AUG 25, 2014 57

How to Use Docker on OS X: The Missing Guide

Update

Aug 17, 2015
Hello. How are you? Thanks for stopping by.

It used to be tricky to get Docker working on OS X, which is why I wrote this here blog post. With the release of Docker 1.8, it just got way easier. Now there's a new thing, Docker Toolbox, that makes it super easy. You should install that instead of reading this post. Or you can read it if you want, I think it's a pretty good blog post and you'll learn some stuff. No pressure, though.

Well, see you later!

Have you heard of Docker? You probably have—everybody's talking about it. It's the new hotness. Even my dad's like, "what's Docker? I saw someone twitter about it on the Facebook. You should call your mom."

Docker is a program that makes running and managing containers super easy. It has the potential to change all aspects of server-side applications, from development and testing to deployment and scaling. It's pretty cool.

Recently, I've been working through The Docker Book. It's a top notch book and I highly recommend it, but I've had some problems running the examples on OS X. After a certain point, the book assumes you're using Linux and skips some of the extra configuration required to make the examples work on OS X. This isn't the book's fault; rather, it speaks to underlying issues with how Docker works on OS X.

This post is a walkthrough of the issues you'll face running Docker on OS X and the workarounds to deal with them. It's not meant to be a tutorial on Docker itself, but I encourage you to follow along and type in all the commands. You'll get a better understanding of how Docker works in general and on OS X specifically. Plus, if you decide to dig deeper into Docker on your Mac, you'll be saved hours of troubleshooting. Don't say I never gave you nothing.

First, let's talk about how Docker works and why running it on OS X no work so good.

HOW DOCKER WORKS

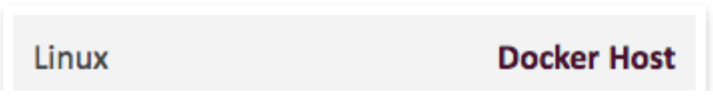
Docker is a client-server application. The Docker **server** is a daemon that does all the heavy lifting: building and downloading images, starting and stopping containers, and the like. It exposes a REST API for remote management.

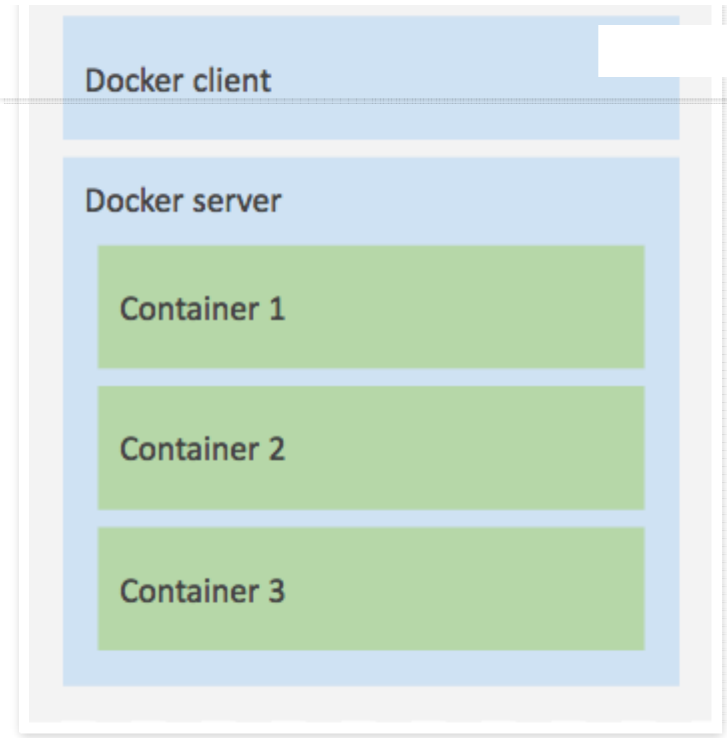
The Docker **client** is a command line program that communicates with the Docker server using the REST API. You will interact with Docker by using the client to send commands to the server.

The machine running the Docker server is called the Docker **host**. The host can be any machine—your laptop, a server in the Cloud™, etc—but, because Docker uses features only available to Linux, that machine must be running Linux (more specifically, the Linux kernel).

Docker on Linux

Suppose we want to run containers directly on our Linux laptop. Here's how it looks:





DOCKING ON LINUX

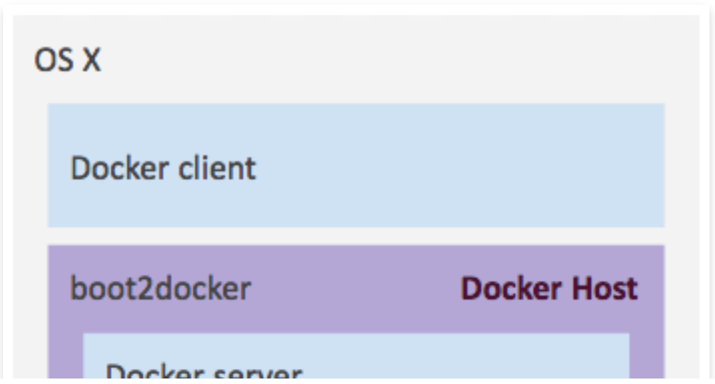
The laptop is running both the client and the server, thus making it the Docker host. Easy.

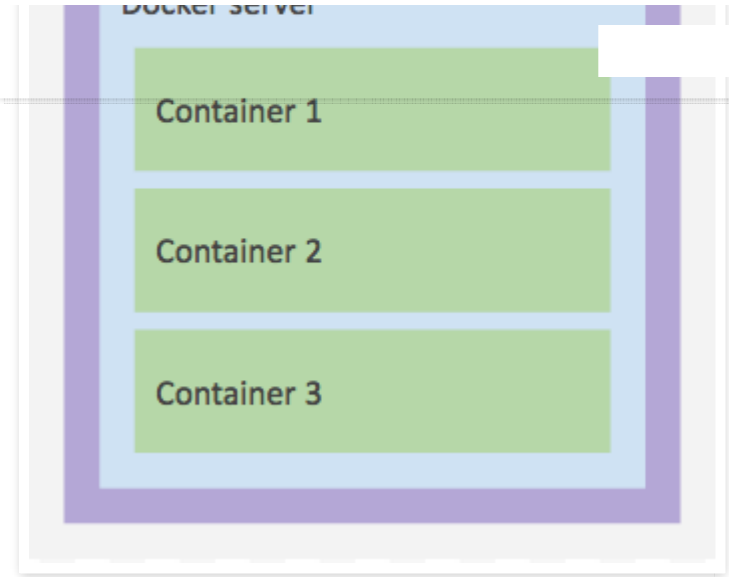
Docker on OS X

Here's the thing about OS X: it's not Linux. It doesn't have the kernel features required to run Docker containers natively. We still need to have Linux running somewhere.

Enter boot2docker. boot2docker is a "lightweight Linux distribution made specifically to run Docker containers." Spoiler alert: you're going to run it in a VM on your Mac.

Here's a diagram of how we'll use boot2docker:





DOCKING ON OS X

We'll run the Docker client natively on OS X, but the Docker server will run inside our boot2docker VM. This also means boot2docker, not OS X, is the Docker host.

Make sense? Let's install dat software.

INSTALLATION

Step 1: Install VirtualBox

Go here and do it. You don't need my help with that.

Step 2: Install Docker and boot2docker

You have two choices: the offical package from the Docker site or homebrew. I prefer homebrew because I like to manage my environment from the command line. The choice is yours.

```
> brew update
> brew install docker
> brew install boot2docker
```

Step 3: Initialize and start b ˆ ˈ ›cker

First, we need to initialize boot2docker (we only have to do this once):

```
> boot2docker init
2014/08/21 13:49:33 Downloading boot2docker ISO image...
    [ ... ]
2014/08/21 13:49:50 Done. Type `boot2docker up` to start the VM.
```

Next, we can start up the VM. Do like it says:

```
> boot2docker up
2014/08/21 13:51:29 Waiting for VM to be started...
.......
2014/08/21 13:51:50 Started.
2014/08/21 13:51:51   Trying to get IP one more time
2014/08/21 13:51:51 To connect the Docker client to the Docker daemon, please set:
2014/08/21 13:51:51     export DOCKER_HOST=tcp://192.168.59.103:2375
```

Step 4: Set the DOCKER_HOST environment variable

The Docker client assumes the Docker host is the current machine. We need to tell it to use our boot2docker VM by setting the `DOCKER_HOST` environment variable:

```
> export DOCKER_HOST=tcp://192.168.59.103:2375
```

☞ *Your VM might have a different IP address—use whatever* `boot2docker up` *told you to use. You probably want to add that environment variable to your shell config.*

Step 5: Profit

Let's test it out:

```
> docker info
Containers: 0
Images: 0
Storage Driver: aufs
 Root Dir: /mnt/sda1/var/lib/docker/aufs
 Dirs: 0
Execution Driver: native-0.2
Kernel Version: 3.15.3-tinycore64
Debug mode (server): true
Debug mode (client): false
Fds: 10
Goroutines: 10
EventsListeners: 0
Init Path: /usr/local/bin/docker
Sockets: [unix:///var/run/docker.sock tcp://0.0.0.0:2375]
```

Great success. To recap: we've set up a VirtualBox VM running boot2docker. The VM runs the Docker server, and we're communicating with it using the Docker client on OS X.

Bueno. Let's do some containers.

COMMON PROBLEMS

We have a "working" Docker installation. Let's see where it falls apart and how we can fix it.

Problem #1: Port Forwarding

The Problem: Docker forwards ports from the container to the host, which is boot2docker, not OS X.

Let's start a container running nginx:

```
> docker run -d -P --name web nginx
Unable to find image 'nginx' locally
Pulling repository nginx
```

```
    [ ... ]
0092c03e1eba5da5ccf9f858cf825af307aa24         1df431e22e03b4c3
```

This command starts a new container as a daemon (`-d`), automatically forwards the ports specified in the image (`-P`), gives it the name 'web' (`--name web`), and uses the `nginx` image. Our new container has the unique identifier `0092c03e1eba...` .

Verify the container is running:

```
> docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
0092c03e1eba        nginx:latest        nginx               44 seconds ago      Up 41 se
```

Under the PORTS heading, we can see our container exposes port 80, and Docker has forwarded this port from the container to a random port, 49153, on the host.

Let's curl our new site:

```
> curl localhost:49153
curl: (7) Failed connect to localhost:49153; Connection refused
```

It didn't work. Why?

Remember, Docker is mapping port 80 to port 49153 on the Docker host. If we were on Linux, our Docker host would be localhost, but we aren't, so it's not. It's our VM.

The Solution: Use the VM's IP address.

boot2docker comes with a command to get the IP address of the VM:

```
> boot2docker ip

The VM's Host only interface IP address is: 192.168.59.103
```

Let's plug that into our curl command:

```
> curl $(boot2docker ip):49153
```

```
The VM’s Host only interface IP addres
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
    [ ... ]
```

Success! Sort of. We got the web page, but we got `The VM’s Host only interface IP address is:`, too. What’s the deal with that nonsense.

Turns out, `boot2docker ip` outputs the IP address to standard output and `The VM's Host only interface IP address is:` to standard error. The `$(boot2docker ip)` subcommand captures standard output but not standard error, which still goes to the terminal. Scumbag boot2docker.

This is annoying. I am annoyed. Here’s a bash function to fix it:

```
docker-ip() {
  boot2docker ip 2> /dev/null
}
```

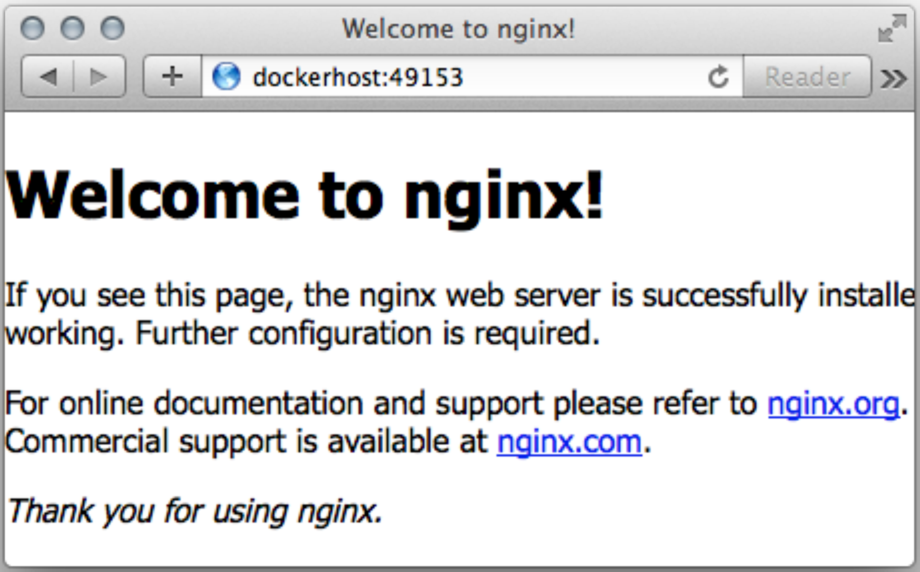
Stick that in your shell config, then use it like so:

```
> curl $(docker-ip):49153
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
    [ ... ]
```

Groovy. This gives us a reference for the IP address in the terminal, but it would be nice to have something similar for other apps, like the browser. Let’s add a `dockerhost` entry to the `/etc/hosts` file:

```
> echo $(docker-ip) dockerhost | sudo tee -a /etc/hosts
```

Now we can use it everywhere:



Welcome to nginx!

If you see this page, the nginx web server is successfully installe
working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

Great success. Make sure to stop and remove the container before continuing:

```
> docker stop web
> docker rm web
```

☞ *VirtualBox assigns IP addresses using DHCP, meaning the IP address could change. If you're only using one VM, it should always get the same IP, but if you're VMing on the reg, it could change. Fair warning.*

Bonus Alternate Solution: Forward all of Docker's ports from the VM to localhost.

If you really want to access your Docker containers via localhost, you can forward all of the ports in Docker's port range from the VM to localhost. Here's a bash script, taken from here, to do that:

```
#!/bin/bash

for i in {49000..49900}; do
```

```
    VBoxManage modifyvm "boot2dock[illegible]atpf1 "tcp-port$i,tcp,,$i,,$i";
    VBoxManage modifyvm "boot2dock[illegible]atpf1 "udp-port$i,udp,,$i,,$i";
done
```

By doing this, Docker will forward port 80 to, say, port 49153 on the VM, and VirtualBox will forward port 49153 from the VM to localhost. Soon, inception. You should really just use the VM's IP address mmkay.

Problem #2: Mounting Volumes

The Problem: Docker mounts volumes from the boot2docker VM, not from OS X.

Docker supports volumes: you can mount a directory from the host into your container. Volumes are one way to give your container access to resources in the outside world. For example, we could start an nginx container that serves files from the host using a volume. Let's try it out.

First, let's create a new directory and add an `index.html`:

```
> cd /Users/Chris
> mkdir web
> cd web
> echo 'yay!' > index.html
```

(Make sure to replace `/Users/Chris` with your own path).

Next, we'll start another nginx container, this time mounting our new directory inside the container at nginx's web root:

```
> docker run -d -P -v /Users/Chris/web:/usr/local/nginx/html --name web nginx
485386b95ee49556b2cf669ea785dffff2ef3eb7f94d93982926579414eec278
```

We need the port number for port 80 on our container:

```
> docker port web 80
0.0.0.0:49154
```

Let's try to curl our new page:

```
> curl dockerhost:49154
<html>
<head><title>403 Forbidden</title></head>
<body bgcolor="white">
<center><h1>403 Forbidden</h1></center>
<hr><center>nginx/1.7.1</center>
</body>
</html>
```

Well, that didn't work. The problem, again, is our VM. Docker is trying to mount `/Users/Chris/web` from the host into our container, but the host is boot2docker, not OS X. boot2docker doesn't know anything about files on OS X.

The Solution: Mount OS X's `/Users` directory into the VM.

By mounting `/Users` into our VM, boot2docker gains a `/Users` volume that points to the same directory on OS X. Referencing `/Users/Chris/web` inside boot2docker now points directly to `/Users/Chris/web` on OS X, and we can mount any path starting with `/Users` into our container. Pretty neat.

boot2docker doesn't support the VirtualBox Guest Additions that allow us to make this work. Fortunately, a very smart person has solved this problem for us with a custom build of boot2docker containing the Guest Additions and the configuration to make this all work. We just have to install it.

First, let's remove the web container and shut down our VM:

```
> docker stop web
> docker rm web
> boot2docker down
```

Next, we'll download the custom build:

```
> curl http://static.dockerfiles.io/boot2docker-v1.2.0-virtualbox-guest-additions-v4.3.14
```

Finally, we share the `/Users` directory with our VM and start it up again:

```
> VBoxManage sharedfolder add boot2docker-vm -name home -hostpath /Users
> boot2docker up
```

☞ *Replacing the boot2docker image won't erase any of the data in your VM, so don't worry about losing any of your containers. Good guy boot2docker.*

Let's try this again:

```
> docker run -d -P -v /Users/Chris/web:/usr/local/nginx/html --name web nginx
0d208064a1ac3c475415c247ea90772d5c60985841e809ec372eba14a4beea3a
> docker port web 80
0.0.0.0:49153
> curl dockerhost:49153
yay!
```

Great success! Let's verify that we're using a volume by creating a new file on OS X and seeing if nginx serves it up:

```
> echo 'hooray!' > hooray.html
> curl dockerhost:49153/hooray.html
hooray!
```

Sweet damn. Make sure to stop and remove the container:

```
> docker stop web
> docker rm web
```

☞ *If you update* `index.html` *and curl it, you won't see your changes. This is because nginx ships with* `sendfile` *turned on, which doesn't play well with VirtualBox. The solution is simple—turn off* `sendfile` *in the nginx config file—but outside the scope of this post.*

Problem #3: Getting Inside ainer

The Problem: How do I get in there?

So you've got your shiny new container running. The ports are forwarding and the volumes are ... voluming. Everything's cool, until you realize something's totally uncool. You'd really like to start a shell in there and poke around.

The Solution: ✨Linux Magic✨

Enter nsenter. nsenter is a program that allows you to run commands inside a kernel namespace. Since a container is just a process running inside its own kernel namespace, this is exactly what we need to start a shell inside our container. Let's make it so.

☞ *This part deals with shells running in three different places. Trés confusing. I'll use a different prompt to distinguish each:*

- `>` *for OS X*
- `$` *for the boot2docker VM*
- `%` *for inside a Docker container*

First, let's SSH into the boot2docker VM:

```
> boot2docker ssh
```

Next, install `nsenter`:

```
$ docker run --rm -v /var/lib/boot2docker:/target jpetazzo/nsenter
```

(*How does that install it?* `jpetazzo/nsenter` *is a Docker image configured to build nsenter from source. When we start a container from this image, it builds nsenter and installs it to* `/target`*, which we've set to be a volume pointing to* `/var/lib/boot2docker` *in our VM.*

In other words, we start a prepackaged build environment for nsenter, which compiles and installs it to our VM using a volume. How awesome is that? Seriously, how awesome? Answer me!)

Finally, we need to add `/var/lib/boot2docker` to the `docker` user's `PATH` inside the VM:

```
$ echo 'export PATH=/var/lib/boot2dock          >> ~/.profile
$ source ~/.profile
```

We should now be able to use the installed binary:

```
$ which nsenter
/var/lib/boot2docker/nsenter
```

Let’s start our nginx container again and see how it works (remember, we’re still SSH’d into our VM):

```
$ docker run -d -P --name web nginx
f4c1b9530fefaf2ac4fedac15fd56aa4e26a1a01fe418bbf25b2a4509a32957f
```

Time to get inside that thing. nsenter needs the pid of the running container. Let’s get it:

```
$ PID=$(docker inspect --format '{{ .State.Pid }}' web)
```

The moment of truth:

```
$ sudo nsenter -m -u -n -i -p -t $PID
% hostname
f4c1b9530fef
```

Great success! Let’s confirm we’re inside our container by listing the running processes (we have to install `ps` first):

```
% apt-get update
% apt-get install -y procps
% ps -A
  PID TTY          TIME CMD
    1 ?        00:00:00 nginx
    8 ?        00:00:00 nginx
   29 ?        00:00:00 bash
  237 ?        00:00:00 ps
% exit
```

We can see two nginx processes, our shell, and ps. How cool is that?

Getting the pid and feeding it to `nsenter` is kind of a pain. jpetazzo/nsenter includes docker-enter, a shell script that does it for you:

```
$ sudo docker-enter web
% hostname
f4c1b9530fef
% exit
```

The default command is `sh`, but we can run any command we want by passing it as arguments:

```
$ sudo docker-enter web ps -A
  PID TTY          TIME CMD
    1 ?        00:00:00 nginx
    8 ?        00:00:00 nginx
  245 ?        00:00:00 ps
```

This is totally awesome. It would be more totally awesomer if we could do it directly from OS X. jpetazzo's got us covered there, too (that guy thinks of everything), with a bash script we can install on OS X. Below is the same script, but with a minor change to default to bash, because that's how I roll.

Just stick this bro anywhere in your OS X PATH (and `chmod +x` it, natch) and you're all set:

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

# Check for nsenter. If not found, install it
boot2docker ssh '[ -f /var/lib/boot2docker/nsenter ] || docker run --rm -v /var/li

# Use bash if no command is specified
args=$@
if [[ $# = 1 ]]; then
```

```
  args+=(/bin/bash)
fi

boot2docker ssh -t sudo /var/lib/boot2docker/docker-enter "${args[@]}"
```

Let’s test it out:

```
> docker-enter web
% hostname
f4c1b9530fef
```

Yes. YES. Cue guitar solo.

Don’t forget to stop and remove your container (nag nag nag):

```
> docker stop web
> docker rm web
```

THE END

You now have a Docker environment running on OS X that does all the things you’d expect. You’ve also hopefully learned a little about how Docker works and how to use it. We’ve had some laughs, and we’ve learned a lot, too. I’m glad we’re friends.

If you’re ready to learn more about Docker, check out The Docker Book. I can’t recommend it enough. Throw some money at that guy.

THE FUTURE SOON

Docker might be the new kid on the block, but we’re already thinking about ways to add it to our workflow. Stay tuned for great justice.

Was this post helpful? How are you using Docker? Let me know down there in the comments box. Have a great Sunday. Call your mom.

57 Comments Viget.com 1 L

Recommend 11 Share Sort by Oldest

Join the discussion…

Ben Adlard • a year ago

Great post Chris. We're looking at Docker to improve Vumi's JavaScript sandbox that our dev partners use to build messaging campaigns: http://vumi-jssandbox-toolkit..... Happiness - keep well.

• Reply • Share ›

vigetchrisj Mod Ben Adlard • a year ago

Thanks Ben! Docker would be an excellent fit for that. Good to hear from you.

• Reply • Share ›

Trey Piepmeier • a year ago

Awesome article, Chris. I was just looking at this walkthrough http://build-podcast.com/docke... right before I saw this post. Docker really is the new hotness, I guess.

• Reply • Share ›

vigetchrisj Mod Trey Piepmeier • a year ago

Thanks Trey, glad you liked it! That walkthrough looks pretty good. I can't recommend The Docker Book enough, so if you're interested in learning more, maybe give that a look.

• Reply • Share ›

Juan B. Rodriguez • a year ago

This is a great post, the step by step explanation is really illustrative. I've been using docker for a while on a linux server (www.apertoire.net/dockeritazio... and now I'll look into running it under OSX, for development purposes.

• Reply • Share ›

vigetchrisj Mod Juan B. Rodriguez • a year ago

I've always liked "run the commands while you read" style posts ... glad you appreciate it to!

That's a cool use case for Docker. I'm currently working to move our CI builds into Docker containers, and someday I'd love to get our production apps running in containers.

CHRIS JONES
Senior Developer

Chris' profile

NEXT POST

Use .pluck If You Only Need a Subset of Model Attributes

Extend is...

a CODE & TECHNOLOGY blog—our thirty foot view. primarily written by/for ruby, ios, and front-end developers.

SEARCH

MORE VIGET BLOGS

Advance
STRATEGY & MARKETING

Extend
CODE & TECHNOLOGY

Flourish
NEWS & CULTURE

Inspire
DESIGN & INTERACTION

WE'RE HIRING

Rails Developer Intern Ruby On Rails Develop

ABOUT

WORK

BLOGS

CONTACT

MORE

CONNECT WITH US

GET EMAIL UPDATES

FALLS CHURCH, VA
DURHAM, NC
BOULDER, CO

703.891.0670

info@viget.com

Strategy, Design, & Development. Vigorous since 1999.