Multi container setups

Task: host a Wordpress site

In this scenario, we are going to deploy the CMS system called Wordpress.

> WordPress is a free and open source blogging tool and a content management system (CMS) based on PHP and MySQL, which runs on a web hosting service.

So we need two containers:

- One container that can serve the Wordpress PHP files
- One container that can serve as a MySQL database for Wordpress.

Both containers already exists on the dockerhub: Wordpress and Mysql.

Separate containers

To start a mysql container, issue the following command

```
docker container run --name mysql-container --rm -p 3306:3306 -e
MYSQL_ROOT_PASSWORD=wordpress -d mysql
```

Let's recap what this command does:

- `docker` invokes the docker engine
- `container` manage the container part of docker
- `run` tells docker to run a new container off an image
- `--name mysql-container` gives the new container a name for better referencing
- `--rm` tells docker to remove the container after it is stopped
- `-p 3306:3306` mounts the host port 3306, to the containers port 3306.
- `-e MYSQL_ROOT_PASSWORD=wordpress` The `-e` option is used to inject environment variables into the container.
- `-d` runs the container detached, in the background.
- `mysql` tells what container to actually run, here mysql:latest (:latest is the default if nothing else is specified)

MySQL is now exposing it's port 3306 on the host, and everybody can attach to it **so do not do this in production without proper security settings**.

We need to connect our wordpress container to the host's IP address. You can get the IP by issuing the `ifconfig` command on the host.

After you have noted down the IP, spin up the wordpress container with the host IP as a variable:

```
docker container run --name wordpress-container --rm -e WORDPRESS_DB_HOST=<YOURIP>
-e WORDPRESS_DB_PASSWORD=wordpress -p 8080:80 -d wordpress
```

You can now browse to the IP:8080 and have your very own wordpress server running.

Making a container network

Even though we in two commands made the setup running in the above scenario, there are some problems here we can fix:

- We need to know the host IP to get them to talk to each other.
- And we have exposed the database to the outside world.

In order to connect multiple docker containers without binding them to the hosts network interface we need to create a docker network.

The `docker network` command securely connect and provide a channel to transfer information from one container to another.

First off make a new network for the containers to communicate through:

```
docker network create if_wordpress
```

Docker will return the networkID for the newly created network. You can reference it by name as well as the ID.

Now you need to connect the two containers to the network, by adding the `--network` option:

```
docker container run --name mysql-container --rm --network if_wordpress -e
MYSQL_ROOT_PASSWORD=wordpress -d mysql
af38acac52301a7c9689d708e6c3255704cdffb1972bcc245d67b02840983a50

docker container run --name wordpress-container --rm --network if_wordpress -e
WORDPRESS_DB_HOST=mysql-container -e WORDPRESS_DB_PASSWORD=wordpress -p 8080:80 -d
wordpress
fd4fd096c064094d7758cefce41d0f1124e78b86623160466973007cf0af8556
```

Notice the `WORDPRESS_DB_HOST` env variable. When you make a container join a network, it automatically gets the container name as DNS name as well, making it super easy to make containers discover each other.

You have now deployed both containers into the network. Take a deeper look into the container network by issuing: `docker network inspect if_wordpress`.

```
docker network inspect if_wordpress
[
    {
        "Name": "if_wordpress",
        "Id": "04e073137ff8c71b9a040ba452c12517ebe5a520960a78bccb7b242b723b5a21",
        "Created": "2017-11-28T17:20:37.83042658+01:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "Containers": {
            "af38acac52301a7c9689d708e6c3255704cdffb1972bcc245d67b02840983a50": {
                "Name": "mysql-container",
                "EndpointID":
"96b4befec46c788d1722d61664e68bfcbd29b5d484f1f004349163249d28a03d",
                "MacAddress": "02:42:ac:12:00:02",
                "IPv4Address": "172.18.0.2/16",
                "IPv6Address": ""
            },
            "fb4dad5cd82b5b40ee4f7f5f0249ff4b7b4116654bab760719261574b2478b52": {
                "Name": "wordpress-container",
                "EndpointID":
"2389930f52893e03a15fdc28ce59316619cb061e716309aa11a2716ef09cde17",
                "MacAddress": "02:42:ac:12:00:03",
                "IPv4Address": "172.18.0.3/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]
```

As, we have linked both the container now wordpress container can be accessed from browser using the address http://localhost:8080 and setup of wordpress can be done easily. MySQL is not accessible from the outside so security is much better than before.

Cleanup

Close both of the containers down by issuing the following command:

```
docker container stop wordpress-container mysql-container
```