Dockerize your Flask Application

In this article, we'll take a look at how to dockerize a Flask application. Flask is a microframework for Python, with a basis in Werkzeug and Jinja 2.

The Docker Hub image

Since Docker Hub doesn't have an official Flask repository (at the time of this writing), we'll explain how to build our own. While you can always use a non-official image, it's generally recommended to make your own Dockerfile to ensure you know what is in the image. We'll start off with the base of Ubuntu, but you can use any available distro you prefer.

Setting Up

We need to set up a basic app and Dockerfile.

Our basic app

Start with creating a new directory; let's call it `flask_web`:

```
mkdir flask_web
```

You can use this basic `app.py` file for your basic application:

```
# flask_web/app.py

From flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world:
    return 'Hey, we have Flask in a Docker container!'


if __name == '__main__':
    app.run(debug=True, host='0.0.0.0')
```

Now, we need to include Flask in our `requirements.txt` file:

```
Flask==0.10.1
```

Flask Dockerfile

We're starting from Linux instead of using the Python repository as our base, as its more clear in which Python version is being installed (what `apt` installs on Ubuntu or Debian, or `yum` installs on Red Hat and CentOS). You always have the option to build off the `python` image instead.

```
FROM ubuntu:16.04

MAINTANER Your Name "youremail@domain.tld"

RUN apt-get update -y && \
    apt-get install -y python-pip python-dev

# We copy just the requirements.txt first to leverage Docker cache
COPY ./requirements.txt /app/requirements.txt

WORKDIR /app

RUN pip install -r requirements.txt
```

```
COPY . /app

ENTRYPOINT [ "python" ]

CMD [ "app.py" ]
```

Let's go over some of these Docker instructions:

1. `MAINTAINER` sets the *Author* field of the image (useful when pushing to Docker Hub)
2. `&& \` isn't Docker specific, but tells Linux to run the next command as part of the existing line (instead of using multiple `RUN` directives, you can use just one)
3. `COPY` copies files from the first parameter (the source `.`) to the destination parameter (in this case, `/app`)
4. `WORKDIR` sets the working directory (all following instructions operate within this directory); you may use `WORKDIR` as often as you like
5. `ENTRYPOINT` configures the container to run as an executable; only the last `ENTRYPOINT` instruction executes

`pip` installs from `requirements.txt` as normal. Since `requirements.txt` only references `Flask 0.1.0`, `pip` only installs `Flask 0.1.0`. If you are using Flask for your app, then you're likely to have more modules specificed for installation.

Build the image

Now that we have a Dockerfile, let's verify it builds correctly:

```
docker build -t flask-tutorial:latest .
```

After the build completes, we can run the container:

```
docker run -d -p 5000:5000 flask-tutorial
```

Further information

Ensure you are using the right ports. Flask by default runs on port 5000 (not 8000 like Django or 80 like Apache). Check out Binding Docker Ports for more information.

Next: Dockerize your Pyramid Application
Get started with dockerizing your Pyramid application.