SEE TEXT ONLY

The Configuration Management Pipe Dream

by Michael W. Lucas

Configuration management is a dream. A glorious, beautiful dream. Define all of your server characteristics in a central database! Deploy cloud servers as workload They'll all be identical! Behave identically! Everything will just work! Absolute reproducibility, absolute predictability!

That sounds wonderful. Out here in the real world, though, life isn't that simple.

Most of my career has been spent in large enterprises: sometimes as an employee, but most often as a hired-gun consultant. These are companies where the server controlling the door lock runs Novell NetWare, because they really knew how to build hard drives in 1993. These organizations deploy applications to solve specific problems. Most often, they buy applications from vendors with promises that the programs require only minimal customization. Some applications have teams of sysadmins dedicated to their care and feeding, while other applications were benignly neglected until they crashed and died and the whole company noticed.

One thing that all these applications had in common was that the server was set up specifically for that application. The salesperson would tell management that "Our *Cures Everything* ERP software needs a Sun Megajillion with 40 gadzooks of REM, and for your workload you'll want a crystal meth coprocessor." We techs would do our best to translate sales speak into actual hardware and software requirements, and then configure the equipment as per the instructions.

To be fair, the specs weren't always that ambiguous. Some were disturbingly specific, requiring hardware that had gone out of manufacture a couple years ago.

And vendors were always sure to mention that if our company's techs couldn't understand their perfectly clear requirements, for reasons of overwork or simple mental deficiency, their $500/hour consultants could come in and deliver us a completely configured host. Sure, it would be managed via telnet, and lack all hard drive redundancy, but the application itself would work just dandy. Plus, the read-write SNMP string of "public" meant anyone in the company could tune the server, so even we lazy oafs couldn't fail to successfully support it.

So not every application was a special snowflake. Some of them were special hurricanes. Or special dingoes with very special rabies.

"Artisanal systems administration" might sound like a joke, but it sure feels like hard cold reality. The idea of configuration management in an environment like this is laughable. It goes into the same category as "if we were starting over and building from scratch, things would be perfect."

No, we had to maintain unique configurations for every host.

All these organizations had standards, mind you. The security and sysadmin teams maintained runbooks, or policy docs, or standards docs, documenting how every service should be configured, in plain readable English. Plain readable English doesn't necessarily translate to /etc/rc.conf settings. And never mind that Red Hat didn't support pam_mkhomedir, or that the critical application that relies on PHP version 2 and MySQL 3 dot mumble on this repurposed desktop running FreeBSD 4.4 absolutely requires SSH version 1.[1]

Sysadmins follow standards as closely as possible. We tune each box as best we can.

And hope. There's a whole great big scoop of hope, every day.

My last job was, in some way, the pathological end case of this management philosophy. A small, friendly company seemed like a great change. The people were fantastic—I'd known them personally for years, and some of them for decades. Everyone knew their area of responsibility, and yet was willing to help others out. The boss was highly technical and understood exactly how the simplest things could go horribly wrong. If I had to make an emergency visit to the data center at bite-me-o'clock, nobody expected to hear from me for a day or two. The company president considered keeping the fridge stocked with caffeine one of their most vital tasks. Dress code? "You must be dressed."

And absolutely no technical standards.

"I need service X? Fine, I'll stand up a VM for it. This VM server looks like it has capacity. I heard Mint is the new hotness; let's see what all the noise is about. It works, great; let's live with it for eight years!"

"Oh, hey, here's an Itanium box! Let's plug it in and see what we can run on it! Will Asterisk work? Cool, this customer is having trouble, let's route their calls through it and see if it helps."

"The whole company uses Apache, but I hear good things about nginx…"

They'd had a coherent design, some 15 years ago. But years of growth and expansion and purchases had buried that design under layers of infrastructure. And nobody had taken the time to automate anything.

The sign over the entrance to IT Hell reads "Every decision seemed sensible at the time."

And in walks "yours truly," the new Head Honcho of Systems Administration. I got a convenient list of all the servers in the company. All the servers anyone could think of at the moment, that is. For a couple weeks, that list grew daily. For months afterwards, that list grew weekly. One server appeared more than a year after I started.

I found myself walking datacenters with a crash cart, plugging the console into one host after another to figure out what it was. Sadly, I lost a bet by never discovering anything running VMS or Ultrix.

One man can't realistically run all of this. It just can't be done. And everyone had their own preferences for everything from operating systems software versions. The company's stated policy was to indulge those preferences.

Weirdly, company morale was good. Frustrations were high, but morale was good.

I grabbed my packet sniffer and started straightening this tangle.

I started with the obvious, the 15 or so unmaintained name servers on different parts of the network. I installed three new authoritative and three recursive nameservers, properly placed around the network. I updated resolv.conf on every server. Every server I knew about, that is.

I found the other servers when I shut off the unmaintained nameservers.

So: add them to the list of servers. Deal with the customer fallout. Apologize to the engineers whose day I've ruined. The staff understood what I was doing, mind you. They knew the situation needed improving… but that didn't mean they had to like the inevitable pain.

I deployed centralized authentication. One server at a time. Across multiple operating systems. You haven't lived until you've installed nine different LDAP authentication modules on 87 different operating systems, all by hand, one host at a time.

Something clearly had to give.

But centralized authentication opened up new possibilities.

And this is where we get to configuration management.

Configuration management isn't just for whole enterprises. Configuration management can start small, and seemingly unambitiously.

If your organization truly doesn't have configuration management, a one-man army can get it started. One configuration management advocate can implement a solution for their own chunks. Patience and demonstrated success will do the rest of the work for you.

I stood up one more host, running Ansible. (I chose the software that made the most sense to me. Pick whatever's best suited to your environment and your requirements.) I didn't start with deploying whole machines via the cloud. Instead, I centralized the contents of /etc/resolv.conf on all of the infrastructure machines I'd just stood up. This let me work out the countless small problems you always find when trying something new. I then brought sshd_config under configuration management.

Suddenly, I could deploy a change to my new infrastructure servers by editing one file and running one command. SSH isn't one of those services you have to change often, but changing dozens of servers manually can ruin entire days.

Right about then, we found an intruder had broken into some of the servers. The intruder didn't appear to be interested in inflicting any damage on our systems, merely running some IRC bots.

Still, the intruder had to go.

From what I could determine, the intruder had broken in via SSH. Yes, some of the hosts still used password authentication. And some of the passwords were, shall we say... poorly chosen.

And it turned out that this wasn't the first time it had happened.

The staff finally decided that SSH passwords were unacceptable and we needed to allow only key-based authentication. I congratulated them on joining the 1990s, ran an Ansible command, and reported that my new servers were all updated but that I'd need more time to do all the others.

That started the configuration management discussion.

Each managed sshd_config file looked weird. They didn't have all the usual commented-out configuration statements, but instead looked much like this.

```
#Configuration Managed by Ansible
#Manual Changes Will be Overwritten
#You Was Warned
Port 22422
PasswordAuthentication no
Subsystem       sftp    /usr/libexec/sftp-server
```

The configuration file didn't need the commented-out defaults, because nobody would be editing this configuration file on this host to change anything. Any edits needed to happen on the configuration management host.

The discussion quickly turned to "how do we do this everywhere?"

And here's the trickiest part of deploying configuration management.

You never say, "Let me go do it."

You say, "I can do this on every host that authenticates via LDAP, but I need everyone to agree that they will no longer configure the SSH server. Once I bring a host's SSH under config management, it stays there. Everything gets documented in the configuration management system. It's no longer your problem, but you don't play with it anymore."

"But what if things go horribly wrong late sometime and you're not around?"

"Then you fix them and talk to me in the morning. But the next time I deploy a config update, your changes will get wiped out and I ain't listening to you whinge about it."

It wasn't quite as easy as that. Each

sshd_config had to include or exclude statements based on the operating system, to cope with each OS's peculiarities. This little project exposed a few hosts that weren't connected to LDAP—not because I didn't know about them, but because the system couldn't handle the downtime to make that switch.

Weirdly, when you can demonstrate that a change will reduce an engineer's workload, they become really cooperative. Hosts that couldn't withstand any downtime at all to make the LDAP switch suddenly became available.

At 3 a.m. on a Sunday, yes, but still.

We weren't deploying virtual machines automatically. But moving just one service to configuration management changed everything.

"Can you push my authorized_keys to all my hosts?"

"Of course I can. You know, nobody wants to upload a new authorized_keys everywhere. What if I take on all the authorized_keys maintenance? And we all know that letting users edit their own key files can be a security risk. We could move them to /etc/ssh/keys, lock these hosts down a little more." People became more likely to generate client keys more often than once a decade.

Failure of a host's name service led to bringing everything's resolv.conf into configuration management.

An LDAP change? Get the client LDAP set up in config management.

I had expected a slow increase in people asking if we could bring services under configuration management. Turns out I had underestimated the intelligence of my coworkers. One day, configuration management was a pipe dream. The next day, dang near everyone asked me to run those pipes through *everything*.

Retrofitting complete configuration management onto an existing artisan system isn't easy. It's actually pretty terrible.

It's much easier to spin up new virtual machines and deploy to those.

Most of those random hosts needed replacing anyway. It's just that configuring the services was a pain so nobody wanted to touch them.

Best of all, we knew certain things were no longer a problem. My boss's new favorite phrase became "we are *done* with this problem." (True, he also used it for things like the scheduled cola delivery, but that's not the point.)

Morale stayed good. Frustration levels remained the same, but folks were frustrated about different things. The lesson here is that some people aren't happy unless they're frustrated.

And we all lived happily ever after.

Except for the part where we deployed a whole bunch more VMs, because automation made it possible to manage them all easily. But there were pizza and paychecks, so, yeah, still pretty happy.

The weird thing is, I'm not the only sysadmin who's had this story.

Everyone who starts with configuration management, even at a small scale, discovers that it's an incredibly powerful tool. You don't need to start with a new, empty environment. Start with your own pain points. Maybe you have a whole bunch of Apache servers and you need to make sure that everyone has an up-to-date TLS includes file. Maybe you want to be sure that every server you manage has a certain list of packages installed, or that the sudo package is always current.

Our brains all know that automation makes everything easier. But running that first command and seeing your updates flow across your network tattoos that knowledge onto your bones.

Configuration management is not perfect. A configuration management system can deploy mistakes at the speed of Ethernet. (Do an Internet search for "sudo bake me a cake" to find Jan-Piet Mens's amusing failure to manage sudo. It's amusing, because it's not me.)

Using configuration management makes testing far more important. These days, when even cheap laptops can run a fleet of virtual machines, testing is a matter of time and attention—always scarce, yes, but better than requiring time, attention, and thousands of dollars of hardware.

Best of all, managers notice that configuration management helps. Configuration management reduces outages from human inconsistency. It helps expose problems. Eventually, you'll hear your boss tell a salesperson, "That sounds nice, but can we manage it with Ansible/Puppet/Chef/whatever?" You can then concentrate on explaining that "40 gadzooks of REM" is not a real thing, and that any salesperson who says you need it is incompetent.

This issue includes articles on using Vagrant, Puppet, SaltStack, and CFEngine. They all work well with FreeBSD, either as a server or a client. These tools are the very definition of a force multiplier, letting one sysadmin support many more servers than without. Yes, the lucky sysadmin who gets to start out with a blank slate might support hundreds of on-demand cloud servers, but the rest of us can make our most tedious tasks evaporate.

Once you touch the configuration management pipe dream, you'll hang on to it for the rest of your career.

MICHAEL W LUCAS is the author of several books on FreeBSD, including *Absolute FreeBSD* and the *FreeBSD Mastery series.* Learn more at *www.michaelwlucas.com*.

[1] I truly wish I was joking.