Using Dnsmasq for local development on OS X

This is a quick guide to installing Dnsmasq on OS X and using it to redirect development sites to your local machine.

Posted by Thomas Sutton on October 23, 2013

Most web developers will be familiar with the process of updating your `/etc/hosts` file to direct traffic for `coolproject.dev` to `127.0.0.1`. Most will also be familiar with the problems of this approach:

- it requires a configuration change every time you add or remove a project; and
- it requires administration access to make the change.

Installing a local DNS server like Dnsmasq (http://www.thekelleys.org.uk/dnsmasq/doc.html) and configuring your system to use *that* server can make these configuration changes a thing of the past. In this post, I'll run through the process of:

1. Installing Dnsmasq on OS X.
2. Configuring Dnsmasq to respond to all `.dev` requests with `127.0.0.1`.

3. Configure OS X to send all `.dev` requests requests to Dnsmasq.

Before we get started, I should give you a warning: these instructions show you how to install new system software and change your system configuration. Like all such changes, you **should not proceed** unless you are confident you have understood them *and* that you can reverse the changes if needed.

Installing Dnsmasq

To quote the Dnsmasq project home page

> *Dnsmasq is a lightweight, easy to configure DNS forwarder and DHCP server [...] is targeted at home networks[.]*

There are plenty of ways to install Dnsmasq but my favourite (on OS X) is to use the Homebrew (http://brew.sh) package manager. Installing Homebrew is fairly simple but beyond my scope here.

Once you have Homebrew installed, using it to install Dnsmasq is easy:

```
# Update your homebrew installation
brew up
# Install dnsmasq
brew install dnsmasq
```

The installation process will output several commands that you can use to start Dnsmasq automatically with a default configuration. I used the following commands but you should use whichever commands `brew` tells you to:

```
# Copy the default configuration file.
cp $(brew list dnsmasq | grep /dnsmasq.conf.example$) /usr/local/etc/dnsmasq.conf
# Copy the daemon configuration file into place.
sudo cp $(brew list dnsmasq | grep /homebrew.mxcl.dnsmasq.plist$) /Library/LaunchDaemons/
# Start Dnsmasq automatically.
sudo launchctl load /Library/LaunchDaemons/homebrew.mxcl.dnsmasq.plist
```

Configuring Dnsmasq

Now that you have Dnsmasq installed and running, it's time to configure it! The configuration file lives at `/usr/local/etc/dnsmasq.conf` by default, so open this file in your favourite editor.

One the many, many things that Dnsmasq can do is compare DNS requests against a database of patterns and use these to determine the correct response. I use this functionality to match any request which ends in `.dev` and send `127.0.0.1` in response. The Dnsmasq configuration directive to do this is very simple:

```
address=/dev/127.0.0.1
```

Insert this into your `/usr/local/etc/dnsmasq.conf` file (I put it near the example `address=/double-click.net/127.0.0.1` entry just to keep them all together) and save the file.

You may need to restart Dnsmasq to get it to recognise this change. Restarting Dnsmasq is the same as any other service running under `launchd`:

```
sudo launchctl stop homebrew.mxcl.dnsmasq
sudo launchctl start homebrew.mxcl.dnsmasq
```

You can test Dnsmasq by sending it a DNS query using the `dig` utility. Pick a name ending in `dev` and use dig to query your new DNS server:

```
dig testing.testing.one.two.three.dev @127.0.0.1
```

You should get back a response something like:

```
;; ANSWER SECTION:
testing.testing.one.two.three.dev. 0 IN A       127.0.0.1
```

Configuring OS X

Now that you have a working DNS server you can configure your operating system to *use* it. There are two approaches to this:

1. Send *all* DNS queries to Dnsmasq.

2. Send *only* `.dev` queries to Dnsmasq.

The first approach is easy – just change your DNS settings in System Preferences – but probably won't work without additional changes to the Dnsmasq configuration.

The second is a bit more tricky, but not much. Most UNIX-like operating systems have a configuration file called `/etc/resolv.conf` which controls the way DNS queries are performed, including the default server to use for DNS queries (this is the setting that gets set automatically when you connect to a network or change your DNS server/s in System Preferences).

OS X also allows you to configure additional *resolvers* by creating configuration files in the `/etc/resolver/` directory. This directory probably won't exist on your system, so your first step should be to create it:

```
sudo mkdir -p /etc/resolver
```

Now you should create a new file in this directory for each resolver you want to configure. Each resolver corresponds – roughly and for our purposes – to a top-level domain like our `dev`. There a number of details you can configure for each resolver but I generally only bother with two:

- the *name* of the resolver (which corresponds to the *domain name* to be resolved); and
- the DNS server to be used.

For more information about these files, see the `resolver(5)` manual page:

```
man 5 resolver
```

Create a new file with *the same name* as your new top-level domain (I'm using `dev`, recall) in the `/etc/resolver/` directory and add a `nameserver` to it by running the following commands:

```
sudo tee /etc/resolver/dev >/dev/null <<EOF
nameserver 127.0.0.1
EOF
```

Here `dev` is the top-level domain name that I've configured Dnsmasq to respond to and `127.0.0.1` is the IP address of the server to use.

Once you've created this file, OS X will automatically read it and you're done!

Testing

Testing you new configuration is easy; just use `ping` check that you can now resolve some DNS names in your new top-level domain:

```
# Make sure you haven't broken your DNS.
ping -c 1 www.google.com
# Check that .dev names work
ping -c 1 this.is.a.test.dev
ping -c 1 iam.the.walrus.dev
```

You should see results that mention the IP address in your Dnsmasq configuration like this:

```
PING iam.the.walrus.dev (127.0.0.1): 56 data bytes
```

You can now just make up new DNS names under `.dev` whenever you please. Congratulations!

Maybe the next step on your journey should be learning how to do configure Apache virtual hosts automatically based on host names? If this sounds interesting, contact me using the details below and I'll write that article too.

This post was published on October 23, 2013 and last modified on June 30, 2015. It is tagged with: dns (../../tags/dns/), howto (../../tags/howto/), dnsmasq (../../tags/dnsmasq/), local (../../tags/local/), development (../../tags/development/).