Somewhere in New Jersey...

Chris Forster

`http://cforster.com/2013/09/sysadmin-for-poets`

In a previous post on a graduate course I taught last Spring, I mentioned the server I ended up using a way to try to establish some uniformity of access to software packages and tools. In this post, I'll try to add a few details.

Virtual Private Servers

The first thing to understand about "the server" is that I rented a *VPS*, a "virtual(ized) private server," rather than some shared space. If you don't know what that means, let me try to explain (if you do know what that means, you may want to move along to the next section).

The word *server* itself is one of the slipperier bits of our contemporary argot. It can name a number of different links in a relationship between one piece of software and another (to say nothing of the people on either end, or perhaps in between). This slipperiness is only compounded with the advent of *virtual private servers*.

Prior to virtualization, if you wanted to host a webpage, you could either run (or rent) your own dedicated hardware, or you go with a "shared hosting" option. For most folks who run their own blog, shared hosting remains the obvious choice. This blog is hosted on such a shared host. Shared hosting relies on a few facts:**LAMP**: that is Linux (an operating system to manage the hardware resources, to schedule processes… you know, to turn electricity, plastic, and rare earth elements into a computer), Apache (the most widely used web server), MySQL (the most widely used database; which has its own database *server*), and (usually) PHP (a scripting language).

- most folks' hosting needs can be solved by a single "stack" of common software, usually the so-called **LAMP** stack
- most folks' blogs don't get sufficient traffic to require that much hardware, and so many blogs (websites, whatever) can be "served" by a single ("shared") host.
- the OS can easily separate out different users; that way, I can run Wordpress and create a database, and you can run Drupal and create a database, all on the same hardware, without either of us being able to destroy one another's data.

The drawback to shared hosting is that you have very little control over the server. You usually can't, for instance, install new software, beyond packages of PHP scripts. Of course, for web hosting, that's no big deal; you don't really need to install anything beyond whatever CMS you want anyway. Anyone who has had problems with a web host's version of PHP, or similar issues, knows that this isn't quite true.

This compromise with respect to control over the server configuration is a function of cost (and of course, expertise; do you really want to have to worry about building that "software stack"?). To have complete control over a server would mean that you had a dedicated server (in this case, "server" means actual *hardware*), perhaps in your office (in theory it could be your laptop), in an IT closet, or rented in a server farm somewhere. With the advent of software virtualization, however, this changed. With virtualization, a single piece of hardware can run multiple "virtual machines"; the host system simulates another machine, on which you can then run software which itself doesn't really know the difference. (If you run Windows on a Mac with Parallels, this idea may be quite familiar.)

You can do the same thing with a server. That computer that talked to the internet, and served web pages (or whatever), is not just a *virtual* machine. And such a virtualized private server (a VPS, as opposed to a *shared* host) reconfigures the costs involve, and makes it more affordable to give you more control over the software installed, without incurring the full cost of owning/renting/running an actual physical hardware server.

A virtualized server is (*very* significantly) cheaper than running your own hardware, but allows you many of the advantages of actually running your own hardware. It is still, in general, more expensive than shared hosting. To give you some sense of the cost, the lowest tier Linode VPS is $20/month. Digital Ocean's lowest tier is only $5/month.

A Customized Server

Because it is so customizable, I thought a VPS could offer a solution to the challenge of trying to make software uniformly available to students enrolled in the digital humanities seminar I was teaching. With such a system I could give everyone access to Python and the NLTK (and its associated corpora) without having to ask folks to install that software on their own machines. I could install MALLET and R (and relevant R packages) and Stanford's Named Entity Recognizer and an XSLT processor. This was also a relatively flexible solution; if someone wanted to try something else, perhaps something I'd never heard of, it was in many easy to install it on the server.

The *easy* in that last sentence is a function of Linux package management. If you're used to installing from .EXEs (or .DMGs) you download from the web, the world of package management can seem arcane. However, for large pieces of software, package management systems are wonderful and can be (deceptively) simple to use. While getting everything up and running is a bit of a trick (you need to first install an operating system—about which a little more below—and then some basic software to let you connect to the server), installing a piece of software, like the R language, is as simple as typing:

{%highlight bash %} sudo apt-get install r-base {%endhighlight %}

And then, you have R installed, and you're ready to go at the command line. In my experience, Linux package management is often *easier* than trying to handle software dependencies on other OSes (at least once you're familiar with conventions of your package management system).

Easy is also relative; if you're comfortable at the command line, using a package manager feels intellectually more intuitive and comprehensible than dragging a DMG into an "Applications" folder, or double-clicking an EXE. But if you're uncomfortable with the command line, this will likely feel as uncomfortable as navigating your filesystem or anything else.

Server Setup

Before you can install packages, though you need to first install a base, Linux operating system. If you're unfamiliar with Linux, this may not be the kindest or easiest way to get acquainted with it, though the good folks at Linode (hardly unbiased) insist "If you're looking to learn, there is no better environment. Experiment with the different Linux flavors, redeploy from scratch in a matter of minutes.". I have spent too much of my life playing with Linux distros, and so this part of the process felt quite natural. And yet, I *still* managed to make what I now consider a wrong choice in configuring the server. Linux comes in a wide array of flavors or distributions.Technically, "Linux" is not an OS, but an OS "kernel." This distinction, and what we call things, can get contentious. Of the options Linode offers (Ubuntu, Arch, OpenSuse, Gentoo, CentOS, Slackware, Debian, Fedora) I opted for Debian. Debian has a reputation for being a very stable distro; and so it would be a great choice for running a web-server. Of course, I *wasn't* running a web and so stability was not, in fact, my *chief* concern. I probably should have chosen a distribution which prioritized not stability, but the ease and availability of new and up-to-date software packages. Arch Linux, with its "rolling release" schedule (and the operating system I once loved with a passion I've not since been able to match) would have made more sense. It would have been *easier* with Arch to install the most update versions of certain Python packages, etc etc. Oh well. Maybe next time.

Once you're base OS is installed; it's time to install the basic packages you will need to do *anything at all*. But since you now are responsible for a computer somewhere in New JerseyI must say, the responsiveness of the Linode servers shocked me; that computer in NJ was consistently more responsive than my home media server. I regularly ran emacs sessions *on the server* and found them completely responsive. you need to worry about security. You don't want someone hijacking your VPS and using to send spam or whatever else. Linode offers some tips and I consulted this page for some advice as well. It wasn't nearly as scary as I imagined. I installed `fail2ban` (and left the default settings), turned off root log-in from ssh, and that was about it. I have (so far) had no problems.

Connecting and Interacting with the Server

I've sort of glossed over a pretty fundamental fact; with the exception of RStudio server (mentioned below), the only way to interact with the server as I've described it is through SSH; and so the only access you have to the server is command line access. For the most part, that was fine for the goals I had for the class. Command line access allowed people to experiment with Python and NLTK; they could run MALLET, and similar things. You couldn't run, say, Gephi on the server though.

And this limitation proved frustrating for folks working with film and images. For one thing, moving large movie/image files back and forth to the server would have proved unpleasant; moreover, software like ImageJ couldn't be installed on the server and had to be installed locally. (I did manage, though, to do some image manipulation with ImageMagick—here is every page of the *Little Review* in a single image (a larger, ~150M, image is available here):

For folks who were interested in using `R`, RStudio Server worked *wonderfully*; it allowed folks to connect to the server through their web browser and have an RStudio Session that looked something like this:

While I gripe about `R`, RStudio is really excellent. If there is comparable server project that will let folks run python and python packages (including matplotlib) through a browser in a similar way, please, *please* let me know. RStudio provides a great way to provide a standardized R environment with a common set of shared packages (and perhaps even data) available to all.

There are other things one could do with sort of set-up; if you wanted to run an old-school Bulletin Board System or MUD, you could do it (here are some ideas about running a BBS system). You also might look at The BBS Corner's Telnet & Dial-Up BBS Guide or Convolution BBS. As a way to offer *certain* pieces of software to people without requiring them to install them, running this server was very helpful. If you're doing tasks that can be scripted and which can often take a very long time to complete (such as topic modeling, or named entity extraction, or POS tagging)I would add certain types of image manipulation—as I did with the *Little Review* example noted above; though as I say, shuttling the images back and forth can be unpleasant; this unpleasantness is partially allayed if you're scripting your image acquisition with a script, "a little `wget` magic", or similar ought to work. you can set them going, disconnect from the server and then check on them later (using a program like `screen` to make this easy; it's *really* great to be walking to home, knowing that somewhere in New Jersey a computer is dutifully seeking 100 topics across 5000 documents).

I've skipped over some of the other unglamorous, sysadminy things one must do: creating user accounts for each person in the class; creating a shared directory that everyone could read and write to; and other things like that. For someone comfortable at the command line, and interested to learn, all of that stuff is entirely manageable.