**Befehle in docker in cmd**

**Docker --help**

(**sudo bei linux) docker info**

**Docker ps:** gibt alle laufenden container aus

**Docker ps -a** : -a gibt quasi alle images an die mal gelaufen sind oder aktuell laufen

**Docker pull busybox** : mit pull wird ein bestimmtes image heruntergladen. In dem fall busybox

**Docker images**: zeigt alle heruntergeladenen images an

**Docker run busybox**: Startet ein image

**Docker remove 77a400431be5 :** das löscht den container, also das image

**Für linux :Sudo docker rm $(sudo docker ps -a -q -f status=exited):** löscht alle container die den status=exited haben

**Und hier für windows cmd: FOR /f "tokens=\*" %i IN ('docker ps -a -q') DO docker rm %i**

**docker run -d --name <Gewünschter-Container-Name> <Image>:** hier wird dem container einen namen zugeordnet

**Docker run -p 5000: 80 nginx :** hier mit wird der port umgeleitet. Er ist standardmäßig auf 80 und nun ist er auf 5000

**docker run -p 5001:80 -d nginx:** mit minus d läuft der container im hintergrund weiter

docker run -p 5002:80 -d --name WoWServer nginx: mit –name wird ein name für den server angelegt

**docker exec -ti 08f2491c2aa6 bash :** Damit gelangt man in den container und kann sich die verzeichnisse ansehen

POSTGRES ERSTELLEN

docker run --name some-postgress -e POSTGRES\_PASSWORD=password -p 5432:5432 -d postgres

**Docker Image erstellen**

Wir nehmen als Beispiel den Chatbot aus Python den wir für die SpringBoot Anwendung nutzen wollen.

**Was wir für Dateien brauchen:** .dockerignore , requirements.txt , ConfigImage.dockerfile

Diese Dateien werden in dem Projektordner erstellt.

In **.dockerignore** schreiben wir rein, welche Dateien oder Ordner wir nicht in das Image laden wollen.

Bsp:

\_\_pycache\_\_

.git

.idea

Venv

Chat.pt

Train.py

In **requirements.txt** schreiben wir alle wichtigen Bibliotheken + die Version rein die für die Anwendung wichtig sind.

Bsp:

requests==2.31.0

torch==2.1.2

numpy==1.26.3

nltk==3.8.1

flask==3.0.1

In **ConfigImage.dockerfile** schreiben wir welche Version von Python genutzt wird, das Arbeitsverzeichnis im Container, wo die Anwendungsdateien liegen sollen usw. Die Schichten die sich am häufigsten Ändern (meist die Projektdateien) sollten diese so weit wie möglich unten im Code erscheinen. Der Cache der Schichten wird so besser genutzt. Das Besondere hier: Die requirements werden nur einmal gedownloaded und müssen nicht jedesmal bei einer änderung der programmierung neu geladen werden.

# Erster Build-Stage: Installiere Abhängigkeiten

FROM python:3.11.7 AS builder

WORKDIR /app

COPY requirements.txt .

RUN pip install --upgrade pip && \

    pip install -r requirements.txt

# NLTK-Ressourcen herunterladen

RUN python -m nltk.downloader punkt

# Zweiter Build-Stage: Kopiere nur notwendige Dateien in ein kleineres Image

FROM python:3.11-slim

WORKDIR /app

# Kopiere Abhängigkeiten aus dem Builder-Image

COPY --from=builder /usr/local/lib/python3.11/site-packages /usr/local/lib/python3.11/site-packages

COPY --from=builder /usr/local/bin /usr/local/bin

# Kopiere deine Anwendungsdateien

COPY . .

# Führe die Anwendung aus

CMD ["python", "chatforSpringBoot.py"]

Mit dem Befehl wird das image erstellt und zum laufen gebracht:

docker build -t chatbotspring -f ConfigImage.dockerfile .

docker run -p 5002:5002 -d --name Chatbot chatbotspring

Der chatbot wird lauscht mit flask auf den port 5002, deshalb 5002:5002

1. docker build: Dieser Teil des Befehls gibt an, dass du ein Docker-Image erstellen möchtest.
2. -t chatbotspring: Der Flag -t steht für "tag" und wird verwendet, um dem erstellten Docker-Image einen Namen und/oder eine Versionsnummer zuzuweisen. In diesem Fall ist der Name des Images "chatbotspring". Der Tag ist optional und wird hier nicht verwendet.
3. -f ConfigImage.dockerfile: Der Flag -f wird verwendet, um das Dockerfile anzugeben, das für den Build-Prozess verwendet werden soll. Hier gibt ConfigImage.dockerfile den Dateinamen des Dockerfiles an.
4. .: Der Punkt am Ende gibt den Build-Kontext an. Das ist das Verzeichnis, in dem sich das Dockerfile und alle Dateien befinden, die für den Build benötigt werden. In diesem Fall wird angenommen, dass sich das Dockerfile im aktuellen Verzeichnis befindet, daher wird . verwendet.