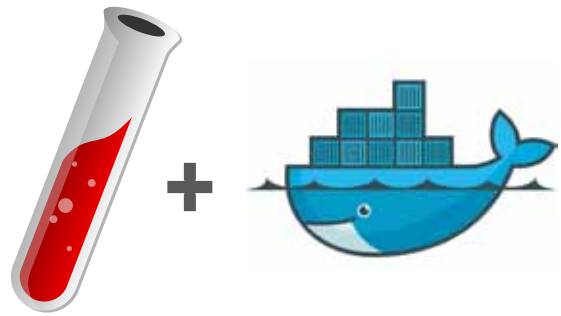
Deploying a Jekyll Blog in Docker

Jekyll is a simple, blog-aware, static site generator. It takes a template directory containing raw text files in various formats, runs it through a converter and spits out a complete, ready-to-publish static website. This article integrates Jekyll with Docker.



Jekyll is a blog alternative that generates static HTML from templates that you create. It is a simple, blog-aware, static site generator written on Ruby by Tom Preston Werner, GitHub's co-founder. It takes a template directory (representing the raw form of a website), runs it through Markdown and Liquid converters, and spits out a complete static website.

Why does one need to use Jekyll?

No server-side language or database: This is only good old HTML/CSS/JS. Frankly, I don't want to have anything to do with a database unless I absolutely need to. This also means it's worry-free.

Simpler workflow: One only needs a text editor and Git to update the site or release a blog post. There's no need for a local PHP server or anything. Plus, synchronising the local environment with the one in production takes no more than a single command.

Fewer dependencies: No more jQuery.paginate for pagination; Jekyll has a built-in plugin to do it. No more *Prism.js* for syntax highlighting; Jekyll comes with Pygments, a Python based syntax highlighter. Less JS (and especially no more jQuery) means a faster site.

Enough of asking why; let's jump to how to go about things!

Creating the directory structure

The final structure will look something like what's shown below:

```
├── app
│   ├── _drafts
│   ├── _includes
│   ├── _layouts
│   │   ├── default.html
│   │   └── post.html
│   ├── _posts
│   ├── css
│   ├── images
│   ├── index.html
│   ├── js
│   └── robots.txt
├── _config.yml
├── Dockerfile
├── Gemfile
└── web
```

Anything that you put into the app/directory will get copied to the generated site too, so put your css, images, js files here.

Let's have a look at the Jekyll config file *_config.yml:*

```
source: app
destination: web

url: http://blog.your-domain.com
permalink: pretty

encoding: utf-8
```

Fill in your domain in the 'url' parameter.

Next, have a look at the views; first up is *app/_layouts/default.html:*

```
<!DOCTYPE html>
<html lang="en" prefix="og: http://ogp.me/ns#">
<head>
    <meta charset="utf-8">
    <link rel="stylesheet" type="text/css" href="/css/style.
css">

    <meta name="viewport" content="width=device-width,
initial-scale=1">

    <title>{% if page.title %}{{ page.title }} &mdash; {%
endif %}Your Blog</title>
```

```
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <meta property="og:url" content="{{ site.url }}{{ page.
url | remove_first:'index.html' }}">
    <meta property="og:site_name" content="blog.your-domain.
com">

    {% if page.title %}
    <meta property="og:title" content="{{ page.title }}">
    {% endif %}

    {% if page.description %}
    <meta name="description" content="{{ page.description
}}">
    <meta name="og:description" content="{{ page.description
}}">
    {% else if page.excerpt %}
    <meta name="description" content="{{ page.excerpt |
strip_html | truncatewords: 25 }}">
    <meta name="og:description" content="{{ page.excerpt |
strip_html | truncatewords: 25 }}">
    {% endif %}

    {% if page.og_image_url %}
    <meta property="og:image" content="{{ page.og_image_url
}}">
    {% else if page.photo_url %}
    <meta property="og:image" content="{{ page.photo_url }}">
    {% endif %}

    {% if page.keywords %}
    <meta name="keywords" content="{{ page.keywords }}" />
    {% endif %}

    {% if page.date %}
    <meta property="og:type" content="article">
    <meta property="article:published_time" content="{{ page.
date | date: "%Y-%m-%d" }}">
    {% endif %}

    <script src="//code.jquery.com/jquery-2.1.1.min.js"></
script>

</head>
<body>

<nav class="navbar navbar-default navbar-static-top">
    <div class="container">
        <!-- Brand and toggle get grouped for better mobile
display-->
    <div class="navbar-header">
            <button type="button" class="navbar-toggle
collapsed" data-toggle="collapse" data-target="#bs-example-
navbar-collapse-1" aria-expanded="false">
                <span class="sr-only">Toggle navigation</
span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
            </button>
            <a class="navbar-brand" href="/">Your Blog</a>
        </div>
    </div> <!-- /.container -->
</nav>

<div class="container">

    {{ content }}

</div>

</body>
</html>
```

Most of the other views will be generated from this, which is not at all complicated. You can change it according to your need.

Next up is *app/_layouts/post.html:*

```
---
layout: default
---

<article>
    <header>
        <h1>{{ page.title }}</h1>
        <div class="post-info">
            <div class="author-published">
                by {{ page.author }}
            </div>
            <div class="date-published">
                <time datetime="{{ page.date }}">{{ page.date
| date: '%B %d, %Y' }} </time>
            </div>
        </div>
    </header>

    <div>{{ content }} </div>

</article>
```

Simple stuff, isn't it? Please notice, we are using Jekyll front matter to add some variables that apply only to this template. The front matter is where Jekyll starts to get really cool. Any file that contains a YAML front matter block will be processed by Jekyll as a special file. The front matter must be the first thing in the file, and must take the form of valid YAML set between triple-dashed lines.

Finally, the last template in our simple blog will be the home page. It's in a different location because it's the home page and so it should be at *app/index.html*:

```
---
layout: default
description: Your Beautiful Blog
---

<h1> Recent Posts </h1>
{% for post in site.posts limit:50 %}
<h2> <a href="{{ post.url }}">{{ post.title }}</a> </h2>
<div class="preview">
    {% if post.description %}
    <span class="body">{{ post.description | strip_html |
truncatewords: 300 }}</span>
    {% else %}
    <span class="body">{{ post.excerpt | strip_html }}</span>
    {% endif %}
</div>
{% endfor %}
```

On the home page, we'll just list the 50 most recent articles with an excerpt from each one. You may have noticed that we are again extending from the default layout.

We're nearly ready to generate the site but need one more thing...

Using Gems plugins

Remember we mentioned plugins? We'll configure them now. Create a Gemfile and put this in it, as shown below:

```
source 'https://rubygems.org'

group :jekyll_plugins do
    gem 'jekyll', '~>3.0'
    gem 'kramdown'
    gem 'rdiscount'
    gem 'jekyll-sitemap'
    gem 'jekyll-redirect-from'
end
```

We'll install these Gems later when we generate the site. At this point, you might want to write a few words of your first blog post and put it into *app/_posts/*. Let's generate our site, now!

We're going to use a Docker container with Jekyll already installed on it. You can of course install Jekyll locally, but using a Docker container makes it more portable; for example, you might decide to build your blog continuously later on, by containing your tools inside Docker containers. You don't have to install them on every server you want to build the blog on.

Run this to install the Gems and build your shiny new blog, as follows:

```
$ docker run --rm \
        -v "$(pwd):/src" \
        -w /src \
        ruby:2.3 \
        sh -c 'bundle install \
            --path vendor/bundle \
            && exec jekyll build --watch'
```

We've added the *–watch* flag at the end, which is handy when we're making changes and we want to see them reflected immediately on the blog.

Voila! Have a look in the Web/folder—you should see lots of HTML files, which are what Jekyll generated. If you were to FTP that entire Web/folder to a server, you would have a working blog, but we're going to put it into a Docker container.

The Dockerfile

Our container will be super simple. We just need to serve the Web/folder that we just generated. Here's our Dockerfile:

```
FROM nginx
EXPOSE 80
COPY web/ /usr/share/nginx/html

That's it! Now, let's build and run the image:
$ docker build -t my-shiny-blog .
$ docker run -d \
        -p 80:80 \
        -v "$(pwd)/web:/usr/share/nginx/html" \
        my-shiny-blog
```

Now hit 127.0.0.1:8080 in your browser to see your blog in action!

Notice that we've mounted the source into the container as a volume, which will allow us to see updates in real-time when Jekyll regenerates the site (since Jekyll is still running with '–watch'), without having to rebuild the image again. Before pushing the image to your registry, just remember to rebuild!

That's it! We can of course get fancy and minimise Javascript or compile less to css using Gulp, but we'll leave that as an exercise for the reader. The trick is to put less source code outside of the app/directory and have Gulp place the final versions in the app/directory. That way, Jekyll will copy them over when the site is generated. END

References

[1] *https://jekyllrb.com/*
[2] *https://github.com/jekyll/docker*

By: Srijan Agarwal

The author is a developer at *WikiToLearn,* an open source enthusiast and works with JS and PHP mostly. You can contact him at *www.srijanagarwal.me*.