Sun microsystems

WORKING WITH SOLARIS™ CONTAINERS AND THE SOLARIS™ SERVICE MANAGER

Joost Pronk van Hoogeveen, Solaris Software

Sun BluePrints™ OnLine—May 2006

SUN BLUEPRINTS
[illegible]
APPROVED

Part No 819-4328-10
Revision 1.0, 5/3/06
Edition: May 2006

Table of Contents

Working with Solaris™ Containers and the Solaris™ Service Manager

Today, companies are looking for new ways to run data center environments. They recognize that deployment models requiring each application to run on its own system are costly to build and maintain. *Applications* and *services* are gaining in importance, and now tend to consist of a collection of smaller applications spread across multiple systems—yet these overloaded and interchangeable terms are still used to describe a single application instance on a given system. As organizations deal with these sprawling hardware and software infrastructures and the need to improve server efficiency, they are beginning to turn to virtualization techniques that can help deploy complex applications while raising resource utilization rates.

In response to these trends, computer system and software vendors have created new technologies to address these demands. With the release of the Solaris™ 10 Operating System, Sun has taken a big step towards delivering functionality that can help address many of these challenges. Solaris™ Containers consist of a set of technologies that help system administrators increase resource utilization by consolidating multiple applications on a single system. With Solaris Containers, administrators can specify the percentage of physical system resources each application receives, as well as isolate each application in its own virtual environment with its own hostname, IP address(es), users, file system, and more.

In addition to virtualization, the reliability of the system, or virtual environment, is becoming critical. Reliability has many aspects—resiliency against hardware faults, understanding what needs to be running on a system for applications to work, and making sure these system services are available to applications. In addition, ensuring the system is not running so many services that it is rendered vulnerable to intrusion or faults is a key concern. Introduced in the Solaris 10 OS, Solaris™ Predictive Self-Healing technology automatically diagnoses, isolates, and recovers from many hardware and application faults. This is particularly important in a world where applications consist of multiple pieces spread over many systems. More moving parts generally results in lower mean time between failures.

The combination of virtualization and reliability is especially important when consolidating this new breed of applications. This Sun BluePrints™ article describes how Solaris Containers and Solaris Predictive Self Healing features work together to address these needs. Not an explanation of the intricate details of these technologies, this article focuses on providing a taste of what the Solaris 10 OS has to offer, as well as some ideas on how to get started and put these new technologies to work. It provides a quick introduction to Solaris Containers and Solaris Predictive Self Healing technology, and moves into discussion of the Solaris Service Management Facility. Emphasis is placed on illustrating how this functionality can be used to create isolated environments customized for specific applications.

Solaris™ Containers Technology

Over the last few years, companies have been pressed to find ways to run the business with greater efficiency. As part of this effort, organizations learned that most systems in the corporate computing infrastructure run at very low utilization rates. This realization sparked an industry wide push toward server consolidation. To help this effort, system vendors began working on resource management tools. Initially, these tools were unbundled. As time went on, many of these tools became part of the underlying operating system. Indeed, Sun integrated resource management tools in the Solaris OS in the form of the Solaris 9 Resource Manager (S9RM) in the Solaris 9 OS, and Solaris Containers in the Solaris 10 OS.

More information on Solaris Containers can be found in *Consolidating Applications with Solaris Containers*, *System Administration Guide: Solaris Containers—Resource Management and Solaris Zones*, *Solaris Containers—What They Are and How to Use Them*, and *BigAdmin System Administration Portal—Solaris Zones* listed in the references at the end of this document.

Solaris Container Technology Components

Solaris Containers technology is the result of an evolutionary approach. Consequently, different features became part of the operating system at various releases. These features can be distinguished and split into two major categories:

- *Resource management*, technologies focused on controlling how many resources an application receives. Most resource management elements were introduced first in the Solaris 9 Resource Manager, although the Solaris 10 OS also incorporates some important enhancements.
- *Virtualization*, technologies focused on controlling the namespace the application sees. Virtualization elements were introduced in the Solaris 10 OS in the form of Solaris™ Zones technology.

A Solaris Container is the sum of both parts.

Solaris Resource Management

Central to resource management is the measurement and control of a *workload*—an application or sub-part of an application—rather than the system as a whole. This focus is needed for successful consolidation of multiple workloads onto a single system. In a consolidation effort, resource management has three roles:

- A method to classify workloads, ensuring the system knows which processes belong to which workload
- The ability to measure the workload and quantify how many system resources the workload is really using
- The ability to control the workload, ensuring it does not interfere with other workloads, yet gets enough system resources to meet its service level requirements

The Solaris OS introduced several new objects to help accomplish these tasks. The creation of a new type of ID, called a *Project*, enables workloads to be classified on a system. New tools were introduced, such as Extended Accounting (`acctadm(1M)`) and new features in the `prstat(1M)` command, that allow the measurement of these projects. Finally, project-based controls were introduced, including the Fair Share Scheduler (FSS) and physical memory capping (`rcapd(1M)`), as well as additional system wide controls like IP Quality of Service (IPQoS) and Resource Pools (`poold(1M)`). These features are enhanced in the Solaris 10 OS. For example, Dynamic Resource Pools now make it possible to define a range of CPUs for a pool. Depending on the load in these pools and the rules specified, CPUs can be moved automatically from one pool to another to best fit rules and conditions.

Figure 1 shows an example of several projects, contained in pools, on a single system.

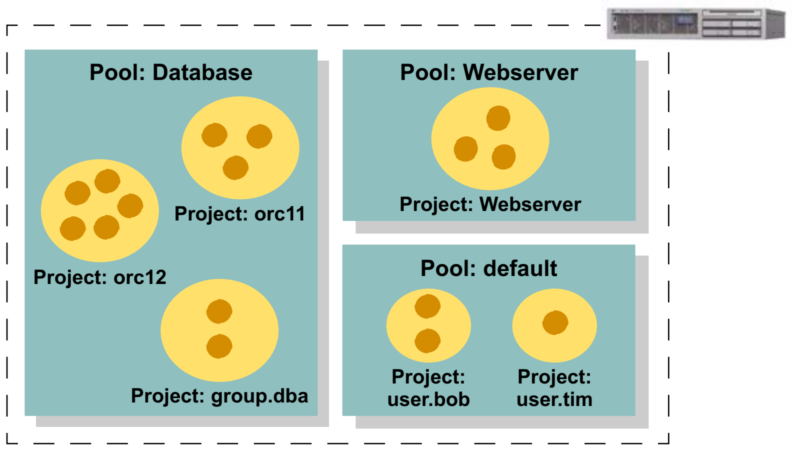

Figure 1. Several projects and pools on a system.

Solaris™ Zones

The Solaris 10 OS incorporates an entirely new aspect of Solaris Containers technology. Solaris Zones technology enables the creation of new virtual environments that are run on a single operating system kernel. These virtual environments appear to be a separate system to applications and users. However, each virtual environment has a separate *namespace* or system identity, including a user namespace, a file namespace, an IP port namespace, etc. Every zone has its own namespace, and therefore has its own users, root user, files, IP addresses, IP ports, hostname, and much more. It has everything it needs to act like an independent system from the application perspective.

The underlying original operating system, called the *global zone*, remains, and has its own namespace. The global zone is the place where the kernel runs, and from where the system is controlled and configured, and where the other *non-global zones* are created. Non-global zones are isolated from each other. Not only do they have a separate namespace, non-global zones cannot *see* one another, their processes, or their attributes, such as IP addresses. Non-global zones also cannot share memory through mechanisms like IPC, and even have their own user level operating system services, such as lie `inetd`, `telnetd`, `sshd`, and so on. Because every zone is isolated in this way, zones can be independently booted and rebooted at will without disturbing the other environments on the system.

A Solaris Container provides isolation (via Solaris Zones technology) and resource control (via Solaris Resource Management features). For a complete introduction to Solaris Containers technology, see *Consolidating Applications with Solaris Containers* and *Solaris Containers—What They Are and How to Use Them*, listed in the references at the end of this document.

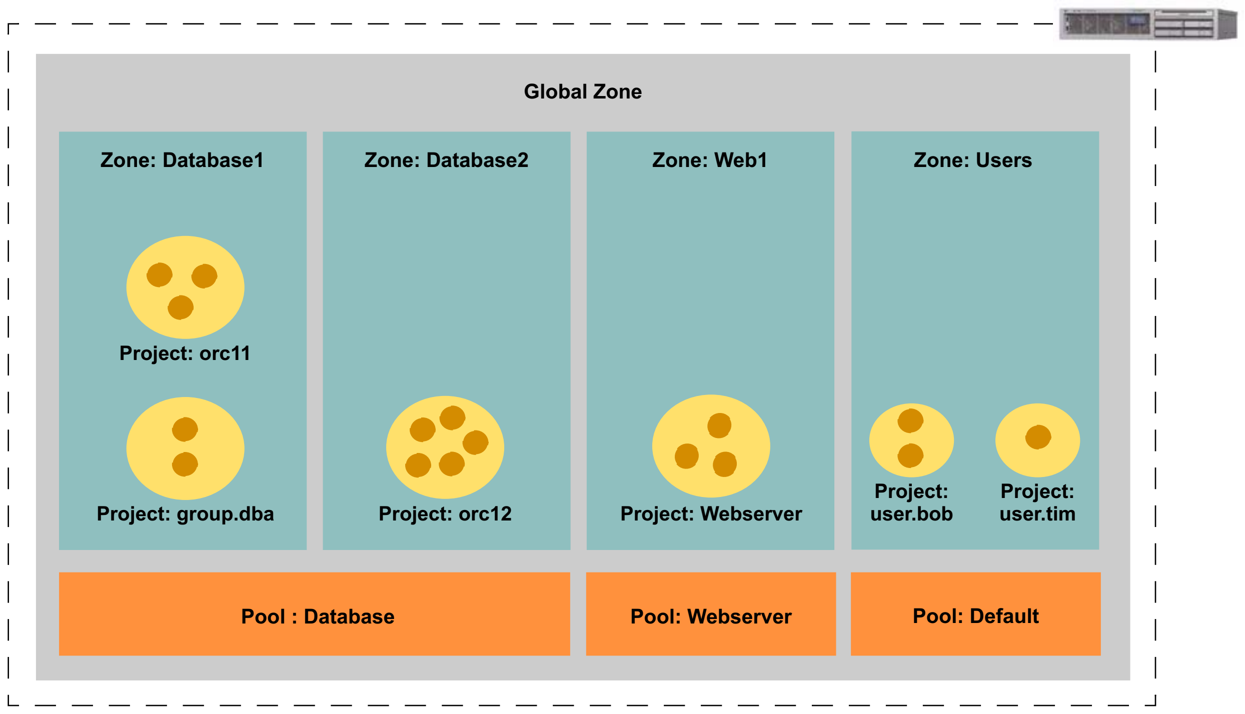

Figure 2. A system with several projects running in Zones that are assigned to resource pools.

Example of a Solaris Zone

Creating a new non-global zone is not difficult. This section describes how to install a typical non-global zone using the command line interface (CLI).

1. Define the zone using the `zonecfg(1M)` command. This example creates a new non-global zone named *mailzone*.

```
# zonecfg -z mailzone
mailzone: No such zone configured
Use 'create' to begin configuring a new zone.
zonecfg:mailzone> create
zonecfg:mailzone> set zonepath=/zones/mailzone
zonecfg:mailzone> add net
zonecfg:mailzone:net> set address=192.168.10.10/24
zonecfg:mailzone:net> set physical=hme0
zonecfg:mailzone:net> end
zonecfg:mailzone> verify
zonecfg:mailzone> commit
zonecfg:mailzone> exit
```

2. Verify the configuration was set correctly using the `zonecfg(1M)` command. Note that `autoboot` is set to `false`. As a result, this zone does not automatically boot when the system is booted. In addition, this zone does not have a `pool` associated with it—the zone boots in the default system pool and utilizes its associated CPUs. Finally, note the four `inherit-pkg-dir` definitions that indicate which directories will be mounted with the loopback file system (`lofs(1M)`) in a read-only fashion. By default, the `zonecfg` command inserts these definitions and creates a *sparse root zone* in order to

save disk space. Using the `create -b` option forces the `zonecfg` command not to use them, creating what is known as a *whole root zone*.

```
# zonecfg -z mailzone info
zonepath: /zones/mailzone
autoboot: false
pool:
inherit-pkg-dir:
        dir: /lib
inherit-pkg-dir:
        dir: /platform
inherit-pkg-dir:
        dir: /sbin
inherit-pkg-dir:
        dir: /usr
net:
        address: 192.168.10.10/24
        physical: hme0
#
```

3. Instruct the system to install the zone onto the disk using the `zoneadm(1M)` command.

```
# zoneadm -z mailzone install
Preparing to install zone <mailzone>.
Creating list of files to copy from the global zone.
Copying <2563> files to the zone.
Initializing zone product registry.
Determining zone package initialization order.
Preparing to initialize <984> packages on the zone.
Initialized <984> packages on zone.
Zone <mailzone> is initialized.
The file </zones/mailzone/root/var/sadm/system/logs/install_log> contains a
log of the zone installation.
```

   Note the number of files and packages installed on the system. The output reveals approximately three files for every package, which could be more or less files than a typical operating system installation. This is the result of mounting `/usr` through the `inherit-pkg-dir` mechanism.

4. Once the zone installation completes, the zone can be booted with the `zoneadm -z` *`zonename`* `boot` command. After the boot process completes, log into the zone's virtual console with the `zlogin -C` *`zonename`* command.

During the initial boot process, the system configuration (`sysidcfg`) phase prompts for system information, such as the locale and root password. If the file is not in place, or is incomplete, the `sysidcfg` phase is initiated when connecting to the zone console.

However, `sysidcfg` information can be created to automated system identification. If used, this configuration information is created and stored in the *`zoneroot`*`/root/etc/sysidcfg` file before the zone is booted. The zone reads the file at boot time and self-configures using the information specified. Following is a sample `sysidcfg` file. Note that this example does not use valid values for the `domain_name` and `root_password` entries. While this example is valid for the Solaris 10 OS, keep in mind that new values may be added in future Solaris OS releases.

```
# cat /zones/mailzone/root/etc/sysidcfg
system_locale=C
timezone=US/Pacific
terminal=xterm
security_policy=none
name_service=NIS {
  domain_name=yourdomain.com
}
network_interface=primary {
  hostname=mailzone
}
root_password=xxx
#
```

Figure 3 provides an overview of a system with many of the Solaris Container technologies enabled. Note that the Container with the zone named `Mailzone` and attached to pool `Pool2` has dedicated resources. All other zones share resources—in this case CPU resources with the Fair Share Scheduler enabled. In addition, the processes in the global zone run on the same shared CPUs with five shares.

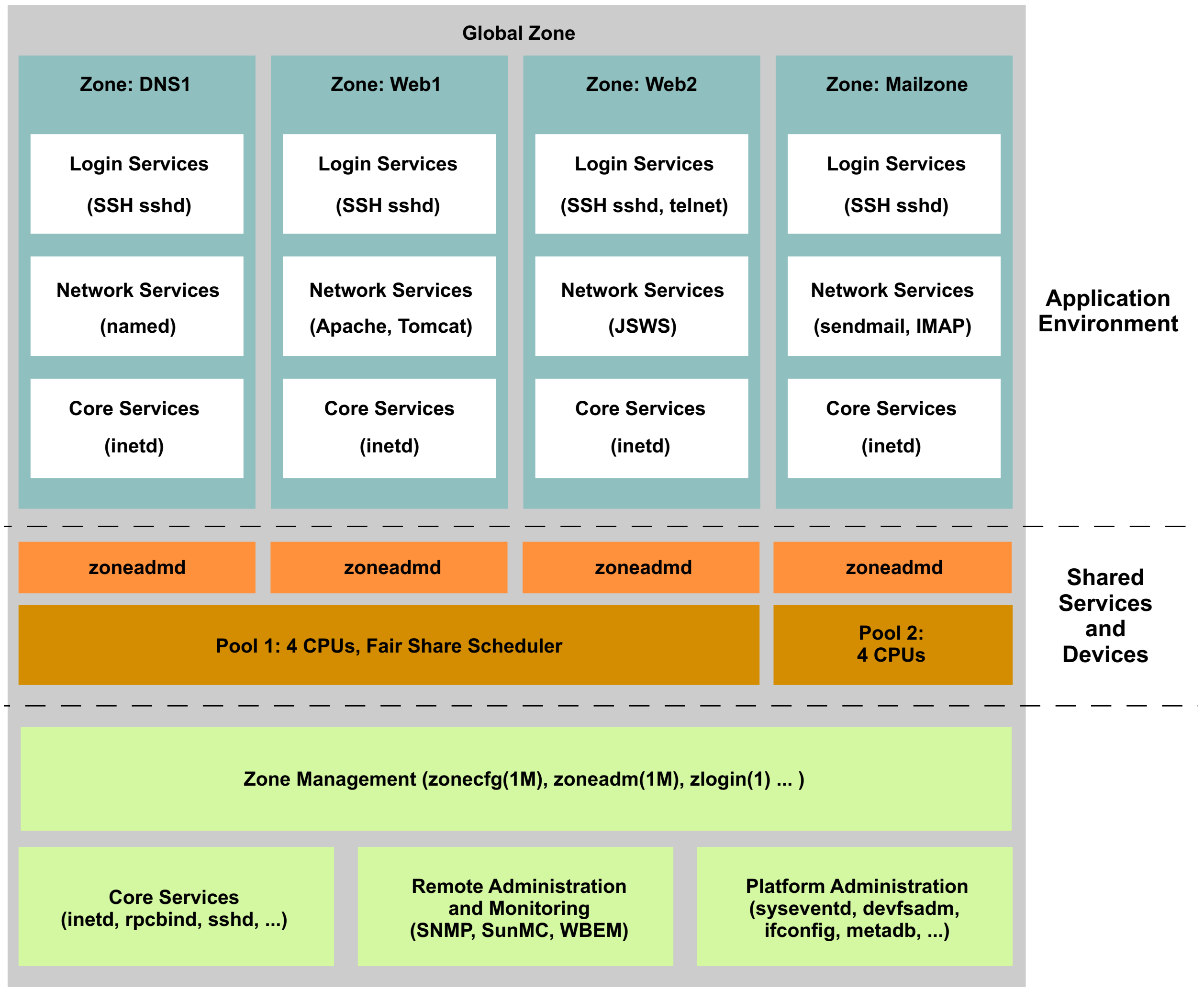
Global Zone
Zone: DNS1
Zone: Web1
Zone: Web2
Zone: Mailzone
Login Services (SSH sshd)
Login Services (SSH sshd)
Login Services (SSH sshd, telnet)
Login Services (SSH sshd)
Network Services (named)
Network Services (Apache, Tomcat)
Network Services (JSWS)
Network Services (sendmail, IMAP)
Core Services (inetd)
Core Services (inetd)
Core Services (inetd)
Core Services (inetd)
Application Environment
zoneadmd
zoneadmd
zoneadmd
zoneadmd
Pool 1: 4 CPUs, Fair Share Scheduler
Pool 2: 4 CPUs
Shared Services and Devices
Zone Management (zonecfg(1M), zoneadm(1M), zlogin(1) ...)
Core Services (inetd, rpcbind, sshd, ...)
Remote Administration and Monitoring (SNMP, SunMC, WBEM)
Platform Administration (syseventd, devfsadm, ifconfig, metadb, ...)

Figure 3. A system with several Containers, including pools and zones.

Solaris Predictive Self-Healing

The Solaris 10 OS brought with it many changes, including the fact that the operating system no longer considers the system flawless—something that results in many other operating systems being surprised when something goes wrong, causing a panic. The introduction of Solaris Predictive Self-Healing technology changes the rules of the game. Now, the Solaris OS takes a different view—it presumes things will break and it is up to the system to catch issues early and attempt to correct any resulting damage. The ability to be proactive requires two major technologies:

1. A technology that enables the system to look for faults, and preferably catch them before they occur or cause any loss of data. In the Solaris 10 OS, this is made possible by the Solaris Fault Manager software.
2. A technology that knows what kernel and application services are running on the system, and their dependencies, and knows how to restart them if and when those actions are needed. In the Solaris 10 OS this is made possible by the Solaris Service Manager software.

More information on the Fault Manager and how it can be used with the Solaris Service Manager software can be found in *System Administration Guide: Basic Administration, Chapter 9, Managing Services (Overview)*, *BigAdmin System Administration Portal—Predictive Self-Healing*, and several Weblogs listed in the references at the end of this document.

Solaris Service Manager Software

On a system, the Solaris Service Manager software is represented by the Service Management Facility `smf(5)`. The Service Management Facility is an infrastructure that provides several functions:

1. Definition of services for the Solaris OS, which can be the state of a device, a running application, or a set of other services. Each service is referred to by a unique identifier.
2. A formal relationship between services, with explicit dependencies.
3. Automatic starting and restarting of services.
4. A repository for storing service state and configuration properties, eliminating the need for dozens of configuration files scattered throughout the system.

At a high level, the system is managed by a master *restarter* named `svc.startd`. This daemon enforces dependencies, starts and stops services, and basically keeps an eye on how the machine is running. All related configuration information is stored in a *repository* on the system, and is managed by the `svc.configd` daemon. One or more *delegated restarters* are given a subset of services to manage, and are written specifically to deal with this subset. For example, the `inetd` daemon manages most networking services as a delegated restarter.

Services

A *service* is the fundamental unit of the Service Management Facility. Each service can have one or more *instances*, specific configurations of a service. For example, an Apache daemon configured to serve *www.sun.com* on port 80 is an instance of the Apache service. The Apache software may have several instances, each with a different configuration. The service holds basic configuration properties that are inherited by each of its instances. However, each instance can override configuration properties as needed. Special services, called *milestones*, correspond to a specific system state, such as *basic networking* or *local file systems available*. Milestones are essentially a list of other services, and are considered to be online when each of their component parts is online.

Each service is identified with a Fault Management Resource Identifier (FMRI), an unique identifier representing a service or instance. For example, the telnet service is represented by `svc:/network/telnet:default`, where `svc:/network/telnet` describes the service, and `default` describes a specific instance. FMRIs can be a bit of a handful to type. As a result, most SMF commands accept shortened versions of a service's FMRI, given that it only has one instance. For example, most utilities accept `network/telnet` as the FMRI for telnet, since it comes installed with only one instance.

Note that `telnet` is preceded with the word `network`. The Service Management Facility contains several categories for services to provide naming organization and uniqueness. This categorization enables administrators to quickly determine where a services lives, and to what it is related. Standard categories include:

- `application`
- `device`
- `milestone`
- `network`
- `platform`
- `site`
- `system`

States

Each service on a machine is always in one of seven discrete states that are observable by the Service Management Facility CLI tools. These states include:

- `degraded`, while the service is running, something is wrong or its capacities are limited in some way
- `disabled`, the service has been disabled and is not running
- `legacy_run`, a legacy *rc*.d* script has been started by the system and is running
- `maintenance`, the instance encountered an error and needs to be repaired by an administrator
- `offline`, the service is enabled but not running usually because a service it depends on is not online
- `online`, the service is both enabled and running successfully
- `uninitialized`, the `svc.startd` daemon has not yet read the service configuration

Manifests

One of the powerful features of the Service Management Facility is that it knows the relationships between different services on the system and how they are related. *Manifests* are the mechanism that enable the system to learn about these relationships. An SMF manifest is an XML file describing a service. All the manifests in the system are stored in `/var/svc/manifest` under categorical subdirectories. If custom services are not intended to be converted to the SMF model, these files do not need to be edited. However, these files can provide a helpful reference.

During the boot process, the `svc.configd` SMF daemon looks in the manifest directory. If new manifests exist, the daemon imports them into the repository. This can also be done manually by administrators. The entire system is run using information in the repository, not the manifests—manifests are simply a delivery mechanism for service descriptions. An active system is administered using the SMF command line tools.

Compatibility

These technological advancements came with an important design goal—ensure all of the layered software installed and functioned just as it did in previous Solaris OS versions. Many Solaris OS users rely on scripts and services they have carefully honed over time. While there are advantages to converting these services to take advantage of the benefits of SMF, it is not required. Custom scripts located in `/etc/rc*.d` continue to execute on run-level transitions. However, some Solaris OS kernel scripts are already converted and no longer need to be used.

The service states include the `legacy_run` state, which is used to identify services started through the old `/etc/rc*.d` mechanism. The SMF observational tools use this state to identify legacy services. When a service is labeled with the `legacy_run` state, it means the script was located in a `/etc/rc*.d` directory that was run upon successful transition to a run level.

Many standard Solaris OS services are already converted to SMF. As a result, there are fewer scripts in the `/etc/init.d` and `/etc/rc*.d` directories than in previous operating system releases.

Starting and Restarting Services

When the system understands how to start and stop services, as well as dependencies between services, a dependency tree can be built. This dependency tree is not new—in many ways, the `/etc/rc*.d` scripts are another expression of this tree. However, the dependency tree and the state of all services is now maintained in the kernel. For the first time, the kernel knows the direct relationship between the service and process namespaces.

The implications are far reaching. For example, when a process exits for any reason, including the accidental or intentional killing of the process, it is terminated by the kernel. Since the kernel now knows whether or not the process represents a service instance that needs to continue, it can immediately intervene and restart the service instance as another process, if needed. Additionally, the kernel understands the relationship of this process to other services on the system, including how they may or may not be impacted by the change. As a result, systems are less likely to run unintentionally in a degraded state when services are signed up for SMF. Systems can now essentially self-heal, or inform administrators through SMF interfaces in the event they cannot do so.

In addition, it is possible to turn enable (turn on) and disable (turn off) services, and understand the repercussions these actions will have on the other services running on the system. For example, the system can be told to enable a specific service, and in the same action enable and start all the services upon which it depends. Similarly, if a service is running and its configuration file is changed, the system can refresh the service and ensure it runs with the new values. If a service is disabled, generally it is for a good reason. For example, administrators may disable the `telnet` service in order to keep individuals from logging into the system remotely using the `telnet` command. The dependency tree enables administrators to identify the other services affected and mitigate the impact.

Service Profiles

SMF also includes the ability to save a service profile. Recall that SMF maintains a list of all services and their instances on the system, including whether or not those services are enabled or disabled. In essence, this list, or *service profile*, defines the personality of the system. SMF gives administrators the ability to capture the current service profile, load a new profile, or revert to an old profile. Whenever a system with a particular set of behaviors (or personality) needs to be created, it can be done so by replicating this list.

Examples of Handy Commands

The best way to understand how SMF works is to walk through some examples. This section provides an overview of the main SMF commands, or *administrative interfaces*, and shows some of them in action.

Administrative Interfaces

A lot of time and effort was put into making the administration of a system running the Solaris OS with SMF as painless as possible. Users no longer need to run the `grep` command and search for processes in output listings, or wonder if those processes are running, or hunt for configuration files. Now, SMF service administration is performed through a central interface, enabling administrators to observe the state of services and their dependencies and properties, and make changes to services.

Table 1 lists the `smf(5)` command line interface tools.

Command	Description
`svcs(1)`	Enables administrators to observe the state of all service instances on the system, and provides detailed views of service dependencies, processes, and more
`svcadm(1M)`	Provides service administration, including the ability to enable, disable, and restart services
`inetadm(1M)`	Enables administrators to observe and configure services that are controlled by the `inetd` daemon
`svccfg(1M)`	Enables administrators to manipulate the contents of a repository, usually properties in a service
`svcprop(1)`	Enables property values to be observed in a read-only manner; outputs are formalized for easy use in shell scripts

More detailed information can be found in the product documentation, or the Weblogs listed in the references at the end of this document.

The Basics

To understand the nature of SMF, start with the `svcs(1)` command. A versatile command, `svcs(1)` is likely to be used in day-to-day administrative work. The basic output is a list of all services on the system that should be running, including the service state, the time the instance started, and the full FMRI.

```
# svcs
STATE          STIME    FMRI
legacy_run     22:36:00 lrc:/etc/rc2_d/S20sysetup
...
legacy_run     22:36:02 lrc:/etc/rc3_d/S90samba
online         22:35:55 svc:/system/svc/restarter:default
...
online         22:36:01 svc:/milestone/multi-user:default
online         22:36:02 svc:/milestone/multi-user-server:d efault
offline        22:35:56 svc:/application/print/ipp-listene r:default
offline        22:35:59 svc:/application/print/rfc1179:def ault
#
```

The `svcs -a` command produces a similar but longer list—it now details all the services that could be running, including those that are not enabled. Output from this command include lines like the following:

```
disabled       22:35:56 svc:/network/nfs/cbd:default
```

An In-Depth Look

Let’s take a look at the `ssh` service. The implications of the state of the `ssh` service help illustrate SMF concepts—either the `ssh` service exists and it is possible to log into the system, or the service is not present.

1. Check the status of the `ssh` service. Because the ssh service is unique, it is not necessary to use the full FMRI.

   ```
   # svcs ssh
   STATE          STIME    FMRI
   online         22:36:00 svc:/network/ssh:default
   ```

2. Obtain the full listing for the `ssh` service using the `-l` option of the `svcs` command. The output shows more information on the current state of the `ssh` service, as well as the services upon which it depends, the type of dependency, and the status of those services.

```
# svcs -l ssh
fmri         svc:/network/ssh:default
name         SSH server
enabled      true
state        online
next_state   none
state_time   Sun Feb 03 22:44:10 2006
logfile      /var/svc/log/network-ssh:default.log
restarter    svc:/system/svc/restarter:default
contract_id  153
dependency   require_all/none svc:/system/filesyst em/local (online)
dependency   optional_all/none svc:/system/filesys tem/autofs (online)
dependency   require_all/none svc:/network/loopbac k (online)
dependency   require_all/none svc:/network/physica l (online)
dependency   require_all/none svc:/system/cryptosv c (online)
dependency   require_all/none svc:/system/utmp (onl ine)
dependency   require_all/restart file://localhost/e tc/ssh/sshd_config
(online)
```

3. Alternatively, obtains the status of the `ssh` service and lists all the processes representing the service instance using the `ssh` command. Note the number 874 is the process ID (PID) of the `sshd` daemon.

```
# ssh -p ssh
STATE          STIME    FMRI
online         22:36:00 svc:/network/ssh:default
               22:36:00      874 sshd
```

4. Verify that a user can connect to the system from a different machine with the `ssh` command. If the login is successful, everything is working fine.

```
[other_system #] ssh zone1
Password:
Last login: Sun Feb 3 22:37:10 2006
Sun Microsystems Inc.   SunOS 5.10      Generic January 2005
#
```

5. Disable the `ssh` service using the `svcadm(1M)` command.

```
# svcadm disable ssh
# svcs -p ssh
STATE          STIME    FMRI
disabled       22:42:33 svc:/network/ssh:default
```

6. Attempt to log into to system from another machine. If the `ssh` service is disabled the action should fail.

```
[other_system #] ssh zone1
ssh: connect to host zone1 port 22: Connection refused
[other_system #]
```

7. Enable the ssh service using the svcadm(1M) command, and verify the service is running. Note the offline state indicates the ssh service is enabled but is waiting for another service to go online.

```
# svcadm enable ssh
# svcs -p
STATE          STIME    FMRI
offline        22:43:40 svc:/network/ssh:default
```

8. Identify the list of services upon which the `ssh` service depends using the `svcs` command. The sample output indicates the `ssh` service depends on the cryptographic services (with a `require_all` dependency). This service is disabled here to illustrate `svcs` functionality.

```
# svcs -d ssh
STATE          STIME    FMRI
disabled      22:43:27 svc:/system/cryptosvc:default
online        22:35:57 svc:/network/physical:default
online        22:35:57 svc:/network/loopback:default
online        22:35:58 svc:/system/filesystem/local:default
online        22:35:59 svc:/system/utmp:default
online        22:36:00 svc:/system/filesystem/autofs:default
```

9. Enable the `ssh` service as well as all the services upon which it depends using the `-r` option of the `svcadm` command. Note the `-v` option enables verbose mode.

```
# svcadm -v enable -r ssh
svc:/network/ssh:default enabled.
svc:/system/filesystem/local enabled.
svc:/milestone/single-user enabled.
svc:/system/identity:node enabled.
svc:/network/loopback enabled.
svc:/system/filesystem/minimal enabled.
svc:/system/filesystem/usr enabled.
svc:/system/filesystem/root enabled.
svc:/system/device/local enabled.
svc:/milestone/devices enabled.
svc:/system/manifest-import enabled.
svc:/network/physical enabled.
svc:/system/cryptosvc enabled.
svc:/system/filesystem/minimal:default enabled.
svc:/system/utmp enabled.
svc:/milestone/sysconfig enabled.
# svcs -p ssh
STATE          STIME    FMRI
online         22:44:10 svc:/network/ssh:default
               22:44:10     1148 sshd
```

10. Log into the system from another machine to verify the `ssh` service is running.

```
[other_system #] ssh zone1
Password:
Last login: Sun Feb 3 22:42:07 2006 from 10.9.2.100
Sun Microsystems Inc.   SunOS 5.10      Generic January 2005
#
```

Note that using the -D flag identifies which services are directly dependent on a service.

Using Service Profiles

At its simplest, a service profile is a list of enabled services. Because services influence the way a system reacts to events, a service profile describes the system personality. Among other things, the `svccfg(1M)` command enables administrators to export this list so it can be saved, or load a previously saved list so the system can adopt that personality. Exported lists are formatted in XML.

```
# svccfg extract
<?xml version='1.0'?>
<!DOCTYPE service_bundle SYSTEM '/usr/share/lib/xml/dtd/
service_bundle.dtd.1'>
<service_bundle type='profile' name='extract'>
  <service name='system/console-login' type='servi ce' version='0'>
    <instance name='default' enabled='true'/>
  </service>
...
</service_bundle>
```

Ideally, a service profile should be saved in a file in the */var/svc/profiles* directory, where the system houses several profiles. This listing with pre-configured profiles enables administrators to get quickly to a particular system state. For example, the `generic_limited_net.xml` profile configures the system with a basic network state, turns off all services that transmit passwords in clear text, and more.

```
# svccfg extract > /var/svc/profile/myprofile.xml
# ls /var/svc/profile
generic.xml                 ns_ldap.xml
generic_limited_net.xml     ns_nis.xml
generic_open.xml            ns_nisplus.xml
inetd_generic.xml           ns_none.xml
inetd_services.xml          platform.xml
inetd_upgrade.xml           platform_i86pc.xml
myprofile.xml               platform_none.xml
name_service.xml            prophist.SUNWcsr
ns_dns.xml                  ns_files.xml
```

Use the `svccfg` command to load a service profile. Use the `-v` option to list all the services this action enables, disables, or updates. Using this tool, administrators can bring systems to a known state. This can be helpful when attempting to move a system to a secure state, or enabling a server to diagnose another system.

```
# svccfg apply /var/svc/profile/myprofile.xml
```

Restarters

Predictive Self-Healing does as its name implies—it enables systems to be regenerated automatically after a fault occurs. A portion of this functionality is handled by the Fault Manager, with the remainder handled by the Service Manager. Consider the following self-healing example.

1. Identify a process that can be made to fail, such as the `sendmail` process.

```
# svcs -p sendmail
STATE          STIME    FMRI
online         22:36:00 svc:/network/smtp:sendmail
               22:36:01      913 sendmail
               22:36:01      914 sendmail
```

2. Inject a fault by killing the process. In this example, the `sendmail` process abruptly terminated, perhaps as a result of running on a failing hardware component, such as a DIMM.

```
# pkill -9 sendmail
```

3. Check to see how the service is functioning. Note the process IDs (PIDs) have changed, and the service is running once again.

```
# svcs -p sendmail
STATE          STIME    FMRI
online         22:48:04 svc:/network/smtp:sendmail
               22:48:04     1188 sendmail
               22:48:04     1189 sendmail
```

While short, this example illustrates the power of this tool—the fault was identified and corrected very quickly. Keep in mind the fault did not lie within the operating system or a zone or require a system or zone reboot. It was located on a small boundary, the process that could be controlled by the restarter for the service.

Putting it all Together

How do Solaris Containers and Predictive Self-Healing technologies work together?

- Every zone has its own Service Manager. As a result, every zone can have its own service profile. Consider a zone that is running a service that needs the `telnetd` daemon, and another zone needs to turn the `telnetd` daemon and other service off for security reasons. This is easily accomplished by both global and local administrators.
- System administrators can create a development Container for determining which services are needed for an application to run. A service profile can then be created for the Container. When moving to a production system, administrators simply need to install the Container—both the zone and resource management components—and apply the service profile. These steps can be reproduced many times, and assure the system is in a known state. When in doubt, administrators can reapply the service profile and revert to this known state, or a debug state with different services enabled or disabled.
- The Service Manager aims to contain faults at the service level. Every zone has his own Service Manager, and so has its own restarters. However, administrators can always choose to reboot the zone, or go to a different init level, if needed.

- Zones also have their own repositories. As a result, administrators can add own unique manifests to the environment. Every time the zone reboots, it checks for new manifests and loads them into the repository.
- If it is important to give services within a particular Container a specific resource guarantee, or limit, services can also be assigned to a project. Consequently, when the Container boots, the service automatically starts in the right project and receives the correct resources, such as CPU fair shares.

Summary

Solaris Containers and Predictive Self-Healing technologies work together by creating separate execution environments, each with their own namespace and assigned resources. Each environment can have its own self-healing personalities—personalities that can be changed, copied, and reloaded as needed. In addition, these technologies enable administrators to determine the current state of the environment, making it easier to use the Solaris OS for consolidation efforts.

References

BigAdmin System Administration Portal—Predictive Self-Healing
`http://www.sun.com/bigadmin/content/selfheal/`

BigAdmin System Administration Portal—Solaris Zones
`http://www.sun.com/bigadmin/content/zones/`

Consolidating Applications with Solaris Containers
`http://www.sun.com/datacenter/consolidation/solaris10_whitepaper.pdf`

OpenSolaris Project
`http://www.opensolaris.org`

Service Management Facility (SMF) in the Solaris 10 OS
`http://www.sun.com/blueprints/0206/819-5150.pdf`

Solaris Containers—What They Are and How to Use Them
`http://www.sun.com/blueprints/0505/819-2679.pdf`

System Administration Guide: Basic Administration, Chapter 9, Managing Services (Overview)
`https://docs.sun.com/app/docs/817-1985/6mhm8o5no?a=view`

System Administration Guide: Solaris Containers—Resource Management and Solaris Zones
`https://docs.sun.com/app/docs/817-1592`

Weblogs
`http://blogs.sun.com/dp`
`http://blogs.sun.com/sch`
`http://blogs.sun.com/lianep`
`http://blogs.sun.com/tobin`
`http://blogs.sun.com/darren`

Ordering Sun Documents

The SunDocs^{SM} program provides more than 250 manuals from Sun Microsystems, Inc. If you live in the United States, Canada, Europe, or Japan, you can purchase documentation sets or individual manuals through this program.

Accessing Sun Documentation Online

The `docs.sun.com` web site enables you to access Sun technical documentation online. You can browse the `docs.sun.com` archive or search for a specific book title or subject. The URL is `http://docs.sun.com/`

To reference Sun BluePrints OnLine articles, visit the Sun BluePrints OnLine Web site at: `http://www.sun.com/blueprints/online.html`