Manually Deploying a Jekyll Website to Amazon S3

Using Jekyll as web site generator is a very powerful way to make use of some interesting ways to create and deliver content. It’s much simpler than having to create and edit the entire site in raw HTML and CSS yourself. There are lots of templates available online to deploy on a Jekyll framework.

What is Jekyll?

Jekyll is a simple ruby-based framework to run slick web sites which are described as “simple, blog-aware static sites” that can range from a single-page application to larger blogs which allow you to write your content in markdown. This is also the framework that is supported by GitHub pages which you can see in examples like on the Virtual Design Master site.

The entire Jekyll documentation and information site is helpful to give you a sense of how to get started with a basic Jekyll implementation. If you’re looking for a place to host an example, GitHub pages also has a great getting started guide which you can find by scrolling through the main site at pages.github.com.

What about using AWS S3?

You may already know that Amazon Web Services S3 (Simple Storage Service) supports the hosting of static websites very easily by simply enabling the feature within the properties of your S3 bucket:
easy fix. Each time that you run the `jekyll serve` command, a new static version of the site is created. If you run the command, you’ll see the output showing it has rendered the site. Simply run and then exit the server:

```
vagrant@jekyllsandbox:~/vagrant/sites/virtualdesignmaster.github.io$ jekyll serve
Configuration file: /vagrant/sites/virtualdesignmaster.github.io/config.yml
Destination: /vagrant/sites/virtualdesignmaster.github.io/_site
Incremental build: disabled. Enable with --incremental
Generating...
  done in 6.423 seconds
Auto-regeneration: enabled for /vagrant/sites/virtualdesignmaster.github.io/
Configuration file: /vagrant/sites/virtualdesignmaster.github.io/config.yml
Server running... press ctrl-c to stop.
```

Back in your filesystem you can find the updated file structure under the `_site` folder:

```
Name
  ├── _data
  │    ├── .DS_Store
  │    └── index.html
  ├── _includes
  │    ├── _404.html
  │    ├── _background.html
  │    ├── _break.html
  │    ├── _favicon.ico
  │    ├── _logo.png
  │    ├── _preloader.css
  │    ├── _privacy.html
  │    ├── _sponsors.html
  │    ├── _static.html
  │    ├── _template.html
  │    └── _top.html
  ├── _layouts
  │    ├── _404.html
  │    ├── _about.html
  │    ├── _all.html
  │    ├── _blog.html
  │    ├── _break.html
  │    ├── _contact.html
  │    ├── _design.html
  │    ├── _doc.html
  │    ├── _event.html
  │    ├── _faq.html
  │    ├── _fonts.html
  │    ├── _gallery.html
  │    ├── _images.html
  │    ├── _javascripts.html
  │    ├── _layout.html
  │    ├── _logo.html
  │    ├── _menu.html
  │    ├── _news.html
  │    ├── _post.html
  │    ├── _privacy.html
  │    ├── _privacy.html
  │    ├── _projects.html
  │    ├── _scheme.html
  │    ├── _services.html
  │    ├── _site.html
  │    ├── _sketch.html
  │    ├── _social.html
  │    ├── _sponsor.html
  │    ├── _terms.html
  │    ├── _top.html
  │    └── _utils.html
  ├── _posts
  │    ├── about.html
  │    ├── blog.html
  │    ├── contact.html
  │    ├── design.html
  │    ├── doc.html
  │    ├── gallery.html
  │    ├── images.html
  │    ├── javascripts.html
  │    ├── layout.html
  │    ├── logo.html
  │    ├── menu.html
  │    ├── news.html
  │    ├── post.html
  │    ├── privacy.html
  │    ├── projects.html
  │    ├── scheme.html
  │    ├── services.html
  │    ├── site.html
  │    ├── sketch.html
  │    ├── social.html
  │    ├── sponsor.html
  │    ├── terms.html
  │    └── top.html
  ├── _sass
  │    ├── _404.scss
  │    ├── _about.scss
  │    ├── _all.scss
  │    ├── _blog.scss
  │    ├── _break.scss
  │    ├── _contact.scss
  │    ├── _design.scss
  │    ├── _doc.scss
  │    ├── _event.scss
  │    ├── _faq.scss
  │    ├── _fonts.scss
  │    ├── _gallery.scss
  │    ├── _images.scss
  │    ├── _javascripts.scss
  │    ├── _layout.scss
  │    ├── _logo.scss
  │    ├── _menu.scss
  │    ├── _news.scss
  │    ├── _post.scss
  │    ├── _privacy.scss
  │    ├── _privacy.scss
  │    ├── _projects.scss
  │    ├── _scheme.scss
  │    ├── _services.scss
  │    ├── _site.scss
  │    ├── _sketch.scss
  │    ├── _social.scss
  │    ├── _sponsor.scss
  │    ├── _terms.scss
  │    └── _top.scss
  └── _site
      ├── 2016
      │    ├── Bath.jpg
      │    ├── Bath.jpg
      │    └── Bath.jpg
      └── blog
          ├── about.html
          ├── blog.html
          ├── contact.html
          ├── design.html
          ├── doc.html
          ├── gallery.html
          ├── images.html
          ├── javascripts.html
          ├── layout.html
          ├── logo.html
          ├── menu.html
          ├── news.html
          ├── post.html
          ├── privacy.html
          ├── projects.html
          ├── scheme.html
          ├── services.html
          ├── site.html
          ├── sketch.html
          └── social.html
```

All that I have to do is upload these files into the AWS S3 bucket and enable them as publicly viewable.

**NOTE:** This is a terribly manual process, but is mean to illustrate the shortest path to using Jekyll locally and running the real site off of AWS S3. There will be follow up posts to show how to fully automate this as well.

You’ve already set up your S3 bucket and configured the web access. If you haven’t, go ahead and do that based on the image at the top of the blog under the S3 bucket properties.

**Uploading your Content to S3 for Static Web Hosting**

Click the Actions button at the top and select Upload to bring up the file upload dialog box. You can drag your files over from your Finder (or Windows Explorer).
Make sure to use the Set Details button rather than the Start Upload button. We have a few more options to configure along the way:

Let’s select the Use Standard Storage option and click the Set Permissions button to enable the viewing permissions next:

This is the important step. You have to enable the Make Public option in order to allow the files to be viewable for anonymous (aka everyone other than you) on S3:
You can set the metadata on this if you’d like as well, or just continue on. You can see that the option is already checked by default to figure out content automatically:

Now it’s time for testing! Use the URL provided in the S3 bucket properties to test out the name resolution and that the content is hosted. You’ve got yourself a life site which was generated by Jekyll and is hosted on S3 which has a very light cost for operations:
You can set up your DNS to point to the S3 bucket using Route 53 using the GUI like I’ve shown in the past, but rather than redirecting like the example shows you can just leave it pointing to the native S3 bucket.

Updating the site can be done by uploading the content again and allowing overwrites. What is even more interesting is that we can now prepare for a proper version control and continuous deployment configuration using Jekyll as the build mechanism, and AWS S3 as the hosting platform.

---

**AWS Route53 Step-by-Step - Redirecting Domains GUI**

While I’ve covered this one before with a specific two-step approach to do simple redirect. This will repeat some of those steps with a new domain pointing to an alternate web site URL. This is a real use-case example where I’ve bought a new domain that I want redirected to DiscoPosse.com as the target.

Sample source zone: EricWright.ca

Sample target website: DiscoPosse.com

**Using Route53 to Redirect to a New URL**

My sample domain was registered with Route53. You can see the registration message here:

<table>
<thead>
<tr>
<th>Resource</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>ericwright.ca</td>
<td>Domain registration successful</td>
</tr>
</tbody>
</table>

A Hosted Zone is created automatically as part of the registration process:
Highlight the radio button and use the **Go to Record Sets** button. That lets us create a new entry with the **Create Record Set** button. Because we want to redirect the default www zone, let’s create a CNAME record for that and directly assign **discoposse.com** in the **Value** field:

We need to redirect the root domain of ericwright.ca as well, but we can’t use a CNAME for a root record. This means we need to create a S3 bucket for the root zone. Name it after the domain you’re redirecting:
Set up the **Static Website Hosting** section in the properties window for the S3 bucket, and put the target domain under the **Redirect all requests to another host name** section:

We may have to exit and reenter the Route53 zone to refresh, so once you’re done with that, set up a new A record, click the **Alias** radio button to **Yes**, and click the **Alias Target** field to trigger the drop down list where you can select the **S3 website endpoint** that matches the domain root:
Wait for a few minutes. 10-15 minutes will ensure that the TTL for the zone and record are passed and it forces a fresh request to the root domain.

**Using the AWS CLI (aka AWS Shell) to Query Route53 Hosted Zones**

You will need to install the [AWS CLI](https://awscli.amazonaws.com) on your machine, or you can also use [my sandbox method to run it inside a VirtualBox VM](https://aws.amazon.com/virtualbox/) to save installing in your workstation.

Once configured, we will use the following command to list all of the zones:

```
aws route53 list-hosted-zones-by-name
```
You will see all of the available zones come back. Clearly that is more detail than we want, so let’s add a couple of parameters which will reduce the output to just the zone we want --dns-name ericwright.ca. (note the trailing dot is important). That still returns a series of zones, so we also add the --max-items 1 parameter as well:

We can confirm the redirects using the cURL command for both domain names using the cURL command and the -IL parameter to show the HTTP response details:
In our next post, we will be doing some DNS record management using the AWS CLI to emulate the changes we did in the GUI in this article.

Unable to Delete Empty Elastic Beanstalk S3 Bucket

For those who are doing AWS work among the different projects, you will most likely do some storage on S3 (Simple Storage Service) for templates and logs. Each AWS service has the ability to write its configuration and logs to S3 and is usually a part of the setup wizard.

Sometimes the permissions set by the AWS wizard may leave you with some challenges. A common and simple example is when using AWS Elastic Beanstalk. When you clear out an Elastic Beanstalk configuration, the S3 bucket is left behind because it is not deleted as part of the removal process.
Normally, we just select the bucket and then you can empty it and delete it. This is what happens instead. First, select your bucket:

Once selected, we then choose the **Delete Bucket** option from the **Actions** button:

Then we are disappointed by seeing this error message:

Access Denied?! That shouldn't be the case. I'm using an account that does have enhanced privileges, and have even attempted it using the root level account for my entire AWS environment. **NOTE**: It's not recommended to use the root account, but I did try it to prove the point.

**Fixing the S3 Bucket Access Denied Issue**

The issue is a simple one as it turns out. Open up the **properties** for the bucket and click the **Edit bucket policy** button:
When the bucket is created by the system, it is created with a specific bucket policy that has been set to deny the `s3:DeleteBucket` action:

```json
{
    "Version": "2008-10-17",
    "Statement": [
        {
            "Sid": "eb-58950a",
            "Effect": "Deny",
            "Principal": {
                "AWS": "*"
            },
            "Action": "s3:DeleteBucket",
            "Resource": "arn:aws:s3:::elasticbeanstalk-us-east-1-507258549789"
        }
    ]
}
```

That’s a safety measure so that we don’t accidentally remove the contents which could be driving an active Elastic Beanstalk configuration. Change the `Deny` effect to `Allow` in the JSON editor and save the policy:

```json
{
    "Version": "2008-10-17",
    "Statement": [
        {
            "Sid": "eb-58950a",
            "Effect": "Allow",
            "Principal": {
                "AWS": "*"
            },
            "Action": "s3:DeleteBucket",
            "Resource": "arn:aws:s3:::elasticbeanstalk-us-east-1-507258549789"
        }
    ]
}
```

Once you’ve saved the policy, go ahead with the `Delete bucket` process under the **Actions** menu again, and you will see a much more appropriate response. This time you will see a **Done** result in the results window.

![Done]

This is one of those oddities around saving ourselves from ourselves by making sure we don’t accidentally delete things. Sometimes we really do want to delete stuff.