Linux* Containers Streamline Virtualization and Complement Hypervisor-Based Virtual Machines

As strategic approaches such as software-defined networking (SDN) and network function virtualization (NFV) become more mainstream, combinations of both OS-level virtualization and hypervisor-based virtual machines (VMs) often provide dramatic cost, agility, and performance benefits, throughout the data center.

1 Executive Summary

By decoupling physical computing resources from the software that executes on them, virtualization has dramatically improved resource utilization, enabled server consolidation, and allowed workloads to migrate among physical hosts for usages such as load balancing and failover. The following key techniques for resource sharing and partitioning have been under continual development and refinement since work began on them in the 1960s:

- **Hypervisor-based virtualization** has been the main focus within the Intel® architecture ecosystem. Increased server headroom and hardware assists from Intel® Virtualization Technology (Intel® VT)[1] have helped broaden its adoption and the scope of workloads for which it is effective on open-standard hardware.
- **OS-level virtualization** has been developed largely by makers of proprietary operating systems and system architectures. Linux* containers, introduced in 2007, extend the capabilities for highly elastic or latency-sensitive workloads on open-standards hardware.

Both approaches have distinct advantages. For example, whereas each hypervisor-based virtual machine (VM) runs an entire copy of the OS, multiple containers share one kernel instance, significantly reducing overhead. While hypervisor-based VMs can be provisioned far more quickly than physical servers—taking only tens of seconds as opposed to several weeks—that lag is unacceptable for workloads that range from real-time data analytics to control systems for autonomous vehicles.

On the other hand, hypervisor-based VMs allow for multi-OS environments and superior workload isolation (even in public clouds). Enterprises are also finding that new virtualization usage models that draw on both containers and hypervisors help them get more value out of strategic approaches such as public cloud, software-defined networking (SDN), and network function virtualization (NFV).

As virtualization using Linux* containers emerges as a viable option for mainstream computing environments, Intel is investing in enablement through hardware and software technologies that include the following:

- **The Intel® Data Plane Development Kit (Intel® DPDK)**
- **Intel® Solid State Drive Data Center Family for PCI Express***
- **Intel® Virtualization Technology**

NOTE: In the context of this paper, the term "container" is a generalized reference for any virtual partition other than a hypervisor-based VM (e.g., a chroot jail, FreeBSD jail, Solaris* container/zone, or Linux container).

Table of Contents

1 Executive Summary 1
2 The Historical Context for Containers and Hypervisors 2
2.1 Resource-Partitioning Techniques Not Based on Hypervisors 3
2.2 Emergence of Software-Only, Hypervisor-Based Virtualization for Intel® Architecture 4
2.3 Hardware-Assisted, Hypervisor-Based Virtualization with Intel® Virtualization Technology 4
3 Virtualization Using Linux Containers 5
3.1 Usage Model: Platform as a Service on Software-Defined Infrastructure 6
3.2 Usage Model: Combined Hypervisor-Based VMs and Containers in the Public Cloud . . 7
4 Enabling Technologies from Intel for Container-Based Virtualization 8
4.1 Intel® Data Plane Development Kit 8
4.2 Intel® Solid-State Drive Data Center Family for PCI Express* 9
4.3 Enhanced Workload Isolation with Intel® Virtualization Technology 9
5 Containers in Real-World Implementations 10
5.1 Google: Combinations and Layers of Complementary Containers and Hypervisors 10
5.2 Heroku and Salesforce.com: Container-Based CRM, Worldwide 11
6 Conclusion 12

This paper introduces technical executives, architects, and engineers to the potential value of Linux containers, particularly in conjunction with hypervisor-based virtualization; it includes the following sections:

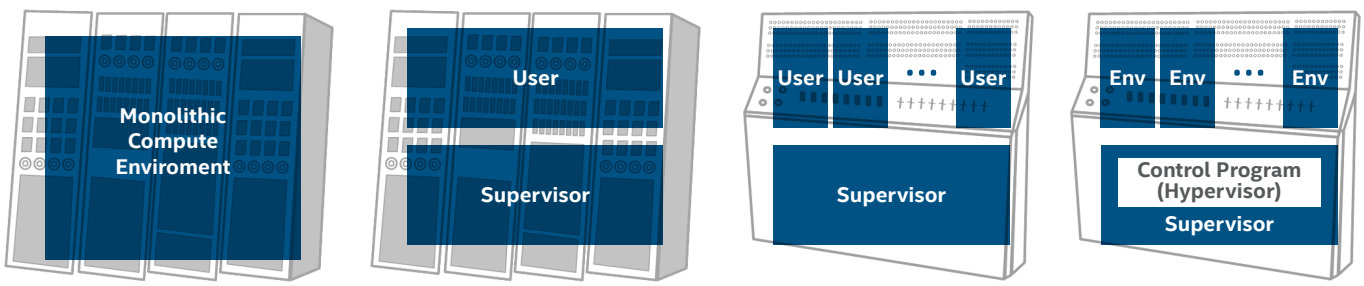
- **The Historical Context for Containers and Hypervisors** traces the development of these technologies from their common origin as an effort to share and partition system resources among workloads.
- **Virtualization Using Linux Containers** introduces the support for container-based virtualization in Linux, illustrated by representative usage models.
- **Enabling Technologies from Intel for Container-Based Virtualization** discusses Intel's hardware and software technologies that help improve performance and data protection when using Linux containers.
- **Containers in Real-World Implementations** examines a few examples of early adopters that are using containers as part of their virtualization infrastructures today.

2 The Historical Context for Containers and Hypervisors

Hypervisors and containers have a joint ancestry, represented in Figure 1, which has been driven by efforts to decrease the prescriptive coupling between software workloads and the hardware they run on. From the beginning, these advances enhanced the flexibility of computing environments, enabling a single system to handle a broader range of tasks. As a result, two related aspects of the interactions between hardware and software have evolved:

- **Resource sharing** among workloads allows greater efficiency compared to the use of dedicated, single-purpose equipment.
- **Resource partitioning** ensures that the system requirements of each workload are met and prevents unwanted interactions among workloads (isolating sensitive data, for example).

The first stage depicted in Figure 1 is that of the monolithic compute environment, where the hardware and software are logically conjoined, in a single-purpose architecture. The ability to make significant changes to either



System Example: GE-635/Multics (circa 1959)

System Example: GE-635/Multics (circa 1962)

System Example: IBM 360 (circa 1964)

System Example: IBM 370 (circa 1972)

Figure 1. Early development of approaches to resource sharing and partitioning.

hardware or software, short of replacing the entire system, is severely limited or even nonexistent. The second stage includes the supervisor (progenitor of the kernel), an intermediary between user space and system resources that allows hardware and software to be changed independently of each other. That de-coupling is advanced further in the third stage, where the supervisor mediates resources among multiple users, allowing time-sharing among multiple, simultaneous jobs.

The fourth stage adds a control program (CP, also known as the "hypervisor") that presents an independent view of the complete environment to each application; applications execute in isolation, unaware of each other. Importantly, various hypervisor-enabled virtual environments can run different operating systems simultaneously on the same hardware, which would be impossible under most approaches to virtualization that are not based on hypervisors. (Running workloads based on different operating systems on the same host is possible to a limited extent using Solaris* containers, as described below.)

The supervisor, as mediator between user space and system resources, has full access to the host and allows a subset of that access to users (workloads). That arrangement allows the supervisor to enforce limitations on access that are necessary for usages such as the time sharing among jobs described in the third stage of Figure 1. In contrast, the CP shown in the fourth stage moderates between the supervisor and the multiple virtual representations of the environment presented to workloads. Accordingly, the CP operates at a level superior to that of the supervisor, conceptually suggesting a role "beyond the supervisor," which is reflected in the term "hypervisor."

IBM introduced its first hypervisor for production (but unsupported) use in 1967, to run on System/360* mainframes, followed by a fully supported version for System/370* equipment in 1972. The company is generally recognized as having held the industry's most prominent role in the development of hypervisor-based virtualization from that point through the rest of the twentieth century.

2.1 Resource-Partitioning Techniques Not Based on Hypervisors

Although hypervisors enrich isolation among virtual environments running on a single physical host, they also add complexity and consume resources. Alternate approaches to partitioning resources were developed as interest in Unix* and Unix-like operating systems grew in the late 1970s and early 1980s, driven by organizations that included Bell Laboratories, AT&T, Digital Equipment Corporation, Sun Microsystems, and academic institutions. These technologies (and others), which are collectively referred to as "OS-level virtualization," provide lightweight approaches to virtualization, acting as foundations for today's Linux containers.

- **The chroot system call** is a critical foundation for container-based virtualization, introduced as a feature of Version 7 Unix by Bell Laboratories in 1979 and as part of 4.2BSD by the University of California, Berkeley in 1983. The chroot mechanism can redefine the root directory for any running program, effectively preventing that program from being able to name or access resources outside that root directory tree. While such partitions, referred to as "chroot jails," support limited virtualization functionality, they must share a single OS kernel. Moreover, chroot is not designed to be tamper-resistant, and it is vulnerable to intentional efforts by users or programs to "break out" of their jails and gain unauthorized access to resources.
- **FreeBSD* jails** are similar in concept to chroot jails, but with a greater emphasis on security. This mechanism was introduced as a feature of FreeBSD 4.0 in 2000. FreeBSD jail definitions can explicitly restrict access outside the sandboxed environment by entities such as files, processes, and user accounts (including accounts created by the jail definition specifically for that purpose). While this approach significantly enhances control over resources compared to chroot, it is likewise incapable of supporting full virtualization, because the FreeBSD jails must share a single OS kernel.
- **Solaris containers (Solaris zones)** build on the virtualization capabilities of chroot and FreeBSD jails. Sun Microsystems introduced this feature with the name "Solaris containers" as part of Solaris 10 in 2005, and Oracle officially changed the name to "Solaris zones" with the release of Solaris 11 in 2011. Zones are individual virtual server instances that can coexist within a single OS instance. Similar to FreeBSD jails, Solaris zones allow for zone-specific user accounts and access restrictions on resources such as network interfaces, memory, storage, and processors. One significant advance toward full virtualization is that while Solaris zones must share a single OS kernel, a capability called "branded zones" (BrandZ) enables individual zones to emulate the behavior of certain operating systems. As mentioned previously, BrandZ allows the environment to simulate cross-OS virtualization to a limited degree.

2.2 Emergence of Software-Only, Hypervisor-Based Virtualization for Intel® Architecture

Shortly after the year 2000, Intel architecture was beginning to reach the upper limits of client-server performance possible from scaling up processor frequency alone. The key factors behind that plateau were the increased power consumption and waste heat associated with raising clock speed. The primary solution that emerged was to create processor architectures with multiple execution cores on a single die.

Multi-core processors can execute multiple threads of programming instructions in parallel, simultaneously and independently of each other. While this approach increases scalability, redesigning serial applications for multithreading—so that work can be split efficiently among multiple cores—is a complex undertaking. Moreover, if done incorrectly, it can produce conflicts among threads that cause runtime errors, unpredictable results, or performance deficits. Other approaches to take advantage of hardware parallelism include the following:

- **Stateless web pages** can support many tasks running in parallel, such as user sessions or transactions.
- **High-performance computing models**, including clusters and grids, coordinate workloads across machines.
- **Hypervisor-based virtualization** allows serial applications to run in separate virtual machines (VMs).
- **Non-hypervisor virtualization** provides resource sharing and partitioning among workloads using mechanisms such as chroot, FreeBSD jails, and Solaris containers.

Of the models described above, hypervisor-based virtualization emerged as the dominant data-center approach, marked by a period of rapid hypervisor development in the Intel architecture ecosystem, led by VMware and the open-source Xen* project. Other prominent providers of virtualization solutions include Citrix (including commercial versions of Xen), Microsoft, Oracle, Parallels, Red Hat, and the Kernel-based Virtual Machine (KVM) open-source project. A high-level representation of virtualization stacks is shown in Figure 2, which depicts the two primary hypervisor variations: Type 1 and Type 2.

Type 1 hypervisors are installed directly on the physical host hardware, whereas Type 2 (also known as "hosted") hypervisors are installed on top of a host OS. Type 1 hypervisors communicate directly with the underlying hardware and support robust virtual networking and dynamic resource allocation through a management server; they are the more prevalent approach for data-center virtualization of servers. Type 2 hypervisors typically rely on the host OS (rather than a management server) for services such as I/O device support and memory management; they are often used for virtualized desktop infrastructure.

2.3 Hardware-Assisted, Hypervisor-Based Virtualization with Intel® Virtualization Technology

In software-only virtualization on Intel architecture, the hypervisor must emulate the hardware environment for VMs, using binary translation. This service provides software-based interfaces to physical resources such as processors, memory, storage, graphics cards, and network adapters. Because this emulation is performed in software, it consumes significant execution resources on the processor that are therefore not available to operate on workloads, which significantly limits scalability.

Intel introduced Intel VT in 2006, which includes a new privilege level for VMs to operate in without the overhead of binary translation, dramatically improving efficiency on the physical host. Intel VT has continued to develop

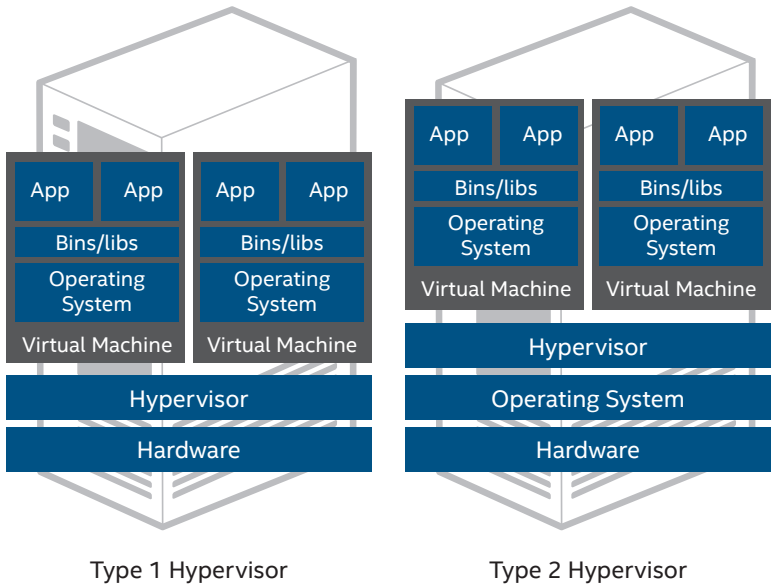


Figure 2. Virtualization based on Type 1 and Type 2 hypervisors.

into an expanding set of features within processors, chipsets, and network hardware to make virtualization increasingly efficient and to enhance the hardware-based isolation between workloads on separate VMs running on the same host.

Higher performance, improved data protection, and decreased latency enabled by Intel VT allowed larger numbers of servers to be consolidated per physical host and expanded the types of workloads suitable for virtualization. At the same time, progressive generations of Intel® Xeon® processors for the highly scalable server segment were engineered with more advanced reliability, availability, and serviceability (RAS) features, making them more suitable for virtualizing mission-critical workloads.

3 Virtualization Using Linux Containers

As discussed above, the requirement under hypervisor-based virtualization for a separate OS instance to run in each VM consumes significant resources. For example, multiple copies of many drivers and services must be run, including some that aren't even needed by any of the running VMs.

Furthermore, compute capacity can be readily increased in virtualized environments by starting additional VMs, but spinning up a new VM (or shutting one down) has time requirements on the order of tens of seconds. Workloads that can't tolerate that level of delay are becoming more prevalent—notably those with elastic capacity needs and real-time or otherwise low-latency requirements, such as the following:

- **Real-time data analytics**, for systems providing services such as business intelligence that rely on immediate access to data as it emerges from dispersed sensors and other sources
- **Remote interaction among users**, including real-time usages such as voice or video communication, online collaboration, and remote control of automated systems, where lag or jitter is unacceptable
- **Latency-sensitive or lossless networking**, as required for systems such as network-attached storage or Ethernet-based storage area networks
- **Page rendering for web content** such as Facebook's News Feed and LinkedIn's Home/Feed, which involves gathering data in parallel from many sources

In contrast to a VM, a container does not require a hypervisor or incorporate a dedicated OS, which presents an attractive approach for workloads such as those described above. The open-source Linux Containers (LXC) project introduced public code in 2007 that provides support for multiple containers on the same physical host (or on the same VM, as discussed below), sharing the same instance of the Linux OS kernel (and in some cases a set of binaries and libraries), as shown in Figure 3. Each container includes only the services that it needs but can't obtain from those shared resources, paring down the size of the software stack running on each container.

Two features of the Linux kernel are at the core of the functionality that underlies Linux containers:

- **Cgroups** provides the ability to govern and measure the use of resources such as memory, processor usage, and disk I/O by collections of processes called "control groups." The kernel provides access to separate subsystems for managing each set of resources, either manually or programmatically.
- **Namespace isolation** provides a software-based means of limiting each control group's view of global resources, such as details about file systems, processes, network interfaces, inter-process communication, host and domain names, and user IDs.

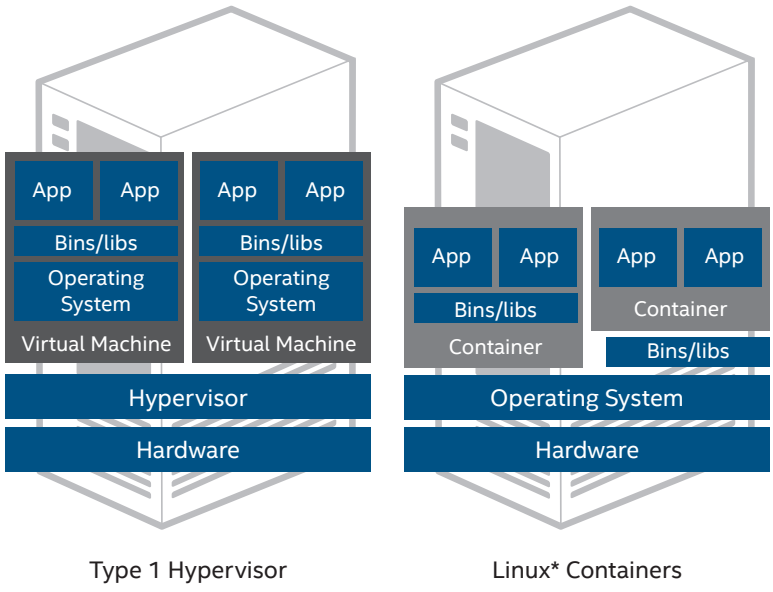


Figure 3. Linux containers compared to a Type 1 hypervisor.

Using these capabilities, the containers themselves are created as sandboxed environments for applications to execute in. Because the containers are unaware of each other, each of them interacts with resources such as the shared OS kernel and file system as if the other containers didn't exist. That isolation extends to the applications within the containers, their security settings, and their access to physical and virtual devices through the host. As discussed later in this paper, however, software-only approaches to isolating workloads are potentially more susceptible to compromise by malicious software than those that incorporate hardware-based mechanisms.

At the present early stage, it would be difficult to anticipate the roles that virtualization based on hypervisors versus virtualization based on containers will eventually take in mainstream data centers. Still, it is apparent that both models will persist for the foreseeable future, because of their complementary nature. For example, the requirement that all containers must use the same Linux OS kernel is an obvious limitation to server consolidation in a typical enterprise that is likely to also include Microsoft Windows*. Desktop virtualization is another common example; organizations that want to run virtual Windows desktops will need to do so using a hypervisor. Moreover, hypervisors and containers can each play a valuable role as the installation substrate for the other, as in the usage models described in the remainder of this paper.

The Container-Based Virtualization Software Ecosystem for Linux*

The following list captures a few noteworthy software offerings related to the use of containers:

- **LXC (Linux Containers)** is a user-space interface (including an API and tools) for the cgroups and namespace-isolation features of the Linux kernel.
- **Docker** is an open-source engine designed to simplify the use of LXC by automating deployment of applications in containers, packaging applications and their dependencies in images with a standard format.
- **Lmctfy (Let Me Contain That For You)** is an open-source version of Google's internal application containerization stack that, in contrast to Docker, prioritizes performance over ease of use.
- **CoreOS** is a fork of Chrome OS*, designed to include only the minimum level of capabilities that are needed to support the operation of applications in containers.
- **Project Atomic** is a community-based effort initiated by Red Hat to create technologies for lightweight container hosts; this work will be used in a new product variant, Red Hat Enterprise Linux Atomic Host.
- **Kubernetes** is an open-source container manager developed by Google that provides deployment, health management, replication, and connectivity services for containers.

NOTE: Lmctfy, CoreOS, and Project Atomic use the Linux kernel's cgroup and namespace-isolation features directly, as opposed to using LXC, although Docker is integrated into all three.

3.1 Usage Model: Platform as a Service on Software-Defined Infrastructure

The container model of virtualization is a good fit with the rack-scale architecture (RSA) approach favored in Intel's vision for the re-imagined data center. RSA looks ahead to the need for dramatically increased levels of server resources in data centers to support developments such as the Internet of Things, which is expected to add billions of connected devices to the global infrastructure in the next decade. To support this vast internetwork, server architectures must allow resources to shift around dynamically as workloads change and adapt.

The RSA approach to meeting that challenge is to pool the processing, memory, networking, and storage resources of an entire server rack, disaggregating them from the host-level units to which we are accustomed. Physically, these components need not even be arranged in the form of servers; for instance, a rack could be provisioned using a chassis for each type of component and the hardware dependencies needed to support it. Logically, the rack replaces the individual system as the unit of management in the data center, with one rack's worth of individual resources referred to as a "tray." That is, one rack as a logical unit consists of a processor tray, a memory tray, etc.

The attributes of the hardware resources are exposed to the software layer, which provision virtual systems on a dynamic, as-needed basis, according to the requirements of a

specific workload or task. RSA offers tremendous flexibility by tailoring software-defined infrastructure to the needs of applications and workloads in real time. This approach optimizes efficiency by dedicating only the level of resources needed for the task at hand, without the need to reserve idle headroom, because the virtual system will be disbanded and the resources returned to the rack-level pool before system demands could shift. It also offers a means of building a system that is far more reliable than any of its components, because each of them is inherently redundant and interchangeable. Overall, RSA extends data-center capabilities and drives up asset utilization, while reducing overall costs.

Container-based virtualization has obvious utility within this approach, because it enables extremely rapid provisioning and de-provisioning of resources, compared to hypervisor-based virtualization. Here, the container environment is scaled horizontally using a platform-as-a-service (PaaS) model, with a cluster-wide resource manager running as a service on all the nodes, coordinated by a central executive scheduler. Several such resource managers integrate with Intel's RSA model, including the following:

- **SLURM** (Simple Linux Utility for Resource Management), an open-source project
- **TORQUE** (Terascale Open-source Resource and QUEue Manager), a community-based project
- **Omega**, created by Google
- **Mesos**, an Apache Software Foundation open-source project
- **YARN** (Yet Another Resource Negotiator), an Apache Software Foundation open-source project for Hadoop*

Usages that particularly benefit from the use of containers on PaaS architectures are those cases where long-running data-services applications with relatively higher priority can run in a multitenant fashion with fast-completion, lower-priority processing tasks such as MapReduce* or Apache Spark*. Implementation of this model is accelerating in segments such as high-performance computing, cloud and big data, telecommunications, the Internet of Things, and financial services (particularly to support high-frequency trading).

3.2 Usage Model: Combined Hypervisor-Based VMs and Containers in the Public Cloud

As they plan how to utilize the cost-effective, on-demand infrastructure offered by public clouds, many organizations see potential efficiencies from deploying workloads to these clouds in containers rather than hypervisor-based VMs. The resulting reduced compute overhead could significantly decrease the amount of resources they would be billed for, reducing operational expenses. At the same time, however, using containers may not meet organizations' needs in terms of isolating data among their own workloads or from other unknown organizations in the shared multitenant environment. In this scenario, both containers and hypervisor-based VMs have unique advantages:

- **Hypervisor-based workload isolation**. The ability to isolate data on a per-VM basis, even within a shared multitenant infrastructure, is well established, including partitioning shared physical memory and I/O devices. Doing so is critical to data protection particularly in cases of sensitive or regulated workloads.
- **Container-based resource efficiency**. Sharing the OS and some libraries and binaries among containers can dramatically reduce virtualization overhead, increasing provisioning speed, performance, and the number of client endpoints that can be supported with a given level of computing resources. That efficiency translates into cost savings for the customer.

The benefits of both approaches can be combined to a significant extent by the model where a customer provisions a set of VMs on a public cloud and then partitions each of those VMs using containers, as shown in Figure 4. Instances of applications deployed in those containers could achieve a valuable balance between data protection and efficiency, particularly

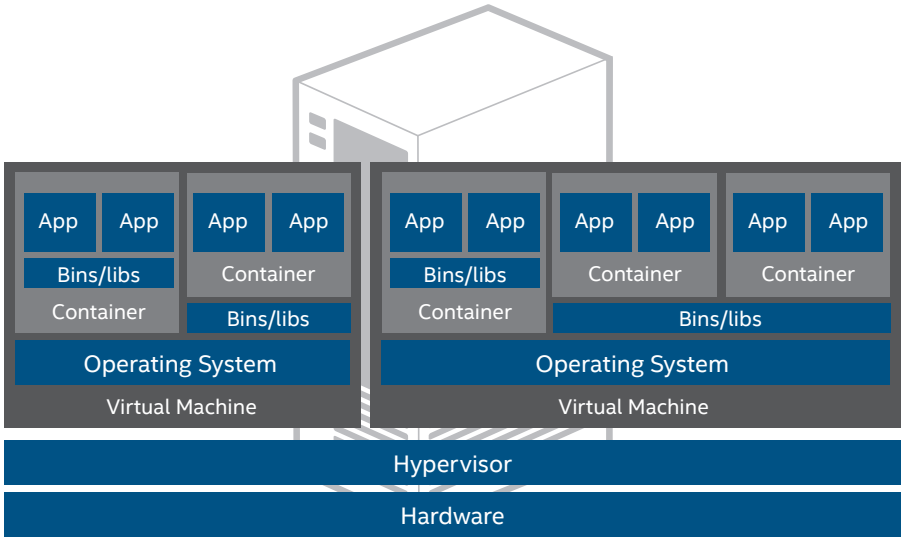


Figure 4. Containers inside hypervisor-based virtual machines.

by using large numbers of containers per VM and a stripped-down OS such as CoreOS, which is built explicitly for containers. At the same time, customers can take advantage of the global reach, cost-effectiveness, and elasticity of PaaS environments offered by global providers such as Amazon and Google.

Moreover, that PaaS environment could use a hybrid cloud model, meaning that it could include containers running on VMs within both internal data centers and public clouds. A virtual network with an umbrella DNS namespace could encompass the entire multi-site environment, providing a single, global service that is resolved using DNS and routing techniques. Hybrid cloud infrastructure can help deliver the benefits of public cloud services while also taking full advantage of existing internal infrastructure assets. Running containers on the VMs within that hybrid cloud provides the rapid provisioning and efficiency benefits of containers, as well.

Deploying combinations of containers and hypervisor-based VMs as described above can help organizations take advantage of public clouds to meet a variety of business needs. For example, many companies—even large multinationals—have consolidated their data-center infrastructures into a small number of physical facilities. While that strategy enables them to achieve cost savings, it also potentially extends the physical distance (and accordingly, the number of intermediate router connections, or "hops") to some endpoints, thereby increasing the latency over the connections to those endpoints. In response, as the number and types of connected devices continues to expand, organizations are assessing the potential for containers in conjunction with hypervisor-based virtualization on leased public clouds.

That approach is particularly germane to supporting real-time or otherwise latency-sensitive workloads based on global networks of mobile devices. One example might arise in an organization fielding a global network of edge devices such as point-of-sale (POS) systems, customer kiosks, smart vending machines, or sensor arrays. Exchanging data between such endpoints and core back-end systems may be a key requirement for real-time data analytics, which can create business value by guiding tactical and strategic decision making in areas such as demand forecasting, dynamic price optimization, and supply-chain management.

Many such implementations could need low-latency connectivity in regions that are geographically distant from any of the company's owned data centers. Likewise, an organization might have highly elastic capacity requirements for certain applications and workloads, separately or in conjunction with the global-endpoint example given above. Workload characteristics that could create such elastic demand might include holiday shopping causing a spike in activity on retail POS systems, seasonal variations in travel volume impacting usage of airline kiosks at airports all over the world, and hot weather or a major sporting event producing a temporary rise in sales from connected vending machines.

4 Enabling Technologies from Intel for Container-Based Virtualization

In keeping with its strategic vision for RSA and software-defined infrastructure, Intel has invested in a number of technologies that will help optimize performance and data protection within container environments.

4.1 Intel® Data Plane Development Kit

To increase agility and reduce costs, the industry is moving toward solutions that decouple network functions from the underlying hardware by means of abstraction. This transition encompasses the interrelated efforts of software-defined networking (SDN) and network function virtualization (NFV).

The Intel® Data Plane Development Kit (Intel® DPDK) is a set of software libraries that support the hardware abstraction required by SDN and NFV, as well as dramatically accelerating packet processing. Those capabilities enable application, control, and signal-processing workloads to be transitioned away from special-purpose hardware, to standards-based servers based on Intel® processors. With the reduction or elimination of network processor units (NPUs), coprocessors, application-specific integrated circuits (ASICs), and field-programmable gate arrays (FPGAs), the hardware environment becomes simpler, more cost-effective, and more scalable.

The more homogeneous, standardized architecture also better supports the dynamic definition of systems in software that underlies the RSA and software-defined infrastructure described earlier in this paper. Intel DPDK therefore can play a significant role in creating an optimal environment for the implementation of container-based virtualization. Key software-library components of Intel DPDK include the following:

- **Environment Abstraction Layer** provides access to low-level resources such as hardware, memory space, and logical cores using a generic interface that obscures the details of those resources from applications and libraries.

- **Memory Manager** allocates pools of objects in huge-page memory space. An alignment helper ensures that objects are padded, to spread them equally across DRAM channels.
- **Buffer Manager** significantly reduces the time the OS spends allocating and de-allocating buffers. The Intel DPDK pre-allocates fixed-size buffers, which are stored in memory pools.
- **Queue Manager** implements safe lockless queues (instead of using spinlocks), allowing different software components to process packets while avoiding unnecessary wait times.
- **Flow Classification** incorporates Intel® Streaming SIMD Extensions (Intel® SSE) to produce a hash based on tuple information, improving throughput by enabling packets to be placed quickly into processing flows.

Successful implementation of NFV depends to a large extent on robust virtual switching capabilities. Open vSwitch is a production-quality, multi-layer virtual switch distributed under the Apache 2.0 License. Intel is committed to working with the Open vSwitch community to extend it with support for Intel DPDK. This support is now included in the openvswitch.org source tree.

A priority for the integration of Open vSwitch and Intel DPDK is supporting NFV applications where high-frequency, small packet sizes are common. To address the challenges presented by processing small packets, Intel provides a user-space kernel-forwarding module (data plane) as part of Open vSwitch. The modifications made to create the Intel® DPDK Accelerated Open vSwitch (Intel® DPDK vSwitch) resulted in dramatic improvements to packet-switching throughput, providing an improved option for use with NFV use cases.

Ongoing work is planned to help ensure Intel DPDK vSwitch compatibility with the full range of container-based virtualization solutions and to make contributions to the main Open vSwitch project. In particular, development is underway to support direct assignment of single-root I/O virtualization (SR-IOV) virtual functions, as well as finer-grained direct assignment at the queue-pair level (i.e., RX and TX queues).

4.2 Intel® Solid-State Drive Data Center Family for PCI Express*

The reduction in virtualization overhead that is enabled by containers compared to hypervisor-only topologies allows for more application instances, users, or transactions to be run with the same level of system resources. One effect of that capability is that it places greater demand for simultaneous data accesses on the storage subsystem. Therefore, containers are particularly sensitive to the inherent limitation that conventional, spinning hard disk drives (HDDs) do not support parallel access to data.

Conversely, the NAND memory in solid-state drives (SSDs) inherently supports parallel data access, in addition to superior data-transfer rates and latency, compared to HDDs. Therefore SSDs have the clear advantage over HDDs for the large numbers of simultaneous instances and rapid provisioning and de-provisioning associated with container-based virtualization. Those advantages are particularly pronounced for enabling highly parallel big data and high-performance computing workloads using containers.

Even using SSDs, however, throughput limitations of SATA interfaces between SSDs and the motherboard is a further potential bottleneck. To address that issue, an industry consortium led by Intel developed the new Non-Volatile Memory Express* (NVMe) specification, which standardizes connectivity for SSDs and other non-volatile memory (NVM)-based devices using PCI Express* (PCIe*) 3.0, to dramatically improve data-transfer rates and latency.

Based on the NVMe standard, the Intel SSD Data Center Family for PCI Express offers a number of benefits that make it well suited to the needs of container-based virtualization:

- **Dramatically improved performance**. The Intel SSD Data Center Family for PCI Express provides up to six times faster data transfer speed than 6 Gbps SAS/SATA SSDs[2].
- **Optimized for multi-core processors**. Hardware and software parallelism enhance performance with features such as deeper command queues, parallel interrupt processing, and lockless thread synchronization.
- **Advanced software support**. Intel's NVMe driver has been incorporated in the Linux kernel since March 2012, and management capabilities are provided by the Intel® SSD Data Center Tool.
- **Enhanced RAS**. Rigorous qualification and compatibility testing, plus 230 years Mean Time Between Failures (MTBF), helps provide business-critical dependability.

4.3 Enhanced Workload Isolation with Intel® Virtualization Technology

A hypervisor-based VM provides a hardware-isolated environment for provisioning containers. While the containers within the VM are isolated from each other using only software capabilities, the group of containers within the VM as a whole can be isolated from the rest of the world using Intel VT.

As discussed above, the Linux kernel provides namespace isolation to restrict access to resources as needed. The measure of assurance provided by this method, however, is limited

to some degree by the fact that it is based in software, making it at least potentially susceptible to software-based attacks. In theory, a malicious process could reach outside its container by attacking the Linux OS and breach its separation from other containers, compromising all data and resources on the host.

Intel VT can mitigate the risk of software-based attacks with data-protection measures based on hardware features of processors, chipsets, and network controllers and adapters based on Intel architecture. Both open-source and proprietary hypervisors and other software offerings are deeply integrated with these hardware-based capabilities, through co-engineering and enabling activities by Intel. Hardware-based isolation of data, memory, and other resources assigned to a specific VM—beneath the level of system software such as OSs and hypervisors—provides an added level of data protection, beyond what's possible with containers operating on bare metal.

Containers that are provisioned inside a hypervisor-based VM have hardware-enforced access only to the code, data, and memory space specified by the hypervisor; Intel VT restricts them from paging outside the VM's memory boundaries. Applications can also be assigned to dedicated processor cores, to increase application isolation even further. Likewise, malware that has infiltrated one VM is effectively contained there, being prevented from affecting the contents of others. Noteworthy processor-level features of Intel VT that contribute to data protection within VMs include the following:

- **Descriptor table exiting**. This feature helps protect guest operating systems from internal attack by preventing the relocation of key system-data structures.
- **Pause-loop exiting**. Taking advantage of the fact that spin-locking code typically uses PAUSE instructions in a loop, this feature detects when the duration of a loop is longer than "normal" (a sign of lock-holder preemption).

Intel® VT for Directed I/O (Intel® VT-d) operates at the chipset level, providing additional capabilities to enhance hardware-based isolation of VMs, including the following:

- **Direct Memory Access (DMA) remapping**. I/O devices move data independently of the processor using DMA. Intel VT-d enables the OS or hypervisor to create protection domains, which are isolated sections of physical memory to which one or more I/O devices can be assigned. The DMA remapping hardware provided by Intel VT d uses address-translation tables to block access to protection domains by I/O devices not assigned to them.
- **Direct assignment of I/O devices**. Specific VMs can be assigned exclusive access to particular physical or virtual I/O devices, each of which is given exclusive access to a dedicated area of physical system memory; other VMs and I/O devices are restricted from reading data from or writing data to those memory locations.

5 Containers in Real-World Implementations

Exploring environments where container-based virtualization is in broad use helps characterize where this model is appropriate, where hypervisors are the better choice, and where the best option is for the two approaches to work together.

5.1 Google: Combinations and Layers of Complementary Containers and Hypervisors

Many of the technologies that enable container-based virtualization were developed at Google. With one of the world's largest computer infrastructures, the company continuously looks for more lightweight, flexible approaches to virtualize those resources. At GlueCon in May 2014, Joe Beda, a senior staff software engineer for the Google Cloud Platform, declared to the audience during his presentation that "*Everything* at Google runs in a container." A high-level look at some of the architecture associated with those containers is instructive.

Google presently has six main cloud product offerings: three storage services (based on a MySQL-compatible relational database, an object store, and a NoSQL database, respectively), the BigQuery analysis service for big data, and two compute services, described below. The use of containers and hypervisors in the compute-services architecture reveals one perspective regarding how these two forms of virtualization contrast and complement each other.

- **Google App Engine* (GAE)** is a container-based PaaS product that offers App Servers to customers—containers that each hold one or more instances of a customer's application, which have reserved, guaranteed resource levels on Google's infrastructure.
- **Google Compute Engine* (GCE)** is a hypervisor-based infrastructure-as-a-service (IaaS) product that offers Linux-based VMs running on top of KVM. Interestingly, KVM itself is installed in a container, and CoreOS (which can only run applications in containers) is emerging as the basis for the default VM image on GCE. Thus, by default, a GCE application runs in a container, within a VM, on top of a hypervisor, which is in a container, as illustrated in Figure 5.

Using containers to subdivide resources among GAE end customers offers efficient use of hardware resources, as well as flexibility to customers in

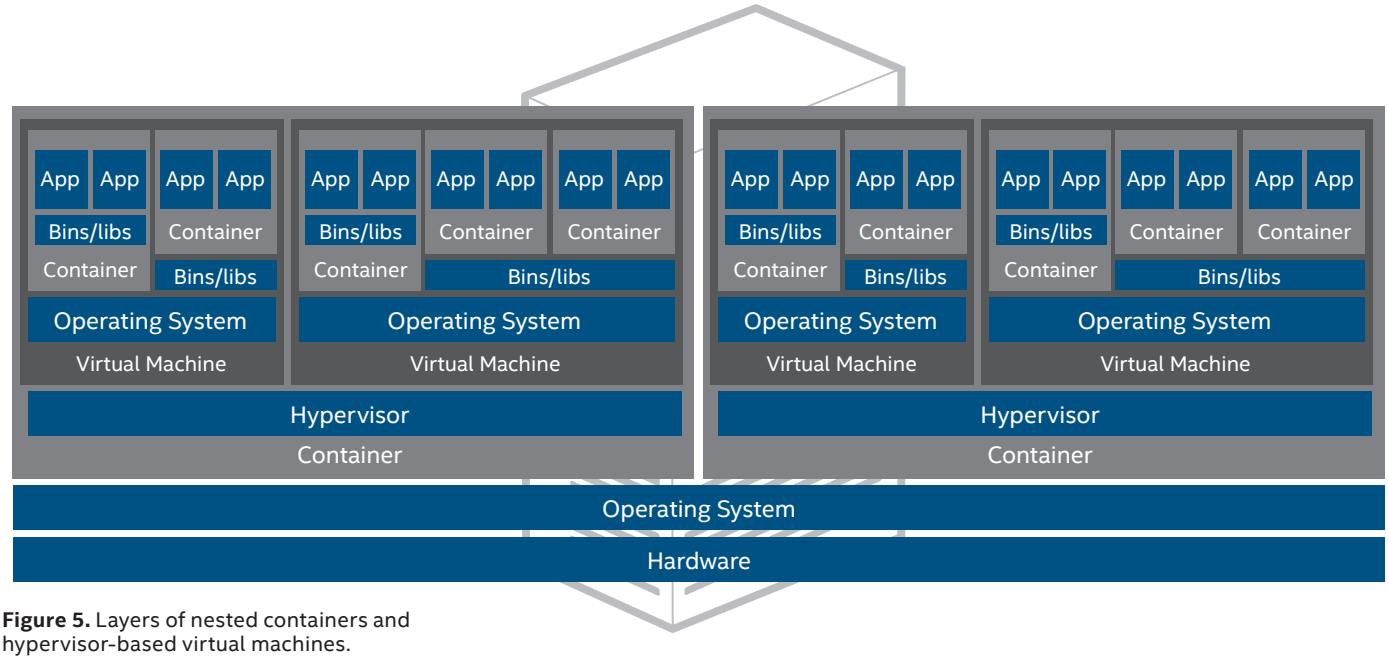


Figure 5. Layers of nested containers and hypervisor-based virtual machines.

terms of static (manual) or dynamic (automated) assignment of those resources. GAE offers simple creation and hosting of applications, websites, and services, with some trade-offs in terms of flexibility, due to the model requiring Google to preselect the finite set of languages, APIs, and frameworks that GAE supports.

The architectural differences between the two offerings are significant. Consider two hypothetical customers that must choose between GAE and GCE. The first evaluates both services as the core infrastructure for a straightforward web-based business application and chooses GAE, planning to develop in PHP with Google's Cloud SQL relational-database service on the back end. The lightweight containers model is a good fit in this case.

The second hypothetical customer wants to deploy a Hadoop* cluster within Google's cloud platform and manage it with Cloud Foundry*, as the basis for worldwide data collection that feeds into real-time data analytics. GAE does not support this use case, and the customer opts to deploy Hadoop and the related business logic in containers built on top of GCE VMs. The implementation uses the BOSH Cloud Provider Interface (CPI) to deploy all the distributed software and the Google Cloud Datastore NoSQL database service for the application's large repository of non-relational data.

5.2 Heroku and Salesforce.com: Container-Based CRM, Worldwide

As one of the largest cloud computing companies, Salesforce.com is another example of a technology provider that uses container-based virtualization in tandem with other approaches to partition compute resources. The company has steadily expanded its portfolio of product offerings in the past several years, building outward from its core Customer Relationship Management (CRM) software-as-a-service (SaaS) offering.

In particular, the company has added applications and services (many of a business-oriented social-networking nature) that integrate with and add value to customers' CRM data. While those additions originate largely from internal development or acquisitions, Salesforce also has a substantial commitment to facilitating the development of software that integrates with its CRM by third parties. The AppExchange* app store has been active since 2005, joined in 2007 by Force.com, a PaaS for the development and deployment of third-party applications that consume Salesforce CRM data.

In May 2014, the container-based PaaS operated by Heroku, a Salesforce subsidiary, was richly integrated for the first time with Salesforce CRM as a data source. Like Force.com, Heroku is designed for developers to build and deploy customer-facing applications, mobile and otherwise. In contrast, however, where Force.com primarily serves developers within enterprises building apps for employees, Heroku customers tend to be at start-ups and smaller independent software vendors. Heroku's architecture is also substantially different from the general approach at Salesforce, as shown in Table 1.

Before this integration, providing access to Salesforce data through a Heroku application involved relatively complex manipulation of APIs. The new topology simplifies that process while preserving the integrity of established workflows, toolchains, and expertise among Heroku developers. Heroku applications and

Table 1. Layers of nested containers and hypervisor-based virtual machines.

	Salesforce/Force.com	Heroku
Physical Infrastructure	Company-owned data centers	Amazon Web Services* (AWS)
Resource Partitioning	Custom multitenant architecture (details not disclosed)	Containers inside Xen*-based AWS VMs
Service Delivery	SaaS (Salesforce); PaaS (Force.com)	PaaS
Data Store	RDBMS: Oracle Database* Object store: Fileforce (proprietary)	Custom Postgres* variant
Programming Languages	Apex (proprietary); abstractions allow development without writing code	Ruby*, Node.js, Python*, Java*, others
Pricing Basis	Number of users	Capacity (e.g., number of HTTP requests)

services access Salesforce data from a Postgres* database at Heroku that synchronizes in both directions with the main Salesforce CRM data store, which is based on Oracle Database* and custom-built object storage.

Establishing data synchronization between Salesforce CRM data and the Heroku application development and hosting PaaS enables organizations to extend the reach of their business data globally through the Amazon public cloud. That reach can strengthen mobility initiatives targeting sales teams and other field resources, while taking advantage of the efficiencies of container-based virtualization and the data protection of hypervisor-based VMs. This infrastructure demonstrates the ability and value of containers to harmoniously enhance complex environments, without displacing existing architectures.

6 Conclusion

Container-based virtualization is emerging as a viable solution for mainstream data-center operators, helping them drive greater efficiency, flexibility, and performance. The capabilities of this model are an excellent fit with strategic trends that IT decision makers are likely to be already implementing or considering, from hypervisor-based virtualization to SDN.

Intel is investing in the development of hardware and software technologies that will support this trend, offering a high degree of performance optimization for Intel architecture-based servers, as well as enhanced, hardware-based data protection. A growing solution ecosystem is enriching the range of technology choices available, for use in public, private, and hybrid clouds. Integration among those options is growing as well.

Looking ahead, the addition of containers to IT infrastructures promises to help make businesses more agile, cost-effective, and capable of meeting their key objectives.

Learn more about Intel® Technology in Communications: **www.intel.com/communications**

[1] Intel® Virtualization Technology requires a computer system with an enabled Intel® processor, BIOS, and virtual machine monitor (VMM). Functionality, performance or other benefits will vary depending on hardware and software configurations. Software applications may not be compatible with all operating systems. Consult your PC manufacturer. For more information, visit www.intel.com/go/virtualization.

[2] Based on the Intel® Solid-State Drive DC P3500, P3600, and P3700 Series Product Specifications. Random I/O Operations based on IOPS.

0914/BY/MESH/PDF 330678-001US