Norwegian University of Science and Technology
Department of Information Security and Communication Technology
TTM4200 Computer Networking

Support Document - Lab 1

Christian Lewin and Sebastian Fuglesang

June 2020

Contents

1 Docker

Docker (https://www.docker.com/) is a software used to make the development of applications smoother. It allows several containers, acting as lightweight-systems, often with just the bare minimum of necessary functionality. These containers run on top of the operating system the machine running Docker is using. Practically, containers can be compared to virtual machines, however, theoretically, they can not. If you are interested, this blogpost explains several of the key differences.

All of the labs in TTM4200 make use of Docker. In addition to this, Docker is a very common tool to use in the development industry, as it streamlines development. Therefore, learning Docker will help you move through this subject more easily, and it is a valuable skill to have. The most commonly used commands are explained in detail below:

1.1 docker run

- The command runs a docker container built from a specified image.
- Syntax: 'docker run --name <container-name> -p 3000:8080 –d --privileged --network <my-network> <my-image>'
  - We use the keyword 'docker' to specify that the following command is a docker command.
  - We use run because that is the docker command we use to run a container
  - --name <container name> gives you the option of naming your container. You can only use names that are not currently used by other containers. If you don't use the --name option docker will provide your container with a random name.
  - -p 3000:8080 will map port 3000 on the host machine to port 8080 on the container
  - Using -d lets you run the container in detached mode and the container will run in the *background* of your terminal.
  - Using --privileged allows the container to run as privileged. This gives the container permission to do more things. It is necessary to run some commands and programs. **NB.** *unless you are told to by us in the labs, do not run containers as privileged.*
  - Using --network <my-network> allows you to specify the network the container will be using. If you don't use the --network option the container will be added to a default network.
  - Finally you need to write the image name that you intend to use to make the container.
- Examples
  - "docker run –name min-container –p 3000:8080 –d –network ttm4200_net ubuntu"
  - "docker run ubuntu"
- After the command runs, a process will be started by docker. Depending on the container and which options you use in the docker run command the process can be finished and exited immediately or keep running until stopped.

1.2 docker ps

- This command will list all running containers.
- Syntax: 'docker ps -a'
  - ps stands for process status.
  - Using the -a option will make it show all existing containers. Both those that are started and stopped.

Sometimes, when you run a container, it might stop immediately. That is due to its main functionality. Some containers, unless specified otherwise, will do one function, and then exit. An example of such a container, is some container ran from the whalesay image. When it is run, it prints something to the terminal, and then exits immediately, as its main function is finished. Other containers, such as one built to function as a host or client, may run until told to stop. Also, if you run containers with the '-i' and '-t' flags, it will allow you to enter it through a terminal-emulation.

1.3 docker images

- This command will list all the docker images that are built on the computer.
- Syntax: 'docker images'

1.4 docker network ls

- This command will list all existing docker networks on the computer.
- Syntax: 'docker network ls'
  - ls stands for list.

1.5 docker network create

- This command allows you to create a new docker network that you can configure yourself with subnet, driver, name and other things.
- Syntax: 'docker network create --subnet <subnet> <my-network>'
  - subnet has to be defined with CIDR notation (x.x.x.x/y)/("x:x:x:x:x:x:x:x/y") for ipv4/ipv6.
  - If you don't use the --subnet option docker will make a subnet for you.
- Examples
  - "docker network create --subnet 10.11.10.0/24 my-network"
  - "docker network create my-network-without-userdefined-subnet"

1.6 docker exec

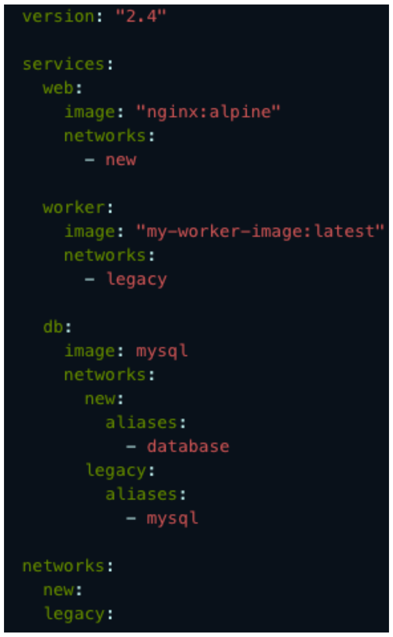
- This command allows you to enter a container in an interactive shell of the container where you can execute commands inside the container. Put in another way, you can control the container through a simulated terminal.
- Syntax: 'docker exec -i -t <Container ID/Container name> /bin/bash'
  - exec is short for execute
  - i is short for interactive
  - t is short for terminal
  - You specify which container you want to enter by typing either the containers ID or by typing its name
  - /bin/bash takes you to the folder bin which lies inside the root folder. Inside bin there are program files and bash is one of these programs. Bash stands for Bourne-Again SHell.
- Examples
  - "docker exec -it a73450b64455 /bin/bash"
  - "docker exec -it min-container /bin/bash"

1.7 docker cp

- This command allows you to copy files between a container and the host. This copying can go both from host to container and from container to host.
- Syntax from conatiner to host: ‘docker cp <container ID>:<source path> <destination path>
  - source path is the path to where the file lies. For example /folder/filename
  - destination path is where the file should be copied to.
- Syntax from host to container: ’docker cp <source path> <container ID>:<destination path>’
  - (Hint) if the filename you use for the destination already exists, the file you copy will replace the existing file
- Examples
  - “docker cp a73450b64455:/home/testfile.txt /documents/”
  - “docker cp my-container:/home/testfile.txt /documents/textdocuments/”
  - ”docker cp /documents/examples/textfile.txt my-container:/home/textfile1.txt”

2 Docker Compose

Docker compose is a tool which allows you to work with multiple docker containers in an efficient way. Especially when it comes to building the different neccesary images and networks and running and stopping several containers. To use docker compose you need to write a docker-compose.yml file where you define the setup of your network and containers. At the moment there are several different versions of the docker-compose.yml format and some of them have different syntax. Therefore it is important to learn the version you will be using. In our explanation we will be using version 2.4.



```
version: "2.4"

services:
  web:
    image: "nginx:alpine"
    networks:
      - new

  worker:
    image: "my-worker-image:latest"
    networks:
      - legacy

  db:
    image: mysql
    networks:
      new:
        aliases:
          - database
      legacy:
        aliases:
          - mysql

networks:
  new:
  legacy:
```

- version: '2.4'
  - Here we specify which version of docker-compose.yml we will be using.
- services:
  - Defines which containers will be set up and how they will be set up. In the picture there are 3 containers; 'web', 'worker' and 'db'.
  - First in every service we have to use specify the image the service will be using. The first way to specify an image is "build: <source path>" where source path is the path to a directory containing a Dockerfile that we can build to get an image for the service. The other option is to use "image: <image name>". This works if we either have an image named <image name> built locally already or if <image name> exists on docker hub and can be pulled.
  - After this there are many options we can choose to use, much like the docker run command. We can map ports, choose and configure networks, set static IP addresses to our containers and much more.
- networks:
  - Here we can define our networks. The networks we define here are the ones we can add our containers to in the docker-compose.yml file.
  - NB: You can't have different networks with subnets that overlap. If that happens you will have to remove one of the networks. You also can't give several containers the same static IP address. You can either remove all docker networks not currently used by a container with the 'docker network prune' command or delete a specific network with the command 'docker network rm <network-name>'.

2.1 docker-compose build

- Builds the images given in the docker-compose.yml file.
- Syntax: 'docker-compose build'
  - To use the command like this you need do be in the directory where the docker-compose.yml file is. This holds for all the docker-compose commands

2.2 docker-compose up

- Will run the containers and start the networks defined in the docker-compose.yml file.
- If the images needed for the containers haven't been built already they will also be built with this command. But they will not be rebuilt even if the images get changed. This can be achieved with the command below or by building the images again with the 'docker-compose build' command.
- Syntax: 'docker-compose up --build'
  - Using --build will make it build the images again if Docker discovers that any of the images have been changed.

2.3 docker-compose down

- Stops and removes the containers and networks defined in the docker-compose.yml file.
- Syntax: 'docker-compose down'

3 Other

3.1 SSH

- SSH is a program for logging onto a remote computer and executing commands on the remote computer. It is intended to provide secure encrypted communications between to untrusted hosts over an unsecured network.
- Need to have set of keys for it to function.
- Syntax: 'ssh <username>@<ip-address>'
  - Will be asked if you trust the unit you try to gain access to (yes/no)
- Example
  - "ssh root@10.11.10.5"

3.2 SCP

- A tool to copy files to and from a remote unit through SSH.
- Stands for 'Secure Copy'
- Syntax: 'scp <username>@<ip-address>:<file-path> <destination-path>'
  - The first part of the command is similar to SSH. After you have chosen where to "log in" you write the path to the file you want copied and then where you want to place it on your own computer.
- Example
  - "scp root@10.11.10.5:/home/testfile.txt /documents/"

3.3 Wireshark

- A tool to analyse data packets. It has many different options for filtering and analysing packets.
- If you want a short introduction to what Wireshark has to offer, we recommend this video.

3.4 tshark

- A tool to capture packets that can later be analysed in wireshark.
- Syntax: ‘tshark -w <filename>’
  - Using -w allows us to write the captured packets to a file.
  - It is possible to filter packets during capturing but we recommend rather filter later in wireshark. This way you don’t accidentally filter out something you need.
  - When we make a file that tshark will write to we name it filename.pcap. pcap stands for packet capture and is a file format like .txt is.

3.5 traceroute

- A tool that we use in this course to show the different ”jumps” packets do when routed from one machine to another destination.
- Syntax: traceroute <IP address/url>
  - IP address/url is the ip address or the url of the destination.
- This video explains how traceroute works.