2.1.1 Part 1: Getting started

Q1: Why you run the container docker/getting-started in detached mode?

Detached mode means that the container will be running in the **background**, without user interaction. This is useful because the terminal where the process was started can closed and the user can run other commands.

Q2: What is the difference between a container and a container image?

The container image is the file that contains all of the instructions for how to **build** the container. It is the blueprint that tells docker how to create the container. Depending on you needs, there are many images available depending on your needs, e.g. a python image that comes with python already installed. The container is a **runnable instance** of the image which is managed by Docker.

2.1.2 Part 2: Sample application

Q1: What is the meaning of the docker's directives used in the docker file? Please comment on each of the directives.

```
# syntax=docker/dockerfile:1

# specifies the base image that we will build our custom image around
FROM node:18-alpine

# sets the working directory for other instructions like COPY. E.g. when referring to the cu
WORKDIR /app

# copies everything from the current host directory, into the /app directory.
COPY . .

# runs the "yarn install --production" command, which installs dependencies
RUN yarn install --production

# the command that is executed when the container is started. Here, it will run the node app
CMD ["node", "src/index.js"]

# opens up port 3000 on the container
EXPOSE 3000
```

Q2: Why it is important to tag a container image?

Tagging images is important because it allows you to **identify** your image after building it. If you do not specify a tag, you will have to use the **image ID** whenever you interact with container, e.g. stopping, starting etc. As the number of images in your repository grows, identifying which image does what becomes increasingly difficult. Additionally, with tags you can specify **versions** of your image, where each tag is a version number, e.g. `0.0.1`, that points to a specific version.

Q3: Why we should bind a host port with the container port?

Binding a host port with a container port forwards any requests that come to the host port to the specified container port. This is import for applications that are running inside the container and arre listening for incoming messages on that port. For example, a containerised webserver that is listening on port `5000`, will only receive messages if it is bound to host port.

2.1.3 Part 3: Update the application

Q1: It is possible to bind two containers on the same host port?

No, it is not possible to bind two containers to the same host port. Each host port can only be assigned to a **single** process.

Q2: Why and when, after stopping a container, you could need to remove it?

You remove containers to free up space on your system.

Q3: It is possible to remove a running container, without stopping it before the removal?

No, to be able to remove a container it first needs to be stopped. Though you can stop and remove it with a **single** command.

2.1.4 Part 4: Share the application

Q1: Given a container image available on a docker image repository, can you start an instance of the image on any docker host? Is there any limitation?

Yes, it should be possible to start the instance of the image on **any** docker host. The whole idea of docker is to separate the application from the host that is running it by shipping the host itself so that it can run on any system with docker installed.

2.1.5 Part 5: Persist the DB

Q1: If you run two instances of the same container image, let's call them container A and container B, and you create a file in container A, is that new file visible in container B?

If all you do is to create the file in container A, it will **not** be visible in container B since their file systems are separated.

Q2: Why in the docker command "docker run -d ubuntu bash -c"shuf -i 1-10000 -n 1 -o /data.txt && tail -f /dev/null" we need to keep the container running with "tail -f /dev/null"?

The command `tail -f /dev/null` watches the `data.txt` file so that the container keeps running after `data.txt` file has been created for us to inspect it.

Q3: What you can do with the "docker exec" command?

The `docker exec` command let's you execute a command inside the container. E.g.

```
docker exec -it <container ID> bash
```

runs the `bash` command inside the container.

Q4: Let assume you decide to use a volume. Why you need to mount the volume in the container file system? Does that mean you modify the container filesystem?

The volume lets you connect a specific path on the container file system to the host file system and persist data from the container on the host. We need to mount the volume into the container file system so that the volume data becomes available inside the container. This means that the container filesystem

is modified, since any data that is in the volume, now lives inside the container as well.

Q5: In this part of the tutorial, you have created a volume. Where is the volume located in the file system of your docker host?

The volume is located at `/var/lib/docker/volumes/` on the docker host.

2.1.6 Part 6: Use bind mounts

Q1: What is a dev-mode container?

A dev-mode container is a container which has the development workspace from the host **mounted** into it. This means that we can write code on the host machine and immediately see the results in the application that is running inside the container.

Q2: Can we run a container, install software dependencies, and then use the updated container without building first the image?

2.1.7 Part 7: Multi container app & Part 8: Use Docker Compose

Q1: What is a Docker service?

Q2: Can we spin up a single instance of a docker container using a docker-compose file?

Yes, we can specify as many services as we want inside the `docker-compose.yaml` file.

Q3: Can we run a MySQL container and store the database structure and data in a volume?

Yes, we can. The MySQL database is stored in a `.db` file, so all we need to do is to create a volume and mount this file when starting the MySQL container.

Q4: Is the order of the services defined in a docker-compose file important, or is it irrelevant which service is defined first?

The order of service definition has no impact on the order in which they are started and will be ready, so if we do not care about starting services in a specific

order, we can define them in any order we like.

Q5: Is it mandatory to define the network in a docker-compose file?

No, it is not mandatory to define the network in the `docker-compose.yaml`. Docker compose will automatically create a network with the service names as aliases, if we do not specify one.

Q6: If you would like to run a multi-container app, is it necessary to use docker compose (i.e. to define a service) or you can achieve the same objective using docker commands from the shell? Does a service offers more than just running a multiple container app with a single command?

It is not necessary to use docker compose for a multi-container app. One can achieve the same thing by using docker commands in the command line. However, using docker compose is a better approach when dealing with multiple containers because the entire configuration is written down as code. This means that the configurartion can be versioned, reviewed, and contributed to through a version control system. Additionally, it is easy to share and to repeatedly execute without resorting to manual intervention.