Docker Compose with Flask Apps

Docker Compose simplifies multi-container Docker environments on a single host. Let's put together a basic web app with Docker Compose and Flask using Redis (we end up with a Python/Flask container and a Redis container all on one host). We do not need custom networking or to design our own networks to run on; Compose does everything for us!

Set Up Your Host

Install Docker Engine and Docker Compose (unless you have them already); review Introduction to Docker Compose if you need help.

1. You'll need a directory for your project on your host machine:

```
$ mkdir compose_flask
$ cd compose_flask
```

2. Add the following to `requirements.txt` in your directory:

```
flask
redis
```

3. Copy and paste the following code into a new file called `app.py` in your project directory:

```
# compose_flask/app.py
from flask import Flask
from redis import Redis

app = Flask(__name__)
redis = Redis(host='redis', port=6379)

@app.route('/')
def hello():
    redis.incr('hits')
    return 'This Compose/Flask demo has been viewed %s time(s).' % redis.get('hit


if __name__ == "__main__":
    app.run(host="0.0.0.0", debug=True)
```

Your Docker Image

1. Create your `Dockerfile` in the `compose_flask` directory and add the following:

```
FROM python:2.7
ADD . /code
WORKDIR /code
RUN pip install -r requirements.txt
CMD python app.py
```

2. Build it: `docker build -t compose-flask .`

Define Your Services

Add the following code to a new file, `docker-compose.yml`, in your project directory:

```
version: '2'
services:
    web:
        build: .
        ports:
            - "5000:5000"
        volumes:
            - .:/code
        depends_on:
            - redis
    redis:
        image: redis
```

How to Read the Docker Compose File

- We define two services, `web` and `redis`.
- The `web` service builds from the `Dockerfile` in the current directory...
- Forwards the container's exposed port (5000) to port 5000 on the host...
- Mounts the project directory on the host to `/code` inside the container (allowing you to modify the code without having to rebuild the image)...
- And links the web service to the Redis service.
- The `redis` service uses the latest Redis image from Docker Hub.

Note this composition documents your application's requirements!

Build and Run with Docker Compose

Start the application from your directory:

```
$ docker-compose up
```

If you have localhost access to your host (i.e., you do not use a remote solution to deploy Docker), point your browser to `http://0.0.0.0:5000`, `http://127.0.0.1:5000`, or `http://localhost:5000`. On a Mac, you need to use `docker-machine ip MACHINE_VM` to get your Docker host's IP address (then use that address like `http://MACHINE_IP:5000` to access your web page). If you *do* use a remote host, simply use that IP address and append `:5000` to the end.

You should see:

```
This Compose/Flask demo has been viewed 1 time(s).
```

When you refresh, you should see:

```
This Compose/Flask demo has been viewed 2 time(s).
```

Each time you refresh, the number should increment.

Stop the application with `CTRL+C` (read below under “Common Issues” for more information) and refresh the page. You should receive something to the effect of `This site can't be reached`.

Restart the application with `docker-compose up -d`. Redis resets the count and you should see:

```
This Compose/Flask demo has been viewed 1 time(s).
```

Use `docker-compose ps` and you should get similar results:

```
        Name                      Command               State           Ports
----------------------------------------------------------------------------------
composeflask_redis_1   docker-entrypoint.sh redis ...   Up      6379/tcp
composeflask_web_1     /bin/sh -c python app.py         Up      0.0.0.0:5000->5000/tc
```

Run `docker-compose stop` to stop the containers:

```
$ docker-compose stop
Stopping composeflask_web_1 ... done
Stopping composeflask_redis_1 ... done
```

Common Issues

Starting and Stopping

If you run `docker-compose up -d`, you need to run `docker-compose stop` to stop the services when you finish. If you *did not*, you can stop the service with `CTRL+C` (hit once to gracefully stop and twice to force kill the containers). If you do not run in the background, you can view the calls made to your container.