Demo - Introduction to containers

Pull a container

```
docker image ls                # no container images
docker search ubuntu           # all images with ubuntu on the name
docker pull ubuntu             # pull ubuntu container image
docker image ls                # container image listed
```

Run a container

Simple base image

```
docker run --name mycontainer -it ubuntu /bin/bash
> exit                          # notice the container has stopped
docker start mycontainer        # the conainer starts but we're not attached to it
docker attach mycontainer       # we're attached to the bash process within the container again
```

Manipulating our container to get something more interesting: a web server

- From inside our container

```
> apt-get update
> apt-get install -y nginx curl
> service nginx start
> ps -ef | grep nginx
> curl localhost
> echo "daemon off;" >> /etc/nginx/nginx.conf
> exit
```

- From outside our container

```
docker commit mycontainer jpalma/mywebserver
# choose your <repository name/image name> instead of jpalma/mywebserver
docker rm mycontainer
docker run -d --name mywebserver --rm jpalma/mywebserver /bin/bash -c 'service nginx start'
curl localhost:8080           # nothing listening locally
curl localhost:80             # nothing listening locally
# we have nginx running now, but can't access it until we're "in" the container:
# like this:
docker exec -it mywebserver /bin/bash
> curl localhost              # nginx is serving us data
> exit                        # notice the container doesn't stop
docker stop mywebserver       # container stops and gets removed "--rm"
```

- exposing a port to the outside:

```
curl localhost:8080            # nothing listening on 8080
docker run --name mywebserver -d --rm -p 8080:80 nginx
curl localhost:8080            # nginx answers from terminal on local machine
```

- attaching to the running process:

```
# as opposed to starting a new process with "docker exec", we can just attach also:
docker attach mywebserver
# in a different terminal window, generate requests:
curl localhost:8080
```

```
# see log entries pop up on attached terminal
> <CTRL C>
```

- inspecting the container logs:

```
# we can now also inspect the logs from a container:
docker logs mywebserver        # one log entry
curl localhost:8080            # hit the server once more to generate another log entry
docker logs mywebserver        # two log entries
docker stop mywebserver        # stop the nginx container
```

Mapping the file system

```
mkdir tmp
vim ./tmp/hello.html               # create a static html file to serve from nginx
docker run --name nginx -d \
           --rm -p 8080:80 \
           -v $PWD/tmp:/usr/share/nginx/html \
           nginx
docker exec -ti nginx /bin/bash    # go inside container and show content of
                                   # \usr\share\nginx\html
curl localhost:8080/hello.html     # content is served by nginx
docker stop nginx                  # stop the nginx container
```

Building a container

Instead of mapping files from the host into a container, a more sustainable solution that allows a container to capture all its dependencies is to simply build your own. This is easier than one might expect.

> Execute below steps in an empty folder.

Create content to copy into container

```
# prepare content for nginx
echo "hello world" > index.html
```

Create a Dockerfile

Create a file called Dockerfile with the contents below:

```
FROM nginx
COPY ./index.html /usr/share/nginx/html
```

Build the container

```
# replace "xstof" with your own docker hub repository name
docker build -t xstof/hello .
# show the image
docker image ls
# push the image to docker hub
docker push xstof/hello
```