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: 21. Februar 2024

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

Inhaltsverzeichnis

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

4.1 Was ist ein Docker?

4.2 Was ist ein Netzwerk?

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`.

4.5.1 Wie stoppt man einen Container ?

Um einen Container zu stoppen, verwendet man den Befehl `$ docker stop` oder `$ docker kill`

Beispiel: `$ docker container stop hello-word`

`$ docker kill hello-world` , damit wird der Container sofort gestoppt. Damit wird ein SIGKILL gesendet und dabei das SIGTERM übersprungen.

Es ist auch möglich alle laufenden Container zu stoppen, indem man folgenden Befehl benutzt: docker stop `$(docker ps -q)`

q Flag zeigt nur die numerischen IDs an

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

5.1 Analyse mit zwei laufenden Containern

5.2 Analyse mit drei laufenden Containern

5.3 Analyse mti 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 21. Februar 2024 Unterschrift:

Junior Ekane

Florian Quaas

Steve Aguiwo

Herman Tsago