Docker Networking
IN720 Virtualisation

Introduction

So far we have mainly worked with just one container at a time, and we generally stick to the guideline that a container runs just one main process. Most real services are provided by multiple processes working in concert, which means that we expect to deploy more than one container to provide the service. In order to get those containers to work together, we need to use *Docker networking*.

In this lab we will create a simple service using a Python/Flask application in one container and an nginx reverse proxy server in a second container. We will create a Docker network to connect the two containers.

1 Create a Flask application container

Make a directory `flaskapp` to serve as a build context for the Python/Flask application image. In that directory, create a dockerfile with the following contents:

```
FROM ubuntu:16.04
MAINTAINER Your name <email@somedomain>
ENV updated_on "2018-08-14 900"

RUN apt-get update
RUN apt-get -y upgrade
RUN apt-get -y install python3 python3-setuptools python3-pip gunicorn3
RUN  update-alternatives --install /usr/bin/python python /usr/bin/python3 10

COPY lab-4-app /flaskapp
WORKDIR /flaskapp
RUN pip3 install -r requirements.txt

EXPOSE 5000
ENTRYPOINT "./startup.sh"
```

This dockerfile creates an image with some dependencies needed to run our app, copies our Flask application code into it, and then runs the app using a gunicorn wsgi server on port 5000. Now we just need to add the application code to our context. Clone the GitHub repository tclark/lab-4-app into your context.

Build your container image with the tag `user/flaskapp`. Test it by running a container with the following command

```
docker run -d -rm --name flaskapp -p 5000:5000 user/flaskapp
```

and verify that it works by checking it with a browser. Our application works, but for a production deployment we need to place it behind a reverse proxy server. For this, we will need a second container. Shut down your `flaskapp` container.

2 Create an nginx Container

Set up a second build context, named `nginx`. Place a dockerfile in it with the following:

```
FROM nginx:1.13
MAINTAINER Tom Clark <tclark@op.ac.nz>
COPY flaskapp.conf /etc/nginx/conf.d/default.conf
EXPOSE 80
```

This will produce a simple nginx container to serve as a front end for our service. We need to add the `flaskapp.conf` file to our context with the following:

```
resolver 127.0.0.11 valid=1s;

server {
  set $alias "flaskapp";
  location / {
    proxy_set_header Host $host;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_pass http://$alias:5000;
  }
  listen 80;
}
```

This will cause nginx to send proxy requests to a `flaskapp` host, or in our case, container. Build this image with the tag `user/nginx`.

3 Test our service

Now, run two containers based on our images with the following commands.

```
docker run -d --rm --name flaskapp user/flaskapp
docker run -d --rm --name -p 8080:80 nginx user/nginx
```

and see if you can see your application output.

Check your containers' IP addresses with the commands

```
docker inspect -f {{.NetworkSettings.Networks.bridge.IPAddress}} flaskapp
docker inspect -f {{.NetworkSettings.Networks.bridge.IPAddress}} nginx
```

Stop your containers.

4 Create a Docker Network

We want to place our containers on their own isolated network. The Flask application container does not need to be reachable by anything but our nginx container. Newer version of Docker make this very easy. First, create our "network" with the command

```
docker network create app
```

And now that's done. You can verify that the network was created with the command

```
docker network ls
```

Now we just need to start our containers again, this time directing them to use this new network.

```
docker run -d --rm --name flaskapp --net=app user/flaskapp
docker run -d --rm --name nginx --net=app -p 8080:8- user/nginx
```

Note that it doesn't matter in which order you start the containers. Now verify one last time that your application works by checking it with a browser.