ZFS Demonstration Tutorial

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
USA

Part No:
July 2007

070719@18135

Contents

CHAPTER 1

ZFS Demonstration Tutorial

This demo sequence has been tested with ZFS on Solaris Nevada release, build 55.

About this Document

This document is intended to provide a demo script of the key features of ZFS and enable you to demo them on a real ZFS system. The document is structured into a set of features, such as ease of set up, ease of management, and so on. Each feature description contains the following sections:

- Goal of Demo Component
- What You Need to Do
- Key Messages
- Why This Feature is Important

The web-based ZFS management tool is not covered in this tutorial except for the instructions on how to start it.

You might not want to demo every capability to all customers. They might be overwhelmed.

Preparatory Work

This section describes the preparatory work that should be performed prior to the demo.

Loading ZFS

You will need to install the `SUNWzfsr` and `SUNWzfsg` packages. These packages are in the latest Solaris Nevada build (Built 27 for the ZFS package, build 28 for the GUI) or from the following releases:

- Solaris Express Community Release (SXCR) from opensolaris.org (November 2005 or later)
- Solaris Express release (December 2005 or later)
- Solaris 10 6/06 release media distribution (or later)

The ZFS packages are included in a full install of the Solaris OS.

Physical Disks

You need a minimum of four disks, preferably five or more and if possible, all the same size. However you create the disks, you will need to know their device names. For example, `c1t0s2`, or if using files in a UFS file system, their pathname. For example, `/zfsdisks/disk1`.

The ideal situation is to create a storage pool configuration that consists of multiple physical disks similar to a real deployment situation. Using physical disks enables you to demonstrate all the capabilities of ZFS. Even better if the disks have visible activity lights.

If you are going to demonstrate a storage pool configuration that consists of multiple slices, which is not recommended for real deployment scenarios, then use the format utility to create some slices.

Make sure to remember the slice details for this additional partition (for example, `c0d1p0` or `c1d1p0`). You will need them later.

Virtualization

If you are running in a virtualized environment, for example, VMWare, the virtualization technology might be able to present files in the host operating systems as multiple disks to the guest operating system. When using VMWare, create the disk drives (VM -> Settings -> Hardware tab -> Add), then edit the VM Configuration file (*vm-name*.vmx in the VM directory, and add the following entry:

```
disk.locking="FALSE"
```

This entry removes the protections for multiple virtual machines that access the device and allows you to simulate external damage to a device.

After you have booted the Solaris environment, you will need to make sure that the devices are recognized by Solaris (the `drvconfig` and `disks` commands help here). Using a virtualization solution allows you to demonstrate all the features of ZFS, though the performance is clearly limited by that of the underlying device. It also has the advantage of being able to demonstrate ZFS with minimal hardware.

Note that some virtualization solutions allow you to use real physical disks and to allocate them to the virtual machine. This is better than using virtual disks but does require more hardware.

If you are using a virtual disk or a physical disk through virtualization, you will need create a second partition to use for ZFS that does not overlap the disks label. Though ZFS itself does correctly preserve the label in the demo, we will be trashing some of these partitions and that process might damage the disk label.

Using UFS-Backed Files

If you do not have additional disks or a way of emulating them, you can create files in an existing UFS file system that ZFS can use as if they were disks.

Use the `mkfile` command to create the files that will be used as disks. For the purposes of this demo, the files do not need to be very large. For example:

```
mkfile 100M /zfsdisks/disk1
```

Remember the pathname to the file. For example, `/zfsdisks/disk1`.

You could create sparse files using the `-n` option if you wished, though, because ZFS writes anywhere. You are simply trading time of allocation during the `mkfile` command for the time during the ZFS write operation.

Zones

One of the advanced demo components examines the use of ZFS with zones. To do this, you must have a zone available.

If you do not have a zone, the following sequence of steps creates a very simple zone that is suitable for the demo. You should probably create the zone prior to the demo. (This assumes you have sufficient space in the root file system (/). If not, use another path.

Zones are really neat to use. After you have done your ZFS run through, I can thoroughly recommend that you look at zones in more detail.

After your zone is up and running, you can reuse it for subsequent demos. You don't have to create a zone each time. Instructions on how to delete a zone are at the end of this demo script.

Note – Remember the name of your zone and also the root password.

Web-Based ZFS Management Tool

A web-based ZFS management tool is available to perform many administrative actions. With this tool, you can perform the following tasks:

- Create a new storage pool

- Add capacity to an existing pool
- Add capacity to an existing pool
- Add capacity to an existing pool
- Move (export) a storage pool to another system
- Import a previously exported storage pool to make it available on another system
- View information about storage pools
- Create a file system
- Create a volume
- Take a snapshot of a file system or a volume
- Roll back a file system to a previous snapshot

You can access the ZFS Administration console through a secure web browser at the following URL:

```
https://system-name:6789/zfs
```

If you type the appropriate URL and are unable to reach the ZFS Administration console, the server might not be started. To start the server, run the following command:

```
 # /usr/sbin/smcwebserver start
```

If you want the server to run automatically when the system boots, run the following command:

```
# /usr/sbin/smcwebserver enable
```

About this Demonstration

Demonstrating a file system in and of itself, beyond saying things like `ls -l`, is difficult because very little is visible. Therefore, this demo focuses on the areas that are different from previous file systems, and in particular, aspects that are seen by system administrators.

ZFS Command Summary

The following command summary provides a *cheat sheet* for running the demo. They encompass the major commands you need to use. These commands do not address using ZFS with zones. Note that devices do not include the slice. For example, use `c0t1d0`, not `c0t1d0s0`.

ZFS Command	Example
Create a ZFS storage pool	`# zpool create mpool mirror c1t0d0 c2t0d0`
Add capacity to a ZFS storage pool	`# zpool add mpool mirror c5t0d0 c6t0d0`
Add hot spares to a ZFS storage pool	`# zpool add mypool spare c6t0d0 c7t0d0`
Replace a device in a storage pool	`# zpool replace mpool c6t0d0 [c7t0d0]`
Display storage pool capacity	`# zpool list`
Display storage pool status	`# zpool status`
Scrub a pool	`# zpool scrub mpool`
Remove a pool	`# zpool destroy mpool`
Create a ZFS file system	`# zfs create mpool/devel`
Create a child ZFS file system	`# zfs create mpool/devel/data`
Remove a file system	`# zfs destroy mpool/devel`
Take a snapshot of a file system	`# zfs snapshot mpool/devel/data@today`
Roll back to a file system snapshot	`# zfs rollback -r mpool/devel/data@today`
Create a writable clone from a snapshot	`# zfs clone mpool/devel/data@today mpool/clones/devdata`
Remove a snapshot	`# zfs destroy mpool/devel/data@today`
Enable compression on a file system	`# zfs set compression=on mpool/clones/devdata`
Disable compression on a file system	`# zfs inherit compression mpool/clones/devdata`
Set a quota on a file system	`# zfs set quota=60G mpool/devel/data`
Set a reservation on a new file system	`# zfs create -o reserv=20G mpool/devel/admin`
Share a file system over NFS	`# zfs set sharenfs=on mpool/devel/data`
Create a ZFS volume	`# zfs create -V 2GB mpool/vol`
Remove a ZFS volume	`# zfs destroy mpool/vol`

Setting Up ZFS is Easy

ZFS is explicitly designed to make it easy to setup. So, very little configuration is required. This set of demo components show how simple ZFS is to set up.

Note – Safety – The `zpool` command used in several of the demo components do not allow you to use a device that is already in use on the system. For example, if the device contains a mounted file system even if the device appears to contain data.

For example, if a device contains a UFS file system that is not mounted, the `zpool create` or `add` commands will not use the device unless you use the `-f` flag. The `zpool` command does not test for all possible data types. You may want to deliberately configure your disks with existing UFS file systems, possibly mounted UFS file systems, to demonstrate the device-in-use checking feature.

Creating a ZFS Storage Pool

Goal of Demo Component – Show how easy it is to create a storage pool and add disks to the pool. Depending on what is already on the disks, you might also demonstrate the steps taken in ZFS to help prevent you from making *muppet* mistakes.

What You Need to Do – Create a storage pool called `mypool` that contains a single mirror. The command to create a pool is flexible so an entire pool with many devices could be created in a single operation. Note that the `zpool` command is designed to be very easy to use and intuitive, unlike most UNIX commands. For example:

```
# zpool create mypool mirror c1t0d0 c2t0d0
```

Note that you can specify the entire disk, in which case ZFS takes over and uses the entire device. No need to identify a specific partition unless you want to restrict ZFS to just that partition. When it has access to the whole device, ZFS can operate with enhanced performance as it can schedule operations more efficiently.

If the command succeeds, you get the command line prompt again. If a problem occurs, you get an error message with details.

You might need to use the `-f` flag. As you can see, the creating a pool is a very fast operation. In fact, all ZFS administration commands are equally fast. It is rare for an administration option to take more than a second.

You can review the storage pool with the `zpool list` command. The size, of course, reflects the mirrored nature of the pool, the raw disk capacity used is twice as large. The health column tells us that all of the devices in the pool are fully operational. If this command reports a degraded mode, it would mean that the storage pool had one or more device failures. But, that actual data is still available. If a sufficient number of devices fail, the data not be available, and the pool status would be reported as faulted.

```
# zpool list
NAME                    SIZE    USED   AVAIL    CAP  HEALTH     ALTROOT
mypool                 95.5M     89K   95.4M     0%  ONLINE     -
```

You can also show the detailed structure of the storage pool by using the `zpool status` command. In this case, everything is online. This pool contains mirrored devices, so if one of the devices was faulted, the pool would still be usable. However, the status would report a degraded mode as the redundancy protection would be lower.

```
# zpool status
  pool: mypool
 state: ONLINE
 scrub: none requested
config:

        NAME         STATE     READ WRITE CKSUM
        mypool       ONLINE       0     0     0
          mirror     ONLINE       0     0     0
            c1t0d0   ONLINE       0     0     0
            c2t0d0   ONLINE       0     0     0

errors: No known data errors
```

The basis of the ZFS architecture is the storage pool. This is a collection of devices that can be real or virtual devices, such as mirrors. The only device-level management that needs to be done in ZFS is to allocate a device to a pool at the redundancy level you want. After that, ZFS deals with all of the device management, freeing the storage administrator from having to worry about the state of the individual components again.

In addition

Why This Feature is Important – Doing the initial setup and configuration of storage is a time consuming task that often requires specialized expertise. In particular, constructing traditional volume management products of the right configuration is a painful exercise. ZFS reduces the time to virtually zero and also significantly reduces the required skills to manage storage.

Creating a ZFS File System

Goal of Demo Component – Show that creating a file system is very fast and a single-step operation compared to a multiple-step operation. Show that file systems are easy to create.

What You Need to Do

In ZFS, the design goals are that file systems should be almost as easy to create as directories. You don't have to create volumes or work out an incantation of magic parameters. In fact, you don't even need to know anything about the devices that form the pool. All you need to know is the pool name and the file system name you want to use.

```
# zfs create mypool/myfs
```

The file system we have just created is ready to go. It's automatically mounted and set up to be mounted at every boot. No need to do any other actions to set up the file system for use.

```
# zfs list
NAME                     USED  AVAIL  REFER  MOUNTPOINT
mypool                   115K  63.4M  24.5K  /mypool
mypool/myfs             24.5K  63.4M  24.5K  /mypool/myfs
```

Note that the AVAIL size from the `zpool list` and `zfs list` commands may vary slightly because the `zfs list` command accounts for a small amount of space reserved for the file system level operations that is not visible from the `zpool list` command.

Key Messages – After you have the storage pool up and running, creating a files system in the pool is very simple and like pool creation, is a very fast operation. This also means a fast response time to changes.

Why This Feature is Important – Doing the initial setup and configuration of file systems is a time-consuming task that often requires specialized expertise. Specifically identifying the right configuration parameters for the file systems can take a lot of planning to get right. ZFS reduces the time to virtually zero and because ZFS operates dynamically. No need to predefine items like inode density and other factors that can impact the file system.

Adding Disks to a ZFS Storage Pool

Goal of Demo Component – Show that adding additional disk capacity is trivial in terms of the work involved and also the time taken. Depending on what is already on the disks, it may also demonstrate the steps taken in ZFS to help prevent you from making *muppet* mistakes.

What You Need to Do – Add two devices to the pool.

```
# zpool add mypool mirror c3t0d0 c4td0
```

You may need to use the `-f` option if the disks were previously in use.

Confirm that the capacity of the storage pool has increased.

```
# zpool list
NAME                    SIZE    USED   AVAIL    CAP  HEALTH     ALTROOT
mpool                   136G    226K    136G     0%  ONLINE     -
mypool                  191M    122K    191M     0%  ONLINE     -
```

You can also display detailed structure of the storage pool by using the `zpool status -v` command.

```
# zpool status -v
  pool: mypool
 state: ONLINE
 scrub: none requested
config:

        NAME         STATE     READ WRITE CKSUM
        mypool       ONLINE       0     0     0
          mirror     ONLINE       0     0     0
            c1t0d0   ONLINE       0     0     0
            c2t0d0   ONLINE       0     0     0
          mirror     ONLINE       0     0     0
            c3t0d0   ONLINE       0     0     0
            c4t0d0   ONLINE       0     0     0
          spares
            c6t0d0   AVAIL
            c7t0d0   AVAIL
errors: No known data errors
```

In addition, it is a good idea to add hot spares to your redundant ZFS storage pool.

Key Messages – Occasionally, you might need to add additional storage capacity to your file systems. With traditional volume management products, this is achieved by the following process:

- Adding the disks to the volume manage
- Assigning some of all of their capacity to a disk set (disk group for VxVM)
- Growing the volume itself and finally increasing the size of the actual file system

Of course, if you want to increase the capacity of several file systems you would have to at least perform volume and file system grow operations for every file system. And, quite possibly, adding disks, and so on, for each file system as well. With ZFS, this becomes a trivial exercise that takes very little time and the additional storage is immediately available to all of the file systems in the pool.

Why This Feature is Important – With most traditional file systems, you have to track when they are getting full and grow the volumes and the file systems when needed. Monitoring scripts do exist for some file systems. This process assumes that capacity is available in the underlying volumes themselves. If not, you have to add additional capacity into the volume manager and make it available to the volumes before you can grow the volumes to grow the file system. With ZFS, adding additional capacity is a very simple and fast operation and this automatically makes space available to the file systems unless they are restricted by quotas.

Using Redundant ZFS Configurations

ZFS supports redundant virtual-device configurations in addition to disks and files:

- mirrored
- RAID-Z (enhanced RAID5)
- double-parity RAID-Z

This demo script uses mirrored disks as they are easier to use with small numbers of disks.

If you have many disks available, you could easily replace the mirror vdev type with `raidz`, which is a advanced form of RAID-5 that has dynamic stripe width, checksums, block level recovery, in addition to the legacy features of basic RAID-5.

When using `raidz`, the *overhead* is less than that used for mirroring, though the redundancy level is lower than a three- or more way mirror.

Managing ZFS is Easy

ZFS is designed to be very easy to administer after it is up and running. This set of demo components shows how easy ZFS is to manage.

Adding Additional ZFS File Systems

Goal of Demo Component – Show that creating additional file systems in ZFS is just as fast as creating the first, and that they can be created on top of an existing storage pool without needing to create one file system per storage pool.

What You Need to Do – Create an additional ZFS file system.

```
# zfs create mypool/myfs2
```

List the file systems.

```
# zfs list
NAME                     USED  AVAIL  REFER  MOUNTPOINT
mypool                   162K   159M  27.5K  /mypool
mypool/myfs             24.5K   159M  24.5K  /mypool/myfs
mypool/myfs2            24.5K   159M  24.5K  /mypool/myfs2
```

Note that new file system is in the same pool, both file systems are sharing the resources of the same storage pool. The space used by the storage pool has increased and both file systems are reporting the same amount of space available.

Key Messages – Of course, creating additional file systems with ZFS is just as simple and fast as creating the first. Even better, you can use the same storage pool so no additional administrative overhead is incurred. Unlike file systems that use traditional volume managers, you do not need to find extra devices or create additional volumes.

Why This Feature is Important – For traditional file system and volume manager configurations, you need to create new volumes, which might involve adding additional storage and integrating that into the volume manager. With ZFS, you simply create a new file system and it can share the available storage in its pool. The act of provisioning the storage and creating the file systems are decoupled, resulting in decreased administration time and a faster response to business needs. Also, the skill level required to administer ZFS is reduced because people with volume management skills are not required.

Setting Reservations on a ZFS File System

Goal of Demo Component – Show that it is possible to guarantee that a file system has a certain level of capacity available to it. Note that you should do this component *after* you have created multiple file systems, but *before* demonstrating the quota section.

What You Need to Do – Determine how much space is available in the pool and then subtract a few Mbytes to set a reservation on a file system. This process enables you to demonstrate not just how to specify the reservation, but also what happens when the reservation is reached.

First, work out the total space available.

```
# zfs list
NAME                     USED  AVAIL  REFER  MOUNTPOINT
mypool                   162K   159M  27.5K  /mypool
mypool/myfs             24.5K   159M  24.5K  /mypool/myfs
mypool/myfs2            24.5K   159M  24.5K  /mypool/myfs2
```

Set the reservation on the first file system.

```
# zfs set reservation=157m mypool/myfs
```

Check that the reservation has been applied.

```
# zfs get reservation mypool/myfs
NAME         PROPERTY     VALUE        SOURCE
mypool/myfs  reservation  157M         local
```

Note – The local value in the source column means that a property is specified for the actual file system. It is also possible to inherit some properties (though not reservations or quotas) from a parent file system. Property inheritance is described later in this document. You can double check this amount through the `zfs list` command. This command shows that one file system (`myfs`) has far more space available than (`myfs2`) because reserving space for one file system means that there is less space available for other file systems.

If you copy data to a file system that exceeds existing allocated space, ZFS actually rolls back the data that was copied over so you will not hit a full file system situation. For example, if you copy a large file to the `myfs2`, then operation fails.

```
# cp /platform/‘uname -m‘/kernel/‘isainfo -k‘/genunix /mypool/myfs2/genunix1
cp: /mypool/myfs2/genunix1: No space left on device
```

Remove the reservation.

```
# zfs set reservation=none mypool/myfs
```

Key Messages – One risk of sharing a multiple file systems between one pool is that one file system could take over all of the capacity in a pool. To prevent this scenario, ZFS has reservations and quotas. Here we are going to look at reservations, which will guarantee that a file system has a minimum amount of space available to it.

Why This Feature is Important – ZFS file systems are not implicitly restricted in size due to the size of their underlying storage. This feature enables you to restrict shared resources between many users, such as in university environment.

Setting Quotas on a ZFS File System

Goal of Demo Component – Show that it is possible to specify a maximum limit on the size a file system to prevent it from using all of the resources in the storage pool, starving other file systems and their users of storage capacity.

What You Need to Do – Set a quota on a file system.

```
# zfs set quota=3M mypool/myfs2
```

Then, check that the quota has been applied and assuming that the reservation has been removed from `mypool/myfs`.

```
# zfs get quota mypool/myfs2
```

Now, try copying a file to this file system.

```
# cp /platform/‘uname -m‘/kernel/‘isainfo -k‘/genunix /mypool/myfs2/
cp: /platform/sun4u/kernel/sparcv9/genunix: Disc quota exceeded
```

As with reservations, ZFS undoes the effects of the copy as it did not complete.

Remove the quota on `myfs2`.

```
# zfs set quota=none mypool/myfs2
```

Key Messages – In some environments, you might want to limit the amount of space that can be used by a file system. Historically, this has been achieved by applying end-user quotas on top of a fixed size volume. Having a shared storage pool is a different approach, which is to limit ZFS file system use by quotas.

In addition, ZFS file systems are points of administration for users, projects, groups, and so on. Per-user quotas are not implemented because the administrative model of ZFS is such that each user or project would have a separate file system. Per-user quotas do not match this administrative model.

Why This Feature is Important – In some situations a system administrator might want to guarantee a minimum level of space is available to a file system. Quotas allow that guarantee to be made.

Sharing ZFS File Systems

Goal of Demo Component – Demonstrate that ZFS even makes NFS administration easier.

What You Need to Do – Use the `share` command to see what is currently being shared on your system. should be no output from the `share` command. Now, set the `sharenfs` property on a ZFS file system to be shared.

```
# zfs set sharenfs=on mypool/myfs
```

Confirm that the property is set.

```
# zfs get sharenfs mypool/myfs
NAME         PROPERTY  VALUE        SOURCE
mypool/myfs  sharenfs  on           local
```

Show that NFS is now sharing the file system.

```
# share
-@mypool/myfs   /mypool/myfs   rw   ""
```

Finally, set the `sharenfs` property to off.

```
# zfs set sharenfs=off mypool/myfs
```

Key Messages – ZFS reduces the cost of configuring NFS and is especially useful for home directory situations where each user might have their own ZFS file system. When used with property inheritance (more later) this becomes especially simple.

Why This Is Important – Setting up NFS shares can take additional effort and time. ZFS automatically shares file systems, which reduces effort and time.

ZFS Snapshots for Data Recovery

Goal of Demo Component – Show snapshots can be taken quickly and with minimal overhead. Demonstrate accessing data by using a ZFS snapshot.

What You Need to Do – Copy two files to an empty ZFS file system. For example:

```
# cp /usr/dict/words /mypool/myfs/words
 # cp /etc/passwd /mypool/myfs/pw
```

Confirm the copy was successful.

```
# ls -l /mypool/myfs
total 520
-rw-r--r--   1 root     root         669 Feb 15 11:43 pw
-r--r--r--   1 root     root      206663 Feb 15 11:42 words
```

Take a snapshot and list the file systems.

```
# zfs snapshot mypool/myfs@first
# zfs list
NAME                     USED  AVAIL  REFER  MOUNTPOINT
mypool                   456K   159M  27.5K  /mypool
mypool/myfs              284K   159M   284K  /mypool/myfs
mypool/myfs@first           0      -   284K  -
mypool/myfs2            24.5K  2.98M  24.5K  /mypool/myfs2
```

The snapshot is currently using 0 Kbytes of storage, but it is referencing 266 Kbytes of storage (the amount that `du -sk` would return). Both the `myfs` and `myfs@first` snapshot are referencing the same storage. Also, the snapshot is not currently mounted or visible. It is mounted on demand.

Create a new file in the file system.

```
# cp /etc/inetd.conf /mypool/myfs
```

Confirm that the copy is successful.

```
# ls -l /mypool/myfs
total 525
-r--r--r--   1 root     root        1921 Feb 15 11:50 inetd.conf
```

```
-rw-r--r--   1 root     root         669 Feb 15 11:43 pw
-r--r--r--   1 root     root      206663 Feb 15 11:42 words
```

All snapshots are visible through the `/pool/filesystem/.zfs/snapshot/` directory in the file system where the snapshot was taken. This allows multiple snapshots to be taken and be accessible by users, although visibility of a snapshot can be disabled. You can see that the new file is not added to the snapshot.

```
# ls -l /mypool/myfs/.zfs/snapshot/first
total 520
-rw-r--r--   1 root     root         669 Feb 15 11:43 pw
-r--r--r--   1 root     root      206663 Feb 15 11:42 words
```

Display the `zfs list` output.

```
# zfs list
NAME                     USED  AVAIL  REFER  MOUNTPOINT
mypool                   492K   159M  27.5K  /mypool
mypool/myfs              310K   159M   286K  /mypool/myfs
mypool/myfs@first       23.5K      -   284K  -
mypool/myfs2            24.5K  2.98M  24.5K  /mypool/myfs2
```

The results are slightly different. The snapshot is now showing 7.5 Kbytes used as it has a different copy of the directory structure, one without the `inetd.conf` file entry. This is the only difference. The snapshot is still referencing 266 Kbytes of data, however.

The file system itself is referencing 268 Kbytes of data (to allow for the contents of the `inetd.conf` file). Now, edit the `pw` file in the original file system, remove a couple of lines (suggest using `vi` and removing the `uucp` lines).

Review the `zfs list` output.

```
# zfs list
NAME                     USED  AVAIL  REFER  MOUNTPOINT
mypool                   494K   159M  27.5K  /mypool
mypool/myfs              311K   159M   286K  /mypool/myfs
mypool/myfs@first       24.5K      -   284K  -
mypool/myfs2            24.5K  2.98M  24.5K  /mypool/myfs2
```

The snapshot is now taking up slightly more space as it not only has a unique view of the directory structure difference but also the `pw` file. Do a `diff` on the two versions of the `pw` file, one in the file system and one in the snapshot.

```
# diff /mypool/myfs/pw /mypool/myfs/.zfs/snapshot/first/pw
6a7,8
> uucp:x:5:5:uucp Admin:/usr/lib/uucp:
> nuucp:x:9:9:uucp Admin:/var/spool/uucppublic:/usr/lib/uucp/uucico
```

Finally, show that it is possible to roll back the file system to the snapshot, note that you need to use the `-R` flag to roll back to a snapshot older then the original one.

```
#  zfs rollback mypool/myfs@first
# zfs list
NAME                     USED  AVAIL  REFER  MOUNTPOINT
mypool                   458K   159M  27.5K  /mypool
mypool/myfs              284K   159M   284K  /mypool/myfs
mypool/myfs@first           0      -   284K  -
mypool/myfs2            24.5K  2.98M  24.5K  /mypool/myfs2
```

The snapshot is now taking no additional space though it still exists and it will track the changes that happen again. Of course the file system has now returned back to it's original state.

```
# ls -l /mypool/myfs
total 520
-rw-r--r--   1 root     root         669 Feb 15 11:43 pw
-r--r--r--   1 root     root      206663 Feb 15 11:42 words
```

Destroy the snapshot.

```
# zfs destroy mypool/myfs@first
# zfs list
NAME                     USED  AVAIL  REFER  MOUNTPOINT
mypool                   458K   159M  27.5K  /mypool
mypool/myfs              284K   159M   284K  /mypool/myfs
mypool/myfs2            24.5K  2.98M  24.5K  /mypool/myfs2
```

Key Messages – Snapshots allow you to capture a view of your data at a point in time, ZFS allows an almost unlimited number of snapshots to be taken. This provides a simple and easy way to recover from *Oops, I didn't mean to do that* type of mistakes or recovering from malicious damage to your data by rolling back your entire file system to the pre-damaged state. Snapshots are space efficient because they only record changes. Snapshots can be accessed directly through the file system without any special tools or suffering the time impact of recovering from tape.

Why This Feature is Important – The ability to easily access older versions of data without having to recover them from backup tapes allows users to handle their own *Oops* moments. Creating a snapshot provides a consistent backup without having to suspend operations during the backup process. The space efficient mechanism (only tracking changes) means that it is possible to create far more snapshots than the traditional way of creating a daily backup tape.

ZFS Property Inheritance

Goal of Demo Component – Show that inheritance can be used to apply properties to multiple ZFS file systems simultaneously.

What You Need to Do – Create a parent file system.

```
# zfs create mypool/homedirs
```

Then, create descendant file systems.

```
# zfs create mypool/homedirs/user1
# zfs create mypool/homedirs/user2
# zfs create mypool/homedirs/user3
```

Display the compression property value on these file systems.

```
# zfs get -r compression mypool/homedirs
NAME                   PROPERTY     VALUE                  SOURCE
mypool/homedirs        compression  off                    default
mypool/homedirs/user1  compression  off                    default
mypool/homedirs/user2  compression  off                    default
mypool/homedirs/user3  compression  off                    default
```

Set the compression property on the parent file system. Then, display the compression property value for these file systems.

```
# zfs get -r compression mypool/homedirs
NAME                   PROPERTY     VALUE            SOURCE
mypool/homedirs        compression  on               local
mypool/homedirs/user1  compression  on               inherited from mypool/homedirs
mypool/homedirs/user2  compression  on               inherited from mypool/homedirs
mypool/homedirs/user3  compression  on               inherited from mypool/homedirs
```

You can override the property for a specific file system even if it has been inherited.

```
# zfs set compression=off mypool/homedirs/user3
# zfs get -r compression mypool/homedirs
NAME                   PROPERTY     VALUE            SOURCE
mypool/homedirs        compression  on               local
mypool/homedirs/user1  compression  on               inherited from mypool/homedirs
mypool/homedirs/user2  compression  on               inherited from mypool/homedirs
mypool/homedirs/user3  compression  off              local
```

If you have a local property set and you want to use inheritance again, this can be applied as follows:

```
# zfs inherit compression mypool/homedirs/user3
# zfs get -r compression mypool/homedirs
NAME                   PROPERTY     VALUE             SOURCE
mypool/homedirs        compression  on                local
mypool/homedirs/user1  compression  on                inherited from mypool/homedirs
mypool/homedirs/user2  compression  on                inherited from mypool/homedirs
mypool/homedirs/user3  compression  on                inherited from mypool/homedirs
```

Key Messages – ZFS home directories might well move from being a directory within /export/home to individual file systems. This model allows the use of quotas and reservations to be applied to each users file system and for properties to be set individually,

In many cases, a system administrator might want to set a property on all of the users file systems, rather than doing so individually. If the file systems are created within a ZFS file system hierarchy, applying the property to the parent file system results in the property being inherited by all descendant file systems. You can nest file systems as deep as you want. Inheritance can be specified at file system level, so a multiple-level properly management approach is simple and feasible in ZFS.

Why This Feature is Important– ZFS provide a hierarchical file system model where individual file systems can be points of administration. In addition, file system property inheritance reduces time and errors from repetitive administrative actions.

Transferring ZFS Data

Goal of Demo Component – Demonstrate that a simple way to transfer ZFS data is by using the zfs send and zfs receive commands.

What You Need to Do – Before you take a snapshot, make sure that the file system is in a stable state and that applications are not changing the data. Different backup vendors have different approaches to this, but for ZFS, the best way is to take a snapshot.

```
# zfs snapshot mypool/myfs@copy
# zfs list
NAME               USED  AVAIL  REFER  MOUNTPOINT
mypool             459K   159M  27.5K  /mypool
mypool/myfs        284K   159M   284K  /mypool/myfs
mypool/myfs@copy      0      -   284K  -
mypool/myfs2      24.5K  2.98M  24.5K  /mypool/myfs2
```

Next, send the snapshot to a file. The output could be sent to another computer, perhaps using ssh to preserve the data confidentiality, but for this demo, we are just going to write the data into a file in /tmp.

```
# zfs send mypool/myfs@copy > /tmp/zfsdata
```

If needed, the zfs send command can generate an incremental backup between two snapshots. Next, we are going to recover the data sent to the file. For the demo, we will reload it into our second file system but this could be on a different machine. Or, in the case of destruction of the storage hardware, perhaps a fire took out the data center, the data could be placed into the original pool.

```
# zfs receive -d mypool/myfs2 <  /tmp/zfsdata
# zfs list
NAME                     USED  AVAIL  REFER  MOUNTPOINT
```

```
mypool                    751K   158M  27.5K  /mypool
mypool/myfs               284K   158M   284K  /mypool/myfs
mypool/myfs@copy             0      -   284K  -
mypool/myfs2              311K  2.70M  26.5K  /mypool/myfs2
mypool/myfs2/myfs         284K  2.70M   284K  /mypool/myfs2/myfs
mypool/myfs2/myfs@copy       0      -   284K  -
```

The `myfs@copy` snapshot is automatically created in `myfs2` and the `mypool/myfs2/myfs` file system is created. Confirm that there is no difference between the file systems.

```
# diff /mypool/myfs /mypool/myfs2/myfs
```

ZFS has also created the `.zfs/snapshot/copy` directories in both locations as well. Remove the snapshots and the new file system.

```
# zfs destroy mypool/myfs2/myfs@copy
# zfs destroy mypool/myfs2/myfs
# zfs destroy mypool/myfs@copy
```

Key messages – You can use the `zfs send` and `zfs receive` commands to transfer data from one ZFS file system to another, and potentially to a file system on a different system.

Why This Feature is Important – EMC Networker 7.3.2 backs up and restores ZFS file systems, including ZFS ACLs. Veritas Netbackup backs up and restore ZFS files, but ACLs are not preserved. Check the following resource to identify the last information about ZFS and third-party backup products:

http://opensolaris.org/os/community/zfs/faq

Snapshots can be used for handling the *Oops, I didn't mean to do that* situation where files are removed accidentally. The general ZFS checksum and recovery mechanisms can handle many device failures, assuming the pool is redundant. The `zfs send` and `zfs receive` commands can be used to store data for possible retrieval.

Transforming ZFS Data (Compression)

Goal of Demo Component – Demonstrate that ZFS can be set to automatically compress data.

What You Need to Do – It is very difficult to show that compressed data results in a shorter overall time frame, without some serious instrumentation and programs to generate data without going through the file system. To simulate a demonstration, copy some data and then do the same copy with compression turned on to see that the file sizes are different.

First check that compression is disabled.

```
# zfs get compression mypool/myfs
NAME          PROPERTY     VALUE        SOURCE
mypool/myfs   compression  off          default
```

Copy a file to the file system.

```
# cp /platform/‘uname -m‘/kernel/‘isainfo -k‘/genunix /mypool/myfs2/gu1
```

Enable compression on the file system and confirm that it is enabled.

```
# zfs set compression=on mypool/myfs
# zfs get compression mypool/myfs
NAME         PROPERTY     VALUE        SOURCE
mypool/myfs  compression  on           local
```

Copy the file over again.

```
# cp /platform/‘uname -m‘/kernel/‘isainfo -k‘/genunix /mypool/myfs/gu2
```

Compare the amount of disk used by both files.

```
# du -k /mypool/myfs/gu*
5381    /mypool/myfs/gu1
2681    /mypool/myfs/gu2
```

The compressed version is occupying about 1/2 of the disk space that is used by the uncompressed version. To see the amount of space actually saved by compression, use the following command.

```
# zfs get compressratio mypool/myfs
NAME         PROPERTY       VALUE        SOURCE
mypool/myfs  compressratio  1.33x        -
```

We only have one compressed file in the pool at the moment but compression ratios on source code have been seen at greater than 2.5 times. Note that enabling compression ONLY affects data written after it is enabled. It is not applied retrospectively.

Key Messages – The actual process of conducting I/Os is quite time consuming and many systems have spare CPU cycles. ZFS can use those spare CPU cycles to compress the data for a smaller I/Os that results in a shorted overall time to write and read data. This feature also has the benefit of reducing the amount of disk space that actually used.

Why This Is Important – I/O performance for compressible data can be increased and disk capacity used can be decreased.

Advanced Management (ZFS Clones)

Goal of Demo Component – Demonstrate the creation of a writeable clone that has an independent life from the original snapshot.

What You Need to Do – Normally, when you think about cloning file systems, you would create a template file system exactly as you wanted it. For example, setting up configuration files, `.cshrc`, and so on, as appropriate. For the purposes of this demo, we're going to assume that all of that is completed.

First, snapshot the original file system to create a starting point for the clones.

```
# zfs snapshot mypool/myfs@snap
```

Now, create a couple of clones.

```
# zfs clone mypool/myfs@snap mypool/clone1
#  zfs clone mypool/myfs@snap mypool/clone2
```

Display the file system information.

```
# zfs list
NAME               USED  AVAIL  REFER  MOUNTPOINT
mypool            8.03M   159M  28.5K  /mypool
mypool/clone1         0   159M  7.90M  /mypool/clone1
mypool/clone2         0   159M  7.90M  /mypool/clone2
mypool/myfs       7.90M   159M  7.90M  /mypool/myfs
mypool/myfs@snap      0      -  7.90M  -
```

Currently, the clones take up zero space even though they refer to everything in the original snapshot.

Make a change in the original and in the first clone.

```
# echo original > /mypool/myfs/clonetest
# echo first clone >  /mypool/clone1/clonetest
```

Display that the original file system and first clone are different.

```
# diff  /mypool/myfs/clonetest  /mypool/clone1/clonetest
1c1
< original
---
> first clone
```

A diff of the second clone doesn't work because the file does not exist.

```
# diff /mypool/myfs/clonetest  /mypool/clone2/clonetest
diff: /mypool/clone2/clonetest: No such file or directory
```

As with ZFS snapshots, clones use storage capacity optimally. They only track the differences between the clone and the underlying snapshot.

```
# zfs list
NAME               USED  AVAIL  REFER  MOUNTPOINT
mypool            8.09M   159M  28.5K  /mypool
mypool/clone1       24K   159M  7.90M  /mypool/clone1
mypool/clone2         0   159M  7.90M  /mypool/clone2
mypool/myfs       7.92M   159M  7.90M  /mypool/myfs
mypool/myfs@snap  23.5K      -  7.90M  -
```

Reset the pool by recursively removing the file system snapshot and clones.

```
# zfs destroy -R mypool/myfs@snap
```

Key Messages – If you need to have a core set of data in multiple file systems and do not want to have to create a copy of that data all the time (maybe because it is unchanging), consider using ZFS clones. ZFS clones allow you to create a file system that can be cloned in a space efficient manner that results in two independent file systems with common data.

Why This Feature is Important – This feature helps set up home directory configurations and other environments where initially common data is required, but then can be easily changed.

Migrating ZFS Storage Pools

Goal of Demo Component – Show that ZFS file systems and pools can be moved easily from one machine to another.

What You Need to Do – You will need two systems (or two virtual machines if using VMware) that have access to the same devices. You must be using devices to do this demo. If you are using files in a file system to hold ZFS data, this feature cannot be demonstrated.

Alternatively, you can show the manually initiated move with a single machine if that is all that is available to you. First we have to export the pool.

```
# zpool export mypool
```

You might need to use the `-f` flag to force the export in case mounts are in use.

Show that the export is successful.

```
# zpool list
no pools available
```

Show that the file systems are not available.

```
# ls /mypool
/mypool: No such file or directory
```

Next, see what pools are available to import.

```
# zpool import
  pool: mypool
    id: 12795948276703959950
 state: ONLINE
action: The pool can be imported using its name or numeric identifier.
config:

        mypool            ONLINE
          mirror          ONLINE
            c1t0d0        ONLINE
            c2t0d0        ONLINE
          mirror          ONLINE
            c3t0d0        ONLINE
            c4t0d0        ONLINE
          spares
            c6t0d0        AVAIL
            c7t0d0        AVAIL
```

Import the pool.

```
# zpool import mypool
```

All of the devices in the storage pool have been imported and the pool is reconstructed. Display the pool and file systems status is what we would expect.

```
# zpool status
  pool: mypool
 state: ONLINE
 scrub: none requested
config:

        NAME              STATE     READ WRITE CKSUM
        mypool            ONLINE       0     0     0
          mirror          ONLINE
            c1t0d0        ONLINE
            c2t0d0        ONLINE
          mirror          ONLINE
            c3t0d0        ONLINE
            c4t0d0        ONLINE
          spares
            c6t0d0        AVAIL
            c7t0d0        AVAIL
errors: No known data errors
```

Show that the file systems are available.

```
# zfs list
NAME          USED  AVAIL  REFER  MOUNTPOINT
mypool        458K   159M  27.5K  /mypool
```

```
mypool/myfs    284K   159M   284K  /mypool/myfs
mypool/myfs2  24.5K  2.98M  24.5K  /mypool/myfs2
```

The file systems are remounted. All in a single command.

Key Messages – High availability access to data (as opposed to continuous data access) is important in many situations. Historically, this has required that complex scripts be created to rebuild all of the disks in the right configuration to create the volumes. If the original server was not cleanly shut down, then you have to run consistency checks (or roll the logs for logging file systems) on all of the file systems that are contained in the volumes before finally creating the mount points and mounting the file systems onto the new server.

The old process is complex, time consuming and error prone, especially because the disk identifiers (c?t?d?s?) may have changed during the move. Even with the support in volume managers for using device IDs, you still have the consistency issues.

With ZFS, this is a simple command and due to the always consistent on disk data format, no need to do time-consuming consistency checking or log rolling. ZFS also allows movement between x86 and SPARC systems that have different endian requirements.

Why This Feature is Important – For disaster recovery because the ability to import the data from one machine to another enables the construction of failover environments that have little downtime. This also helps provide flexibility.

In addition, you can use the `zpool import -D` command to recover pools that have been destroyed.

Replacing Devices in Your ZFS Storage Pool

Goal of Demo Component – Demonstrate that ZFS can handle failed devices and recover once the device is replaced in a short period of time.

What You Need to Do – Occasionally, you might need to replace a device, perhaps because it is showing a large number of errors or maybe because it is obsolete and you are replacing it with newer hardware.

```
# zpool status
  pool: mypool
 state: ONLINE
 scrub: none requested
config:

        NAME                STATE     READ WRITE CKSUM
        mypool              ONLINE       0     0     0
          mirror            ONLINE
            c1t0d0          ONLINE
            c2t0d0        ONLINE
          mirror          ONLINE
            c3t0d0        ONLINE
            c4t0d0        ONLINE
          spares
            c6t0d0        AVAIL
            c7t0d0        AVAIL

errors: No known data errors
```

Looking at the storage pool, we can see that no problems exist, but for demo purposes we will pretend that `c4t0d0` has an issue. Let's replace the disk. We could also add a new side to the mirror with the `zfs attach` command, then after the resilvering is complete, detach the old device with the `zfs detach` command, but this is easier.

```
# zpool replace mypool c4t0d0 c5t0d0
```

Confirm the status of the pool.

```
# zpool status
  pool: mypool
 state: ONLINE
 scrub: resilver completed with 0 errors on Thu Feb 15 15:17:10 2007
config:

        NAME              STATE     READ WRITE CKSUM
        mypool            ONLINE       0     0     0
          mirror          ONLINE
            c1t0d0        ONLINE
            c2t0d0        ONLINE
          mirror          ONLINE
            c3t0d0        ONLINE
            c5t0d0        ONLINE
          spares
            c6t0d0        AVAIL
            c7t0d0        AVAIL

errors: No known data errors
```

The storage pool reflects the new device. ZFS only resilvers the amount of data, not the entire 100 Mbytes of the actual device. This is because ZFS is aware of the data that is actually in use in file systems, snapshots, volumes, and so on, and can do an optimal resilvering by duplicating the actual information in use. Independent file systems and volume manager cannot do this.

Key Messages – Occasionally, a storage device fails. ZFS can provide volume manager like recovery from this situation, assuming that the data is redundant. Unlike traditional volume managers, ZFS can do so in an optimal manner because it knows the file system metadata and data on the underlying storage devices. As such, ZFS only needs to recover the actual data in use

Why This Feature is Important – Recovering from a failed device is needed to restore resilience. The optimal recovery by ZFS means that the "window of risk" from multiple failures is minimized and that the recovery process does not place an unnecessary load on the system. You may also need to replace a device because it is obsolete or you are simply reorganizing your storage. In the past this has been difficult. With ZFS, this is a simple and fast operation.

Scrubbing a ZFS Storage Pool

Goal of Demo Component – Demonstrate that ZFS can check the validity of all of the checksums.

What You Need to Do – Initiate a scrub on a storage pool.

```
# zpool scrub mypool
```

Review the pool status. If you are quick, a scrub in progress is reported, but given the small storage pools used for this demo and that the scrub operation is optimal, you might not see a scrub in progress message.

```
# zpool status
  pool: mypool
 state: ONLINE
 scrub: scrub completed with 0 errors on Thu Feb 15 15:33:58 2007
config:

        NAME              STATE     READ WRITE CKSUM
        mypool            ONLINE       0     0     0
          mirror          ONLINE
            c1t0d0        ONLINE
            c2t0d0        ONLINE
          mirror          ONLINE
            c3t0d0        ONLINE
            c5t0d0        ONLINE
          spares
            c6t0d0        AVAIL
            c7t0d0        AVAIL
errors: No known data errors
```

The scrub has finished and no corruption is reported. Nice to know that your data is not at risk.

Key Messages – ZFS automatically detects data that fails the checksum and if the pool is redundant, will correct corruption. Rather than waiting for corruption to occur, ask ZFS to check the data for you and discover any problems in advance. This can also be used for peace of mind.

Why This Feature is Important – Disk storage is very reliable, but given increasing amounts of data, problems will be detected with more frequency. The ability to detect problems and repair them is built into ZFS. Scrubbing a disk can help confirm the safety of the data, perhaps before you remove one side of a mirror.

ZFS Checksum Recovery

Goal of Demo Component – Demonstrate that not only can ZFS detect corrupted data but also recover from corruption provided you have a redundant ZFS storage pool.

Caution – Be aware that this demo can wipe out data or a disk label, which can lead to more severe problems. Use it at your own risk.

What You Need to Do – You need to have a redundant storage pool. Copy some data into the pool, but this step is not mandatory.

```
# cp /usr/bin/v* /mypool/myfs
# ls -l /mypool/myfs
total 3320
-r-xr-xr-x   1 root     root      118984 Feb 22 13:57 vacation
-r-xr-xr-x   1 root     root       47856 Feb 22 13:57 vacuumdb
-r-xr-xr-x   1 root     root       21624 Feb 22 13:57 vacuumlo
-r-xr-xr-x   1 root     root       13092 Feb 22 13:57 vax
-r-xr-xr-x   1 root     root      283704 Feb 22 13:57 vedit
-r-xr-xr-x   1 root     root        5449 Feb 22 13:57 vgrind
-r-xr-xr-x   1 root     root      283704 Feb 22 13:57 vi
-r-xr-xr-x   1 root     root      283704 Feb 22 13:57 view
-rwxr-xr-x   1 root     root       44420 Feb 22 13:57 vino-preferences
-rwxr-xr-x   1 root     root       14368 Feb 22 13:57 vino-session
-r-xr-xr-x   1 root     root       65468 Feb 22 13:57 vmstat
-r-xr-xr-x   1 root     root       48964 Feb 22 13:57 volcheck
-r-xr-xr-x   1 root     root       57172 Feb 22 13:57 volrmmount
-rwxr-xr-x   1 root     root       14192 Feb 22 13:57 vsig
-rwxr-xr-x   1 root     root       25524 Feb 22 13:57 vte
-rwxr-xr-x   1 root     root       25620 Feb 22 13:57 vumeter
```

Check that your ZFS storage pools are healthy.

```
# zpool status -x
all pools are healthy
```

Then, destroy the contents of one of the disks by copying the random device. Note that the transfer size is limited.

Caution – Be very careful to ensure you only copy onto one side of a mirror pair and do not copy to slice 0 or you will overwrite the disk label.

If you are using small disks this is fast, but for large disks you may have quite a wait. Make sure you copy to the raw device (`/dev/rdsk`) not the buffered device (`/dev/dsk`). If you use the buffered device, the random data is cached and might not get written out to the disk.

```
# dd if=/dev/urandom of=/dev/rdsk/c5t0d0 count=100 bs=512k
100+0 records in
100+0 records out
# zpool status -x
all pools are healthy
# ls -l mypool/myfs
total 3320
-r-xr-xr-x   1 root     root      118984 Feb 22 13:57 vacation
-r-xr-xr-x   1 root     root       47856 Feb 22 13:57 vacuumdb
-r-xr-xr-x   1 root     root       21624 Feb 22 13:57 vacuumlo
-r-xr-xr-x   1 root     root       13092 Feb 22 13:57 vax
-r-xr-xr-x   1 root     root      283704 Feb 22 13:57 vedit
-r-xr-xr-x   1 root     root        5449 Feb 22 13:57 vgrind
-r-xr-xr-x   1 root     root      283704 Feb 22 13:57 vi
-r-xr-xr-x   1 root     root      283704 Feb 22 13:57 view
-rwxr-xr-x   1 root     root       44420 Feb 22 13:57 vino-preferences
-rwxr-xr-x   1 root     root       14368 Feb 22 13:57 vino-session
-r-xr-xr-x   1 root     root       65468 Feb 22 13:57 vmstat
-r-xr-xr-x   1 root     root       48964 Feb 22 13:57 volcheck
-r-xr-xr-x   1 root     root       57172 Feb 22 13:57 volrmmount
-rwxr-xr-x   1 root     root       14192 Feb 22 13:57 vsig
-rwxr-xr-x   1 root     root       25524 Feb 22 13:57 vte
-rwxr-xr-x   1 root     root       25620 Feb 22 13:57 vumeter
```

To force ZFS to check all of the blocks, use the `zpool scrub` command to initiate a scan of all the blocks. Then, use the `zpool status` command to view the results.

```
# zpool scrub mypool
# zpool status
  pool: mypool
 state: ONLINE
status: One or more devices has experienced an unrecoverable error.  An
        attempt was made to correct the error.  Applications are unaffected.
action: Determine if the device needs to be replaced, and clear the errors
        using ’zpool clear’ or replace the device with ’zpool replace’.
   see: http://www.sun.com/msg/ZFS-8000-9P
 scrub: scrub completed with 0 errors on Thu Feb 22 14:03:27 2007
config:
        NAME          STATE     READ WRITE CKSUM
        mypool        ONLINE       0     0     0
          mirror      ONLINE       0     0     0
            c1t0d0    ONLINE       0     0     0
            c2t0d0    ONLINE       0     0     0
          mirror      ONLINE       0     0     0
            c3t0d0    ONLINE       0     0     0
            c5t0d0    ONLINE       0     0    59
          spares
            c6t0d0    AVAIL
            c7t0d0    AVAIL

errors: No known data errors
# zpool clear mypool c5t0d0
# zpool status
  pool: mypool
 state: ONLINE
 scrub: scrub completed with 0 errors on Thu Feb 22 14:03:27 2007
config:

        NAME          STATE     READ WRITE CKSUM
        mypool        ONLINE       0     0     0
          mirror      ONLINE       0     0     0
            c1t0d0    ONLINE       0     0     0
            c2t0d0    ONLINE       0     0     0
          mirror      ONLINE       0     0     0
            c3t0d0    ONLINE       0     0     0
            c5t0d0    ONLINE       0     0     0
          spares
            c6t0d0    AVAIL
            c7t0d0    AVAIL

errors: No known data errors
```

Important features about the `zpool status` output are:

- No long sequences of arcane error messages are displayed on the console
- You are informed of unrecoverable errors on a device but that applications are unaffected. This feature is because ZFS is able to recover the data from the other side of the mirror and has rebuilt the corrupted data at the same time.
- In the event that recover was not possible you would be told, and you would not have to guess the situation. You are told what needs to be done and provided with a URL to go to for further information. The specific device that is experiencing problems is clearly identified and the extent of the damage is shown.

Key Messages – Detecting corrupted data is useful but the real benefit comes in being able to recover from it. When ZFS detects corrupted data and the data has some level of redundancy (mirror or RAID-Z), it can use the redundancy to recover and rebuild the original data.

Why This Feature is Important – In the event of data corruption, ZFS can recover the original data. ZFS allows this at low cost and in advance of traditional volume managers capabilities.

Using ZFS with Solaris Zones

Goal of Demo Component – Demonstrate the interaction of ZFS on a system with Solaris zones installed.

What You Need to Do – You will need to have a zone configured and running for this component of the demonstration. These steps must be performed in the global zone. Create a zone called `myzone`.

```
# mkdir /zones
# mkdir -m 700 /zones/myzone# zonecfg -z myzone
myzone: No such zone configured
Use ’create’ to begin configuring a new zone.
zonecfg:myzone> create
zonecfg:myzone> set zonepath=/zones/myzone
zonecfg:myzone> verify
zonecfg:myzone> exit
# zoneadm -z myzone install
Preparing to install zone <myzone>
Creating list of files to copy from the global zone.
Copying <7626> files to the zone.
Initializing zone product registry.
Determining zone package initialization order.
Preparing to initialize <1151> packages on the zone.
Initialized <1151> packages on zone.
Zone <myzone> is initialized.
Installation of these packages generated warnings: <UNWkvm SUNWcsl SUNWcsr SUNWcsu
SUNWesu SUNWcslr SUNWopenssl-libraries SUNWsmapi SUNWpolkit SUNWhal SUNWpapi SUNWpcu
SUNWkrbu SUNWtnfc SUNWtnfd SUNWdoc SUNWaudit SUNWrcmdc SUNWscpu SUNWmdu SUNWippcore
SUNWdhcsu SUNWdmgtu SUNWpool SUNWnisu SUNWarcr SUNWpiclu SUNWman SUNWcpcu SUNWsasnm
SUNWpsu SUNWppm SUNWdhcsb SUNWncau SUNWpoold SUNWpsm-ipp SUNWpsm-lpd SUNWsmedia
SUNWsrh>
The file </zones/myzone/root/var/sadm/system/logs/install_log> contains a log of the
zone installation.
.
.
.
```

Create a ZFS file system and make it available to the zone. Then, apply a quota to the file system.

```
# zfs create mypool/myzonefs
# zfs set quota=10m mypool/myzonefs
# zfs list
```

```
NAME              USED  AVAIL  REFER  MOUNTPOINT
mypool            119K   167M  25.5K  /mypool
mypool/myzonefs  24.5K  9.98M  24.5K  /mypool/myzonefs
```

Next, update the zone configuration. You have to do this with the zone shutdown otherwise the zone configuration changes will not be visible in the zone until the next reboot. Again, this must be done in the global zone. This example assumes that `myzone` is the name of your non-global zone.

```
# zonecfg -z myzone
zonecfg:myzone>  add dataset
zonecfg:myzone:dataset> set name=mypool/myzonefs
zonecfg:myzone:dataset> end
zonecfg:myzone> commit
zonecfg:myzone> exit
```

Next, boot the zone.

```
# zoneadm -z myzone boot
```

For the remainder of this sequence, open multiple terminal window, one in the global zone, and the other in the non-global zone. Log in to the zone. It may take a few seconds for the zone to boot.

```
# zlogin myzone
[Connected to zone 'myzone' pts/1]
Last login: Fri Feb 23 12:55:14 on console
Sun Microsystems Inc.   SunOS 5.11      snv_55  October 2007
```

List the ZFS file systems in the zone.

```
# zfs list
NAME              USED  AVAIL  REFER  MOUNTPOINT
mypool           8.56M   158M  30.5K  /mypool
mypool/myzonefs  24.5K  9.98M  24.5K  /mypool/myzonefs
```

The 10 Mbyte maximum that is set from the external quota and the other file systems in the pool are not visible from the non-global zone. Administering file systems in the non-global zone is just like in the global zone. However, you are limited to operating with the `mypool/myzonefs` space.

Try to create a file system outside of the `mypool/myzonefs` space.

```
# zfs create mypool/myzonefs1
cannot create 'mypool/myzonefs1': permission denied
```

Create a new ZFS file system and list the file systems.

```
#  zfs create mypool/myzonefs/user1
# zfs list
NAME                    USED  AVAIL  REFER  MOUNTPOINT
mypool                 8.59M   158M  30.5K  /mypool
mypool/myzonefs          49K  9.95M  24.5K  /mypool/myzonefs
mypool/myzonefs/user1  24.5K  9.95M  24.5K  /mypool/myzonefs/user1
```

This file system is administered by the zone administrator, not the global administrator.

Apply a quota to the new file system, but it cannot be more than 10 Mbytes because that's the limit available to all of the file systems below `mypool/myzonefs`.

```
# zfs set quota=5m mypool/myzonefs/user1
# zfs list
NAME                    USED  AVAIL  REFER  MOUNTPOINT
mypool                 8.59M   158M  30.5K  /mypool
mypool/myzonefs          50K  9.95M  25.5K  /mypool/myzonefs
mypool/myzonefs/user1  24.5K  4.98M  24.5K  /mypool/myzonefs/user1
```

Properties specified in the global zone are inherited by the local zone.

```
# zfs get -r compression mypool
NAME                   PROPERTY     VALUE              SOURCE
mypool                 compression  off                default
mypool/myzonefs        compression  off                default
mypool/myzonefs/user1  compression  off                default
```

In the global zone, set the compression on the file system.

```
global-zone# zfs set compression=on mypool/myzonefs
```

In the non-global zone, display the compression property for the file systems.

```
non-global zone# zfs get -r compression mypool
NAME                   PROPERTY     VALUE             SOURCE
mypool                 compression  off               default
mypool/myzonefs        compression  on                local
mypool/myzonefs/user1  compression  on                inherited from mypool/myzonefs
```

The compression property has been inherited, but the local zone administrator can change it if desired. The non-global zone administrator can take snapshots, create descendant file systems, backup and create clones, and so on.

In the non-global zone, take a snapshot of the file system.

```
# zfs snapshot mypool/myzonefs@first
# zfs list
NAME                    USED  AVAIL  REFER  MOUNTPOINT
```

```
mypool                  8.59M   158M  30.5K  /mypool
mypool/myzonefs           50K  9.95M  25.5K  /mypool/myzonefs
mypool/myzonefs@first      0      -   25.5K  -
mypool/myzonefs/user1  24.5K  4.98M  24.5K  /mypool/myzonefs/user1
```

Next, stop the zone, remember to do this in the non-global zone, not the global zone.

```
# halt

[Connection to zone 'myzone' pts/1 closed]
```

In the global zone, remove and destroy the zone file system.

```
# zonecfg -z myzone
zonecfg:myzone> remove dataset name=mypool/myzonefs
zonecfg:myzone> commit
zonecfg:myzone> exit
# zfs destroy -r mypool/myzonefs
```

Key Messages – Zones, like ZFS, is great technology. Combining the two is easy and simple to manage system, but with true separation of resources between the zones.

Why This Feature is Important – Most customers want to increase the utilization of their resources. Using Solaris zones is a good way to do this. ZFS allows simple and easy management of file systems within a zone and can support the separation of resources between zones. You can limit the storage that is used by a single zone, without the complexity of having to manage multiple storage devices, which has a high risk of breaking security, if incorrectly configured.

Using ZFS Volumes

Goal of Demo Component – Show that ZFS can also present a volume interface for applications that require raw or block-based data storage, but want to take advantage of the ease of management, redundancy features, and other ZFS improvements.

What You Need to Do Create a ZFS volume that is 50 Mbytes in size.

```
# zfs create -V 50M mypool/myvol
```

Confirm that the volume is created.

```
# zfs list
NAME                     USED  AVAIL  REFER  MOUNTPOINT
mypool                  50.1M   109M  25.5K  /mypool
mypool/myfs             24.5K   109M  24.5K  /mypool/myfs
mypool/myvol            22.5K   159M  22.5K  -
```

Note – No mount point is listed because volumes are not mountable. ZFS guarantees that 50 Mbytes are available. The other file systems' available space has been reduced by 50 Mbytes.

For demo purposes, create a UFS file system on the ZFS volume. For example:

```
# newfs /dev/zvol/rdsk/mypool/myvol
newfs: construct a new file system /dev/zvol/rdsk/mypool/myvol: (y/n)? y
Warning: 2082 sector(s) in last cylinder unallocated
/dev/zvol/rdsk/mypool/myvol:    102366 sectors in 17 cylinders of 48 tracks,
 128 sectors
        50.0MB in 2 cyl groups (14 c/g, 42.00MB/g, 20160 i/g)
super-block backups (for fsck -F ufs -o b=#) at:
 32, 86176,
```

This UFS file system now benefits from the capabilities of ZFS. For example, the volume is fully checksummed and protected with redundancy. You can mount the file system (remember to use the raw block device in `/dev/zvol/dsk/mypool/myvol`) if you want to really show it's doing what it claims. Confirm that the space is used by the volume. For example:

```
# zfs list
NAME                     USED  AVAIL  REFER  MOUNTPOINT
mypool                  50.1M   109M  25.5K  /mypool
mypool/myfs             24.5K   109M  24.5K  /mypool/myfs
mypool/myvol            5.05M   154M  5.05M  -
```

Note that the `myvol` is using approximately 5 Mbytes, which is the amount of data touched by the `newfs` command itself. The other file system cannot encroach on the remaining 45 Mbytes because it is reserved. More on this later.

Lastly, destroy the volume to recover the space.

```
# zfs destroy mypool/myvol
```

Key Messages – ZFS can present volumes to applications that prefer or require the use of volumes. The volumes deliver all of the capabilities of ZFS, such as snapshots, checksummed data, and intelligent prefetch, without the cost and overhead of an additional volume manager. It is even possible (though not recommended) to create "sparse" volumes.

Why This Feature is Important – Some applications only deal with block based storage, especially some database applications. The ZFS volume capability allows these applications to continue to be used (and also benefit from the other ZFS features).

ZFS Demonstration Clean Up

Destroy your ZFS storage pool to release all of the resources for the next demo.

```
# zpool destroy mypool
# zpool list
no pools available
```