Interview with James Bottomley

RIK FARROW

James Bottomley is a Distinguished Engineer at IBM Research, where he works on cloud and container technology. He is also a Linux kernel maintainer of the SCSI subsystem. He has been a Director on the Board of the Linux Foundation and Chair of its Technical Advisory Board. He went to Cambridge University for both his undergraduate and doctoral degrees after which he joined AT&T Bell Labs to work on distributed lock manager technology for clustering. In 2000 he helped found SteelEye Technology, a high availability company for Linux and Windows, becoming Vice President and CTO. He joined Novell in 2008 as a Distinguished Engineer at SUSE Labs, Parallels (later Odin) in 2011 as CTO of server virtualization, and IBM Research in 2016. james.bottomley@hansenpartnership.com

Rik is the editor of *;login:*. rik@usenix.org

I first met James Bottomley during a Linux File System and Storage workshop that took place before FAST in 2007. James' focus has been on the SCSI subsystem of Linux. But, as the CTO of Parallels, James has also worked on containers. James and Pavel Emelyanor wrote an article comparing containerization to virtualization for *;login:* back in 2014 [1].

While attending LISA '16, I heard many conversations from people in the hallway that suggested that they understood neither the purpose of containers nor how they were implemented. And, it turns out, I didn't understand how containers work under Linux either.

Rik Farrow: Looks like you may not be at Parallels anymore.

James Bottomley: That's right...I'm at IBM Research now.

RF: My problem is that lots of people don't consider container tech important.

JB: Heh, well, there's a strong political reason for that: the main contenders vying to be the enterprise container power have no expertise in the core technology of containers (OS virtualization), so they're anxious to concentrate on stuff they can control. Plus if you look at what industry is after with container technology, development process simplification and agility, although these are enabled by OS virtualization, they're nowhere directly connected to virtualization.

RF: By "main contenders," you mean Docker, Red Hat, Core, and some others I am not thinking of?

JB: Yes: other orchestration companies like Mesos, Joyent, and now even VMware.

RF: You include VMware in the list of companies offering orchestration. Could you clear that up for me?

JB: Yes, VMware's province is still very much hypervisors and thus hardware virtualization not OS virtualization. Admittedly, VMware does have a Linux kernel team, which gives them the capacity to get into the OS virtualization infrastructure in Linux very quickly unlike most of the other orchestration owners, but there's little sign (from kernel commit logs) that they're doing this.

RF: I think that industry wants what you suggest, simpler development and more agility, but they also appreciate having containers that are much lighter weight than VMs.

JB: Remember, I worked for Parallels, which was a container company before it was fashionable. In 2004, Parallels tried to sell containers to the enterprise in place of VMs on the grounds that they were faster and more lightweight. Parallels failed primarily because that's not what the enterprise wanted.

Enterprise CIOs have a problem they try to conceal with excess hardware capacity; something that uses capacity more efficiently is really an unwelcome technology.

The first company to have a genuine need for lightweight virtualization technology was Google in around 2006-2007 because they realized that to run a service at cloud scale you require this type of transactional efficiency—that's when they adopted containers wholesale. Very few traditional enterprises are building out cloud-scale datacenters still.

RF: I've heard that you can run 10 times as many containers as VMs on the same hardware. And they can spin up containers much faster, too.

JB: Yes, that's because there's a single kernel doing all the resource management. Containers are essentially small groups of UNIX processes, so if you want to run 100 Apache servers, it's far cheaper in resources to run 100 Apache processes each in a container than to run 100 VMs with a full OS complement.

Full operating systems are very complex and resource-intensive beasts. The person who just wants to run *X* applications really doesn't care what the OS is doing and really doesn't want to manage it, which is the Achilles' heel of VMs. The world wants to move away from infrastructure, but a VM is anchored there.

RF: I attended a workshop (HotCloud '14), where they broke up into groups discussing different topics. I attended the Container group, and one thing some Google person said stuck in my mind: we run associated containers within a VM, and we use VMs for security isolation. I thought about that a lot.

JB: Google has a particular problem: being the first adopters, they bent the technology to serve themselves. Google actually hired about everyone they could who was working on Linux cgroups in 2006. The Google datacenters grew to be container-centric but supported Google written workloads. The Google cloud allowed you to bring your data but not your code in those days. If you write all the code, you can take a lot of shortcuts with security (which Google did).

Then when they wanted to offer a-bring-your-own-code service, Google App Engine, they had to turn to some external technology to add security. This problem is unique to Google. But every former or current hypervisor company is trying to also smear container security because they fear it's the only way they'll stay in the game, so you hear this type of statement from a lot of sources.

The reality is, of course, that containers were being sold as hypervisor replacements to the hosting industry by Parallels from about 2001 on. With no need of any VM to provide security. The technology itself can be made secure enough on bare metal.

The key phrase is "can be made." The problem with container technology is that it's not all or nothing like VM technology. You can't really emulate just some virtual hardware, so if you don't turn on the OS virtualizations securely, you don't get security. Most of the modern application packaging container technology, like Docker, doesn't turn all the security features on.

RF: In the article you and Pavel wrote [1], you explained that containers are based on cgroups and namespaces. Cgroups (control-groups) provide limits to resource usage, and namespaces limit access to, well, namespaces, such as files, directories, devices, and networks. Is that a good description of how containers work?

JB: Sort of. The problem is that the OS itself has no concept at all of a "container": all the OS knows is that there are a group of processes for which certain OS virtualization features have been set up. So the way "containers" work is potentially hugely variable. For instance, the Kubernetes concept of a "pod" means a set of "containers" that share certain namespaces, like network or IPC (meaning they see each other's network interface, and you can set up IPC message passing between them).

All container systems without exception use the core Linux APIs of namespaces and cgroups, but they can use them in very different ways (so LXC is very different from, say, Docker in how it sets up what it thinks of as a container).

RF: There must also have been some API support added, so a root-EUID process could start up containers.

JB: Actually, the largest amount of work in Linux is going on in the realm of what are called unprivileged containers. This means OS virtualization that can be controlled by non-root users.

What you say above is currently true—most orchestration systems do run as root, but that causes security problems, so they'd actually also be interested in running unprivileged.

RF: I'm guessing that this is involved in orchestration schemes, but there must be more to orchestration than just firing up containers. You need a way to keep track of them, as well as methods for both connecting them as well as constraining them through the orchestration system.

JB: Right. Usually the way an orchestration system keeps track of containers to think of each container as being a collection of processes. Usually the container has some unique ID, and each process within the container carries it as either a mark or a mapping. Most often the way you can see this from outside is that each container is a separate PID namespace. So Docker uses UUIDs, and it keeps a runtime map of UUID<->PID namespace (which changes every time you start and stop a container) so that it can uniquely identify every process in a container by interrogating the PID namespace.

Now that I've told you the above, I have to confess that when I set up my architecture emulation containers, I don't actually use a PID namespace, so the above isn't universal (but realistically nothing in containers is).

RF: That really helped me understand containers: that the UUIDs that Docker creates is just the Docker tool's own way of identifying a group of processes. I found myself wondering whether there was a "create container" system call. Instead I discovered that most of the work is done by clone() by setting certain flags when creating a new process.

JB: Yes, there are essentially two namespace creation system calls, clone() and unshare(), and one namespace entry system call, setns(). Cgroups don't have any system calls at all; it's currently all done by manipulating files in the cgroup file systems, which are usually mounted under /sys/fs/cgroup.

Reference

[1] James Bottomley and Pavel Emelyanov, "Containers," *;login:* , vol. 39, no. 5 (October 2014): https://www.usenix.org/publications/login/october-2014-vol-39-no-5/containers.