DOCKER CHEAT SHEET

Create Dockerfiles

To build Docker images, you first need a Dockerfile. A Dockerfile is a text file named Dockerfile and has no file extension.

```
FROM <baseimage> # get them from https://hub.docker.com

# set the working directory inside the image
WORKDIR /app

# copy the source code from the host machine to the image
# (host: your development machine or your server)
# the first dot (.) is the current working directory (CWD)
# of the host, the second dot (.) is the CWD of the image
COPY . .

# run any shell command inside the image
RUN <command> # e.g., [npm run build]

# does not really expose the port 80 (only documentation)
# so that we know which port to expose
EXPOSE 80 # we expose ports with the --publish flag

# this command gets executed when the container starts
CMD <command> # e.g., [”node”, “server.js”]
```

Create Multistage Dockerfiles

Multistage Dockerfiles are used to optimize Dockerfiles. One use case is to create a builder and a serve stage with separate base images. This strategy can be used to make the final Image smaller and have a lower attack because it has less system libraries. Each stage starts with FROM.

```
# with as, you can give the current stage a variable name
FROM <baseimage> as builder

# now do something, install dependencies, build your code
RUN <command> # e.g., g++ main.cpp

# the second stage could use a smaller image
# small images are based on alpine or you can build
# FROM scratch (this is a Docker image)
# if you do not need any system libraries
FROM scratch as serve

# you can now copy files from the builder stage
# e.g., the binary file that you build in that stage
COPY --from=builder ./hello-world ./hello-world

# this command gets executed when the container starts
CMD [”./hello-world”]
```

Create Docker Images

```
# <path> sets the context for the docker build command
# set to dot (.) it will use the CWD of the hostmachine
# to find the Dockerfile
$ docker build <path>

# if the Dockerfile is located somewhere else
# or the Dockerfile is named differently use --file
# remember docker will still use the <path> as context
$ docker build --file ./path/to/Dockerfile
./path/to/context

# name images: <image-name> equals <name>:<tag> where
# name is separated into <username>/<repository>:<version>
$ docker build --tag user/repo:0.1.0 .

$ docker image ls # list all images
```

Create Docker Containers

```
# Docker containers are running images
$ docker run <image-name>

# you can run public images from Docker Hub
# or images from a private registry
$ docker run https://privateregistry.com/<image-name>

# containers are started in the foreground. as soon as
# you kill the process e.g., the terminal, it will stop
# the container
# to run a container in the background, you need to run
# it in detached mode
$ docker run --detached <image-name>

# list all containers
$ docker container ls
# or shorthand syntax
$ docker ps
# list all containers, including stopped ones
$ docker container ls --all

# stop a container
$ docker stop <container-id>
# remove a container
# only stopped containers can be removed
$ docker rm <container-id>
# start a stopped container
$ docker start <container-id>
# restart a running container
$ docker restart <container-id>

# automatically remove a container when it is stopped
$ docker run --rm <image-name>
```

Access Docker Containers

```
# publish ports, e.g., forward container port to host port
$ docker run --publish <host-port>:<container-port>
  <image-name>

# execute shell command in a container
$ docker exec --interactive --tty <container-id> <command>

# open an interactive shell (like connecting to a server)
$ docker exec --interactive --tty <container-id> sh

# exit the container
$ exit
```

Create Docker Volumes

To persist data from Docker containers between starts of containers you need volumes. When a container is removed all data from the container will be lost if you do not use volumes.

```
# using a named volume (docker handles location on host)
$ docker run --volume <volume-name>:/path/in/container
  <image-name>

# using a mounted volume (you handle location on host)
$ docker run --volume /path/on/host:/path/in/container
  <image-name>

# list all volumes incl. metadata
$ docker volume ls
```

Blog: https://devopscycle.com/blog/the-ultimate-docker-cheat-sheet/
GitHub: https://github.com/aichbauer/the-ultimate-docker-cheat-sheet/
Consulting: https://devopscycle.com/