electronic design

print | close

What's the Difference Between Containers and Virtual Machines?

Electronic Design
William Wong
Fri, 2016-07-15 16:32

Embedded developers need to deploy ever-more complex systems to take advantage of hardware whose functionality and performance continue to grow at accelerated rates. Writing a single application is still manageable for a small microcontroller, but it's impractical for larger systems that need a more modular approach. Adding an operating system to the mix makes multiple applications easier to work with; however, these days even a single operating system isn't always the best solution.

Related

How To Choose The Right Hypervisor

Hypervisor Supports Virtual Machines

Embedded Hypervisor Delivers Separation Kernel

Hypervisors are a way to manage virtual machines (VMs) on processors that support the virtual replication of hardware. Not all processors have this type of hardware—it's typically found in mid- to high-end microprocessors. It's standard fare on server processors like Intel's Xeon and found on most application processors such as the Arm Cortex-A series. Typically, a VM will run any software that runs on the bare metal hardware while providing isolation from the real hardware. Type 1 hypervisors run on bare metal, while Type 2s have an underlying operating system *(see figure, a)*.

Containers vs. VMs

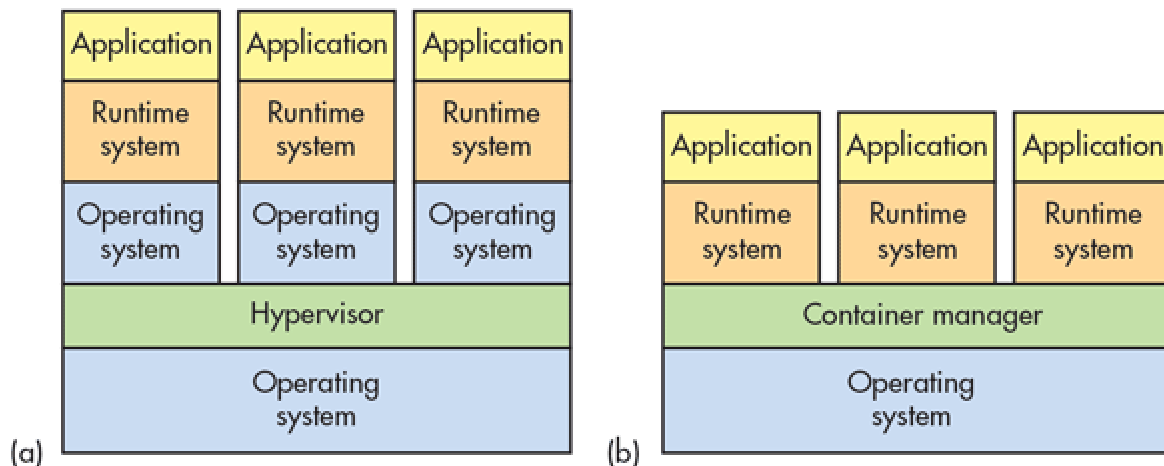
Containers also provide a way to isolate applications and provide a virtual platform for applications to run on *(see figure, b)*. Two main differences exist between a container and a hypervisor system.

The container's system requires an underlying operating system that provides the basic services to all of the containerized applications using virtual-memory support for isolation. A hypervisor, on the other hand, runs VMs that have their own operating system using hardware VM support. Container systems have a lower overhead than VMs and container systems typically target environments where thousands of containers are in play. Container systems usually provide service isolation between containers. As a result, container services such as file systems or network support can have limited resource access.

There is also something called para-virtualization, which is sort of a mix between the two approaches. It uses virtual-memory support for isolation, but it requires special device drivers in the VM that are linked through the hypervisor to the underlying operating system, which in turn provides the device services.

A hardware VM system forces any communication with a VM to go through the hardware. Some systems allow real hardware to map directly to a VM's environment, enabling the VM's device driver to directly handle the hardware. Hardware I/O virtualization also allows a single hardware device like an Ethernet adapter to present

ltiple, virtual instances of itself so that multiple VMs can manage their instance directly.



In a nutshell, a VM provides an abstract machine that uses device drivers targeting the abstract machine, while a container provides an abstract OS. A para-virtualized VM environment provides an abstract hardware abstraction layer (HAL) that requires HAL-specific device drivers. Applications running in a container environment share an underlying operating system, while VM systems can run different operating systems. Typically a VM will host multiple applications whose mix may change over time versus a container that will normally have a single application. However, it's possible to have a fixed set of applications in a single container.

Virtual-machine technology is well-known in the embedded community, but containers tend to be the new kid on the block, so they warrant a bit more coverage in this article. Containers have been the rage on servers and the cloud, with companies like Facebook and Google investing heavily in container technology. For example, each Google Docs service gets a container per user instance.

A number of container technologies are available, with Linux leading the charge. One of the more popular platforms is Docker, which is now based on Linux libcontainer. Actually, Docker is a management system that's used to create, manage, and monitor Linux containers. Ansible is another container-management system favored by Red Hat.

Microsoft is a late arrival to the container approach, but its Windows Containers is a way to provide container services on a Windows platform. Of course, it's possible to host a Linux container service as a VM on Microsoft server platforms like Hyper-V. Container-management systems like Docker and Ansible can manage Windows-based servers providing container support.

Based-File Systems, Virtual Containers and Thin VMs

Containers provide a number of advantages over VMs, although some can be addressed using other techniques. One advantage is the low overhead of containers and, therefore, the ability to start new containers quickly. This is because starting the underlying OS in a VM takes time, memory, and the space needed for the VM disk storage. It may be difficult to address the time issue, but the other two can be addressed.

The easiest is the VM disk storage. Normally, a VM needs at least one unique image file for every running instance of a VM. It contains the OS and often the application code and data as well. Much of this is common among similar VMs. In the case of a raw image, a complete copy of the file is needed for each instance. This could require copying multiple gigabytes per instance.

e alternative is to use a based-file format like QEMU's qcow2, which is supported by Linux's KVM virtual-chine manager. In this case, an initial instance of the VM is set up and the operating system is installed possibly with additional applications. The VM is then terminated and the resulting file is used as the base for subsequent qcow2 files.

Setting up one of these subsequent files takes minimal time and space. It can then be used by a new VM, where changes made to the disk are recorded in the new file. Typically, the based file will contain information that will not change in the new file, although doing something like updating the operating system may cause the new file to grow significantly. This masks the original file to the point where the original will not be referenced, since all of its data has been overwritten.

The chain of based files can continue so that there may be a starting image with just the operating system. The next in the chain may add services like a database application. Another might add a web server. Starting up a new instance of a database server would build a new file starting from the image with the database in it, while a web server with database server would start from the database/web-server file.

The use of based files addressed duplication of file storage. For memory deduplication, we need to turn to the hypervisor. Some hypervisors can determine when particular memory blocks are duplicates, such as the underlying OS code assuming two or more VMs use the same OS with the exact same code. This approach can significantly reduce the amount of memory required, depending on the size of shared code or data that can be identified.

An issue with containers is the requirement that the underlying OS be the same for all containers being supported. This is often a happy occurrence for embedded systems in which applications can be planned to use the same OS. Of course, this isn't always the case; this can even be an issue in the cloud. The answer is to run the container system in its own VM. In fact, the management tools can handle this, because a collection of services/containers will often be designed to run on a common container platform.

Finally, there's the idea of thin VMs. These VMs have a minimal OS and run a single application. Many times, the OS forwards most of the service requests, such as file access, to a network server. Stripped-down versions of standard operating systems like Linux are substantially smaller. In the extreme case, the OS support is actually linked into the application so that the VM is just running a single program. For embedded applications, the network communication may be done via shared memory, providing a quick way to communicate with other VMs on the same system.

No one approach addresses all embedded applications, and there may be more than one reasonable alternative to deploying multiple program instances. It will be more critical to consider the alternatives when designing a system as the world moves from single-core platforms to ones with many, many cores.

Looking for parts? Go to SourceESB.

Source URL: http://electronicdesign.com/dev-tools/what-s-difference-between-containers-and-virtual-machines