trevor.bramwell.net Documentation

Release

Trevor Bramwell

July 07, 2016

Contents

CHAPTER 1

About Me

My name is Trevor Bramwell, and I am currently a student at Oregon State University. I work as a Student Software Developer and Systems Engineer at the Oregon State University Open Source Lab (OSUOSL). Most of my time there has been spent working on Ganeti Web Manager: A Django web frontend for managing Ganeti clusters.

When I have free time, I like hacking on open source projects, working on my book, or dancing.

CHAPTER 2

Posts

2.1 Setting up LetsEncrypt with ReadTheDocs

With the recent announcement of LetEncrypt coming out of beta, I figured it was time to give it a shot. I've always been bothered that this site wasn't served over HTTPS, and LetsEncrypt provides a simple and straight forward way to make it happen.

This guide documents the steps I took to transition my site on ReadTheDocs (RTD) to use HTTPS, and closely follows NGINX's guide on setting up LetEncrypt.

Before you follow this guide, you should have a few things already in place:

1. A site hosted on RTD that you'd like to setup HTTPS for (ex: trevorbramwellnet)
2. A server with nginx, and git installed.
3. Control of DNS for your site.

2.1.1 Alternate Domains

ReadTheDocs allows for serving documentation from an alternate domain using a DNS CNAME. If you're unfamiliar with DNS, a CNAME record is simply a symlink from one hostname to another.

```
bar.example.com  A          192.168.2.23
foo.example.com  CNAME   bar.example.com
```

Here's the problem though: If you setup a CNAME for your documentation hosted on RTD, you can't serve it over HTTPS. This is because RTD uses a star cert (*.readthedocs.org) to enable HTTPS on all the documentation projects they host. When your browser redirects you to the real domain, it still looks for an SSL certificate for the original domain. So in order to get HTTPS to work, you'll instead need to setup NGINX as a proxy.

2.1.2 NGINX

To install NGINX on a Debian based operating system, you run:

```
$ sudo apt-get install nginx
```

Configuring NGINX is done through files under `/etc/nginx`. Your `/etc/nginx/conf.d/default` file should look something like this:

Replace `exampleproject` in `proxy_pass` and `proxy_set_header` with your RTD site name, and replace `example.com` with your domain.

Reload NGINX:

```
# service nginx reload
```

And ensure your site correctly proxies requests to RTD:

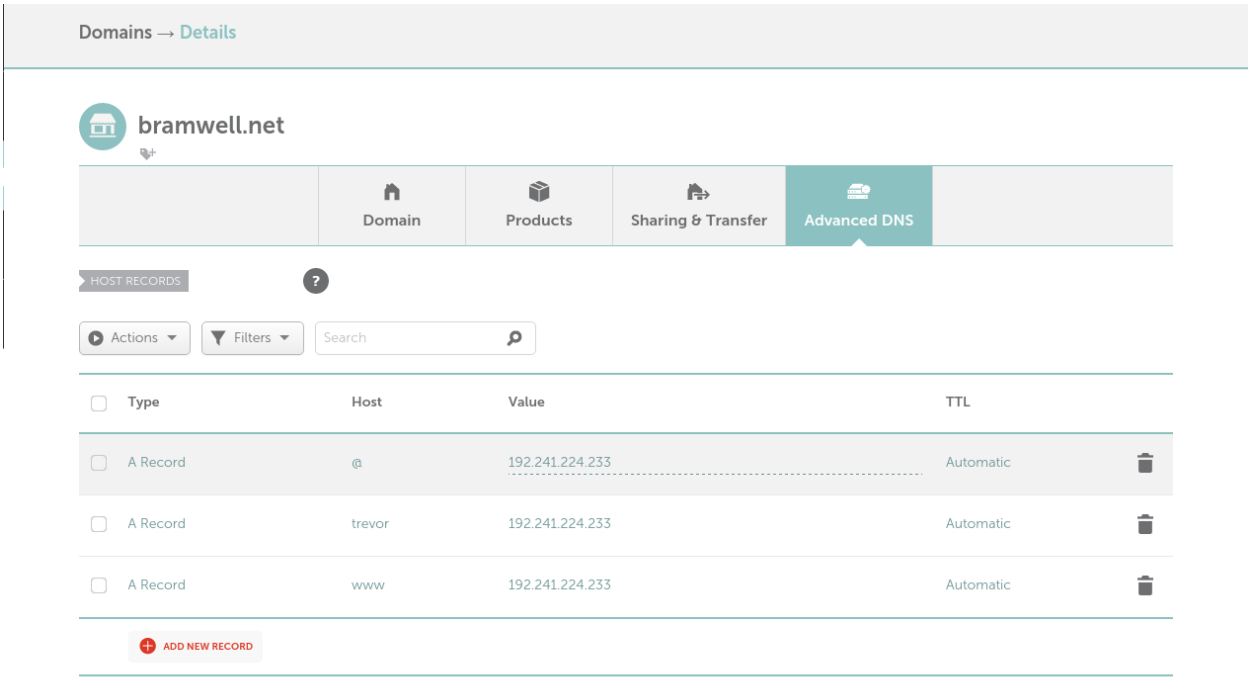
```
$ curl -IL http://127.0.0.1:80/

HTTP/1.1 302 FOUND
Server: nginx/1.2.1
Date: Mon, 25 Apr 2016 13:23:13 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 229
Connection: keep-alive
X-Deity: web01
X-Served: Flask
Location: http://127.0.0.1/en/latest/
X-Redirct-From: Flask

HTTP/1.1 200 OK
Server: nginx/1.2.1
Date: Mon, 25 Apr 2016 13:23:13 GMT
Content-Type: text/html
Content-Length: 22769
Connection: keep-alive
X-Deity: web03
Vary: Accept-Encoding
X-Served: nginx-via-django-cname-user-builds
Accept-Ranges: bytes
ETag: "570aef2a-58f1"
Domain=readthedocs.org; expires=Mon, 09-May-2016 13:23:13 GMT;
httponly; Max-Age=1209600; Path=/
Last-Modified: Mon, 11 Apr 2016 00:26:18 GMT
```

2.1.3 Point DNS to Server

If you haven't already done so, you'll need to update DNS to direct traffic to your server hosting NGINX. For example, this domain (which is hosted by Namecheap) has the following records:



You should now be able to hit `example.com` and be recieve your site hosted at `exampleproject.readthedocs.org`.

2.1.4 Install the LetsEncrypt Client

The LetsEncrypt team has been working to get packages for most major distributions, and recently the EFF subsumed the LetsEncrypt client with certbot. If a package is not available for your linux distribution (either through apt or yum) you can still use git to install it.

```
$ sudo mkdir /opt/
$ sudo chown $USER:$USER /opt
$ git clone https://github.com/letsencrypt/letsencrypt /opt/letsencrypt
$ cd /opt/letsencrypt/
$ ./letsencrypt-auto
```

Running `letsencrypt-auto` will bootstrap your system, through sudo, with the packages required to run LetsEncrypt.

2.1.5 Configure NGINX - ACME Challenge

Verifying your domain is done through placing a file in a location that the LetsEncrypt server can access. We will do this by adding the route `/.well-known/acme-challenge/` to NGINX, and serving `/usr/share/nginx/www` (the default NGINX root) from that location.

```
server {
    # ...

    location ~ ^/.well-known/acme-challenge/ {
        root /usr/share/nginx/www;
    }
```

```
    location / {
      # ...
    }
}
```

2.1.6 (Optional) Configure LetsEncrypt

In order to serve the correct file, we need to configure LetsEncrypt to know which domains to validate, along with where the challenge key it generates should go. This is done by using a configuration file, but could also be done by directly calling the client.

2.1.7 Obtain Certificates

/etc/letsencrypt/live

```
./letsencrypt-auto --config /etc/letsencrypt/config/bramwell.net.conf certonly
```

```
$ cd /opt/letsencrypt
$ ./letsencrypt-auto certonly --agree-tos --email foo@example.com \
  -d bramwell.net \
  -d trevor.bramwell.net \
  -d www.bramwell.net \
  -w /usr/share/nginx/www
  --test-cert
```

2.1.8 Update NGINX Configuration

- Generate 2048 DH Params

```
$ openssl dhparam -out /etc/letsencrypt/live/bramwell.net/dhparams.pem 2048
```

- Add HTTPS server to NGINX
- Ensure HTTPS works
- Rewrite HTTP traffic to HTTPS

```
server {
    listen 80 default;

    # ...

    location / {
        rewrite ^ https://$server_name$request_uri? permanent;
    }
}
```

- Restart Nginx

```
# nginx -t && service nginx restart
```

2.1.9 Setup LetsEncrypt Renewal

- Test renewal

```
$ ./letsencrypt-auto renew --dry-run
```

- Create Cron script

2.2 Full Files

2.2.1 /etc/nginx/conf.d/bramwell.net.conf

```
server {
    # listen 192.241.224.233:80 default;
    listen 80 default;
    server_name bramwell.net trevor.bramwell.net www.bramwell.net;

    location ~ ^/.well-known/acme-challenge/ {
        root /usr/share/nginx/www;
    }

    location / {
        rewrite ^ https://$server_name$request_uri? permanent;
    }
}

server {
    listen 192.241.224.233:443 ssl;
    server_name bramwell.net trevor.bramwell.net www.bramwell.net;

    if ($host = 'www.bramwell.net') {
        rewrite ^/(.*)$ https://trevor.bramwell.net/$1 permanent;
    }

    add_header Strict-Transport-Security "max-age=31536000";

    ssl on;
    ssl_certificate /etc/letsencrypt/live/bramwell.net/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/bramwell.net/privkey.pem;
    ssl_dhparam /etc/letsencrypt/live/bramwell.net/dhparams.pem;

    include /etc/letsencrypt/options-ssl-nginx.conf;

    location / {
        proxy_pass https://trevorbramwellnet.readthedocs.org:443;
        proxy_set_header Host $http_host;
        proxy_set_header X-Forwarded-Proto http;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Scheme $scheme;
        proxy_set_header X-RTD-SLUG trevorbramwellnet;
        proxy_connect_timeout 10s;
        proxy_read_timeout 20s;
    }
}
```

2.2.2 /etc/cron.monthly/letsencrypt

```
#! /bin/sh

# Renew certs
cd /opt/letsencrypt && ./letsencrypt-auto renew

# make sure nginx picks them up
[ $? -eq 0 ] && ( nginx -t && service nginx restart )
```

2.2.3 /etc/letsencrypt/renewal/bramwell.net.conf

```
cert = /etc/letsencrypt/live/bramwell.net/cert.pem
privkey = /etc/letsencrypt/live/bramwell.net/privkey.pem
chain = /etc/letsencrypt/live/bramwell.net/chain.pem
fullchain = /etc/letsencrypt/live/bramwell.net/fullchain.pem

# Options used in the renewal process
[renewalparams]
authenticator = webroot
installer = None
account = ACCOUNT_KEY
server = https://acme-v01.api.letsencrypt.org/directory
rsa_key_size = 4096
webroot_path = /usr/share/nginx/www,
[[webroot_map]]
bramwell.net = /usr/share/nginx/www
trevor.bramwell.net = /usr/share/nginx/www
www.bramwell.net = /usr/share/nginx/www
```

2.3 Field - extract fields from a file

I recently put the final touches on a python project I've been working on since February, and I am happy to announce the *0.2.0* release of field!

field is a command-line application for extracting fields from files. It was written to be a simpler version of `awk '{ print COLUMN; ... }'`, and address some of the shortcomings of cut, such as field ordering, whitespace squashing, and repeated output.

2.3.1 Examples

Extract the user, pid, cpu percent, and command from `ps`:

```
$ ps ux | field 1-3 11 | column -t
USER       PID   %CPU  COMMAND
bramwelt   2157  0.0   /usr/bin/gnome-keyring-daemon
bramwelt   2161  0.0   i3
bramwelt   2195  0.0   xscreensaver
bramwelt   2196  0.0   nm-applet
...
```

Extract pid, cpu percent, pid, and command (in that order):

```
$ ps ux | field 2,3,1,11 | column -t
PID   %CPU  USER      COMMAND
2157  0.0   bramwelt  /usr/bin/gnome-keyring-daemon
2161  0.0   bramwelt  i3
2195  0.0   bramwelt  xscreensaver
2196  0.0   bramwelt  nm-applet
...
```

Extract user, shell, homedir, uid and gid from `/etc/passwd` using *':'* as the delimiter:

```
$ field -f /etc/passwd -d':' 1 7-6 4,3 | column -t
root    /bin/bash          /root      0      0
daemon  /usr/sbin/nologin  /usr/sbin  1      1
bin     /usr/sbin/nologin  /bin       2      2
sys     /usr/sbin/nologin  /dev       3      3
sync    /bin/sync          /bin       65534  4
...
```

Further examples can be found in field's manpage.

2.3.2 Download

field can be installed using pip:

```
virtualenv field && source field/bin/activate
pip install field
```

For a site-wide install, use:

```
sudo pip install field
```

2.3.3 Contributing

Field is free software and licensed under the GLPv3 (*or later*). The source can be found on github. If you find any bugs or would like to suggest improvements, please submit an issue and I'll do my best to address it!

2.4 Distribute Native Packages

date Aug 28, 2014

author Trevor Bramwell

As a system administrator I deploy a lot of code. As web developer I write a lot of code. Yet these two parts of me are always at odds.

When working on web applications the system administrator in me says "Just ship it!"

While the web developer screams "It's not perfect yet!"

And when the code finally gets deployed, things inevitably fall over in the process.

My years of experience being both a developer and system administrator have taught me that the best way to distribute applications is by using native packages.

Turning unmanagable deployments...

into well oiled machines.

Native packages provide three major benefits over other deployment strategies:

1. Security
2. Modularity
3. Maintainability

2.4.1 1. Security

The first benefit native package provide is security. This is an optional benefit, as signing packages does not preclude them from being deployed.

Every time I have to install RVM, I die a little inside.

The RVM website provides a single method of installation:

```
$ \curl -sSL https://get.rvm.io | bash -s stable
```

This is horribly insecure, dangerous, and the definition of remote code execution.

Though RVM encourages installation over HTTPS, SSL is not enough security for software distribution, because does not verify the integrity of the content.

By using signed native packages instead, you can verify two things:

1. The package contents haven't changed since it was created.
2. The package can only have come from the owner of the signing key.

This is because servers installing the package will already contain the author's public key.

2.4.2 2. Modularity

Modularity is the second benefit native packages provide.

Native package create modularity by providing a clear distinction between the build and deployment steps of a web application. This leads to a highly scalable and continuously deployable application, which falls in line with The Twelve-Factor App definition.

Applications that don't have a clear distinction between these processes tend to have deployments that organically grow into complex monstrosities.

This complexity can be seen clearly around the upgrade and rollback processes. Modularity comes from taking these processes and sectioning them into before/after install and remove scripts. Instead of being ran by an external process that requires elevated privileges to a production server, these scripts - which are packaged with the application - execute during upgrades (installing a new version) and rollbacks (downgrading versions).

2.4.3 3. Maintainability

The final benefit native packages provide, is maintainability.

Instead of maintaining complex execution definitions or a plethora of SCM resources in configuration management, system administrators have only to write a few simple resources: install application, upload configurations, restart services.

This leaves them with more time to fix important issues (like putting out fires), and keeps them from having to spend multiple hours deploying new versions.

2.4.4 Where it All Breaks Down

Even with all these benefits, native packages are not a silver bullet.

A new (but simple) piece of infrastructure will need to be setup to host package. If binaries are being signed, public keys will need to be shipped with server images, and signing will need to be integrated into the build processes. Migrating existing deployment architecture will take time, along with the creation of packaging scripts.

And finally, multiple packages may need to be created for any server that runs multiple versions of an application.

2.4.5 Concluding Remarks

If learning *rpmbuild* or *debuild* feel like it requires a robe and wizard hat (because it surely does), tools like fpm make building packages extremely simple.

Overall native package managers provide web application with greater security, modularity, and maintainability. They also reduce deployment times, and make system administrators lives easier. And even though they introduce a new set of problems, theses can be resolved with few extra resources and a little bit of time. The benefits of distributing native packages far outweigh the costs, and will eventually save large amounts time spent on development and administration.

2.5 Getting Started In Open Source

date Mar 24, 2014

author Trevor Bramwell

I have been an open source contributor for over 3 years now, and ever since I started I have heard this question come up over and over again:

```
How do I get involved in open source?
```

Here are 5 things I have learned through my experience that I feel will help answer this question.

2.5.1 1. Scratch the Itch

The next question I usually hear is:

```
How do I choose a project to contribute to?
```

The answer to this question is simple: Just choose one.

Choose a project, any project, any project at all. As long as it's something you find fun and interesting, you won't fail in making a successful contribution.

2.5.2 2. Take Ownership

Once you've found a project, tell yourself you're a contributor. Heck, tell your parents, your significant other, your dog. That's it, you're a contributor now! This is the *Fake It Till You Make It* or *Self-Fulfilling Prophecy* method.

For the first few days or weeks you commit to working on the project, most of your time will be spent reading, not coding, not writing documentation, not building cool things, not really *contributing*.

Telling yourself you are a contributor right away will help you get over this initial hurdle of feeling unhelpful. Since you've already made it, there is nothing left to prove.

2.5.3 3. Set Aside Time

Time scheduled is time saved, and life is full of distractions. That time you've been looking for, you know that time you "don't have" to contribute to the project you want? You create that time by scheduling it in your calendar.

When you schedule time to work on projects, you will find that time magically appearing. It can be as little as 30 minutes, or as much as a weekend, just as long as you schedule it. The time you think you "don't have" exists in the many random minutes you spend browsing the web, shopping for those shoes you don't need, reading hacker news, or watching other people's lives go by on Facebook.

2.5.4 4. Provide Support

The most useful contributor is the one who builds community.

Most open source project are run by a small team of volunteers who, like you, have a very limited supply of time. The time they have they spend programming, or building the project, because it is the best investment they can make. Anything other than the code gets neglected because of this.

When you spend time providing support through IRC, email, or the bug tracker, you learn more about the project yourself. Providing support is alway appreciated, and will make you visible to the community and leadership.

2.5.5 5. Document ALL the things!

Through providing support you are helping people find solutions to their problems. You might realize after a while you're solving the same problems over and over again.

Writing these solutions down will help save time for you and others. These can either be added to the project FAQ, or user documentation. That way in the future you can kindly point other to the solution, and spend your time answering other questions and documenting more solutions.

2.5.6 Conclusion

Finding a project that is fun and interesting to work on is the first step in a long journey to becoming an open source contributor. Taking ownership and setting aside time to contribute will help get you over the initial hurdle of reading. Providing support and documenting solutions will help others getting started with the project.

Best of luck on your contributions!