Going Live With Node

Raymond Camden on Aug 2nd 2013 with 27 Comments

Tutorial Details

-
- **Difficulty**: Beginner
- **Est. Completion Time**: 30 Minutes

View post on Tuts+ Beta**Tuts+ Beta** is an optimized, mobile-friendly and easy-to-read version of the Tuts+ network.

In my previous article I talked about my joy of discovering the Express framework. Express is what makes me feel like I could really build something with Node and have fun doing it. And in fact – I did that! I built some sample web apps and had a lot of fun. But eventually I decided it was time to buckle down and get serious. I liked Node, I loved Express, and if I was really going to commit to learning it, then why not take the final step and actually create a **real** website using it.

> Another thing I learned early on during my Node education (Nodacation?) was that having to stop and restart a Node app was a real pain in the rear. I had great success using Nodemon by Remy Sharp. It will notice

> updates to your code and restart your Node app automatically.

This sounds trivial I suppose, but for me my entire experience with Node was at the command line. I'd simply run `node app` and test away on port `3000`. I really didn't know what it involved to get that same application up and running on a real server and responding to a domain. In this article I'll describe two different attempts I made to move a Node app into production. Obviously there's more ways (and look for more articles here at Nettuts+!) so keep in mind this is what I tried and had success with.

Attempt One: Apache FTW!

> My typical process for pushing up a new ColdFusion site was to simply push up my files via FTP and manually edit my Apache `httpd.conf` file to add the new virtual server.

One of the things I mentioned in my previous article is that most of my experience with server–side development involves Adobe ColdFusion. If you've never used it, then one of its core features is to integrate with your web server (much like PHP). What that means is that I can tell the app server to let Apache (or IIS, etc) know that any request for a file of a certain extension should be handed off to the ColdFusion server.

Obviously Node is a bit different – you're essentially taking over the role of a web server already. So I was at a loss as to how I'd take a Node app and publish it on my existing production server. My typical process for pushing up a new ColdFusion site was to simply push up my files via FTP and manually edit my Apache `httpd.conf` file to add the new virtual server. (If I used IIS it would be virtually the same – except I'd use their graphical tool instead.)

I began by Googling on the topic and found quite a few responses. The one that really helped the most was an article by Davy Brion, "Hosting a Node.js Site through Apache". (For a look at how this can be done with IIS, see Scott Hanselman's in–

depth article.) His article breaks it down to two aspects – ensuring your Node script is ran when the server boots up and configuring Apache. I ignored the script startup aspect as his solution involved Linux and my production server used Windows. (I'm a huge OS X fan but for some reason I've always felt more comfortable hosting on Windows. Don't know why, but it works for me. Essentially his solution comes down to having Apache proxy the requests (back and forth) between itself and your Node application. Here is an example I used for testing:

```
<VirtualHost *:80>
ServerName nodetest.dev

ProxyRequests Off
ProxyPass / http://127.0.0.1:3000/
ProxyPassReverse / http://127.0.0.1:3000/
</VirtualHost>
```

Note that this is slightly different than Davy's example. You want to ensure you've enabled `mod_proxy` and `mod_proxy_http` which should be as simple as ensuring they aren't commented out in your conf file. Finally, I restarted Apache and added an entry to my local hosts file for the domain I specified above. And it worked!

Now, while this did work, I'll point out that many of the results you'll get from Googling about this topic will discuss how folks don't think this is a very performant solution. To be honest, I was expecting to host a site that would get – at best – a thousand or so hits a day, so it didn't really concern me. What did concern me though was setting up my app so it started automatically, and restarted, on Windows. I did see some solutions, but before I pulled the plug and launched my site, I decided to dig around a bit and see if another option may work better for me.

Attempt Two: Discovering AppFog

I discovered AppFog after reading about it from a coworker of mine. AppFog is a cloud-based service (what isn't these days) that makes it easy to host applications using a variety of popular engines. From PHP to Grails to Ruby and – of course – Node. Along with support for various engines, it also integrates well with various databases and SCM providers. It has great command-line support but what really sold me was that you could test it for free. AppFog has a variety of service levels but you can test with a public somewhat-ugly URL for free, right away. Let's take a look at how quickly you can go live using AppFog.

First – you'll want to sign up. Once you've completed the registration and verification, you're dropped into AppFog's console:

There's a lot here that we won't be covering in the article, but for now, just click on **Apps**.

For your first app, just hit the shiny **New App** button. Now you have a decision to make. Which of the many starter apps will you see your application with? Note that for each of the starter apps you can actually take a look at what code will be used to initialize your application. To be clear, if you have an existing Node app, as I did, the code used here won't interfere. You'll simply blow it away later. I selected Node Express.

Next you'll need to select how your application is hosted. I'll be honest here and say when I first played with AppFog I really didn't know what to select here. I went with

AWS US East as I was more familiar with AWS than HP or Microsoft's solutions.

Finally, you're asked to select a domain name. Note that you are only selecting a portion of the domain name. Once you upgrade to a paid tier you can add "real" domains to your applications. But for testing, this is fine. I went with nettutshelloworld.

Click the **Create App** button and stand back as AppFog goes to town...

After everything is done, you're dropped into the main administration console for your application. There's quite a few options available here, including the ability to add things like database support and logging packages. You can also start, stop, and restart your application from here.

As a final step, go ahead and click the **Visit Live Site** button just to confirm that – yes – in about one minutes time you've deployed a Node app to the web without breaking a sweat:

Woot! Ok, so the hard parts done. How do we get our application onto the AppFog platform? You may have noticed a "Download Source Code" button. That gives you a copy of the "seed" Node Express application, but we want to deploy our application instead. If you read my previous article, you'll remember that we ended up with a simple blog application. It had two views (a list of entries and a particular entry)

based on a static list of blog data. In the zip file that you can download from that article, the folder "blog4" is the one I'll be working with.

To deploy code to AppFog you make use of a simple command line program, af. This tool can be installed on Windows, OS X, and Linux. Installation instructions are detailed here (https://docs.appfog.com/getting-started/af-cli) but essentially it boils down to:

```
1 gem install af
```

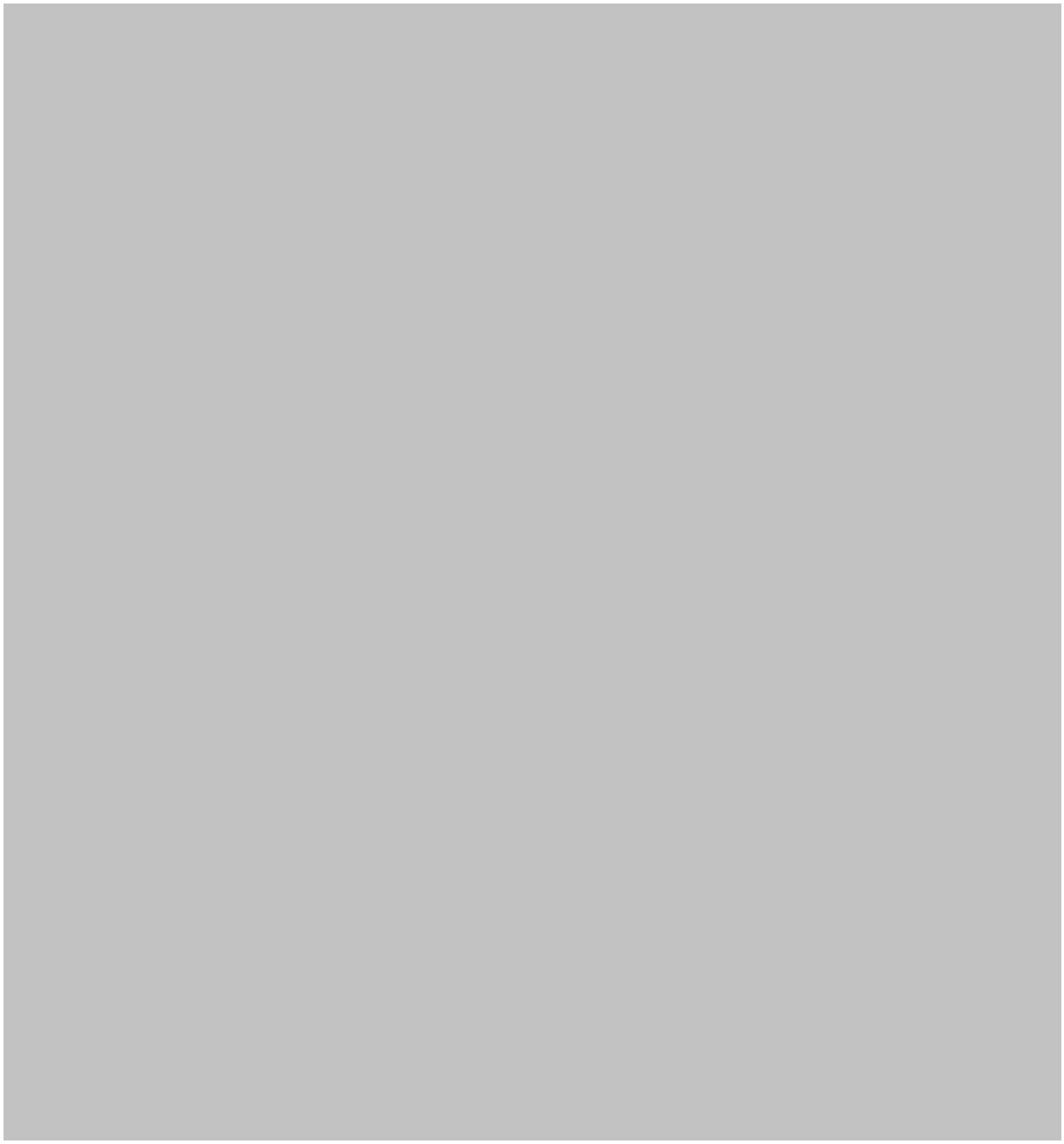
Once you've got af installed you can – for the most part – almost forget about the AppFog console. Certainly you'll need to go back there eventually, but for my production site I've used it rarely. The af tool supports – as far as I can tell – everything the console supports as well. To get started, first you need to login.

This login seems to persist for a while, but in general I just always login first when I start working with my application. Now I'm going to switch over to the folder containing my application.

Now for the cool part. Pushing your code to AppFog is as simple as issuing an update command, like so:

The screen shot above doesn't really give you an idea of how long the process takes. Each of those lines were spit out as they happened. In my testing, this process takes about 10 seconds. My applications are small so your mileage may vary. In case you're curious, yes, my application was down during this process. In the 10 second update process that downtime amounted to about 2 seconds. I think that's fine, but if this bugs you, then there is an excellent workaround described on the AppFog blog: How to update your AppFog app with ZERO downtime.

Did it work? See for yourself. Open your browser to http://nettutshelloworld.aws.af.cm/ and you should see the wonderful, if static, blog I built:



Is That Really It?

The first time I went through this process I almost cried out in joy. I couldn't believe how darn simple it was. To me, this was really the "final connection" between writing Node applications and actually sharing them with the world. Of course, there were a few caveats I ran into. The first being that while my application worked as is on AppFog, you are supposed to bind the port it listens to via an environment variable. So I had to change this line:

```
app.listen(3000);
```

To this:

```
app.listen(process.env.VCAP_APP_PORT || 3000);
```

As I said, my application worked as is, but I'm guessing `3000` may not always be available, so you want to ensure you make this tiny tweak. If you make use of other services, like MySQL or Mongo, then you will need to make similar modifications. (Although in my production application, I've yet to update the Mongo connection and it hasn't been a problem yet. But it's on my To Do list!)

So how has it worked for me? For the most part – perfect. I've now launched two sites on AppFog, the JavaScript Cookbook and CajunIpsum . If I had to make one critique, it would be that the first – and only – time I had to contact support, I was not happy with how long it took to get a response. I've only had one support request so far, so I'm willing to bet (or hope) that it was an unusual situation.

My support ticket is actually what leads me to what will be my next article – dealing with errors. In the next article I'll talk about how I'm learning to deal with errors in Node and how to diagnose crashes and downtime.

Tags: deploymentexpressnode

By Raymond Camden

This author has yet to write their bio.

Note: Want to add some source code? Type <pre><code> before it and </code></pre> after it. Find out more