What is All the Container Hype?

Mini White Paper

Dr. James Bottomley, Chief Technology Officer, Server Virtualization, Parallels, and Board Director, Linux Foundation

April 2014

Abstract
Of all the different types of virtualization technology, containers have long been regarded either as a cheap way of packing a hosting environment, or simply as a curiosity. Now however, with the advent of the cloud revolution and the focus on elasticity and density within a lean data center, containers are coming into their own as the densest and most elastic virtualization technology for supporting cloud environments.

In this paper we will cover the principles of containerization, its similarities and differences from traditional virtualization technologies, and why it achieves much higher densities and far greater elasticity. We will also discuss the advantages and disadvantages of containers versus more traditional hypervisor-based virtualization, in particular how they might be applied to network function virtualization, how a company might reliably deploy containers through management platforms like OpenStack, and where containers are going in the future.
Introduction

Virtualization can be defined as the art of running one computer operating system on top of another, and has a long and venerable history. Long before hypervisors and UNIX, virtualization was used by mainframes to partition between different operating systems. However, virtualization really took off in the UNIX and Linux market around 2001, when VMware produced a server product for hypervisor-based virtualization which attracted huge attention in the enterprise market. At almost exactly the same time, a company called SWSoft (now Parallels) released Virtuozzo, a container virtualization product that gained great prominence in the hosting market. This division for containers remained for almost 12 years; while hypervisor-based virtualization did take some part of the hosting market, containers never penetrated into the enterprise, at least until 2013 with the rise of Docker and the beginning of enterprise interest in container technologies.

The Difference Between Containers and Hypervisors

At a base level, as shown in Figure 1, a hypervisor works by having the host operating system emulate machine hardware and then bringing up other virtual machines (VMs) as guest operating systems on top of that hardware. This means that the communication between guest and host operating systems must follow a hardware paradigm (anything that can be done in hardware can be done by the host to the guest). On the other hand, container virtualization (shown in figure 2), is virtualization at the operating system level, instead of the hardware level.

This means each of the guest operating systems shares the same kernel, and sometimes parts of the operating system, with the host. This enhanced sharing gives containers a great advantage in that they are leaner and smaller than hypervisor guests, simply because they’re sharing much more of the pieces with the host. It also gives them the huge advantage that the guest kernel is much more efficient about sharing resources between containers, because it sees the containers as simply resources to be managed. To give a simple example, if

---

**Figure 1: Hypervisor Diagram**

**Figure 2: Container Diagram**
Container 1 and Container 2 open the same file, the host kernel opens the file and puts pages from it into the kernel page cache. These pages are then handed out to Container 1 and Container 2 as they are needed, and if both want to read the same position, they both get the same page.

In the case of VM1 and VM2 doing the same thing, the host opens the file (creating pages in the host page cache) but then each of the kernels in VM1 and VM2 does the same thing, meaning if VM1 and VM2 read the same file, there are now three separate pages (one in the page caches of the host, VM1 and VM2 kernels) simply because they cannot share the page in the same way a container can. This advanced sharing of containers means that the density (number of containers of Virtual Machines you can run on the system) is up to three times higher in the container case as with the Hypervisor case.

This density advantage is one of the principle reasons containers found a home in the Virtual Private Servers (VPS) hosting market; simply that if you can serve three times as many VPSs per physical server, your fixed costs per VPS are reduced by 66% allowing for greater profitability.

Of course, it’s not all perfect in the container world. One of the things you have to give up when you share a kernel is the ability to run different kernels in the system. This means that a containerized system cannot bring up Windows and Linux on the same physical hardware, which is an easy trick for Hypervisors. However, at least in Linux, this is mitigated by the ABI and library guarantees, meaning it is possible to bring up different Linux distributions in containers on the same system sharing the same kernel but the cost for doing this is less sharing between the containers. The true benefit of containers can be realised when the environment is homogeneous.

**Containers History**

In 2005, Google was struggling with the problems of delivering a web-based services at scale; specifically how to scale consumption of resources in their data center elastically to give all users of the service a great experience however many (or few) of them were using the service at any given instant, while simultaneously using any leftover resources for background jobs.

Google experimented with virtualization for this task but quickly found it unsuitable; the main problems being that the performance penalty for virtualizing was too high (equivalent to the density being too low) and the response wasn’t elastic enough to support a web-based service at scale. This latter point is interesting because web services cannot predict at any given time whether they’ll be servicing 10, 100 or one million requests. However, the users sitting on the other end expect instant response (for some value of instant equating to roughly tolerating around a second of lag between pushing the button and seeing the results) regardless of how many others are simultaneously using the service. With the average spin-up time of a hypervisor being measured in tens of seconds, it’s clear that the user expectations of results within a second simply cannot be met.

At the same time, a group of engineers were experimenting in Linux with a concept based on cgroups called process containers. Within a matter of months, Google hired this group and set about containerizing its data centers to solve the elasticity at scale problem.

In January 2008, some of the cgroup technology used by Google made its way into the Linux Kernel and the LXC (LinuX Containers) project was born. At around this
same time, SWSoft became Parallels, and Virtuozzo was released into Open Source as the OpenVZ project. In 2011, Google and Parallels agreed to collaborate on their respective container technologies, resulting in the 2013 release of Linux Kernel 3.8, in which all the Linux container technologies (at least as far as the kernel goes) were unified, thus avoiding a repeat of the damaging Xen/KVM split within the kernel.

**Containers and the Enterprise, Why Now?**

For hosting providers, the primary benefit is density. However, the enterprise doesn't really need density in most data centers; indeed, virtualization as a technology appealed to the enterprise precisely because most of their servers were under-utilized and it was a good way of finding alternative uses for spare capacity. So technology that helped increase density and thus reduce overall utilization had no real value to them. Obviously then, because containers had no apparent value, it was a technology ignored by enterprise IT and thus, since the enterprise is 85% of the virtualization market, by the world outside of the hosting providers.

Fast forwarding to 2010, the emergence of the Cloud as a delivery channel for services to end users had enterprises trying to take advantage of this channel and running into delivery at scale problems identical to those that Google and Parallels had tackled half a decade earlier. With hindsight and a look at history, it is obvious that the solution to this problem for the enterprise is the same one that worked for Google. The problem with seeing this from an enterprise point of view is that when all your thinking about virtualization is skewed by hypervisors and a lot of your data center budget going towards buying and managing hypervisors, adopting a new virtualization technology doesn't seem like such an obvious or even a good idea.

It wasn't until Docker came along in 2013 and showed how easy it could be to package a containerized application on Linux and deploy it directly at scale into dotCloud (Docker's Platform as a Service) that the enterprise started paying attention. Simultaneously, OpenStack was promising to unify Cloud Management to a single platform, and with the two converging trends, the enterprise could see a path to managing their hypervisor-based data centers with a single tool that could simultaneously deploy containerized applications at scale. Now enterprises started paying attention and the container hype began in earnest.

**So are Containers the Answer?**

Depending on the question, possibly. Containers can help package and deploy web applications at scale, and using container features in web applications and platforms can solve hard problems like multi-tenancy, so in that sense they've proven themselves to be a vital technology for addressing today's cloud delivery problems.

But they're not a panacea, particularly for enterprises that have already invested in hypervisor-centric technologies like SR-IOV, VT-D and Network Function Virtualization. In each case, the technology is designed to connect a virtual machine guest directly to a hardware or fabric by means of a special driver sitting inside the guest kernel. Since there's no separate guest kernel with containers, there's nothing to insert a driver into, and therefore no apparent way to make use of the technology.

However, analysis only shows the technology doesn't work if one thinks in the hypervisor paradigm. Instead, let's consider how one of these technologies might be made accessible to containers.
Containers and Network Function Virtualization (NFV)

Traditionally, in NFV, one takes a network function exported via the fabric and proxies it into a guest kernel via means of a fabric connected virtual driver. This allows the guest to process network traffic with the same efficiency as the fabric data plane but in a virtual environment whose operation is under software control. The actual proxied fabric function appears in the guest by means of one (or a set of) fairly standard network interfaces.

If instead of asking the question about driver insertion, we asked the question “can a container be attached to a proxied fabric function by means of a network interface?” the answer is simple and “yes.” Part of a container is a technology called a network namespace whose job is to make a network interface from the shared kernel appear exclusively in one container. Therefore, as long as the special driver can be inserted into the shared kernel, the network interface can still be proxied into a separate container via a network namespace.

This would give the container an identical ability to process network traffic to the virtual environment with the special driver, although now the network function code can operate with even greater density and efficiency thanks to the lighter weight container infrastructure.

One can even take this concept a step further and note that a network namespace is also independent of the rest of the controls of a container, so one could take a set of applications running in a physical system and attach each of them individually to a proxied fabric function by means of running each application in its own network namespace. In this approach, called application containerization, only enough of the application itself is placed in a partial container (depending on what limits you need to impose) and can operate with densities approaching 100x what is possible with traditional virtualization.

Conclusion

Containers are a proven technology for density, elasticity and scale. However, in order to reap the benefits of containers in technologies designed for consumption via hypervisors, as the Network Function Virtualization case demonstrates, you may have to think about the problem in a different way. However, once you think about container solutions, often additional ways of taking advantage of the technologies become apparent which may have applicability beyond the virtual machine paradigm.

So, in conclusion, look at your enterprise today and everywhere you see a V for virtualization ask yourself “how would this work if I thought instead of a C for containerization?” The results may surprise you.

For more information on containers & hypervisors:
www.parallels.com/products/pcs

For an early look at the OpenStack Nova driver for Parallels Cloud Server:
https://github.com/parallels/pcs-nova-driver