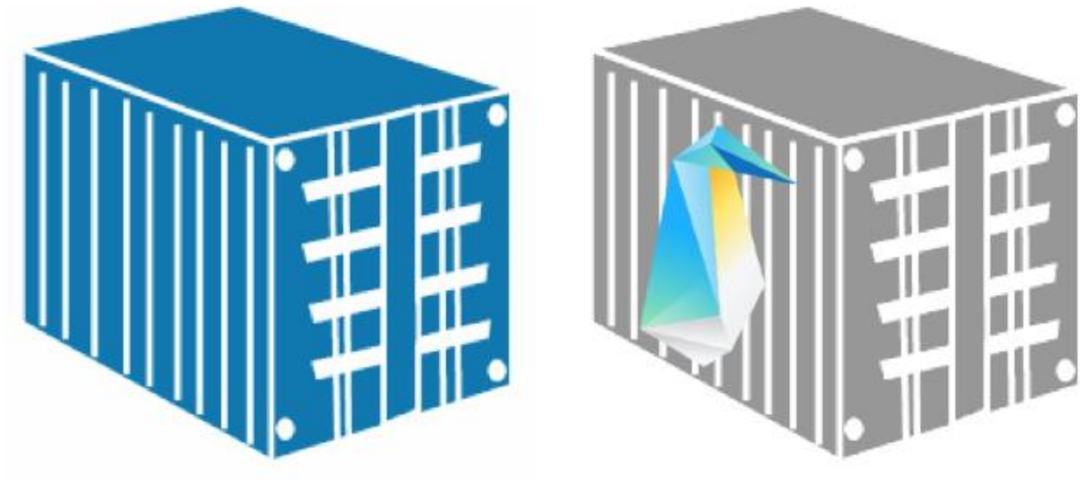
Intel® Clear Containers 1: The Container Landscape

Introduction



This article introduces the concept of Intel® Clear Containers technology and how it fits into the overall landscape of current container-based technologies. Many introductory articles have already been written about containers and virtual machines (VMs) over the last several years. Here's a good overview: A Beginner-Friendly Introduction To Containers, VMs, and Docker

This article briefly summarizes the most salient features of existing VMs and containers.

Intel Clear Containers offer advantages to data center managers and developers because of their security and shared memory efficiencies, while their performance remains sufficient for running containerized applications.

What's In A Word: Container

The word “container” gets thrown around a lot in this article and many others, and it's important to understand its meaning.

If you’re an application developer, you probably think of a container as some kind of useful application running inside a private chunk of a computer's resources. For example, you would discuss downloading a container image of a web server from Docker* Hub*, and then running that container on a host.

If you’re a data center manager, your perception of a container might be slightly different. You know that application developers are running all kinds of things in your data center, but you are more concerned with the composition and management of the bounded resources the applications run in than the applications themselves. You might discuss running “an LXC container using Docker” for the developer's web server. You would refer to the application image and application itself as the containerized workload.

This is a subtle, but important, difference. Since we'll be looking directly at container technology, independent of the workloads run using it, we'll use the data center manager's definition of container: the technology that creates an instance of bounded resources for a containerized workload to use.

Virtual Machines versus Containers

It is technically a misstatement to suggest that containers came after VMs. In fact, containers are more-or-less a form of virtualized resources themselves, and both technologies are descendants in a long line of hardware abstractions that stretch back to early computing.

However, in the marketplace of modern IT, it's relatively clear that the era from roughly the mid-2000s to around 2014 (or so) was dominated by the rise of paravirtual machine usage, both in the traditional data center and in cloud computing environments. The increasing power of servers, combined with advancements in hardware platforms friendly to VMs like Intel® Virtualization Technology for IA-32, Intel® 64 and Intel® Architecture and Intel® Virtualization Technology for Directed I/O allowed data center managers to more flexibly assign workloads.

Early concerns about VMs included performance and security. As the platforms grew more and more robust, these concerns became less and less relevant. Eventually commodity hypervisors were capable of delivering performance with less than 2–3 percent falloff from direct physical access. From a security standpoint, VMs became more isolated, allowing them to run in user space, meaning a single ill-behaved VM that became compromised did not automatically allow an attacker access outside the VM itself.

Within the last few years, containers began exploding upon the scene. A container, whether provided by Linux* Containers (LXC), libcontainer*, or other types, offers direct access to hardware, so there's no performance penalty. They can be instantiated far more quickly than a regular VM since they don't have to go through a bootup process. They don't require the heavyweight overhead of an entire OS installation to run. Most importantly, the powerful trio of a container, a container management system (Docker), and a robust library of containerized applications (DockerHub) gives application developers access to rapid deployment and scaling of their applications that could not be equaled by traditional VMs.

While offering these huge rewards, containers reintroduced a security problem: they represent direct access to the server hardware that underpins them. A compromised container allows an attacker the capability to escape to the rest of the OS beneath it.

[Note: We don't mean to imply that this access is automatic or easy. There are many steps to secure containers in the current market. However, a compromise of the container itself—NOT the containerized application—results in likely elevation to the kernel level.]

That leaves us with this admittedly highly generalized statement of pros and cons for VMs versus containers:

```
+=====================================+
|  Virtual Machine  |    Container    |
|-------------------------------------|
| -  Slow Boot      | + Rapid Start   |
| -  Heavy Mgmt.    | + Easy Mgmt.    |
| +  Security       | - Security      |
| =  Performance^   | = Performance   |
+=====================================+
```

```
^VMs take a negligible performance deficit due to hardware abstraction.
```

Best of Both Worlds: Intel Clear Containers

Intel has developed a new, open source method of launching containerized workloads called Intel Clear Containers. An Intel Clear Container, running on Intel architecture with Intel® Virtualization Technology enabled, is:

- A highly-customized version of the QEMU-KVM* hypervisor, called qemu-lite.
  - Most of the boot-time probes and early system setup associated with a full-fledged hypervisor are unnecessary and stripped away.
  - This reduces startup time to be on a par with a normal container process.
- A mini-OS that consists of:
  - A highly-optimized Linux kernel
  - An optimized version of *systemd*.
  - Just enough drivers and additional binaries to bring up an overlay filesystem, set up networking, and attach volumes.
- The correct tooling to bring up containerized workload images exactly as a normal container process would.

Intel Clear Containers can also be integrated with Docker 1.12, allowing the use of Docker just exactly as though operating normal OS containers via the native Docker execution engine. This drop-in is possible because the runtime is compatible with the Open Container Initiative* (OCI*). The important point is that from the application developer perspective, where "container" means the containerized workload, **an Intel Clear Container looks and behaves just like a "normal" OS container**.

There are some additional but less obvious benefits. Since the mini-OS uses a 4.0+ Linux kernel, it can take advantage of the "direct access" (DAX) feature of the kernel to replace what would be overhead associated with VM memory page cache management. The result is faster performance by the mini-OS kernel and a significant reduction in the memory footprint of the base OS and filesystem; only one copy needs to be resident in memory on a host that could be running thousands of containers.

In addition, Kernel Shared Memory (KSM) allows the containerized VMs to share memory securely for static information that is not already shared by DAX via a process of de-duplication. This results in an even more efficient memory profile. The upshot of these two combined technologies is that the system's memory gets used for the actual workloads, rather than redundant copies of the same OS and library data.

Given the entry of Intel Clear Containers onto the scene, we can expand the table from above:

Virtual Machine	Container	Intel® Clear Container
- Slow Boot	+ Rapid Start	+ Rapid Start^
- Heavy Mgmt.	+ Easy Mgmt.	+ Easy Mgmt.
+ Security	- Security	+ Security
= Performance^	= Performance	= Performance^

^VMs take a negligible performance deficit due to hardware abstraction.

Conclusion

Intel Clear Containers offer a means of combining the best features of VMs with the power and flexibility that containers bring to application developers.

You can find more information about Intel Clear Containers at the official website here: https://clearlinux.org/features/intel%C2%AE-clear-containers

This is the first in a series of blog posts about Intel Clear Containers. In the second, we'll be demonstrating how to get started using Intel Clear Containers yourself.

About the Author

Jim Chamings is a Sr. Software Engineer at Intel Corporation, who focuses on enabling cloud technology for Intel's Developer Relations Division. Before that he worked in Intel's Open Source Technology Center (OTC), on both Intel Clear Containers and the Clear Linux for Intel Architecture Project. He'd be happy to hear from you about this article at: jim.chamings@intel.com.