Build Instructions

- `FROM` hier wird das Image angegeben auf dem das neue Image basiert. Wenn es auf keinem Image basieren soll, kann auch `Scratch` angegeben werden.
- `MAINTAINER` Wer ist für dieses Image zuständig und führt die Wartung durch?
- `COPY` kopiert den Inhalt des angegebenen Pfades in den Container zu dem angegebenen Pfad
- `ADD` wie `COPY`, nur kann dieser Befehl komprimierte Dateien automatisch öffnen und entpacken. Wenn diese Funktion nicht benötigt wird sollte `COPY` stattdessen verwendet werden.
- `RUN` führt ein Shell Script aus. In Linux auf der `/bin/bash` in Windows auf der `cmd`. Das auszuführende Script muss sich schon innerhalb des Containers befinden. Dieser Befehl wird verwendet um den Container zu erstellen. Wenn dieser bereits erstellt und gestartet wurde werden die Befehle mit `CMD` oder `ENTRYPOINT` angegeben.
- `ENV` setzt Umgebungsvariablen innerhalb des Containers
- `USER` setzt den User unter welchem die Scripte ausgeführt werden
- `ENTRYPOINT` legt fest welcher Befehl beim Start des Containers ausgeführt werden soll.
- `CMD` dieser Befehl wird nach dem Starten des Docker Containers ausgeführt. Es gibt drei verschiedene Formen des Befehls. Die Hauptaufgabe des `CMD` Befehles ist es aber Defaultwerte für die Ausführung von `ENTRYPOINT` bereit zu stellen.
    - `CMD ["executeable","param1","param2"]`, führt "excuteable" mit den angegebenen Parametern aus.
    - `CMD ["param1","param2"]`, diese Parameter werden dem executable des Entrypoint hinzugefügt.
    - `CMD command param1 param2`, shell Format zum Ausführen von Befehlen

      Die Befehle `CMD` und `ENTRYPOINT` darf es jeweils nur einmal geben. Kommen diese Befehle dennoch öfter in einem Dockerfile vor, hat nur der jeweils letzte Befehl einen Effekt.
- `VOLUME` mountpoint von Hostsystem in dem Container. Der angebenene Ordner steht auch innerhalb vom Container zur Verfügung. Auf diese Weise können Daten zwischen den Containern ausgetauscht werden.
- `EXPOSE` gibt die Ports an auf welche der Container hört. Wenn nichts angeben wird handelt es sicht um TCP, wenn `/udp` angegeben wird kann auf das UDP Protokoll umgestellt werden.
- `ONBUILD` fügt einen Trigger hinzu der ausgeführt wird wenn dieses Image als Base Image (also im `FROM` Befehl eines anderen Dockerfiles) verwendet wird. Nach `ONBUILD` kann eine der oben genannten `build instructions` angegeben werden.

Docker Commands

Eine vollständige Übersicht über alle Docker command line Befehle gibt es bei Docker unter [https://docs.docker.com/engine/reference/commandline/docker/#child-commands], die wichtigsten Befehle werden hier kurz erklärt. Diese Befehle können in der Unix `shell` oder unter Windows im `cmd` angewendet werden. Auf Windows empfiehlt es sich aber hierfür die `Powershell` zu nutzen.

Wenn `(CONTAINER | SERVICENAME)` angegeben ist, dann kann statt dem Containernamen auch der Name des Services eines Docker-Composestacks (Docker-Compose wird nachfolgend in `Anwendungsstacks mit docker-compose` genauer erkläert) angegeben werden

- `docker build -t "imagename"` buildet ein Docker Image aus einer Datei namens " *Dockerfile*" (ohne Dateiendung), das Image bekommt den Namen der mit `-t` als Parameter übergeben wurde.
- `docker run -d -p 8080:80 "imagename"` erstellt und startet einen Container basierend auf dem Images,
    - `-p` mappt den port 8080 (des Hostsystems) im Container auf den port 80.
    - `-d` startet den Container im detached mode, der Container wird im Hintergrund gestartet und die console wird nicht von der Ausgabe des Docker Containers blockiert.
- `docker images` zeigt alle gebauten oder heruntergeladenen Images an
- `docker image rm IMAGE` entfernt das angegebene Image
- `docker container ls -a` zeigt alle laufenden Container an, der Parameter `-a` zeigt auch alle derzeit nicht laufenden an. Dieser Command ersetzt `docker ps` welcher aber auch weiterhin noch funktioniert.
- `docker start CONTAINER` startet den angegeben Container.
- `docker logs -f (CONTAINER | SERVICENAME)` zeigt den log output des Containers
    - `-f` : mit dem Parameter -f wird der folgende log output live ausgegeben.
- `docker attach (CONTAINER | SERVICENAME)` hängt den Standard Input und Output auf die aktuelle Konsole. Man kann wie mit ssh eine Verbindung in den Container aufbauen. Achtung: dies ist keine richtige ssh Verbindung. Das merkt man insbesondere wenn man die Verbindung wieder auflösen will. `CTRL-c` oder exit stoppt auch den Container. Die Verbindung kann mit `CTRL-p CTRL-q` gelöst werden.
- `docker exec -it CONTAINER /bin/sh` mit exec kann ein Befehl innerhalb des Containers ausgeführt werden. Wird der Befehl `/bin/sh` (wenn eine bash vorhanden ist kann auch `/bin/bash` verwendet werden) angegeben, ist dies eine alternative Variante um auf die shell im Container zuzugreifen. Wichtig ist das die Option `-it` angegeben wird. Der Vorteil dieser Variante gegenüber docker attach ist das die shell wie gewohnt mit exit verlassen werden kann und der Container dabei nicht gestoppt wird.
- `docker stop CONTAINER` stoppt den angegeben Container. Der Container wird heruntergefahren.
- `docker kill CONTAINER` der Container wird mit einem Kill signal gestoppt.

- `docker rm CONTAINER` entfernt den Docker Container (nur den Container nicht das Image)

Die folgenden Befehle können nicht in der Windows commandline `cmd` eingegeben werden, sondern müssen auf Windowssystemen in der Powershell ausgeführt werden. In Unix Systemen können diese Befehle normal ausgeführt werden.

- `docker rm $(docker ps -aq)` löscht alle Container (`docker ps -aq` der parameter `-q` beschränkt die Ausgabe auf die Ids. Der Befehl gibt also die Ids aller Container zurück und mit `$` wird über all diese Ids iteriert und docker rm ausgeführt)
- `docker image rm $(docker images -q)` löscht alle Images aus dem lokalen Repository (wie oben beschrieben nur mit docker images `-q`)

Docker-compose Commands:

Immer wenn [SERVICENAME] angegeben ist kann auch optional ein Service angegeben werden und der Befehl gilt nur für dieses Service

- `docker-compose build` baut die Services
- `docker-compose up -d [SERVICENAME]` erstellt die Container und startet diese. Die Container werden in der korrekten Reihenfolge erstellt und gestartet. Wenn die Container bereits erstellt sind und laufen, bewirkt dieser Befehl einen Restart.
  `-d` : startet die Container im Hintergrund
- `docker-compose down` stoppt und entfernt alle Container, Netzwerke, Images und Volumes
- `docker-compose start [SERVICENAME]` startet die Container, dazu müssen diese aber vorher eingestellt worden sein.
- `docker-compose stop [SERVICENAME]` stoppt alle zu dem Stack gehörenden Container.
- `docker-compose rm -f` entfernt alle zu dem Stack gehörenden Container, diese müssen gestoppt sein.
  `-f` : erzwingt das Löschen der Container
- `docker-compose logs` zeigt die logs aller zu dem Stack gehörenden Container an. Wenn der Stack mit docker-compose up `-d` im Hintergrund gestartet wurde können so die Logsfiles verfolgt werden.
- `docker-compose ps` zeigt die Liste aller dem Stack zugehörenden Container und deren Status an.
- `docker-compose exec SERVICENAME /bin/sh` gleich wie bei docker exec wird ein Befehl innerhalb des Containers ausgeführt. Da die Container, welche von docker-compose erstellt werden, immer den Namen des Ordners in dem sich das docker-compose.yml file befindet, enthält, ist der Name des Containers meist ORDERNAME_SERVICENAME_1 (für die erste Instanz des Services) und könnte auch mit docker exec ORDERNAME_SERVICENAME_1 angesprochen werden. Bei `docker-compose exec` kann der

Container aber eben direkt über den SERVICENAMEN angesprochen werden. Hier werden die Parameter -it nicht benötigt.

- `docker-compose scale SERVICENAME=X` die Anzahl der laufenden Containerinstanzen des Service SERVICENAME wird auf X erhöht oder verringert. Dabei muss darauf geachtet werden, dass die Definition des Services im `docker-compose.yml` File skalierbar ist. Wenn zum Beispiel `ports: -80:80` angegeben ist kann es zu einem Konflikt kommen. Es kann nur ein Container von diesem Service erstellt werden, da ein zweiter ebenfalls auf den Port 80 des Hostsystem gemappt haben möchte, dieser aber bereits in Verwendung ist. In der `docker-compose.yml` muss `ports: -80:80` zu `ports: -80` geändert werden. Docker vergibt und managed die internen Ports selbst.