Dokumentation:

Modul 347 Dienst mit Container anwenden

Lernziele:

Sie können die Begriffe "Containerisierung", "Image", "Layer", "Container", "Repository", "Registry" und “Dockerfile” erklären

Containerisierung:

Containerisierung bezeichnet die Virtualisierung mit Hilfe von Containern. Eine Software wird mit all ihren benötigten Komponenten in einen Container verpackt, um sie in verschiedenen Umgebungen ausführen zu können, wodurch der Container eine tragbare Rechenumgebung darstellt.

Image:

Ein Image ist eine Datei, die das Betriebssystem, Anwendungen und alle zugehörigen Daten enthält und als Vorlage für das Erstellen und Bereitstellen von virtuellen Maschinen oder Containern verwendet wird.

Layer:

Teil eines Images. Enthält einen Befehl oder eine Datei, die dem Image hinzugefügt wurde.

Container:

Ein Container ist eine isolierte, standardisierte Softwareumgebung, die Anwendungen und deren Abhängigkeiten kapselt und sie portabel und konsistent über verschiedene IT-Infrastrukturen hinweg macht.

Repository: Satz gleichnamiger Images mit verschiedenen Tags, zumeist Versionen.

Registry:

Verwaltet Repositories. z.B. DockerHub

Dockerfile: Textdatei mit allen Befehlen, um ein Image zusammenzustellen.

Abfolge:

- Containerisierung

- Image
  - Layer
- Container
- Repository
  - Registry
- Dockerfile

Sie können die Begriffe Virtualisierung und Containerisierung voneinander trennen

Virtualisierung ermöglicht die Erstellung virtueller Instanzen von Computern oder Betriebssystemen, während Containerisierung sich auf die isolierte Ausführung von Anwendungen und ihren Abhängigkeiten in standardisierten Umgebungen konzentriert.

 Untitled

Während in jeder virtuellen Maschine ein vollständiges Betriebssystem läuft, bringt ein Container nur die wirklich benötigten Komponenten ohne Betriebssystem mit. Die Container verwenden alle denselben Kernel, nämlich denjenigen des Host-Rechners. Dies spart eine Menge Ressourcen und es können daher sehr viel mehr Container auf einem Host laufen als klassische virtuelle Maschinen

Sie kenne die elementaren Commands, um Images und Container zu erzeugen, starten, beenden und löschen

 Untitled

docker pull¶

Zweck: Ein Image in einem Repository wird auf den lokalen Rechner heruntergeladen.

Beispiel: `docker pull nginx`

docker stop¶

Zweck: Ein laufender Container wird gestoppt. Der Container wird nicht gelöscht.

Beispiel: `docker stop my-nginx-container`

docker start¶

Zweck: Ein vorhandener gestoppter Container wird wieder gestartet.

Beispiel: `docker start my-nginx-container`

docker rm¶

Zweck: Ein Container wird gelöscht. Dazu muss er gestoppt sein.

Beispiel: `docker rm my-nginx-container`

docker rmi¶

Zweck: Ein Image auf dem Host wird gelöscht. Es dürfen keine abgeleiteten Container von diesem Image vorhanden sein (weder laufend noch gestoppt)

Beispiel: `docker rmi nginx`

docker ps¶

Zweck: Zeigt alle laufenden Container an, wie der command ls

Beispiele:

`docker ps` Zeigt alle laufenden Container an.

`docker ps -a` Zeigt alle Container an (laufende und gestoppte).

docker images¶

Zweck: Zeigt alle auf dem Host vorhandenen Images an.

Beispiel: `docker images` oder `docker image ls`

docker exec¶

Zweck: Ruft ein Command auf einem laufenden Container auf.

Beispiel:

`docker run --name mycontainer -d -i -t alpine /bin/sh docker exec -it mycontainer /bin/sh`

docker build Zweck: Aus einem Dockerfile wird ein Image auf dem Host zusammengestellt (siehe Kapitel 3.2) **Beispiel**: `docker build -t my-example-image .`

Sie können die wichtigsten Optionen von docker run (--name, --rm, --network, --ip, -d, -it, -p, -v, -e) korrekt anwenden

docker run¶

Zweck: Aus einem Image wird ein Container gestartet. Befindet sich das Image nicht auf dem Host, wird zunächst automatisch ein docker pull ausgeführt.

Optionen (Auszug):

Option	Beschreibung
--detach, -d	Führt den Container im Hintergrund aus. Gibt beim Start die Container-ID in der Konsole aus.
--interactive, -i	Lässt STDIN (Standard Input) geöffnet, auch wenn der Container im Hintergrund ausgeführt wird.
--name	Weist dem Container einen Namen zu. Per Default wird ein zufälliger Name zugewiesen.
--publish, -p	Veröffentlicht den Port eines Containers für den Host z.B. -p 80:8080 mappt Container-Port 8080 zu Host-Port 80.
--tty, -t	Ordnet eine Pseudo-TTY (Pseudo-Terminal) zu.
--rm	entfernt den Container bei Programmende.

Vollständige Terminalfunktionalität

Um die vollständige Terminalfunktionalität zu aktivieren, werden die beiden Option -i -t normalerweise zusammen übergeben. Sie können auch als **-it** zusammengefasst werden.

Beispiel: `docker run -d --name my-nginx-container -p 8080:80 nginx`

Sie können verschiedene Versionen (tags) eines Container-Images nutzen

Der "Tag" in `<image_name>:<tag>` ist eine Bezeichnung für eine bestimmte Version des Container-Images.

Der "Tag" in `<image_name>:<tag>` ist eine Bezeichnung für eine bestimmte Version des Container-Images. Typischerweise kann der Tag eine Versionsnummer, einen Commit-Hash, eine Codename oder eine beliebige andere Zeichenfolge sein, die verwendet wird, um die Version des Images zu identifizieren.

Hier sind einige Beispiele für Tags:

- `latest`: Bezieht sich auf die neueste Version des Images.
- `v1.0`, `v2.0`, usw.: Versionsnummern des Images.
- `commit-hash`: Der Hash-Wert des Git-Commits, auf dem das Image basiert.
- `alpha`, `beta`, `stable`: Codenames, um den Entwicklungsstatus des Images anzuzeigen.

Beispiel:

docker run ubuntu:latest

Sie können Portweiterleitungen in Betrieb nehmen.

Stellt ein Container einen Serverdienst über einen bestimmten Port zur Verfügung kann dieser mit einem anderen Port (oder auch dem gleichen) Port auf dem Host verknüpft werden.

Läuft beispielsweise im Container eine Webanwendung auf Port 80 (http), läss sich dieser Port beim Start des Containers mit dem Parameter -p mit dem Port 80 des Hostrechners so verbinden

```
docker run -p 80:80 <image>
```

Die Syntax für -p lautet

```
p hostport:containerport
```

Die erste Zahl bezieht sich also auf die Portnummer des Hostrechners und die zweite der Portnummer im Container drin. Die Syntax folgt der in Docker üblichen Reihenfolge auch bei anderen Zuordnungen, immer zuerst den Host und dann den Container anzugeben.

Sollte der Port 80 des Hostrechners im obigen Beispeil bereits belegt sein, weil z.B. der Host selber ein Webserver ist und damit Port 80 verwendet, kann man einfach eine andere Portnummer verwenden. Für Webdienste üblich ist z.B. Port 8080:

`docker run -p 8080:80 <image>`

Die Containerwebseite ist dann unter `http://localhost:8080` erreichbar.

Sie können benannte und gemountete Volumen korrekt einsetzen.

Named Volumes: Diese werden in Docker verwendet, um persistente Daten zu speichern, die von Containern genutzt werden. Sie bieten eine Möglichkeit, Daten außerhalb des Containers zu speichern. Das bedeutet, dass die Daten erhalten bleiben, selbst wenn der Container gelöscht oder neu erstellt wird. Named Volumes sind nützlich für den Datenaustausch zwischen Containern und sie erleichtern die Datenverwaltung in einer Docker-basierten Umgebung.

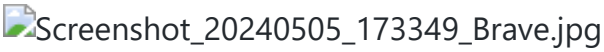
`docker run -d --name irgendEnName -v IrgendEhVerzeichnis:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=geheim mariadb` ; `(-v volumename:containerverzeichnis)`

Mounted Volumes:

Gemountete Volumes in Docker ermöglichen den Datenaustausch zwischen Host-Betriebssystem und Container. Sie hängen ein Verzeichnis oder eine Datei auf dem Host in den Container ein und sind ideal für das Teilen von Konfigurations- und Logdateien oder die persistente Speicherung. Im Gegensatz zu Named Volumes werden sie durch Host-Pfade oder Dateien identifiziert, nicht durch Namen.

`docker run -d --name irgendEnName -v irgendEhVerzeichnis:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=geheim mariadb` ; `(-v hostverzeichnis:containerverzeichnis)`

Sie verstehen den Standardaufbau eines Netzwerks mit Docker.


Screenshot_20240505_173349_Brave.jpg

Das Bridge Netzwerk gilt als Standard Netzwerk für alle Container.

Wir können auch neue Netzwerken erstellen, wieso? Unterschiedliche Anwendungen sollten aus sicherheitsgründen voneinander isoliert werden.

> Durch die Verwendung separater Netzwerke können wir den Datenverkehr zwischen verschiedenen Anwendungen kontrollieren und Zugriffsbeschränkungen implementieren, um die Angriffsfläche zu verringern und sensible Daten zu schützen.

Sie kennen die Syntax von Dockerfiles.

```
# Verwende das offizielle Node.js-Image als BasisFROM node:alpine
# Setze das Arbeitsverzeichnis im ContainerWORKDIR /usr/src/app
# Kopiere die Paketdefinitionen und die Lock-DateiCOPY package*.json ./
# Installiere AbhängigkeitenRUN npm install
# Kopiere den Rest der AnwendungCOPY . .
# Exponiere den Port 3000EXPOSE 3000
# Definiere den Befehl zum Ausführen der AnwendungCMD ["node", "app.js"]
```

Sie können die 12 verschiedenen Anweisungen von Dockerfiles erklären

- Sie können zwischen ENTRYPOINT und CMD sowie COPY und ADD unterscheiden

Oben

- Sie können eigene Dockerfiles für eine Webseite erstellen, testen und dokumentieren.

Praktisch

Standardimages:

Die folgenden Images aus dem dockerhub dienen vielfach als Basisimages für alle möglichen Anwendungen und sind damit besonders wichtig. Sie werden deshalb in diesem Kapitel kurz beschrieben.

Die kleine Grösse bewirkt eine grosse Performace beim Starten von Anwendungen, die auf diesem Image beruhen. Die geringe Grösse wird natürlich mit einem Verzicht auf Komfort und bekannte Linux Standardtools erkauft.