CODEHENGE

WHO BUILT IT…AND WHY? A BLOG ABOUT SOFTWARE, PROGRAMMING, AND TECHNOLOGY.

Search

Automate Your Development Environment With Vagrant

Node.js, Open Source, Programming, Technology, Virtualization Add comments

100% Certified!

Works on My Machine!

I'd like to talk about my favorite software development meme: The 100%, certified, "Works on my machine!" build. We've all seen it, and it's funny every time. Take a second and consider how many times, in fact, you have seen this situation. You write and test your code thoroughly, and it all goes to hell when you commit to your build environment. Or worse, everything's working fine in your staging environment, and something breaks unexpectedly when you push to production. How much frustration can be chalked up such scenarios? How much time have you lost?

Let's explore how this phenomenon occurs. The root cause is a lack of parity between environments throughout your software development process. Your development environment is not the same as your test environment. Your staging environment differs from your production environment. This happens naturally, especially with development machines. Developers work on many projects, simultaneously or over the course of time. They pile up applications, libraries, and different versions of tools, much in the same way a basement workshop becomes cluttered after years of home repair projects. Consider your average developer. Maybe they get a new machine and start working on a Java project. They install Java6, Tomcat6, Spring, etc. Then a few more project come along, and Java7, Tomcat7, Python 2.7.2, Django, and C++ are installed. Of course, our developer wants to keep up on the latest in the field, so they also install Node.js, Clojure, MongoDB, Python3, and C++11 to tinker with. And that's not to mention other, ahem, exercises in mental acuity. Unfortunately, this leads to a fairly cluttered system, with plenty of potential for unanticipated software interactions:

Follow Me

Pages

- About Me
- Contact
- Courses
- Projects
- Talks

Recent Posts:

- Automate Your Development Environment With Vagrant
- Node.js Tutorial: Hello, World!
- Screencast: Node.js and MongoDB Using MongoJS
- Using Socket.io With Express 3.x
- JavaScript Performance Rundown, 2012

Tags

adb Android announcement book book review CHService cloud development example github google guide hacking html image analysis imaging Java javacript javascript listview Mobile mongodb node node.js nodejs nosql Open Source presentation Programming python redis research review science social socket.io socketio Software software-development talk tutorial twitter vagrant virtualization web

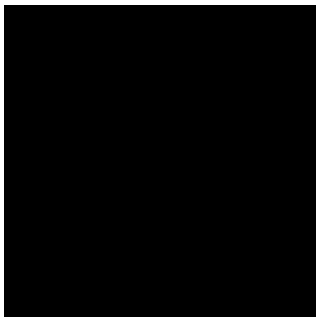
Recommended Blogs

- 53 Bytes
- Coding Horror
- Codus Operandi
- DoubleCloud
- Joel on Software

What could go wrong?

On top of this, developers prefer different development environments. Some prefer OSX, some windows, some a particular flavor of linux. Just try to get an Ubuntu fan to code in Windows. Go ahead, I'll wait. And, of course, different IDEs, tools, debuggers, network configurations… Each installed application, each change in configuration, causes divergence from the clean operating environment you desire for development and testing. The environment that matches your pristine staging and production servers. So what to do? The ideal solution would create unique, reproducible environments for each project. Each developer's environment would be identical, as would the staging and production servers.

Enter Vagrant…



Vagrant leverages virtualization and automated system configuration to make this dream a reality. With VirtualBox and its comprehensive API under the hood, Vagrant creates and manages custom virtual environments to your exact specification. But wait, we've been able to create virtual machines for years. What's new here? The problem is configuration of a brand new virtual macine for each project is a massive chore, reinstalling all of your dev tools each time sounds like torture, and developers will still each want VMs of different operating systems…what are we solving? Vagrant does it differently. By giving you the option to leverage powerful, proven automated configuration technologies such as Chef or Puppet (as well as your own custom shell scripts, if you like), Vagrant takes the time and tedium out of configuring a virtual environment. Need a Ruby on Rails development environment? Use this open source Chef cookbook already designed by someone else. Python and Django? Yo. How about Node.js? Here, I'll give you mine. Don't you love FOSS? But your development tools, such as IDEs, browsers, etc, and the preferred operating systems…this is the best part. By clever use of shared directories and port mapping, Vagrant allows you to keep your code on your host machine, with all of your current tools and apps. You can even run and debug your web application in your own host browser. The VM running in the background, serving up your app, is transparent to you, doing nothing but running your app in a custom, controlled environment.

Let's try a quick demo, to make this workflow completely clear. First, download and install VirtualBox (if you don't have it already) and Vagrant. Vagrant will be creating a custom virtual machine for you, based on a configuration i'll give you. To do this, Vagrant will need a template,

or base VM, to copy and customize. The vagrant dev team is nice enough to provide a number of base linux VMs for our use. To install one, run the following command:

```
$ vagrant box add precise32 http://files.vagrantup.com/precise32.box
```

This will download a base VM of Ubuntu 12.04 32-bit, about 250MB in size, and refer to it by the name “precise32”. Next, you’ll need a project to work on, and a development environment configuration. Here’s a Node.js demo I have prepared for you. Clone this repository, and move into the project directory:

```
$ git clone https://github.com/cacois/vagrant-node-mongo.git
$ cd vagrant-node-mongo
```

By the way, are you interested in **mastering Node.js**? Check out my online course Learn Node.js by Example for detailed screencasts and example applications. Sign up today from this link and receive 50% off!

Take a look inside this directory.

```
vagrant-node-mongo /
   README.md
   Vagrantfile
   /app
   /cookbooks
```

Inside the ‘app’ subdirectory, I have have a small node.js app that returns some text output to an HTTP request. In ‘cookbooks’ are some Chef cookbooks, or collections of recipes that define automated configuration commands to install Node.js, MongoDB, apt-get, and other essential tools. Let’s take a closer look at the file named ‘Vagrantfile’. This is the Vagrant configuration file, defining how to set up your custom project VM. It should look like this:

```
Vagrant::Config.run do |config|
  config.vm.box = "precise32"

  config.vm.forward_port 3000, 3000

  config.vm.share_folder "app", "/home/vagrant/app", "app"

  # allow for symlinks in the app folder
  config.vm.customize ["setextradata", :id, "VBoxInternal2/SharedFolder
  config.vm.customize ["modifyvm", :id, "--memory", 512]

  config.vm.provision :chef_solo do |chef|
    chef.cookbooks_path = "cookbooks"
    chef.add_recipe "apt"
    chef.add_recipe "mongodb"
    chef.add_recipe "build-essential"
    chef.add_recipe "nodejs::install_from_package"
    chef.json = {
      "nodejs" => {
        "version" => "0.8.0"
        # uncomment the following line to force
        # recent versions (> 0.8.4) to be built from
        # the source code
        # , "from_source" => true
      }
    }
```

```
  end
end
```

Let's go through the config file to understand what its doing. First, we are telling Vagrant to use the base box named 'precise32' you downloaded a few moments ago. Then we configure port forwarding between the Vagrant VM and our host over port 3000, and a shared folder in which we will put our application code. Then some VM hardware settings, such as RAM configuration. Finally, we configure the automation system (chef_solo, in this case), and add the recipes from our '/cookbooks' directory defining the applications and services we want to install in the Vagrant VM. Note that I can also pass in specific configurations in JSON format – I'm using this to specify the exact version of Node.js (0.8.0) I want to install.

To power on your custom VM, from inside your application directory issue the following command:

```
$ vagrant up
```

If this is the first time you have issued this command for a new project, Vagrant will actually create the VM from scratch, which may take a few moments. The next time you enter the command, it will just power the VM on, which is substantially faster. When the 'vagrant up' command is issued, Vagrant works in the background to stand up the VM required for that environment (as defined in the Vagrantfile). If you open up the VirtualBox UI you can actually watch the VM appear, but there's no real reason to do so – you won't need the UI to interact with your VM. Instead, once the VM is up, type:

```
$ vagrant ssh
```

This will automatically give you an ssh session into your Vagrant VM, without the need to enter a hostname or credentials. You should see console output similar to the following:

```
mbp15:vagrant-node-mongo constantinecois$ vagrant ssh
Welcome to Ubuntu 12.04 LTS (GNU/Linux 3.2.0-23-generic-pae i686)

 * Documentation:  https://help.ubuntu.com/

121 packages can be updated.
51 updates are security updates.

Welcome to your Vagrant-built virtual machine.
Last login: Fri Sep 14 06:22:31 2012 from 10.0.2.2
vagrant@precise32:~$
```

As you can see, Vagrant has created and ssh session for me into my Vagrant VM. This is the running environment for your application. It's a clean VM, with only a base operating system and the exact applications you configured installed. Issue the following commands to verify the installation of Node.js, NPM, and MongoDB:

```
$ node-v
$ npm -v
$ mongo -version
```

if you do an 'ls', you will also notice a directory '/app', located in your vagrant home directory. Inside is your application code – these files are actually located on your host system, but are visible on your vagrant VM as a VirtualBox shared directory. This is where you will run your code. The shared directory means I can pull up Sublime Text 2 in my host OS (OSX), and edit the files that will be run in the Vagrant VM. I could start the test application now to see it running, but to get an optimal workflow, I want something that will restart my application when changes are made to the code. To get this functionality in Node.js, go ahead and install supervisor on your Vagrant VM from the ssh command line:

```
$ sudo npm install supervisor -g
```

(Note: I could eventually roll this installation into my chef recipes that set up this environment, to save myself a step)

Now launch the application:

```
$ supervisor app/app.js
```

Open up a text editor or IDE in your host system, and open the vagrant-node-mongo/app/app.js file for editing. Now here's a really cool part. Open a browser in your host system, and navigate to http://localhost:3000. Since your Node.js app is running in your Vagrant VM, you will see the following:

The application is running in your Vagrant VM, but you can test it on your host browser, just as if it were running locally! Now go into your text editor and change the app.js code from

```
var http = require('http');

server = http.createServer(function(req, res) {
  res.writeHead(200, {"Content-Type": "text/html"});
  res.write("<html><body><h1>Hello, World!</h1></body></html>");
  res.end();
}).listen(3000);
```

to

```
var http = require('http');

server = http.createServer(function(req, res) {
  res.writeHead(200, {"Content-Type": "text/html"});
  res.write("<html><body><h1>Hello, from Vagrant!</h1></body></html>");
  res.end();
}).listen(3000);
```

Then refresh your browser window. You will see your changes reflected in the running application.

This is the real power of Vagrant – I can code on my local machine, using my tools, applications, IDE, etc. I can run the application on my custom, controller Vagrant VM with a few short commands, and test the app using my favorite local browsers and testing tools. I"m working in a VM, but really don't even have to know about it. Vagrant handles all the management of the VM for me. When I'm done working, I just exit out of the Vagant ssh session:

```
$ exit
```

and tell Vagrant to power down the VM:

```
$ vagrant halt
```

Then I can move on to another project. I can have a Vagrant VM for each project I'm working on. Moreover, this is a *revolution* for team workflows. I can check my Vagrantfile and cookbooks into source control, just as I did with the github application we've been working on, and I can be assured that each member of my team will be developing his code against exactly the same runtime environment. What's more, since we are using Chef to do the VM configuration, I can even use the same Chef cookbooks to configure my centralized test servers, my staging environment, or even production. If we change something in the runtime environment for the application, each developer issues a vagrant destroy command:

```
$ vagrant destroy
```

and then creates a new VM with the updated configurations. This process rarely takes more than 2 minutes, and results in a completely fresh, 100% matching environment for the entire team, across the entire project workflow.

I can be absolutely certain that I'll never again hear "it works on my machine", because I'll know that every machine running my code is exactly the same.

If you liked this article, help me out by sharing a 50% discount to my Node.js course here: Tweet Thanks!

You should follow me on Twitter here: Follow @AaronCois

Posted by cacois at 12:12 pm

Tagged with: automate, automated, automation, configuration, devops, node, node.js, vagrant, virtualbox, virtualization

9 comments • 38 reactions ★ 0

Leave a message...

Best Community Share

Ron • 4 months ago
Launching VM, I'm getting error: "The guest additions on this VM do not match the install version of
VirtualBox" Which version of VirtualBox did you use? I've got 4.2.6 for OSX.
Reply Share ›

Constantine Aaron Cois Mod → Ron • 4 months ago
That error shouldn't cause you any problems. I'm not sure what version of VB the Vagrant base boxes on vagrantup.com were built for, chances are you just have a slightly newer version of VirtualBox.
Reply Share ›

Ron → Constantine Aaron Cois • 4 months ago
Thanks for they reply Constantine. Looking in VirtualBox GUI it looks like specific error is "Failed to load VMMR0.r0 (VERR_SUPLIB_WORLD_WRITABLE)." Wondering which folder it's referring to as vagrant-node-mongo and VirtualBox VMs are writable only by me.
Reply Share ›

Constantine Aaron Cois Mod → Ron • 4 months ago
Take a look at this post:

http://whatan00b.com/virtualbo...

Let me know if this helps.
Reply Share ›

Ron → Constantine Aaron Cois • 4 months ago
That did it! Thanks!
Reply Share ›

Constantine Aaron Cois Mod → Ron • 4 months ago
Excellent!
Reply Share ›

Brian • 4 months ago
This is an excellent tutorial. I hope we will be adopting this very soon it will really help automate our processes! Thanks for this!
Reply Share ›

Mike Borg • 4 months ago
This is exactly the sort of thing I've been looking for. Excellent post.
Reply Share ›

Constantine Aaron Cois Mod → Mike Borg • 4 months ago
Thanks Mike, glad you found it useful!
Reply Share ›

Node.js Tutorial: Hello, World!