ORACLE®
SOLARIS

An Oracle White Paper
July 2011

Consolidating Applications with Oracle Solaris Containers

ORACLE®

Introduction

The main focus in IT departments today is increasing service levels while reducing the cost of the IT infrastructure. To reduce costs, businesses are eager to consolidate applications onto fewer servers, but they must be careful to isolate these applications to provide adequate resources and security. In order to effectively consolidate applications onto fewer systems, it must still be possible to manage the applications independently. This requires the ability to control resource utilization, isolate faults, and manage security between multiple applications. This is all possible on one system with hardware partitioning, such as dynamic system domains, but until the advent of the Oracle Solaris 10 operating system (OS) it was not possible to accomplish within one instance of the OS. With these abilities added to the OS, there is an opportunity to further reduce costs by managing fewer OS instances and sharing resources.

Hardware partitioning provides a very high degree of application isolation but is more expensive to implement and is generally limited to high-end systems. In addition, the mechanisms for allocating and reallocating resources—adding a CPU/memory board to a domain or reconfiguring one to another domain—are sometimes less flexible than desired for today's changing application requirements. Because they are less flexible, there is less sharing of resources. If applications can share a common infrastructure, more resources can be shared, which can lower the amount of resources required, simplify administration, and thus reduce the total cost of ownership (TCO) for the applications.

Oracle Solaris resource management offers more flexibility for sharing resources between applications running on a single system or domain, and it has been available as Oracle Solaris Resource Manger since the release of Oracle Solaris 2.6 (unbundled from releases 2.6 to 8) and is now integrated into the OS as the feature Oracle Solaris resource management (since the release of Oracle Solaris 9). However, by itself it does not provide isolation between applications on the same system or domain.

How Oracle Solaris Containers Solve These Problems

Oracle Solaris Containers implement OS virtualization using containers that are designed to provide a complete, isolated, secure runtime environment for applications—and in situations where resources can be managed with sub-CPU granularity. Oracle Solaris Containers allow application components to be isolated from each other using flexible, software-defined boundaries within a single instance of the OS. These applications can then be managed independently to allocate the quantity of system resources each workload is permitted to utilize. Oracle Solaris Containers provide secure, partitioned environments for applications and users, they give administrators almost unlimited flexibility to assign and isolate resources to particular containers, and in many cases they can be used to allocate resources within a single container.

Containers establish boundaries for resources such as CPUs and can be expanded to adapt to the changing processing requirements of the application or applications running in the container. A container is a virtualized OS environment created within a single instance of Oracle Solaris. Applications within containers are isolated, preventing processes in one container from monitoring or affecting processes running in another container. Even a superuser process from one container cannot view or affect activity in other containers. A container also provides an abstract layer that separates applications from the physical attributes of the system on which they are deployed. Examples of these attributes include physical device paths.

Containers enable more-efficient utilization of the system. Dynamic resource reallocation permits unused resources to be shifted to other containers as needed. Fault and security isolation means that poorly behaved applications do not require a dedicated and underutilized system. With containers, these applications can be consolidated with other applications. Containers also allow the system administrator to delegate some administrative functions while maintaining overall system security.

Oracle Solaris Containers are designed to provide fine-grained control over the resources that applications use, allowing multiple applications to operate on a single server while maintaining specified quality of service (QoS) levels. Fixed resources such as processors and memory can be partitioned into pools on multiprocessor systems, with different pools shared by different *projects* (a specified collection of processes) and isolated application environments. Dynamic resource sharing allows different projects to be assigned different ratios of system resources. Oracle Solaris IP quality of service (IPQoS) can be employed to manage network bandwidth used by multiple, competing network applications. When resources such as CPUs and memory are dynamically allocated, resource capping controls can be used to set limits on the amount of resources used by a project. With all of these resource management capabilities, organizations can consolidate many applications onto one server, helping to reduce operational and administrative costs while increasing availability.

How This White Paper Is Organized

The goal of this white paper is to provide information about consolidating applications onto a single server using Oracle Solaris Containers to isolate applications and efficiently manage system resources between them. The following sections contain details on the steps to consolidate applications. This white paper is written as a design guide rather than an installation guide. For this reason, the actual tasks that an administrator performs to set up the system may be the exact opposite of those outlined here.

There are two general ways to approach a consolidation project using Oracle Solaris Containers. The first is to start with an existing system that is to house the consolidated applications and architect all of the pieces within the constraints of that system. The second way is to decide which applications should be consolidated onto a single server and build the server to meet the needs of the consolidated applications. This white paper uses the second approach to demonstrate how to consolidate applications using Oracle Solaris Containers.

When to Deploy Containers

Containers can be used to isolate and manage multiple applications on the same server to reduce costs and complexity. The technology enables IT to deploy multiple, competing applications on the same platform, and multiple organizations can use a single server, each with its own secure environment. For example, containers can be used to effectively deploy hosting facilities, consolidate multiple database instances, consolidate development and test environments, stage applications, and consolidate Web services systems.

Hosting Facilities for Customers or Users

With containers, each customer or user can have their own virtualized environment, including IP address, disk storage, applications (such as Web servers, telnet, and sendmail), and even a root password. Containers are ideal for universities, financial institutions, or any other industry where individual users require varying resources or more control over their own environment. For example, many university students require root access to perform installation or reboot tasks. Rather than provide a system for every student, it is now possible to provide a container per student on a system with one instance of the OS and ratio of the system resources, as shown in Figure 1. This greatly reduces administration tasks for the administrators responsible for maintaining those systems. For financial users, it is now possible to assign heavy users to their own containers with an allocated amount of resources, to enable them to run complicated queries without affecting the performance of the other users on the system, and to have the freedom to run different versions of an application.
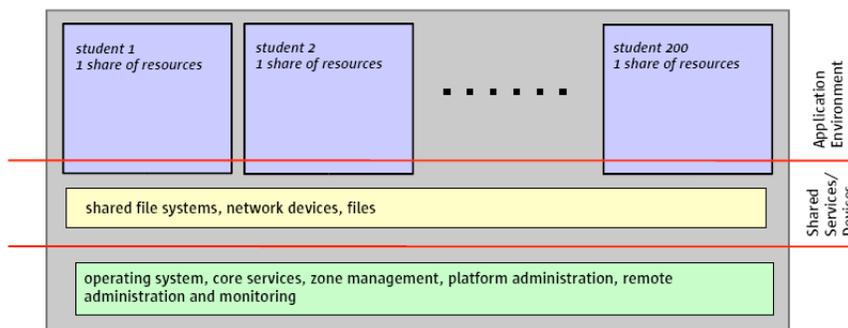
Figure 1. This figure illustrates the use of containers for university students.

Consolidating Multiple Database Instances

Consolidating multiple database instances into separate containers on the same system enables competing applications, such as online transaction processing (OLTP) and data warehousing, to run with resource allocation, changing as business needs change. For example, in Figure 2, the OLTP container is allocated 70 shares of the CPU resources, while the data warehouse is allocated 10 shares, resulting in a 7:1 ratio of CPU resources allocated to each container. Shares allow unused cycles to be used by other applications, or the allocation can be changed dynamically during peak times to provide

more CPU resources to either container. In addition, the resources for each container are further subdivided, allocating a portion of resources to each project within the container. This helps ensure that each project always has the resources it requires to function predictably. (Note: Shares are not required to add up to 100. They simply express a ratio.)

Database administrators can have complete control over their isolated environment. In addition, a separate project can be created specifically for database administrators in order to limit their access to resources, which can keep commands such as `find` from consuming critical CPU resources and negatively affecting the performance of the database.
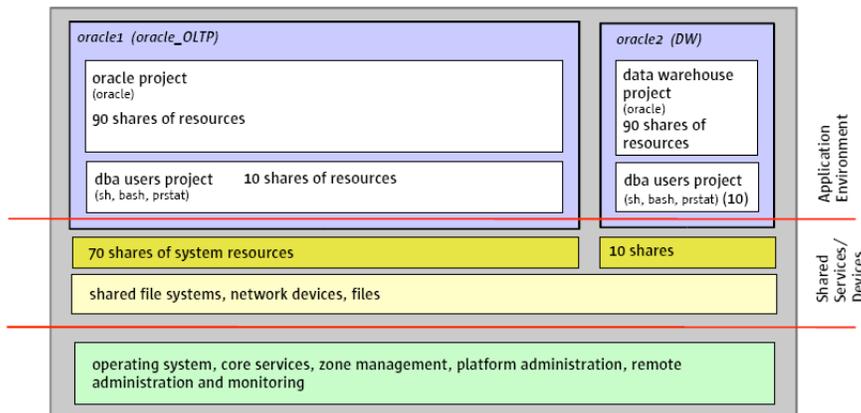
Figure 2. This figure illustrates consolidating multiple database instances and limiting database administrator access.

Software Development and Test

With containers, development and test systems using the same version of the OS can be consolidated onto a single system, as illustrated in Figure 3. Developers can test and develop software with root access to install or run applications in separate containers. They can also exercise different scenarios, each in its own container. In addition, developers can quickly test or develop new applications or versions of applications without needing to go through a purchasing process to acquire new systems. This provides the dual benefit of reducing the time to acquire and set up systems and reducing the cost of development.

Containers are dynamic—they can be predefined, kept inactive, and then booted quickly. They can be frozen or swapped out when needed. This makes them ideal for development and test environments.
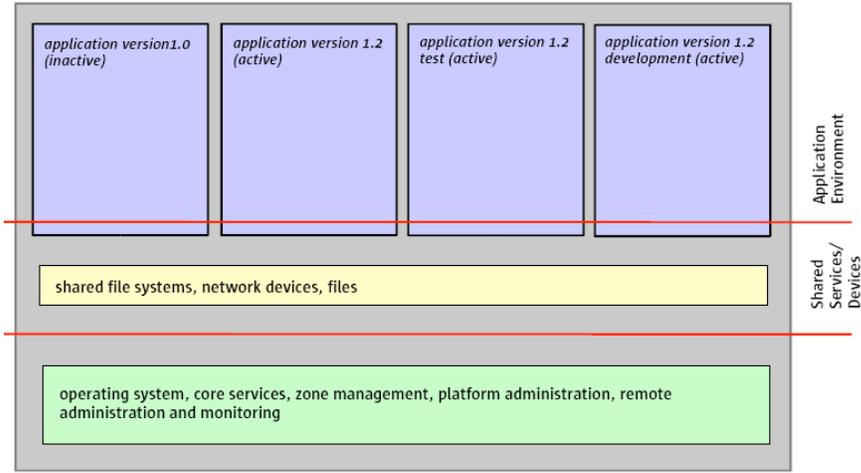
Figure 3. This figure illustrates consolidated development and test environments.

Application Staging

With containers, IT organizations can run a production application in one container and stage a new version in another container, as depicted in Figure 4. Each application has its own environment and isolated configuration files, so when the time comes to migrate the staged version into the live production system, all that is required is changing network addresses and resource allocations. This is also advantageous because the OS instance for both containers is the same; therefore, there is no need to worry about replicating an exact environment for staging new applications. And, if there are problems with the new version of the application, it is easy to return the production system to the previous version.
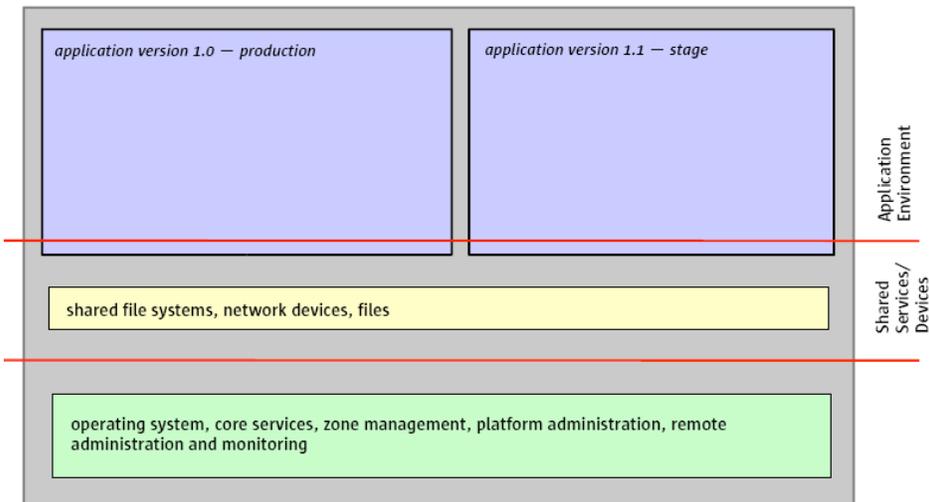
Figure 4. This figure illustrates staging different versions of an application on the same system.

Web Services

Containers can also be employed to consolidate vertically scaled application and database servers onto the same multiprocessor server, cutting in half the number of servers required on the back end yet still providing isolation between the applications, as depicted in Figure 5. With containers, the system resources can be divided between the Web application server container and the database container and further subdivided within every container, enabling each project to access the CPU resources it needs.
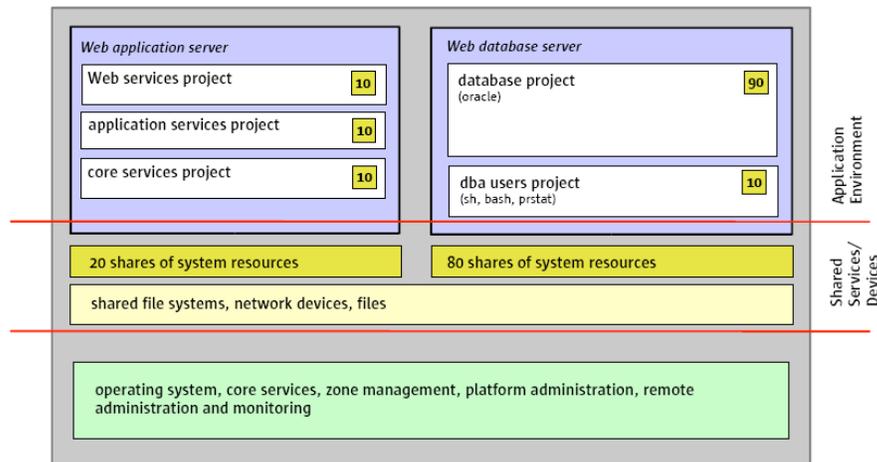
Figure 5. This figure illustrates consolidating Web application and database servers.

Isolating Applications

Oracle Solaris Containers consist of two distinct parts—Oracle Solaris Zones and resource management. Oracle Solaris Zones partitioning technology is the mechanism for isolating applications within Oracle Solaris Containers that are then allocated resources with resource management software and utilities. An Oracle Solaris Container combines both zones and resource management.

This section provides details on how to determine the performance needs of applications in order to classify them into projects. It then introduces zones and discusses how they work to isolate applications running on the same system and how to create projects within zones in order to isolate them from projects in other zones. The next section, "Managing CPU Resources," discusses how to manage resources to support the applications within an Oracle Solaris Container.

Determining the Needs of Applications

After choosing applications to consolidate, the next step is to identify or classify the components, such as processes and users, that make up the workload for a given application. Possible approaches can be to identify workloads by username or by processname. Workloads can also be classified in terms of users, groups of users, and groups of applications. For example, financial services users that routinely execute complex queries can be grouped into a specific workload.

The next step is to determine the performance requirements for each defined workload. This task involves monitoring the real-time activity of the applications on their current systems, including CPU, memory, network, and storage requirements and usage. In addition, it is necessary to ascertain which types of file systems, shared file systems, and shared libraries the workloads utilize in order to properly configure the new system and to efficiently share resources such as read-only file systems (libraries, man pages, and so on). The more information that can be gathered about the applications, the more efficiently the applications can be consolidated.

The final step in determining the needs of the applications is to prioritize the workloads that are to share the resources of the system. The workloads should be prioritized in terms of which applications require the most resources and the time periods in which they need them. It is also important to identify competing workloads that are to be housed on the same system, and to distinguish those workloads that are not in conflict from those with performance requirements that compromise the primary workloads. The next step is to create projects for these workloads.

Creating Projects

Oracle Solaris provides a facility called *projects* to name workloads once they are identified. The project serves as an administrative tag used to group related work in a manner deemed useful by the system administrator—for instance, one project for a sales application and another project for a marketing application. By placing all processes related to the sales application in the sales project and the processes for the marketing application in the marketing project, as shown in Figure 6, it is now possible to separate and control the workloads in a way that fulfills the business objectives of the company.

Figure 6. This figure depicts separating processes for applications into projects.

Projects are defined in the project database. The project database can be a local file or in a name service such as NIS or LDAP. With the project database in NIS or LDAP, the project definition can be shared across multiple containers or systems.

Every process since the Oracle Solaris 8 release runs in a project, such as `system`, `default`, or `user.root`. Freshly installed systems always have a local project database in `/etc/project` containing five standard projects: `system`, `user.root`, `noproject`, `default`, and `group.staff`. The system project is used for all system processes and daemons. When a user logs in as root, all of the root user processes run in the `user.root` project. The `noproject` project is to catch exceptions. The `default` project serves as a catchall and is used for users not matching any of the other projects. The `group.staff` project is used for all users in the group `staff`.

Projects are created with the `projadd` command. The following example creates projects to manage separate workloads for an Oracle marketing instance and an Oracle sales instance. It also creates a default project for the user `oracle`:

```
# projadd -c "Oracle default project" group.dba
# projadd -c "Oracle Marketing" -U oracle ora_mkt
# projadd -c "Oracle Sales" -U oracle ora_sales
```

The -U oracle option specifies that the oracle user is allowed to run processes in these projects. The /etc/project file now contains

```
# cat /etc/project
system:0::::
user.root:1::::
noproject:2::::
default:3::::
group.staff:4::::
group.dba:100:Oracle default project:::
ora_mkt:101:Oracle Marketing:oracle::
ora_sales:102:Oarcle Sales:oracle::
```

Isolating Applications in Containers

The next step in designing a system to consolidate applications with Oracle Solaris Containers is to isolate projects or groups of projects from each other in separate containers. The technology employed to do this is Oracle Solaris Zones. It is necessary to have a general understanding of zones in order to determine how to apply them when consolidating applications.

Oracle Solaris Zones

Every Oracle Solaris 10 system contains a general global environment, like previous versions of the OS, called a *global zone*. The global zone performs dual functions—it is both the default zone for the system and the zone used for systemwide administrative control. All processes run in the global zone if no *nonglobal zones*, referred to simply as *zones*, are created by the *global administrator*.

The Global Zone

The global zone is the only zone from which a nonglobal zone can be configured, installed, managed, or uninstalled. Only the global zone is bootable from the system hardware. Administrative functions of the system infrastructure, such as physical devices, routing, or dynamic reconfiguration (DR), are only possible in the global zone. Appropriately privileged processes or users running in the global zone can access objects associated with other zones.

Unprivileged processes or users in the global zone might be able to perform operations not allowed to privileged processes or users in a nonglobal zone. For example, users in the global zone can view information about every process in the system. If this capability presents a problem for a site, access can be restricted to the global zone for particular users. Thus, zones allow the administrator to delegate some administrative functions while maintaining overall system security.

Nonglobal Zones

A nonglobal zone does not need a dedicated CPU, a physical device, or a portion of physical memory. These resources can be either multiplexed across a number of zones running within a single domain or

system, or allocated on a per-zone basis using the resource management features available in the OS. Zones can be booted and rebooted without affecting other zones on the system.

Each zone can provide a customized set of services. To enforce basic process isolation, a process can see or signal only those processes that exist in the same zone. Basic communication between zones is accomplished by giving each zone at least one logical network interface. An application running in one zone cannot observe the network traffic of another zone. This isolation is maintained even though the respective streams of packets travel through the same physical interface.

Each zone that requires network connectivity is configured with one or more dedicated IP addresses. A process assigned to a zone can manipulate, monitor, and directly communicate with other processes that are assigned to the same zone. The process cannot perform these functions with processes that are assigned to other zones in the system or with processes that are not assigned to a zone. Processes that are assigned to different zones are only able to communicate through network application programming interfaces (APIs).

Zones do not present a new API or application binary interface (ABI) to which applications must be ported. Instead, zones provide the standard Oracle Solaris interfaces and application environment, with some restrictions. The restrictions primarily affect applications that attempt to perform privileged operations. Applications in the global zone run without modification, whether or not additional zones are configured. Applications or tools that require certain root privileges for system management, such as adding or loading device drivers or changing IP addresses, may not be suitable or may have to be rewritten in order to work in a zone environment. In other words, things that may impact security between zones generally must be run in the global zone.

Zones are supported on any system running Oracle Solaris 10. The practical upper limit for zones on one system is 8,000. The number of zones per system is determined by the total resource requirements of the application software running in all of the zones.

Creating Projects in Zones

With a general understanding of zones, the next step is to isolate the projects defined in the steps above into zones. After identifying cooperating and conflicting workloads, determine which workloads can share zones and which require their own zone. This can be based on security requirements, sharing requirements, and so on. If not much is known about an application or workload, it can be isolated in a zone by itself until more information can be gathered.

Some workloads need to share a namespace, in which case they should reside within the same zone. Other workloads can share a namespace and can therefore share a zone. For example, applications that run the same version, such as two instances of a database, can be defined by two projects in the same zone. Some applications need separate namespaces and should be placed in separate zones. For example, Web servers that use port 80 or install configuration files in the same place (`/etc/hpptd`) should reside in separate zones. Other applications may pose security risks, such as mail servers, and should be isolated in their own zones. If applications or workloads can share a namespace, either because they can or because they must, put them in the same zone in different projects. If they cannot share a namespace—because of conflicts, security, or lack of information—put them in separate zones.

The following examples illustrate how to create zones and then create projects within zones. If there is only one project in a zone, it can use the default project. To keep this example simple, a directory under /export is used for the disk space for each zone. Two zones are created, one for the marketing database and one for the sales database. The example configures a virtual network interface for the zones on the physical interface hme0. Global zone administrators create the directories for the zones in the global zone. The directories used for the zones must be owned by root and have mode 700 to prevent normal users in the global zone from accessing the zone's file systems. This section of the example creates the marketing zone represented in Figure 7.

```
global #zonecfg -z mkt
mkt: No such zone configured
Use 'create' to begin configuring a new zone.
zonecfg:mkt> create
zonecfg:mkt> set zonepath=/export/zone/mkt
zonecfg:mkt> set autoboot=true
zonecfg:mkt> add net
zonecfg:mkt:net> set address=10.6.49.141
zonecfg:mkt:net> set physical=hme0
zonecfg:mkt:net> end
zonecfg:mkt> verify
zonecfg:mkt> commit
zonecfg:mkt> exit

global # zoneadm -z mkt install (installs the zone)
global # zoneadm -z mkt boot (boots the newly installed zone)
```
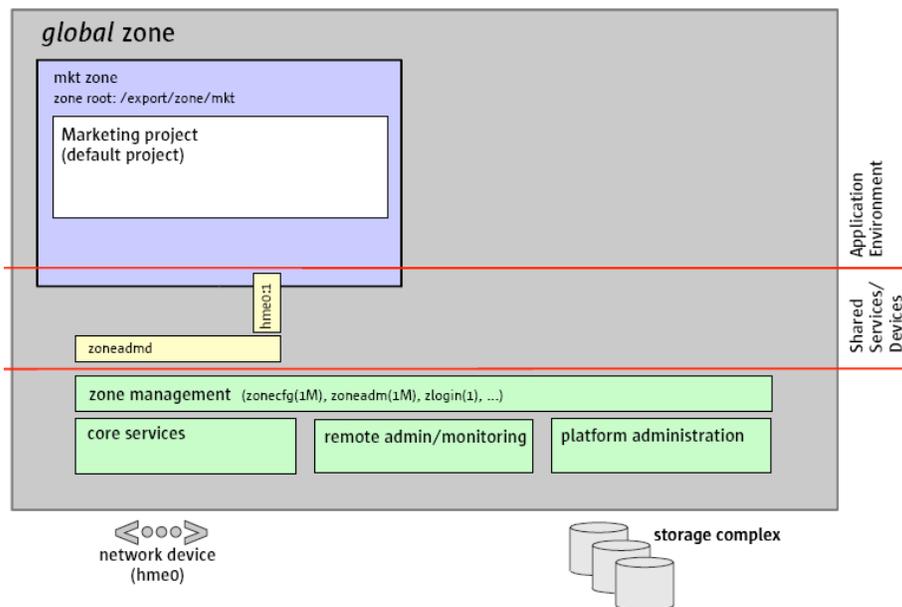

Figure 7. This figure illustrates the creation of a marketing zone.

To run Oracle in the newly created zone, the group dba and the user Oracle are required. A project is required in order to set the shared memory segment size for Oracle. Prior to Oracle Solaris 10, the shared memory segment size was set in /etc/system. In Oracle Solaris 10, it is possible to set it

more granularly within a project or zone. Because there is only one Oracle instance in this zone (running in the default project), the simplest way is to create a `user.oracle` project and add the `project.max-shm-memory` resource control to it, as illustrated below and in Figure 8.

```
mkt # mkdir -p /export/home
mkt # groupadd dba
mkt # useradd -g dba -d /export/home/oracle -m -s /bin/bash oracle
mkt # passwd oracle
mkt # projadd -c "Oracle" user.oracle
mkt # cat /etc/project
system:0::::
user.root:1::::
noproject:2::::
default:3::::
group.staff:10::::
user.oracle:100:Oracle:::project.max-shm-memory=(privileged,2147483648,deny)
```
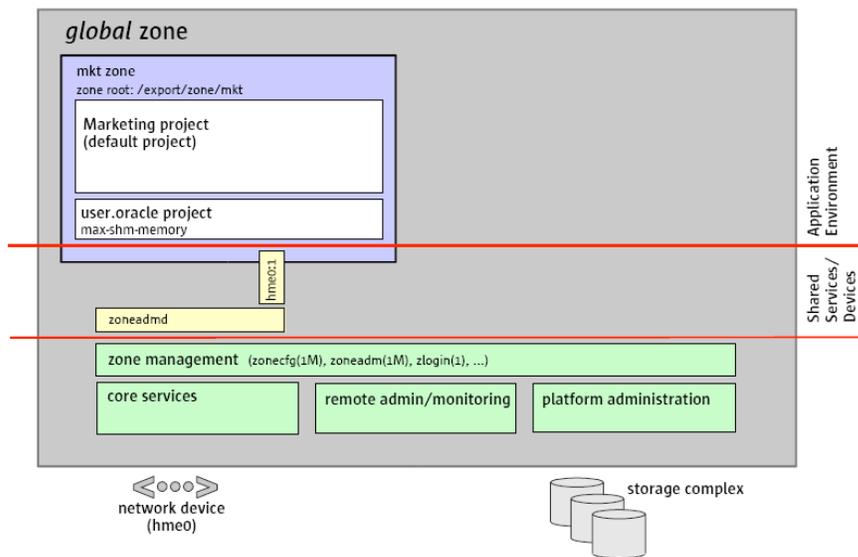

Figure 8. This figure illustrates the process of adding a `user.oracle` project to the zone.

Note that the zone has its own namespace and that the `user`, `group`, and `project` just created are therefore only visible inside the `mkt` zone. The next example creates the sales database zone using a small volume (`/dev/md/dsk/d100`) to mount the zone root file system. This example also uses slice `c1t3d0s0` for a new file system to house the Oracle software. Please note that this is simply one example of how to create a zone. Please see the administrator guides for other options.

```
global # newfs /dev/md/rdsk/d100
global # mount /dev/md/dsk/d100 /export/zone/sales

global #zonecfg -z sales
sales: No such zone configured
Use 'create' to begin configuring a new zone.
zonecfg:sales> create
zonecfg:sales> set zonepath=/export/zone/sales
zonecfg:sales> set autoboot=true
zonecfg:sales> add net
zonecfg:sales:net> set address=10.6.49.142
zonecfg:sales:net> set physical=hme0
zonecfg:sales:net> end
zonecfg:sales> add device
zonecfg:sales:device> set match=/dev/rdsk/c1t3d0s0
zonecfg:sales:device> end
zonecfg:sales> add device
zonecfg:sales:device> set match=/dev/dsk/c1t3d0s0
zonecfg:sales:device> end
zonecfg:sales> verify
zonecfg:sales> commit
zonecfg:sales> exit

global # zoneadm -z sales install (installs the zone)
global # zoneadm -z sales boot (boots the newly installed zone)
```

The additional slice can now be mounted into the zone:

```
sales # newfs /dev/rdsk/c1t3d0s0
sales # mkdir /u01
sales # mount /dev/dsk/c1t3d0s0 /u01
```

Now a project is added to the zone. Because the zones have completely separate namespaces, the user, group, and project for Oracle, must be created in this zone as well. The two zones, marketing and sales, are shown in Figure 9 below.

```
sales # mkdir -p /export/home
sales # groupadd dba
sales # useradd -g dba -d /export/home/oracle -m -s /bin/bash oracle
sales # passwd oracle
sales # projadd -c "Oracle" user.oracle
sales # cat /etc/project
system:0::::
user.root:1::::
noproject:2::::
default:3::::
group.staff:10::::
user.oracle:100:Oracle:::project.max-shm-memory=(privileged,2147483648,deny)
```
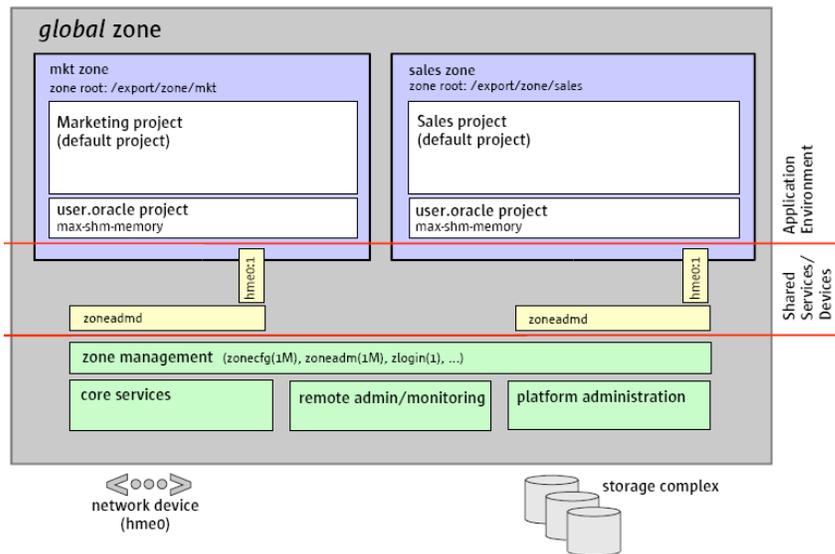
Figure 9. This figure illustrates marketing and sales zones with projects.

There are other things that need to be considered when configuring zones, such as which services need to be running (such as `inetd`, `sshd`, `telnetd`, and `sendmail`), which file systems or parts of file systems need to be mounted, which network ports need to be imported with which IP address, and so on. After all of these things are determined, the next step is to manage the system resources so that each application is allocated the proper amount of CPU, memory, and bandwidth to perform its function. These topics are covered in later sections of this white paper.

Managing CPU Resources

Now that the workloads of the applications are divided into projects and those projects are isolated within zones, the next step is managing the system resources between the zones. This is accomplished using the resource management component of Oracle Solaris Containers. Oracle Solaris resource management enables the physical system to be divided into resource pools that can be further subdivided between the zones and the projects within each zone.

Solaris Resource Management

In order to efficiently consolidate applications on a single system, the OS must be able to flexibly respond to the varying workloads generated by different applications. If the resource management features of Oracle Solaris Containers are not used, the Oracle Solaris OS responds to workload demands by providing equal access to resources. Oracle Solaris resource management features offer the ability to treat workloads individually. These features enable administrators to

• Restrict access to a specific resource

• Offer resources to workloads on a preferential basis

- Isolate workloads from each other

- Deny resources or prefer one application over another for a larger set of allocations than otherwise permitted

- Prevent an application from consuming resources indiscriminately

- Change an application's priority based on external events

- Balance resource guarantees to a set of applications against the goal of maximizing system utilization

These features enable Oracle Solaris Containers to deliver predictable service levels. Effective resource management is enabled in the Oracle Solaris OS with control, notification, and monitoring mechanisms. Many of these capabilities are provided through enhancements to existing mechanisms such as the `proc`(4) file system, processor sets, and scheduling classes, and new mechanisms such as dynamic resource pools.

Resource pools enable the system to be divided so that workloads do not consume overlapping resources. They provide a persistent configuration for processor sets and scheduling class assignment. Resource pools now provide a mechanism (as of the Oracle Solaris 10 release) for dynamically adjusting each pool's resource allocation in response to system events and application load changes. And, scheduling classes can be assigned to pools to further divide and dynamically manage CPU resource allocation with sub-CPU granularity.

Resource Pools

Resource pools mechanism is primarily for use on large systems of more than four CPUs. However, small systems can still benefit from this functionality to divide the system's critical resources. With the introduction of multicore CPUs, pools become even more important for smaller systems.

Dynamic resource pools—that can adjust each pool's resource allocation in response to system events and application load changes—simplify and reduce the number of decisions required from the administrator. Pools are automatically adjusted to preserve the system performance goals. These features are enacted through the resource controller `poold`, a system daemon that should always be active when dynamic resource allocation is required. Periodically, `poold` examines the load on the system and determines whether intervention is required to enable the system to maintain its optimal performance.

Classifying resources, as opposed to identifying resources, can be performed along a number of axes. The axes can be implicitly requested as opposed to explicitly requested, time-based (such as CPU time), compared to time independent (such as assigned CPU shares), and so on.

Generally, scheduler-based resource management is applied to resources that the application can implicitly request. For example, to continue execution, an application implicitly requests additional CPU time. To write data to a network socket, an application implicitly requests bandwidth. Constraints can be implemented on the aggregate total use of an implicitly requested resource. Additional interfaces can be presented so that bandwidth or CPU service levels can be explicitly negotiated. Resources that are explicitly requested, such as a request for an additional thread, can be managed by constraint.

Resource Management Control Mechanism

The three types of control mechanisms that are available in Oracle Solaris are constraints, scheduling, and partitioning.

• **Constraint mechanisms.** This is a resource sharing mechanism that sets bounds on the amount of specific resources a workload can consume. It can also be used to control ill-behaved applications, such as applications with memory leaks that can otherwise compromise performance or availability through unregulated resource requests. Constraints can present complications for an application when the relationship between the application and the system is modified to the point that the application is no longer able to function. One approach that can mitigate this risk is to gradually narrow the constraints on applications with unknown resource behavior.

• **Scheduling mechanisms.** Scheduling is also a resource sharing mechanism that refers to making a sequence of resource allocation decisions at specific intervals based on a predictable algorithm. An application that does not need its current allocation leaves the resource available for another application's use. For example, if an OLTP database and a data warehouse are in the same resource pool, the OLTP database is able to access the resources of the data warehouse if it is not using them. Scheduling-based resource management enables full utilization of an undercommitted configuration, while providing controlled allocations in a critically committed or overcommitted scenario. The algorithm determines the level of control. For example, it might guarantee that all applications have some access to the resource. The fair share scheduler is an example of a scheduling mechanism that manages application access to CPU resources in a controlled manner.

• **Partitioning mechanisms.** Partitioning is a more rigid mechanism used to bind a workload to a subset of the system's available resources. This binding guarantees that a known amount of resources is always available to the workload. Resource pools are a partitioning mechanism that limits workloads to a specific subset of the resources of the system. Partitioning can be used to avoid systemwide overcommitment. However, in avoiding this overcommitment, the ability to achieve high utilizations can be reduced because resources bound to one pool are not available for use by a workload in another pool when the workload bound to them is idle, unless a policy for dynamic resource pools is employed. A good candidate for this type of control mechanism might be transaction processing systems that must be guaranteed a certain amount of resources at all times.

Managing CPU Resources with Resource Pools

The ability to partition a server using processor sets has been available since version 2.6 of Oracle Solaris. Every system contains at least one processor set—the system or default processor set that consists of all of the processors in the system. Additional processor sets can be dynamically created and removed on a running system using `poolcfg` and `pooladm`(1M), providing that at least one CPU remains for the system processor set.

Resource pools enable administrators to create a processor set by specifying the number of processors required, rather than CPU physical IDs. The definition of a processor set is therefore not tied to any particular type of hardware. It is also possible to specify a minimum and maximum number of

processors for a pool. Multiple configurations can be defined to adapt to changing resource requirements, such as different daily, nightly, or seasonal workloads.

Resource pools can have different scheduling classes. Scheduling classes work per resource pool. The two most common are the *fair share scheduler* and the *timesharing scheduler*.

Fair Share Scheduler

The fair share scheduler (FSS) allocates CPU resources using CPU shares. The FSS helps ensure that CPU resources are distributed among active zones or projects based on the number of shares each zone or project is allocated. Therefore, more-important workloads should be allocated more CPU shares. This section discusses setting shares for projects. (Assigning shares to zones is discussed earlier in this white paper.)

A CPU share defines the portion of the CPU resources available to a project in a resource pool. It is important to note that CPU *shares* are not the same as CPU *percentages*. Shares define the relative importance of projects with respect to other projects. If the sales project is twice as important as the marketing project, the sales project should be assigned twice as many shares as the marketing project. The actual number of shares assigned is largely irrelevant—2 shares for the sales project versus 1 share for the marketing project yields the same results as 18 shares for the sales project versus 9 shares for the marketing project. The sales project is entitled to twice the amount of CPU as the marketing project in both cases. The importance of the sales project relative to the marketing project can be increased by assigning more shares to the sales project while keeping the same number of shares for the marketing project. Project shares used by the FSS are specified through the `project.cpu-shares` property in the `project`(4) database.

The FSS calculates the proportion of CPU allocated to a project by dividing the shares for the project by the total number of active projects. An *active project* is a project with at least one process using the CPU. Shares for idle projects—projects with no active processes—are not used in the calculations. For example, if projects sales, marketing, and transaction with two, one, and four shares respectively, are all active, then the sales project is entitled to two-sevenths, the marketing project to one-seventh, and the transaction project to four-sevenths of the available CPU resources for that resource pool. If the sales project is idle, the marketing project is entitled to one-fifth and the transaction project to four-fifths of the CPU. Note that even though the actual CPU entitlement for the marketing and transaction projects increases, the proportion between the marketing and transaction projects stays the same.

An important thing to note is that the FSS only limits CPU usage if there is competition for the CPU. A project that is the only active project on the system can use 100 percent of the CPU, regardless of the number of shares it holds. CPU cycles are never wasted—if a project does not use all of the CPU it is entitled to because it has no work to perform, the remaining CPU resources are distributed between other active processes.

If a small share is allocated to a busy workload, it might slow that workload's performance. However, the workload is not prevented from completing its work if the system is not overloaded.

A project that does not have the `project.cpu-shares` resource control defined is assigned one share. Processes in projects with zero shares always run at the lowest system priority (0). These processes only run when projects with nonzero shares are not using CPU resources. The following example shows an entry in the `/etc/project` file that sets the number of shares for the project marketing to five:

```
marketing:100::::project.cpu-shares=(privleged, 5, none)
```

FSS and Processor Sets

The previous discussion of the FSS assumes that all processors are in the same processor set. When processor sets are present, the FSS treats every processor set as a separate partition. CPU entitlement for a project is based on CPU usage in that processor set only. The CPU usage of a project in a processor set does not influence its entitlement in a different processor set. The FSS calculates the proportion of CPU allocated to a project in a processor set by dividing the shares of the project by the number of shares of active projects in the processor set.

For example, assume a system with two processor sets of one CPU each. The database project is assigned two shares, and the marketing project is assigned one share, and both projects have enough processes to use all available CPU. The marketing project is the only project running in the first processor set. Because it is the only project in this set, it is entitled to all of the CPU in the set. Both projects run in the second processor set. The number of active shares in this processor set is three, so the sales database is entitled to two-thirds of the processor set, and the marketing project is entitled to one-third. The marketing project's CPU use in the first processor set does not influence its entitlement in the second processor set.

Timesharing Scheduler

The default scheduler in Oracle Solaris, the timesharing scheduler (TS), tries to provide every process relatively equal access to the available CPUs, allocating CPU time based on priority. The TS does not need to be administered, which makes it easy to use. However, it cannot guarantee performance to particular applications. It should be used if CPU allocation is not required.

For example, if two projects are assigned to an FSS resource pool and they each have two shares, it does not matter how many processes are running in those projects—each of the projects can only access 50 percent of the available CPU. In other words, if one process is running in the sales project and 99 are running in the marketing project, the one process in the sales project can access 50 percent of the CPU and the 99 in the marketing project must share 50 percent of the available CPU resources.

In a TS resource pool, the CPU is allocated per process. So, in the example above, the one process in the sales project has access to only one percent of the CPU, while the 99 processes in the marketing project have access to the other 99 percent of the available CPU resources.

Resource Pools and Dynamic Reconfiguration Operations

Dynamic reconfiguration allows the hardware to be reconfigured while the system is running. A dynamic reconfiguration operation can increase, reduce, or have no effect on a given type of resource. Because dynamic reconfiguration can affect available resource amounts, the pools facility must be included in these operations. When a dynamic reconfiguration operation is initiated, the pools framework acts to validate the configuration.

If the dynamic reconfiguration operation can proceed without causing the current pools configuration to become invalid, then the private configuration file is updated. An invalid configuration is one that cannot be supported by the available resources.

If the dynamic reconfiguration operation causes the pools configuration to be invalid, then the operation fails and the administrator is notified by a message to the message log. The configuration can be forced to complete using the dynamic reconfiguration force option. The pools configuration is then modified to comply with the new resource configuration. For information on the dynamic reconfiguration process and the force option, see the dynamic reconfiguration user guide for the Oracle hardware on which the pools are running.

Resource Pools Example

In this example, five applications are consolidated onto a single system with a single instance of Oracle Solaris and five resource pools. The target applications have resource requirements that vary, different user populations, and different architectures. Currently, each application exists on a dedicated server that is designed to meet the requirements of the application as described in Table 1. This is simply an example to illustrate a concept. Please see the administration guides for details on how to create and manage resource pools.

TABLE 1. EXAMPLE APPLICATIONS FOR CONSOLIDATION

APPLICATION DESCRIPTION	CHARACTERISTICS
Application server	Exhibits negative scalability beyond two CPUs
Database instance for application server	Heavy transaction processing
Application server in test/development	GUI-based, with untested code execution
Transaction processing server	Primary concern is response time
Standalone database instance	Processes a large number of transactions and serves multiple time zones

The following configuration is used to consolidate the applications onto a single system:

• The application server is configured with a two-CPU resource pool.

• The database instance for the application server and the standalone database instance are consolidated onto a single resource pool of at least four CPUs. The standalone database instance is guaranteed 75 percent of that resource.

- The test and development application server requires the interactive scheduling class to help ensure user interface responsiveness. Memory limitations are imposed to lessen the effects of bad code builds.

- The transaction processing server is assigned a dedicated resource pool of at least two CPUs, to minimize response latency.

Constraints can be established that allow the processor resource to be transferred to pools where the resource is required:

- The `wt-load` objective is set to allow resource sets that are highly utilized to obtain greater resource allocations than sets that are utilized less.

- The `locality` objective is set to `tight`, which is used to maximize processor locality.

An additional constraint to prevent utilization of any one pool from exceeding 80 percent is also applied. This constraint helps ensure that applications can access the resources they require. The `/etc/project` file for this configuration contains

```
user.app_server:2001:Production Application Server:::project.pool=appserver_pool
user.app_db:2002:App Server DB:::project.pool=db_pool;project.cpu-shares(privileged,1,deny)
development:2003:Test and development::staff:project.pool=dev_pool;process.max-address-
space=(privileged,536870912,deny)
user.tp_engine:2004:Transaction Engine:::project.pool=tp_pool
user.geo_db:2005:EDI DB:::project.pool=db_pool;project.cpu-shares=(privileged,3,deny)
```

An input file named `pool.host` is created for this example to configure the required resource pools:

```
create system host
create pset dev_pset (uint pset.min = 0; uint pset.max = 2)
create pset tp_pset (uint pset.min = 2; uint pset.max=8)
create pset db_pset (uint pset.min = 4; uint pset.max = 6)
create pset app_pset (uint pset.min = 1; uint pset.max = 2)
create pool dev_pool (string pool.scheduler="IA")
create pool appserver_pool (string pool.scheduler="TS")
create pool db_pool (string pool.scheduler="FSS")
create pool tp_pool (string pool.scheduler="TS")
associate pool dev_pool (pset dev_pset)
associate pool appserver_pool (pset app_pset)
associate pool db_pool (pset db_pset)
associate pool tp_pool (pset tp_pset)
modify system tester (string system.poold.objectives="wtload")
modify pset dev_pset (string pset.poold.objectives="locality tight; utilization < 80")
modify pset tp_pset (string pset.poold.objectives="locality tight; 2: utilization < 80")
modify pset db_pset (string pset.poold.objectives="locality tight;utilization < 80")
modify pset app_pset (string pset.poold.objectives="locality tight; utilization < 80")
```

The configuration for this example is updated by using the `pool.host` file as input to the `poolcfg` command, and the `pooladm` command makes this configuration active:

```
#poolcfg -f pool.host

#pooladm -c
```

The configuration for this example is shown in Figure 10. In order to isolate the applications in each of these pools, projects can be configured in zones within the pools, as discussed in the next section. (Note: The project to the pool link is ignored if zones are used.)
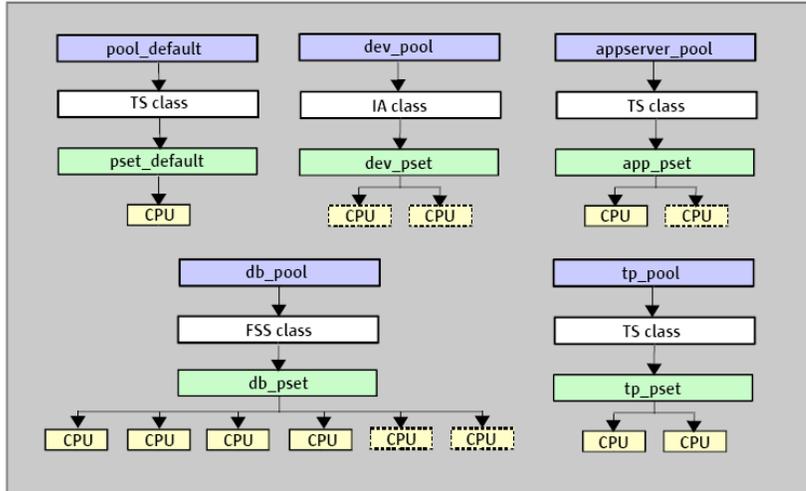

Figure 10. This figure illustrates pool configuration for multiple applications on one system.

Assigning Zones to Resource Pools

Nonglobal zones can be associated with only one resource pool. If a scheduling class is set for a pool, any nonglobal zones that are associated with it use that scheduling class by default. The following example associates a pool with the zone sales:

```
zonecfg:sales> set pool=db_pool
```

Because the db_pool is set with the FSS, the sales zone inherits that scheduling class by default.

Assigning CPU Shares to Zones

The `zone.cpu-shares` resource control can be used to limit the CPU usage of zones with respect to other zones sharing the same resource pool. This is set through `zonecfg`(1M). Zonewide resource controls do not take effect if only set in the `project` file. The global zone controls FSS CPU shares through the zonewide resource control `zone.cpu-shares`. The `project.cpu-shares` control set for each zone further subdivides the shares set through the zonewide control. A zonewide resource control is set through the `zonecfg` utility.

FSS shares are hierarchical. In Oracle Solaris 10, the global zone is assigned one share by default. Therefore, if a nonglobal zone is assigned two shares and the global zone has one share, the ratio is 2:1. If the ratio of nonglobal zones to global zone shares is too high—for example, 20:1—the processes running in the global zone may be negatively affected.

The FSS can be used in conjunction with processor sets (resource pools) to provide more fine-grained controls over allocations of CPU resources among projects that run on each processor set than would be available with processor sets alone. The FSS treats processor sets as entirely independent partitions, with each processor set controlled independently with respect to CPU allocations.

Because projects are not competing for the same resources, the CPU allocations of projects running in one processor set are not affected by the CPU shares or activity of projects running in another processor set. Projects only compete with each other if they are running within the same processor set.

In the example below, assume that there are more users in the sales zone and that they perform more CPU-intensive transactions than the users in the marketing zone. To assign the sales zone twice the amount of CPU as the marketing zone, the number of `zone.cpu-shares` of the sales zone is set to twice the number of the marketing zone, as shown in Figure 11.

```
global # zonecfg -z sales
zonecfg:sales> add rctl
zonecfg:sales:rctl> set name=zone.cpu-shares
zonecfg:sales:rctl> add value (priv=privileged, limit=20, action=none)
zonecfg:sales:rctl> end
zonecfg:sales> exit
global # zonecfg -z mkt
zonecfg:mkt> add rctl
zonecfg:mkt:rctl> set name=zone.cpu-shares
zonecfg:mkt:rctl> add value (priv=privileged, limit=10, action=none)
zonecfg:mkt:rctl> end
zonecfg:mkt> exit
```

In this example, the shares only take effect after the zone is rebooted. To set the `zone.cpu-shares` resource control on a running zone, the `prctl`(1) command can be used. (Note: This change only persists until the next zone boot. Also, shares only take effect if the FSS is activated.)

```
global # prctl -n zone.cpu-shares -r -v 20 -i zone sales
global # prctl -n zone.cpu-shares -r -v 10 -i zone mkt
```
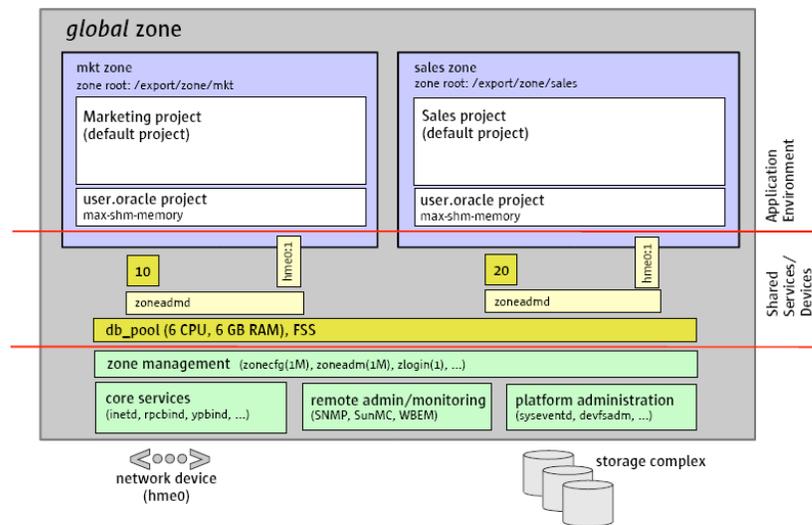

Figure 11. This figure illustrates the example of a sales zone assigned twice the amount of CPU as a marketing zone.

Dividing CPU Shares Between Projects Within Zones

The amount of CPU consumed inside a zone is controlled by the `project.cpu-shares` resource control. The nonglobal zone administrator can control CPU consumption inside the zone because zones have their own project database. To demonstrate this, the CPU shares of the default project are set to 40 and the shares for the `user.oracle` project are set to 10, assigning the default project four times more CPU resources than the second project (see Figure 12). This may be useful to keep database administrators from consuming more resources than they need, such as when executing a `find` command.

(Note: If shares are not specified for a project, it defaults to one share. Shares are only used if processes are running in the zone.)

```
sales # cat /etc/project
system:0::::
user.root:1::::
noproject:2::::
default:3::::project.cpu-shares=(privileged, 40, none)
group.staff:10::::
user.oracle:100:Oracle::project.max-shm-memory=(privileged, 2147483648, deny):project.cpu-
shares=(privileged, 10, none)
```
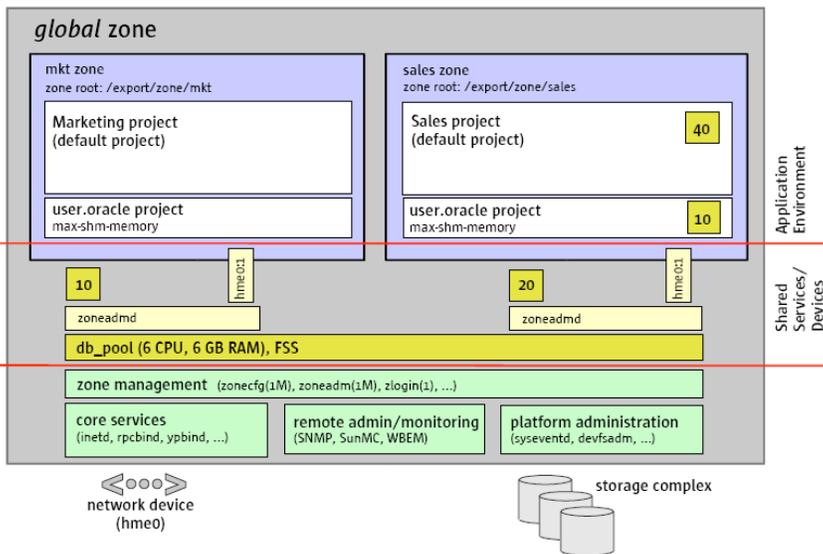

Figure 12. This figure illustrates dividing CPU shares between projects.

In this example, with only the sales and marketing zones active, the sales zone is entitled to

$$\text{zone.cpu-shares}_{sales}/(\text{zone.cpu-shares}_{sales} + \text{zone.cpu-shares}_{mkt}+ = 20/(20 +10) * 100 = 66 \text{ percent}$$

of the available CPU. This 66 percent is then distributed among the projects in that zone. The default project is entitled to

$$\text{project.cpu-shares}_{default}/(\text{project.cpu-shares}_{default} + \text{project.cpu-shares}_{user.oracle}) * 66 \text{ percent} = 40/(40 +10) * 66 \text{ percent} = 53 \text{ percent}$$

of the total CPU available. The `user_oracle` project is entitled to

```
10/(40 +10) * 66 percent = 13 percent
```

of the total CPU.

Examples

This section contains examples of consolidating applications using Oracle Solaris Containers. The first example consolidates multiple applications using one container per application. The second example demonstrates a test and development environment with different versions of an application in separate containers. The third example illustrates how containers can be used to upgrade applications easily, quickly, and inexpensively. The fourth example shows the advantages of using containers for a multiple-user environment. The fifth example builds on the first example with multiple applications sharing a single container. And the sixth example illustrates how flexible tier 0 applications can be deployed using containers to reprovision systems on the fly.

Single Applications

Single-application containers use the isolated namespace to shrink-wrap or isolate an application in its own environment, where only the needed services, such as `ssh` and `inetd`, are turned on. This example is particularly useful for replicating very similar services such as Web servers. In this case, every zone has its own port 80, and zone shares can make this environment very efficient. These applications may not require a whole machine, but they are run on separate systems in order to isolate them.

Many customers have their applications isolated on separate systems because they are concerned about interference from other applications. Examples are DNS, e-mail, and Web servers. In the case of a Web server, for instance, the concerns are namespace isolation and security risks. The Web server runs by default on port 80, and only one Web server can listen to port 80, making it difficult to run multiple Web servers on a single system. In addition, port 80 is a privileged port that requires root privileges to access. This can open up security risks because the Web server administrator knows the root password, and the Web server is quite often running as root or with root privileges. This can be a problem if the system is compromised. These issues can be addressed by putting each application into its own container where every zone has its own set of ports (including port 80) and its own root user that is isolated to that zone.

A DNS server is similar in that it is a core service that needs to be isolated in case it is compromised and to prevent something else from bringing it down. For example, if sendmail is running on the DNS server and it is hacked and the hacker becomes root, they can either change the DNS server or bring it down. This is why many DNS servers only run DNS services and nothing else. However, DNS servers usually require very few resources because they are not frequently accessed. Thus, it would be more cost effective to be able to consolidate the DNS server with other applications. With containers it is

possible to run DNS safely with minimal services that cannot be compromised by access to other applications or the applications themselves.

Mail servers, especially if using sendmail, tend to be more easily compromised, which means they could bring down the server. If run in an isolated container it cannot affect the rest of the system even if it is hacked.

Figure 13 illustrates a system with single-application containers for DNS, mail, and two Web server zones. There are three aspects that should be considered for each zone—the services it is running, the devices it is importing, and the resources the zone is assigned and possibly sharing.
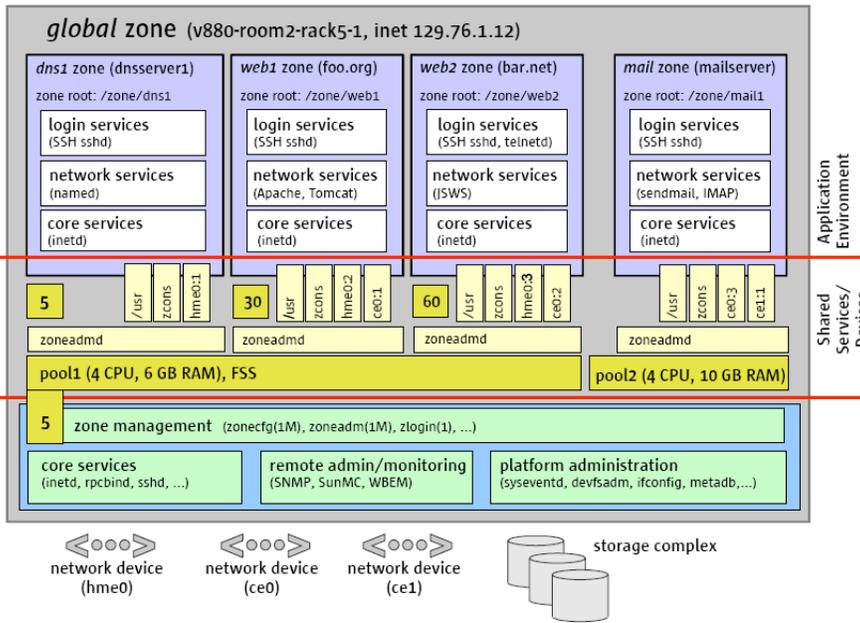

Figure 13. This figure illustrates single applications in separate containers.

The DNS zone only has a few services running—`sshd, named, and inetd`. All other services that could be vulnerable to attack, including `telnet` and `ftp`, are turned off. The service manager feature of Oracle Solaris can help with this by profiling the services that should be running on a zone and helping to ensure that only those services are started or restarted in the event of failure. The DNS zone is also importing `/usr`, which is all of the `/usr` directory from the global zone, as well as `zcons` to enable the global zone administrator to monitor the zone from a virtual console. And it is importing `hme0`, enabling the name service with whatever IP address it is assigned to listen on `hme0`. Any traffic on `ce0` or `ce1` (the Gigabit Ethernet ports) is not visible to this zone and vice versa. Finally, the DNS zone is running on pool1, which is configured with a subset (four CPUs) of the eight CPUs in the system and 6 GB of memory. The FSS is used to distribute CPU shares between the three zones running on pool1. In this case, 5 shares are assigned to the global zone for zone management processes, 5 to the DNS zone, 30 to the web1 zone, and 60 to the web2 zone. This provides a guaranteed amount of resources to each zone.

For the web1 zone, the services running are `sshd`, Apache, Tomcat, and `inetd`. The same services are running in the web2 zone with the addition of `telnetd`, which makes this zone more vulnerable. The web2 zone is also using a different Web server (Java System Web Server), illustrating the fact that the zones can run different Web server applications all listening to their own port 80. In addition to importing `/usr`, `zcons`, and `hme0` (which could be the management network or name service network), both zones also import `ce0`, which could be the external Web network.

If all three zones on pool1 are active at the same time, the FSS allocates 5 shares of the four CPUs to the DNS zone, 30 shares to the web1 zone, and 60 shares to the web2 zone. If one of the zones is not active—for instance, the DNS zone—the remaining zones can be allocated shares of the four CPUs at a ratio of 30:60, enabling full utilization of the CPU resources while guaranteeing a minimum amount of resources for each zone.

Pool2 contains four CPUs with only the mail zone configured on it. The services running in this zone are `sshd`, `sendmail`, `imapd`, and `inetd`, which are the core services required to run a mail server. It imports `/usr`, `zcons`, `ce0`, and `ce1`, which in this case could be an internal mail network. Because it is on a dedicated pool, it is not necessary to run the FSS. Architecturally, this zone is acting as a separate machine. If it is not using all of its CPU cycles, they are not available to any other zones, as is the case in pool1. On the other hand, if this zone is compromised, it does not affect the applications running elsewhere on the system.

The zone management processes run in the default pool, in this case pool1, and need to be allocated some amount of CPU resources in order to run. In this example, the zone management processes are allocated five shares of the CPU resources in pool1.

One important thing to note is that each of these zones can be rebooted without affecting the others, and that reboot times are very short because only the zone is rebooting, not the entire system (which includes storage).

Without zones, a reboot of a system with multiple applications shuts down all of the applications on that system.

Test and Development Systems

In typical development environments, a new machine with a fresh OS is required for every build of an application. For large software projects, with weekly build schedules, this entails a lot of systems to test the new build on multiple versions of OSs, and to keep the last several builds. For example, if a development group is testing software on four versions of OS and has five builds, they need 20 systems. This may be cost prohibitive for smaller organizations, forcing them to cannibalize existing systems. The other solution is to go to purchasing for a purchase order and order a new system that may take weeks or months to arrive, which is costly in terms of budget and time. Both approaches also consume an hour to several hours—depending on the number of patches and the time it takes to test it—each time a new or repurposed system is installed with a fresh OS.

Oracle Solaris Containers solve both the cost and time issues by enabling test and development systems that share the same version of the OS to be consolidated. Multiple developers can share the

same system with root privileges that allow them to reboot their own zone without affecting the other developers on the system, and multiple builds of an application can be housed on the same system, in either active or inactive states. Therefore, in the example above, only 4 systems are required instead of 20 systems. Consolidating development and test environments onto a single system can also be less expensive in terms of software licences and administrative cycles because only one system, OS, and set of tools needs to be maintained, as opposed to one system, OS, and set of tools for every developer and test bed. Again, using the example above, the total number of systems in a development environment can be further reduced if the developers are running in containers on the same system as the test beds.

Containers enable developers to recreate a *clean* system to quickly try new things, without the need to purchase a new system and recreate an exact replica of the environment. It also allows them to easily set up containers for different versions of applications that can then be quickly rolled back or destroyed as needed.

The example in Figure 14 illustrates a consolidated test and development environment with three different versions of an application running in three separate zones. Each zone shares `/usr`, as well as `/data`, so that each version of the application is tested against the same data.

In this example, each zone is running a different version of the application, but in reality probably only one zone is active at a time, so there is no need to configure the FSS. In this case, a zone is created to test version 2.0 with Apache, Oracle8*i* Database, and Java System Application Server 6. In the version 2.1 zone, everything is the same except that the developer has decided to test the application using Oracle9*i* Database. Normally this would require the developer to go to purchasing to acquire a new system and then take the time to set it up. With Oracle Solaris Containers, it only takes approximately 10 minutes to set up a new zone to test a change in the services stack, or in the case of the version 3.0 zone, the entire services stack.
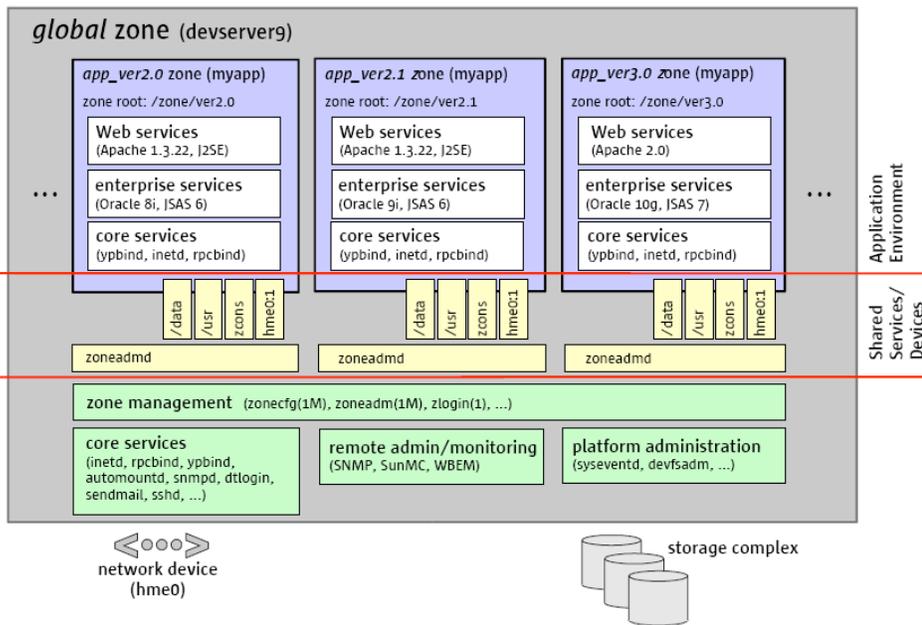
Figure 14. This figure illustrates different versions of an application development environment on the same system.

Each zone in this example is importing /data, which could be mounted read/write and contain the developer tools, application data, and the application itself. Thus, each zone sees and uses the same data. This helps increase consistency and decrease the amount of time it would take to copy this data to separate systems.

In addition, if only one of these zones is active at a time, they could all have the same IP address, making it easier to use a third-party test application against each version of the application because the test application does not need to be reconfigured for each new version. It is a simple matter of deactivating one zone, activating the next, and restarting the test suite using the same IP address.

It is important to note that only one version of the OS can be running per system, unless it is a system that supports domains. Therefore, the example at the beginning of this section uses only four systems, one for each version of the OS. This is still one-fifth of the systems required using traditional methods and is much less expensive in terms of licences, hardware, and time to acquire and administer. Another benefit is that multiple versions of builds can be kept for longer periods of time in inactive zones. Of course, the biggest advantage is that developers can do a lot more with a smaller budget and in less time.

Upgrading Production Systems Via Containers

Oracle Solaris Containers can also be used to upgrade production systems quickly and easily. In this example, version 2.0 of an application server is running in a single zone on the production system with hme0 connected to the management network, ce0 connected to the Web network, and ce1 connected to the database network, as illustrated in Figure 15.
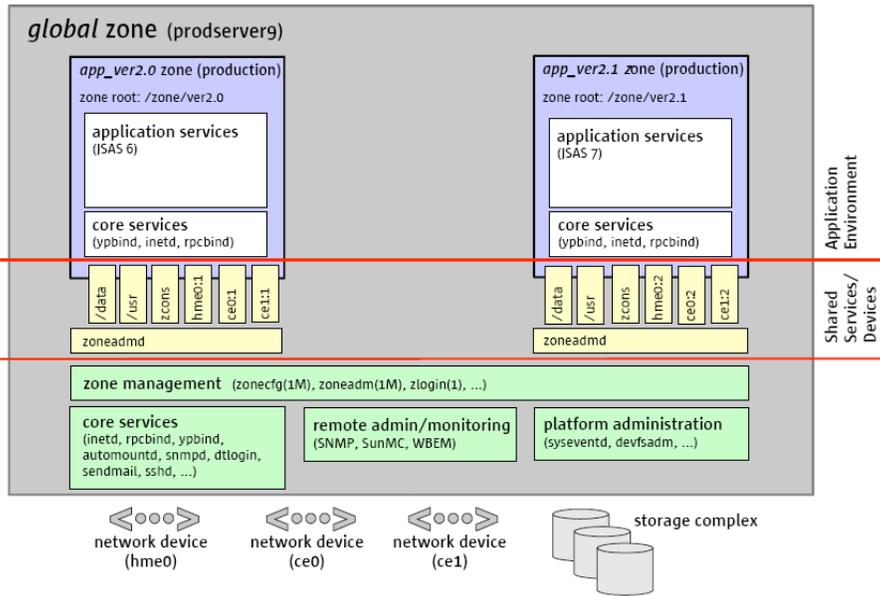
Figure 15. This figure illustrates the process of migrating to a new version of the production system.

An administrator can log in through the management network, create the version 2.1 zone, and copy in the new data without affecting the running production application. Once the zone is set up, the administrator can configure the new zone with the version 2.0 zone's IP address, halt the version 2.0 zone, and boot the version 2.1 zone, which is now running as the production server with the correct IP address.

The benefits here are twofold—cost and time. First, the production system can be upgraded without purchasing a new system; physically installing it; loading the OS, patches, and data; and recabling switches, storage systems, and so on. In addition, all of the volume management and file system management tasks only need to be performed once, by the global administrator. This can save a great deal of time and the cost of additional volume management licenses.

Secondly, because it only takes a matter of minutes to create a new zone, the production system can be upgraded in a fraction of the time it would take with traditional methods. Finally, if the new version exhibits problems, the previous version can be reactivated in minutes without physically recabling everything. This method of upgrading system applications has the potential to greatly increase the efficiency and ease of upgrading remotely administered systems as well.

Multiple-User Environments in Separate Containers

Oracle Solaris Containers are ideal for any university or corporation with users that require their own systems for either guaranteed performance or root privileges. In a university, many students require root privilege in order to carry out their assignments. This normally requires the university to purchase and maintain a system for every student with these requirements, which is both costly and time consuming. With Oracle Solaris Containers, all of the student environments can be housed on a single

system, dramatically reducing costs, including OS and other licensing fees. In addition, only one version of the OS, tools, and applications needs to be maintained, rather than possibly hundreds.

In Figure 16, each student is assigned a zone that imports `/usr` and `/tools`, which could be mounted read only and consist of all of the compilers and tools required, as well as any other services the students need for their activity but do not need to modify. Two network devices are also imported to provide access to internal and external networks. Each student has the freedom of root privileges in their zone and can reboot, install new software, and generally experiment without affecting the other students. In this environment, the cost of adding an incremental student is very low, and it only takes a few minutes to create a new zone.
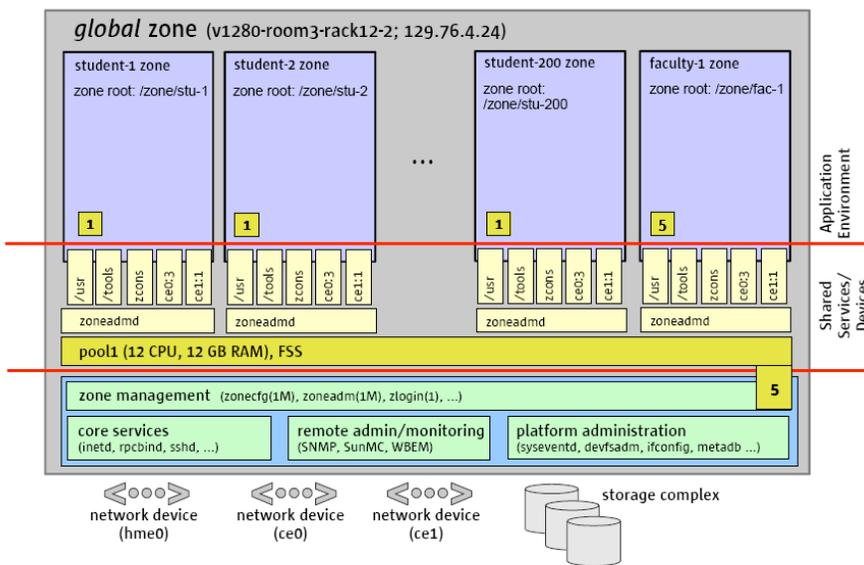

Figure 16. This figure illustrates maintaining multiple students on one system.

To keep one student from monopolizing the entire system, the FSS is utilized, giving each student one share of the system resources in pool1. (Note: A pool is not required to run FSS.) This example also includes faculty and students that are allocated five shares of the system to perform more-advanced tasks. Notably, the global zone administrator assigns each zone a unique IP address. This IP address cannot be changed within the zone. In other words, a student cannot change the IP address in their zone and re-plumb the address in order to spoof another student's network traffic or gain access to another system on the network. This is one of the security isolation features inherent in Oracle Solaris Containers. It provides the added benefit that only one network-savvy administrator is required, rather than the several it might take to administer hundreds of systems.

Multiple Applications in One Container

In general, it is recommended that each application occupy its own zone. However, applications that need to share memory or a namespace can be installed in the same zone. For example, an Oracle

database and its application server that uses direct access can be installed in the same zone. Resources in the zone can be allocated between the applications with FSS, memory capping, and `shmem` settings (per project).

The example in Figure 17 shows a system with three containers, one with a single application and two with multiple applications per container. The system is first divided into two pools. Pool1 is assigned four CPUs and 10 GB of memory with the default scheduler (the TS). Pool12 is allocated eight CPUs and 4 GB of memory that are configured to be managed with the FSS.

The mail zone is running with only `sshd`, `sendmail`, `imapd`, and `inetd` running. It is isolated to its own zone so that if it should be compromised, it cannot affect the other zones on the system. It is importing `/usr`, `zcons`, and `ce0` and `ce1` for access to the internal and external networks, respectively. Because it is running on pool1 with the TS, it has access to all of the resources in the pool. In addition, zone management runs in pool1 because it is the default pool.
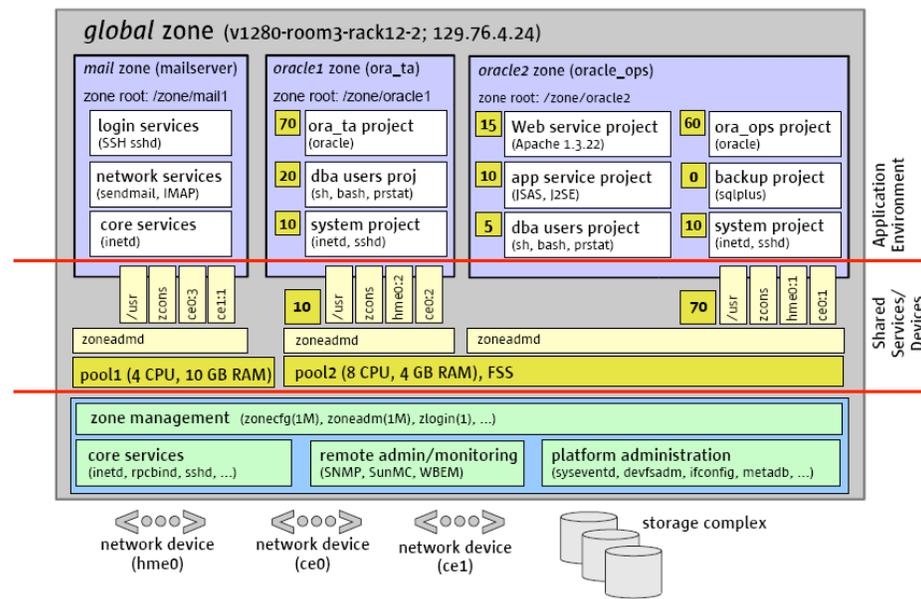

Figure 17. This figure illustrates applications sharing the same container.

The two zones in pool2 use a more complex resource management scheme than previously discussed. Therefore, this type of configuration should be utilized on more stable systems. Each zone in pool2 is assigned a number of CPU shares (10 and 70), which means that oracle1 zone is allocated one-eighth or one CPU and oracle2 is allocated seven-eighths or seven CPUs. The shares are further divided between the projects in each zone. The oracle1 zone contains three projects: one for oracle, one for dba users, and one for system services. The projects are assigned 70, 20, and 10 shares, respectively, equating to a ratio of 7:2:1. In other words, the ora_ta project is allocated .7 of one CPU, and so on. Separating dba users and services into projects and assigning them shares helps ensure that runaway processes such as the `find` command cannot negatively affect the performance of the database.

The oracle2 zone is a little more complex, with six separate projects—including a Web server and application server—and shares assigned to each. Note that the backup project is assigned zero shares,

allowing it to only run when other projects in the zone are using less than their full amount of shares. This allows it to run in the background and dynamically use unused CPU cycles. In addition, if the oracle1 zone is inactive, projects in the oracle2 zone can utilize the unused CPU cycles from that zone as well. This enables the system to be more fully utilized and can help provide a better return on investment (ROI).

Each of the database zones on this system is configured with two network interfaces: `hme0` for management and `ce0` for access to the internal network. The global administrator assigns IP addresses and also configures the storage volumes, RAID levels, and file systems for the system.

In this example, security is maintained between zones via the process isolation that Oracle Solaris Containers provides. Access between applications within the same zone can be restricted using Oracle Solaris' user and process rights management features (discussed in a later section).

Flexible Scalability with Containers

Another excellent use for Oracle Solaris Containers is to increase the flexibility to scale tier 0 applications. For instance, many Websites run Web servers behind a pair of virtual load balancers for redundancy, which are frequently on the same system. For this example, assume that the load balancers serve two separate Websites, A and B. Website A generally requires three Web servers, while Website B requires five Web servers. If Website A is less busy than B, a server from A can be reprovisioned to run on Website B by rebooting the server to boot up a Website B environment or stopping the Web server, running some scripts, and then restarting the Web server. Then it needs to be unlinked from one load balancer and linked into the other. This can all take time.

A faster method is to preconfigure two zones on each Web server, one for Website A and one for Website B. Then the only thing necessary to transfer a server to another Website is to tell the load balancer to point to a different IP address, drastically reducing the time to reprovision a server. An even faster method is to keep both zones active and connected to the load balancer and simply move CPU shares from one zone to another. With either method, scaling between one and eight systems for either Website is a fast and easy process.

Managing Other Resources

Resource management helps ensure that the applications that are consolidated and running on a single system (as well as single-application servers) meet required response times and service levels. It can also be used to increase resource utilization. Reserve capacity can be used during off-peak periods by categorizing and prioritizing usage, often eliminating the need for additional processing power. This section provides details on managing memory, network bandwidth, and storage resources between multiple applications running on the same system.

Managing Memory Resources

The resource capping daemon `rcapd` can be used to regulate the amount of physical memory that is consumed by processes running in projects that have resource caps defined. To use `rcapd` in a zone, a project entry must be added and the daemon must be configured in each zone where the daemon is to run. `rcapd` does not act on processes in zones other than the one in which it is running.

It is also possible to regulate the amount of memory a project can consume by configuring `rcap` in the project database—for example, `rcap.max-rss`. However, while the kernel synchronously enforces resource controls, resource caps are asynchronously enforced at the user level by the resource cap enforcement daemon `rcapd`.

For software that needs to start automatically and be capped by running as a project, the RC startup script must be modified. First, rename the original RC script and disable it from running upon boot by changing the initial capital "S" to a lowercase "s":

```
server# mv /etc/rc3.d/S50apache /etc/rc3.d/s50apache
```

Then create a new s50apache script that looks like this:

```
#!/bin/ksh
newtask -p tq /etc/rc3.d/sapache $1
```

This causes the Apache Web server to start up as a project so that its resources can be controlled. Another method is to simply modify the internals of the RC script so that when the application is started, it starts under the resource manager. (Note: In general, an additional 40 MB of RAM per zone is suggested, but it is not required on a system with sufficient swap space.)

How rcapd Works

The `rcap` daemon repeatedly samples the resource utilization of projects that are configured with physical memory caps. The administrator specifies a sampling interval used by the daemon. When the system's physical memory utilization exceeds the threshold for cap enforcement and other conditions are met, the daemon takes action to reduce the resource consumption of projects with memory caps to levels at or below the caps.

The virtual memory system divides physical memory into segments known as *pages*. Pages are the fundamental unit of physical memory in the Oracle Solaris memory management subsystem. To read data from a file into memory, the virtual memory system reads in one page at a time, or *pages in* a file. To reduce resource consumption, the daemon can *page out*, or relocate, infrequently used pages to a swap device, which is an area outside of physical memory.

The daemon manages physical memory by regulating the size of a project workload's *resident set* relative to the size of its *working set*. The resident set is the set of pages that are resident in physical memory. The working set is the set of pages that the workload actively uses during its processing cycle. The working set changes over time, depending on the process's mode of operation and the type of data that is processed. Ideally, every workload can access enough physical memory to enable its working set to remain resident. However, the working set can also include the use of secondary disk storage to hold the memory that does not fit in physical memory.

Determining Cap Values

If a project cap is set too low, there might not be enough memory for the workload to proceed effectively under normal conditions. The paging that occurs because the workload requires more memory can have a negative effect on system performance.

Projects that have caps set too high can consume available physical memory before their caps are exceeded. In this case, physical memory is effectively managed by the kernel and not by `rcapd`. In determining caps on projects, consider these factors:

- **Impact on I/O system.** The daemon can attempt to reduce a project workload's physical memory usage whenever the sampled usage exceeds the project's cap. During cap enforcement, the swap devices and other devices that contain files that the workload maps are used. The performance of the swap devices is a critical factor in determining the performance of a workload that routinely exceeds its cap. The execution of the workload is similar to running it on a machine with the same amount of physical memory as the workload's cap.

- **Impact on CPU usage.** The daemon's CPU usage varies with the number of processes in the project workloads it is capping and the sizes of the workloads' address spaces. A small portion of the daemon's CPU time is spent sampling the usage of each workload. Adding processes to workloads increases the time spent sampling usage. Another portion of the daemon's CPU time is spent enforcing caps when they are exceeded. The time spent is proportional to the amount of virtual memory involved. CPU time spent increases or decreases in response to corresponding changes in the total size of a workload's address space. This information is reported in the vm column of `rcapstat` output. See the `rcapstat`(1) man page for more information.

- **Reporting on shared memory.** The daemon cannot determine which pages of memory are shared with other processes or which are mapped multiple times within the same process. Because `rcapd` assumes that each page is unique, this results in a discrepancy between the reported (estimated) resident set size (RSS) and the actual RSS. Certain workloads, such as databases, use shared memory extensively. For these workloads, sample a project's regular usage to determine a suitable initial cap value. Use output from the `prstat` command with the `-J` option. See the `prstat`(1M) man page for more information.

The performance of the entire server, including all zones, can degrade when a project reaches its RSS limits. This problem is not contained to a single zone. As a result, caution is advised when capping memory usage.

Shared Memory

In Oracle Solaris 10, shared memory settings are virtualized into a set of resource controls that can be set on a per-project basis (and many do not need to be tuned at all, because the defaults are more reasonable), so one of the more common changes of `/etc/system` is no longer necessary. See *System Administration Guide: Solaris Containers, Resource Management, and Solaris Zones* for a complete list of the resource controls that can be set within a project.

Managing Network Resources

In a zones environment, the zones can communicate with each other over the network. Basic communications between zones is accomplished by configuring at least one logical network interface for each zone. The IP addresses assigned to these interfaces identify each zone. An application running in one zone cannot observe the network traffic of another zone. This isolation is maintained even though the respective streams of packets travel through the same physical interface.

Currently two types of network connections are available inside a zone, *Shared IP Non-Global Zones* and *Exclusive IP Non-Global Zones*. With shared stack

Shared IP Non-Global Zones

The zones all have separate *bindings*, or connections, and they can all run their own server daemons. These daemons can listen on the same port numbers without any conflict. The IP stack resolves conflicts by considering the IP address for incoming connections.

Each zone has its own set of binds. Each zone can be running the same application listening on the same port number without binds failing because the address is already in use. Bindings between upper-layer streams and logical interfaces are restricted. A stream can only establish bindings to logical interfaces in the same zone. Likewise, packets from a logical interface can only be passed to upper-layer streams in the same zone as the logical interface. Each zone can run its own version of `inetd`, `sendmail`, `apache`, `sshd`, `telnetd`, and so on. These processes can also be turned off to increase security, i.e., sendmail can be a security hazard but not in zones that are not running it.

Each zone has complete control of the parameters related to its network identity, such as its nodename, hostname, naming service, and RPC domainname. A shared IP non-global zone does not allow plumbing of network interfaces inside a zone or using IP addresses that are not assigned by the global zone administrator. Also, many of the global TCP/IP parameters that can be set via `ndd(1M)` are global in nature, in that they affect all zones. However, many of these no longer need to be changed in Oracle Solaris 10 with the advent of its new networking stack.

Exclusive IP Non-Global Zones

An exclusive-IP zone has its own IP-related state and tuning variables. The zone is assigned its own set of data-links when the zone is configured. This is achieved by assigning a specific physical NIC port to the zone i.e. a data-link name must be assigned exclusively to a single zone. Essentially, while providing more control than shared IP non-global zones is limited to the number of physical ports available to the system.

 Exclusive-IP zones have separate TCP/IP stacks, so the separation reaches down to the data-link layer. One or more data-link names, which can be a NIC or a VLAN on a NIC, are assigned to an exclusive-IP zone by the global administrator. The zone administrator can configure IP on those data-links with the same flexibility and options as in the global zone.

There is no internal loopback of IP packets between exclusive-IP zones. All packets are sent down to the data-link. Typically, this means that the packets are sent out on a network interface. Then, devices like Ethernet switches or IP routers can forward the packets toward their destination, which might be a different zone on the same machine as the sender.

Zones other than the global zone have restricted access to the network. The standard TCP and UDP socket interfaces are available, but SOCK_RAW socket interfaces are restricted to Internet Control Message Protocol (ICMP). ICMP is necessary for detecting and reporting network error conditions or using the ping command.

For optimal network performance, make sure that there are enough network cards in the system to support the throughput of all of the applications on the system and provide redundant paths if higher availability is required.

IP Quality of Service

The Oracle Solaris 9 9/02 release introduced new network resource management technology, superseding Oracle Solaris' bandwidth manager feature available for previous Oracle Solaris releases. This technology is usually referred to as IP quality of service (IPQoS). IPQoS is ideal for the controlling, monitoring, and accounting of network resources. IPQoS forms the foundation of managing the network resource aspects of Oracle Solaris Containers with features that can help make network performance more efficient. It is implemented at the IP level of the TCP/IP protocol stack and is configured for the global zone unless nonglobal zone traffic is routed outside of the system.

The Internet Engineering Task Force (IETF) is responsible for several standards drafts and specifications for class-based traffic differentiation, by classifying and marking traffic aggregates. This is referred to as Differentiated Services (DiffServ). Basically, the idea behind DiffServ is to give certain traffic aggregates different packet-forwarding treatment than others. Examples of such aggregates are all Voice over IP (VoIP) traffic, latency sensitive, bulk, and never drop. IPQoS utilizes DiffServ to label traffic to add value to the QoS in switches. For example, it can prefilter NFS versus VoIP traffic with an 802.1D (VLAN) class of service field so that the switches know what to do with the packets.

IPQoS helps to enable workload-centric datacenter management, which is essential for consolidating applications onto a single server. It helps ensure that a group or application does not consume more than its allotted bandwidth. Users can be charged for the exact amount of network resources they consume, and resources can be dynamically assigned to the workloads that require them, when they need them.

Managing Storage Resources

Storage is managed at the global zone level. Each zone is configured with a portion of the file system hierarchy that is located under the zone root. Because each zone is configured to its subtree of the file system hierarchy, a workload running in a particular zone cannot access the ondisk data of another workload running in a different zone.

Zones allow sharing of the file system data, especially read-only data such as executables and libraries. Parts of the file system can be shared between zones in the system by using the read-only loopback file system (LOFS), which allows a directory and its contents to be inserted into another part of the file system. lofs is improved in Oracle Solaris 10 to support read-only mounts, preventing nonglobal zones from writing in the shared directory. This not only substantially reduces the amount of disk space used by each container but also reduces the time to install zones and apply patches, allowing for greater sharing of text pages in the virtual memory system.

It is possible to share some directories with the global zone with read/write permission by defining a file system resource as lofs. But be aware that this sets up a channel where the nonglobal zone can potentially affect the global zone, by exhausting the space on the file system. This can be addressed by restricting space on file systems and/or users to control how much space they can consume.

The nature of the packages installed in the global zone affects the space requirements of the nonglobal zones that are created. The number of packages and space requirements are factors. As a general guideline, a zone requires 100 MB of free disk space per zone when the global zone is installed with all of the standard Oracle Solaris packages. By default, any additional packages installed in the global zone also populate the nonglobal zones. The amount of disk space must be increased accordingly.

There are no limits on the amount of disk space that can be consumed by a zone. The global zone administrator is responsible for restricting space. There are several ways to restrict the size of a zone:

- It can be mounted on a `lofi`-mounted partition. This action limits the amount of space consumed by the zone to that of the file used by `lofi`. For more information, see the `lofiadm`(1M) and `lofi`(7D) man pages.

- Standard partitions of a disk can be used for zone roots to limit per-zone disk consumption.

For more information on options for mounting file systems in nonglobal zones, see *System Administration Guide: Oracle Solaris Containers, Resource Management, and Oracle Solaris Zones.*

Using Oracle Solaris ZFS with Containers

Oracle Solaris 10 includes Oracle Solaris ZFS, a 128-bit file system/volume manager model that builds file systems on top of shared virtual storage pools, making it far less complex to create and delete file systems.

With Oracle Solaris ZFS, a separate volume manager is not needed. Instead, disks are put into a single, automatically optimized storage pool, which can be shared by multiple file systems. This allows for efficient use of storage resources. Space is shared dynamically between the file systems in the pool without the need to grow or shrink the file systems, because blocks are parceled out from the pool of storage devices as file systems request them.

Storage can be added or removed from storage pools dynamically without interrupting service, and the file systems grow and shrink dynamically as users add or remove data. Administrators can set quota and reservations on file systems or groups of file systems in order to prevent unfair use or overuse of the available disk space in the pool.

Oracle Solaris ZFS avoids data corruption by keeping data on the disk self-consistent at all times. It manages data using transaction groups that employ copy-on-write technology to write data to a new block ondisk before changing the pointers to the data and committing the write. Because the file system is always consistent, time-consuming recovery procedures such as `fsck` are not required if the file system is shut down in an unclean manner. This feature can be especially helpful in development environments where frequent reboots are necessary. In addition, it is designed to provide end-to-end 64-bit check summing on all data, helping to reduce the risk of data corruption and loss.

Using Oracle Solaris DTrace with Containers

Oracle Solaris DTrace is a dynamic tracing framework for troubleshooting systemic problems in real time on production systems. DTrace can safely, dynamically instrument the running OS kernel and running applications—without rebooting the kernel and without recompiling (or even restarting) the applications. And, when not explicitly enabled, it has zero effect on the system.

DTrace is zone aware, making it especially useful for troubleshooting problems between applications running in Oracle Solaris Containers. For example, prior to Oracle Solaris 10, it was very difficult to find and rectify performance issues between different systems—such as a Web server, application server, and database server all running on separate systems. With Oracle Solaris 10 and Oracle Solaris Containers, all three of these applications can run in different (or the same) containers on a single system. Using DTrace, it is now possible and easy to pinpoint performance issues or transient problems occurring between applications that may have been extremely difficult, time consuming, and expensive to uncover in the past. In addition, DTrace can be used to ascertain the level of resources an application uses, helping to more finely tune resource allocation on a zone or project level.

Monitoring and Accounting

When running many applications on one system, it is important to constantly monitor the system, resource pools, zones, and projects within the system in order to efficiently manage the resources of the system to best meet the needs of the applications and users on that system. This section provides information on the utilities available in Oracle Solaris to manage each of the resources discussed.

Extended Accounting

The extended accounting subsystem `acctadm(1M)` within Oracle Solaris provides a flexible way to record system and network resource consumption on a task or process basis, or on the basis of selectors provided by the IPQoS `flowacct` module. The extended accounting subsystem collects and reports information for the entire system (including nonglobal zones) when run in the global zone. It labels usage records with the project for which the work is performed, and the global administrator can also determine resource consumption on a per-zone basis.

The `/etc/acctadm.conf` file contains the current extended accounting configuration. The file is edited via the `acctadm` interface, not by the user. The directory `/var/adm/exacct` is the

standard location for extended accounting data. The `acctadm` command can be used to specify an alternate location.

The `wracct(1M)` process writes extended accounting records for active processes and tasks. The files that are produced can be used for planning, charge-back, and billing purposes. With extended accounting data available, it is possible to develop or purchase software for resource charge-back, workload monitoring, or capacity planning. There is also a perl interface to `libexacct` that enables the development of customized reporting and extraction scripts.

Oracle Solaris Auditing in Zones

An audit record describes an event, such as logging into a system or writing to a file. The record is comprised of tokens, which are sets of audit data. By using the `zonename` token, Oracle Solaris auditing can be configured to identify audit events by zone, which is particularly helpful when zones contain identical user IDs. Using the `zonename` token can produce the following information:

- Audit records that are marked with the name of the zone that generated the record

- An audit log for a specific zone that the global administrator can make available to the nonglobal zone administrator

Users, groups, and projects within zones have unique IDs within the zone. Translation in the global zone could pick up multiple users/groups/projects in Oracle Solaris auditing. Therefore, `zone,project` needs to be specified in order to separate the IDs properly.

Monitoring Resource Pools

The `poolstat` utility is used to monitor resource utilization when pools are enabled on the system. This utility iteratively examines all of the active pools on a system and reports statistics based on the selected output mode. The `poolstat` statistics help to determine which resource partitions are heavily utilized. These statistics can be analyzed to make decisions about resource reallocation when the system is under resource pressure.

The `poolstat` utility includes options that can be used to examine only specified pools and report resource set-specific statistics. If `poolstat` is used in a nonglobal zone, information about the resources associated with the zone's pool are displayed. For more information about the `poolstat` utility, see the `poolstat(1M)` man page.

Monitoring Network Usage

The IPQoS `flowacct` module can be used to collect information about traffic flows on networks. For example, source and destination addresses, the number of packets in a flow, and similar data can be collected. The process of accumulating and recording information about flows is called *flow accounting*. The results of flow accounting on traffic of a particular class are recorded in a table of *flow records*. Each flow record consists of a series of attributes. These attributes contain data about traffic flows of a particular class over an interval of time.

Flow accounting is particularly useful for billing clients as defined in their service-level agreements (SLAs). Flow accounting can also be used to obtain flow statistics for critical applications to observe their behavior. For more information on flow accounting see, *"Using Flow Accounting and Statistics Gathering (Tasks)"* in *System Administrator Guide: IP Services.*

Monitoring Capped Memory

Use `rcapstat` to monitor the resource utilization of projects that are configured with memory caps. The rcapstat report includes the project ID of the capped project, the project name, the virtual memory size of all processes in the project, the total RSS of the project's processes, the RSS cap for the project, and so on. See `rcapstat`(1) for more information.

Monitoring Resource Controls

Often the resources that a process consumes are unknown. To obtain more information, try using the global resource control actions that are available with the `rctladm`(1M) command. `rctladm` can be employed to establish a `syslog` action on a resource control such as shared memory. Then, if any entity managed by that resource control encounters a threshold value, a system message is logged at the configured logging level.

For example, it may be necessary to determine whether a Web server application is allocated sufficient CPU resources for its typical workload. The `sar` data could be analyzed for idle CPU time and load average, or the extended accounting data could be examined to determine the number of simultaneous processes that are running for the Web server process.

However, an easier approach is to place the Web server in a task and then set a global action to notify the administrator whenever a task exceeds a scheduled number of lightweight processes (LWPs) appropriate for that application.

Miscellaneous

This section covers a variety of information, including issues of high availability and security.

IP Network Multipathing

IP network multipathing provides failover to an alternate network adapter when multiple network interface cards are connected to the same IP link. It also provides load balancing. All network configuration is performed in the global zone. IP network multipathing is configured in the global zone and then extended into nonglobal zones. If one of the interfaces in the global zone fails, the nonglobal zone addresses migrate to another network interface card.

Security

Network services can be run in a zone. By running network services in a zone, the damage of a security violation is limited to that zone. An intruder who successfully exploits a security flaw in software

running within a zone is confined to the restricted set of actions possible within that zone. The privileges available within a zone are a subset of those available in the system as a whole. This set of actions can be further restricted with Oracle Solaris' process rights management feature.

Oracle Solaris' Process Rights Management

Process rights management in Oracle Solaris helps ensure that applications—even those run with root privileges—are constrained to access resources only in their own containers, limiting process access to resources for fine-grained security control while reducing the risk of successful exploits. Process rights management is a key technology that supports Oracle Solaris Containers, granting processes and applications rights to only the resources they require to perform their work. This limits the resources that can be accessed should a successful intrusion occur.

With process rights management, processes with less-than root privileges can have access to a restricted set of system objects, significantly limiting the damage they can inflict if they are somehow compromised. Process rights management is integrated with user rights management through Oracle Solaris' role-based access control (RBAC) so that administrators can assign rights that constrain user or process actions.

Enhancements to RBAC, referred to in Oracle Solaris 10 as Oracle Solaris user rights management, allow administrators to assign specific access rights to programs and commands for each user, providing strict control over the access rights that both applications and users are allowed to exercise. This reduces the chance of administrative errors or accidental or malicious use of IT resources. For example, a Web server can be given only the specific right to bind to a low-numbered port (such as port 80), allowing it to perform its function while preventing it from doing more.

Process rights management implements a model that allows a process to inherit a set of rights known as its *limit set*. Once it begins executing, it can shrink the set of rights that it allows itself by reducing its *permitted set*. It can further reduce its own set of rights by shrinking an *effective set*. A process may expand and contract its effective set as needed to perform different functions, but its effective set can never be larger than its permitted set, and its permitted set can only shrink—never grow.

A Web server, for example, can run with the root userid, but without full privileges, just long enough to complete a bind to port 80. Then it can change its userid and drop the privilege to bind to low-numbered ports from its effective set. Now the Web server can run as a nonroot process with its limit set containing the bind-to-port-80 right. At no point in this process does the Web server have full access to all system resources.

Internet Protocol Security Architecture

The Internet Protocol Security Architecture (IPsec) provides IP datagram protection. It protects IP packets by authenticating the packets, by encrypting the packets, or by doing both. IPsec is performed

inside the IP module, well below the application layer. The Internet Key Exchange (IKE) protocol is used to manage the required keying material for authentication and encryption automatically.

IPsec can be used in the global zone. However, IPsec in a nonglobal zone cannot use IKE. Therefore, IPsec keys and policy for the nonglobal zones must be managed by running the `ipseckey` and `ipsecconf` commands from the global zone. Use the source address that corresponds to the nonglobal zone that is to be configured.

It is also possible to configure IPsec policy and keys in the global zone. However, it is not possible to use IKE to manage keys in a nonglobal zone. To configure IPsec for a nonglobal zone, manually create keys with the `ipseckey` command.

Traffic between zones hits the loopback before it hits the higher layers of the IP stack where IPsec resides. IPsec can be used between zones if traffic is forced to go out onto the wire via the routing table.

Firewalls such as IPFilter cannot filter traffic between different zones, such as between the zone and the global zone because this traffic is looped back within IP. IP Filtering is the same as IPsec, in terms of hitting the loopback between zones. One way to implement IP Filtering between zones is to put the zones in separate subnets with routers between them so that traffic is forced outside of the system. This introduces a slight trade-off in performance for higher security between applications running in different zones.

If using firewalls, install them in the global zone and then configure specific rules for the zone itself. Packets sent from the zone always have the zone's IP address as the source address, so this property can be used to filter traffic from the zone.

Filtering IP Traffic Between Zones on the Same System

Between two zones on the same system, packet delivery is only allowed if there is a matching route for the destination and the zone in the forwarding table. The matching information is implemented as follows:

- The source address for the packets is selected on the output interface specified by the matching route.

- By default, traffic is permitted between two zones that have addresses on the same subnet. The matching route in this case is the interface route for the subnet.

- If there is a default route for a given zone, where the gateway is on one of the zone's subnets, traffic from that zone to all other zones is allowed. The matching route in this case is the default route.

- If there is a matching route with the RTF_REJECT flag, packets trigger an ICMP unreachable message. If there is a matching route with the RTF_BLACKHOLE flag, packets are discarded.

The global zone administrator can use the `route` command options described below to create routes with these flags:

```
-reject RTF_REJECT Emit an ICMP unreachable message when matched
-blackhole RTF_BLACKHOLE Silently discard packets during updates
```

Naming Services

Files used by naming services reside within a zone's own root file system view. Thus, naming services in different zones are isolated from one another, and the services can be configured differently.

Exclusive-Use Devices

It may be desirable to assign specific devices to specific zones. Allowing unprivileged users to access specific devices could permit those devices to be used to cause a system panic, bus resets, or other adverse effects. Before making such assignments, consider the following issues:

- Before assigning a SCSI tape device to a specific zone, consult the sgen(7D) man page.

- Configuring a physical device into more than one zone risks creating a covert channel between zones. Global zone applications using such a device risk compromise or data corruption by a nonglobal zone.

In general, applications that use privileged operations that affect the system as a whole should not be installed in a container. Examples of these operations are setting the system clock, locking down physical memory, or connecting to raw devices. The following utilities do not work within a nonglobal zone, because they rely on direct access to hardware devices: eeprom, prtconf, and prtdiag.

In addition, the /lib, /platform, /sbin, and /usr directories are mounted in a zone using a read-only loopback file system. These file systems are accessed from a nonglobal zone in this manner so that the nonglobal zone cannot affect the system as a whole. Additionally, there are certain devices that simply do not exist within a nonglobal zone, such as /dev/kmem and /dev/ip.

Any application requiring a loadable kernel module may not be installed in a nonglobal zone. These applications must run in the global zone. This includes but is not limited to

- Volume managers that require a loadable kernel module

- Firewalls that require a loadable kernel module

- Other applications that require a loadable kernel module

NFS Servers

NFS mounts established within a zone are local to that zone. The mounts cannot be accessed from other zones, including the global zone, which means that a nonglobal zone cannot act as an NFS server. The mounts are removed when the zone is halted or rebooted.

As documented in the mount_nfs(1M) man page, an NFS server should not attempt to mount its own file systems. Thus, a zone cannot NFS-mount a file system exported by the global zone. NFS mounts from within a zone behave as if mounted with the nodevices option.

Conclusion

IT departments today are tasked with increasing service levels while reducing costs. Businesses are eager to consolidate applications onto fewer servers, but they must be careful to isolate these applications to provide adequate resources and security. Oracle Solaris Containers are designed to provide a complete, isolated, secure runtime environment for applications. Additionally, they give administrators almost unlimited flexibility to assign and isolate resources to particular containers, and in many cases can be used to allocate resources within a single container. With all of these resource management capabilities, organizations can consolidate many applications onto one server, helping to reduce operational and administrative costs while increasing availability.

References

For more information about Oracle Solaris Containers, visit:
http://www.oracle.com/technetwork/server-storage/solaris/containers-169727.html
For Oracle Solaris Product Documentation, visit:
http://www.oracle.com/technetwork/server-storage/solaris/documentation/index-jsp-135724.html

ORACLE®

Oracle is committed to developing practices and products that help protect the environment

Consolidating Applications with
Oracle Solaris Containers
July 2011

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com

SOFTWARE. HARDWARE. COMPLETE.