ghaff@
illuminata.com

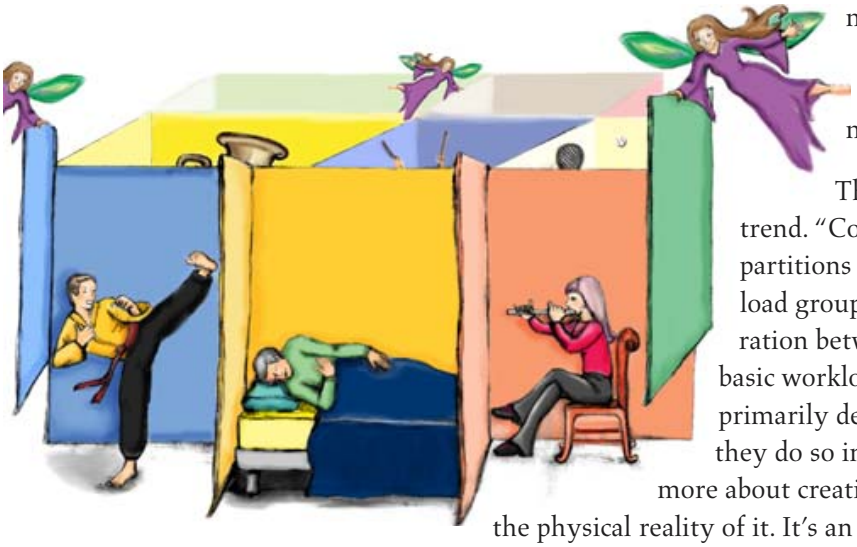
New Containments for New Times

Research Note

Gordon Haff
24 January 2005

Subdividing computers into smaller, more useful chunks is an enormously popular idea these days. Once the sole province of the mainframes that pioneered them, technologies such as partitioning, virtual machines (VM), and workload management are now available across the entire server spectrum—from behemoth Unix SMP super-servers right down to single-processor blades running Windows or Linux.[1] These different approaches to subdividing systems have legitimate technical distinctions; they are also the source of considerable marketing claims, counter-claims, and battles.[2]

The products and technologies have evolved enormously over the past several years. Certainly they've become much more mainstream. They've also diverged from largely focusing on the containment of hardware faults and system errors toward approaches that center on finer-grained, lower-overhead, and more functional subdivisions.[3]



The latest major entrant continues this trend. "Containers" are intermediate between partitions (including VMs) and traditional workload groups. They impose more order and separation between programs and workloads than do basic workload management tools—which are primarily designed to regulate performance. But they do so in a relatively lightweight way that is more about creating the "illusion" of separation than the physical reality of it. It's an illusion that can be useful for management, for security, and for efficiency.

Many Sizes and Shapes

As with other types of workload management and system partitioning, containers aren't a single "thing" with a single form. We think of containers as falling into two broad categories: application containers and more generic operating system

1. See Illuminata report "Walling Off Workloads with Partitions" (June 2002).
2. See Illuminata report "The Partitioning Bazaar: 2002" (November 2002).
3. See e.g. Illuminata reports "POWER By The Piece" (July 2004) and "VMware on the March" (August 2004).

*507A2A19199FC7ED87D2

Gordon R Haff
Illuminata, Inc.

containers. It's a distinction that isn't so much technological as one of aim and intended use. OS containers essentially emulate the way partitions or VMs carve a single system into multiple "logical systems"—but without the multiple OS images or overhead. Application containers use some of the same technical approaches, but rather than a partitioning alternative, are specifically focused on encapsulating applications to ease their deployment and migration.

At one extreme application containers can be little more than a packaging approach. In the case of J2EE application servers, "container" describes the protected collections of classes and objects typically deployed for a given application; they act as the interface between a component and the low-level platform-specific functionality that supports the component. Containers manage services such as enterprise bean and servlet life cycles, database connection resource pooling, data persistence, and access to the J2EE platform APIs. They thereby reduce the amount of low-level detail with which developers have to concern themselves. For example, a *web container* is a J2EE server component that enables access to servlets and JSP pages.

Other forms are more sophisticated, operate at the OS rather than middleware level, and use some of the same isolation techniques as their more general-purpose container brethren. A Trigence AE or Meiosys MetaCluster container is an intercepting layer that effectively decouples applications from the infrastructure on which they run, and thereby makes it easier to move them from one system (or collection of systems) to another. Each container includes necessary program files, a definition of the physical resources that they need to run, and basic state information—such as an IP address. In the case of Meiosys MetaCluster, the state of the network connections is also part of the container so as to allow relocation of connected applications without service interruption.

Separating Workloads

Type	Description	Separate OS?	Examples
Physical partition	Electrically-isolated partitions. Maximum isolation but high cost and low flexibility.	Yes	Sun Dynamic System Domains, HP nPars
Logical partition	Primarily software-based but microprocessors and firmware may provide some additional hardware-based fault isolation.	Yes	IBM LPARs, HP vPars
Virtual machine	Breaks link between hardware and its logical representation to the OS. Modest to moderate performance overhead.	Yes	IBM z/VM, VMware ESX/GSX Server, Microsoft Virtual Server
Operating system container	Virtualize at the OS rather than hardware level. "Hardened" namespace-isolated workgroups under a single OS instance.	No (but some libraries, etc. may be replicated)	Solaris containers, SWsoft Virtuozzo, Ensim VPS, BSD jails
Application container	Wrapper around application components isolate from platform and simplify migration and deployment.	No	J2EE containers, Trigence AE, Meiosys MetaCluster
Workload group management/"resource partition"	Manages processes under an OS as groups, typically to guarantee or limit resources used by an application.	No	Sun S9RM, HP PRM/WLM, IBM AIX WLM, Microsoft WSRM, Aurema ARMTech
Process resource management	Built-in, base-level control of individual processes.	No	Unix ulimit

Operating system containers are more general-purpose. They build off the workload group concept that was part of the mainframe in the form of *jobs* and which came later to Unix (including Linux) and Windows in the form of workload management add-ons like AIX's WLM, Aurema's ARMtech, HP-UX's PRM, Solaris' S9RM, and Windows' WSRM. This form of containers extends what have sometimes been called "workload management groups" or "resource partitions" with a level of "namespace isolation" between process groups that helps these OS containers evolve beyond straightforward workload management.

SWsoft's Virtuozzo has been shipping a web hosting-focused commercial product in this vein for several years.[4] BSD jails have been another example. However, it's Sun, with its long-talked-about Solaris Containers in its new Solaris 10 release that has broached this style of container as a technique on a par with VMs and other forms of partitioning. It is this container form that is the focus of the remainder of this note.

Virtualize Software, Not Hardware

Like partitions, a container presents the appearance of being a separate and independent OS image—a full system, really.[5] But, like the workload groups that containers extend, there's only one actual copy of an operating system running on a physical server.[6] Are containers lightweight partitions or reinforced workload groups? That's really a matter of definition and interpretation, because they have characteristics of each. It may help to think of them as "enhanced resource partitions" that effectively bridge the two categories.

4. See Illuminata report "Virtuozzo: The Lighter Side of Virtual Machines" (August 2004).
5. For the most part. As in the case of the "single system image" holy grail for clusters, the illusion is rarely perfect in all cases for all circumstances.
6. Or partition; these techniques can be used in combination. Thus, one physical server can be divided into two or more partitions and then subdivided further using a lighter-weight technique such as containers.

Containers virtualize an OS; the applications running in each container believe that they have full, unshared access to their very own copy of that OS. This is analogous to what VMs do when they virtualize at a lower level, the *hardware*. The OS running in each VM believes it has full, unshared access to an entire physical machine. A layer of software and firmware called a hypervisor maintains the illusion. In the case of containers, it's the OS that does the virtualization and maintains the illusion.

The Progenitors

Containers build from the basic Unix process model that forms the basis for separation.[7] Although a process is not *truly* an independent environment, it does provide basic isolation and consistent interfaces. For example, each process has its own identity and security attributes, address space, copies of registers, and independent references to common system resources. These various features standardize communications between processes and help reduce the degree to which wayward processes and applications can affect the system as a whole.

Unix also builds in some basic resource management at the process level—including priority-based scheduling, augmented by things like the intrinsic function `ulimit`, which can be used to set maximum resources such as CPU cycles, file descriptors, and locked memory used by a process and its descendents.

Add-on resource management products go much further. First of all, they group processes together in flexible and logical ways that basic Unix does not. For example, the Windows System Resource Manager (WSRM) defines process groups using what it calls "Process Matching Criteria"—rules based on process names, filenames, paths, user names, and other criteria.[8] They also provide multiple ways to carve up a system and guarantee performance (or limit it) for this group of processes

7. Windows and OpenVMS use a similar, though heavier-weight, process model.

that may correspond to a full application environment.

However, these process or workload groups still don't provide any more security or fault containment than do the basic Unix services. That requires an additional level of protection.

Making the Curtains Heavier

An historical example of upping the isolation of workload groups dates to 1999 when the FreeBSD `jail(2)` function reused the `chroot` implementation, but blocked off the normal routes to escape `chroot` confinement.[9] Jails partition the file system, process, and networking namespaces, and remove the super-user privileges that objects not entirely inside the jail would normally have.

As a newer and more sophisticated example, Solaris 10 bases its approach on a technology and administrative concept called *zones*.[10] Solaris Containers blend the capabilities and functions of these zones with Solaris resource management.

Each Solaris 10 system can have one *global zone* and up to 8191 non-global zones. Solaris assigns each zone an ID when it's booted. The global zone is always ID 0; it is the only zone that contains a bootable Solaris kernel and is aware of all devices, file systems, and other zones. The global zone is also the only zone that can configure, install, and manage other zones.

Indeed, non-global zones can't see or interact with other zones at all—except through standard network interfaces. For example, the process ID "namespace" is partitioned. Therefore, although processes within the same zone interact as usual, they can't interact with—or indeed even see—processes in other zones.[11] Each zone also has its own root directory that is allocated only an isolated part of the file system hierarchy—thus, one zone can't see another zone's data. Zones also can't access "pseudo-devices" that could be used to interact with other zones.[12]

Zones contain only a small subset of the operating system—mostly the libraries or writable structures that can differ from one OS instance to another. Non-global zones also contain localized configuration information and other zone-specific files and directories. The `zoneadm` utility creates a clean zone image based on the global zone; it resets configuration files to their "out-of-the-box" state and skips any files that only make sense or are only allowed (such as for security isolation reasons) in the global zone.

Containers: The Good

Containers can be very low-overhead. Because they run atop a single copy of the operating system, they consume very few system resources such as memory and CPU cycles. In particular, they require far fewer resources than workload management approaches that require a full OS copy for each isolated instance. Virtuozzo, for instances, has 500 containers[13] running today on a single x86 server in a service provider production environment. To be sure, this is a high-water mark, and the individual tasks are lightweight. Still, it's an example of the dramatic multiplicity that containers can reach compared to more resource-intensive partitioning techniques. IBM zSeries and pSeries systems can perhaps also stretch to multi-hundred instance

8. WSRM is included with Windows Server 2003 Enterprise and Datacenter Editions. See Illuminata report "Windows Learns to Juggle" (May 2003) for a detailed description of how this product works.
9. A `chroot`ed directory allows access to specific files and directories—and, hence, operating system features—that have been copied there.
10. Indeed, "Zones" was one of the many names that Solaris Containers went through on its long path to announcement, although Sun has now banished the term from its marketing lexicon.
11. For example, the `ps` command and the `proc` file system, a "pseudo file system" (which is used as an interface to kernel data structures) only provide information about processes in the local zone.
12. Such as `/dev/kmem`, which allows privileged software read/write access to virtual memory.
13. Which it calls Virtual Private Servers, or VPSs for short.

counts with LPARs or VMs—but only on multi-million dollar Big Iron.

Containers tend to have lower management overhead, given that there's but a single OS to be patched and kept current with security and bug fixes. Once a set of patches is applied and the system restarted, all containers automatically and immediately benefit. With other forms of partitioning, each OS instance needs to be patched and updated separately, just as they would if they were on independent, physical servers.

These are critical attributes for certain environments. The fact that SWsoft has been particularly successful with its container-like Virtuozzo product in web hosting environments is neither a surprise nor a coincidence.[14] In shared hosting, resources like CPU cycles translate directly into dollars—a lighter-weight approach means that more virtual servers can be configured on each physical one. And *that* means more profits either directly or through the ability to offer more competitive pricing. And, with thousands or tens of thousands of virtual instances to administer, patching once per physical server rather than per virtual one is attractive indeed—especially in a hosting environment where OS images are typically quite standardized anyway.

Vendors like SWsoft and Trigence augment these inherent container characteristics with extensive service provider control panels and application provisioning systems, respectively.

Finally, while containers don't provide additional fault isolation, the fact that processes running within one container are 99.999% hidden from processes running in other containers contributes significantly to security hardening. What you can't see or even know exists, you can't muck with. Such "namespace isolation" also can help performance tuning and problem identification/resolution, by narrowing the scope of concerns and reducing the number of variables that must be addressed.

14. For example, leading hosting providers such as Go Daddy and EV1Server have major VPS initiatives powered by SWsoft's Virtuozzo.

Containers: The Bad

The major downside of containers is that they do not provide much if any additional fault isolation for problems arising *outside* the process or group of processes being contained. If the operating system or underlying hardware goes, so go its containers—that is, *every* container running on the system.[15] While software running inside the competing VM technology may be susceptible to a hardware fault, it is at least isolated from operating system, driver, and most security issues.[16]

"Hard" forms of partitioning are even more isolated—from both software and hardware faults—but are only available on Big Iron gear with backplanes that can electrically isolate modules. Indeed, it's not unusual to see hard partitions combined with softer forms—VMs or containers for ultimate flexibility and physical partitions to limit the scope of more severe problems. Think of it as defense in depth.

The one OS for every container model has some other potential downsides. VMs and other forms of partitioning are frequently used to run different versions of operating systems, middleware, and applications—whether to handle legacy code, varying upgrade cycles, software interdependencies, or to conduct testing. Containers, however, need to run just one OS version and type in all containers on a system. It is possible to load some container-specific libraries, and thus have minor variations. However, the more unique containers are, the less their efficiency benefits, and the more they start to look like more typical partitions—just less isolated ones.

15. The same might be said of the hypervisor used in other software-based partitioning techniques. But hypervisors are much smaller and simpler than typical full-OSs and therefore, in principle at least, less likely to fail in complicated and hard-to-predict ways.

16. Logical partitions (LPARs) such as those on IBM's POWER5 are also primarily software-based, but they take advantage of their less-abstracted relationship with the underlying hardware to also provide protection against certain types of hardware faults. They also enjoy some tweaks and enforcement assistance from the underlying CPU design and instruction set.

Gordon R Haff
Illuminata, Inc.

Containers also carry with them some configuration restrictions and limitations that don't exist with other partition types—the result of playing slight-of-hand within the OS to maintain their illusion of separation.[17] For example, in the case of Solaris Containers, the system administrator has to carefully manage how devices get assigned to specific zones. Allowing unprivileged users to access certain types of hardware could permit those devices to be used to cause system-wide panic, bus resets, or other adverse effects. Overriding the default configuration and placing a physical device into more than one zone could also create a covert channel between zones or allow corruption by one zone to affect another—but this is really no different from the case of a SAN connected to multiple physical systems or VMs.

Nor have containers yet been truly accepted by ISVs like Oracle as a legitimate means to subdivide systems for licensing purposes. If Oracle software runs on an eight-processor partition—whether physical, logical, or VM—within a 16-processor system, Oracle only requires an eight-processor license. But if the separation is only enforced by resource management techniques, Oracle clearly requires that the full 16 processors be licensed. Thus, containers often can't be used to manage (read: decrease) software licensing costs in the same way as harder forms of separation. Sun is working with Oracle to change that policy (and, indeed, that policy should change), but today it can disadvantage containers *vis-à-vis* other approaches.

While genuine, these drawbacks aren't show-stoppers for many applications. Today, many applications successfully run side-by-side under a single OS image without any additional isolation at all. Containers provide a way to introduce additional isolation, without requiring much of the system resources or administrative overhead. Containers are also certainly suitable for application development and test scenarios, and even for some server consolidation scenarios, especially those of service provisioning.

17. Other partitions, especially VMs, are performing their own magicians' slight-of-hand. However, because they're doing so at a much lower level in the system, the effects are better hidden from users and administrators.

Conclusion

A variety of products already providing containers include BSD (Jails), Sun's Solaris 10, SWsoft's Virtuozzo, and Trigence's Application Environment (AE). In short, containers are getting to be a very common option for workload management and isolation.

There are good reasons for this. Underlying hardware and OSs have gotten more reliable. Sure, they may still "blow up" from time to time—but dramatically less frequently than in any year past. Today, when the OS fails, it's often the result of its being misconfigured or improperly updated, an operational problem that containers can help to solve because one patch can apply to all the containers on a system. OSs have also gotten more sophisticated as well as more reliable. They have more dials and knobs to tweak performance and parameters. They have more gauges to see what's going on inside. These foundations for controllability and isolation give containers—which must build on existing OS services to a large degree—a solid foundation.

Marketing hype notwithstanding, one size rarely fits all, and it's the rare technical approach that is drawback-free. Containers aren't the panacea that some of its more fervent supporters would have you believe they are. It's still sometimes important to isolate faults in a most robust way, for example, for critical, state-heavy enterprise apps like databases. But for many other cases, the underlying infrastructure is now reliable enough that the focus can shift to lighter-weight and more flexible ways of carving up a system—especially on the smaller systems that have displaced Big Iron across so much of the typical IT infrastructure. Containers don't suit every requirement, but they're increasingly a useful arrow in the quiver of datacenter architects and admins.