MENU

Create your own Heroku on EC2 with Vagrant, Docker, and Dokku

25 SEPTEMBER 2014 on Engineering

Heroku is an awesome component to the developer toolchain and lets you get something up and running easily. For me at least, it's transformed how I think about and architect applications. However, there are also a lot of reasons why you might choose an alternative. Maybe it's speed, control, or maybe you just like tinkering.

For us, the reason was cost--pure and simple. Clearbit has a huge cluster of machines crawling the web, and Heroku would have been prohibitively expensive. As it is, we use EC2 spot instances, scaling up our cluster based on demand at a fraction of the cost of most hosting providers.

Luckily there now exist some great alternatives to Heroku like Docker and Dokku that offer a lot of the benefit without some of the drawbacks.

In this article I'm going to take you step by step through building your own 'mini' Heroku on EC2 using Docker, Dokku and Vagrant. While by no means a replacement to the whole of Heroku (like the instant scaling), it'll get you some of the way there.

Tools

Firstly let's take a look at the tools we'll be using:

Vagrant

Vagrant abstracts building and starting machines behind a simple DSL. Their tagline is: *create and configure lightweight, reproducible, and portable development environments.* In practice though I've found Vagrant great in production, particularly with its AWS extension.

machine. Docker images are similar to Heroku slugs, and you can distribute them amongst your machines without having to consider local environmental issues or conflicts.

We're not going to use Docker directly in this tutorial, but rather through an abstraction: Dokku.

Dokku

Dokku is a bunch of shell scripts written around Docker that emulates a lot of the behavior Heroku gives you, such as deploys over Git and auto-detection of your app environment. Indeed Dokku actually uses the same open source buildpacks that Heroku uses.

We're actually going to be using a variation of Dokku in this tutorial, the Dokku Alt project which is a more complete solution than the original Dokku, with plugins covering most use-cases.

Setup

Firstly, let's install Vagrant. Choose the appropriate package for your local OS you're developing on.

Next install the Vagrant AWS Provider:

```
$ vagrant plugin install vagrant-aws
```

That's all we need to install locally.

Key Pair

You'll need to create an AWS account and a EC2 Key Pair if you haven't already. Note down the name of the Key Pair, as we're going to need

environment they represent, say *production*. You'll need to at least allow incoming traffic from port 22 (SSH) and port 80 (HTTP).

Environmental Variables

You'll need to fetch your AWS credentials and generate a new Access Key. Set the AWS Access Key ID and Secret Access Key as ENV vars, either locally in a shell, in a dotenv file, or in your bash file.

```
export AWS_ACCESS_KEY_ID=123
export AWS_SECRET_ACCESS_KEY=456
```

Vagrantfile

Next we're going to create an EC2 image that will be the basis for all of our app servers. We'll use a *Vagrantfile* to describe our server so we can easily recreate it in the future.

Enter the following in a file called `Vagrantfile` inside your app's root directory. Here's a curl command for your convenience:

```
$ curl
https://gist.githubusercontent.com/maccman/a2f52d067572dcb71dd3/raw/ad04a9e2
acd2191e1b43db5e3b96d9154bb9d97d/Vagrantfile > Vagrantfile
```

And the full Vagrantfile:

```
Vagrant::configure('2') do |config|
  config.vm.define :box1, autostart: false do |box|
    box.vm.box      = 'raring'
```



```
    box.vm.provider :aws do |aws, override|
      aws.access_key_id     = ENV['AWS_ACCESS_KEY_ID']
      aws.secret_access_key = ENV['AWS_SECRET_ACCESS_KEY']
      aws.ami               = 'ami-864d84ee'
      aws.instance_type     = 'm3.xlarge'

      # Override this to your keypair name
      aws.keypair_name      = 'aws'

      # Customize this to the security group you're using
      aws.security_groups   = 'production'

      # To mount EBS volumes
      aws.block_device_mapping = [
        {
          :DeviceName => "/dev/sdb",
          :VirtualName => "ephemeral0"
        },
        {
          :DeviceName => "/dev/sdc",
          :VirtualName => "ephemeral1"
        }
      ]
```

```
    override.ssh.username = 'ubuntu'

    # Customize this to your AWS keypair path
    override.ssh.private_key_path = '~/.ssh/aws.pem'
  end

  # To make sure we use EBS for our tmp files
  box.vm.provision "shell" do |s|
    s.privileged = true
```



```
  end

  # To make sure packages are up to date
  box.vm.provision "shell" do |s|
    s.privileged = true
    s.inline = %{
      export DEBIAN_FRONTEND=noninteractive
      apt-get update
      apt-get --yes --force-yes upgrade
    }
  end

  # Install dokku-alt
  box.vm.provision "shell" do |s|
    s.privileged = true
    s.inline = %{
      export DEBIAN_FRONTEND=noninteractive
      echo deb https://get.docker.io/ubuntu docker main >
/etc/apt/sources.list.d/docker.list
      echo deb https://dokku-alt.github.io/dokku-alt / >
/etc/apt/sources.list.d/dokku-alt.list

      apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys
```

```
36A1D7869245C8950F966E92D8576A8BA88D21E9
        apt-key adv --keyserver pgp.mit.edu --recv-keys EAD883AF
        apt-get update -y
        apt-get install -y dokku-alt
      }
    end
  end
end
```

You'll need to set the `hostname`, `keypair_name`, `security_groups` and

The AMI referenced, `ami-864d84ee`, is just the default Ubuntu AMI. Although the `box_url` looks fake, it's actually correct - it's just a blank Vagrant box to coax Vagrant into working with AWS.

Notice that the box we defined is called `box1`.

Provisioning boxes

Now we're actually going to boot `box1` using Vagrant. Run the following inside the directory you defined the `Vagrantfile`:

```
$ vagrant up box1 --provider=aws
```

If all goes well, Vagrant should have booted up your machine on EC2 and run all the configuration shell scripts to set up our environment. If you are prompted to 'open your web browser to finish configuration', press Ctrl-C since we're doing the installation manually.

Vagrant will also rsync your current directory straight over to the server under `/vagrant` as a convenience.

SSHing into the server

To SSH in your newly provisioned server, run:

```
$ vagrant ssh box1
```

Now you can interact with your server as you normally would. You can see the SSH configuration Vagrant is using by running:

```
$ vagrant ssh-config box1
```


Authorizing your public key

Let's go ahead and authorize our public key with the `dokku` user so we can do deploys. Run:

```
cat ~/.ssh/id_rsa.pub | vagrant ssh box1 -- sudo sshcommand acl-add dokku ${USER}
```

Let's try interacting with dokku to make sure our configuration is set up right. Dokku pipes all SSH input to the `dokku` executable. In other words you can interact with Dokku, setting configuration vars for example, straight over SSH.

The `ec2-box-address` referenced below is the IP address you got when running `ssh-config`.

Run:

```
$ ssh dokku@ec2-box-address

    apps:disable <app>              Disable specific app
```

```
apps:enable <app>                Re-enable specific app
apps:list                        List app
config <app>                     display the config vars for an app
....
```

Deploying

Dokku apps are deployed over Git--the same as Heroku. All you need to do is add a Git remote to the app and push.

First add the remotes. Notice the remote branch name `node-js-app` - Dokku uses this as the application's name.

```
$ git clone https://github.com/heroku/node-js-sample
$ cd node-js-sample
$ git remote add dokku dokku@ec2-box-address:node-js-app
```

And then push:

```
$ git push dokku master
Counting objects: 296, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (254/254), done.
Writing objects: 100% (296/296), 193.59 KiB, done.
Total 296 (delta 25), reused 276 (delta 13)
-----> Building node-js-app ...
       Node.js app detected
-----> Resolving engine versions
```

Since we haven't set up DNS yet, we'll need to configure the application's custom domain to that of our EC2 host. In practice you'll

want to point a domain at your EC2 box, and maybe configure a subdomain wildcard. For now, so we can access our server, run:

```
ssh dokku@ec2-box-address domains:set node-js-app ec2-box-address
```

Now you can visit your site!

```
$ open http://ec2-box-address
```


In particular, here's a few things worth looking into:

Env vars

To set env vars use the `config:set` Dokku command. In this example we're using the `dokku` executable directly on the server, but you can run this command straight over SSH as before.

```
dokku config:set APP RACK_ENV=production
```

Preboot

You can also instruct Dokku to preboot your server, warming up the application code and preventing any downtime during deployments.

```
dokku preboot:enable APP
```

Multiple servers

Vagrant can support provisioning multiple servers at once, simply use a bit of Ruby to define multiple servers in your Vagrant file.

```
Vagrant::configure('2') do |config|
  (1..20).each do |i|
    config.vm.define :"phworker#{i}" do |box|
      # Define box ...
    end
  end
end
```

Vagrant can take a regular expression when booting up or provisioning servers, in this case booting up nine servers at once.

Custom AMIs

At Clearbit we have a custom AMI that all our servers are based off. This means we don't need to build a server from scratch when we first boot it--the box is all ready to go. This allows us to scale with demand quickly.

Vagrant includes a useful little plugin called vagrant-ami which lets you create EC2 custom AMIs:

```
$ vagrant create-ami image1 --name my-ami --desc "My AMI"
```

Then you can replace the AMI ID in your Vagrantfile with your custom one.

Alex MacCaw

Read more posts by this author.

READ THIS NEXT

Node bindings and more

I'm excited to share another raft of Clearbit features that we've shipped over the last week. Node

YOU MIGHT ENJOY

Poor man's realtime

Streaming to browsers has got a lot easier with the advent of APIs like WebSockets and Server-sent