Advanced - ZFS Pools as SMF services and iSCSI loopback mounts

THIS ARTICLE IS A WORK IN PROGRESS.

You may consult it for ideas, but I strongly suggest you do not blindly follow it yet. Some of the scripts are not even the newest of their kind, I am searching my archives for their latest versions.

As my other setup options, the one described here is an "advanced" variant which may be cumbersome to set up, has its benefits and maybe drawbacks, and is not "required" for any and all usage scenarios (just for some).

What?

This article describes two related setups, the first one may be used by itself, and the second one builds on the first:

- configuration of non-root ZFS pool import-export as an SMF service, instead of auto-import with the OS,
- and configuration of iSCSI (with COMSTAR) as a server (target) and client (initiator) for "loopback" mounting of a volume on the storage host as a container of another ZFS pool, possibly with some different storage strategies. Of course, such a pool-in-a-volume can also be mounted from some external host (one at a time), given proper permissions.

Please keep in mind that this page and example services are at this time posted from my notes and home-brew hacks. In particular, the SMF integration can be remade in a prettier manner (i.e. each pool managed by a separate instance, etc.) I publish these "as is", and they will likely need customization for your systems. Maybe I (or other contributors) would later remake these pieces into a fine generalized redistributable work of art 😎

Why?

There may be several reasons to pursue such a setup. I had a few to make it, in the first place:

- Work with a botched pool which can cause system crashes, hangs or large delays in boot. Putting it into an SMF service allows the system to be quickly available for work and administration, even though the SMF-wrapped pool may become available much later (if at all).
  As a historical anecdote, my situation had to do with removal of a large amount of data from a deduped pool on a machine which was, as I later learned, too under-spec'ed for dedup (little RAM, just 8Gb). Removal of data caused large traversals of the metadata (DDT and all) and caused hangs every few hours, but I could monitor with `zdb` that the "Deferred delete" list of blocks was getting shorter every time. And by having the pool in a service (with a timeout and a "catapult button", as seen below), I could disable the import attempt of this pool during a particular boot – if I needed to use the machine rather than have it loop to clean up that pool. Took a couple of weeks, overall...
- A secondary "virtual" pool contained in a ZFS volume in the "physical" pool. There may be a way to import it right from the `rdsk` device associated with the `zvol`, but I was told that a more generic solution is to use iSCSI and make a loopback sort of mount. However, with default setup this creates a deadlock: networking starts after the filesystems, and is needed to import the iSCSI-served pool.
  Possibly, the wrapper described below would be useful for any other (non-loopback) iSCSI initiators as well. Indeed, I used it on a remote system (with more RAM) which worked with the "virtual" pool which had `dedup` enabled for some of my experiments.
- Work of other services can depend on the pool – for example, it can house some or all of your local zones. After wrapping the pool as an SMF resource, you can enable Zones as SMF services and have them depend on the ZFS pool which stores their filesystems. Or several pools (if zone-roots with ZBEs and user-data are kept separately).

Overall, one can point out several typical variants:

- "This" system is a standalone box which contains a pool, whose import time or reliability is a potential problem. For example, even a dedicated storage box might benefit from availability of administrative SSH and other services while the user-data pool is or is not available.
- "This" system serves iSCSI targets, and in order to unmount and export a pool you have to stop those targets. Set-up of SMF dependencies allows to automate this, and any other kind of dependencies (SMFized zones and other services, for example).
- "This" system is a client of iSCSI, and attempts to import a remote pool should depend on availability of networking and readiness of the iSCSI initiator.
- "This" system has a physical pool which houses a volume which is shared over iSCSI and re-used over iSCSI by the same machine as a client, and imported as a virtual pool from that device. This enforces a specific chain of startup and shutdown, which can be best automated by SMF.

How?

Some of the steps below assume the following environment variables:

```
### Set the names for "physical" (or main for data) and "virtual" (stored in zvol on physical) pools
:; POOL_PHYS=pool
:; POOL_VIRT=dcpool
```

Again, note that at this moment the sample scripts and instructions are from a custom installation at my rig. Generalization may follow-up later. So far a curious reader would have to adapt the instructions.

Phase 1: ZFS pool as an SMF service

So, the first phase is rather trivial...

- Get the SMF wrapping script mount-main-pool and manifest mount-main-pool.xml:

```
:; wget -O /lib/svc/method/mount-main-pool \
   http://wiki.openindiana.org/download/attachments/27230301/mount-main-pool

:; mkdir -p /var/svc/manifest/network/iscsi
:; wget -O /var/svc/manifest/network/iscsi/mount-main-pool.xml \
   http://wiki.openindiana.org/download/attachments/27230301/mount-main-pool.xml
```

Don't mind the "iscsi" part of the naming – this is historical due to the second phase of this setup.

- **Edit the method script**. This file, as it is now, is tuned for my installation, and too much is hardcoded.
  Script logic: the main data pool named `pool` contains a `/pool/tmp` directory (or automountable child dataset). The method script verifies that this directory exists; if not – the pool can be imported on start (waits for listing and status to complete, and logs the results; then mounts all ZFS filesystems (note – not only from this pool), and only then does the method script complete), if yes – it can be exported (loops until success) on stop.
  In order to protect the tested directory from bogusly appearing on the root filesystem (of the `rpool`) you can use an immutable mountpoint (detailed below).
  The script includes several anti-import precautions: except for disablement of the service (as it depends on **non-existance** of the file `/etc/zfs/noimport-pool`), a delay-file `/etc/zfs/delay-pool` which can contain the timeout (in seconds) or just exist (defaults to 600 sec), and an automatic lock-file to prevent subsequent imports of pools that can not complete and hang or crash your system.
  Also note that here the import is done without a cachefile and with an alternate root (even if `/` by default). For larger pools made of many vdevs, you can speed up the imports by using an alternate `cachefile=/etc/zfs/zpool-main.cache` or something like that, just not the default one.
  You can also `touch /etc/zfs/noautomount-$POOL` in order to avoid auto-mounting of filesystem datasets (`zfs mount -a`) at the end of the routine; the pool is initially imported without automounting anything at all.
  You might want to add different options and/or logic at your taste.
  TODO: Replace hardcoding with config-file and/or SMF attribute modifiable configuration.
- **Revise the manifest file**. It currently sets a dependency on `filesystem/local`; you might want something else (such as `svc:/network/ssh:default`) so that you can have a while to disable the pool-importing service. If revising dependencies, make sure to avoid loops (SMF commands should help here).
  Also the service depends on the *absence of lock-files* `/etc/zfs/.autolock.pool` (created and removed by the method script around import attempts) and `/etc/zfs/noimport-pool` (maintained by the user to optionally disable auto-import); the `pool` part in these filenames (or rather the complete filenames, as synthesized by default) should match what is defined for the service in the method script.
  It also defines `smb/server` and `zones` as dependent services so that these resources hosted on the data `pool` are only started when it is mounted; you might also want to add `nfs/server`, or set them to `optional_all` type of dependency, if your `rpool` also hosts some zones and /or files and can do so without a present data pool.
- Install the SMF wrapping scripts:

  ```
  :; svccfg import /var/svc/manifest/network/iscsi/mount-main-pool.xml
  ```

  This creates the (disabled) service for main-pool importing, which calls the script above as the method script.
- Remove the pool in question from auto-import database by exporting it (NOTE: this unshares and unmounts all its filesystems in the process, will block or fail on any active users, over-mounted filesystem nodes, used volumes, etc.):

  ```
  :; POOL=$POOL_PHYS
  :; zpool export $POOL
  ```

  As a result, this pool should no longer be cached in `/etc/zfs/zpool.cache` for faster and automated imports at OS startup.
- Protect the mountpoint from method script's test failures:

  ```
  :; df -k /$POOL
  :; ls -la /$POOL
  ### Make sure that the pool is exported and its mountpoint directory does not exist or is empty

  :; mkdir /$POOL
  :; /bin/chmod S+ci /$POOL
  ```

  The immutable mountpoint can not be written even by `root`, such as when an untimely `zfs mount` would try to create subdirectories without mounting the pool's root dataset first and break our setup.
- Enable the service, which should mount your pool, you can monitor the progress and ultimately the pool status in the service log:

```
### Not done before, so you have time to revise the steps instead of blind copy-pasting ;)
:; chmod +x /lib/svc/method/mount-main-pool

### Temp-enable while we are testing
:; svcadm enable -t mount-main-pool
:; tail -f /var/svc/log/*mount-main-pool*log

### If all is ok, you may want to enable the service to start at boot... or maybe not.

:; mkdir /$POOL/tmp
:; svcadm enable mount-main-pool
```

At this point, your data pool is imported not blindly by the OS, but by your service. Which you can disable in case of problems (at the moment this may require to boot into a livecd to create the block-file `/etc/zfs/noimport-pool` which would cancel the service startup, if for some reason an automatic creation and clearing of a block-file around the start/stop calls does not help).

Perhaps more importantly, you now have the main pool wrapped as an SMF resource on which other services can depend (or not depend) for orderly startup. If this pool takes very long to import and/or can fail in the process, it does not delay the startup of other services (like `ssh`), and you can monitor the state of the import as an SMF status with `svcs` or numerous remote-management tools.

Phase 2: iSCSI target (server)

Here we set up the zvol and share over iSCSI which would store "virtual" ZFS pool, named below `dcpool` for historical reasons (it was **d**eduped inside and **c**ompressed outside on my test rig, so I hoped to compress only the unique data written).

TODO: find my notes on setup of the server – unfortunately, the HomeNAS itself is not currently available to look at... but there was (IIRC) not much different from usual COMSTAR iSCSI (with `stmf`). Enable services, create a backing store for a LU implemented as a zvol, allow localhost and/or remote hosts to access it.

One possible caveat is that the iSCSI server services should be made dependent on the `main-pool-import` service created above (assuming that it holds the `zvol`). If there are several physical pools, and others serve iSCSI too – a separate instance (or full service made as a replica) of `iscsi/target` may be in order, to wrap just the creation/teardown and sharing/unsharing of target(s) on the SMFized pool – see iscsi-lun-dcpool for an example (NOTE: hardcoded values would need adaptation for your systems).

Phase 3: iSCSI initiator (client)

This phase involves wrapping of the iSCSI-mounted pool as an SMF service. Indeed, some readers who simply use remote iSCSI pools, might start reading the article here

First, there is the initiator part: the networking client. There is really no magic here, this service was needed just for peace of mind about not conflicting with system services (i.e. over OS upgrades) while I create specific dependency setups. It uses the same system logic as `iscsi/initiator` for actual work. Possibly, just one needs to be enabled at a time.

- Get the method script iscsi-initiator-dcpool and manifest initiator-dcpool.xml:

  ```
  :; wget -O /lib/svc/method/iscsi-initiator-dcpool \
     http://wiki.openindiana.org/download/attachments/27230301/iscsi-initiator-dcpool
  :; wget -O /var/svc/manifest/network/iscsi/initiator-dcpool.xml \
     http://wiki.openindiana.org/download/attachments/27230301/initiator-dcpool.xml
  ```

- **Revise the files**. The script contains a callout to the system's standard `/lib/svc/method/iscsi-initiator` wrapped with 10-second sleeps (after start and before stop).
  The manifest declares a dependency on networking (at least `loopback`) and on `multi-user-server` milestone,
  On a system which serves the volume (real loopback-import) you'd also add a dependency on the iSCSI target service (or its instance or replica dedicated to serving this particular volume).
- Install the service:

  ```
  :; svccfg import /var/svc/manifest/network/iscsi/initiator-dcpool.xml
  ```

  This creates a (disabled) instance of `network/iscsi/initiator-dcpool:default`.
- Enable the service:

  ```
  :; svcadm enable initiator-dcpool
  ```

  This should allow the system to use iSCSI and find the defined (elsewhere) targets.

Second, prepare the mountpoint (also protected from modifications with immutability):

```
:; POOL=$POOL_VIRT

:; df -k /$POOL
:; ls -la /$POOL
### Make sure that the pool is exported and its mountpoint directory does not exist or is empty

:; mkdir /$POOL
:; /bin/chmod S+ci /$POOL
```

Third, set up the import of the pool over iSCSI (assuming that the COMSTAR initiator has been set up to query the needed target servers, and now `zpool` knows where to find iSCSI-backed vdevs):

- Get the method script iscsi-mount-dcpool and manifest mount-dcpool.xml:

  ```
  :; wget -O /lib/svc/method/iscsi-mount-dcpool \
     http://wiki.openindiana.org/download/attachments/27230301/iscsi-mount-dcpool
  :; wget -O /var/svc/manifest/network/iscsi/mount-dcpool.xml \
     http://wiki.openindiana.org/download/attachments/27230301/mount-dcpool.xml
  ```

- **Revise the files**. The script is currently hardcoded to import or export a `dcpool`, subject to absence of `/etc/zfs/noimport-dcpool` block-file, and determines presence of an already-imported pool as presence of the `/dcpool/export` directory. If the file `/etc/zfs/delay.dcpool` exists and contains a number, the startup of the service is delayed by this number of seconds; if the file exists and is empty (or not a number), the delay is 600 seconds (10 minutes). The import bypasses the default cachefile, but otherwise is not tweaked. The export loops until successful. The manifest declares dependencies on `iscsi/initiator-dcpool` and on networking (`loopback` here, you may want `physical` for remote mounts), and a reverse-dependency on the block-file.
- Install the service:

  ```
  :; svccfg import /var/svc/manifest/network/iscsi/mount-dcpool.xml
  ```

  This creates a (disabled) instance of `network/iscsi/mount-dcpool:default`.
- Enable the service:

  ```
  :; svcadm enable mount-dcpool
  :; tail -f /var/svc/log/*mount-dcpool*log
  ```

Finally, I also had a watchdog service to actively monitor the viability of the "virtual" pool, as things tended to lock up once in a while, either due to internetworking faults or the experimental server problems. But that was too much of an in-house hack to publish at the moment (and relied on some currently proprietary status-testing methods).

HTH,
//Jim Klimov