Moving Octopress to Docker

Peter Humburg

Published at *Genomic Campfire* on Sep 09 2016

Contents

This blog has been quiet for a while. Okay, for rather a long time. It is time to get it going again. This time with the help of Docker. The aim is to make the local environment I use to write and maintain the blog more portable. The interruption in activity around here was in part because I've simply been busy with other things but there was another problem. I got a new computer. Although generally a good thing it meant that I lost my Octopress setup. Sure, it is all hosted on GitHub and no content was lost but since the blog needs to be compiled on my machine before it is published there is a fair amount of software that needs to be set-up. This shouldn't be too bad, I've done that before after all. But things were complicated by the fact that the new laptop runs Windows[1] [2]. So long story short, there were simply too many obstacles to writing. I've now decided that this blog has been languishing for long enough and that I will resurrect it with the help of Docker.

The Docker image

I started by looking around for an existing Docker image I could use. There certainly is no shortage of Octopress images. Unfortunately they were all either not quite providing what I needed or I couldn't quite figure out how I was supposed to use them. So I rolled my own. This is loosely based on the image by Traun Leyden. The final product is available on GitHub.

Let's take a look at what is inside.

```
FROM ubuntu:16.04
MAINTAINER Peter Humburg <peter.humburg@gmail.com>
ENV LC_ALL C.UTF-8
```

The only thing to note here is that UTF-8 support needs to be enabled explicitly to stop Ruby from complaining about non-ASCII characters. With that out of the way we are ready to install the dependencies, using `apt-get` as much as possible. That is where things start to get a bit more interesting. Since I use *pandoc* for my Markdown conversions that needs to be installed in addition to other Octopress dependencies. The *pandoc* version available in the official Ubuntu repositories tends to be woefully out of date so I'll compile it myself. That in turn will require *stack*. I also use XeLaTeX to generate PDF versions of my blog posts. So we'll need that too.

```
RUN apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys 575159689BEFB442 && \
  echo 'deb http://download.fpcomplete.com/ubuntu xenial main'| tee /etc/apt/sources.list.d/fpco.list &
  apt-get update && apt-get install -y \
  texlive-xetex \
  bundler \
  curl \
  git \
  ruby \
  stack \
  wget && \
  apt-get clean
```

That is all the package installation wrapped up in a single *RUN* command to keep the number of layers in check. Now we just need to install *pandoc*. I opted to install this directly from GitHub (using version 1.17.2).

```
RUN git clone https://github.com/jgm/pandoc.git /source/pandoc && \
    cd /source/pandoc && \
    git checkout tags/1.17.2 -b v1.17.2 && \
    git submodule update --init && \
    stack setup && \
    stack --local-bin-path /usr/local/bin install && \
    cd / && rm -r /source/pandoc
```

The only thing to note here is that *stack* wants to install everything locally for the current user, avoiding conflicts with a system wide installation. Usually that is convenient but not at all what I want for this Docker

[1] We get along very well, thank you.
[2] for the most part anyway

image. Hence I use the `--local-bin-path` option to ensure *pandoc* is available globally. That really only leaves the installation of *Octopress*. As usual this is simply cloned from GitHub.

```
RUN git clone git://github.com/imathis/octopress.git /octopress && \
    cd /octopress && \
    gem install bundler && \
    bundle install
```

Note that I'm installing into `/octopress`. This could then serve as the starting point for a new blog but I'm mostly interested in getting my existing blog to work with this again. More on that below.

Finally, a few finishing touches. I'm exposing the default port used by Octopress to serve previews of the blog and change the working directory to the directory that contains all the blog contents.

```
EXPOSE 4000
WORKDIR /octopress
```

One other thing that might be useful here is the addition of an entry point to run `rake`. Currently I'm running rake tasks from a bash shell in the container but it may be more convenient to simply execute rake tasks from the terminal of the host system.

Customization

Now that a basic Octopress installation is in place it is time to get the existing blog connected to it. After cloning the blog's repo from GitHub I simply mounted it on top of the default repo contained in the Docker image.

```
docker run -ti --name blog -v $(pwd):/octopress humburg/octopress bash
```

Then inside the container I ran `bundle install` again to ensure that any additional dependencies I had added would be installed as well. To make sure I didn't have to do this again any time soon I committed the resulting changes to a new image and tagged it with the blog's name.

```
docker commit -a "Peter Humburg" -m "Additional dependencies for Genomic Campfire blog" blog genomic_ca
```

And that should be it. Or so I thought. I ran the newly created image:

```
docker run -ti -p 4000:4000 -v $(pwd):/octopress genomic_campfire bash
```

Followed by `rake preview` inside the container. That seemed to run fine but when I pointed my browser at `localhost:4000` I only got an empty response error. Eventually I figured out that (under Windows at least) it is necessary to provide an IP address to the web server that is started by the `preview` task. To this end replace this line in the rakefile

```
rackupPid = Process.spawn("rackup --port #{server_port}")
```

with this one

```
rackupPid = Process.spawn("rackup --host 0.0.0.0 --port #{server_port}")
```

That fixed that issue and I was pleased to see that I had my blog back.

Deployment

Everything looked good so I went ahead, wrote this blog post and ran `rake gen_deploy`. And it turns out there is one more piece of the puzzle that I had completely forgotten about. You see, Octopress makes somewhat unconventional use of branches in the git repository when using it in conjunction with GitHub. There is the *source* branch that holds the working copy of the blog. That is where you write new posts, tweak the appearance of the blog and so on. Then there is another branch (either *master* or *gh-pages* depending on whether this is a user or project page). This is where Octopress pushes updated versions of the blog to publish them. All of this happens in the _deploy directory, which of course isn't itself tracked by git and therefore was missing from my repository. Which means that for deployment to work this needs to be set-up first.

There is a rake task that helps with this when setting up a new Octopress blog but since my one was mostly set up already I didn't dare to use that. Instead I just cloned the git repository into _deploy.

```
git clone git@github.com:humburg/humburg.github.io.git _deploy
```

If you are using Octopress with a project repo you'll also need

```
git checkout gh-pages
```

And that's it. For real this time.