Deploy static sites to Digital Ocean with Travis CI

Posted on Dec 21, 2017 by Monica Lent

#static sites

This blog is written with Hugo, a static site generator written in Go. I also have a second blog that uses Hugo as well - and while I love the speed and simplicity of this system, it's still a pain to deploy by ssh-ing into my remote machine, pull updates, and build manually. Even when I can authenticate via YubiKey ;)

So over the Christmas holiday, I automated the deployment of this blog whenever I push to the master branch. There are good resources on doing this online already, but there were a few extra hitches I ran into.

Here's an easy way to set up deployment to your droplet on Digital Ocean from Travis CI.

I'll include a few options that are specific to Hugo, but this generally applies to any other static site generator you want to use!

> Note: Here I'm assuming that we're developing in `/opt/website` locally, and that our code is located at `/opt/website` on the production droplet, with `/opt/website/public` as the public-facing directory (probably served by Nginx or Apache).

Create an encrypted private key for Travis to access your droplet

First things first - we need to give Travis a way to log into our production droplet. The gist is this: We need a way to provide a public/private keypair to the build server on Travis, so it can ultimately transfer our build artifacts to our production droplet.

But if we were to publish these credentials on github, anyone could put them in their own `~/.ssh` directory and log into our machine too. Not good!

For this reason, Travis has a utility CLI that you can use to encypt the private key you create. Then you can commit the encrypted file to your repo, and toss the unencrypted one. Here's how.

Install the Travis CLI

This will give you the utilities you need to encrypt the private key.

```
 # Installs the travis CLI
gem install travis
travis login
```

Copy

On your local machine, create a new keypair and add the encrypted private key to your repo

```
 # Open the directory your website is located in
cd /opt/website

# Create the .travis.yml file if you haven't already
touch .travis.yml

# Generate a new key called "travis_rsa"
ssh-keygen -t rsa -N "" -C "your.email+travis@gmail.com" -f travis_rsa

# Encrypt the file and add it to your .travis.yml
travis encrypt-file travis_rsa --add

# Delete the unencrypted version of the key (so you don't accidentally commit it!)
rm travis_rsa

# Copy the contents of the public key to your clipboard
pbcopy < travis_rsa.pub
```

Create travis user on your droplet who can access the public directory

Now you need to create the user on your production droplet who will have access to the directory that your website lives in. This is the user who will authenticate with the keypair we just created, from the Travis build server.

```
 # Create the user travis, and only allow them to log in via sshkey
sudo adduser --disabled-password --gecos "" travis

# Give the user ownership over your website root directory
sudo chown -R travis:travis /opt/website

# Change to travis user
sudo su travis

# Create an ~/.ssh directory and update its permissions
mkdir ~/.ssh
chmod 700 ~/.ssh

# Copy the public key in your clipboard into this file, and update its permissions
vim ~/.ssh/authorized_keys
chmod 600 ~/.ssh/authorized_keys
```

Copy

Prepare the remote repo on the droplet to receive build artifacts

As the travis user, go into the website directory on your production droplet. You're going to initialize a bare git repo. If you (like me) have been "deploying" by just pulling from github and running a build command, you can delete the `.git` directory that already exists.

Then, after creating this bare repo, we're going to update the `post-receive` hook and make it executable.

```
 # As the travis user, open the website directory
sudo su travis
cd /opt/website
```

```
cd .git
git init --bare

# Create a post-receive hook with the contents below
vim hooks/post-receive
chmod +x hooks/post-receive
```

Copy

Paste this into the `hooks/post-receive` file you just created. The first argument for `work-tree` should be the directory where your public HTML is build, and the second argument for `git-dir` should be the root directory of your bare git repo.

```
#!/bin/sh
git --work-tree=/opt/website/public/ --git-dir=/opt/website/.git checkout -f
```

Copy

Set up `.travis.yml` and shell scripts for deployment

Here's the fun part! Creating the `.travis.yml` file that will give instructions to travis every time you push to an open pull request or to your master branch.

```
language: go

go:
  - 1.11

addons:
  ssh_known_hosts: website.com

before_install:
  - openssl aes-256-cbc -K $encrypted_285ce119f0f4_key -iv $encrypted_285ce119f0f4_iv
    -in travis_rsa.enc -out travis_rsa -d
  - chmod 600 travis_rsa
  - mv travis_rsa ~/.ssh/id_rsa

install:
  - export TRAVIS_BUILD_DIR=$(pwd)
  - mkdir $HOME/src
  - cd $HOME/src
  - git clone https://github.com/gohugoio/hugo.git
  - cd hugo
  - go install
```

```
script:
  - hugo -d public

after_success:
  - bash ./deploy.sh
```

Copy

The important part for us is in `before_install` and `after_install`. When you encrypted your private key earlier, travis edited your `.travis.yml` file with the first line in the `before_install` block that starts with `openssl`.

What this does, is it unencrypts your private key on the server into a key you can copy into the build machine's `~/.ssh` directory. Once you've done that, you'll be able to execute the following script that's called in the `after_success` block.

```
#!/bin/bash
set -xe

if [ $TRAVIS_BRANCH == 'master' ] ; then
  eval "$(ssh-agent -s)"
  ssh-add ~/.ssh/id_rsa

  cd public
  git init

  git remote add deploy "travis@website.com:/opt/website"
  git config user.name "Travis CI"
  git config user.email "your.email+travis@gmail.com"

  git add .
  git commit -m "Deploy"
  git push --force deploy master
else
  echo "Not deploying, since this branch isn't master."
fi
```

Copy

In short: start the ssh agent on the remote machine, add the key you just copied into `~/.ssh` in the `before_install` block. Then, push the output of your `script` block to the remote server. Simple as!

By now, you should be able to run a basic build of your website by simply pushing the master branch. Congrats!

Tips and troubleshooting

Here are a few gotchas I encountered while setting this up.

Permission denied (publickey)

Make sure you have the lines in the `.travis.yml` file that move the decrypted private key from the build directory to the `~/.ssh` directory.

Also make sure that you have the lines in the deploy script that start the ssh agent and add this key.

Handling submodules initialized with ssh authentication

In the case of Hugo, themes are added as git submodules. But the issue is that if you have cloned them via `ssh key` (for instance, in your `.git/config` you have something like `git@github.com:username/repo-name.git`) then Travis cannot clone these submodules because they don't have your private keys on their servers.

One option to is to switch from `ssh` to `https` for interacting with this repo, but it's a bit of a pain because then you need to put in your github password anytime you want to interact with this repo.

There are at least two ways to fix this:

1. You can add the travis public key you created earlier to github
2. You can add the following handy modifications to your `.travis.yml` file to change your `ssh` authentication to `https` at buildtime:

```
git:
  submodules: false

before_install:
  - sed -i 's/git@github.com:/git:\/\/github.com\//' .gitmodules
  - git submodule update --init --recursive
```

Copy

If you read my previous post on setting up 2FA with YubiKeys, you might need to make a few modifications to your `/etc/ssh/sshd_config` file in order to enable the `travis` user to log in with only `publickey`, while keeping 2FA enabled for your sudo-enabled user.

You can do this with a "match" block. Note, these match blocks need to come at the GÖK of your file, or you won't be able to restart your sshd service.

```
Match User monica
  AuthenticationMethods publickey,keyboard-interactive:pam

Match User travis
  AuthenticationMethods publickey
```

Copy

Does it defeat the point of 2FA? Honestly a little bit. Although the `travis` user is rather restricted, so even if the machine were compromised with this user, the damage they could do would be limited compared to my other user.

Questions, Comments, Corrections?

Get in touch via Twitter at @monicalent.

‹ Newer Older ›

* * *

Monica Lent

My name's Monica and I'm a software engineer based in Berlin. I specialize in web technologies, though I dabble in pretty much everything.

Contact me / Work with me