NAME

zpoolprops - properties of ZFS storage pools

DESCRIPTION

Each pool has several properties associated with it. Some properties are read-only statistics while others are configurable and change the behavior of the pool.

User properties have no effect on ZFS behavior. Use them to annotate pools in a way that is meaningful in your environment. For more information about user properties, see the *User Properties* section.

The following are read-only properties:

allocated Amount of storage used within the pool. See **fragmentation** and **free** for more information.

bcloneratio The ratio of the total amount of storage that would be required to store all the cloned blocks without cloning to the actual storage used. The **bcloneratio** property is calculated as:

((**bclonesaved + bcloneused**) *** 100**) **/ bcloneused**

bclonesaved The amount of additional storage that would be required if block cloning was not used.

bcloneused The amount of storage used by cloned blocks.

capacity Percentage of pool space used. This property can also be referred to by its shortened column name, **cap**.

expandsize Amount of uninitialized space within the pool or device that can be used to increase the total capacity of the pool. On whole-disk vdevs, this is the space beyond the end of the GPT - typically occurring when a LUN is dynamically expanded or a disk replaced with a larger one. On partition vdevs, this is the space appended to the partition after it was added to the pool - most likely by resizing it in-place. The space can be claimed for the pool by bringing it online with **autoexpand=on** or using **zpool online -e**.

fragmentation The amount of fragmentation in the pool. As the amount of space **allocated** increases, it becomes more difficult to locate **free** space. This may result in lower write performance compared to pools with more unfragmented free space.

free The amount of free space available in the pool. By contrast, the zfs(8) **available** property describes how much new data can be written to ZFS filesystems/volumes. The zpool **free** property is not generally useful for this purpose, and can be substantially more than the zfs **available** space. This discrepancy is due to several factors, including raidz parity; zfs reservation, quota, refreservation, and refquota properties; and space set aside by **spa_slop_shift** (see zfs(4) for more information).

freeing After a file system or snapshot is destroyed, the space it was using is returned to the pool asynchronously. **freeing** is the amount of space remaining to be reclaimed. Over time **freeing** will decrease while **free** increases.

guid A unique identifier for the pool.

health The current health of the pool. Health can be one of **ONLINE**, **DEGRADED**, **FAULTED**, **OFFLINE, REMOVED**, **UNAVAIL**.

leaked Space not released while **freeing** due to corruption, now permanently leaked into the pool.

load_guid A unique identifier for the pool. Unlike the **guid** property, this identifier is generated every time we load the pool (i.e. does not persist across imports/exports) and never changes while the pool is loaded (even if a **reguid** operation takes place).

size Total size of the storage pool.

unsupported@*guid* Information about unsupported features that are enabled on the pool. See zpool-features(7) for details.

The space usage properties report actual physical space available to the storage pool. The physical space can be different from the total amount of space that any contained datasets can actually use. The amount of space used in a raidz configuration depends on the characteristics of the data being written. In addition, ZFS reserves some space for internal accounting that the zfs(8) command takes into account, but the **zpoolprops** command does not. For non-full pools of a reasonable size, these effects should be invisible. For small pools, or pools that are close to being completely full, these discrepancies may become more noticeable.

The following property can be set at creation time and import time:

altroot Alternate root directory. If set, this directory is prepended to any mount points within the pool. This can be used when examining an unknown pool where the mount points cannot be trusted,

or in an alternate boot environment, where the typical paths are not valid. **altroot** is not a persistent property. It is valid only while the system is up. Setting **altroot** defaults to using **cachefile**=**none**, though this may be overridden using an explicit setting.

The following property can be set only at import time:

readonly=**on**|**off**

If set to **on**, the pool will be imported in read-only mode. This property can also be referred to by its shortened column name, **rdonly**.

The following properties can be set at creation time and import time, and later changed with the **zpool set** command:

ashift=*ashift*

Pool sector size exponent, to the power of **2** (internally referred to as **ashift**). Values from 9 to 16, inclusive, are valid; also, the value 0 (the default) means to auto-detect using the kernel's block layer and a ZFS internal exception list. I/O operations will be aligned to the specified size boundaries. Additionally, the minimum (disk) write size will be set to the specified size, so this represents a space/performance trade-off. For optimal performance, the pool sector size should be greater than or equal to the sector size of the underlying disks. The typical case for setting this property is when performance is important and the underlying disks use 4KiB sectors but report 512B sectors to the OS (for compatibility reasons); in that case, set **ashift**=**12** (which is **1<<12** = **4096**). When set, this property is used as the default hint value in subsequent vdev operations (add, attach and replace). Changing this value will not modify any existing vdev, not even on disk replacement; however it can be used, for instance, to replace a dying 512B sectors disk with a newer 4KiB sectors device: this will probably result in bad performance but at the same time could prevent loss of data.

autoexpand=**on**|**off**

Controls automatic pool expansion when the underlying LUN is grown. If set to **on**, the pool will be resized according to the size of the expanded device. If the device is part of a mirror or raidz then all devices within that mirror/raidz group must be expanded before the new space is made available to the pool. The default behavior is **off**. This property can also be referred to by its shortened column name, **expand**.

autoreplace=**on**|**off**

Controls automatic device replacement. If set to **off**, device replacement must be initiated by the administrator by using the **zpool replace** command. If set to **on**, any new device, found in the same physical location as a device that previously belonged to the pool, is automatically formatted and replaced. The default behavior is **off**. This property can also be referred to by its

shortened column name, **replace**. Autoreplace can also be used with virtual disks (like device mapper) provided that you use the /dev/disk/by-vdev paths setup by vdev_id.conf. See the vdev_id(8) manual page for more details. Autoreplace and autoonline require the ZFS Event Daemon be configured and running. See the zed(8) manual page for more details.

autotrim=**on**|**off**

When set to **on** space which has been recently freed, and is no longer allocated by the pool, will be periodically trimmed. This allows block device vdevs which support BLKDISCARD, such as SSDs, or file vdevs on which the underlying file system supports hole-punching, to reclaim unused blocks. The default value for this property is **off**.

Automatic TRIM does not immediately reclaim blocks after a free. Instead, it will optimistically delay allowing smaller ranges to be aggregated into a few larger ones. These can then be issued more efficiently to the storage. TRIM on L2ARC devices is enabled by setting **l2arc_trim_ahead > 0**.

Be aware that automatic trimming of recently freed data blocks can put significant stress on the underlying storage devices. This will vary depending of how well the specific device handles these commands. For lower-end devices it is often possible to achieve most of the benefits of automatic trimming by running an on-demand (manual) TRIM periodically using the **zpool trim** command.

bootfs=**(unset)**|*pool*[/*dataset*]

Identifies the default bootable dataset for the root pool. This property is expected to be set mainly by the installation and upgrade programs. Not all Linux distribution boot processes use the bootfs property.

cachefile=*path*|**none**

Controls the location of where the pool configuration is cached. Discovering all pools on system startup requires a cached copy of the configuration data that is stored on the root file system. All pools in this cache are automatically imported when the system boots. Some environments, such as install and clustering, need to cache this information in a different location so that pools are not automatically imported. Setting this property caches the pool configuration in a different location that can later be imported with **zpool import -c**. Setting it to the value **none** creates a temporary pool that is never cached, and the "" (empty string) uses the default location.

Multiple pools can share the same cache file. Because the kernel destroys and recreates this file when pools are added and removed, care should be taken when attempting to access this file. When the last pool using a **cachefile** is exported or destroyed, the file will be empty.

comment=*text*

A text string consisting of printable ASCII characters that will be stored such that it is available even if the pool becomes faulted. An administrator can provide additional information about a pool using this property.

compatibility=**off**|**legacy**|*file*[,*file*]<?>

Specifies that the pool maintain compatibility with specific feature sets. When set to **off** (or unset) compatibility is disabled (all features may be enabled); when set to **legacy**no features may be enabled. When set to a comma-separated list of filenames (each filename may either be an absolute path, or relative to */etc/zfs/compatibility.d* or */usr/share/zfs/compatibility.d*) the lists of requested features are read from those files, separated by whitespace and/or commas. Only features present in all files may be enabled.

See zpool-features(7), zpool-create(8) and zpool-upgrade(8) for more information on the operation of compatibility feature sets.

dedupditto=*number*

This property is deprecated and no longer has any effect.

delegation=**on**|**off**

Controls whether a non-privileged user is granted access based on the dataset permissions defined on the dataset. See zfs(8) for more information on ZFS delegated administration.

failmode=**wait**|**continue**|**panic**

Controls the system behavior in the event of catastrophic pool failure. This condition is typically a result of a loss of connectivity to the underlying storage device(s) or a failure of all devices within the pool. The behavior of such an event is determined as follows:

wait Blocks all I/O access until the device connectivity is recovered and the errors are cleared with **zpool clear**. This is the default behavior.

continue Returns EIO to any new write I/O requests but allows reads to any of the remaining healthy devices. Any write requests that have yet to be committed to disk would be blocked.

panic Prints out a message to the console and generates a system crash dump.

feature@*feature_name*=**enabled**

The value of this property is the current state of *feature_name*. The only valid value when setting this property is **enabled** which moves *feature_name* to the enabled state. See

zpool-features(7) for details on feature states.

listsnapshots=**on**|**off**

Controls whether information about snapshots associated with this pool is output when **zfs list** is run without the **-t** option. The default value is **off**. This property can also be referred to by its shortened name, **listsnaps**.

multihost=**on**|**off**

Controls whether a pool activity check should be performed during **zpool import**. When a pool is determined to be active it cannot be imported, even with the **-f** option. This property is intended to be used in failover configurations where multiple hosts have access to a pool on shared storage.

Multihost provides protection on import only. It does not protect against an individual device being used in multiple pools, regardless of the type of vdev. See the discussion under **zpool create**.

When this property is on, periodic writes to storage occur to show the pool is in use. See **zfs_multihost_interval** in the zfs(4) manual page. In order to enable this property each host must set a unique hostid. See genhostid(1) zgenhostid(8) spl(4) for additional details. The default value is **off**.

version=*version*

The current on-disk version of the pool. This can be increased, but never decreased. The preferred method of updating pools is with the **zpool upgrade** command, though this property can be used when a specific version is needed for backwards compatibility. Once feature flags are enabled on a pool this property will no longer have a value.

User Properties

In addition to the standard native properties, ZFS supports arbitrary user properties. User properties have no effect on ZFS behavior, but applications or administrators can use them to annotate pools.

User property names must contain a colon ("**:**") character to distinguish them from native properties. They may contain lowercase letters, numbers, and the following punctuation characters: colon ("**:**"), dash ("**-**"), period ("**.**"), and underscore ("_"). The expected convention is that the property name is divided into two portions such as *module*:*property*, but this namespace is not enforced by ZFS. User property names can be at most 256 characters, and cannot begin with a dash ("**-**").

When making programmatic use of user properties, it is strongly suggested to use a reversed DNS domain name for the *module* component of property names to reduce the chance that two independently-

developed packages use the same property name for different purposes.

The values of user properties are arbitrary strings and are never validated. All of the commands that operate on properties (**zpool list**, **zpool get**, **zpool set**, and so forth) can be used to manipulate both native properties and user properties. Use **zpool set** *name*= to clear a user property. Property values are limited to 8192 bytes.