Electronic Design

WILLIAM WONG | Technology Editor

VM, Containers, and Serverless Programming for Embedded Developers

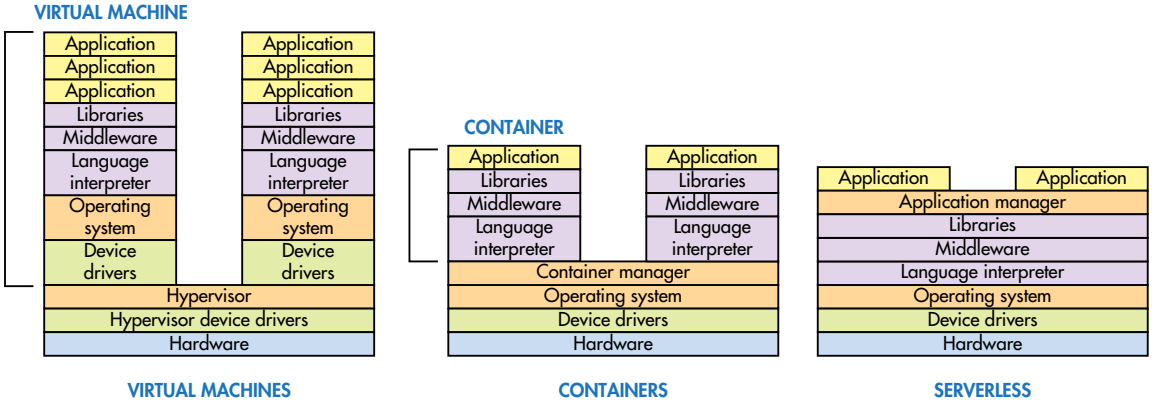
Enterprise computing has delivered virtual machines, containers and now serverless programming. Find out where it fits for embedded developers.

Having already provided developers with virtual machines (VM) and containers, enterprise computing has now rolled out serverless programming *(Fig. 1)*. This migration takes advantage of the growth in cores and processors within the cloud and enterprise systems. The number of cores involved is staggering compared to a typical embedded system that employs a virtual memory operating system like Linux. I mention this to partition out lower-end embedded systems that may lack even a memory management unit.

Enterprise computing tends to be way ahead of embedded development technology for a variety of reasons, including the need to push the envelope in terms of size, performance, management, and capacity. Embedded systems rarely push the same limits, and designs tend to be more conservative since they often need to run for years or decades. Still, many of the technologies that have been refined at the enterprise level are finding their way into embedded systems development depending upon how applicable and how robust the solutions are. VMs are one example where the technology has been commonplace in the enterprise and is becoming more common in embedded systems.

One reason for the enterprise focus application modularity is the way services can be sold to developers and companies as more computing gets moved to the public or private cloud. This has led to a range of systems include System as a Service (SaaS), Platform as a Service (PaaS), Infrastructure as a Service (IaaS), Containers as a Service (CaaS) and now FaaS (Function as a Service). FaaS is also known as serverless computing. It isn't really serverless, but from a user/developer perspective there is no server to manage.

IaaS is implemented using VMs. The others can be implemented within a VM but the underlying software is hidden from application developers. VMs take advantage of VM hardware that lets a VM to think it has a complete machine to itself. It does mean that an operating system is



1. Virtual machines, containers, and serverless programming run applications, but how they are supported differs.Virtual machines incorporate everything including the operating system. Containers provide access to a common operating system while serverless systems provide a standard application interface.

part of the package although it is possible to do a bare-metal application that is essentially its own operating system with built-in device drivers.

A hypervisor manages multiple VMs and typically the hypervisor can support many different operating systems. For example, Linux's KVM and Microsoft's Hyper-V can run a range of operating systems from Solaris to BSD. Of course, Linux, Microsoft Windows, and Microsoft Server are in the mix as well.

1. Virtual Machines

VMs can be isolated by the hypervisor providing a hardware-based security environment. Communication between the VMs can be achieved by using a network or virtual network interface, but some hypervisors allow shared peripherals and shared memory.

The downside to VMs is their size and complexity. They need to be large to include all the components that multiple applications will need. The advantage is that multiple instances of systems running on computers can be moved to a single computer running multiple VMs. The multicore processors make this practical and help provide a scalable environment, since high-end CPU core performance has essentially flattened.

It is possible to run VMs that use more cores and memory than the underlying hardware provides. Cores are time shared and swapped disk storage is used to provide more memory. This overprovisioning is practical for some systems, but it can significantly slow down systems that would regularly use more cores and memory than are actually available.

A number of techniques are used to reduce overhead in running and managing VMs. For example, file formats like QCOW2 can be configured in a hierarchical fashion *(Fig. 2)*, where top-level files contain a VM but are referenced by other files that are used for individual VMs. These files only contain changes for a running an instance of the VM. Starting a new VM is a matter of creating a new QCOW2 file based on an existing file.

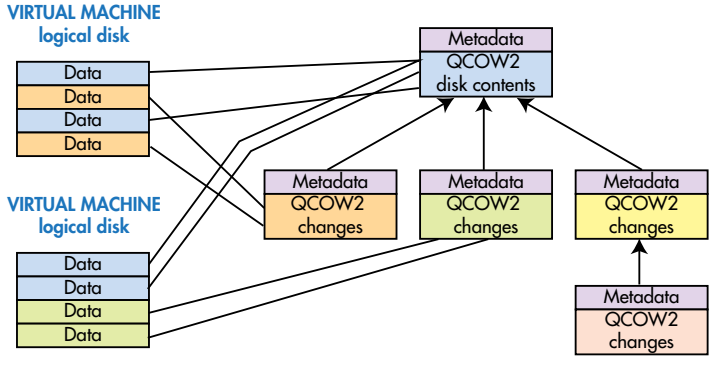
A typical creation process would be installing an operating system on the root file as the VM boot disk. New VMs can be based on this VM disk. In the next step, a new QCOW2 file is set up for a new VM where a database server is installed. The process is repeated to create a VM database that will be used to store data. The upper level files will be read-only, and a running VM will use that data if it has not been changed. This is usually space-efficient since most data does not change. For example, program code would not change unless software is upgraded. Creating new VMs is fast because they are based on existing files and the new files only contain a small amount of metadata.

VM hypervisors sometimes perform similar optimizations by tracking the memory blocks that are mirrored in a virtual disk. In many cases it is possible to determine that a read-only memory block from a virtual disk is the same for two or more active VMs. In this case, a single memory block can be shared among the VMs in a read-only fashion. The memory management system can provide this protection. A hypervisor can support changes to these blocks if it can remap the block or if one of the VMs makes a change. The remaining VMs can still share the original block, but the other VM will need to have its own block. This support will be transparent to the VMs.

2. Containers

Container systems, like Docker, require virtual memory hardware but not virtual machine support to host multiple applications. A Docker instance can run in a VM, but it can also run alone. It runs on top of an operating system like Linux and can run one or more containers.

Containers use services provided by the underlying container manager that in turn gets its support from the host operating system. Containers are isolated from each other and the underlying system using security features like Linux chroot, control groups (cgroups), and namespaces. These are typically implemented using the operating system's virtual memory system and security support. The security features tend to be more varied than a VM, where systems tend to be isolated by default and resources include memory and peripherals.



2. The root and upper level QCOW2 files are read-only. The leaf files can be used with a single virtual machine that can make changes that will be reflected in the leaf file.

Containers do have an advantage because they can often take advantage of the container system's security support, which is based on the operating system's security. For example, this allows a Linux-based container system to employ security systems like TOMOYO, AppArmor, SELinux, and GRSEC. These have advanced policy-based features that are typically not found in a VM environment.

Containers have other advantages over VMs besides just a more customizable security

environment. There is more commonality among containers, since they are all based on the same framework that is built on a common operating system. Developers do not have to manage or worry about the underlying operating system from a container perspective. The containers include less code and data and tend to be easier to configure because there is no need to address operating system functionality.

On the downside, containers are specific to the operating and container environment. It is possible to migrate between platforms in the same way that it is possible to migrate VM virtual disk files between different host VMs, but both of these tend to be exceptions associated with moving existing applications to a new host environment.

A server can typically support many more containers than VMs using the same resources because containers inherently have less overhead compared to applications running in a VM. They also tend to be easier to configure and upgrade since there are fewer items involved. For example, there is no operating system to worry about upgrading.

Not all container systems are created equal. Some allow containers to have a single application while others allow multiple applications to reside within a container. The latter would allow an SSH server to be included in a container for remote access to the container. Likewise, a single container could include a web server and a database server. The other approach would require two containers, one for the database server and one for the web server. There are advantages to both approaches including differences in configuration, upgradeability, support and so on.

Container management systems usually allow groups of containers to be configured and deployed. This is very necessary where containers are restricted to a single application. Container applications are linked via network connections so related applications can be on the same virtual network or on another server. Container migration between servers is normally available for load leveling and redundancy.

Containers tend to be run in a similar fashion as VMs. They tend to run for a long period of time and are normally started by a user when a system starts up, or in response to a limited number of events.

3. Serverless Computing

Serverless computing services like Amazon's AWS Lambda or platforms like the open-source Apache OpenWhisk take the idea of containers to their extreme. Serverless computing is based on functions that are essentially small applications often designed to run on demand for short periods of time. Some systems even limit the time and resources a function can utilize. Containers and VMs can also have limitations, but they tend to be higher since they incorporate more services.

One reasons for serverless computing is to provide a system that is easy to track for billing purposes. A serverless system may simply track the amount of time a function runs while limiting the resources it can use.

Another reason is to support event-driven workloads that often arise with the Internet of things (IoT). A function can be started on the reception of an IoT message. It is easy to scale the response based on demand with the underlying system handling this management instead of building it into the application.

Serverless functions can be written in languages like Javascript or Python, although it is possible to write them in almost any language, including C and C++. The approach does allow the functions to be written in a more generic fashion that is not dependent upon the underlying implementation or operating system.

Security in serverless environments is similar to containers in that the operating system and management environment provide mechanisms to allow the serverless functions to only access their provided resources. The potential for problems exists simply because of the number of functions normally involved in a solution and the need to implement policies properly.

Another issue with serverless solutions as well as containers is vendor or platform lock-in, since the application is now dependent on the underlying framework. This is less of an issue for embedded solutions that maintain their own platform but, in this IoT age, part of a solution may reside in the cloud on a public service like AWS Lambda.

System solutions may incorporate a range of platforms from VMs to containers to serverless functions. Some functionality may be more applicable to one approach over another.

Some enterprise solutions will be overkill for many embedded applications, but often solutions can be scaled to fit embedded needs. There also tend to be a wide range of options for all these approaches, and some may be more amenable to embedded developers.

At this point, VM support tends to be readily available and the same is true for container systems. It is possible to use the serverless approach in embedded applications. It may be a good fit for event-based applications, but developers will have to keep its advantages and limitations in mind when creating functions, since it tends to be a bit different than writing applications for VMs or containers.