Simple Docker Compose

Docker Build example

Docker Compose has many different keys available, one of the most useful in a development environment is the `build` key which allows building a Docker image from source files.

In this lab you will build a simple `Python` web server from source files then create a `Dockerfile` and `docker-compose.yml` file to automate building and running of the application.

Manually

To show the value of using Docker Compose we are going to do things manually at first and then use Compose to automate the building of this simple application.

First let's start by reviewing the Python web server application code.
`helloworld/app.py`

```
from flask import Flask
import sys
import optparse
import time

app = Flask(__name__)

start = int(round(time.time()))

@app.route("/")
def hello_world():

    return "Hello again!"

if __name__ == '__main__':
    parser = optparse.OptionParser(usage="python simpleapp.py -p ")
    parser.add_option('-p', '--port', action='store', dest='port', help='The
port to listen on.')
    (args, _) = parser.parse_args()
    if args.port == None:
        print "Missing required argument: -p/--port"
        sys.exit(1)
    app.run(host='0.0.0.0', port=int(args.port), debug=False)
```

As we can see this application requires `Python`, `PIP`, and `Flask`

Look at the following `compose/helloworld/Dockerfile`

```
FROM alpine:3.1

# Update
RUN apk add --update python py-pip

# Install app dependencies
RUN pip install Flask

EXPOSE  8000
WORKDIR /code
CMD ["python", "app.py", "-p 5000"]
```

This `Dockerfile` starts with the `alpine:3.1` image and installs the required packages

- Python
- PIP
- Flask

It then sets the `WORKDIR` to `/code` and then runs `app.py` defining the port to listen on.

Ok, now that you've reviewed the files used to build this image let's run the build commands.

Using `docker build` create an image named `helloworld` with a tag of `1.0`

Then use `docker run` to create a container using the image `helloworld:1.0` that maps port `5000` from container to host, and creates a volume mounting the application code in `helloworld` directory to `/code` in the container.

After the container is started confirm it loads successfully in a browser by visiting `http://localhost:5000` or by using curl

```
curl http://localhost:5000
```

Docker Compose

Ok, the above steps were cumbersome and require us to rebuild our image manually any time we want to make a change to the Python web server code. Now let's automate this process using `docker-compose`

We will be using the same `Dockerfile` and `app.py` from above but now we're going to create a compose file that builds the image from source files if it's not already built and runs it.

Review `docker-compose.yml` in `compose/docker-compose.yml`

```
version: '2'
services:
  helloworld:
    build: ./helloworld
    image: helloworld:1.0
    ports:
    - "5000:5000"
    volumes:
    - ../helloworld:/code
```

We can see that it is looking in the `helloworld` directory, building the image from the `Dockerfile` and tagging it `helloworld:1.0`. Port `5000` is also mapped to the host and the application code is mapped into the container at `/code`

Run Docker Compose and confirm application started successfully.

```
docker-compose up
```

You should see a bunch of output while it builds the image and then when it starts the application something like the following.

```
Attaching to compose_helloworld_1
helloworld_1  |  * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
helloworld_1  | 172.18.0.1 - - [09/Apr/2018 03:12:41] "GET / HTTP/1.1" 200 -
```

Let's confirm the web server is running by visiting `http://localhost:5000` in a browser, or you can curl it.

```
curl http://localhost:5000
```

Lab Complete