Using Docker compose

If you have started working with Docker and are building container images for your application services, you most likely have noticed that after a while you may end up writing long `docker container run` commands. These commands, while very intuitive, can become cumbersome to write, especially if you are developing a multi-container applications and spinning up containers quickly.

Docker Compose is a "*tool for defining and running your multi-container Docker applications*".

Your applications can be defined in a YAML file where all the options that you used in `docker container run` are defined.

Compose also allows you to manage your application as a single entity rather than dealing with individual containers.

This file defines all of the containers and settings you need to launch your set of clusters. The properties map onto how you use the docker run commands, however, are now stored in source control and shared along with your code.

Terminology

- `docker-compose.yml` The YAML file where all your configuration of your docker containers go.
- `docker-compose` The cli tool that enables you to define and run multi-container applications with Docker
  - `run` :
  - `up` : creates and starts the services stated in the compose file
  - `down` : stops and removes containers, networks, images, and volumes
  - `restart` :
  - `logs` : streams the acummulated logs from all the containers in the compose file
  - `ps` : same as pure `docker container` variant; shows you all containers that are currently running.
  - `rm` : removes all the containers from the given compose file.
  - `start` : starts the services
  - `stop` : stops the services

The docker cli is used when managing individual containers on a docker engine. It is the client command line to access the docker daemon api.

The docker-compose cli together with the yaml files can be used to manage a multi-container application.

Compose-erizing your wordpress

So we want to take advantage of docker-compose to run our wordpress site.

In order to to this we need to:

1. Transform our setup into a docker-compose.yaml file
2. Invoke docker-compose and watch the magic happen!

Head over to this labs folder:

```
cd labs/multi-container
```

Open the file `docker.compose.yaml` with a text editor:

```
nano docker-compose.yaml
```

You should see something like this:

```
version: '3.1'

services:

#  wordpress_container:

  mysql_container:
    image: mysql
    ports:
      - 3306:3306
    environment:
      MYSQL_ROOT_PASSWORD: wordpress
```

This is the template we are building our compose file upon so let's drill this one down:

- `version` indicate what version of the compose syntax we are using
- `services` is the section where we put our containers
    - `wordpress_container` is the section where we define our wordpress container
    - `mysql_container` is the ditto of MySQL.

> For more information on docker-compose yaml files, head over to the documentation.

The `services` part is equivalent to our `docker container run` command. Likewise there is a `network` and `volumes` section for those as well corresponding to `docker network create` and `docker volume create`.

Let's look the mysql_container part together, making you able to create the other container yourself. Look at the original command we made to spin up the container:

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

The command gives out following information: a `name`, a `port` mapping, an `environment` variable and the `image` we want to run.

Now look at the docker-compose example again:

- `mysql_container` defines the name of the container
- `image:wordpress` describes what image the container spins up from.
- `ports` defines a list of port mappings from host to container
- `environment` describes the `-e` variable made before in a yaml list

Try to spin up the container in detached mode:

```
docker-compose up -d
Creating network "multicontainer_default" with the default driver
Creating multicontainer_mysql_container_1 ...
Creating multicontainer_mysql_container_1 ... done
```

Looking at the output you can see that it made a `docker network` named `multicontainer_default` as well as the MySQL container named `multicontainer_mysql_container_1`.

Issue a `docker container ls` as well as `docker network ls` to see that both the container and network are listed.

To shut down the container and network, issue a `docker-compose down`

> **note**: The command docker-compose down removes the containers and default network.

Creating the wordpress container

You now have all the pieces of information to make the Wordpress container. We copied the run command from before if you cant remember it by hart.

`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`

You must

- un-comment the `wordpress_container` part of the services section
- map the pieces of information from the docker container run command to the yaml format.
- remove MySQL port mapping to close that from outside reach.

When you made that, run `docker-compose up -d` and access your wordpress site from http://IP:8080

Hint: If you are stuck, look at the file docker-compose_final.yaml in the same folder.