizienz:** Docker nutzt die Ressourcen des Host-Betriebssystems effizienter als traditionelle Virtualisierungstechnologien, was zu einer besseren Hardwareauslastung führt.
- **Schnelle Bereitstellung:** Container können innerhalb von Sekunden gestartet und gestoppt werden, was die Bereitstellungszeiten verkürzt und eine schnelle Skalierung ermöglicht.
- **Entwicklungseffizienz:** Docker unterstützt Entwickler mit der Möglichkeit, lokale Umgebungen schnell einzurichten, was die Entwicklungszyklen beschleunigt.
- **Mikroservice-Architekturen:** Docker eignet sich hervorragend für die Entwicklung und den Betrieb von mikroservice-basierten Anwendungen, indem es die Unabhängigkeit und Isolation der Services fördert.

4.6.2 Nachteile

Trotz seiner vielen Vorteile hat Docker auch einige Nachteile, die berücksichtigt werden sollten:

- **Sicherheitsrisiken:** Die Isolation zwischen Containern ist nicht so stark wie bei vollständig virtualisierten Umgebungen, was potenzielle Sicherheitsrisiken birgt. Es ist wichtig, Sicherheitspraktiken wie das Prinzip der geringsten Privilegien und regelmäßige Updates zu befolgen.

- **Komplexität:** Die Verwaltung von Docker-Containern und -Orchestrierung, insbesondere in großem Maßstab, kann komplex sein und erfordert ein gutes Verständnis von Docker-Netzwerken, Volumes und Sicherheitskonfigurationen.
- **Speicher- und Datenmanagement:** Persistente Daten und deren Verwaltung können in containerisierten Anwendungen eine Herausforderung darstellen, besonders wenn es um Zustandsbehaftete Dienste wie Datenbanken geht.
- **Plattformabhängigkeiten:** Obwohl Docker auf Portabilität ausgelegt ist, können bestimmte Images aufgrund von Systemabhängigkeiten (z.B. Kernel-Versionen) nicht auf allen Plattformen ausgeführt werden.
- **Performance-Überlegungen:** Während Container im Allgemeinen effizient sind, können spezifische Workloads oder Konfigurationen in Containern im Vergleich zu nativen Ausführungen oder anderen Virtualisierungstechnologien Leistungseinbußen erleiden.
- **Lernkurve:** Die umfangreichen Funktionen und das Ökosystem von Docker können für Neueinsteiger überwältigend sein und erfordern Zeit und Ressourcen, um effektiv genutzt zu werden.

4.7 Wie haben wir Docker verwendet?

In unseren Skripten haben wir Docker verwendet, um eine isolierte und konsistente Umgebung für unsere Anwendungen zu schaffen. Die Skripte automatisieren den Prozess des Builds, der Bereitstellung und des Managements von Containern. Hier einige Beispiele:

- **Image-Erstellung:** Wir nutzen den `docker image build` Befehl, um Docker-Images aus unseren Anwendungsquellen zu erstellen. Dies ermöglicht eine einheitliche und reproduzierbare Umgebung für die Entwicklung und das Deployment unserer Anwendung.
- **Container-Management:** Unsere Skripte automatisieren das Erstellen, Starten, Stoppen und Entfernen von Containern. Dies vereinfacht die Verwaltung der Lebenszyklen unserer Anwendungen erheblich.
- **Netzwerk und Volumen:** Wir konfigurieren Netzwerke und binden Volumes ein, um die Kommunikation zwischen Containern zu ermöglichen und persistente Daten zu verwalten. Dies ist entscheidend für komplexe Anwendungen, die aus mehreren Diensten bestehen und Zustandsdaten speichern müssen.

5 Ergebnis

Bei dieser Ausarbeitung haben wir sehr viel Wert darauf gelegt, eine fundierte Analyse anhand unsere Gelernten aus der Vorlesung abzugeben. AUs diesem Grund haben wir nicht nur irgendeinen Code aus der Vorlesung kopiert und eingefügt, sondern, wir haben das Wissen aus der Vorlesung zu unserem Nutzen gemacht und daraus erfolgten verschiedene Analysen mit einem und zwei und noch mehr Containern, damit wir effektiv zeigen können, wie eine Dockerumgebung verwendet werden kann.

5.1 Analyse mit zwei laufenden Containern

Basierend auf unsere Idee, ein Java-Servlet mit verschiedenen Dockerkonstellationen laufen zu lassen, haben wir verschiedenen Sachen implementiert. Bei der ersten sowie bei den anderen Anaylse stand im Vordergrund die Vorbereitung der Arbeitsumgebung. Wir mussten uns eine geeignete Dockerumgebung erstellen. Hierfür haben wir zuerst die vorbereitete Umgebung aus der Vorlesung verwendet, aber vorher haben wir die noch ein wenig angepasst, sodass es unsere Vorstellungen entsprechen kann.

Der Ordner, den wir hierfür ausgewählt haben, war

```
/docker-fancy-tomcat-split-sql
```

, weil wir gefunden haben, dass er leichter wäre noch mehrfach aufzuteilen. Für dieses erste Experiment waren die Docker nur in zwei Container jeweils `work und services`, wobei **work** unsere Arbeitsumgebung darstellt. Dort wird nur gearbeitet und im Anschluss daran später zum Container **services** deployt. Im Container **servcies** laufen im Gegenteil zu **work** all unsere Dienste, wie zum Beispiel: `Redis, MariaDB, Tomcat, ein ssh-Server und ein Apache-Server`. Beide Container werden durch einen ssh-Server verbunden und es ist so eingerichtet worde, dass man sich von einem Container zu einem anderen passwortlos bewegen kann.

Der Grund, warum wir nicht mit einem Container gearbeitet haben ist leicht zu erklären. Es liegt einfach daran, dass wir vom Vorteil solch einer Umgebung Gebrauch machen wollten, und zwar die Isolation. Die Prozesse auf eine einem Docker finden nämlich isoliert statt, was zu verbesserten Leistungen führt. Noch wichtig zu der Isolation wäre die erhöhte Flexibilität. Nehmen wir als Beispiel einen Fall, den wir bei uns hatten mit einem nicht aktuellen Container hatten. In so einer Situation ließ sich dieser mit wenig Aufwand durch einen aktuellen Container, der unserer Anforderungen entsprach, tauschen.

5.1.1 Aufbau der Image: `fancy`

In einer Dockerumgebung laufen die Container auf einer sogenannten Image, die vorabgebaut werden muss. Was eine Image ist, wird hier nicht mehr erläutert, hierfür steht die Beschreibung oben als Referenz. Unsere Image haben wir wie seit dem ersten Semester gelernt automatisert aufgebaut in einem shell-Skript names `/build.sh`. Unser Sckript sieht dann folgendermaßen aus:

```
1 #!/usr/bin/env bash
2 . local/config.txt
3 bin/download-tomcat.sh
4 bin/download-libs.sh
5 docker image build --progress=plain -t image-fancy context
```

: build.sh

5.1.2 Aufbau unseres lokalen Netzwerkes: mynet

Das Arbeiten mit verteilten Container erfordert einen regelmäßigen Austausch zwischen denen, da der einer Container einen Dienst brauchen könnte, der nur auf dem anderen verfügbar sein kann. Daher ist die Verwendung einer benutzerdefinierten Netzwerkumgebung innerhalb von Docker durchaus notwendig, um eine isolierte Netzwerkumgebung für unsere Docker-COntainer zu schaffen. Dies haben wir uns auch in dieser Ausarbeitung erarbeitet. Hierfür haben wir ein bash-Skript namens `create-mynett-network.sh` erstellt. In diesem bash-Skript ist nur eine einzige Befehlzeile vorhanden:

```
1 #!/bin/bash
2 docker network create mynet --subnet 172.27.0.0/16
3
```

: create-mynet-network.sh

Mit diesem Befelhzeile wird ein neues Netwerk erstellt, dessen Name `mynet` ist. Dazu fügen wird eine optionale Konfiguration hinzugefügt mit `--subnet 172.27.0.0/16`. Diese definiert das Subnetz des Netzwerkes. In diesem Fall legt die definierte IP-Adresse das Subnetz fest, das alle IP-Adressen von 172.27.0.1 bis 172.27.255.254 umfasst und mit `/16` als Subnetzmaske angibt, die bestimmt welcher Teil der IP-Adresse das Netzwerk und welcher Teil die spezifischen Hots innerhalb dieses Netzwerks darstellt.

abgesehen vom Vorteil der Isolation, von dem wir mehjr genug in dieser Ausarbeitung geschrieben haben, bietet das Erstellen eines lokalen Netzes für verteilte Docker-Container erheblich viele Vorteile an. Hier kann die benutzerdefinierte IP-Adressierung angesprochhen werden, wodurch die Konfiguration von Diensten und die Kommunikation zwischen Containern erleichtert wird, da wir schon wissen, in welchem IP-Bereich sich die Container befinden.

Erwähnen können wir noch die vereinfachte Dienstentdeckung, die von Docker-Container unterstützt werden, was bedeutet, dass Container sich gegenseitig über den Container-Namen (der als Hostname fungiert) finden und kommunizieren können. Dadurch wird die Konfiguration von Anwendungen, die über mehrere Conatiner verteilt sind, erheblich vereinfacht.

Ein letzter Vorteil liegt bei der Performance bzw. Leistungsfähigkeit. Wenn wir unseren Netzwerkverkehr in einem isolierten Netzwerks halten, können wir potenziell die Netzwerkperformance verbessern, da der Verkehr nicht durch den Haupt-Docker-Daemon geleitet wrden muss. Es geschieht alles lokal und es muss nichts geladen werden.

5.1.3 Aufbau der Container: `work und service`

Wie oben erwähnt haben wir uns in diesem ersten Fall mit zwei laufenden Container auseinandergesetzt. und zwar `work` und `service`. Betrachten wir nun mal den Aufbau dieser beiden Container.

Wir beginnen erstmal mit `work`. In diesem Container wurde hauptsächlich gearbeitet, aus diesem Grund ist seine Struktur nur grundlegend und enthält nur einen ssh-Dienst, der beim Starten vom Container ebenfalls gestartet wird. Hierfü stand schon das bash-Skript `myinit-work.sh` zur Verfügung. dieser Skript ist dafür zuständig einen User anzulegen und den ssh-Dienst zu starten. natürlich gibt es auch einen Aschnitt bzw eine Funktion im Skript, der/die sicherstellt, dass der gestartete Dienst auch entsprechend gestoppt, wird, wenn der Container beendet wird.

```
1  # start services
2  {
3    /startup/create-user.sh
4    #pw=$(grep "^containerpassword=" /startup/config.txt|cut -d'=' -f2)
5    service ssh start
6    #rm /startup/*
7    echo ready >> /log/state-work.txt
8  } >>$log
9
```

Listing 5.1: Starten von Diensten in **work**

Zum Starten vom Container selbst wird ein anderes Skript names `start-work.sh` verwendet. Dieses Skript sorgt einfach dafür, dass der work-Container gestartet wird, aber mit zusätzlichen Optionen, wie zum Beispiel, dass es in einem lokal erstellten Netzwerk läuft, vorbereitetete Konfigurationen übernimmt, die sich schon vorab in dem Verzeichnis Startup in `context/` befinden, und auch zu vorhandenen bekannten Hots hinzugefügt wird. Dadurch wird es möglich sich in diesem Container per ssh zu bewegen, statt auf der Kommandozeile

```
docker container exec -ti work bash
```

eingeben zu müssen. Zu den wichtigsten Sachen die work beim Starten kopiert werden, zählen wir, Java-Programme, die Teil unseres Servelt sind.

```
1 scp -qr javawebdemo $containername:
2 scp -qr brotundbutter-swe $containername:
3 scp -qr unittester $containername:
```

Da Work nur der Arbeitsbereich ist und dort nur gecode wird, ist das tatsächlich so, dass man zum Ausführen von Programmen, die zu dem anderen Container deployt. dort laufen alle notwendige Dienste zur korrekten Ausführung der Programme. Die werden ebenfalls in von einem Skript namens `myinit-services.sh` gestartet und beenden. Es gibt unter anderem solche Dienst wie Redis, Tomcat, MariaDB und einen SSH-Dienst.

5.1.4 Lasttest mit zwei Containern

Nach dem unser Servlet bereitgestellt und in den entsprechenden Containern deployt wurde. haben wir zur Vervollständigung der Analyse mit einer testphase angefangen. Hierbei wollten, wir das Verhalten unserer Dockerumgebung bei hoher anfrageaufkommen analysieren und Erkenntnisse daraus ziehen.

Zum Durchführen von Lasttest wird eine geeignete Anwendung benötigt, die die Aufgabe erfüllen kann, vermehrte Anfragen an eine gleiche Adresse zu schicken, damit wir die Leistung von unserem System unter unterschiedlichen Arbeitslasten zu prüfen. Dies würde uns dann die Möglichkeit geben, herauszufinden, wo die Schwächen und Grenzen in unserem System liegen. So ein test inst nicht nur im Rahmen dieser Veranstaltung wichtig, sondern wir werden es auch kpnftig als Softwareentwickler oder -tester im Arbeitsumfeld brauchen. heutzutage gilt in der Geschäftswelt - Zeit ist geld - besonders noch in der IT-Welt als in der realen Welt. Deshalb ist es wichtig zu gewährleisten, dass Systeme unabhängig von fauer und Nutzungslast problemlos arbeiten können. Das haben wir uns in dieser ersten Ananlyse auch gewidmet.

Das Werkzeug, dass wir nach Überlegungen und vergleich ausgwählt haben, ist **Grafana k6**. Es ist ein Open-Source-Tool für Lasttest, .it dem Leistungstets für Engineering-teams einfach und produktiv sind. als Open-Source-Tool ist es auch natürlich kostenfrei. K6 haben wir zwischen all den anderen Tools wie:

- curl
- wrk
- ab

weil es einfacher ist zu bedienen (zwar nicht leichter als curl, aber leistungsfähiger) und kann eine viel höhere Last setzen. Jeder test bietet mti k6 eine Visualisierung der testmetriken. Zudem können die Ergebnisse davon in andere Dashboards übertargen werden, um tiefergehende Analyse zu ermöglichen. K6 wird mit JavaScript implemtiert, um Anfragen über HTTP-Methoden zu senden. Es wird hierfür eien JavaScript-Datei (bei uns `script.js`) erstellt. Unsere sieht folgendermaßen aus:

```
1 import http from 'k6/http';
2 export const options = {
3   noCookiesReset: true,
4 }
5
6 export default function () {
7   let jar = http.cookieJar();
8   const res = http.get('http://192.168.0.177:8080/sql');
9 }
```

: script.js

5.1.5 Hoffnung und Erwartungen

Im Laufe des Semesters konnten wir durch unsere Teilnahme an den Veranstaltungen sehr viele Ideen zum Testen haben. Das haben wir versucht in diesem projekt wiederzuspiegeln. Darauf haben wir stets abgezielt, die beigebrachten Konzepten anzuwenden. Wir wollten zum Beispiel Networking anwenden, was ein Zusammenschluss von zwei oder mehr Computern oder anderen, elektronischen Geräten, der den Austausch von Daten und die Nutzung gemeinsamer Ressourcen ermöglicht **ionos**. Wenn bei uns jeder Container einzeln als Computer betrachtet wird, können wir auch von Networking reden. In einem Netz hat jeder Computer eine eigene IP-Adresse, mit der er über das Netz angesprochen werden kann. Dies haben wir uns bei der testphase zur Nutze gemacht, um wiederholt Anfragen an unseren Programmen zu senden.

Bei dem Thema IP-Adresse spricht man von Adressierung. Eine Sache, mit der wir uns auch im Semester beschäftigt haben und durch die vielen geteilten Videos gelernt haben. Dabei wurde immer hauptsächlich von dem TCP/IP (aus dem Englischen: Transmission Control Protocol/ Internet Protocol) gesprochen, dessen Grundlagen wir in unserem Projekt haben. Es ist ein Konzept, das die Identifizierung von en einem Netzwerk teilnehmenden Rechnern über IP-Adresse ermöglicht. Die Funktionsweise basiert zum Beispiel im OSI-Model auf 3- bis 4-Schichten-Protokolle. Die Kommunikation erfolgt wie in den letzten Veranstaltungen gezeigt in Form von Datenpakette, die gesendet und auch auf der anderen Seite empfängt werden. Es bestht zur Gewährleistung der Sicherheit eine Art Codierung in der Kommunikation zwischen den zwei austauschenden Geräten, sodass der einer eine bestimmte Anzahl an Byte sendet und der Empfänger als Antwort auch eine entsprechende Anzahl an Byte mit dem korrekten Anfang und die korrekte Folge zurücksendet, damit der Absender erstmal weiß, dass der Empfänger die anchricht bzw. das Datenpakett empfangen hat und davon die Antwort schon vorhanden ist. Dieses Prinzip ist auch bei uns vorhanden uns wird in einer späteren Analyse gezeigt.
Noch wärhend der Veranstaltung wurden wir in die POSIX-Standard eingeführt. Was wir dazu wissen ist, POSIX heißt (aus dem Englischen) Portable Operating System Interface und ist eine gemeinsam von IEEE und der Open Group für Unix entwickelte standardisierte Programmierschnittstelle, welche die Schnittstelle zwischen Anwendungssoftware und Betriebssystem darstellt. Es ist zu einer Norm geworden, die bei allen Unic-Betriebssystemen verwendet wird. Dabei ist ein betriebssystem wie Windows ausgeschlossen, denn dieses System hat seine eigene Standart. Früher hat jedoch Microsoft versucht, sich an diese Standards zu halten, sind aber später davon rausgegangen. Wir haben uns diese Standards eingeprägt und auch in unserer Implementierung mit einbezogen.

5.2 Analyse mit drei laufenden Containern

5.3 Analyse mit vier laufenden Containern

6 Lasttest

6.1 Was haben wir analysiert?

6.2 Wie haben wir das analysiert?

6.3 Endergebnis

7 Reflexion

8 Fazit

Selbstständigkeitserklärung

Wir versichern, die von uns vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, haben wir als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die wir für die Arbeit benutzt haben, sind angegeben. Die Arbeit haben wir mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegt.
Bremerhaven, den 28. Februar 2024 Unterschrift: