Containers and Cloud: From LXC to Docker to Kubernetes

WELCOME TO CLOUD TIDBITS! In each issue, I'll look at a different "tidbit" of technology that I consider unique or eye-catching, and of particular interest to the *IEEE Cloud Computing* readers.

Today's tidbit focuses on container technology and how it's emerging as an important part of the cloud computing infrastructure.

Cloud Computing's Multiple OS Capability

Many formal definitions of cloud computing exist. The National Institute of Standards and Technology's internationally accepted definition calls for "resource pooling," where the "provider's computing resources are pooled to serve multiple consumers using a multitenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand."[1] It also calls for "rapid elasticity," where "capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand."

Most agree that the definition implies some kind of technology that provides an isolation and multitenancy layer, and where computing resources are split up and dynamically shared using an operating technique that implements the specified multitenant model. Two technologies are commonly used here: the *hypervisor* and the *container*. You might be familiar with how a hypervisor provides for virtual machines (VMs). You might be less familiar with containers, the most common of which rely on Linux kernel containment features, more commonly known as LXC (https://linuxcontainers.org). Both technologies support isolation and multitenancy.

Not all agree that a hypervisor or container is required to call a given system a cloud; several specialized service providers offer what is generally called a *bare metal cloud*, where they apply the referenced elasticity and automation to the rapid provisioning and assignment of physical servers, eliminating the overhead of a hypervisor or container altogether. Although interesting for the most demanding applications, the somewhat oxymoron term "bare metal cloud" is something perhaps Tidbits will look at in more detail in a later column.

Thus, we're left with the working definition that cloud computing, at its core, has hypervisors or containers as a fundamental technology.

Cloud Systems with Hypervisors and Containers

Most commercial cloud computing systems—both services and cloud operating system software products—use hypervisors. Enterprise VMware installations, which can rightly be called early private clouds, use the ESXi Hypervisor (www.vmware.com/products/esxi-and-esx/overview). Some public clouds (Terremark, Savvis, and Bluelock, for example) use ESXi as well. Both Rackspace and Amazon Web Services (AWS) use the XEN Hypervisor (www.xenproject.org/developers/teams/hypervisor.html), which gained tremendous popularity because of its early open source inclusion with Linux. Because Linux has now shifted to support KVM (www.linux-kvm.org), another open source

DAVID BERNSTEIN

Cloud Strategy Partners,
david@cloudstrategypartners.com

2325-6095/14/$31.00 © 2014 IEEE

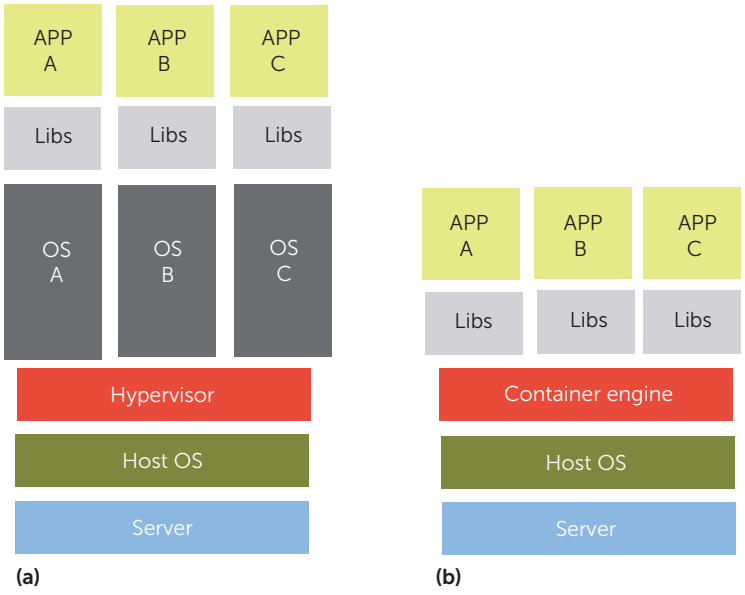

Figure 1. Comparison of (a) hypervisor and (b) container-based deployments. A hypervisor-based deployment is ideal when applications on the same cloud require different operating systems or different OS versions; in container-based systems, applications share an operating system, so these deployments can be significantly smaller in size.

alternative, KVM has found its way into more recently constructed clouds (such as AT&T, HP, Comcast, and Orange). KVM is also a favorite hypervisor of the OpenStack project and is used in most OpenStack distributions (such as RedHat, Cloudscaling, Piston, and Nebula). Of course, Microsoft uses its Hyper-V hypervisor underneath both Microsoft Azure and Microsoft Private Cloud (www.microsoft.com/en-us/server-cloud/solutions/virtualization.aspx).

However, not all well-known public clouds use hypervisors. For example, Google, IBM/Softlayer, and Joyent are all examples of extremely successful public cloud platforms using containers, not VMs.

Some trace inspiration for containers back to the Unix chroot command, which was introduced as part of Unix version 7 in 1979. In 1998, an extended version of chroot was implemented in FreeBSD and called *jail*. In 2004, the capability was improved and released with Solaris 10 as *zones*. By Solaris 11, a full-blown capability based on zones was completed and called *containers*. By that time, other proprietary Unix vendors offered similar capabilities—for example, HP-UX containers and IBM AIX workload partitions.

As Linux emerged as the dominant open platform, replacing these earlier variations, the technology found its way into the standard distribution in the form of LXC.

Figure 1 compares application deployment using a hypervisor and a container. As the figure shows, the hypervisor-based deployment is ideal when applications on the same cloud require different operating systems or OS versions (for example, RHEL Linux, Debian Linux, Ubuntu Linux, Windows 2000, Windows 2008, Windows 2012). The abstraction must be at the VM level to provide this capability of running different OS versions.

With containers, applications share an OS (and, where appropriate, binaries and libraries), and as a result these deployments will be significantly smaller in size than hypervisor deployments, making it possible to store hundreds of containers on a physical host (versus a strictly limited number of VMs). Because containers use the host OS, restarting a container doesn't mean restarting or rebooting the OS.

Those familiar with Linux implementations know that there's a great degree of binary application portability among Linux variants, with libraries occasionally required to complete the portability. Therefore, it's practical to have one container package that will run on almost all Linux-based clouds.

Docker Containers

Docker (www.docker.com) is an open source project providing a systematic way to automate the faster deployment of Linux applications inside portable containers. Basically, Docker extends LXC with a kernel-and application-level API that together run processes in isolation: CPU, memory, I/O, network, and so on. Docker also uses namespaces to completely isolate an application's view of the underlying operating environment, including process trees, network, user IDs, and file systems.

Docker containers are created using base images. A Docker image can include just the OS fundamentals, or it can consist of a sophisticated prebuilt application stack ready for launch. When building images with Docker, each action taken (that is, command executed, such as apt-get install) forms a new layer on top of the previous one. Commands can be executed manually or automatically using Dockerfiles.

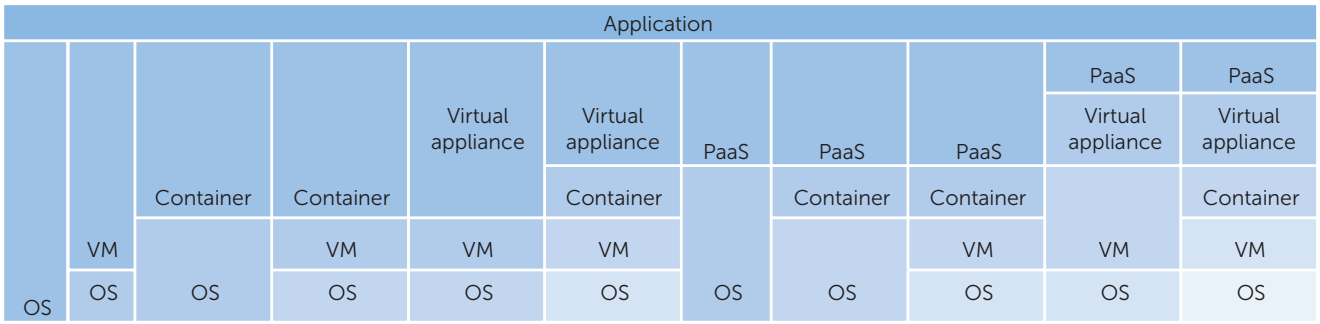

Figure 2. Possible layering combinations for application runtimes.

Each Dockerfile is a script composed of various commands (instructions) and arguments listed successively to automatically perform actions on a base image to create (or form) a new image. They're used to organize deployment artifacts and simplify the deployment process from start to finish.

Containers can run on VMs too. If a cloud has the right native container runtime (such as some of the clouds mentioned) a container can run directly on the VM. If the cloud only supports hypervisor-based VMs, there's no problem—the entire application, container, and OS stack can be placed on a VM and run just like any other application to the OS stack.

Abstractions on Top of VMs and Containers

Both VMs and containers provide a rather low-level construct. Basically, both present an operating system interface to the developer. In the case of the VM, it's a complete implementation of the OS; you can run any OS that runs on the bare metal. The container gives you a "view" or a "slice" of an OS already running. You access OS constructs as if you were running an application directly on the OS. Developers often build on this level of abstraction to provide more application runtime constructs, so users don't feel like they're running on a bare machine or a bare OS, but on an application runtime of some kind.

Virtual appliances, such as VirtualBox (www.virtualbox.org), Rightscale Appliance,[2] and Bitnami (https://bitnami.com), provide application runtime environments that shield the application from the bare OS by providing an interface for applications with higher-level, more portable constructs. Virtual appliances gained popularity with equipment manufacturers who wanted to provide a vehicle for distributing software versions of an appliance—for example, a network load balancer, WAN optimizer, or firewall. Virtual appliances can run on top of a VM or a container (native LXC-based or running on top of a VM).

For even more isolation from the OS, especially desired by application programmers, application runtimes can be reconfigured into total platform-as-a-service (PaaS) runtimes. Readers will remember that last issue I discussed Cloud Foundry PaaS, and mentioned that it uses container technology for deployment. It's for precisely this reason they do so—the distribution can be targeted precisely for the container engine and Linux OS on the cloud, and like the virtual appliance can also run on top of a VM.

As Figure 2 shows, there are many possible layering combinations, depending on the OS's capabilities, the deployment/portability strategy, and whether a PaaS is used.

How does one choose? As mentioned earlier, the virtual appliance approach is a favorite vehicle used by network equipment manufacturers to create a portable software appliance.

Those who want to deploy applications with the least infrastructure will choose the simple container-to-OS approach. This is why container-based cloud vendors can claim improved performance when compared to hypervisor-based clouds. A recent benchmark of a "fast data" NewSQL system claimed that in an apples-to-apples comparison, running on IBM Softlayer using containers resulted in a fivefold performance improvement over the same benchmark running on Amazon AWS using a hypervisor.[3]

Software developers tend to prefer using PaaS, which will use a container if available for its runtime, to maximize performance as well as to manage application clustering. If not, the PaaS will run a container on a VM. Consequently, as PaaS gains in popularity, so do containers.

However, using containers for security isolation might not be a good idea. In an August 2013 blog,[4] one of Docker's engineers expressed optimism that containers would eventually catch up to VMs from a security standpoint. But in a presentation given in January 2014,[5] the same engineer said that the only way to have real isolation with Docker is to either run one Docker per host, or one Docker per VM. If high security is needed, it might be worth sacrificing the performance of a pure-container deployment by introducing a VM to obtain more tried and true isolation. As with any other technology, you need to know the deployment's security requirements, and make appropriate decisions.

Open Source Cluster Manager for Docker Containers

As mentioned earlier, one of containers' nicest features is that they can be managed specifically for application clustering, especially when used in a PaaS environment. Answering this need, at the June 2014 Google Developer Forum, Google announced *Kubernetes*, an open source cluster manager for Docker containers.[6] According to Google, "Kubernetes is the decoupling of application containers from the details of the systems on which they run. Google Cloud Platform provides a homogenous set of raw resources . . . to Kubernetes, and in turn, Kubernetes schedules containers to use those resources. This decoupling simplifies application development since users only ask for abstract resources like cores and memory, and it also simplifies data center operations."

Google goes on to describe network-centric deployment improvements in Kubernetes: "While running individual containers is sufficient for some use cases, the real power of containers comes from implementing distributed systems, and to do this you need a network. However, you don't just need any network. Containers provide end users with an abstraction that makes each container a self-contained unit of computation. Traditionally, one place where this has broken down is networking, where containers are exposed on the network via the shared host machine's address. In Kubernetes, we've taken an alternative approach: that each group of containers (called a Pod) deserves its own, unique IP address that's reachable from any other Pod in the cluster, whether they're co-located on the same physical machine or not."

Industry Movement around Kubernetes

Shortly after Google's announcements, several players endorsed Kubernetes—and therefore Docker and containers—as a core cloud deployment technology.[7] In addition to a host of start-ups (such as CoreOS, MesoSphere, and SaltStack), Kubernetes supporters include:

- Google (for Google Cloud Engine, GCE),
- Microsoft (for Microsoft Azure),
- VMware,
- IBM (for Softlayer and OpenStack), and
- Red Hat (its OpenStack distribution).

Although HP, Canonical, AWS, and Rackspace are "Docker friendly," they haven't explicitly endorsed Kubernetes. Industry speculation is that once a more neutral governance/collaboration structure is put together around Docker (a start-up company) and Kubernetes (still controlled by Google), organizations will agree on a common packaging and deployment approach—and here we have practically everyone already thinking about it. I'm not aware of any cloud project with this level of alignment on anything!

CONTAINERS, DOCKER, AND KUBERNETES SEEM TO HAVE SPARKED THE HOPE OF A UNIVERSAL CLOUD APPLICATION AND DEPLOYMENT TECHNOLOGY. And that, my friends, qualified it to be this issue's Cloud Tidbit. I hope you enjoyed it!

References

1. P. Mell and T. Grance, *The NIST Definition of Cloud Computing: Recommendations of the National Institute of Standards and Technology*, NIST Special Publication 800-145, 2011.
2. U. Thakrar, "Introducing RightScale Cloud Appliance for vSphere," blog, 10 Dec. 2013; www.rightscale.com/blog/enterprise-cloud-strategies/introducing-rightscale-cloud-appliance-vsphere.
3. B. Kepes, "VoltDB Puts the Boot into Amazon Web Services, Claims IBM Is Five Times Faster," *Forbes*, 6 Aug. 2014; www.forbes.com/sites/benkepes/2014/08/06/voltdb-puts-the-boot-into-amazon-web-services-claims-ibm-5-faster.
4. J. Petazzoni, "Containers & Docker: How Secure Are They?" blog, 21 Aug. 2013; http://blog.docker.com/2013/08/containers-docker-how-secure-are-they.
5. J. Petazzoni, "Linux Containers (LXC), Docker, and Security," 31 Jan. 2014; www.slideshare.net/jpetazzo/linux-containers-lxc-docker-and-security.
6. C. Mcluckie, "Containers, VMs, Kubernetes and VMware," blog, 25 Aug. 2014; http://googlecloudplatform.blogspot.com/2014/08/containers-vms-kubernetes-and-vmware.html.
7. B. Butler, "Containers: Buzzword du Jour, or Game-Changing Technology?" *NetworkWorld*, 3 Sept. 2014; www.networkworld.com/article/2601925/cloud-computing/container-party-vmware-microsoft-cisco-and-red-hat-all-get-in-on-app-hoopla.html.

DAVID BERNSTEIN *is the managing director of Cloud Strategy Partners, cofounder of the IEEE Cloud Computing Initiative, founding chair of the IEEE P2302 Working Group, and originator and chief architect of the IEEE Intercloud Testbed Project. His research interests include cloud computing, distributed systems, and converged communications. Bernstein was a University of California Regents Scholar with highest honors BS degrees in both mathematics and physics. Contact him at david@cloudstrategypartners.com.*