Aaron Toponce

{ 2012 12 12 }

ZFS Administration, Part VII- Zpool Properties

Table of Contents

Continuing from our last post on scrubbing and resilvering data in zpools, we move on to changing properties in the zpool.

Motivation

With ext4, and many filesystems in GNU/Linux, we have a way for tuning various flags in the filesystem. Things like setting labels, default mount options, and other tunables. With ZFS, it's no different, and in fact, is far more verbose. These properties allow us to modify all sorts of variables, both for the pool, and for the datasets it contains. Thus, we can "tune" the filesystem to our liking or needs. However, not every property is tunable. Some are read-only. But, we'll define what each of the properties are and how they affect the pool. Note, we are only looking at zpool properties, and we will get to ZFS dataset properties when we reach the dataset subtopic.

Zpool Properties

- allocated: The amount of data that has been committed into the pool by all of the ZFS datasets. This setting is read-only.
- altroot: Identifies an alternate root directory. If set, this directory is prepended to any mount points within the pool. This property can be used when examining an unknown pool, if the mount points cannot be trusted, or in an alternate boot environment, where the typical paths are not valid.Setting altroot defaults to using "cachefile=none", though this may be overridden using an explicit setting.
- ashift: **Can only be set at pool creation time.** Pool sector size exponent, to the power of 2. I/O operations will be aligned to the specified size boundaries. Default value is "9", as 2^9 = 512, the standard sector size operating system utilities use for reading and writing data. For advanced format drives with 4 KiB boundaries, the value should be set to "ashift=12", as 2^12 = 4096.
- autoexpand: **Must be set before replacing the first drive in your pool.** Controls automatic pool expansion when the underlying LUN is grown. Default is "off". After all drives in the pool have been replaced with larger drives, the pool will automatically grow to the new size. This setting is a boolean, with values either "on" or "off".
- autoreplace: Controls automatic device replacement of a "spare" VDEV in your pool. **Default is set to "off".** As such, device replacement must be initiated manually by using the "zpool replace" command. This setting is a boolean, with values either "on" or "off".
- bootfs: Read-only setting that defines the bootable ZFS dataset in the pool. This is typically set by an installation program.
- cachefile: Controls the location of where the pool configuration is cached. When importing a zpool on a system, ZFS can detect the drive geometry using the metadata on

the disks. However, in some clustering environments, the cache file may need to be stored in a different location for pools that would not automatically be imported. Can be set to any string, but for most ZFS installations, the default location of "/etc/zfs/zpool.cache" should be sufficient.

- capacity: Read-only value that identifies the percentage of pool space used.
- comment: A text string consisting of no more than 32 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 setting.
- dedupditto: Sets a block deduplication threshold, and if the reference count for a deduplicated block goes above the threshold, a duplicate copy of the block is stored automatically. The default value is 0. Can be any positive number.
- dedupratio: Read-only deduplication ratio specified for a pool, expressed as a multiplier
- delegation: Controls whether a non-privileged user can be granted access permissions that are defined for the dataset. The setting is a boolean, defaults to "on" and can be "on" or "off".
- expandsize: Amount of uninitialized space within the pool or device that can be used to increase the total capacity of the pool. Uninitialized space consists of any space on an EFI labeled vdev which has not been brought online (i.e. "zpool online -e"). This space occurs when a LUN is dynamically expanded.
- failmode: 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. 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.
- free: Read-only value that identifies the number of blocks within the pool that are not allocated.
- guid: Read-only property that identifies the unique identifier for the pool. Similar to the UUID string for ext4 filesystems.
- health: Read-only property that identifies the current health of the pool, as either ONLINE, DEGRADED, FAULTED, OFFLINE, REMOVED, or UNAVAIL.
- listsnapshots: Controls whether snapshot information that is associated with this pool is displayed with the "zfs list" command. If this property is disabled, snapshot information can be displayed with the "zfs list -t snapshot" command. The default value is "off". Boolean value that can be either "off" or "on".
- readonly: Boolean value that can be either "off" or "on". Default value is "off". Controls setting the pool into read-only mode to prevent writes and/or data corruption.
- size: Read-only property that identifies the total size of the storage pool.
- version: Writable setting that identifies the current on-disk version of the pool. Can be any value from 1 to the output of the "zpool upgrade -v" command. This property can be used when a specific version is needed for backwards compatibility.

Getting and Setting Properties

There are a few ways you can get to the properties of your pool- you can get all properties at once, only one property, or more than one, comma-separated. For example, suppose I wanted to get just the health of the pool. I could issue the following command:

```
# zpool get health tank
NAME  PROPERTY  VALUE   SOURCE
tank  health    ONLINE  -
```

If I wanted to get multiple settings, say the health of the system, how much is free, and how much is allocated, I could issue this command instead:

```
# zpool get health,free,allocated tank
NAME  PROPERTY   VALUE   SOURCE
tank  health     ONLINE  -
tank  free       176G    -
tank  allocated  32.2G   -
```

And of course, if I wanted to get all the settings available, I could run:

```
# zpool get all tank
NAME  PROPERTY       VALUE       SOURCE
tank  size           208G        -
tank  capacity       15%         -
tank  altroot        -           default
tank  health         ONLINE      -
tank  guid           16951123779703469 70  default
```

```
tank  version        28          default
tank  bootfs         -           default
tank  delegation     on          default
tank  autoreplace    off         default
tank  cachefile      -           default
tank  failmode       wait        default
tank  listsnapshots  off         default
tank  autoexpand     off         default
tank  dedupditto     0           default
tank  dedupratio     1.00x       -
tank  free           176G        -
tank  allocated      32.2G       -
tank  readonly       off         -
tank  ashift         0           default
tank  comment        -           default
tank  expandsize     0           -
```

Setting a property is just as easy. However, there is a catch. For properties that require a string argument, there is no way to get it back to default. At least not that I am aware of. With the rest of the properties, if you try to set a property to an invalid argument, an error will print to the screen letting you know what is available, but it will not notify you as to what is default. However, you can look at the 'SOURCE' column. If the value in that column is "default", then it's default. If it's "local", then it was user-defined.

Suppose I wanted to change the "comment" property, this is how I would do it:

```
# zpool set comment="Contact admins@example.com" tank
# zpool get comment tank
NAME  PROPERTY  VALUE                          SOURCE
tank  comment   Contact admins@example.com  local
```

As you can see, the SOURCE is "local" for the "comment" property. Thus, it was user-defined. As mentioned, I don't know of a way to get string properties back to default after being set. Further, any modifiable property can be set at pool creation time by using the "-o" switch, as follows:

```
# zpool create -o ashift=12 tank raid1 sda sdb
```

Final Thoughts

The zpool properties apply to the entire pool, which means ZFS datasets will inherit that property from the pool. Some properties that you set on your ZFS dataset, which will be discussed towards the end of this series, apply to the whole pool. For example, if you enable block deduplication for a ZFS dataset, it dedupes blocks found in the entire pool, not just in your dataset. However, only blocks in that dataset will be actively deduped, while other ZFS datasets may not. Also, setting a property is not retroactive. In the case of your "autoexpand" zpool property to automatically expand the zpool size when all the drives have been replaced, if you replaced a drive before enabling the property, that drive will be considered a smaller drive, even if it physically isn't. Setting properties only applies to operations on the data moving forward, and never backward.

Despite a few of these caveats, having the ability to change some parameters of your pool to fit your needs as a GNU/Linux storage administrator gives you great control that other filesystems don't. And, as we've discovered thus far, everything can be handled with a single command "zpool", and easy-to-recall subcommands. We'll have one more post discussing a thorough examination of caveats that you will want to consider before creating your pools, then we will leave the zpool category, and work our way towards ZFS datasets, the bread and butter of ZFS as a whole. If there is anything additional about zpools you would like me to post on, let me know now, and I can squeeze it in.

Posted by Aaron Toponce on Wednesday, December 12, 2012, at 6:00 am. Filed under Debian, Linux, Ubuntu, ZFS. Follow any responses to this post with its comments RSS feed. You can post a comment or trackback from your blog. For IM, Email or Microblogs, here is the Shortlink.

{ 6 } Comments

1. Dominik Honnef | January 4, 2013 at 11:34 am | Permalink