Docker

TP1

2º Semestre 2019/2020

Gestão e Virtualização de Redes

4º Ano MIEI

Fábio Gonçalves

Computer Communications and Networks

Departamento de Informática, Universidade do Minho

This work should allow to acquire the basic knowledge to work with Docker virtualization. The main goal of this work is to obtain the tools needed for:

- Create, delete, and manage docker containers;
- Create docker images;
- Use docker-compose;
- Create automated builds.

Docker Installation

Docker can be installed in any operating system, although, in windows, it is only possible to so in the Pro version. The guides presented here were made in an Ubuntu machine but, should be platform agnostic.

Docker can be installed by following one of the following guides, depending on your OS:

- Linux - https://docs.docker.com/install/linux/docker-ce/ubuntu/
- Mac - https://store.docker.com/editions/community/docker-ce-desktop-mac/plans/docker-ce-desktop-mac-tier?tab=instructions
- Windows - https://store.docker.com/editions/community/docker-ce-desktop-windows/plans/docker-ce-desktop-windows-tier?tab=instructions

After installation do test the container by running **docker run hello-world.**

DO NOT FORGUET TO UPDATE THE PERMISSIONS

sudo usermod -aG docker

Basic Commands

There is a set of basic commands that allow to do most of the managing operations in docker: create, remove and list containers.

Some of these commands are:

- **docker run** – execute a command in a new container:
  - **docker run ubuntu:latest**;
- **docker exec** – execute a command in an existing container:
  - **docker exec existing_container ls**;
- **docker stop** – stop the *container*;
- **docker start** – start a stopped *container*;
- **docker rm** – delete a stopped *container*;
- **docker images** – list all existing images;
- **docker ps -a** – list existing *containers*;
- **docker logs** – see the *container logs*;
- **docker rmi** – delete an image.

1 – Containers Basics

Docker containers are created from pre-built images. These can be obtained directly from the docker hub repository(https://hub.docker.com) or manually created using a Dockerfile.

1.1 – Managing docker images

The docker command allows to pull, delete or create new images. For this exercise, an image is going to be pulled from docker repository. To do so, choose a Linux images from the https://hub.docker.com website. Them download it by issuing:

docker pull image_name:tag

The previous command receives the **image_name** and the **tag** as arguments. The **image_name** is the name of the image to be downloaded. The **tag** is the version of this image. If no tag or the keyword **latest**, is used, the latest image will be downloaded.

To list the images downloaded images, use the command:

docker images

These can be deleted with the command:

docker rmi image_id

1.2 – Managing containers

Using one of the downloaded images, is now possible to create a container from it. In this example, a Linux container is going to be created. The issued command will open a terminal inside the container. This will allow to explore the container as normal operating system. In this step the image downloaded should be used.

docker run -it ubuntu:latest /bin/bash

The containers state can be seen with the command:

docker ps -a

One of the fields in the result from the command above is the container name. If none was assigned, the container will have a random name. Using the name in table, the container may be stopped and deleted:

docker stop container_name

docker rm container_name

To assign a container name, facilitating its management, the flag **--name** should be added when creating the container:

docker run --name test -it ubuntu:latest /bin/bash

In the previous examples, all the containers have been created using docker run. This enables to simultaneously create container and open a shell. The next command can be used to open a shell in already running container.

docker exec -it nome_do_container /bin/bash

In this example, the container name will be the previously created **test.** After successful finishing this example, the container should be stopped and removed. Then, create a new container using an image that has not been yet downloaded, another Linux flavor, for example.

1.3 – Persistence

Container data is not persistent. Meaning that all container data is removed when it is deleted.

There are three ways of persisting container data:

- **Volumes:** This type of persistence allows containers to share data between the containers. These are created in the host system and managed directly by docker;
- **Bind Mount:** These allow containers to mount directories in the host system. Thus, enabling the access to the data directly from the host. Commonly used to share data between host and containers;
- **Tmpfs:** These are only stored in memory and never on the host disk;

1.3.1 – Persistency – Volumes

To use docker volumes, the first step is creating the volume itself:

docker volume create test-vol

Then, the volume can be seen with the following commands:

docker volume ls

docker volume inspect test-vol

The first command shows the existing volumes. The next, allows to see all the characteristics of the selected volume. Docker volumes allow the usage of several driver. Their usage can be seen in the docker documentation.

To remove a volume:

docker volume rm test-vol

To exemplify the capabilities of docker volumes, lets start by creating a new volume:

docker volume create new-test-vol

Then, a container a container will be created with mounting the volume on the container directory /home/test.

docker run --name test-vol-container --mount source=new-test-vol,target=/home/test

The above command will open a shell terminal inside the container. Then, a file should be created in the mounted directory:

echo “test 123” >> /home/test/test.txt

To check if the persistence is working, remove the container and create it again with the same command.

If in another terminal the next command is run:

docker run --name test-vol-container-2 --mount source=new-test-vol,target=/home/test2

It is possible to see that this file exists in also in this new container. This time in the directory /home/test2. The content of the directories is shared between the containers.

1.3.2 – Persistence – Mount bind

To directly mount a host directory in the container, issue the following command (Adjust for your user and system):

docker run --name test-mount-container -v /home/nome_do_user/test-dir:/home/test -it ubuntu:latest /bin/bash

It is possible to see that the system host has now a new directory called **test-dir.** If a file is created in this directory, it will also exist inside the container. Executing the following command, it is possible to see the newly created files inside the container:

ls /home/test

1.4 – Port Mapping

A container is a closed system without network connection from the outside. However, it is possible to map internal ports to the host ports. To do so, the **-p** flag is used:

docker run -it --name test-port -p 8888:9999 ubuntu:latest /bin/bash

Using the netcat tool is possible to check this functionality. To do so, the next commands can be executed:

apt update

apt install netcat

nc -l -vv -p 9999

And, in the host system, execute:

wget localhost:8888

In the container, it should be possible to see the connection and the received data. Thus, it is possible to see that the command mapped the docker internal port 9999 to the host 8888. However, it does not allow the communication using udp packets. To do so, the following command can be used:

docker run -it --name test-port-udp -p 8888:9999/udp ubuntu:latest /bin/bash

1.4 – Networking

There are several types of docker networks. However, **bridge** connections are used by default. The command to list all the docker networks is:

docker network ls

By default, 3 networks should exist: **bridge, host** and **null. Bridge** network allows containers to be connected between themselves. The host network is used for the container to connect to the **host**. Finally, the **null** network exists to allow a container to run without any network device.

To test the docker networks, firstly two containers are going to be created:

docker run -dit --name test-net-1 ubuntu:latest /bin/bash

docker run -dit --name test-net-2 ubuntu:latest /bin/bash

In this case the flag **-d** is used with the flags **-it**. It allows the container to be started in background in daemon mode. To connect to this container the following command may be used:

docker attach test-net-1

After creating the containers, it is possible to check the network configuration. The following command is used to do so:

docker network inspect bridge

Now it is possible to compare the configurations obtained with the previous command and issuing **ipconfig** inside the container.

With a shell terminal inside the container, it is possible to see that it is a connection to the external network and to the internet. It can be tested by pinging google, for example.

However, if trying to ping between container by their name, it is not possible:

docker attach test-net-1

ping test-net-2

To do so, they should have been connected to the same network, as in the next example:

docker network create user-net

Them the containers are connected to the network using the flag **--network:**

docker run -dit --name test-user-net-1 --network user-net ubuntu:latest /bin/bash

docker run -dit --name test-user-net-2 --network user-net ubuntu:latest /bin/bash

After inspecting this new network, it can be seen that it is has the configuration for the two containers. The connection can be tested with:

docker attach test-user-net-1

ping test-user-net-2

The networks may be deleted with the following command:

docker network rm network_name

2 – docker-compose

Docker-compose allows to define and run several configurations and also configure the applications or service that it will start. Thus, only command one command is needed to start all the containers. The installation instructions can be found in https://docs.docker.com/compose/install/#install-compose.

Docker-compose basic commands are:

- **docker-compose up**: allows to initiate the containers
- **docker-compose stop**: terminates all running containers;
- **docker-compose rm**: deletes the containers containers;
- **docker-compose logs**: to see the logs of all the services described in the docker-compose.

Figure 1 shows an example of a docker-compose.yml. To test it, create a file called **docker-compose.yml** and copy all the contents from Figure 1.

```
version: '3'
services:
  service1:
   image: httpd:latest
   ports:
     - "8080:80"
   volumes:
     - httpd-vol:/usr/local/apache2/
   networks:
     - container-net
volumes:
  httpd-vol:
networks:
  container-net:
```

Figura 1-docker-compose.yml

This file has the version of the docker-compose to be used, the list of services to be run and their configuration, the volumes and networks. In each service is its configuration, where all the parameters needed to each service.

After creating the file, in the same directory where the file is, the following command should be run:

docker-compose up -d

This command initiates all the containers in the docker-compose. The flag **-d** means that the containers will be sent to the background in daemon mode.

After the containers are started, in the host a browser should be open in the url http://localhost:8080. A web service should be running in the port 8080.

Then, the commands **docker ls, docker network inspect, docker volume ls** and **docker volume inspect,** should be used to check all the container parameters.

3 – Dockerfile

Dockerfile is a text file that contains a list of commands describing how a docker image will be created. This will be run sequentially.

Usually, the first line indicates with image is going to be used. Figure 2 shows an example of a dockerfile.

```
FROM ubuntu:latest
EXPOSE 8888
RUN apt-get update && apt-get -y install netcat
VOLUME /home/output
ENTRYPOINT ["/bin/nc" , "-k", "-l", "-p", "8888", "-vv"]
```

Figura 2- Dockerfile

The first line indicates that the base image that is going to be used is an Ubuntu. The, the **EXPOSE** key word means that the port 8888 will be exposed. This means that latter this can be mapped to a host port.

The next line serves to install the netcat tool. Then, it is created the **/home/output** volume. This last step allows to create a volume that can be later be persisted.

Finally, the **ENTRYPOINT** key word is the command that will be executed when the container is initiated.

To test the created Dockerfile, first the image must be created. In the Docker file directory execute:

docker build . -t test:0.1

The **-t** flag allows to assign a name to this image. In this case **test:0.1.**

After creating the image, a container can be created. To do so:

docker run --name test-dockerfile -p 8888:8888 -d test:0.1

This allows a container to be initiated in daemon mode. The container will have the port 8888, as indicated in the Docker file. Then execute the following command:

docker logs test-dockerfile

This allows to see the container logs. In another terminal now run:

wget localhost:8080

In the docker logs, it should be possible to see that the connection was made successfully.

4 – Automatic builds

Docker hub allows to automatically build images. To do so a dockerfile must be provided. To do so, it is necessary to create a github and docker hub account. The steps needed to create an automatic build are:

1. Create a github project:
2. Pushing the docker file to github;
3. Create a project in docker hub;
4. Linking both projects;

4 – Practical Exercises

1. How to create a container from a Linux alpine that has the container 9999 port mapped in the host 8668= This should also have a bind mount, mounting the container /home/internal_dir in the hosts /home/user/docker_dir/.
2. How to create the volume “my-volume-1”?
    a. What is the volume mountpoint?
    b. Which driver is being used?
3. Create two basic containers (They should not be attached to any manually created network).
    a. Is it possible to inspect the bridge network and find their IP addresses?
    b. Is it possible for the containers to communicate among themselves using their names?
    c. And with their IPs?
4. What is the command to create the network “my-network-1”?
    a. How to attach two containers to that network?
    b. Which relevant parameter is possible to found about that network?
    c. Can the containers ping one another using their name?
5. Which were the results obtained from the commands docker network ls, docker network inspect, docker volume ls and docker network inspect in section 2 of this document? What conclusions is possible to take from the result of these commands?
6. Create a docker-compose that starts at least two services:
    a. These should be in the same docker network;
    b. These should share the same docker volume:
    c. One of the services should also have one bind mount;
    d. One of the services should have their 9999 port exposed in the 8888 host port;
    e. Using the inspect and ls commands, what conclusions can you take about their results?
7. Create a Docker file that:
    a. Have some tool installed. Any that is chosen by the student. Extra points for any application that is listening from outside requests (web server, for example);
    b. Has a volume, for later persistence;
8. Create an automatic build. The docker file built in the previous exercise may be used.

The exercises should be answered in a word file, adding, whenever needed screenshots. In the questions 5 and 6 the docker-compose and Dockerfiles should also be added. For question 7, only the docker hub repository url needs to be indicated.

A zip file with all the content is to be sent via blackboard until March 1st.