Zenoss

The Zenoss Enablement Series:

Virtualization and Docker Containerization for Poets

Document Version 500-P2

Zenoss, Inc.

www.zenoss.com

Zenoss, Inc.

Copyright © 2014 Zenoss, Inc 11305 Four Points Drive, Bldg 1 - Suite 300, Austin, Texas 78726, U.S.A.
All rights reserved.

Zenoss and the Zenoss logo are trademarks or registered trademarks of Zenoss, Inc. in the United States and other countries. All other trademarks, logos, and service marks are the property of Zenoss or other third parties. Use of these marks is prohibited without the express written consent of Zenoss, Inc. or the third-party owner.

Cisco, Cisco UCS, Cisco Unified Computing System, Cisco Catalyst, and Cisco Nexus are trademarks or registered trademarks of Cisco and/or its affiliates in the United States and certain other countries.

Flash is a registered trademark of Adobe Systems Incorporated.

Oracle, the Oracle logo, Java, and MySQL are registered trademarks of the Oracle Corporation and/or its affiliates.

Linux is a registered trademark of Linus Torvalds.

SNMP Informant is a trademark of Garth K. Williams (Informant Systems, Inc.). Sybase is a registered trademark of Sybase, Inc.

Tomcat is a trademark of the Apache Software Foundation.

vSphere is a trademark of VMware, Inc. in the United States and/or other jurisdictions.

Windows is a registered trademark of Microsoft Corporation in the United States and other countries. All other companies and products mentioned are trademarks and property of their respective owners.

Table of Contents

Virtualization and Docker Containerization for Poets

With the release of the Europa version of Zenoss fast approaching, some system administrators may be struggling to understand exactly what Docker is and what “containerization” of software means.

The public Docker Website draws an analogy between Docker containers in the software world and shipping items within standardized containers for transport overseas in the physical world. There might be a better analogy, but before we conjure an alternative it will be instructive to take a conceptual step all the way back to what operating systems and applications are in the first place. If you're a technical reader, you may have a pretty good grasp on these concepts already and can skim over the first two sections of this document. If you're a non-technical reader, read straight through the article and perhaps by the end of it you'll have a whole new understanding of some of our industry's technical jargon – including, hopefully, what Docker containerization is!

Let’s Start at the Beginning: What is an Operating System?

Have you ever stopped to really ponder that question? If you're technical, you may have found yourself trying to describe what an operating system is to a friend or a loved one who asks something like “what is this Linux thing you keep talking about? How is that different from Windows?” If you're not technical, you may never have asked yourself what Windows or Mac OS X “are.” They're just the stuff that boots when you turn on your computer, right?

In either case, it will be helpful to use a framework to boil the question down. This framework fits the bill: an operating system is a collection of software that provides an environment within which application software can run. In this context we consider “application software” to be anything that provides or supports a service that humans consume – either directly or indirectly. Examples of “applications” as that term is applied here would be email server programs, Microsoft Office programs, web server programs, database server programs, and apps on mobile devices like games.

With that foundation laid, we can look to the physical world for an analogy to the operating system by asking a similarly odd question about the physical world that very few people are likely to have pondered: what is a fish tank? The concepts of an operating system in the software world and a fish tank the physical world are actually pretty much the same. In setting up a fish tank, what we are doing is creating an environment with everything fish need to thrive. One can draw the parallel between the tank's glass enclosure and the physical hardware an operating system runs on (“hardware” includes anything from the laptop you are reading this on to a server in a data center). Fish tanks come in different sizes – from the tiny goldfish bowl (draw an analogy here to your mobile phone handset) all the way up to a tank in the Baltimore Aquarium (= an enterprise grade server).

The fish tank is more than the glass enclosure, though. The tank includes water, of course, but also supporting elements like filters, heaters, lighting, bottom cover, food and many other features that provide the precise environment the tank's inhabitants need to thrive. Consider some aspects of that environment: the correct water temperature, cleanliness, water PH balance and lighting. Similarly, an operating system provides the environment applications need to thrive. Aspects of that environment include access to needed software libraries, connectivity to input devices like keyboards or touch screens, network access for connectivity, access to disks for storage, and much more.

Parenthetically, there is an interesting twist on the analogy we have drawn here in the process of how software gets created. In the software world, the “fish” get “created” to suit an environment, instead of the other way around! We set up fish tanks to suit the needs of different species, for example simple fresh water environments for goldfish, or highly specific and complex environments for fragile salt water species. In the

software world, a small handful of operating system environments dominate. Among the most common are Windows, Unix, Linux, and Mac OS X. An individual or organization that is interested in creating an application for gratification or for profit decides which of these environments is best suited to reach the intended consumers, then crafts the application to run on one or more of those environments. If you're reading this document as a PDF, for example, someone (Adobe?) wrote an application (Adobe Reader?) for the specific operating system you are running (Microsoft Windows 7?). The version of that software written for Windows can't run (“live') on an Apple Macintosh. So if the creators wanted to provide the application to Mac users, they would need to write an entirely different version of the software for Mac OS X.

What is Virtualization?

Remember, we're poets here, so we need to progress in stages from a better understanding of operating systems to an understanding of Docker containerization. Virtualization can be thought of as an intermediate step to on the way to Docker containers. So what is virtualization?

We can consider a real world example to best understand virtualization. In the section above we discussed the concept of applications being written to run within particular environments (for example, written “for Windows” or “for Linux”), and we talked about application writers making a decision as to which environment was best suited to reach their intended consumer. In some cases, as with Adobe Reader, the software writers create versions for multiple operating systems to reach the widest possible range of consumers. But in other examples they may not. This creates a difficulty for system administrators who may have a group of applications that run in one environment, but still wish to use one or more applications written for another. Let’s consider a concrete example. Suppose there is a hypothetical media company that hosts websites, and that the website server applications they use run on Linux. Perhaps they decide that they also wish to begin hosting their own Microsoft Exchange email server program, which only runs on the Microsoft Windows operating system. Before the days of virtualization, they would need to add an additional server to their farm running Microsoft Windows, which is the only operating system Exchange runs on.

That’s not a major problem right? They can just buy another server. But here's the rub: as the need to buy servers specifically to host applications that conflict with other applications (either because they run on different operating systems or because they literally conflict with one another [think of fish that fight if they are placed in the same tank!]) the number of servers in given environment can grow large, with many servers being underutilized. That creates a waste of resources and higher than necessary expenses. Drawing an analogy back to the physical world, this would represent an aquarium needing to maintain a bunch of extra tanks each with only a few fish in it.

Enter virtualization to solve the problem. Virtualization is a technology that allows a physical server to run multiple operating systems inside of it. These “guest” operating systems run as “programs” within the “host” (main) operating system. In our hypothetical example above, the web hosting company could simply install virtualization software on one of their Linux servers and then run a copy of Windows on the same server as a “program” within Linux[1]. That Windows operating system could host their Exchange email server program. Now they avoid the expense of purchasing a whole new server, and yet they are able to run both their Linux and Windows applications on the same physical server. With virtualization, they cut their expenses by running a smaller number of servers with higher capacity utilization.

Turning back to our fish tank analogy, what we have with virtualization is a fish tank into which other fish tanks have been deposited, to isolate groups of fish from one another. In this analogy each fish tank within the larger tank is a complete tank, however. It includes not only the glass enclosure, but the ground cover, filters, lighting, heaters, etc.

[1] Over time virtualization evolved to become even more efficient, as vendors of virtualization software such as VMware created specialized, very small versions of operating systems designed to do nothing more than host virtualization. These specialized operating systems are known of as “hypervisors,” and are often highly simplified versions of Linux or, in Microsoft's case, Windows.

What is Docker Containerization?

Virtualization improves hardware utilization and lowers costs, but considering how it works more closely, you will notice that there is a large duplication of resources between “guest” (hosted) operating systems. Turning back to the fish tank analogy, does each tank really need to have its own filters, lighting, etc.? Does everyone need to entirely isolate the fish using a complete glass enclosure? What if there was a better way – a way to shrink down each of those tanks into only those resources each application uniquely needed and no others, allowing each application to share common resources? That 'crush down' of virtualization pretty much captures what containerization means. It means packaging up applications with only what they uniquely need, and running them within an operating system as isolated, independent ‘containers.’

These containers are smaller and less complex than complete operating systems. We can think of them as little magic wire mesh baskets with only the ground cover, lighting, food etc. that each fish uniquely needs inside the basket. Now many more different kinds of “fish” can live within a “tank” - each one isolated from the others (to prevent conflicts). Resources needed by the fish inside the basket that are common to the other fish are shared. For example, if the water in the rest of the tank works for the fish in the basket, water is allowed to flow through the mesh. But resources the fish uniquely need are packed within the basket just for them.

Apart from more efficient use of server hardware, containerization has benefits such as excellent portability between host systems - because each “basket” is smaller and lighter than a whole tank, it is much easier to move from one “tank” to another. That allows for whole groups of servers (known of as “resource pools”) to be assembled to host applications, across which application containers can be moved as needed to balance the load across hosts.

It all sounds pretty good, but naturally there are pros and cons to containerization as opposed to visualization. Here is a list of the negatives associated with containerization, and the ways in which Zenoss is able to address them in Version 5.0 of Resource Manager:

- Containers can only be created for applications written for operating systems that are similar, but not necessarily identical to, the host operating system. For example, while you can create a container for an application that runs on the “CentOS” version of the Linux operating system and run that container on a host running the “Ubuntu” version of Linux, one couldn't create a container for a Windows application and run it within a Linux host. This limitation intuitively makes sense because by definition nothing would be shared between the container and the host operating system, so in that case virtualization would be the better solution in any case. As such, this limitation isn't a large drawback in practice.
- Isolating applications within containers creates the need route network traffic into and out of each container. This can cause the need for complicated network configurations as the number of containers grows. Fortunately, the Docker community of software engineers has created a variety of software platforms to automate this work. Better still, our own Zenoss engineering team has written one customized for Resource Manager 5.x. It’s called Zenoss Control Center, and it, too is an open source project.
- Access to the process running within a container is limited, making configuration and diagnostics more complex than for virtualized operating system. In the specific case of Zenoss Resource Manager, however, the configuration files and logs have been centralized in Control Center, mitigating this downside.

The combination of Docker with Zenoss Control Center provides the best of all worlds for the next release of Zenoss. Control Center allows us to take advantage of the tremendous increases in flexibility and scalability offered by Docker while overcoming this powerful technology’s limited downsides.