Docker on Development 2

1) Create a Dockerfile

You will start by creating a Flask application inside a container. Flask is a framework to web development in python. (Have a look : *http://flask.pocoo.org/*)
As an example we are going to use the simple Hello World application defined by the following Python script :

```
#!/usr/bin/env python
from flask import Flask
app = Flask(__name_)

@app.route('/hi')
def hello_world():
        return 'Hello World!'
if __name__ == '__main__':
        app.run(host='0.0.0.0', port=5000)
```

To get this application running inside a Docker container, you need to write a Dockerfile that installs the pre-requisites needed to run the application and exposes the port 5000 that the application runs. You also need to move the application (Hello World) inside the container filesystem.

2)Run the application

To run the application, we use the -d option of docker run which daemonizes the container, and we also use the -P option of docker run to let Docker choose a port on the Docker host that will be mapped to the exposed port specified in the Dockerfile (e.g 5000).

```
$ docker run -d -P flask
5ac72ed12a72f0e2bec0001b3e78f11660905d20f40e670d42aee292263cb890

$ docker ps

CONTAINER ID     IMAGE          COMMAND             CREATED       STATUS
5ac72ed12a72     flask:latest   "python /tmp/hello.p 4 days ago    Up 2 seconds
```

We see that the container returned, it has been daemonized and we are not logged into an interactive shell. The PORTS shows a mapping between port 5000 of the container and port 49153 of the Docker host. A simple curl to http://localhost:49153/hi will return Hello World , or you can also open your browser to the same url.
Since our Dockerfile specified a command to run via CMD, we do not need to specify a command after the name of the image to use. However since we used CMD and not ENTRYPOINT, we could override it and start the container in interactive mode, to explore it

```
$ docker run -t -i -P flask /bin/bash
# ls -l /tmp
total 4
-rw-r--r-- 1 root root 194 Fev 27 10:41 hello.py
```

#
You have written a Dockerfile and build an image for a useful container. Now you want to share this image with everyone.

3) Share this image on the Docker hub.

Linking containers

We can choose which port exposes a container to others (with -p option). You won't always want to expose your services to the host machine or the outside world, but you will want to connect containers to one another. This technique shows how you can achieve this by using Docker's link flag, ensuring outsiders can't access your internal services.

Problem :

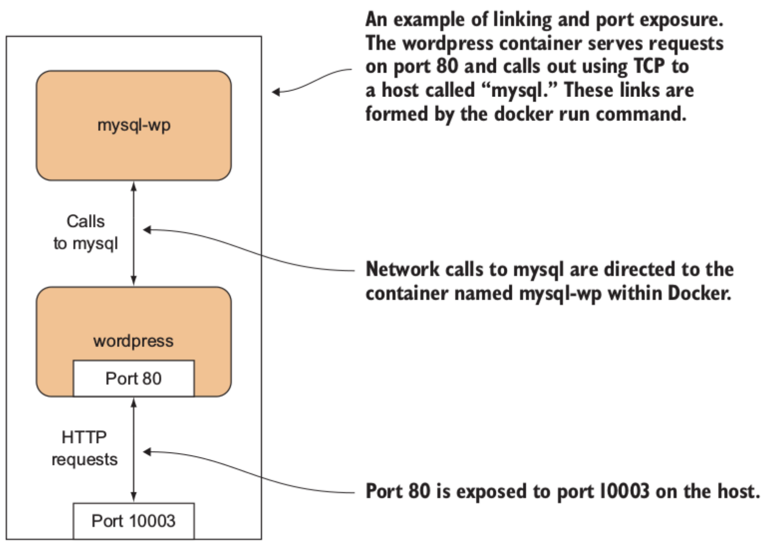
You want to allow communication between containers for internal purposes.

Solution :

Use Docker's linking functionality to allow the containers to communicate with each other.

Discussion :

In order to set up a WordPress/database container, we're going to separate the mysql database tier from the wordpress container, and link these to each other without port configuration. Figure gives an overview of the final state.


mysql-wp
Calls to mysql
wordpress
Port 80
HTTP requests
Port 10003
An example of linking and port exposure. The wordpress container serves requests on port 80 and calls out using TCP to a host called "mysql." These links are formed by the docker run command.
Network calls to mysql are directed to the container named mysql-wp within Docker.
Port 80 is exposed to port 10003 on the host.

WHY IS THIS USEFUL? Why bother with linking if you can already expose ports to the host and use that? Linking allows you to encapsulate and define the relationships between containers without exposing services to the host's network (and potentially, to the outside world). You might want to do this for security reasons, for example.

Run your containers like so, in the following order, pausing for about a minute between the first and second commands:

```
1)$ docker run --name wp-mysql -e MYSQL_ROOT_PASSWORD=yoursecretpassword -d mysql
2)$ docker run --name wordpress --link wp-mysql:mysql -p 10003:80 -d wordpress
```

First you give the mysql container the name wp-mysql so you can refer to it later. You also must supply an environment variable so the mysql container can initialize the database (-e MYSQL_ROOT_PASSWORD=yoursecretpassword). You run both containers as daemons (-d) and use the Docker Hub reference for the official mysql image.

In the second command 2) you give the wordpress image the name wordpress , in case you want to refer to it later. You also link the wp-mysql container to the wordpress container (--link wp-mysql:mysql). References to a mysql server within the wordpress container will be sent to the container named wp-mysql . You also use a local port mapping (-p 10003:80) and add the Docker Hub reference for the official wordpress image (wordpress). Be aware that links won't wait for services in linked containers to start; hence the instruction to pause between commands. A more precise way of doing this is to look for mysqid: ready for connections in the output of docker logs wp-mysql before running the wordpress container.

If you now navigate to http://localhost:10003, you'll see the introductory wordpress screen and you can set up your wordpress instance.

The meat of this example is the --link flag in the second command. This flag sets up the container's host file so that the wordpress container can refer to a mysql server, and this will be routed to whatever container has the name "wp-mysql." This has the significant benefit that different mysql containers can be swapped in without requiring any change at all to the wordpress container, making configuration management of these different services much easier.

In order for containers to be linked in this way, their ports must be specified as exposed when building the images. This is achieved using the EXPOSE command within the image build's Dockerfile. You have now seen a simple example of Docker orchestration, and you've taken a step toward a microservices architecture. This fine-grained control over running services is one of the key operational benefits of a microservices architecture.