Containers vs. virtual machines: How to tell which is the right choice for your enterprise

Autor:* *Steven J. Vaughan-Nichols *publicado en* CIO

Name a tech company, any tech company, and they're investing in containers. Google, of course. IBM, yes. Microsoft, check. But, just because containers are extremely popular, doesn't mean virtual machines are out of date. They're not.

Yes, containers can enable your company to pack a lot more applications into a single physical server than a virtual machine (VM) can. Container technologies, such as Docker, beat VMs at this part of the cloud or data-center game.

VMs take up a lot of system resources. Each VM runs not just a full copy of an operating system, but a virtual copy of all the hardware that the operating system needs to run. This quickly adds up to a lot of RAM and CPU cycles. In contrast, all that a container requires is enough of an operating system, supporting programs and libraries, and system resources to run a specific program.

What this means in practice is you can put two to three times as many as applications on a single server with containers than you can with a VM.

In addition, with containers you can create a portable, consistent operating environment for development, testing, and deployment. That's a winning trifecta.

If that's all there was to containers vs. virtual machines then I'd be writing an obituary for VMs. But, there's a lot more to it than just how many apps you can put in a box.

Container problem #1: Security

The top problem, which often gets overlooked in today's excitement about containers, is security. As Daniel Walsh, a security engineer at Red Hat who works mainly on Docker and containers puts it: Containers do not contain. Take Docker, for example, which uses libcontainers as its container technology. Libcontainers accesses five namespaces -- Process, Network, Mount, Hostname, and Shared Memory -- to work with Linux. That's great as far as it goes, but there's a lot of important Linux kernel subsystems outside the container.

These include all devices, SELinux, Cgroups and all file systems under /sys. This means if a user or application has superuser privileges within the container, the underlying operating system could, in theory, be cracked.

That's a **bad** thing.

Now, there are many ways to secure Docker and other container technologies. For example, you can mount a /sys file system as read-only, force container processes to write only to container-specific file systems, and set up the network namespace so it only connects with a specified private intranet and so on. But, none of this is built in by default. It takes sweat to secure containers.

The basic rule is that you'll need to treat containers the same way you would any server application. That is, as Walsh spells out:

- Drop privileges as quickly as possible
- Run your services as non-root whenever possible
- Treat root within a container as if it is root outside of the container

Another security issue is that many people are releasing containerized applications. Now, some of those are worse than others. If, for example, you or your staff are inclined to be, shall we say, a little bit lazy, and install the first container that comes to hand, you may have brought a Trojan Horse into your server. You need to make your people understand they cannot simply download apps from the Internet like they do games for their smartphone.

Mind you they shouldn't be downloading games willy-nilly either, but that's a different kind of security problem!

Other container concerns

OK, so if we can lick the security problem, containers will rule all, right? Well, no. You need to consider other container aspects.

Rob Hirschfeld, CEO of RackN and OpenStack Foundation board member, observed that: "Packaging is still tricky: Creating a locked box helps solve part of [the] downstream problem (you know what you have) but not the upstream problem (you don't know what you depend on)."

Breaking deployments into more functional discrete parts is smart, but that means we have MORE PARTS to manage. There's an inflection point between separation of concerns and sprawl. -- *Rob Hirschfeld*

The hit list

To this, I would add that while this is a security problem, it's also a quality assurance problem. Sure, X container can run the NGINX web server, but is it the version you want? Does it include the TCP Load Balancing update? It's easy to deploy an app in a container, but if you're installing the wrong one, you've still ended up wasting time.

Hirschfeld also pointed that out container sprawl can be a real problem. By this he means you should be aware that "Breaking deployments into more functional discrete parts is smart, but that means we have MORE PARTS to manage. There's an inflection point between separation of concerns and sprawl."

Remember, the whole point of a container is to run a single application. The more functionality you stick into a container, the more likely it is you should been using a virtual machine in the first place.

True, some container technologies, such as Linux Containers (LXC), can be used in lieu of a VM. For example, you could use LXC to run Red Hat Enterprise Linux (RHEL) 6 specific applications on a RHEL 7 instance. Generally speaking though you want to use containers to run a single application and VMs to run multiple applications.

Deciding between containers and VMs

So how do you go about deciding between VMs and containers anyway? Scott S. Lowe, a VMware engineering architect, suggests that you look at the "scope" of your work. In other words if you want run multiple copies of a single app, say MySQL, you use a container. If you want the flexibility of running multiple applications you use a virtual machine.

In addition, containers tend to lock you into a particular operating system version. That can be a good thing: You don't have to worry about dependencies once you have the application running properly in a container. But it also limits you. With VMs, no matter what hypervisor you're using -- KVM, Hyper-V, vSphere, Xen, whatever -- you can pretty much run any operating system. Do you need to run an obscure app that only runs on QNX? That's easy with a VM; it's not so simple with the current generation of containers.

So let me spell it out for you.

Do you need to run the maximum amount of particular applications on a minimum of servers? If that's you, then you want to use containers -- keeping in mind that you're going to need to have a close eye on your systems running containers until container security is locked down.

If you need to run multiple applications on servers and/or have a wide variety of operating systems you'll want to use VMs. And if security is close to job number one for your company, then you're also going to want to stay with VMs for now.

In the real world, I expect most of us are going to be running both containers and VMs on our clouds and data-centers. The economy of containers at scale makes too much financial sense for anyone to ignore. At the same time, VMs still have their virtues.

As container technology matures, what I really expect to happen, as Thorsten von Eicken, CTO of enterprise cloud management company RightScale, put it is that VM and containers will come together to form a cloud portability nirvana. We're not there yet, but we will get there.