108
Hs
Hassium
[269]

Electronic Publishing

Hassium Labs

Table of Contents

Lab 1 - Electronic Publishing Overview

This lab was created to demonstrate the way in which Hassium Labs content is produced and published.

Source Code

All of the source for this lab is available from the Hassium Labs GitHub Organization

Part One - Setup

The first step in the publishing process is to gather the needed tools to process the content to be published.

Part 1.1 - Components

The components involved in the publishing process are the content which is marked up using the simple but powerful AsciiDoc and the Gitbook toolchain of publishing libraries. To further simplify the collection and installation of the toolchain components I've encapsulated them all into a Docker image which is then fronted with a script to run the various commands to keep the tools that have to be locally installed to a minimum.

Locally Installed

- Make - Wraps docker commands
- Git - Source Control
- Docker - Text processing toolchains installed on containers

Publishing toolchains

Gitbook is used to publish the labs and Hugo to publish the website. Both of these can be installed locally but for ease of integration with continuous integration services they've been installed into docker images

- GitBook - Lab content published as HTML, PDF, MOBI and EPUB
- Hugo - Website publishing

Services

- GitHub - Git Repos
- Docker Hub - Docker Repos
- Travis CI - Build service
- AWS - Infrastructure
  - S3 - Simple Storage Service - Content Storage
  - CloudFront - Content Distribution
  - Route 53 - DNS

Part Two - Content Creation

With the tools installed we're ready to create and annotate the content for publishing.

Part 2.1 - Markup Content

The website uses Markdown to annotate content and the labs use the slightly more complex AsciiDoc to be able to do some more advanced text formatting. Regardless of the language used the ability to focus on content rather than the formatting greatly accelerates the writing process. The links above show the markup capabilities for each of the languages, Markdown has the advantage of being incredibly simple to learn but has limited capabilities. AsciiDoc can be used in a simplistic way to do basic formatting but has advanced capabilities when needed.

Part Three - Compilation

The payoff for going through the markup process is that the tools can compile the content into many different types of output formats by applying different styles. Each of the toolchains have been installed into docker containers so if you use the provided Docker and Make script to build artifacts you'll get the same output as when built using the automated method.

GitBook Dockerfile

Hugo Dockerfile

Part 3.1 - Output Formats

With GitBook, once content has been annotated it can be transformed into many different final products. Using the Gitbook toolchain we can produce HTML, PDF, MOBI and EPUB documents from a the same source content by applying different style sheets.

Part Four - Publishing

Part 4.1 - Serving Content from AWS

S3 Storage

In the git repo holding the content a naming convention using the names of branches correspond to a production and staging environment for the website. When code is pushed to the GitHub repo test branch a Travis CI build is kicked off which runs a deployments script which syncs the generated artifacts to a specific location within the S3 bucket.

Bucket file structure

```
hassiumlabs-website/
├── production
│   ├── labs
│   └── site
└── test
    ├── labs
    └── site
```

travis-deploy script

```
if [ "$TRAVIS_BRANCH" == "master" ]; then
  aws s3 sync _book/ s3://hassiumlabs-website/production/labs/lab-001-electronic-publishing/ --delete
fi

if [ "$TRAVIS_BRANCH" == "test" ]; then
  aws s3 sync _book/ s3://hassiumlabs-website/test/labs/lab-001-electronic-publishing/ --delete
fi
```

IAM Service User

To allow Travis CI to publish to S3 an IAM (Identity and Access Management) user with API credentials was created which allows read/write access to the S3 bucket in which the finished artifacts are staged. The user credentials can be configured within the Travis CI build and injected into the build as environment variables so you don’t have to hard-code credentials into a build script.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::hassiumlabs-website"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:DeleteObject"
            ],
            "Resource": [
                "arn:aws:s3:::hassiumlabs-website/*"
            ]
        }
    ]
}
```

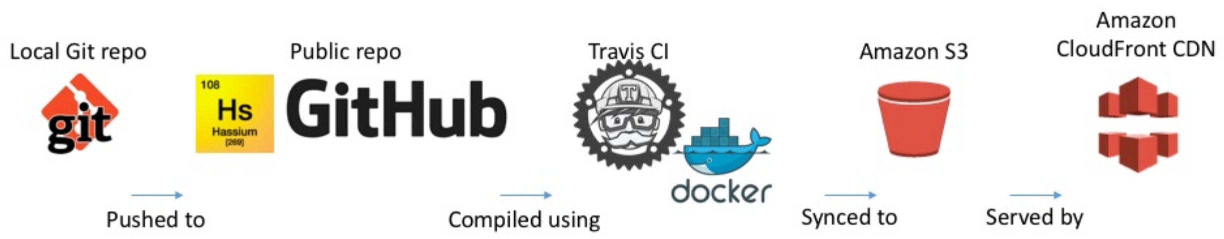
CloudFront Content Distribution Network

When configuring CloudFront, production and test distributions were created which both have base origin settings of the same bucket but use different origin paths within the bucket (/production/site or /test/labs). Each of the distributions may also have different settings like the Time To Live (TTL) which is much more aggressive for the production site to allow for more effective caching and rather short for the test site so updates can be previewed more rapidly.

Route53 DNS

Both the production and test CloudFront distributions have their own DNS records which in Route53 are set to A and AAAA (IPV6) alias records pointing to the CloudFront distribution names. Both the test and production content is stored in the same S3 bucket and synchronization is done based on the git branch used. The travis-deploy script shown above is the mechanism that examines the branch being pushed to the GitHub repo and uses the IAM service user credentials saved within the Travis CI job to push the content to the appropriate location within the S3 bucket.

Data Flow


Local Git repo
Pushed to
Public repo
Compiled using
Travis CI
Amazon S3
Synced to
Amazon CloudFront CDN
Served by