Docker and Docker Compose Commands

In this lesson, we'll study a few Docker and Docker Compose commands.

WE'LL COVER THE FOLLOWING

Docker Compose is used for the coordination of multiple Docker containers. Microservices systems usually consist of many Docker containers. Therefore, it makes sense to start and stop the containers with Docker Compose.

Docker Compose

Docker Compose uses the file `docker-compose.yml` to store information about the containers. The Docker documentation explains the structure of this file. The Docker Compose lesson contains an example of a Docker Compose file.

Upon starting, `docker-compose` outputs all possible commands. The most important commands for Docker Compose are:

- `docker-compose build` generates the Docker images for the containers with the help of the `Dockerfiles` referenced in `docker-compose.yml`.
- `docker-compose pull` downloads the Docker images referenced in `docker-compose.yml` from Docker hub.
- `docker-compose up -d` starts the Docker containers in the background.

Without `-d` the containers will start in the foreground so that the output

of all Docker containers happens on the console. It is not particularly clear which output originates from which Docker container. The option `--scale` can start multiple instances of a service, e.g. `docker-compose up -d --scale order=2` starts two instances of the order service. The default value is one instance.

- `docker-compose down` stops and deletes the containers. In addition, the network and the Docker file systems are deleted.
- `docker-compose stop` stops the containers. Network, file systems and containers are not deleted.

Docker

At startup, `docker` outputs all valid commands without parameters.

Tip: Tab-pressing completes names and IDs of containers and images.

Here is an overview of the most important commands. The container `ms_catalog_1` is used as an example.

State of a container

- `docker ps` displays all running Docker containers. `docker ps -a` also shows stopped Docker containers. The containers like the images have a hexadecimal ID and a name. `docker ps` outputs all this information. For other commands, containers can be identified by name or hexadecimal ID. For the example in this course, the containers have names like `ms_catalog_1`. This name consists of a prefix `ms` for the project, the name of the service `catalog` and the sequence number `1`. The name of the container is often confused with the name of the image (e. g. `ms_catalog`).
- `docker logs ms_catalog_1` shows the previous output of the container `ms_catalog_1`. `docker logs -f ms_catalog_1` also displays all other outputs that the container still outputs.

Lifecycle of a container

- `docker run ms_catalog --name="container_name"` starts a new container

with the image `ms_catalog`, which gets the name `container_name`. The parameter `--name` is optional. The container then executes the command that is stored in the `CMD` entry of the `Dockerfile`. But you can execute a command in a container with `docker run <image> <command>`. `docker run ewolff/docker-java /bin/ls` executes the command `/bin/ls` in a container with the Docker image `ewolff/docker-java`. So the command displays the files in the root directory of the container. If the image does not yet exist locally, it is automatically downloaded from the Docker hub on the Internet. When the command has been executed, the container shuts itself down.

- `docker exec ms_catalog_1 /bin/ls` executes `/bin/ls` in the running container `ms_catalog_1`. Thus, with these commands you can start tools in an already running container. `docker exec -it ms_catalog_1 /bin/sh` starts a shell and redirects input and output to the current terminal. This way, you have a shell in the Docker container and can interactively work with the container.

- `docker stop ms_catalog_1` stops the container. It first sends a SIGTERM so that the container can shut down cleanly, and then a SIGKILL.

- `docker kill ms_catalog_1` terminates execution of the container with a SIGKILL, but the container is still there.

- `docker rm ms_catalog_1` permanently deletes the container.

- `docker start ms_catalog_1` starts the container again that was stopped before. As the data is not deleted when the container was stopped, all data is still available.

- `docker restart ms_catalog_1` restarts the container.

Docker images

- `docker images` displays all Docker images. The images have a hexadecimal ID and a name. For other commands, images can be identified by both mechanisms.

- `docker build -t=<name> build <path>` creates an image with the name, `name`. The `Dockerfile` has to be stored in the directory `path`. When no version is indicated, the image gets the version `latest`. As an alternative,

the version can also be indicated in the format `-t=<name:version>`. The Dockerfiles lesson describes the format of the `Dockerfiles`.

- `docker history <image>` shows the layers of an image. For each layer, the ID, the executed command and the size of the layer are displayed. The image to be displayed can be identified by its name if there is only one version of the image with that name. Otherwise, the name and version must be specified via `name:version`. Of course, you can also use the hexadecimal ID of the image.
- `docker rmi <image>` deletes an image. As long as a container is still using the image, it cannot be deleted.
- `docker push` and `docker pull` store Docker images in a registry or load them from a registry. If no other registry is configured, the public Docker hub is used.

Cleaning up

There are several commands to clean up the Docker environment.

- `docker container prune` deletes all stopped containers.
- `docker image prune` deletes all images that do not have a name.
- `docker network prune` deletes all unused Docker networks.
- `docker volume prune` deletes all Docker volumes which are not used by a Docker container.
- `docker system prune -a` deletes all stopped containers, all unused networks, and all images which are not used by at least one container. So, all that remains is what the currently running containers need.

Troubleshooting

If an example does not work:

- Are all containers running? `docker ps` displays the running containers, `docker ps -a` also shows the terminated ones.

- Logs can be displayed with `docker logs`. This also works for terminated containers. The term `Killed` in the logs denotes that too little memory is available. Under Windows and macOS you can find the settings for this in the Docker application under Preferences/ Advanced. Docker should have about 4 GB assigned.
- In case of more complex problems, you can start a shell in the container with `docker exec -it ms_catalog_1 /bin/sh` and examine the container more closely.