A - Running Docker containers

Preparation

Run the Docker exercises on the **Docker** VM. Launch the machine with `sudo virsh start c8-docker`, obtain its IP address with `sudo virsh net-dhcp-leases default` and ssh to the VM with `ssh 192.168.122.21` (assuming this is the IP address).

Run `sudo docker info` to check if Docker is ready. You may have to launch the docker daemon with `sudo systemctl enable docker --now`. Double-check with
`systemctl status docker`.

1. My first Docker container

You may have to enable and launch the docker daemon.

```
$ sudo systemctl enable docker --now
```

A first look at an image description

Use your PC's browser to navigate to http://hub.docker.org/_/wordpress. This page describes the official WordPress Docker image.

Note that several image versions are available. What software is contained this WordPress image?

Scroll down to the section **What is WordPress** and have a look at the description.

Scroll to the section **How to use this image**. Several `docker run` commands are listed. Try to make sense of them (but don't get frustrated if you don't understand). Also try to make sense of the environment variables named `WORDPRESS_<something>`.

It seems that WordPress requires a database; is it contained in the wordpress image?

Scroll to the section **Docker Secrets**. What do you think Docker Secrets are for?

View the example `stack.yml` and try to understand what it describes. What are the environment values for? Which images are used? Where is the database stored?

Read the second paragraph of **Adding additional libraries / extensions**. What is needed to add extensions to this WordPress installation?

Copy the image to your lab machine

The required command is displayed right at the beginning of the Docker Hub Wordpress page. Watch what happens.

```
$ sudo docker pull wordpress
Using default tag: latest
latest: Pulling from library/wordpress
f8416d8bac72: Pull complete
...
ca00a0f9ead6: Pull complete
720eba734fe9: Pull complete
Digest: sha256:33ed1fa01dc28b65407a5932791d55deba87b819cb54a83c60c9770d3aeebb71
Status: Downloaded newer image for wordpress:latest
docker.io/library/wordpress:latest
```

Notice: The image seems to consist of several components, labeled with hexadecimal numbers, that are downloaded, then unpacked in parallel.

What could the word *latest* mean in the context of this image pull?

List images.

```
$ sudo docker image ls
```

Notice that *latest* appears here as well.

Launch a container from the image

Sample commands are on the Docker Hub WordPress page. You will execute a simpler version of them.

```
$ sudo docker run --name wpfg wordpress
```

Read the messages that are displayed. Where do they come from?

Interrupt the command with control-C and run it again, this time adding an option.

```
$ sudo docker run -d --name wp wordpress
```

What does the `-d` option do? Hint: Use the `docker help run` command to find out.

View a background container's output

Containers tend to run in the background. To see their output, use the logs command.

```
$ sudo docker logs wp
```

The `-f` option has the same effect as `tail -f`.

```
$ sudo docker logs -f wp
```

This is a useful troubleshooting tool.

List and remove containers

```
$ sudo docker container ls
$ sudo docker ps
```

These two commands are identical. They list all *running* containers. The output is easier to read in a wide terminal.

Which ports are listed for container `wp`?

```
$ sudo docker container ls -a
```

This also lists **stopped containers**, such as `wpfg`, which you interrupted.

You need identify a container to stop it, either using its **name** `wpfg` or its **ID**. You don't have to provide the full ID; just enough digits to make it unique. For example, if you have three containers as follows:

```
$ sudo docker container ls -a

CONTAINER ID   IMAGE        COMMAND                  ...
40fc42ec6803   wordpress    "docker-entrypoint.s…"   ...
39c1be76129d   wordpress    "docker-entrypoint.s…"   ...
390246d29b8b   mariadb      "docker-entrypoint.s…"   ...
```

In this example, you can remove the first container like this: `sudo docker container rm 4`.
To remove the second container, you need three digits to make the ID unique: `sudo docker container rm 39c`.

The command below uses its name to remove the container.

```
$ sudo docker container rm wpfg
```

2. Explore a container

Explore the running container

List the container's details. While you won't understand most of them, some might make sense to you.

```
$ sudo docker container inspect wp | more
```

Obtain the container's IP Address and access the container.

```
$ sudo docker container inspect wp | grep IPAddr
$ curl -i CONTAINER_IP_ADDRESS
HTTP/1.1 302 Found
...
Location: http://CONTAINER_IP_ADDRESS/wp-admin/setup-config.php
$ curl http://CONTAINER_IP_ADDRESS/wp-admin/setup-config.php | more
```

You should see a line "<title>WordPress › Setup Configuration File</title>". This shows that you were able to access WordPress.

Note that the container's IP address is only accessible from the host. You can't use this container as an internet-facing blog.

Explore the container's data

The container has a filesystem but keeps the web site code on a volume. Find the volume in the output of `docker container inspect`, or run this command:

```
$ sudo docker container inspect wp|grep -A3 volume
```

Record the volume's name, its source and its destination.

List all volumes and the details of this container's volume.

```
$ sudo docker volume ls
$ sudo docker volume inspect VOLUME_NAME
```

cd to the mountpoint and list the files there. What do you see?

Run commands in the container

List the web site directory.

```
$ sudo docker exec wp ls /var/www/html
```

Add the `-t` option, which runs the command on a terminal that it creates. `ls` uses a different output format when it detects that its standard output is a terminal:

```
$ sudo docker exec -t wp ls /var/www/html
```

To run interactive commands in the container, you need the `-i` option:

```
$ sudo docker exec -it wp bash
```

Feel free to explore the filesystem in the container. Then exit the contained `bash`.

Which processes run in the container?

```
$ sudo docker exec wp ps -ef
```

Explore the container's processes on the host

Container processes are visible on the host. List Apache processes on the host as well.

```
$ ps -ef | grep apache2
```

Their process IDs are different from the process IDs in the container.

Which process belongs to which container? There is no simple method of answering this question, but the container details contain the very first process. List its children and its parent.

```
$ sudo docker inspect wp --format '{{ .State.Pid }}'
5332
$ ps -ef|grep 5332
root      5332  5311  0 12:24 ?        00:00:00 apache2 -DFOREGROUND
www-data  5382  5332  0 12:24 ?        00:00:00 apache2 -DFOREGROUND
www-data  5383  5332  0 12:24 ?        00:00:00 apache2 -DFOREGROUND
...
$ ps -ef|grep 5311
```

```
root      5311     1  0 12:24 ?        00:00:00 /usr/bin/containerd-shim-runc-v2 -namespace moby -id
39c1be76129db59cb94b08e52d227e76c6faac54ade214183555417266581328 -address
/run/containerd/containerd.sock
root      5332  5311  0 12:24 ?        00:00:00 apache2 -DFOREGROUND
```

List the files in the web site directory. This list should look familiar. Which files are they? Why?

Access the container from an external machine

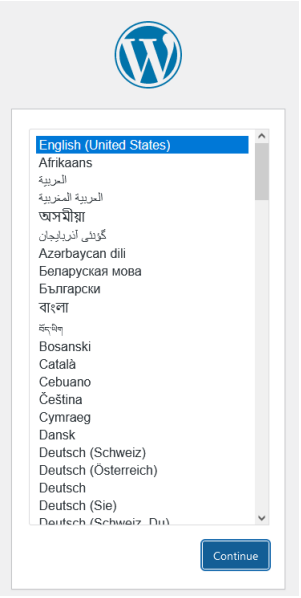
The container's IP address is not routable. It can only be accessed from the container host. To access the WordPress web site from your Desktop's browser, some network magic needs to happen.

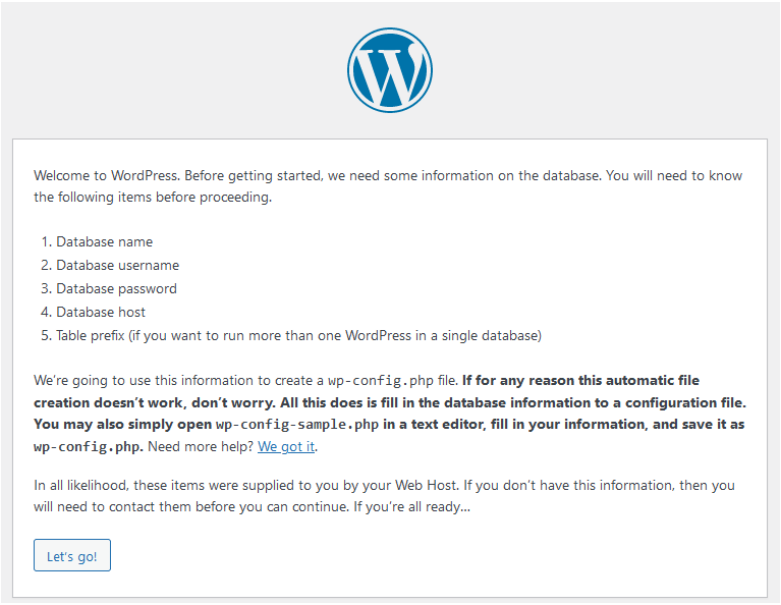
Delete the `wp` container and run a new one, publishing its network port.

```
$ sudo docker container rm wp
$ sudo docker container rm wp --force
$ sudo docker run -d --name wp -p 8000:80 wordpress
```

Open a browser window and enter `192.168.122.100:8000`. You should see a page asking you to select a language for WordPress setup. Select **English** and click **Continue**.

The next page tells you that database access information is required. You can't set up your WordPress site without a database. You will do this in the next section.





3. Run two cooperating containers

WordPress requires a SQL database, which is not included in the wordpress container. You will launch a MariaDB container and a new wordpress container and link the two together.

First remove the wordpress container with `sudo docker rm wp --force`.

Read documentation about the MariaDB container

Open https://hub.docker.com/_/mariadb on your PC and scroll down to the **Environment Variables** section. The variables documented here allow you to create a database and set its credentials.

Launch a MariaDB container and get its network properties

Launch a MariaDB container, setting its database name to *wordpress*, its user account to *wpuser*, and the passwords for *root* and *wpuser* to *pw*. This time, don't explicitly pull the image.

```
$ sudo docker run -d --name wpdb \
                  -e MARIADB_ROOT_PASSWORD=pw -e MARIADB_DATABASE=wordpress \
```

```
                -e MARIADB_USER=wpuser -e MARIADB_PASSWORD=pw mariadb
```

Obtain IP address and port.

```
$ sudo docker container inspect wpdb | grep IPAddr
$ sudo docker container ls
```

Or get both from the `inspect` output, for example this way:

```
$ sudo docker container inspect wpdb | grep -A2 -e IPAddress -e ExposedPorts
```

Alternatively, use Docker's native output formatter. Since Docker is written in Go, output can be formatted using Go templates. They are not easy to use, but you can try this:

```
$ sudo docker inspect wpdb --format '{{ .NetworkSettings.IPAddress }} {{ .NetworkSettings.Ports }}'
```

The port should be 3306.

If you are interested in Go templates, a quick and easy Docker-oriented overview is found in a Stackexchange answer.

Test the database container by accessing it

If the mysql client is not installed, use `apt update && apt install mariadb-client -y`.

```
$ mysql -h CONTAINER_IP_ADDRESS -u wpuser -ppw
MariaDB [(none)]> use wordpress
Database changed
MariaDB [wordpress]> show tables;
Empty set (0.000 sec)
MariaDB [wordpress]> exit;
Bye
```

This shows that the MariaDB container has set up both database and user account as requested by your environment variables.

Add a WordPress container and explore it

You want the WordPress container to access the database. You will use environment variables to send it access information. To see which variables may be relevant, feel free to re-read the How to use this image section at https://hub.docker.com/_/wordpress.

So far, you allowed the wordpress container to set up an anonymous volume. This time, specify the host directory to which the volume will be mapped.

```
$ mkdir ~/wpvol
$ sudo docker run -d --name wp -v ~/wpvol:/var/www/html -p 8000:80 \
                  -e WORDPRESS_DB_HOST=MARIADB_IP_ADDRESS -e WORDPRESS_DB_USER=wpuser \
                  -e WORDPRESS_DB_PASSWORD=pw -e WORDPRESS_DB_NAME=wordpress \
                  wordpress
```

Confirm the volume mapping.

```
$ sudo docker inspect wp | grep Source
                "Source": "/home/nobleprog/wpvol",
```

View the variables inside the container.

```
$ sudo docker exec -it wp bash -c export
...
declare -x WORDPRESS_DB_HOST="172.17.0.3"
declare -x WORDPRESS_DB_NAME="wordpress"
declare -x WORDPRESS_DB_PASSWORD="pw"
declare -x WORDPRESS_DB_USER="wpuser"
```

Check the WordPress installation in the volume. First, view it from inside the container, then from the host.

```
$ sudo docker exec -t wp ls /var/www/html
$ ls ~/wpvol
```

The same file names should be listed. Which is not surprising, since these are the same files.

Check the WordPress configuration file. You could look into it with `less ~/wpvol/wp-config.php`, or this way:

```
$ grep WORDPRESS_DB ~/wpvol/wp-config.php
define( 'DB_NAME', getenv_docker('WORDPRESS_DB_NAME', 'wordpress') );
define( 'DB_USER', getenv_docker('WORDPRESS_DB_USER', 'example username') );
define( 'DB_PASSWORD', getenv_docker('WORDPRESS_DB_PASSWORD', 'example password') );
define( 'DB_HOST', getenv_docker('WORDPRESS_DB_HOST', 'mysql') );
...
```

The PHP code accesses the environment variables that you set when launching the container. This way, WordPress knows how to access the database.

Access the WordPress site from a browser

Enter 192.168.122.100:8000 in the browser's address bar. Select English. Optionally, set up the WordPress site and start blogging.

5. Use Docker's built-in DNS

So far you have manually retrieved the database container's IP address and using it as a parameter when starting the wordpress container. A more elegant solution is using Docker's DNS. You need to attach the containers to a non-default network.

Create a network

```
$ sudo docker network create wpnet
```

By default, a network is implemented by a Linux bridge named after the network's ID like this:

```
$ sudo docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
...
b6be71ef9f15   wpnet     bridge    local
```

Launch the containers

First, delete the current containers.

```
$ sudo docker rm wp wpdb --force
$ sudo docker run --network wpnet -d --name wpdb \
             -e MARIADB_ROOT_PASSWORD=pw -e MARIADB_DATABASE=wordpress \
             -e MARIADB_USER=wpuser -e MARIADB_PASSWORD=pw mariadb
$ sudo docker run --network wpnet -d --name wp -v ~/wpvol:/var/www/html -p 8000:80 \
             -e WORDPRESS_DB_HOST=wpdb -e WORDPRESS_DB_USER=wpuser -e WORDPRESS_DB_PASSWORD=pw \
             -e WORDPRESS_DB_NAME=wordpress wordpress
```

To connect the containers to the new network, add the --network option. Furthermore, you launch the wordpress container with `-e WORDPRESS_DB_HOST=wpdb` . There is no need for IP addresses.

Run a shell in the wordpress container and use *ping* to confirm that DNS works. Since *ping* is not installed, add it on the fly.

```
$ sudo docker exec -it wp bash
root@b9e7c1d002ac:/var/www/html# apt update && apt install iputils-ping -y
root@b9e7c1d002ac:/var/www/html# ping wpdb
PING wpdb (172.18.0.2) 56(84) bytes of data.
64 bytes from wpdb.wpnet (172.18.0.2): icmp_seq=1 ttl=64 time=0.231 ms
64 bytes from wpdb.wpnet (172.18.0.2): icmp_seq=2 ttl=64 time=0.115 ms
...
```

Use the browser to confirm that the WordPress web site works as expected.

Explore the network implementation

On the container host, list the bridge and the interfaces that are plugged into it. The bridge name is `br-DOCKER_NETWORK_ID`. You obtained the ID from the above `docker network ls` command.

```
$ ip address show dev br-DOCKER_NETWORK_ID
```

```
22: br-b6be71ef9f15: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:f6:34:71:cb brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.1/16 brd 172.18.255.255 scope global br-b6be71ef9f15
...
$ ip address | grep DOCKER_NETWORK_ID
19: br-xxxxxxxxxxxx: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    inet 172.19.0.1/16 brd 172.19.255.255 scope global br-fb63ccb4799d
21: veth08b9249@if20: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-xxxxxxxxxxxx
state UP group default
23: veth439a346@if22: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-xxxxxxxxxxxx
state UP group default
```

The two *veth* interfaces are parts of veth pairs. The other end of each veth pair is in a container.

```
$ sudo docker exec wpdb ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
...
20: eth0@if21: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
...
```

The string `@if21` indicates that the veth pair peer of eth0 is the interface at index 21. This is `veth08b9249` on the container host.

The wordpress container does not include the `ip` command. If you are interested, install it there with `apt update && apt install iproute2`, then run `sudo docker exec wp ip a`.

6. docker-compose

Install docker-compose if necessary

Use `docker-compose -h` to check if docker-compose is installed. If not, install it (also see the instructions on the Docker website):

```
$ sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-
$(uname -m)" -o /usr/local/bin/docker-compose
$ sudo chmod +x /usr/local/bin/docker-compose
```

Create a docker-compose file

```
$ mkdir ~/myblog
$ cd ~/myblog
```

Create a docker-compose file named `wp.yaml` with the following content:

```
$ cat > wp.yaml <<EOF
services:

  wordpress:
    image: wordpress
    restart: always
    ports:
      - 8000:80
    environment:
      WORDPRESS_DB_HOST: wpdb
      WORDPRESS_DB_USER: wpuser
      WORDPRESS_DB_PASSWORD: pw
      WORDPRESS_DB_NAME: wordpress
    volumes:
      - wordpress:/var/www/html

  wpdb:
    image: mariadb
    restart: always
    environment:
      MYSQL_DATABASE: wordpress
```

```
      MYSQL_USER: wpuser
      MYSQL_PASSWORD: pw
      MYSQL_ROOT_PASSWORD: pw
    volumes:
      - db:/var/lib/mysql

volumes:
  wordpress:
  db:
EOF
```

This file declares two containers (named *services* in docker-compose terminology) and two volumes. Although no network is declared in the file, it also creates a network to which it connects the two containers.

Notice that the `WORDPRESS_DB_HOST` variable in the `wordpress` service refers to the second service `wpdb`. Each container refers to one of the volumes and maps them to a container directory.

Launch the multi-container application

```
$ sudo docker-compose -f wp.yaml up -d
```

Verify containers, volumes and the network created by this operation.

```
$ sudo docker container ls
$ sudo docker network ls
$ sudo docker volume ls
```

How do you explain the names of containers, network and volumes?

Use the browser to confirm that the WordPress site works.

Test the application's automatic restart

Due to the `restart: always` clause in the docker-compose file, Docker restarts a container when its main process stops.

Find the main process of the wp container and kill it.

```
$ sudo docker container inspect myblog_wordpress_1 --format '{{ .State.Pid }}'
5569
$ sudo kill 5569
```

Check the container's status, and find the main process again.

```
$ sudo docker container ls
CONTAINER ID   IMAGE       COMMAND                  CREATED         STATUS
6edfd18c8aba   wordpress   "docker-entrypoint.s…"   4 minutes ago   Up 41 seconds
$ sudo docker container inspect myblog_wordpress_1 --format '{{ .State.Pid }}'
5958
```

The wordpress container has been up for just a few seconds, but was created a few minutes ago. This shows that it was restarted. Furthermore, the main process has a different process ID (this is not too surprising after killing the old process) - a new process was created.

Scale the application

docker-compose can scale a service up or down by adding or removing containers. You will scale the wordpress service (i.e. add and remove wordpress containers). Unfortunately, it doesn't work right now:

```
$ sudo docker-compose -f wp.yaml up -d --scale wordpress=3
myblog_wpdb_1 is up-to-date
WARNING: The "wordpress" service specifies a port on the host. If multiple containers for this service
are created on a single host, the port will clash.
...
WARNING: Host is already in use by another container
...
```

```
ERROR: for myblog_wordpress_2  Cannot start service wordpress: driver failed programming external
connectivity on endpoint myblog_wordpress_2 (...): Bind for 0.0.0.0:8000 failed: port is already
allocated
ERROR: for myblog_wordpress_3  Cannot start service wordpress: driver failed programming external
connectivity on endpoint myblog_wordpress_3 (...): Bind for 0.0.0.0:8000 failed: port is already
allocated
...
ERROR: Encountered errors while bringing up the project.
```

The problem is that all containers use the same port 8000. Create a new docker-compose file that specifies a host port range instead of a single port.

```
$ sudo docker-compose -f wp.yaml down
$ sed s/8000/8000-8009/ wp.yaml > wp-scale.yaml
$ cat wp-scale.yaml
...
  wordpress:
...
    ports:
      - 8000-8009:80
```

Now, Docker can choose from ten ports to launch a container.

Start the application, then scale it up and down.

```
$ sudo docker-compose -f wp-scale.yaml up -d
$ sudo docker container ls
$ sudo docker-compose -f wp-scale.yaml up -d --scale wordpress=5
$ sudo docker container ls
```

There should be three wordpress containers whose names contain numbers 1 to 5.

```
$ sudo docker-compose -f wp-scale.yaml up -d --scale wordpress=3
$ sudo docker container ls
```

Note that this application does not make much sense. Running several web servers is common, but a load balancer is needed to access them via a single IP address and port.

Clean up

```
$ sudo docker-compose -f wp-scale.yaml down
```

This command removes all containers and networks, but not volumes.

```
$ sudo docker volume ls
```

Remove all unused volumes.

```
$ sudo docker volume prune
$ sudo docker volume ls
```