Configuring Oracle Solaris ZFS for an Oracle Database

Configuration Best Practices for Oracle Solaris 11.4

September 2020 | Version 4.0

TABLE OF CONTENTS

INTRODUCTION

This document provides planning information and step-by-step instructions for configuring Oracle Solaris ZFS file systems for an Oracle database.

A large number of enterprise customers today run an Oracle relational database on top of storage arrays with different sophistication levels and many customers would like to take advantage of the ZFS file system. Using the Oracle Solaris ZFS file system can reduce the cost and complexity of managing data and storage devices for hosting your Oracle database. Since storage product capabilities often overlap ZFS features, a wide range of options are offered for deployment. This document provides guidance on how to economically deploy an Oracle database on ZFS to maximize data integrity and performance. We start with a brief planning section and follow with step-by-step instructions for configuring and maintaining ZFS file systems for an Oracle database.

PLANNING FOR AN ORACLE DATABASE ON ZFS CONFIGURATION

The following sections provide specific planning information that is needed before you configure ZFS file systems for an Oracle database.

- Oracle Solaris 11.4 is the recommended Solaris OS release
- Storage – Recommendation is to present individual disks from a storage controller to ZFS or use a RAID controller in JBOD mode so ZFS can manage devices directly and provide RAID redundancy. By allowing ZFS to directly provide RAID redundancy allows, it to both report and recover from any data inconsistencies.
- Storage array considerations
    - Confirm with your array vendor that the disk array is not flushing its non-volatile cache after write cache flush requests issued by ZFS.
    - If you must use a RAID array, consider using in JBOD mode.

ZFS Storage Pool Considerations

We recommend using a ZFS redundancy configuration, such as a mirrored storage pool for redundancy protection and best performance for the Oracle Database. With a mirrored storage pool or any ZFS based redundancy configuration, ZFS can automatically detect and repair data storage corruption. We do not recommend RAIDZ redundancy when performance and IOPS are the main criteria for selecting a storage configuration for your database. For more information about creating redundant ZFS storage pools, see the *Oracle Solaris ZFS Administration Guide*.

Review the following steps to prepare your ZFS environment for the Oracle Database:

- Create ZFS storage pool with physical devices from the storage controller that include both HDDs for the main pool devices and SSDs that can be designated to improve performance of database log operation
- If necessary, create LUNs or requests LUNs from your SAN storage to create ZFS storage pools
- Create a single ZFS pool with specialized file systems for each database for both OLTP and OLAP workloads. If Oracle DB (OLTP) performance is critical, then consider creating two separate pools to optimize performance for OLTP workload.

ZFS Storage Pool Sizing

The size of the pool or the sum of the device sizes or LUN sizes configured in the pool should be made to accommodate all the current database sizing, while leaving room for growth and free space. Pool capacity at 80% usually helps to achieve the highest level of performance. In all cases, always operate with at least 5% of free space. In addition to the data devices, the pool is configured with one or more pool log devices that is used to accelerate synchronous operations. The use of a pool log device also helps reduce the internal fragmentation and is greatly recommended.

Managing Pool Space

In an environment with a high rate of data update (or churn), it is advisable to maintain pool capacity at 80%. ZFS is a copy-on-write file system and relocates all writes to free disk space. Keeping some amount of free blocks allows ZFS to easily find space in big chunks. This practice also allows ZFS to aggregate writes and reduce the write IOPS demand on hard disks. Streaming aggregated writes can be 50-100 times less demanding on hard drives than doing small random writes. Therefore, the incentive for keeping free space is high, especially in a high churn environment, such as an active OLTP database. If at some point in the life of the pool, you see the total IOPS grow and the avg I/O size drops to serve the same DB workload, this is a sign that creating free space would help achieve higher performance. Creating free space is often done by deleting older snapshots.

In a data warehouse environment, data is inserted in the DB and normally stays present for long periods with little churn. This use case accommodates a much higher level of occupancy on the storage pool, such as 95% capacity.

As a way to enforce this free space, create a special unused file system with a ZFS file system reservation. This reservation prevents other file systems in the system from growing beyond the desired point. As you reach the limit, ZFS does not accept the creation of new reservations. As that happens, you may decide to reduce the free, unused reservation, reclaim reservations from other file systems, redirect projects to other storage, or simply add storage space.

Adding storage space to a full pool should be done with care. Adding devices to a pool is challenging because it creates a imbalance in free space. To illustrate with an extreme example, if a pool of 10 x 1 TB LUNs is made ~100% full and 2 new 1TB devices are added (to make the pool 80% full), then all new allocation will target the new devices creating a performance problem. A better alternative is to grow the existing LUNs in the storage array and allow ZFS to use the expanded space (see `zpool.1m` `autoexpand` option).

Using Oracle's File Auto Extent

Regarding space consumption, you can use the Oracle database file auto extent feature to instruct the Oracle database not to preallocate the entire data file size, but grow it in controlled increments.[1] Using this method leaves plenty of free space to ZFS at least until the DB has reached its full size, at which point the ZFS pool can be expanded.

In addition to the above reservation, data is located in a smaller range of hard disk addresses leading to greater efficiency. When using this feature, you specify an upper limit to the size of each data file at the Oracle configuration level. At the ZFS level, it is also possible to set up a corresponding ZFS file system reservation, which *guarantees* never running out of space. This space can be reclaimed at any time if it turns out the sum of the data file maximum size was over provisioned.

While this feature allows you to run with more free space in the pool and brings some flexibility in operations, it does come with a downside, in that it is not always compatible with every database application.

Be aware also that when the database needs to run through an extension, it causes some disruption to operations. As a remedy using a small next auto extent, typically of 1MB, allows a very short allocation time and limits the impact of that disruption for the application. While a ZFS reservation helps you deal with space concerns, your final experience is the key in determining if the feature is desirable in your environment.

For more information about using ZFS quotas and reservation, see the *Oracle Solaris ZFS Administration Guide*.

ZFS File System Considerations

Previous recommendations were to set the Oracle DB tables (data files) file system `recordsize` to `db_block_size`. The current recommendation is to match the ZFS `recordsize` for the data file file system

[1] For more information, see http://www.dba-oracle.com/t_alter_autoextend_on.htm.

to the average network I/O size as closely as possible. This size might be significantly larger than `db_block_size`.

- For OLTP workloads the general recommendation is 32K
- For data warehouse or OLAP workloads the record size of the datafile share should be 128K

Important Note: Keep in mind that modifying the ZFS file system `recordsize` parameter affects only the files created after the change. To change the size that is used by an existing data file, you must first change the `recordsize` property of the file system used to store the file, and then copy the files back into the file system.

For additional performance in the case of a large database, you might use different `db_block_size` values for different database components. In these cases, you should use one ZFS filesystem per `db_block_size` and match file system's record size to `db_block_size`. Also, different DB file types should be segregated to specific filesystems of a pool as described below.

Improving Database Writing and Caching Performance

Use the following properties to improve Oracle database performance:

- Set the `logbias` property per file system
- Use the `primarycache` property to control what is stored in main memory cache

Tuning Synchronous Write Activity

ZFS manages synchronous writes internally with the ZFS intent log (ZIL). By default, synchronous writes are stored in the ZIL at low latency and shortly thereafter (within approximately 5 seconds), ZFS does a bulk update of the pool state by committing a transaction group.[2]

Another way exists for the ZIL to handle synchronous write activity, which is of slightly higher latency, but also involves almost half as much traffic on the channel to the storage array. This mode is triggered by the `logbias=throughput` value. This mode is recommended for file systems holding Oracle data files since writes to those files are highly threaded background operations from the database perspective, which does not warrant the utmost latency requirements. Using `logbias=throughput` for data files frees up the storage resource and as a consequence, allows the redolog to be handled with lower latency. The redo log itself should generally not be tuned for this property.

Recommended File System Settings for OLTP Workloads

FILE SYSTEM	RECORDSIZE	LOGBIAS	PRIMARYCACHE	COMPRESSION
Tables	32K	latency	all (data and metadata)	LZ4
Redo	128K	latency	Do not use	off (default)
Index	32K	throughput	all (data and metadata)	off (default)
Undo	1 MB	throughput	all (data and metadata)	off (default)
Temp	128K	latency	Do not use	off (default)
Archive	1 MB	throughput	Do not use	LZ4

[2] Note that data for a transaction group is kept in memory and not read from the ZIL, which is only used in case of a system failure.

ZFS File System Property Settings for Data Warehouse Workloads

Some databases hold a mix of volatile and historical data. The distinction is that historical data is subject to a reduced rate of modification compared to the volatile nature of evolving OLTP data. The historical data has effectively become frozen. Such data can still be kept in the main database for the purpose of reporting and trend analysis.

Historic data that is generally static and that is read in database scan operations should benefit from being stored on a large ZFS `recordsize`, and if necessary, compressed by ZFS using LZ4 compression, to reduce the on-disk footprint. Separately, index data might benefit from using a larger `db_block_size` (32 KB). For historical data, we also advocate setting the `logbias` file system property to throughput. Note also, that contrasted to volatile data, file systems holding historical data on ZFS interact well with storage based dynamic provisioning, the amount of data stored after compression is closer to the amount provisioned by the storage. The data warehouse workloads usually have creation and modification during integration phase, but have nearly no modification on integrated data. This modification pattern allows data warehouse databases to push up the fill percentage of a ZFS pool to the 95% capacity without adverse effect.

Recommended File System Settings for OLAP Workloads

FILE SYSTEM	RECORDSIZE	LOGBIAS	PRIMARYCACHE	COMPRESSION
Tables	`128K`	`throughput`	`Do not use`	`LZ4`
Index	`32K`	`throughput`	`all (data and metadata)`	`off (default)`

Oracle Large Object (LOB) Considerations

The Oracle LOB, CLOB, NCLOB or BLOB data types are somewhat less common, but when present can benefit from special handling. Because this type of data leads to large read and write I/Os, such as a full table scan, setting up these objects can be done by using the same guidelines as the ones given for data warehouses.

For these applications, we can use the Oracle SQL syntax to set apart the LOB columns in specific tablespaces. The LOB data types are frequently large, many MB, and are split over many Oracle data blocks. Consider creating a separate file system and specify a larger record size for the LOB tablespace.[3] You may use a ZFS `recordsize` of 128 KB or even 1MB, but we advise to use a ZFS `recordsize` no greater that the average size of individual LOBs.

Recommended File System Settings for LOB Applications

FILE SYSTEM	RECORDSIZE	LOGBIAS	PRIMARYCACHE	COMPRESSION
LOB	`1M`	`throughput`	`all (data and metadata)`	`LZ4`

Using ZFS Secondary Cache (L2ARC)

You can add SSDs as secondary cache (L2ARC) devices in a storage pool to store a cached copy of disk content for quick access and serve to increase the total random read throughput of the storage.

Setting the `secondarycache` property determines which file systems will use the secondary cache.

Using a secondary cache is beneficial for read-latency sensitive workloads, for both index and database blocks. As usual, a cache layer is most effective when cache hit rates are high, implying that the total size of cache devices

[3] For more information, see Configuring Multiple Block Sizes for an Oracle Database.

should be sized according to the application's warm working set. Moreover, in an environment where performance peaks when physical disks are saturated with random reads, the caching devices offer a venue for additional read IOPS.

- Caching devices do retain data between system reboots
- Enabling the secondary cache for redo logs is not recommended

Allow for days of operation to gauge their effectiveness in your environment.

Using ZFS Snapshots with the Oracle Database

Creating ZFS snapshots and clones of your Oracle database can reduce the time spent on data management.

A key ZFS feature is creating a snapshot of a given file system or sets of file systems. Creating a snapshot is nearly instantaneous and allows you to operate on a read-only point in time representation of a ZFS file system. A snapshot can be used to keep previous images of the database at a very modest space cost. Multiple snapshots can be generated during the day, each representing a potential recovery point. Generally, a snapshot and the file system from which it originated actually point to the same physical blocks. Therefore, the cost of taking snapshots of disk blocks is equal to only data that has been modified in the database since the last snapshot was taken.

Blocks that are referenced from either the snapshot or from the database image actually share ZFS cache buffers for a reduced memory footprint.

Taking a snapshot of a database can involve placing the database in hot-backup mode just for the instant necessary to capture the ZFS snapshot. The database can then be returned to normal operation.

Snapshots can also be used to replicate the database to a remote host by using the `zfs send` and `zfs receive` commands. When creating a series of snapshots, only data that was updated between the two snapshots needs to be transferred over the network. If done regularly, the *incremental* data to be transferred is either in the cache or on the disk with blocks that have physical proximity to one another for more efficient operation.

For backups, either the snapshot captured on the primary host or the replicated snapshot on a remote host can serve as the source of a backup. The remote host replicates the full snapshot while the primary host transfers the incremental snapshot changes. It is also possible to clone and mount a copy of the database on a replicated site and share the largest part of the replicated volume. The replicated database is some hours, days, behind the production database and can be used for business intelligence queries or as a pre-production environment.

Sending and receiving ZFS snapshot streams is an alternative to the commonly used storage-based replication. With storage-based replication of an ZFS storage pool, be aware you **must** configure all LUNs belonging to a ZFS storage pool to be in the *same storage coherency group* in the SAN array. Failure to do so prevents the replicated LUNs from being imported by ZFS on the replication site. On the other hand, a successful receive operation is immediately accessible on the replication site. In addition, during reception, previously replicated file systems snapshots are still accessible and ready for use. For more information, see the *Oracle Solaris ZFS Administration Guide*.

ZFS Cloning for Test and Development

While snapshots are read-only file systems, it is also possible to create a clone of any ZFS snapshot. A *clone* is a full read/write accessible fork of the original database. Therefore, by using either the primary or a replicated storage system, you can cheaply create hundreds of copies of your *up-to-date* database to run test and development workloads under real data conditions. Clones, snapshots, and the original file systems running in the same storage, share cached blocks between each other.

For more information about sending and receiving snapshots, see the *Oracle Solaris ZFS Administration Guide*.

Additional System Configuration

Memory and Swap Considerations

The ZFS cache grows to consume most of unused memory and shrinks when applications generate memory demand. But while ZFS can shrink its cache quickly, it does take time for the free memory list to be restored. When applications demand memory, usually at application startup, memory is consumed faster than the pace with which free memory is generated. Have enough swap configured to allow applications to start up smoothly. The swap space acts as a buffer zone for the application's allocation of pages for itself while the Oracle Solaris OS is converting freed buffers into free pages. Having sufficient swap space prevents standard application from aborting due to ENOMEM errors.

Configuring an amount of swap equivalent to *all* of system memory is always enough (in fact excessively so) for this purpose. This swap space is not expected to be used, but is needed as a reservation area. Using a much smaller amount of swap is normally sufficient to cover memory demand bursts from regular applications.

For information about increasing swap space, see the *Oracle Solaris ZFS Administration Guide*.

ISM or DISM Shared Memory Considerations

Applications that use ISM or DISM shared memory (`shmget(2)`, `shmat(2)`) should be set up to retry when getting errors. If the amount requested is very large, then the timeout for such operations needs to be scaled appropriately. It takes minutes to convert hundreds of GB of freed buffers into free memory.

Another way to deal with bursts of memory demand is to cap the maximum ZFS cache by setting `zfs:zfs_arc_max` in `/etc/system`. This is especially appropriate if the memory consumption of applications is known in advance. ZFS is then instructed to never grow beyond a desired size. See Tuning the ZFS ARC (zfs_arc_max and zfs_arc_max_percent).

Another benefit of limiting the ZFS cache is to preserve the set of large memory pages that can then be used by the database's ISM/DISM segments. When those pages are available, some high intensity databases are observed to perform better. Note that the preferred way to insure availability of large pages for a critical database is to startup the instance shortly after boot.

Here are few considerations of trying to balance memory use between the Oracle database and ZFS.

A database SGA is a great place to cache data. When a system is devoted to running a single database instance, it usually makes sense to configure a maximum amount of memory to the SGA. However, many large scale systems run a number of independent database and it is not practical to maximize the SGA for each one. In these cases, it makes sense to size each of the SGA a minimum, and then allow ZFS to do the unified system caching in front of the disk subsystem.

In cases where the SGA is maximized and is the primary location of caching information for a database, it is possible to limit the ZFS cache size. When the cache size is set very low, it is possible to adjust the cache by setting `primarycache=metadata` on some ZFS file systems. This file system property reclaims more I/O on the storage. However, file systems with `primarycache` set to metadata negate the benefit of prefetching, which can be a problem during any long read as full scan, backup, restore (`archivelog`). Therefore, we recommend undoing this setting during a restore operation because it relies on prefetching for performance. Also consider that a file system holding an index file normally benefits significantly from file system level caching. Generally, those file systems should not have the `primarycache` property set.

Set `primarycache=metadata` on the archivelog file system, but it can slow down the copy of the archivelog file system to external storage. So, set `primarycache=all` on the archivelog file system for a restore operation.

Configuring Multiple Block Sizes for an Oracle Database

You can define a specific tablespace for the LOB using a 32 KB block size tablespace, which also requires some 32 KB cache in the SGA (System Global Area), as we do for a data warehouse environment, by using syntax similar to the following:

```
zfs create -o recordsize=32k,mountpoint=/my_db_path/data_32k dbpool/data_32k
zfs create -o recordsize=1M,mountpoint=/my_db_path/data_1M dbpool/data_1Mk
```

In `init.ora`, you would find numbers similar to the following example:

```
DB_BLOCK_SIZE=8192
SGA_TARGET=20G
DB_32K_CACHE_SIZE=400M
```

In SQL, you would find numbers as follows:

```
CREATE TABLESPACE data_32k_example
BLOCKSIZE 32K
DATAFILE '/my_db_path/data_32k/image_lob01.dbf' SIZE 1G
AUTOEXTEND ON NEXT 10M MAXSIZE 20G;
```

Tuning the ZFS ARC (`zfs_arc_max and zfs_arc_max_percent`)

The `zfs_arc_max` parameter is in bytes and accepts decimal or hexadecimal values. It is highly recommended to put a date and justification as comments alongside the tuning.

The following text shows how to set this parameter to 16 GB, as an example:

Add this text with appropriate values to `/etc/system` and then reboot the system.

```
* August 19, 2020; RB.
* Cap the ARC to 16GB reserving 500GB for applications
* set zfs:zfs_arc_max=0x400000000
```

Since Solaris 11.4 SRU26, the preferred way to limit the arc cache is through a new `zfs_arc_max_percent` which expresses the limit as a percentage of total memory. The default is 90% but ZFS will only grow to the point where is sees memory pressure from applications and other kernel demands. Setting `zfs_arc_max_percent` to 70% has been shown to help reduce disruptive events (where we see a large ARC reduction under memory pressure along with multi-second periods of no I/Os). True the smaller ARC may lead to extra cache misses and more IOPS requested to the storage devices but this can be the acceptable price in exchange of the extra performance stability that a smaller ARC provides.

Add this text with appropriate values to `/etc/system` and then reboot the system.

```
* August 19, 2020; RB.
* Cap the ARC to 70% of physmem
* set zfs:zfs_arc_max_percent=70
```

Display the current memory size in bytes that is used as ZFS cache:

```
# kstat zfs::arcstats:size
```

Monitor ZFS cache sizes with the preceding command and re-adjust the `zfs_arc_max_percent` parameter when needed. If the `vmstat` command shows high level of free memory, consider increasing the value of `zfs_arc_max_percent` to allow for better caching.

HOW TO CONFIGURE ZFS FILE SYSTEMS FOR AN ORACLE DATABASE

After you have reviewed the preceding planning section, use the steps below to configure ZFS for an Oracle database.

1. Confirm that you are running the latest Oracle Solaris 11.4 release.

```
# cat /etc/release
                                   Oracle Solaris 11.4 X86
                    Copyright (c) 1983, 2020, Oracle and/or its affiliates.
                                  Assembled 28 August 2020
```

2. Create a mirrored pool for best Oracle Database performance.

```
# zpool create dbpool mirror c0t5000CCA02F02A15Cd0 c0t5000CCA02F02ADF4d0 mirror
c0t5000CCA02F027D84d0 c0t5000CCA02F0219C4d0 mirror c0t5000CCA02F0277A0d0
c0t5000CCA02F0284E4d0 mirror c0t5000CCA02F02092Cd0 c0t5000CCA02F026694d0  cache mirror
c0t5000CCA0536CBC84d0 c0t5000CCA0536CC430d0 log mirror c0t5000CCA04E1C1C20d0
c0t5000CCA04EB4C30Cd0 spare c0t5000CCA02F028604d0 c0t5000CCA02F029598d
# zpool status dbpool
pool: dbpool
state: ONLINE
scan: none requested
config:

       NAME                       STATE      READ WRITE CKSUM
       dbpool                     ONLINE        0     0     0
         mirror-0                 ONLINE        0     0     0
           c0t5000CCA02F02A15Cd0  ONLINE        0     0     0
           c0t5000CCA02F02ADF4d0  ONLINE        0     0     0
         mirror-1                 ONLINE        0     0     0
           c0t5000CCA02F027D84d0  ONLINE        0     0     0
           c0t5000CCA02F0219C4d0  ONLINE        0     0     0
         mirror-2                 ONLINE        0     0     0
           c0t5000CCA02F0277A0d0  ONLINE        0     0     0
           c0t5000CCA02F0284E4d0  ONLINE        0     0     0
         mirror-3                 ONLINE        0     0     0
           c0t5000CCA02F02092Cd0  ONLINE        0     0     0
           c0t5000CCA02F026694d0  ONLINE        0     0     0
       caches
         mirror-5                 ONLINE        0     0     0
           c0t5000CCA0536CBC84d0  ONLINE        0     0     0
           c0t5000CCA0536CC430d0  ONLINE        0     0     0
       logs
         mirror-4                 ONLINE        0     0     0
           c0t5000CCA04E1C1C20d0  ONLINE        0     0     0
           c0t5000CCA04EB4C30Cd0  ONLINE        0     0     0
       spares
         c0t5000CCA02F028604d0    AVAIL
         c0t5000CCA02F029598d0    AVAIL

errors: No known data errors
```

3. Create ZFS file systems and set the specific file system properties by following guidelines provided previously.

```
# zfs create -o recordsize=32k -o mountpoint=/my_db_path/data dbpool/data
# zfs set logbias=latency dbpool/data
# zfs get primarycache,recordsize,logbias dbpool/data
NAME           PROPERTY      VALUE         SOURCE
dbpool/data    primarycache  all           default
dbpool/data    recordsize    32K           local
dbpool/data    logbias       latency       local
```

```
# zfs create -o recordsize=32k -o mountpoint=/my_db_path/index dbpool/index
# zfs set logbias=throughput dbpool/index
# zfs get primarycache,recordsize,logbias dbpool/index
NAME            PROPERTY      VALUE         SOURCE
dbpool/index    primarycache  all           default
dbpool/index    recordsize    32K            local
dbpool/index    logbias       throughput    local
```

4. Create file systems for temporary and undo table spaces, using the default `recordsize` and `primarycache` values.

```
# zfs create -o recordsize=1m -o mountpoint=/my_db_path/temp dbpool/temp
# zfs set logbias=throughput dbpool/temp
# zfs create -o recordsize=1m -o mountpoint=/my_db_path/undo dbpool/undo
# zfs set logbias=throughput dbpool/undo
```

5. Create a file system for redo logs. Use default file system values for `primarycache` and `recordsize`.

```
# zfs create -o mountpoint=/my_db_path/redo redopool/redo
# zfs set logbias=latency redopool/redo
```

6. Create a file system for the `archivelog` files, enable compression and use the default value for `recordsize`.

```
# zfs create -o compression=lz4 -o recordsize=1M -o mountpoint=/my_db_admin_path/archive
dbpool/archive
# zfs get primarycache,recordsize,compressratio,compression,available,
used,quota dbpool/archive
NAME              PROPERTY       VALUE         SOURCE
dbpool/archive    primarycache   all           local
dbpool/archive    recordsize     1M            local
dbpool/archive    compressratio  1.32x         -
dbpool/archive    compression    on            local
dbpool/archive    available      40.0G         -
dbpool/archive    used           10.0G         -
dbpool/archive    quota          50G           local
```

7. Consider the following steps to help maintain disk space and to create snapshots.

   a) Set a reservation on the main database file system to reserve the required database file system space.

   ```
   # zfs set reservation=200gb dbpool/data
   ```

   b) Set a reservation on a dummy file system to reserve 10% of pool space to maintain pool performance.

   ```
   # zfs set reservation=20gb dbpool/freespace
   ```

   c) Consider setting a quota on the archive file system if other data is also archived in the archive pool.

   ```
   # zfs set quota=40g archivepool/archive
   ```

FOR MORE INFORMATION

For more information on Oracle Solaris 11.4 and ZFS technology, see the following references.

REFERENCES FOR MORE INFORMATION ABOUT ORACLE SOLARIS 11 AND ZFS TECHNOLOGY

Oracle Solaris 11	http://www.oracle.com/solaris
Oracle Solaris 11 technical content on OTN	http://www.oracle.com/technetwork/server-storage/solaris11/overview/index.html
Oracle Solaris 11 documentation Oracle Solaris ZFS Administration Guide	http://www.oracle.com/technetwork/server-storage/solaris11/documentation/index.html https://docs.oracle.com/cd/E37838_01/html/E61017/index.html
Oracle Solaris 11 Articles	http://www.oracle.com/technetwork/server-storage/solaris11/documentation/how-to-517481.html

CONNECT WITH US

Call +1.800.ORACLE1 or visit oracle.com.
Outside North America, find your local office at oracle.com/contact.

blogs.oracle.com facebook.com/oracle twitter.com/oracle

Configuring Oracle Solaris ZFS for an Oracle Database September 2020
Original Authors: Roch Bourbonnais, Alain Chereau, and Cindy Swearingen

Updated by Cindy Swearingen