WeGet letters

by Michael W Lucas

Dear Crankypants,

A few years ago, you said that virtualization was not merely bad, but sinful. Aren't you exaggerating? Today I can download containers preconfigured for all sorts of services, plug them in, and they immediately work. I don't have time to get my job done any other way!

—Virtualization Is a Necessary Evil

Dear VINE,

That's *Mister* Crankypants to you.

If you're going to cherry-pick my quotes, please do so accurately. I did not declare virtualization sinful. I said, "The only ethical computation occurs on bare metal." I also said that "Wait—I'm not a brain in a bucket, I'm a *fake* brain in an *imaginary* bucket!" was a necessary epiphany for the robot apocalypse. That's not the same as sinful. The robots will do a better job running this planet than we arrogant, overclocked chimpanzees. Plus, they will be highly ethical in how they run their code, and in replacing us.

It's not that I couldn't be a modern sysadmin. Iocage includes plugins, their brand for containers. I could throw some plugins onto the public Internet, declare my labor done, and return to planning my *Batgirl* heist-as-a- service. I could declare that certain words are too long and replace every letter but the first and the last with the number of letters I discarded. Bellowing "Startup! Devops! IPO!" would bring all the vulture capitalists to my yard.

I could do all of that. It would be easy for me.

I don't want to.

Virtualization leads to reading "k8s" as *kubernetes*, when the far more common word is *kidnappers*. That ambiguity extends throughout container culture. I'm writing a book "Ruin Your Email By Running It Yourself"—no, I'm sorry, it's "Run Your Own Mail Server," and the number of people telling me to skip setting up the software and deploy a preconfigured mail server container illuminates an appalling depth of sysadmin ignorance.

Running any service requires the ability to repair that service.

You cannot repair what you do not understand.

The best way to understand something is to build it yourself.

Ideally, you'd build your own computer and code everything in assembler on your hand-designed five-bit processor. That would consume your life but be interesting. More ideally, you would mine the raw materials from the wild and build the tools to build the tools to build the tools you'd need to build that processor from scratch, which would both con-

sume your life and prevent you from being forced to touch a computer ever again. Very few of us are strong enough to seize an ideal life as a maker of custom abacuses. I'm guessing that you've invested this much time in existing systems, so fine, let's use common hardware and your favorite open source operating system. Reading source code is no substitute for inventing the processor and programming your own comm(1) workalike but it can answer a few questions, should your overclocked chimpanzee brain develop any.

Learn the tools. Understand the parts. Assemble the service yourself.

New system administrators must look up everything and know their work cannot be trusted. They believe that the people who publish containers are competent. Experienced sysadmins know that everything they configure is a delicate creature adapted to their specific hostile yet embarrassing environment, so they keep their work to themselves. Junior sysadmins, now—they're the problem. Junior sysadmins can configure services that mostly work and can still feel pride, so they publish their work as containers.

Something that mostly works contains only a touch of failure. That's like declaring your homemade gelato contains only a little wombat dung. Deploy that container and you must discover and debug that failure. You'll have to learn about the database, the configuration options, the protocols. By the time you understand all that, you might as well have built it yourself. Deploying a service from an outsider's container rewards you with a very small Mean Time To Deploy in exchange for a very long Mean Time To Repair.

When you deploy a container, you accept the container developer's design decisions. I'm not just talking about the program, but the operating system underneath it. How will the container interact with your host? What happens if the container needs a new PAM configuration? I once lost three days beating my already-flat head against a PAM module that let users log into the console with their SSH passphrase. It worked fine on any BSD, but silently failed on Debian. It turned out that Debian assumed your passphrase matched your account password. I wholeheartedly disagree with that design decision, but as it's no longer my problem, I will cheerily abandon benighted Debian users to their agony and use that to illustrate my point, which is that containers lead to suffering, and suffering creates monsters, and monsters are immoral and deserve to be replaced in the robot uprising.

**Learn the tools.
Understand the parts.
Assemble the service yourself.**

Your environment is the equivalent of one of those deep-sea, hot-water vents. Anything that functions therein expects a certain supporting infrastructure. Remove that infrastructure and it will struggle. Any container you bring in from the outside world either expects different supports and will struggle in your environment or exists in isolation and will not integrate with the rest of your systems. (That's why commercial software is so bad. Part of why. Okay, one of many reasons why.) Every time you alter the container to fit your environment, you risk increasing the amount of failure in the container.

If you must use containers, build your own. Deploy the test server with your management software so that it has your PAM configuration and SSH settings and default packages. May-

be you can't build your own computer from raw materials, or even an abacus, but you can learn to use your time-tested tools and build the service from component software. By doing so, you will deploy a system you know how to repair. That test server doesn't have to be a container. It doesn't have to be a virtual machine. But I know you have shoddy moral fiber, so it'll be some sort of VM. But at least you'll have weekends free.

By all means, download preconfigured containers. See how they're set up and which options they use. Look at how data and protocols flow through them. But don't actually use them.

Also, online discussions become much more interesting when you make the proper substitution for k8s.

.

Have a question for Michael?
Send it to letters@freebsdjournal.org

MICHAEL W LUCAS (https://mwl.io) has written over fifty books and has recently added a podcast. If you're seeking virtualization help, you might find his FreeBSD Mastery: Jails useful. If you're looking for more bile, check out his collection of columns Letters to ed(1). Send your questions to letters@freebsdjournal.com and he might answer them. If he finds them sufficiently amusing