Arbeitspaket - Build & Docker

Voraussetzungen

Installationsvoraussetzung: Docker installiert[1]

Weiterhin ist es sinnvoll, wenn ihr euch den Foliensatz zu Docker anschaut.

Aufgaben

Ziel dieses Arbeitspakets ist es erste Erfahrungen mit Docker zu sammeln. Dazu gehört das Erstellen von Docker-Images, das Instanziieren dieser als Container, das Schreiben von Dockerfiles und die Nutzung von Docker-Compose. Zu Beginn des Arbeitspakets ein kleines Tutorial durchgegangen werden, um erste Erfahrungen mit Docker zu sammeln. Anschließend soll eine eigene Docker-Compose Datei erstellt werden.

1) Tutorial: Einführung in Docker

In dieser Teilaufgabe sollt ihr ein Tutorial bezüglich Docker durchgehen, um erste Erfahrung mit dem Umgang von Docker zu sammeln. Befolgt dazu folgende Schritte:

Schritt 1: Starte zunächst ein Terminal (z.B. CMD, Powershell). Erstellt einen Docker-Container auf Basis des Docker-Image von Ubuntu Betriebssystem mit folgendem Befehl:

docker run --name sep ubuntu:latest

Der Container wurde nun erstellt und gestartet. Der Name des Containers lautet „sep“. Über diesen Namen kann der Container nun angesprochen/inspiziert werden. Da der Container jedoch keine Aufgabe erfüllt wird er sofort wieder beendet.

Schritt 2: Mit dem Befehl *docker ps -a* kann eine Liste aller Container angezeigt werden. Der vorher erstellte Container mit dem Namen „sep“ wird in dieser Liste angezeigt. Mit dem Befehl *docker ps* (ohne -a) werden nur die aktuell laufenden Container angezeigt.

Schritt 3: Um den Container wieder zu löschen kann der Befehl *docker rm sep* genutzt werden. Wird nun der Befehl *docker ps -a* eingeben, ist der Container „sep“ nicht mehr in der Liste enthalten.

Schritt 4: Um das Ubuntu Betriebssystem innerhalb des Containers nutzen zu können (ohne dass der Container sofort wieder beendet wird), kann der Container mit dem Tag *-it* im interaktiven Modus gestartet werden. Der Befehl sieht wie folgt aus:

docker run -it --rm --name sep ubuntu:latest

Der zusätzliche *--rm* Tag sorgt dafür, dass der Container automatisch nach Beendigung entfernt/gelöscht wird, was zum Testen sinnvoll sein kann. Nun habt ihr Zugriff auf die Shell innerhalb

[1] https://docs.docker.com/engine/install/

des Containers und könnt typische Shell-Befehle, beispielsweise zum Navigieren innerhalb des Dateisystems, ausführen. Beispielhafte Shell-Befehle sind:

- *ls* (zeigt das aktuelle Verzeichnis an)
- *cd lib* (wechselt in das lib Verzeichnis).

Schritt 5: Als nächstes sollen die Container inspiziert werden. Dazu soll ein zweites Terminal gestartet werden. Anschließend soll der Befehl *docker info* in dem Terminal eingegeben werden, um generelle Informationen (z.B. die Anzahl der Container/Images) zu erhalten.

Um den zuvor gestarteten *sep Container* zu inspizieren, kann *docker inspect sep* genutzt werden. Hier kann z.B. die IP-Adresse des Containers gefunden werden. Ebenfalls zeigt der Punkt Ports an, dass der Container gerade keine Ports geöffnet hat.

Nun kann der Container beendet werden, indem entweder *exit* in der Container-Shell eingegeben wird oder in dem zweiten Terminal *docker stop sep* ausgeführt wird.

Schritt 6: Im Folgenden wir das Networking zwischen Docker-Containern betrachtet. Dazu sollt ihr zunächst mit dem Befehl *docker run -it --rm --name sep_server nginx:latest* einen NGINX-Webserver starten und diesen aus einem zweiten Terminal inspiziere um die IP-Adresse zu ermitteln. Auf dem Endpunkt http://<IP-Adresse>:80 wird die NGINX Willkommensseite (sprich ein Platzhalter) gehostet.

Nun versuche diesen Endpunkt (sprich die Willkommensseite)

1. Ausgehend von dem Host (dein Betriebssystem) zu öffnen, indem du die Adresse in einem Browser eingibst.
2. Aus einem zweiten Container zu öffnen. Hierzu kannst du z.B.:
    1. das Ubuntu-Image erneut starten instanziieren (siehe **Schritt 4**) und daraufhin in der Shell cURL mit den folgenden zwei Befehlen installieren:
        i. *apt update*
        ii. *apt install curl*
    2. Den Endpunkt (sprich die Willkommensseite) mit cURL durch folgenden Befehl öffnen: *curl http://<IP-Adresse>:80*

Lediglich die zweite Methode führt dazu, dass die Willkommensseite tatsächlich angezeigt wird. Dies ist darin begründet, dass Docker-Container sich standardmäßig in einem separaten virtuellen Netzwerk befinden. Das Hostsystem ist kein Teil dieses virtuellen Netzwerks.

Um Endpunkte eines Containers auch für Prozesse des Hostsystems zugreifbar zu machen, können Ports mit dem *-p* Tag veröffentlich werden.

Beende den NGINX-Webserver mit dem Befehl *docker stop sep_server* und starte ihn erneut, wobei der Container-Port 80 auf den Port 8080 des Host Netzwerks gemappt wird:

docker run -it --rm --name sep_server -p 8080:80 nginx:latest

Die NGINX Willkommensseite kann nun in einem Browser auf dem Hostsystem unter http://localhost:8080 angezeigt werden.

Schritt 7: Im Folgenden sollt ihr euch mit dem Erstellen von Docker Images auseinandersetzen. Docker-Images werden auf Basis einer Dockerfile konfiguriert. Bei der Erstellung von Docker-Images werden in der Regel bestehende Images verwendet und weitere Dateien, Konfigurationen (z.B. bzgl. Networking) etc. hinzugefügt. Zur Demonstration des Erstellens eines Images sollt ihr zunächst ein simples Konsolenskript schreiben. Legt dazu einen leeren Ordner an und öffne ein Terminal in diesem (z.B. via shift + Rechtsklick bei Windows). Erstelle hier die Datei „hello.sh“ mit dem Inhalt:

```
#!/bin/sh
echo "Hello world"
bash
```

Schließt nun die Datei „hello.sh“ und erstellt eine Datei namens „Dockerfile“ (Wichtig: Der Name muss identisch sein). Füllt den Inhalt der Datei „Dockerfile“ wie folgt:

```
FROM ubuntu:22.04
COPY hello.sh / hello.sh
CMD ["sh", "/ hello.sh"]
```

Der Inhalt der Dockerfile hat folgende Bedeutung:

- Mit *FROM ubuntu:22.04* wird *ubuntu:22.04* als Base-Image festgelegt,
- Mit *COPY* hello.*sh / *hello.*sh* wird das Skript „hello.sh“ von der Host-Festplatte in den Container kopiert
- Mit *CMD ["sh", "/* hello.*sh"]* wird festgelegt, dass das Skript „hello.sh“ beim Start des Containers ausgeführt wird.

Schritt 8: Als nächste sollt ihr auf Basis des Dockerfiles ein Image mit dem Tag „my_image“ mit folgendem Befehl erstellen:

docker build -t my_image .

Wichtig: Der Punkt darf nicht vergessen werden (sofern der Befehl im gleichen Verzeichnis ausgeführt wird, in welchem sich das Dockerfile befindet). Mit dem Befehl *docker image ls* wird nun das neu erstellte Image „my_image“ angezeigt und kann via *docker run* (siehe **Schritt 4**) gestartet werden. Beim Start des Containers wird nun das Script „hello.sh“ ausgeführt.

2) Schreiben einer Docker-Compose File

In dieser Teilaufgabe sollt ihr die bereits angelegte Datei „docker-compose.yml“, welche sich im Ordner des Arbeitspaketes befindet, vollständig ausfüllen. Lest euch dazu zunächst die Dokumentation von Docker-Compose[2] durch.

Hilfe

- Docker Images finden: https://hub.docker.com/
- Docker Docs: https://docs.docker.com/

[2] https://docs.docker.com/compose/compose-file/compose-file-v3/