Virtualization VS Containers

Maryam MARIH

December 3, 2014

Contents

Chapter 1

Virtualization

1.1 Introduction :

Many people tend to confuse virtualization with other forms of technology and sometimes confuse the difference between virtualization and say, cloud computing. Virtualization is the process of creating a “virtual” version of something, such as computing environments, operating systems, storage devices or network components, instead of utilizing a physical version for certain aspects of a company’s infrastructure.For example, server virtualization is the process in which multiple operating systems (OS) and applications run on the same server at the same time, as opposed to one server running one operating system. We can think of it as one large server being cut into pieces. The server then imitates or pretends to be multiple servers on the network when in reality it’s only one.

1.2 Types of Virtualization :

Virtualization can be utilized in many different ways and can take many forms. The main types include application, desktop, user, network, storage and server.

- **Application virtualization** allows the user to access the application, not from their workstation, but from a remotely located server. The server stores all personal information and other characteristics of the application, but can still run on a local workstation. Technically, the application is not installed, but acts like it is.

- **Desktop virtualization** allows the users’ OS to be remotely stored on a server in the data center, allowing the user to then access their desktop virtually, from any location.

- **User virtualization** is pretty similar to desktop, but allows users the ability to maintain a fully personalized virtual desktop when not on the company network. Users can basically log into their “desktop” from different types of devices like smartphones and tablets.

- **Network Virtualization** with network virtualization, the network is "carved up" and can be used for multiple purposes such as running a protocol analyzer inside an Ethernet switch. Components of a virtual network could include NICs, switches, VLANs, network storage devices, virtual network containers, and network media.
- Storage virtualization is the process of grouping the physical storage from multiple network storage devices so that it acts as if it's on one storage device.
- **Server Virtualization** Using server virtualization, multiple operating systems can run on a single physical server as virtual machines, each with access to the underlying server's computing resources. Most servers operate less than 15 percent of capacity; not only is this highly inefficient, it also introduces server sprawl and complexity. Server virtualization addresses these inefficiencies.

There are three ways to create virtual servers: **full virtualization, para-virtualization** and **OS-level virtualization**. They all share a few common traits. The physical server is called the host. The virtual servers are called guests. The virtual servers behave like physical machines. Each system uses a different approach to allocate physical server resources to virtual server needs.

1.3 What Benefits Does Virtualization Provide?

There are numerous benefits to virtualization including decreasing costs, saving time and energy and minimizing risk overall.

Benefits for Companies :

- Greater efficiency and company agility
- Ability to more-effectively manage resources
- Increased productivity, as employees access the company network from any location
- Data stored on one centralized server results in a decrease in risk of lost or stolen data

Benefits for Data Centers :

Not only is it beneficial for companies, but virtualization provides several benefits for data centers as well, including:

- Cutting waste and costs associated with maintaining and cooling its servers by maximizing the capabilities of one server

- Allows data centers to be smaller in size, resulting in overall savings due to a reduction in –

1. Energy needed
2. Hardware used
3. Time and money needed for maintenance

1.4 Hypervisor Virtualization

When we're talking about hypervisor virtualization on Linux, we're talking about running a full operating system on top of a host operating system, usually a mixture of operating systems. For example, running Red Hat Enterprise Linux and Microsoft Server on top of a Red Hat Enterprise Linux (RHEL) host using VMware or Xen. Hypervisor virtualization is more or less operating system agnostic when it comes to guest OSes, though in practice it's not guaranteed that every x86/x86-64 guest is going to run flawlessly on top of any given hypervisor-based solution.

Initially, virtualization options on Linux x86 were limited to full virtualization, where the virtualization software had to completely isolate the guest operating system and emulate the hardware entirely. The x86 platform was not originally designed with virtualization in mind, though the concept of virtualization had been around since the 70s. One of the first virtualization solutions on the scene for Linux was VMware's Workstation in 1998 or so. Back then, it was a novel idea that you could run a guest OS on top of Linux. A lot of folks used hypervisor virtualization as a way to run a couple of Windows apps on top of Linux in order to move away from Windows on the desktop – or just to consolidate their OSes onto one workstation or laptop so they could avoid dual-booting or maintaining two machines. Performance took a hit, but overall it was usable for desktop machines.

Fast forward a bit more than a decade, and we have plenty of choice for running multiple OSes on the desktop and server. In fact, hypervisor virtualization is actually baked into the Linux kernel now in the form of the Linux Kernel Virtual Machine, which is a way to turn the Linux kernel into a hypervisor by adding a kernel module.

Some solutions offer what's called paravirtualization, which requires the guest operating systems to be modified. Paravirtualization allows the guest OS to interact more directly with the host system's hardware, and it provides a performance benefit for the guest OS. The downside is that the guest OS has to be modified to be aware that it's being virtualized. This limits one of the advantages of hypervisor virtualization, which is that you can typically run almost any operating system that's designed for that hardware architecture.

For example, using Linux as a guest or host OS for Xen is not a problem if the kernel supports it. It's not a default feature in the kernel, but many distros add the appropriate patches for Xen to work as a host (dom0) or guest (domU). You're a bit more limited, though, in running older releases of Microsoft Windows Server on top of Xen because they're not Xen-aware.

The primary advantage of hypervisor-based solutions is that they allow you to run a fuller range of operating systems. Solutions like KVM, VMware Server or ESX, Parallels Server Bare Metal and Xen (to name only a few) allow you to run just about any x86 operating system as a guest on a wide variety of host OSes.

This is very effective for server consolidation when you're looking at consolidating existing workloads into a virtualized environment. So, if you have 20 servers running Microsoft Windows Server 2000 and 50 servers running SUSE Linux Enterprise Server 9, you'd likely want to look at consolidating on top of one of the hypervisor solutions.

Performance, however, is likely to take at least a slight hit when running on a hypervisor. You're introducing an extra layer of abstraction between the operating system and hardware with hypervisor virtualization, and you'll see some performance impact as a result. You also have to have a complete OS stack for each guest when using hypervisor virtualization, from the kernel to libraries, applications, and so on. So you'll have additional storage overhead and memory use from running OSes entirely separate.

That being said, performance is less of a factor today than a few years ago. The various hypervisor solutions have been pretty well-optimized for heavy loads, and continue to improve at a good clip.

Chapter 2

Containers

2.1 Definitions :

The other contender is container-based virtualization. Container-based virtualization is also called operating system virtualization, and sometimes it's not really talked about as virtualization at all. One of the first container technologies on x86 was actually on FreeBSD, in the form of FreeBSD Jails.

Instead of trying to run an entire guest OS, container virtualization isolates the guests, but doesn't try to virtualize the hardware. Instead, you have containers (hence the name) for each virtual environment. With container-based technologies, you'll need a patched kernel and user tools to run the virtual environments. The kernel provides process isolation and performs resource management. This means that even though all the virtual machines are running under the same kernel, they effectively have their own filesystem, processes, memory, devices, etc.

The net effect is very similar to hypervisor virtualization, and there's a good chance that users of the guest systems will never know the difference between using a system that's running on bare metal, under a hypervisor, or in a container. However, the actual experience for admins and planning for deployment varies

With container-based virtualization, installing a guest OS is not as straightforward as hypervisor solutions. That is, you can't just pop in a DVD or CD to whip up a new guest machine. You'll need to create a container template, if you're using something like OpenVZ or Parallels Virtuozzo Containers. Usually you'll not need to create these on your own, though as OpenVZ provides quite a few templates and Parallels provides supported templates for its users.

You're also usually limited to a single operating system with containers. You can't run Linux and Windows together, for example. In some configurations, you can run Linux and Solaris/OpenSolaris using Solaris Zones, but for Linux virtualization, you're limited to running Linux guests on a Linux host. You're not limited to a specific distribution, though. You can run Debian and RHEL templates, for instance, on top of a CentOS host running OpenVZ or Virtuozzo.

If this sounds limiting, remember there are also advantages to container-based virtualization in terms of performance and scalability. Hypervisor virtualization usually has limits in terms of how many CPUs and how much memory a guest can address, whereas the container-based solutions should be able to address as many CPUs and as much RAM as the host kernel.

It mostly breaks down to the type of workload you have. If you're going to be deploying dozens or hundreds of Linux guests, then a container-based solution works very well and might be a better option over hypervisor virtualization. Containers are especially popular in hosting environments or any scenario where there's a need to consolidate a large number of Linux instances.

2.2 Docker :

Docker is an open platform for developers and sysadmins to build, ship, and run distributed applications. Consisting of Docker Engine, a portable, lightweight runtime and packaging tool, and Docker Hub, a cloud service for sharing applications and automating workflows, Docker enables apps to be quickly assembled from components and eliminates the friction between development, QA, and production environments. As a result, IT can ship faster and run the same app, unchanged, on laptops, data center VMs, and any cloud.

Chapter 3

Containers VS Virtualization

3.1 Difference between Docker Containers and Virtual Machine :

Virtual machines have a full OS with its own memory management installed with the associated overhead of virtual device drivers. In a virtual machine, valuable resources are emulated for the guest OS and hypervisor, which makes it possible to run many instances of one or more operating systems in parallel on a single machine (or host). Every guest OS runs as an individual entity from the host system.

On the other hand Docker containers are executed with the Docker engine rather than the hypervisor. Containers are therefore smaller than Virtual Machines and enable faster start up with better performance, less isolation and greater compatibility possible due to sharing of the host's kernel.

3.2 Docker Containers versus Virtual Machines :

When it comes to comparing the two, it could be said that Docker Containers have much more potential than Virtual Machines. It's evident as Docker Containers are able to share a single kernel and share application libraries. Containers present a lower system overhead than Virtual Machines and performance of the application inside a container is generally same or better as compared to the same application running within a Virtual Machine.

There is one key metric where Docker Containers are weaker than Virtual Machines, and that's "Isolation". Intel's VT-d and VT- x technologies have provided Virtual Machines with ring-1 hardware isolation of which, it takes full advantage. It helps Virtual Machines from breaking down and interfering with each other. Docker Containers yet don't have any hardware isolation, thus making them receptive to exploits.

As compared to virtual machines, containers can be faster and less resource heavy as long as the user is willing to stick to a single platform to provide the shared OS. A virtual machine could take up several minutes to create and launch whereas a container can be created and launched just in a few seconds. Applications contained in containers offer superior performance, compared to running the application within a virtual machine.

There is an estimation being done by Docker that application running in a container can go twice as fast as one in a virtual machine. Also, a single server can pack more than one containers as OS is not duplicated for each application.

3.3 Virtual Machines and Containers: better together

You can sometimes use a hybrid approach which uses both VM and Docker. There are also workloads which are best suited for physical hardware. If both are placed in a hybrid approach, it might lead to a better and efficient scenario. With this Hybrid setup, users can benefit from the advantages if they have workloads that fit the model.

Following are a few of them, that explain how they work together as a Hybrid:

1. Docker Containers and Virtual Machines by themselves are not sufficient to operate an application in production. So one should be considering how are the Docker Containers going to run in an enterprise data center.

2. Application probability and enabling the accordant provisioning of the application across infrastructure is provided by containers. But other operational requirements such as security, performance and capacity management and various management tool integrations are still a challenge in front of Docker Containers, thus leaving everyone in a big puzzle.

3. Security isolation can be equally achieved by both Docker Containers and Virtual Machines.

4. Docker Containers can run inside Virtual Machines though they are positioned as two separate technologies and provide them with pros like proven isolation, security properties, mobility, dynamic virtual networking, software-defined storage and massive ecosystem.

3.4 Who wins amongst the two?

Answer to this question so far cannot be ascertained but depending upon their configurations and constraints one could say that containers are overcoming virtual machines. Application design is the one standpoint suggesting which one of the two should be chosen. If application is designed to provide scalability and high availability then containers are the best choice else application can be placed in a virtual machine, though Docker containers have surely challenged virtualization market with containers. Well, keeping the debate aside, it is easy to say that containers in Virtual Machines are twice as robust as one without the other.

Chapter 4

Example

Google Runs All Software In Containers

The overhead of full-on server virtualization is too much for a lot of hyperscale datacenter operators as well as their peers (some might say rivals) in the supercomputing arena. But the ease of management and resource allocation control that comes from virtualization are hard to resist and this has fomented a third option between bare metal and server virtualization. It is called containerization and Google recently gave a glimpse into how it is using containers at scale on its internal infrastructure as well as on its public cloud. We are talking about billions of containers being fired up a week here, just so you get a sense of the scale.

Google began its journey into containerization in the mid-2000s when some engineers donated a tech named cgroups into the Linux kernel. This technology "provides a mechanism for aggregating/partitioning sets of tasks, and all their future children, into hierarchical groups with specialized behaviour."

In the same way Google's publication of the MapReduce and GFS papers let Yahoo! engineers create open source data analysis framework Hadoop, the addition of cgroups let to further innovations by other companies.

The kernel feature cgroups became a crucial component of LXC, LinuX Containers, which combines cgroups with network namespaces to make it easy to create containers and the ways they need to connect with other infrastructure. While useful, this still required sophisticated users.

Beda, a senior staff software engineer for Google Cloud Platform, the public cloud arm of the search engine giant, talked about the company's decade-long effort to create a more lightweight virtualization layer and how it is deployed internally on Google's infrastructure.

Beda was not specific about what different types and levels of containers Google uses in its massive infrastructure (with well over 1 million servers scattered around the globe) but he did say that "everything at Google runs in a container" and that "we start over 2 billion containers per week."

Google didn't say how many containers it retires per week. Presumably this is also a large number as workloads shift around. Here is what the container stack looks like at Google, conceptually:

Google's take on containerization is slightly different, as it places more emphasis on performance and less on ease of use. To try to help developers understand the difference, Google has developed a variant of LXC named, charmingly, lmctfy, short for Let Me Contain That For You.

Google describes lmctfy as "the open source version of Google's container stack, which provides Linux application containers. These containers allow for the isolation of resources used by multiple applications running on a single machine. This gives the applications the impression of running exclusively on a machine. The applications may be container-aware and thus be able to create and manage their own subcontainers," the company explains on its Github page.

"The project aims to provide the container abstraction through a high-level API built around user intent," Google writes. "The containers created are themselves container-aware within the hierarchy and can be delegated to be managed by other user agents.

For its part, Docker isn't threatened by lmctfy, and plans to run it as an optional execution engine within the Docker software.

Google has been on a mission to come up with ways to isolate workloads from each other on shared hardware for the past decade.The company came up with some limited isolation for its Linux-based systems back in 2004 and two years later Google cooked up something called control groups, shortened to cgroups, as a Linux kernel extension that allowed for the control of allocation of resources for single processes on systems all the way up to various kinds of containers on Linux. The cgroups code was also created to allow for checkpointing of processes so they could be recovered, to set priorities for resources for code, and to measure what processes in software were consuming specific resources for chargeback. Google uses cgroups as the underlying container virtualization layer and layers KVM on top of that when it needs to run an operating system other than the raw cgroups variant of Linux it has.

The cgroups code has been merged into the Linux kernel and is the basis for the Linux containers (LXC) that is now being commercialized in the latest Linux releases from Red Hat, SUSE Linux, Canonical, and others. They are distinct, however. Just like lmctfy and Docker, another Linux container technology, are different.

Google came up with its own variant of LXC containers, which it calls lmctfy and which it launched in 2013. The software's name stands for Let Me Contain That For You and is currently in beta at the 0.5.0 release level; it is open source software so you can play around with it as you see fit. While cgroups and LXC are a little bit complex to manage, lmctfy containers are supposed to be easier to manage and to hook into job schedulers running on clusters of machines. Lmctfy has two layers, called CL1, which currently provides CPU and memory capacity isolation in an application container, and CL2, which is a Linux daemon that sets the policies and enforces them for the CL1 layer. CL2 is still in development.

At the moment, lmctfy does not allow for the overcommitment of resources, a common feature in bare metal hypervisors that helps to drive up utilization on systems. Over time, the CL1 layer will provide for disk and network isolation, support for root file systems and the importing and exporting of disk images based on those root file systems, and checkpoint and restore on containers. The CL2 daemon controls the fair sharing of resources on a machine and will eventually be updated with code to set quality of service levels and enforce them as well as monitoring the containers and extracting statistics from them for job schedulers. Google has tested lmctfy on the Ubuntu Server 3.3 and 3.8 kernels but says it will work on others.

Interestingly, Beda said that Google does have overcommitment features for its internal infrastructure but not for its Compute Engine public cloud.

The main thing about lmctfy is that it separates the container policies from the enforcement of those policies, which is an engineering approach that Google takes again and again in the hyperscale systems it designs. (The centralized, hierarchical approach was demonstrated again back in April.

Chapter 5

Conclusion : Containers, Virtualization and Docker

People often compare containers to virtual machines, given they have similar isolation concepts

The notion of a “container” is that it provides operating system-level process isolation, similar in concept to hardware virtualization. The difference is that the isolation is done in the OS rather than at the hardware abstraction layer. Containers have been around in various forms for years. Google realized the potential of containers early on as well and started contributing to the Linux kernel to add process isolation functionality to various subsystems.

Even though Linux was building a fledging set of container technologies and even with Google strongly advocating containers for application delivery, not many people noticed or showed interest in containers until Docker hit the scene almost a year and a half ago. Docker has taken Linux containers to the next level by creating a very simple and elegant application packaging system that leverages containers and enables true portability across Linux distros and between dev and prod environments, while enabling efficient creation of a full application image and its associated libraries. Docker has also crafted their APIs such that they fit seamlessly into the developer workflow. In other words, Docker made containers easy and approachable for any and every developer.

Bibliography

[1] http://www.enterprisetech.com/2014/05/28/google-runs-software-containers/

[2] http://www.theregister.co.uk/2014/05/23/google_containerization_two_billion/

[3] http://www.linux.com/component/content/article/186-virtualization/300057-containers-vs-hypervisors-choosing-the-best-virtualization-technology-

[4] http://www.networkcomputing.com/cloud-infrastructure/virtual-machines-vs-containers-a-matter-of-scope/a/d-id/1269190

[5] http://devops.com/blogs/devops-toolbox/docker-vs-vms/

[6] Virtualisation [2012] - Nicolas DEWAELE - Cours GRETA Académie du Nord

[7] http://www.networkcomputing.com/cloud-infrastructure/virtual-machines-vs-containers-a-matter-of-scope/a/d-id/1269190

[8] https://www.docker.com/whatisdocker/

[9] http://blogs.vmware.com/cto/vmware-containers-containers-without-compromise/