# Docker

## What is docker and installation

install docker  
- Not going over that

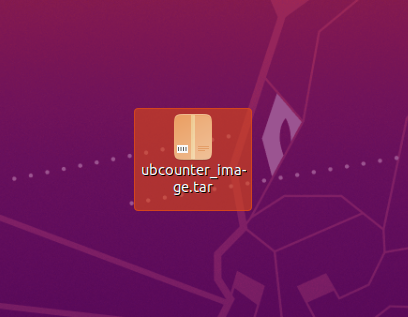
Start a hello-world container  
- docker run hello-world

Look what following commands do:

* docker system info  
  - returns information on about docker on the system
* docker system df  
  - Shows stats of images, containers, local volumes and build cache
* docker system events  
  -

## Docker Hub & Images

We will download this image: <https://drive.google.com/file/d/13AwHEbt390zUKzJxNNUoTiTYDQvJXH2x/view>  
This we will need to import to a docker image and run it, we will use the option -it

When we download receive a .tar file

In order to use this file we will do  
- docker load < ubcounter\_image.tar

We then do   
- docker run -it ubcounter  
We add -it behind run so we can stop the counter with CTRL + C

If we want to pull ubuntu 18.04, CentOS 7.9.2009 and the latest fedora we can use the following:

* docker pull ubuntu:18.04
* docker pull centos:7.9.2009
* docker pull fedora:latest

To save our image we can use  
- docker save -o ubuntu.tar ubuntu

now we need to enter the docker container to get data out of it in order to do this we have to run our Ubuntu container using  
- docker run ubuntu:18.04  
Then we have to find the name of our running container using docker ps  
- docker ps

then we can enter the containers bash using  
- docker exec -it suspicious\_ellis /bin/bash

When we entered out bash we can use  
- cat /etc/\*-release   
to get our distribution information

We can see what is running the shell using  
- echo “$SHELL”

To get our kernel version we can do   
- uname -r

Save the image with:  
- docker save <image>

## Working with containers

We will work with a centos container

For this we first need to run our container  
- docker run -itd centos

* Give the container the name, we can do this on our first run or with rename  
  - docker run –name MijnCentOSContainer centos  
  - docker container rename MijnCentOSContainer mcc
* Check containers  
  - docker container ls -a  
  - docker container ls
* Access the docker container shell with  
  - docker exec -it mcc /bin/bash
* Show centos version  
  - cat /etc/os-release
* compare both kernels with  
  - uname -r
* create a test in the home folder  
  - cd ~  
  - vi test  
  - hello this is a test  
  - cat test
* Leave container without stopping  
  - CTRL + P –> CTRL + Q
* When accidentally doing CTRL + D you can restart your container with  
  - docker start mcc
* Reattach to container  
  - docker exec -it mcc /bin/bash
* Leave container without stopping  
  - CTRL + P –> CTRL + Q
* Pause the container  
  - docker pause mcc
* Stop the container  
  - docker stop mcc
* Start the container  
  - docker start mcc
* Reattach to the container  
  - docker exec -it mcc /bin/bash
* look at your test folder, it still exists  
  - cd ~  
  - cat test
* Leave container and stop it  
  - CTRL + D  
  - docker stop mcc
* Remove container  
  - docker rm mcc

Create 3 CentOS Containers that stop immediately

* docker run –name COS1 centos
* docker run –name COS2 centos
* docker run –name COS3 centos

Create 3 CentOS Containers that keep running

* docker run -itd –name COR1 centos
* docker run -itd –name COR2 centos
* docker run -itd –name COR3 centos

When going through ps you can filter on:

|  |  |
| --- | --- |
| id | Container’s ID |
| name | Container’s name |
| label | An arbitrary string representing either a key or a key-value pair. Expressed as <key> or <key>=<value> |
| exited | An integer representing the container’s exit code. Only useful with --all. |
| status | One of created, restarting, running, removing, paused, exited, or dead |
| ancestor | Filters containers which share a given image as an ancestor. Expressed as <image-name>[:<tag>], <image id>, or <image@digest> |
| before or since | Filters containers created before or after a given container ID or name |
| volume | Filters running containers which have mounted a given volume or bind mount. |
| network | Filters running containers connected to a given network. |
| publish or expose | Filters containers which publish or expose a given port. Expressed as <port>[/<proto>] or <startport-endport>/[<proto>] |
| health | Filters containers based on their healthcheck status. One of starting, healthy, unhealthy or none. |
| isolation | Windows daemon only. One of default, process, or hyperv. |
| is-task | Filters containers that are a “task” for a service. Boolean option (true or false) |

We can also format our output

| **Placeholder** | **Description** |
| --- | --- |
| .ID | Container ID |
| .Image | Image ID |
| .Command | Quoted command |
| .CreatedAt | Time when the container was created. |
| .RunningFor | Elapsed time since the container was started. |
| .Ports | Exposed ports. |
| .State | Container status (for example; “created”, “running”, “exited”). |
| .Status | Container status with details about duration and health-status. |
| .Size | Container disk size. |
| .Names | Container names. |
| .Labels | All labels assigned to the container. |
| .Label | Value of a specific label for this container. For example '{{.Label "com.docker.swarm.cpu"}}' |
| .Mounts | Names of the volumes mounted in this container. |
| .Networks | Names of the networks attached to this container. |

We need the shortID of all running containers:

* docker ps -a –filter ‘status=running’ –format “table {{.ID}}”

We now want the short-d of containers running in the background

* docker ps –format “table {{.ID}}”

Remove COS1

* docker rm COS1

remove COR1 without stopping it first  
- docker rm -f COR1

Remove all stopped containers  
- docker container prune

## Docker container data & layers & OverlayFS

Create a docker container named ubuntu\_ping  
- docker run -it –name “ubuntu\_ping” ubuntu /bin/bash

Add the ping apt to the container  
- apt update && apt -y install iputils-ping  
- exit

Shows what changed between in the container  
- docker diff “ubuntu\_ping”

Commit your new container to a new image  
- docker commit ubuntu\_ping mbp

We can now see our new image with ping on it with  
- docker images

we can also check the history of the commando’s used  
- docker history mbp

Now you can run the ping commando on the image  
- docker run mbp ping -c 8.8.8.8

## Processes & Logs & Cgroups & Namespaces

Start an apache container  
- docker run -itd httpd

we forgot to name our container so we can do  
- docker rename focused\_lehmann apache

Check logging  
- docker logs apache  
Inside logging the ip 192.17.0.3 is given when searching this you get a website with “It works!” on it.

Check the processes  
- docker top apache

We can use docker inspect to also find our ip address  
- docker inspect apache | grep IPAddress

Now we will do a stress test in which we will divide the cpu power of 1 cpu in 4 cpu shares are equal to 1024 so we take our values from there  
- docker run -d --name stress1 --cpuset-cpus 0 --cpu-shares 337 benhall/stress  
- docker run -d --name stress2 --cpuset-cpus 0 --cpu-shares 337 benhall/stress  
- docker run -d --name stress3 --cpuset-cpus 0 --cpu-shares 168 benhall/stress  
- docker run -d --name stress4 --cpuset-cpus 0 --cpu-shares 168 benhall/stress

We can also run a nginx container and surf to the website with curl  
- docker run nginx  
- curl localhost

## Shell access in container

we will first create an apache container   
- docker run -itd --name ap httpd

to get shell access we can do nsenter, docker-enter or docker exec  
- I have no clue about nsenter  
- docker-enter a3b2482c3553  
- docker exec ap

## Dockerfile and pushing to Docker hub

## Container volumes

Create a directory named dockerfiles  
- mkdir dockerfiles

In this directory place a file named testfile  
- touch dockerfiles/testfile  
- echo “This is a test” > dockerfiles/testfile

Access this folder through an ubuntu container  
- docker run -it -v ~/dockerfiles/:/containervol ubuntu /bin/bash  
In Ubuntu bash we can then change the text  
- echo “This the second test” > containervol/testfile

## Container networking

We will first create an apache image we will use in this exercise  
- mkdir myapache  
- cd myapache  
- vi Dockerfile  
| # Image for port forwarding  
| FROM networking  
| LABEL maintainer=”hello.hello@hello.bye”  
| RUN apt update && apt -y install apache2  
| EXPOSE 80  
| ENTRYPOINT [“apache2ctl]  
| CMD [“D”,”FOREGROUND”]  
- docker build -t myapache  
- docker images

We then prepare some other directories  
- mkdir html  
- echo "ELLO YOUR OCMPUTER HAS VIRUS" > html/index.html  
- mkdir apachelogs

We want to run our apache container at 4444:80  
- docker run -d -p 4444:80 --name apacheoefening myapache

# Ubuntu

In what file are multi-user groups defined?  
- /etc/group

/etc/passwd us used to store userdata, what command do we use to get more info about the data inside?:  
- google it, I can’t find the command  
What are the 7 different fields?:

1. Username
2. X (password)
3. User id
4. Primary group id
5. Description
6. Home directory
7. Shell

Passwords are stored in  
- /etc/shadow

Create a group named testgroup with id 200  
- sudo su -  
- groupadd -g 200 testgroup

Change the group name to pcactive  
- groupmod -n pcactive testgroup

Remove this group  
- groupdel pcactive

Create this user, for this we can use usermod:  
- useradd -m -u 201 -g 100 -c “Testuser Jan” -s /bin/bash jan

We now need to set jan’s password:  
- passwd jan

To get your information use  
- cat /etc/passwd | grep -w jan  
The passwd line containing jan

Log in as jan:  
- su jan

What do you see when you do ls /root  
- permission denied

Go back to root  
- exit

Give Jan the root group  
- usermod -a -G root jan  
We can check this with  
- groups jan

Nothing has changed in /etc/group and its still not possible to enter /root, I have no clue why

now remove jan  
- userdel jan  
  
Nothing has changed in /etc/group

the directory /home/jan still exists