StorPool User Guide

StorPool version 16.01

Document version 2017-06-20

StorPool Overview

StorPool is distributed storage software. It pools the attached storage (hard disks or SSDs) of standard servers to create a single pool of shared storage. The StorPool software is installed on each server in the cluster. It combines the performance and capacity of all drives attached to the servers into one global namespace.

StorPool provides standard block devices. You can create one or more volumes through its sophisticated volume manager. StorPool is compatible with ext4 and XFS file systems and with any system designed to work with a block device, e.g. databases and cluster file systems (like OCFS and GFS). StorPool can also be used with no file system, for example when using volumes to store VM images directly or as LVM physical volumes.

Redundancy is provided by multiple copies (replicas) of the data written synchronously across the cluster. Users may set the number of replication copies. We recommend 3 copies as a standard and 2 copies for data that is less critical. The replication level directly correlates with the number of servers that may be down without interruption in the service. For replication 3 the number of the servers that may be down simultaneously without losing access to the data is 2.

StorPool protects data and guarantees its integrity by a 64-bit checksum and version for each sector maintained by StorPool. StorPool provides a very high degree of flexibility in volume management. Unlike other storage technologies, such as RAID or ZFS, StorPool does not rely on device mirroring (pairing drives for redundancy). So every disk that is added to a StorPool cluster adds capacity to the cluster, not just for new data but also for existing data. Provided that there are sufficient copies of the data, drives can be added or taken away with no impact to the storage service. Unlike rigid systems like RAID, StorPool does not impose any strict hierarchical storage structure dictated by the underlying disks. StorPool simply creates a single pool of storage that utilises the full capacity and performance of a set of commodity drives.

Architecture

StorPool works on a cluster of servers in a distributed shared-nothing architecture. All functions are performed by all servers on an equal peer basis. It works on standard off-the-shelf servers running GNU/Linux.

Each storage node is responsible for data stored on its local hard drives. Storage nodes collaborate to provide the storage service. StorPool provides a shared storage pool combining all the available storage capacity. It uses synchronous replication across servers. The StorPool client communicates in parallel with all StorPool servers.

The software consists of two parts - a storage server and a storage client - that are installed on each physical server (host, node). Each host can be a storage server, a storage client, or both. To storage clients StorPool volumes appear as block devices under /dev/storpool/* and behave as normal disk devices. The data on the volumes can be read and written by all clients simultaneously; its consistency is guaranteed through a synchronous replication protocol. Volumes may be used by clients as they would use a local hard drive or disk array.

Feature Highlights

Scale-out, not Scale-Up

The StorPool solution is fundamentally about scaling out (scaling by adding more drives or nodes) rather than scaling up (adding capacity by replacing a storage box with larger storage box). This means StorPool can scale independently by IOPS, storage space and bandwidth. There is no bottleneck or single point of failure. StorPool can grow without interruption and in small steps - one hard drive, one server and one network interface at a time.

High Performance

StorPool combines the IOPS performance of all drives in the cluster and optimizes drive access patterns to provide low latency and handling of storage traffic bursts. The load is distributed equally between all servers through striping and sharding.

High Availability and Reliability

StorPool uses a replication mechanism that slices and stores copies of the data on different servers. For primary, high performance storage this solution has many advantages compared to RAID systems and provides considerably higher levels of reliability and availability. In case of a drive, server, or other component failure, StorPool uses another copy of the data located on another server (or rack) and none of your data is lost or even temporarily unavailable.

Commodity Hardware

StorPool supports drives and servers in a vendor-agnostic manner, allowing you to avoid vendor lock-in. This allows the use of commodity hardware, while preserving reliability and performance requirements. Moreover, unlike RAID, StorPool is drive agnostic - you can mix drives of various types, make, speed or size in a StorPool cluster.

Shared Block Device

StorPool provides shared block devices with semantics identical to a shared iSCSI or FC disk array.

Co-existence with hypervisor software

StorPool can utilize repurposed existing servers and can co-exist with hypervisor software on the same server. This means that there is no dedicated hardware for storage, and growing an IaaS cloud solution is achieved by simply adding more servers to the cluster.

Compatibility

StorPool is compatible with 64-bit Intel and AMD based servers. We support all Linux-based hypervisors and hypervisor management software. Any Linux software designed to work with a shared storage solution such as an iSCSI or FC disk array will work with StorPool. StorPool guarantees the functionality and availability of the storage solution at the Linux block device interface.

CLI interface and API

StorPool provides an easy yet powerful command-line interface (CLI) tool for administration of the data storage solution. It is simple and user-friendly - making configuration changes, provisioning and monitoring is fast and efficient. StorPool also provides a RESTful JSON API, exposing all the available functionality, so you can integrate it with any existing management system.

Reliable Support

StorPool comes with reliable dedicated support: remote installation and initial configuration by StorPool's specialists; 24x7 support; software updates.

Hardware Requirements

All distributed storage systems are highly dependent on the underlying hardware. There are some aspects that will help achieve maximum performance with StorPool and are best considered in advance. Each node in the cluster can be used as server, client or both; depending on the role, hardware requirements vary.

Minimum StorPool cluster

- 3 industry-standard x86 servers;
- any x86-64 CPU with 2 threads or more;
- 32 GB ECC RAM per node (8+ GB used by StorPool);
- any hard drive controller in JBOD mode;
- 3x SATA2 hard drives;
- dedicated 10GE LAN;

Recommended StorPool cluster

- 5 industry-standard x86 servers;
- IPMI, iLO/LOM/DRAC desirable;
- Intel Nehalem generation (or newer) Xeon processor(s);
- 64GB ECC RAM or more in every node;
- Infiniband QDR/FDR;

- any hard drive controller in JBOD mode;
- 5+ hard drives per storage node;
- 2+ SATA3 SSD per storage node;

How StorPool relies on hardware

CPU

When the system load is increased, CPUs are saturated with system interrupts. To avoid the negative effects of this, StorPool's server and client processes can be given one or more dedicated CPU cores. This significantly improves overall the performance and the performance consistency.

RAM

ECC memory can detect and correct the most common kinds of in-memory data corruption thus maintains a memory system immune to single-bit errors. Using ECC memory is an essential requirement for improving the reliability of the node. In fact, StorPool is not designed to work with non-ECC memory.

Storage (HDDs / SSDs)

StorPool ensures the best drive utilization. Replication and data integrity are core functionality, so RAID controllers are not required and all storage devices might be connected as JBOD.

Network

StorPool is a distributed system which means that the network is an essential part of it. Designed for efficiency, StorPool combines data transfer from other nodes in the cluster. This greatly improves the data throughput, compared with access to local devices, even if they are SSD.

Software Compatibility

Operating Systems

- Linux
- Windows and VMWare in our roadmap

File Systems

Developed and optimized for Linux, StorPool is best tested on CentOS and Ubuntu. Compatible with ext4 and XFS file systems and with any system designed to work with a block device, e.g. databases and cluster file systems (like GFS2 or OCFS2). StorPool can also be used with no file system, for example when using volumes to store VM images directly. StorPool is compatible with other technologies from the Linux storage stack, such as LVM, dm-cache/bcache, and LIO.

Hypervisors & Cloud Management/Orchestration

- KVM
- LXC/Containers
- OpenStack
- OpenNebula
- OnApp
- CloudStack
- any other technology compatible with the Linux storage stack.

Installation and Upgrade

Currently the installation and upgrade procedures are performed by StorPool support team.

Configuration Guide

Minimal Node Configuration

To configure nodes Storpool uses a configuration file, which can be found at `/etc/storpool.conf`. Host specific configuration can be placed in `/etc/storpool.conf.d/` folder. The minimum working configuration must specify the network interface, number of expected nodes, authentication tokens and unique ID of the node like in the following example:

```
#-
# Copyright (c) 2013, 2014  StorPool.
# All rights reserved.
#


# Interface for storpool communication
#
# Default: eth0
SP_IFACE=eth1


# expected nodes for beacon operation
#
# !!! Must be specified !!!
#
SP_EXPECTED_NODES=3


# API authentication token
#
# 64bit random value
# generate for example with: 'od -vAn -N8 -tu8 /dev/random'
SP_AUTH_TOKEN=4306865639163977196


##########################


[spnode1.example.com]
SP_OURID = 1
```

Full Configuration Options List

The following is a complete list of the configuration options with short explanation for each of them.

Cluster name

Required for the pro-active monitoring performed by StorPool support team. Usually in the form `<Company-Name>-<City-or-nearest-airport>`:

```
SP_CLUSTER_NAME=StorPoolLab-Sofia
```

Cluster ID

The Cluster ID is computed from the StorPool Support ID and consists of two parts - location and cluster separated by a dot (`"."`). In this release each location consist of a single cluster. This will be extended to multiple clusters at a location in future releases:

```
SP_CLUSTER_ID=nzkr.b
```

Non-voting beacon node

For client only nodes, the `storpool_server` service will refuse to start on a node with `SP_NODE_NON_VOTING`. Default is 0:

```
SP_NODE_NON_VOTING=1
```

Attention

It is strongly recommended to configure `SP_NODE_NON_VOTING` at the per-host configuration sections in `storpool.conf` (see Per host configuration for more)

Communication interface for StorPool cluster

Recommended to have two or more dedicated network interfaces for communication between the nodes (maximum four):

```
SP_IFACE=eth1,eth2
```

MTU of network interfaces used by StorPool

If not specified, the default MTU is 9000. This greatly improves the performance, to set it to a different value for the different interfaces use:

```
SP_IFACE=eth1=1500,eth2=9000
```

Address for the API management (`storpool_mgmt`)

Used by the CLI. Multiple clients can simultaneously send requests to the API. The management service is usually started on one or more nodes in the cluster at a time. By default it is bound on localhost:

```
SP_API_HTTP_HOST=127.0.0.1
```

For cluster wide access and automatic failover between the nodes, multiple nodes might have it started. The specified IP address is brought up only in one of the nodes in the cluster at a time - the so called `active` API service. To configure an interface (`SP_API_IFACE`) for bringing up or down the API address (`SP_API_HTTP_HOST`) when migrating the API service use:

```
SP_API_HTTP_HOST=10.10.10.240
SP_API_IFACE=eth1
```

Note

The script that adds or deletes the `SP_API_HTTP_HOST` address is located at `/usr/lib/storpool/api-ip` and could be easily modified for other use cases (e.g. configure routing, firewalls, etc.).

Port for the API management (`storpool_mgmt`)

Port for the API management service, the default is:

```
SP_API_HTTP_PORT=81
```

Address for the bridge service (`storpool_bridge`)

Required for the local bridge service, this is the address where the bridge binds to:

```
SP_BRIDGE_HOST=180.220.200.8
```

Working directory

Used for fifos, sockets, core files, etc., default:

```
SP_WORKDIR=/var/run/storpool
```

Hint

On nodes with `/var/run` in RAM and limited memory, better use `/var/run/storpool/run`.

Report directory

Location for collecting automated bug reports and shared memory dumps:

```
SP_REPORTDIR=/var/spool/storpool
```

Restart automatically in case of crash

Restart the service in case of crash if there are less than 3 crashes during this interval in seconds. If this value is 0 service will not restart at all and will have to be started manually, default is:

```
SP_RESTART_ON_CRASH=1800
```

Expected nodes

Minimum expected nodes for beacon operation, usually equal to the number of nodes with `storpool_server` service:

```
SP_EXPECTED_NODES=3
```

Local user for report collection

User to change the ownership of reports and crashes. Unset by default:

```
SP_CRASH_USER=
```

Note

In case of no configuration during installation, this user will be set by default to `storpool`.

Remote user for report collection

Remote user for sending reports to StorPool. Usually lowercase `SP_CLUSTER_NAME`. Used by rsync in the `storpool_repsync` utility:

```
SP_CRASH_REMOTE_USER=storpoollab-sofia
```

Remote host address for sending reports

The default remote is `reports.storpool.com`, which could be altered in case a jumphost or a custom collection node is used:

```
SP_CRASH_REMOTE_ADDRESS=reports.storpool.com
```

Port on the remote host for sending reports

The default port is `2266`, might be altered in case a jumphost or a custom collection node is used:

```
SP_CRASH_REMOTE_PORT=2266
```

Group owner for the StorPool devices

The system group to use for the `/dev/storpool` directory and the `/dev/sp-*` raw disk devices:

```
SP_DISK_GROUP=disk
```

Permissions for the StorPool devices

The access mode to set on the `/dev/sp-*` raw disk devices:

```
SP_DISK_MODE=0660
```

Cgroup setup

Enable the usage of cgroups, default is on. Each StorPool process requires a specification of the cgroups it should be started into, there is a default configuration for each service and example configuration in `/usr/share/doc/storpool/examples` directory on a node where StorPool is installed. One or more processes may be placed in the same cgroup or each one may be in a cgroup of its own, as appropriate:

```
SP_USE_CGROUPS=1
```

It is mandatory to specify a `SP_RDMA_CGROUPS` setting for the kernel threads started by the StorPool modules:

```
SP_RDMA_CGROUPS=-g memory:/storpool.slice -g cpuset:/storpool.slice/rdma
```

Set cgroups for the `storpool_block` service. Usually runs on the same core as the RDMA with 2 isolated cores. Isolating the `storpool_block` on a dedicated core further improves the performance:

```
SP_BLOCK_CGROUPS=-g memory:/storpool.slice -g cpuset:/storpool.slice/block
```

Set cgroups for the `storpool_bridge` service. Depending on the load runs either on a dedicated core or on the one where the `storpool_mgmt` service is running:

```
SP_BRIDGE_CGROUPS=-g memory:/storpool.slice -g cpuset:storpool.slice/mgmt
```

Set cgroups for the `storpool_server` service. Isolating the `storpool_server` on a dedicated core improves the performance of the storage operations significantly, with multiple servers the defaults are:

```
SP_SERVER_CGROUPS=-g memory:/storpool.slice -g cpuset:/storpool.slice/server
SP_SERVER1_CGROUPS=-g memory:/storpool.slice -g cpuset:/storpool.slice/server_1
SP_SERVER2_CGROUPS=-g memory:/storpool.slice -g cpuset:/storpool.slice/server_2
SP_SERVER3_CGROUPS=-g memory:/storpool.slice -g cpuset:/storpool.slice/server_3
```

Set cgroups for the `storpool_beacon` service, usually on the core or the thread where the server is living:

```
SP_BEACON_CGROUPS=-g memory:/storpool.slice -g cpuset:/storpool.slice/beacon
```

Set cgroups for the `storpool_mgmt` service:

```
SP_MGMT_CGROUPS=-g memory:/storpool.slice -g cpuset:/storpool.slice/mgmt
```

Network interrupts affinity

Set affinity for the dedicated network interfaces IRQ interrupts. The recommended configuration is with the same CPU core as for the network related processes. The example below is with regular name for the first interface and “predictable network interface name” from `/proc/interrupts`:

```
SP_CPUS_NIC_IRQ=0
SP_CPUS_NIC_REGEX=(eth2|ens1f0)
```

Storage controller interrupts affinity

Set affinity for the HBA adapters IRQ interrupts. The recommended configuration is with the same CPU core as for the `storpool_server` process. The example is for server with two integrated to the MB storage controllers and one HBA all interrupts pinned to CPU core `1`:

```
SP_CPUS_HBA_IRQ=1
SP_CPUS_HBA_REGEX=(ahci|isci|mvsas)
```

Cache size

Each `storpool_server` process allocates this amount of RAM (in MB) for caching. The size of the cache depends on the number of storage devices on each `storpool_server` instance:

```
SP_CACHE_SIZE=4096
```

Note

A node with three `storpool_server` processes running will use 4096*3 = 12GB cache total.

Attention

Changing the size of the cache file requires first stopping the server instance, then removing its cache file from `/dev/shm/storpool.cache` and then starting it back again.

Set the internal write-back caching to on:

```
SP_WRITE_BACK_CACHE_ENABLED=1
```

Attention

UPS is mandatory with WBC, clean server shutdown is required before the UPS batteries are depleted.

API authentication token

This value must be unique integer for each cluster:

```
SP_AUTH_TOKEN=0123456789
```

Hint

Generate with: `od -vAn -N8 -tu8 /dev/random`

NVMe SSD drives

To instruct the `storpool_server` which is the PCIe ID of the NVMe SSD configure the following:

```
SP_NVME_PCI_ID=0000:04:00.0
```

Per host configuration

Specific details per host. The value in the square brackets should be the name of the host as returned by the `hostname` command. The ID of the node must be unique throughout the cluster:

```
[spnode1.example.com]
SP_OURID=1
```

The highest ID in this release is `62` or up to this number of nodes in a single cluster.

Specific configuration details might be added for each host individually, e.g.:

```
[spnode1.example.com]
SP_OURID=1
SP_IFACE=eth3,eth4
SP_REPORTDIR=/mnt/reportdir
```

Prepare Storage Devices

All hard drives, SSD, or NVME drives that will be used by StorPool must have one properly aligned partition and must have an assigned ID. The ID should be a number between 1 and 4000 and must be unique within the StorPool cluster. An example command for creating a partition on the whole drive with the proper alignment:

```
parted -s --align optimal /dev/{block_device} mklabel gpt -- mkpart primary 2M 100%
```

On a brand new cluster installation it is necessary to have one drive formatted with the “init” (`-I`) flag of `storpool_initdisk`. This device is necessary only for the first start and therefore it is best to pick the first drive in the cluster.

Initializing the first drive on the first server node with the init flag:

```
# storpool_initdisk -I {diskId} /dev/{partition}
```

Initializing an SSD or NVME SSD device with the SSD flag set:

```
# storpool_initdisk -s {diskId} /dev/{partition}
```

Initializing an HDD drive:

```
# storpool_initdisk {diskId} /dev/{partition}
```

List all initialized devices:

```
# storpool_initdisk --list
```

Example output:

```
/dev/nvme0n1, diskId 1101, version 10007, server instance 0, cluster nzkr.b, SSD
/dev/sdu1, diskId 1111, version 10007, server instance 0, cluster nzkr.b, WBC
/dev/sdd1, diskId 1112, version 10007, server instance 0, cluster nzkr.b, WBC
/dev/sdv1, diskId 1102, version 10007, server instance 0, cluster nzkr.b, WBC
/dev/sdw1, diskId 1103, version 10007, server instance 0, cluster nzkr.b, WBC
```

Other avaialble options:

- `-i` - Specify server instance, used when more than one `storpool_server` instances are running on the same node
- `-r` - Used to return an ejected disk back to the cluster or change some of the parameters
- `-F` - Forget this disk and mark it as ejected (succeeds only without a running `storpool_server` instance that has the drive opened)
- `-s` - set SSD flag - on new initialize only, not revertible with `-r`).
- `--wbc (y|n)` - Used for HDDs when the internal write-back caching is enabled, implies `SP_WRITE_BACK_CACHE_ENABLED` to have an effect. Turned off by default.
- `--no-trim (y|n)` - Used to forcefully disable TRIM support for an SSD device. Useful when the drive is behind a RAID controller without support for TRIM pass-through.
- `--force` - Used when re-initializing an already initialized StorPool drive. Use with caution.

Verify the Installation

A StorPool installation provides the following daemons that take care of different functionality on each participant node in the cluster.

storpool_beacon

The beacon must be the first started process on all nodes in cluster. It informs all members about the availability of the node on which it is installed. If the number of the visible nodes changes, every `storpool_beacon` service checks that its node still participates is the quorum, i.e. it can communicate with more than half of the expected nodes, including itself (see `SP_EXPECTED_NODES` in the Full Configuration Options List section). If the `storpool_beacon` service has been started successfully, it will send to the system log (`/var/log/messages`, `/var/log/syslog`, or similar) messages such as the following for every node that comes up in the StorPool cluster:

```
[snip]
Jan 21 16:22:18 s01 storpool_beacon[18839]: [info] incVotes(1) from 0 to 1, voteOwner 1
Jan 21 16:23:10 s01 storpool_beacon[18839]: [info] peer 2, beaconStatus UP bootupTime 1390314187662389
Jan 21 16:23:10 s01 storpool_beacon[18839]: [info] incVotes(1) from 1 to 2, voteOwner 2
Jan 21 16:23:10 s01 storpool_beacon[18839]: [info] peer up 1
[snip]
```

storpool_server

The `storpool_server` service must be started on each node that provides its hard drives and SSDs to the cluster. If the service has started successfully, all the hard drives intended to be used as StorPool disks should be listed in the system log, e.g.:

```
Dec 14 09:54:19 s11 storpool_server[13658]: [info] /dev/sdl1: adding as data disk 1101 (ssd)
Dec 14 09:54:19 s11 storpool_server[13658]: [info] /dev/sdb1: adding as data disk 1111
Dec 14 09:54:20 s11 storpool_server[13658]: [info] /dev/sda1: adding as data disk 1114
Dec 14 09:54:20 s11 storpool_server[13658]: [info] /dev/sdk1: adding as data disk 1102 (ssd)
Dec 14 09:54:20 s11 storpool_server[13658]: [info] /dev/sdj1: adding as data disk 1113
Dec 14 09:54:22 s11 storpool_server[13658]: [info] /dev/sdi1: adding as data disk 1112
```

On a dedicated or node with a larger amount of spare resources, more than one `storpool_server` instance could be started (up to four instances).

storpool_block

The `storpool_block` service provides the client (initiator) functionality. StorPool volumes can be attached only to the nodes where this service is running. When attached to a node, a volume can be used and manipulated as a regular block device via the `/dev/stopool/{volume_name}` symlink:

```
# lsblk /dev/storpool/test
NAME MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sp-2 251:2    0  100G  0 disk
```

storpool_mgmt

The `storpool_mgmt` service should be started on the management node. It receives requests from user space tools (CLI or API), executes them in the StorPool cluster and returns the results back to the sender. Each node can be used as an API management server, with only one node active at a time. An automatic failover mechanism is available: when the node with the active `storpool_mgmt` service fails, the `SP_API_HTTP_HOST` IP address is configured on the next node with the lowest `SP_OURID` with a running `storpool_mgmt` service.

storpool_bridge

The `storpool_bridge` service is started on one of the nodes in the cluster. This service synchronizes snapshots for the backup and disaster recovery use cases between this and one or more StorPool clusters in different locations.

CLI Tutorial

StorPool provides an easy yet powerful Command Line Interface (CLI) for administrating the data storage cluster. It has an integrated help system that provides useful information on every step. There are various ways to execute commands in the CLI, depending on the style and needs of the administrator. The StorPool CLI gets its configuration from `/etc/storpool.conf` file and command line options.

Type regular shell command with parameters:

```
# storpool service list
```

Use interactive StorPool shell:

```
# storpool
StorPool> service list
```

Pipe command output to StorPool CLI:

```
# echo "service list" | storpool
```

Redirect the standard input from a predefined file with commands:

```
# storpool < input_file
```

Display the available command line options:

```
# storpool --help
```

Error message with possible options will be displayed if the shell command is incomplete or wrong:

```
# storpool attach
Error: incomplete command! Expected:
    list - list the current attachments
    timeout - seconds to wait for the client to appear
    volume - specify a volume to attach
    here - attach here
    noWait - do not wait for the client
    snapshot - specify a snapshot to attach
    mode - specify the read/write mode
    client - specify a client to attach the volume to

# storpool attach volume
Error: incomplete command! Expected:
  volume - the volume to attach
```

Interactive shell help can be invoked by pressing the question mark key (`?`):

```
# storpool
StorPool> attach
  client - specify a client to attach the volume to {M}
  here - attach here {M}
  list - list the current attachments
  mode - specify the read/write mode {M}
  noWait - do not wait for the client {M}
  snapshot - specify a snapshot to attach {M}
  timeout - seconds to wait for the client to appear {M}
  volume - specify a volume to attach {M}
```

Shell autocomplete, invoked by double-pressing the `Tab` key, will show available options for current step:

```
StorPool> attach <tab> <tab>
client    here      list      mode      noWait    snapshot  timeout   volume
```

StorPool shell can detect incomplete lines and suggest options:

```
# storpool
StorPool> attach <enter>
.................^
Error: incomplete command! Expected:
    volume - specify a volume to attach
    client - specify a client to attach the volume to
    list - list the current attachments
    here - attach here
    mode - specify the read/write mode
    snapshot - specify a snapshot to attach
    timeout - seconds to wait for the client to appear
    noWait - do not wait for the client
```

To exit the shell use `quit` or `exit` commands or directly use the `Ctrl+C` or `Ctrl+D` keyboard shortcuts of your terminal.

Location

The `location` submenu is used for configuring other StorPool clusters for disaster recovery and backup purposes. The location ID is the first part (left of the `.`) in the `SP_CLUSTER_ID` configured in the remote cluster, for example to add a remote location with `SP_CLUSTER_ID=nzkr.b` use:

```
# storpool location add nzkr StorPoolLab-Sofia
OK
```

To list the configured remote locations use:

```
# storpool location list
----------------------------
| id   | name              |
----------------------------
| nzkr | StorPoolLab-Sofia |
----------------------------
```

Cluster

The `cluster` submenu is used for configuring a cluster for an already configured remote `location`. The cluster ID is the second part (right from the `.`) in the `SP_CLUSTER_ID` configured in the remote cluster, for example to add the cluster `b` for the remote location `nzkr` use:

```
# storpool cluster add StorPoolLab-Sofia b
OK
```

To list the configured remote clusters use:

```
# storpool cluster list
-------------------------
| id | location          |
-------------------------
| b  | StorPoolLab-Sofia |
-------------------------
```

Remote Bridge

The `remoteBridge` submenu is used to register or deregister a remote bridge for a configured remote location.

To register a remote bridge use `storpool remoteBridge register <location-name> <IP address> <public-key>`, for example:

```
# storpool remoteBridge register StorPoolLab-Sofia 10.1.100.10 ju9jtefeb8idz.ngmrsntnzhsei.grefq7kzmj7zo.nno515u6ftna6
OK
```

Will register the `StorPoolLab-Sofia` location with an IP address of `10.1.100.10` and the above public key.

In case of a change in the IP address or the public key of a remote location the remote bridge could be de-registered and then registered again with the required parameters, e.g.:

```
# storpool remoteBridge de-register 10.1.100.10
OK
# storpool remoteBridge register StorPoolLab-Sofia 78.90.13.150 8nbr9q162tjh.ahb6ueg16kk2.mb7y2zj2hn1ru.5km8qut54x7z
OK
```

To enable deferred deletion on unexport from the remote site the `minimumDeleteDelay` flag should also be set, the format of the command is `storpool remoteBridge register <location-name> <IP address> <public-key> <minimumDeleteDelay>`, where the last parameter is a time period provided as `X[smhd]` - X is an integer and `s`, `m`, `h`, and `d` are seconds, minutes, hours and days accordingly.

For example if we register the remote bridge for `StorPoolLab-Sofia` location with a `minimumDeleteDelay` of one day the register would look like this:

```
# storpool remoteBridge register StorPoolLab-Sofia 78.90.13.150 8nbr9q162tjh.ahb6ueg16kk2.mb7y2zj2hn1ru.5km8qut54x7z 1d
OK
```

After this operation all snapshots sent from the remote cluster could also be unexported with the `deleteAfter` parameter set (check the Remote snapshots section). Any `deleteAfter` parameters lower than the `minimumDeleteDelay` will be overridden by the latter.

To list all registered remote bridges use:

```
# storpool remoteBridge list
-------------------------------------------------------------------------------------------------------------------------
| ip          | location                 | minimumDeleteDelay | publicKey                                              |
-------------------------------------------------------------------------------------------------------------------------
| 10.1.100.10 | StorPoolLab-Sofia        |                    | 8nbr9q162tjh.ahb6ueg16kk2.mb7y2zj2hn1ru.5km8qut54x7z    |
-------------------------------------------------------------------------------------------------------------------------
```

Check the Multi Site section from this user guide for more on deferred delete.

Network

To list basic details about the cluster network use:

```
# storpool net list
--------------------------------------------------------------------------------------------------------------------------------------
----
| nodeId | flags | MAC - net 1       | MAC - net 2       | MAC - net 3       | MAC - net 4       | rdma port 1         | rdma port 2
|
--------------------------------------------------------------------------------------------------------------------------------------
----
|     11 | uU +  | F4:52:14:76:9C:B0 | F4:52:14:76:9C:B1 | -                 | -                 | -                   | -
|
|     12 | uU +  | 00:02:C9:3C:E3:80 | 00:02:C9:3C:E3:81 | -                 | -                 | -                   | -
|
|     13 | uU +  | F4:52:14:76:9B:B0 | F4:52:14:76:9B:B1 | -                 | -                 | -                   | -
|
|     14 | uU +  | F4:52:14:76:9C:E0 | F4:52:14:76:9C:E1 | -                 | -                 | -                   | -
|
--------------------------------------------------------------------------------------------------------------------------------------
----
Flags:
u - packets recently received from this node
d - no packets recently received from this node

U - this node has enough votes to be in the quorum
D - this node does not have enough votes to be in the quorum
+ - this node considers itself in the quorum

N - a non-voting node
```

With InfiniBand or RDMA over Converged Ethernet (RoCE) networks this output is a bit different, having populated only the `rdma` ports, this is an example output with redundand IB network:

```
# storpool net list
--------------------------------------------------------------------------------------------------------------------------------------------
----
| nodeId | flags | MAC - net 1         | MAC - net 2         | MAC - net 3         | MAC - net 4         | rdma port 1          | rdma port 2
|
--------------------------------------------------------------------------------------------------------------------------------------------
----
|     11 |   uUN+ | -                   | -                   | -                   | -                   | O 0x2c9030030ca51    | O
0x2c9030030ca52   |
|     12 |   uUN+ | -                   | -                   | -                   | -                   | O 0x2c9030030aef1    | O
0x2c9030030aef2   |
|     13 |   uUN+ | -                   | -                   | -                   | -                   | O 0x2c9030030abe1    | O
0x2c9030030abe2   |
|     14 |   uUN+ | -                   | -                   | -                   | -                   | O 0x2c9030030fc21    | O
0x2c9030030fc22   |
--------------------------------------------------------------------------------------------------------------------------------------------
----
[snip]
```

Note

The flags for the RDMA ports are `I` - “Idle”, `C` - “GidReceived”, `C` - “Connecting”, `O` - “Connected”, `E` - “pendingError” or “Error”.

Server

To list the nodes that are configured as StorPool servers and their `storpool_server` instances use:

```
# storpool server list
cluster running, mgmt on node 11
    server  11.0 running on node 11
    server  12.0 running on node 12
    server  13.0 running on node 13
    server  14.0 running on node 14
    server  11.1 running on node 11
    server  12.1 running on node 12
    server  13.1 running on node 13
    server  14.1 running on node 14
```

To get more information about which storage devices are provided by a particular server use the storpool server <ID>:

```
# storpool server 11 disk list
disk  |   server  |     size    |     used    |   est.free  |      %  |  free entries  |  on-disk size  |  allocated objects |  errors
1103  |     11.0  |     447 GB  |     3.1 GB  |     424 GB  |    1 %  |       1919912  |         20 MB  |    40100 / 480000  |       0
1104  |     11.0  |     447 GB  |     3.1 GB  |     424 GB  |    1 %  |       1919907  |         20 MB  |    40100 / 480000  |       0
1111  |     11.0  |     465 GB  |     2.6 GB  |     442 GB  |    1 %  |        494977  |         20 MB  |    40100 / 495000  |       0
1112  |     11.0  |     365 GB  |     2.6 GB  |     346 GB  |    1 %  |        389977  |         20 MB  |    40100 / 390000  |       0
1125  |     11.0  |     931 GB  |     2.6 GB  |     894 GB  |    0 %  |        974979  |         20 MB  |    40100 / 975000  |       0
1126  |     11.0  |     931 GB  |     2.6 GB  |     894 GB  |    0 %  |        974979  |         20 MB  |    40100 / 975000  |       0
--------------------------------------------------------------------------------------------------------------------------------------
   6  |      1.0  |     3.5 TB  |      16 GB  |     3.4 TB  |    0 %  |       6674731  |        122 MB  |  240600 / 3795000 |       0
```

Note

Without specifying instance the first instance is assumed - `11.0` as in the above example. The second, third and fourth `storpool_server` instance would be `11.1`, `11.2` and `11.3` accordingly.

To list the servers that are blocked and could not join the cluster for some reason:

```
# storpool server blocked
cluster waiting, mgmt on node 12
  server  11.0 waiting on node 11
missing:1201,1202,1203,1204,1212,1221,1222,1223,1224,1225,1226,1301,1302,1303,1304,1311,1312,1321,1322,1323,1324,1325,1326,4043
pending:1103,1104,1111,1112,1125,1126
  server  12.0    down on node 12
  server  13.0    down on node 13
  server  14.0 waiting on node 14
missing:1201,1202,1203,1204,1212,1221,1222,1223,1224,1225,1226,1301,1302,1303,1304,1311,1312,1321,1322,1323,1324,1325,1326,4043
pending:1403,1404,1411,1412,1421,1423
  server  11.1 waiting on node 11
missing:1201,1202,1203,1204,1212,1221,1222,1223,1224,1225,1226,1301,1302,1303,1304,1311,1312,1321,1322,1323,1324,1325,1326,4043
pending:1101,1102,1121,1122,1123,1124
  server  12.1    down on node 12
  server  13.1    down on node 13
  server  14.1 waiting on node 14
missing:1201,1202,1203,1204,1212,1221,1222,1223,1224,1225,1226,1301,1302,1303,1304,1311,1312,1321,1322,1323,1324,1325,1326,4043
pending:1401,1402,1424,1425,1426
```

Services

Check the state of all services presently running in the cluster and their uptime:

```
# storpool service list
cluster running, mgmt on node 12
      mgmt    11 running on node 11 ver 16.01.779, started 2017-01-17 14:23:53, uptime 18:30:09
      mgmt    12 running on node 12 ver 16.01.779, started 2017-01-17 14:23:49, uptime 18:30:13 active
      mgmt    13 running on node 13 ver 16.01.779, started 2017-01-17 14:23:47, uptime 18:30:15
      mgmt    14 running on node 14 ver 16.01.779, started 2017-01-18 05:06:53, uptime 03:47:09
    server  11.0 running on node 11 ver 16.01.779, started 2017-01-18 08:29:15, uptime 00:24:47
    server  12.0 running on node 12 ver 16.01.779, started 2017-01-18 08:29:35, uptime 00:24:27
    server  13.0 running on node 13 ver 16.01.779, started 2017-01-18 08:29:54, uptime 00:24:08
    server  14.0 running on node 14 ver 16.01.779, started 2017-01-18 08:28:21, uptime 00:25:41
    server  11.1 running on node 11 ver 16.01.779, started 2017-01-18 08:29:25, uptime 00:24:37
    server  12.1 running on node 12 ver 16.01.779, started 2017-01-18 08:29:41, uptime 00:24:21
    server  13.1 running on node 13 ver 16.01.779, started 2017-01-18 08:34:12, uptime 00:19:50
    server  14.1 running on node 14 ver 16.01.779, started 2017-01-18 08:28:24, uptime 00:25:38
    client    11 running on node 11 ver 16.01.779, started 2017-01-17 14:23:52, uptime 18:30:10
    client    12 running on node 12 ver 16.01.779, started 2017-01-17 14:23:48, uptime 18:30:14
    client    13 running on node 13 ver 16.01.779, started 2017-01-17 14:23:47, uptime 18:30:15
    client    14 running on node 14 ver 16.01.779, started 2017-01-18 05:06:54, uptime 03:47:08
    bridge    11 running on node 11 ver 16.01.779, started 2017-01-18 08:27:54, uptime 00:26:08
```

Disk

The disk submenu is for quering or managing the avaiable disks in the cluster.

To display all available disks in all server instances in the cluster:

```
# storpool disk list
disk  |   server  |    size    |    used    |  est.free  |      %  |  free entries  |  on-disk size  |  allocated objects |  errors
1101  |     11.1  |    447 GB  |    3.6 GB  |    424 GB  |    1 %  |       1919976  |         20 MB  |    40100 / 480000  |       0
1102  |     11.1  |    447 GB  |    3.6 GB  |    424 GB  |    1 %  |       1919976  |         20 MB  |    40100 / 480000  |       0
1103  |     11.0  |    447 GB  |    3.6 GB  |    424 GB  |    1 %  |       1919976  |         20 MB  |    40100 / 480000  |       0
1104  |     11.0  |    447 GB  |    3.6 GB  |    424 GB  |    1 %  |       1919976  |         20 MB  |    40100 / 480000  |       0
1111  |     11.0  |    465 GB  |    3.1 GB  |    442 GB  |    1 %  |        494976  |         20 MB  |    40100 / 495000  |       0
1112  |     11.0  |    365 GB  |    3.1 GB  |    345 GB  |    1 %  |        389976  |         20 MB  |    40100 / 390000  |       0
[snip]
1425  |     14.1  |    931 GB  |    2.6 GB  |    894 GB  |    0 %  |        974980  |         20 MB  |    40100 / 975000  |       0
1426  |     14.1  |    931 GB  |    2.6 GB  |    894 GB  |    0 %  |        974979  |         20 MB  |    40100 / 975000  |       0
--------------------------------------------------------------------------------------------------------------------------------------
47  |     8.0  |     30 TB  |    149 GB  |     29 TB  |    0 %  |      53308967  |        932 MB  |  1844600 / 32430000 |       0
```

To display additional info regarding disks:

```
# storpool disk list info
disk   |  server  |    device    |       model        |           serial            |             description          |  SSD  |  WBC  |
flags          |
1101  |    11.1  |  /dev/sdf1   |  Micron_M500DC_MTFDDAK480MBB  |  14250C63689B               |                                  |  SSD
|       |                        |
1102  |    11.1  |  /dev/sde1   |  Micron_M500DC_MTFDDAK480MBB  |  14250C6368EC               |                                  |  SSD
|       |                        |
1103  |    11.0  |  /dev/sdk1   |  Micron_M500DC_MTFDDAK480MBB  |  14250C636872               |                                  |  SSD
|       |                        |
1104  |    11.0  |  /dev/sdl1   |  Micron_M500DC_MTFDDAK480MBB  |  14250C6368E5               |                                  |  SSD
|       |                        |
1111  |    11.0  |  /dev/sdj1   |  Hitachi_HDS721050CLA360  |  JP1512FR1GZBNK             |                                  |       |
WBC  |                        |
1112  |    11.0  |  /dev/sdi1   |  Hitachi_HDS721050CLA360  |  JP1532FR1HAU4K             |                                  |       |
WBC  |                        |
[snip]
1425  |    14.1  |  /dev/sdg1   |  Hitachi_HUA722010CLA330  |  JPW9K0N13RJMVL             |                                  |       |
WBC  |                        |
1426  |    14.1  |  /dev/sdf1   |  Hitachi_HUA722010CLA330  |  JPW9K0N13SJ3DL             |                                  |       |
WBC  |                        |
```

To display internal statistics about each disk:

```
# storpool disk list internal
------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------
| disk | server |        aggregate scores        |         wbc pages         |  wbc iops   |  wbc bw.    |     scrub bw |
scrub ETA | last scrub completed |
------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------
| 1101 |   11.1 |        0 |        0 |        0 |       - + -     / -     |          - |          - |          - |
- |                      - |
| 1102 |   11.1 |        0 |        0 |        0 |       - + -     / -     |          - |          - |          - |
- |                      - |
| 1103 |   11.0 |        0 |        0 |        0 |       - + -     / -     |          - |          - |          - |
- |                      - |
| 1104 |   11.0 |        0 |        0 |        0 |       - + -     / -     |          - |          - |          - |
- |                      - |
| 1111 |   11.0 |        0 |        0 |        0 |       0 + 0     / 2560  |       8249 |  20 MB/sec. |          - |
- |                      - |
| 1112 |   11.0 |        0 |        0 |        0 |       0 + 0     / 2560  |       6499 |  20 MB/sec. |          - |
- |                      - |
[snip]
| 1425 |   14.1 |        0 |        0 |        0 |       0 + 0     / 2560  |      16249 |  20 MB/sec. |          - |
- |                      - |
| 1426 |   14.1 |        0 |        0 |        0 |       0 + 0     / 2560  |      16249 |  20 MB/sec. |          - |
- |                      - |
------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------
```

The sections in this output explained:

- `aggregate scores` - Internal values representing how much data is about to be defragmented on the particular drive. Usually between 0 and 1, on heavily loaded clusters the rightmost column might get into the hundreds or even thousands if some drives are severely loaded.
- `wbc pages`, `wbc iops`, `wbc bw.` - Internal statistics for each drive that have the write back cache in StorPool enabled.
- `scrub bw` - The scrubbing speed in MB/s
- `scrub ETA` - Approximate time/date when the scrubbing operation will complete for this drive.
- `last scrub completed` - The last time/date when the drive was scrubbed

Note

The default installation includes a cron job on the management nodes that starts a scrubbing job for all drives in the cluster once per week.

To set additional information for some of the disks, seen in description field with `storpool disk list info`:

```
# storpool disk 1111 description HBA2_port7
OK
# storpool disk 1104 description FAILING_SMART
OK
```

To mark a device as temporarily unavailable:

```
# storpool disk 1111 eject
OK
```

This will stop data replication for this disk, but will keep info on the placement groups in which it participated and which volume objects it contained.

Note

The command above will refuse to eject the disk if this operation would lead to volumes or snapshots in `down` state. Usually when the last up-to-date copy for some parts of a volume/snapshot is on this disk.

This drive will be visible with `storpool disk list` as missing, e.g.:

```
# storpool disk list
    disk  |  server  |    size    |    used    |  est.free  |      %  |  free entries  |  on-disk size  |  allocated objects |  errors
    [snip]
    1422  |    14.1  |          -  |          -  |          -  |    - %  |              -  |              -  |        - / -       |       -
    [snip]
```

Attention

This operation leads to degraded redundancy for all volumes or snapshots that have data on the ejected disk.

To mark a disk as unavailable by first re-balancing all data out to the other disks in the cluster and only then eject it:

```
# storpool disk 1422 softEject
OK
Balancer auto mode currently OFF. Must be ON for soft-eject to complete.
```

Note

This option requires StorPool balancer to be started after the above was issued, see more in the Balancer section below.

To remove a disk from the list of reported disks and all placement groups it participates in:

```
# storpool disk 1422 forget
OK
```

To get detailed information about given disk:

```
# storpool disk 1101 info
OK
```

To get detailed information about the objects on a particular disk:

```
# storpool disk 1101 list
```

To get detailed information about the active requests that the disk is performing at the moment:

```
# storpool disk 1101 activeRequests
-----------------------------------------------------------------------------------------------------------------------------------
| request ID                     |  request IDX |               volume |         address |       size |       op |     time active |
-----------------------------------------------------------------------------------------------------------------------------------
| 9226469746279625682:285697101441249070 |            9 |           testvolume |     85276782592 |     4.0 KB |      read |          0
msec |
| 9226469746279625682:282600876697431861 |           13 |           testvolume |     96372936704 |     4.0 KB |      read |          0
msec |
| 9226469746279625682:278097277070061367 |           19 |           testvolume |     46629707776 |     4.0 KB |      read |          0
msec |
| 9226469746279625682:278660227023482671 |          265 |           testvolume |     56680042496 |     4.0 KB |     write |          0
msec |
-----------------------------------------------------------------------------------------------------------------------------------
```

To issue retrim operation on a disk (available for SSD disks only):

```
# storpool disk 1101 retrim
OK
```

To start, pause or continue a scrubbing operation for a disk:

```
# storpool disk 1101 scrubbing start
OK
# storpool disk 1101 scrubbing pause
OK
# storpool disk 1101 scrubbing continue
OK
```

Note

Use `storpool disk list internal` to check the status of a running scrub operation or when was the last completed scrubbing operation for this disk.

Placement Groups

The placement groups are predefined sets of disks, over which volume objects will be replicated. It is possible to specify which individual disks to add to the group or alternatively to add all disks provided by a specific server (useful for servers with the same set of disks).

To display the defined placement groups in the cluster:

```
# storpool placementGroup list
name
default
hdd
ssd
```

To display details about a placement group:

```
# storpool placementGroup ssd list
type   | id
disk   | 1101 1201 1301 1401
```

Creating a new placement group or extend an existing one requires specifying its name and providing one or more disks to be added:

```
# storpool placementGroup ssd addDisk 1102
OK
# storpool placementGroup ssd addDisk 1202
OK
# storpool placementGroup ssd addDisk 1302 addDisk 1402
OK
# storpool placementGroup ssd list
type | id
disk   | 1101 1102 1201 1202 1301 1302 1401 1402
```

To remove one or more disks from a placement group use:

```
# storpool placementGroup ssd rmDisk 1402
OK
# storpool placementGroup ssd list
type | id
disk   | 1101 1102 1201 1202 1301 1302 1401
```

To rename a placement group:

```
# storpool placementGroup ssd rename M500DC
OK
```

The unused placement groups can be removed. To avoid accidents, the name of the group must be entered twice:

```
# storpool placementGroup ssd delete ssd
OK
```

Volumes

The volumes are the basic service of the StorPool storage system. A volume always have a name and a certain size. It can be read from and written to. It could be attached to hosts as read-only or read-write block device under the `/dev/storpool` directory. The volume name is a string consisting of one or more of the allowed characters - upper and lower latin letters (`a-z,A-Z`), numbers (`0-9`) and the delimiter dot (`.`), colon (`:`), dash (`-`) and underscore (`_`).

When a volume is created, at minimum the <volumeName>, the size and replication must be specified:

```
# storpool volume testvolume size 100G replication 3
```

Additional parameters that can be used:

- `placeAll` - place all objects in placementGroup (Default value: default)
- `placeTail` - name of placementGroup for reader (Default value: same as `placeAll` value)
- `template` - use template with preconfigured placement, replication and/or limits (please check for more in Templates section)
- `parent` - use a snapshot as a parent for this volume
- `baseOn` - use parent volume, this will create a transient snapshot used as a parent (please check for more in `Snapshots`_ section)
- `iops` - set the maximum IOPS limit for this volume (in IOPS)
- `bw` - set maximum bandwidth limit (in MB/s)

To list all available volumes:

```
# storpool volume list
-----------------------------------------------------------------------------------------------------------------------------------------
| volume                 |   size  | repl. | placeAll    | placeTail  |   iops  |    bw   | parent                 | template             |
-----------------------------------------------------------------------------------------------------------------------------------------
| testvolume             |  100 GB |     3 | ultrastar   | ssd        |       - |       - | testvolume@35691       | hybrid-u-r3          |
-----------------------------------------------------------------------------------------------------------------------------------------
```

To get an overview of all volumes and snapshots and their state in the system use:

```
# storpool volume status
------------------------------------------------------------------------------------------------------------------------------------
| volume                 |    size | repl. | alloc % |  stored | on disk | syncing | missing | status      | flags | drives down    |
------------------------------------------------------------------------------------------------------------------------------------
| testvolume             |  100 GB |     3 |   0.0 % |    0  B |    0  B |    0  B |    0  B | up          |       |                |
| testvolume@35691       |  100 GB |     3 | 100.0 % |  100 GB |  317 GB |    0  B |    0  B | up          | S     |                |
------------------------------------------------------------------------------------------------------------------------------------
| 2 volumes              |  200 GB |       |  50.0 % |  100 GB |  317 GB |    0  B |    0  B |             |       |                |
------------------------------------------------------------------------------------------------------------------------------------

Flags:
  S - snapshot
  B - balancer blocked on this volume
  D - decreased redundancy (degraded)
  M - migrating data to a new disk
```

The columns in this output are:

- `volume` - name of the volume or snapshot (see `flags` below)
- `size` - provisioned volume size, the visible size inside a VM for example
- `repl.` - number of copies for this volume
- `alloc %` - how much space was used on this volume in percent
- `stored` - space allocated on this volume
- `on disk` - the size allocated on all drives in the cluster after replication and the overhead from data protection
- `syncing` - how much data is not in sync after a drive or server was missing, the data is recovered automatically once the missing drive or server is back in the cluster
- `missing` - shows how much data is not available for this volume when the volume is with status `down`, see `status` below
- `status` - shows the status of the volume, which could be one of:
    - `up` - all copies are available
    - `down` - none of the copies are available for some parts of the volume
    - `up soon` - all copies are available and the volume will soon get up
- `flags` - flags denoting features of this volume:
    - `S` - stands for snapshot, which is essentially a read-only (frozen) volume
    - `B` - used to denote that the balancer is blocked for this volume (usually when some of the drives are missing)
    - `D` - this flag is displayed when some of the copies is either not available or outdated and the volume is with decreased redundancy
    - `M` - displayed when changing the replication or a cluster re-balance is in progress
    - `drives down` - displayed when the volume is in `down` state, displaying the drives required to get the volume back up.

Size is in `B`/`KB`/`MB`/`GB` or `TB`.

To check the estimated used space by the volumes in the system use:

```
# storpool volume usedSpace
----------------------------------------------------------------------
| volume                  |       size | repl. |     stored |       used |
----------------------------------------------------------------------
| testvolume              |     100 GB |     3 |     1.9 GB |     100 GB |
----------------------------------------------------------------------
```

Note

The used column shows how much data is accessible and reserved for this volume.

To list the target disk sets and objects of a volume:

```
# storpool volume testvolume list
volume testvolume
size 100 GB
replication 3
placeAll ultrastar
placeTail ssd
target disk sets:
       0: 1122 1323 1203
       1: 1424 1222 1301
       2: 1121 1324 1201
[snip]
  object: disks
       0: 1122 1323 1203
       1: 1424 1222 1301
       2: 1121 1324 1201
[snip]
```

Hint

In this example the volume is with hybrid placement with two copies on HDDs and one copy on SSDs (the rightmost column). The target disk sets are list of triplets of drives in the cluster used as a template for the actual objects of the volume.

To get detailed info about the disks used for this volume and the number of objects on each of them use:

```
# storpool volume testvolume info
    diskId | count
  1101 |   200
  1102 |   200
  1103 |   200
  [snip]
chain                 | count
1121-1222-1404        |  25
1121-1226-1303        |  25
1121-1226-1403        |  25
```

To rename a volume use:

```
# storpool volume testvolume rename newvolume
OK
```

To resize a volume up:

```
# storpool volume testvolume size +1G
OK
```

To shrink a volume (resize down):

```
# storpool volume testvolume size 50G shrinkOk
```

Attention

Shrinking of a storpool volume changes the size of the block device, but does not adjust the size of LVM or filesystem contained in the volume. Failing to adjust the size of the filesystem or LVM prior to shrinking the StorPool volume would result in data loss.

To delete a volume use:

```
# storpool volume vol1 delete vol1
```

Note

To avoid accidents, the volume name must be entered twice. Attached volumes cannot be deleted even when not used as a safety precaution, more in Attachments.

A volume could be converted from based on a snapshot to a stand-alone volume. For example the `testvolume` below is based on an anonymous snapshot:

```
# /usr/lib/storpool/storpool_tree.pl
StorPool
  `-testvolume@37126
     `-testvolume
```

To rebase it against root (known also as “promote”) use:

```
# storpool volume testvolume rebase
OK
# /usr/lib/storpool/storpool_tree.pl
StorPool
  `-testvolume
  `-testvolume@37126
```

The rebase operation could also be to a particular snapshot from a chain of parent snapshots on this child volume:

```
# /usr/lib/storpool/storpool_tree.pl
StorPool
  `-testvolume-snap1
     `-testvolume-snap2
        `-testvolume-snap3
           `-testvolume
# storpool volume testvolume rebase testvolume-snap2
OK
```

After the operation the volume is directly based on `testvolume-snap2` and includes all changes from `testvolume-snap3`:

```
# /usr/lib/storpool/storpool_tree.pl
StorPool
  `-testvolume-snap1
     `-testvolume-snap2
        `-testvolume
        `-testvolume-snap3
```

To backup a volume named `testvolume` in a configured remote location `StorPoolLab-Sofia` use:

```
# storpool volume testvolume backup StorPoolLab-Sofia
OK
```

After this operation a temporary snapshot will be created and will be transferred in `StorPoolLab-Sofia` location. After the transfer completes, the local temporary snapshot will be deleted and the remote snapshot will be visible as exported from `StorPoolLab-Sofia`, check Remote Snapshots for more on working with snapshot exports.

Snapshots

Snapshots are read-only point-in-time images of volumes. They are created once and cannot be changed. They can be attached to hosts as read-only block devices under `/dev/storpool`. Volumes and snapshots share the same name-space, thus their names are unique within a StorPool cluster. Volumes can be based on snapshots. Such volumes contain only the changes since the snapshot was taken. After a volume is created from a snapshot, writes will be recorded within the volume. Reads from volume may be served by volume or by its parent snapshot depending on whether volume contains changed data for the read request or not.

To create an unnamed (known also as anonymous) snapshot of a volume use:

```
# storpool volume testvolume snapshot
OK
```

This will create a snapshot named `testvolume@<ID>`, where `ID` is an unique serial number.

To create a named snapshot of a volume use:

```
# storpool volume testvolume snapshot testsnap
OK
```

To list the snapshots use:

```
# storpool snapshot list
----------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------
| snapshot                 |  size   | repl. | placeAll   | placeTail  | created on          | volume               |   iops  |    bw
| parent                  | template             | flags |
----------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------
| testsnap                 |  100 GB |     3 | hdd        | ssd        | 2017-01-30 04:11:23 | testvolume           |       - |       -
| testvolume@1430         | hybrid-r3            |       |
| testvolume@1430          |  100 GB |     3 | hdd        | ssd        | 2017-01-30 03:56:58 | testvolume           |       - |       -
|                         | hybrid-r3            | A     |
----------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------

Flags:
  A - anonymous snapshot with auto-generated name
  B - bound snapshot
  D - snapshot currently in the process of deletion
  T - transient snapshot (created during volume cloning)
```

To list the snapshots only for a particular volume use:

```
# storpool volume testvolume list snapshots
[snip]
```

A volume might directly be converted to a snapshot, the operation is also known as `freeze`:

```
# storpool volume testvolume freeze
OK
```

Note that the operation will fail if the volume is attached read-write, more in Attachments

To create a bound snapshot on a volume use:

```
# storpool volume testvolume bound snapshot testsnap-bound
OK
```

This snapshot will be automatically deleted when the last child volume created from it is deleted. Useful for non-persistent images.

To list the target disk sets and objects of a snapshot:

```
# storpool snapshot testsnap list
[snip]
```

The output is similar as with `storpool volume <volumename> list`.

To get detailed info about the disks used for this snapshot and the number of objects on each of them use:

```
# storpool snapshot testsnap info
[snip]
```

The output is similar to the `storpool volume <volumename> info`.

To create a volume based on an existing snapshot (cloning) use:

```
# storpool volume testvolume parent centos73-base-snap
OK
```

Same possible through the use of templates with a parent snapshot (See Templates):

```
# storpool volume spd template centos73-base
OK
```

Create a volume based on another existing volume (cloning):

```
# storpool volume testvolume1 baseOn testvolume
OK
```

Note

This opeartion will first create an anonymous bound snapshot on `testvolume` and then will create `testvolume1` with this snapshot as parent. The snapshot will exist until both volumes are deleted and will be automatically deleted afterwards.

To delete a snapshot use:

```
# storpool snapshot spdb_snap1 delete spdb_snap1
OK
```

Note

To avoid accidents, the name of the snapshot must be entered twice.

A snapshot could also be binded to its child volumes, it will exist until all child volumes are deleted:

```
# storpool snapshot testsnap bind
OK
```

The opposite operation is also possible, to unbind such snapshot use:

```
# storpool snapshot testsnap unbind
OK
```

To get the space that will be freed if a snapshot is deleted use:

```
# storpool snapshot space
----------------------------------------------------------------------------------------------
| snapshot              | on volume              |      size | repl. |      stored |       used |
----------------------------------------------------------------------------------------------
| testsnap              | testvolume             |    100 GB |     3 |       27 GB |    -135 GB |
| testvolume@3794       | testvolume             |    100 GB |     3 |       27 GB |     1.9 GB |
| testvolume@3897       | testvolume             |    100 GB |     3 |      507 MB |     432 KB |
| testvolume@3899       | testvolume             |    100 GB |     3 |      334 MB |     224 KB |
| testvolume@4332       | testvolume             |    100 GB |     3 |       73 MB |      36 KB |
| testvolume@4333       | testvolume             |    100 GB |     3 |       45 MB |      40 KB |
| testvolume@4334       | testvolume             |    100 GB |     3 |       59 MB |      16 KB |
----------------------------------------------------------------------------------------------
```

Used mainly for accounting purposes. The used column stands for the amount of data that would be freed from the underlying drives (before replication) if the snapshot is removed. The used column could be negative in some cases when the snapshot has more than one child. In these cases deleting the snapshot would "free" negative space i.e. will end up taking more space on the underlying disks.

Similar to volumes a snapshot could have different placementGroups or other attributes, as well as templates:

```
# storpool snapshot testsnap template all-ssd
OK
```

Additional parameters that may be used:

- `placeAll` - place all objects in placementGroup (Default value: default)
- `placeTail` - name of placementGroup for reader (Default value: same as `placeAll` value)
- `template` - use template with preconfigured placement, replication and/or limits (please check for more in Templates section)
- `iops` - set the maximum IOPS limit for this snapshot (in IOPS)
- `bw` - set maximum bandwidth limit (in MB/s)

Note

The bandwidth and IOPS limits are concerning only the particular snapshot if it is attached and does not limit any child volumes using this snapshot as parent.

Also similar to the same operation with volumes a snapshot could be renamed with:

```
# storpool snapshot testsnap rename ubuntu1604-base
OK
```

A snapshot could also be rebased to root (promoted) or rebased to another parent snapshot in a chain:

```
# storpool snapshot testsnap rebase # [parent-snapshot-name]
OK
```

To delete a snapshot use:

```
# storpool snapshot testsnap delete testsnap
OK
```

Note

A snapshot sometimes will not get deleted immediately, during this period of time it will be visible with `*` in the output of `storpool volume status` or `storpool snapshot list`.

To set a snapshot for deferred deletion use:

```
# storpool snapshot testsnap deleteAfter 1d
OK
```

The above will set a target delete date for this snapshot in exactly one day from the present time.

Note

The snapshot will be deleted at the desired point in time only if delayed snapshot delete was enabled in the local cluster, check Management configuration section from this guide.

Remote snapshots

In case multi-site is enabled (the cluster have a bridge service running) a snapshot could be exported and become visible to other configured locations.

For example to export a snapshot `snap1` to a remote location named `StorPoolLab-Sofia` use:

```
# storpool snapshot snap1 export StorPoolLab-Sofia
OK
```

To list the presently exported snapshots use:

```
# storpool snapshot list exports
-----------------------------------------------------------------
| location                | snapshot     | globalId     | backingUp |
-----------------------------------------------------------------
| StorPoolLab-Sofia       | snap1        | nzkr.b.cuj   | false     |
-----------------------------------------------------------------
```

To list the snapshots exported from remote sites use:

```
# storpool snapshot list remote
---------------------------------------------------------------------------------
| location | remoteId | name      | onVolume | size         | creationTimestamp   |
---------------------------------------------------------------------------------
| s02      | a.o.cxz  | snapshot1 |          | 107374182400 | 2017-02-10 03:21:42 |
---------------------------------------------------------------------------------
```

Single snapshot could be exported to multiple configured locations.

To unexport a local snapshot use:

```
# storpool snapshot snap1 unexport StorPoolLab-Sofia
OK
```

The remote location could be swapped by the keyword `all`. This will attempt to unexport the snapshot from all location it was previously exported to.

Note

If the snapshot is presently being transferred the `unexport` operation will fail. It could be forced by adding `force` to the end of the unexport command, however this is discouraged in favor to waiting for any active transfer to complete.

To unexport a remote snapshot use:

```
# storpool snapshot remote s02 a.o.cxz unexport
OK
```

The snapshot will no longer be visible with `storpool snapshot list remote`.

To unexport a remote snapshot and also set for deferred deletion in the remote site:

```
# storpool snapshot remote s02 a.o.cxz unexport deleteAfter 1h
OK
```

This will attempt to set a target delete date for `a.o.cxz` in the remote site in exactly one hour from the present time for this snapshot. If the `minimumDeleteDelay` in the remote site has a higher value, e.g. 1 day the selected value will be overwritten with the `minimumDeleteDelay` - in this example 1 day. For more info on deferred deletion check the Multi Site section of this guide.

Attachments

Attaching a volume or snapshot makes it accessible to client under `/dev/storpool`. Volumes can be attached as read-only or read-write. Snapshots are always attached read-only.

Attaching a volume `testvolume` to a client with ID `1`. This creates the block device `/dev/storpool/testvolume`:

```
# storpool attach volume testvolume client 1
OK
```

To attach a volume/snapshot to the node you are currently connected to use:

```
# storpool attach volume testvolume here
OK
# storpool attach snapshot testsnap here
OK
```

By default this command will block until the volume is attached to the client and the `/dev/storpool/<volumename>` symlink is created. For example if the `storpool_block` service has not been started the command will wait indefinitely. To set a timeout for this operation use:

```
# storpool attach volume testvolume here timeout 10 # seconds
OK
```

To completely disregard the readiness check use:

```
# storpool attach volume testvolume here noWait
OK
```

Note

The use of `noWait` is discouraged in favor of the default behaviour of the `attach` command.

Attaching a volume will create a read-write block device attachment by default. To attach it read-only use:

```
# storpool volume testvolume2 attach client 12 mode ro
OK
```

To list all attachments use:

```
# storpool attach list
----------------------------------------
| client | volume               | mode |
----------------------------------------
|     11 | testvolume           | RW   |
|     12 | testvolume1          | RW   |
|     12 | testvolume2          | RO   |
|     14 | testsnap             | RO   |
----------------------------------------
```

To detach use:

```
# storpool detach volume testvolume client 1 # or 'here' if you are on client 1
```

If a volume is actively being written or read from, a detach operation will fail:

```
# storpool detach volume testvolume client 11
Error: 'testvolume' is open at client 11
```

In this case the detach could be forced with, however beware that forcing a detachment is discouraged:

```
# storpool detach volume testvolume client 11 force yes
OK
```

If a volume or snapshot is attached to more than one client it could be detached from all nodes with a single CLI command:

```
# storpool detach volume testvolume all
OK
# storpool detach snapshot testsnap all
OK
```

Client

To check the status of the active `storpool_block` services in the cluster use:

```
# storpool client status
-----------------------------------
|  client  |        status         |
-----------------------------------
|       11 | ok                    |
|       12 | ok                    |
|       13 | ok                    |
|       14 | ok                    |
-----------------------------------
```

To wait until a client is updated use:

```
# storpool client 13 sync
OK
```

This is a way to ensure a volume with changed size is visible with its new size to any clients it is attached to.

To show detailed information about the active requests on particular client in this moment use:

```
# storpool client 13 activeRequests
------------------------------------------------------------------------------------------------------------------------------------
| request ID                      | request IDX |                volume |         address |       size |       op |    time active |
------------------------------------------------------------------------------------------------------------------------------------
| 9224499360847016133:3181950     | 1044        | testvolume            |     10562306048 |     128 KB |    write |        65 msec |
| 9224499360847016133:3188784     | 1033        | testvolume            |     10562437120 |      32 KB |     read |        63 msec |
| 9224499360847016133:3188977     | 1029        | testvolume            |     10562568192 |     128 KB |     read |        21 msec |
| 9224499360847016133:3189104     | 1026        | testvolume            |     10596122624 |     128 KB |     read |         3 msec |
| 9224499360847016133:3189114     | 1035        | testvolume            |     10563092480 |     128 KB |     read |         2 msec |
| 9224499360847016133:3189396     | 1048        | testvolume            |     10629808128 |     128 KB |     read |         1 msec |
------------------------------------------------------------------------------------------------------------------------------------
```

Templates

Templates are enabling easy and consistent setup for large number of volumes with common attributes, e.g. replication, placement groups or common parent snapshot.

To create a template use:

```
# storpool template magnetic replication 3 placeAll hdd
OK
# storpool template hybrid replication 3 placeAll hdd placeTail ssd
OK
```

To list all created templates use:

```
# storpool template list
-------------------------------------------------------------------------------------------------------------
| template              |   size  | repl. | placeAll    | placeTail   |   iops  |    bw   | parent               |
-------------------------------------------------------------------------------------------------------------
| magnetic              |       - |     3 | hdd         | hdd         |       - |       - |                      |
| hybrid                |       - |     3 | hdd         | ssd         |       - |       - |                      |
-------------------------------------------------------------------------------------------------------------
```

To get the status of a template with detailed info on the usage and the available space left with this placement use:

```
# storpool template status
-------------------------------------------------------------------------------------------------------------------------------------------
-----------
| template              | place all   | place tail | repl. | volumes | snapshots/removing |    size | capacity |  avail. | avail. all |
avail. tail |
-------------------------------------------------------------------------------------------------------------------------------------------
-----------
| magnetic              | hdd         | hdd        |     3 |     115 |       631/0        |   28 TB |    80 TB |   52 TB |     240 TB |
240 TB |
| hybrid                | hdd         | hdd        |     3 |     208 |       347/9        |   17 TB |    72 TB |   55 TB |     240 TB |
72 TB |
-------------------------------------------------------------------------------------------------------------------------------------------
-----------
```

To change template attributes directly use:

```
# storpool template hdd-only size 120G
OK
# storpool template hybrid size 40G iops 4000
OK
```

Parameters that can be set:

- `replication` - change the number of copies for volumes or snapshots created with this template
- `size` - default size if not specified for each volume created with this template
- `placeAll` - place all objects in placementGroup (Default value: default)

- `placeTail` - name of placementGroup for reader (Default value: same as `placeAll` value)
- `iops` - set the maximum IOPS limit for this snapshot (in IOPS)
- `bw` - set maximum bandwidth limit (in MB/s)
- `parent` - set parent snapshot for all volumes created in this template

Changing parameters on a template concerns only new volumes created with this template. To actually change the updated parameters for all volumes already created with this template use `propagate`, the following example changes the bandwidth limit for all volumes and snapshots created with this template:

```
# storpool template magnetic bw 100MB propagate
OK
```

Attention

Dropping the replication (e.g. from triple to dual) of a large number of volumes is an almost instant operation, however returning them back to triple is similar to creating the third copy for the first time.

To rename a template use:

```
# storpool template magnetic rename backup
OK
```

To delete a template use:

```
# storpool template hdd-only delete hdd-only
OK
```

Note

The delete operation might fail if there are volumes/snapshots that are created with this template.

Relocator

The relocator is internal StorPool service that takes care of data reallocation in case of change of volume's replication group parameters or in case of any pending rebase operations. This service is turned on by default.

In case of need the relocator could be turned off with:

```
# storpool relocator off
OK
```

To turn back on use:

```
# storpool relocator on
OK
```

To display the relocator status:

```
# storpool relocator status
relocator on, no volumes to relocate
```

The following additional relocator commands are available:

- `storpool relocator disks` - returns the state of the disks after the relocator finishes all presently running tasks, as well as the quantity of objects and data each drive still needs to recover. The output is the same as with `storpool balancer disks` after the balancing task has been committed, see Balancer for more details.

- `storpool relocator volume <volumename> disks` or `storpool relocator snapshot <snapshotname> disks` - shows the same information as the `storpool relocator disks` with the pending operations specific volume or snapshot.

Balancer

The balancer is used to redistribute data in case a disk or set of disks (e.g. new node) were added to or removed from the cluster. By default it is off. It has to be turned on in case of changes in cluster configuration for redistribution of data to occur.

To display the status of the balancer:

```
# storpool balancer status
balancer waiting, auto off
```

To start the balancer and to check what will be the expected results from the re-balance operation before it is started use:

```
# storpool balancer run
OK
```

This will calculate the needed operations for the relocator to complete the re-balancing procedure. The expected results could be seen once the balancer completes the calculation with:

```
# storpool balancer disks
-----------------------------------------------------------------------------------------------------------------------------------
----------------------------------
|     disk | server |   size   |                  stored                  |                  on-disk                  |
objects                      |
-----------------------------------------------------------------------------------------------------------------------------------
----------------------------------
[snip]
|     1416 |   14.0 |   932 GB |   262 GB ->  0 GB    (-232 GB / 0  B)    |   271 GB ->  1 GB    (-239 GB / 1  GB)    |   60016 -> 26879
(-33137 / +0)      / 975000  |
[snip]
-----------------------------------------------------------------------------------------------------------------------------------
----------------------------------
|       37 |    4.0 |    23 TB |   6.9 TB -> 6.9 TB   (-2.3 GB / 488 GB)  |   7.1 TB -> 7.1 TB   (-1.3 GB / 505 GB)  | 1740950 ->
1734020   (-6930 / +64762)  / 24750000 |
-----------------------------------------------------------------------------------------------------------------------------------
----------------------------------
```

The output shows what changes will occur to the stored, on-disk and objects parameters for each disk during the rebalance if the proposed configuration is committed, in the above example all data from disk with ID `1411` will be dispersed to other drives in the cluster.

Checking this output is especially important in case of decreasing the number of disks in the cluster to ensure enough space on each disk will be available to complete the re-balance operation.

Some additional commands to check the status of the drives and data before proceeding with the re-balancing operation:

- `storpool balancer volume status` - shows which volumes and snapshots will be reallocated by the balancer.
- `storpool balancer volume disks` and `storpool balancer snapshot disks` - returns the new allocation per volume or snapshot on the disks.
- `storpool balancer volume <volumename> diskSets` and `storpool balancer snapshot <snapshotname> diskSets` - returns the new target disk sets for the given volume.
- `storpool balancer groups` - shows groups of volumes and snapshots with common root, the child volumes or snapshots usually inherit disk sets from root snapshot with minor exceptions.

After checking the additional information the re-balance operation could be either discarded or started.

To discard the re-balancing operation use:

```
# storpool balancer stop
OK
```

To actually commit the changes and start the relocations to the proposed changes use:

```
# storpool balancer commit
OK
```

After the commit all changes will be only visible with `storpool relocator disks` and many volumes and snapshots will have the `M` flag in the output of `storpool volume status` until all relocations are completed. The progress could be followed with `storpool task list` (see Tasks).

Tasks

The tasks are all outstanding operations on recovering or relocating data in the present or between two connected clusters.

For example if a disk with ID `1401` was not in the cluster for a period of time and is then returned, all outdated objects will be recovered from the other drives with the latest changes.

These recovery opertions could be listed with:

```
# storpool task list
----------------------------------------------------------------------------------------
|     disk |  task id | total size |  completed |    started |  remaining | % complete |
----------------------------------------------------------------------------------------
|     1401 |        0 |      51 GB |      45 GB |      32 MB |     6.7 GB |        86% |
|     1401 |        0 |      44 GB |       0  B |       0  B |      44 GB |         0% |
----------------------------------------------------------------------------------------
|    total |          |      96 GB |      45 GB |      32 MB |      51 GB |        46% |
----------------------------------------------------------------------------------------
```

Other cases when tasks operations could be listed are when a re-balancing operation was commited and relocations are in progress, as well as when a cloning operation for a remote snapshot in the local cluster is in progress.

Management Configuration

The `mgmtConfig` submenu is used to set some internal configuration parameters.

To list the presently configured parameters use:

```
# storpool mgmtConfig list
relocator on, interval 5 sec
relocator transaction: min objects 320, max objects 4294967295
relocator recovery: max tasks per disk 2, max objects per disk 3200
relocator recovery objects trigger 32
balancer auto off, interval 5 sec
snapshot delete interval 5 sec
disks soft-eject interval 5 sec
snapshot delayed delete off
```

To turn balancer into auto mode (default off) use:

```
# storpool mgmtConfig balancerAuto on
OK
```

When in `auto` mode, the balancer will which will periodically check for unequally distributed data over a set of drives and will start a re-balancing operation.

To select different interval for the periodic checks use (default 5 seconds):

```
# storpool mgmtConfig balancerAutoInterval 300000 # value in ms - 300 seconds or 5 minutes
```

The `balancerAuto` mode is discouraged in favor of manual re-balancing operations when needed, to turn off the auto re-balancing use:

```
# storpool mgmtConfig balancerAuto off
OK
```

To enable deferred snapshot deletion (default off) use:

```
# storpool mgmtConfig delayedSnapshotDelete on
OK
```

When enabled all snapshots with configured time to be deleted will be cleared at the configured date and time.

To change the default interval between periodic checks whether disks marked for ejection can actually be ejected (5 sec.) use:

```
# storpool mgmtConfig disksSoftEjectInterval 20000 # value in ms - 20 sec.
OK
```

To change the default interval (5 sec.) for the relocator to check if there is new work to be done use:

```
# storpool mgmtConfig relocatorInterval 20000 # value is in ms - 20 sec.
OK
```

To set a different than the default number of objects per disk (3200) in recovery at a time:

```
# storpool mgmtConfig relocatorMaxRecoveryObjectsPerDisk 2000 # value in number of objects per disk
OK
```

To change the default maximum number of recovery tasks per disk (2 tasks) use:

```
# storpool mgmtConfig relocatorMaxRecoveryTasksPerDisk 4 # value is number of tasks per disk - will set 4 tasks
OK
```

To change the minimum (default 320) or the maximum (default 4294967295) number of objects per transaction for the relocator use:

```
# storpool mgmtConfig relocatorMaxTrObjects 2147483647
OK
# storpool mgmtConfig relocatorMinTrObjects 640
OK
```

To change the maximum number of objects in recovery for a disk to be usable by the relocator (default 32) use:

```
# storpool mgmtConfig relocatorRecoveryObjectsTrigger 64
```

Please consult with StorPool support before changing the management configuration defaults.

Mode

Support for couple of different output modes is available both for the interactive shell and when the CLI is invoked directly. Some custom format options are available only for some operations.

Available modes:

- `csv` - Semicolon-separated values for some commands
- `json` - Processed JSON output for some commands
- `pass` - Pass the JSON response through
- `raw` - Raw output (display the HTTP request and response)

- `text` - Human readable output (default)

Example with switching to `csv` mode in the interactive shell:

```
StorPool> mode csv
OK
StorPool> net list
nodeId;flags;MAC - net 1;MAC - net 2;MAC - net 3;MAC - net 4
9;uU +;00:25:90:49:1B:DA;00:25:90:3C:B7:06;-;-
10;uU +;00:25:90:3B:B7:FF;00:25:90:3B:B7:FE;-;-
11;uU +;90:E2:BA:52:1A:6D;90:E2:BA:52:1A:6D;-;-
13;uU +;00:25:90:48:1A:F0;00:25:90:48:1A:F0;-;-
```

The same applies when using the CLI directly:

```
# storpool -f csv net list # the output is the same as above
[snip]
```

Multi server

The multi-server feature enables using up to four separate `storpool_server` instances on a single node. This makes sense for dedicated storage nodes or in case of heavily loaded converged setup with more resources isolated for storage networking.

For example a dedicated storage node with 36 drives would provide better peak performance with 4 server instances each controlling 1/4th of all disks/SSDs than a single instance. Another good example would be a converged node with 16 SSDs/HDDs, which would provide better peak performance with two server instances each controlling half of the drives and running on separate CPU cores or even running on two threads on a single CPU core, than the performance provided from a single server instance.

Enabling multi-server is as easy as intalling the `multiserver` module at install time.

To configure the CPUs on which the different instances are running, configure them in the cpuset cgroups in which each of the instances is started, check for the defaults in Cgroup setup.

Controlling which instance is controlling each of the drives in the node is controlled with `storpool_initdisk`.

For example if we have two drives with IDs of `1101` and `1102`, both controlled by the first server instance the output from `storpool_initdisk` would look like this:

```
# storpool_initdisk --list
/dev/sde1, diskId 1101, version 10007, server instance 0, cluster init.b, SSD
/dev/sdf1, diskId 1102, version 10007, server instance 0, cluster init.b, SSD
```

Setting the second SSD drive (`1102`) to be controlled by the second server instance would look like this:

```
#  storpool_initdisk -r -i 1 /dev/sdf1
```

Hint

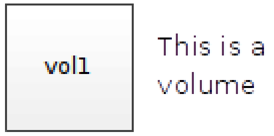
The above command will fail if the `storpool_server` service is running, please eject the disk prior to setting an instance.

In some occasions if the first server instance was configured with large amount of cache (check `SP_CACHE_SIZE` in Configuration Guide) first splice the cache between the different instances (e.g. from `8192` to `4096` for each instance).

Attention

Changing the size of the cache file requires removing it from `/dev/shm/storpool.cache` after shutting down the `storpool_server` instance, prior to starting it again.

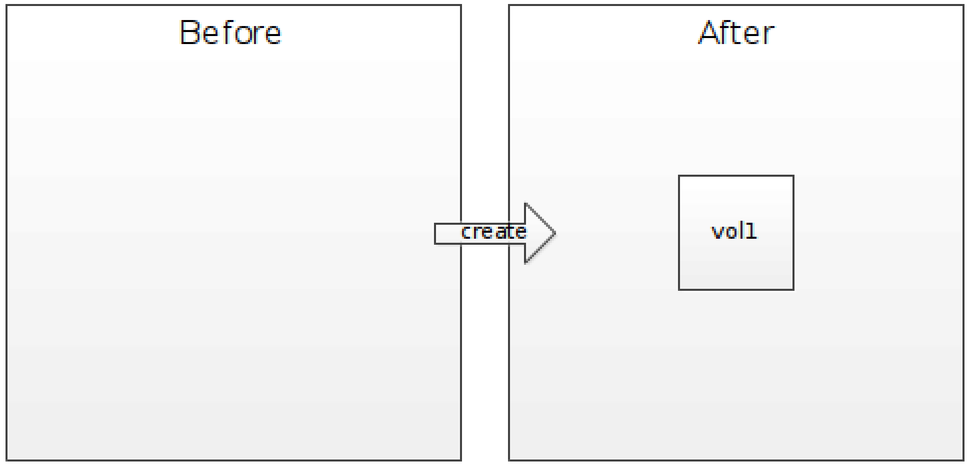
Volume management



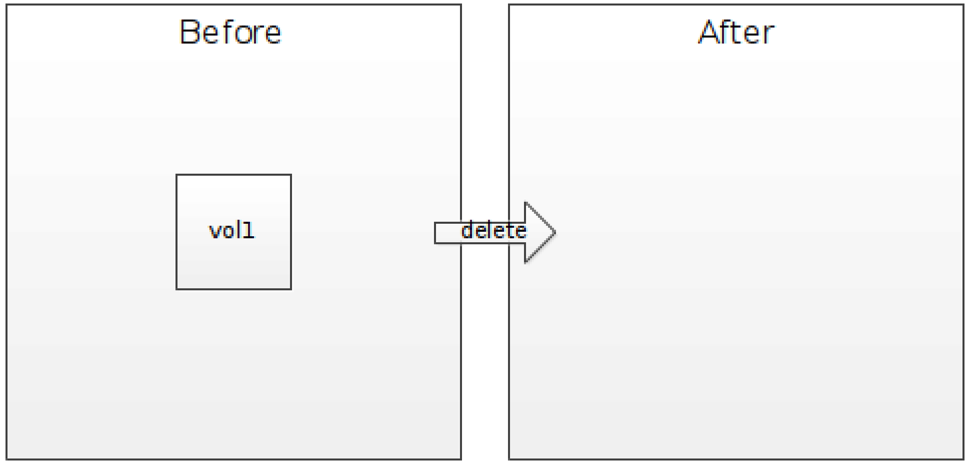
Volumes are the basic service of the StorPool storage system. A volume always have a name and a certain size. It can be read from and written to. It could be attached to hosts as read-only or read-write block device under the `/dev/storpool` directory.

The volume name is a string consisting of one or more of the allowed characters - upper and lower latin letters (`a-z,A-Z`), numbers (`0-9`) and the delimiter dot (`.`), colon (`:`), dash (`-`) and underscore (`_`).

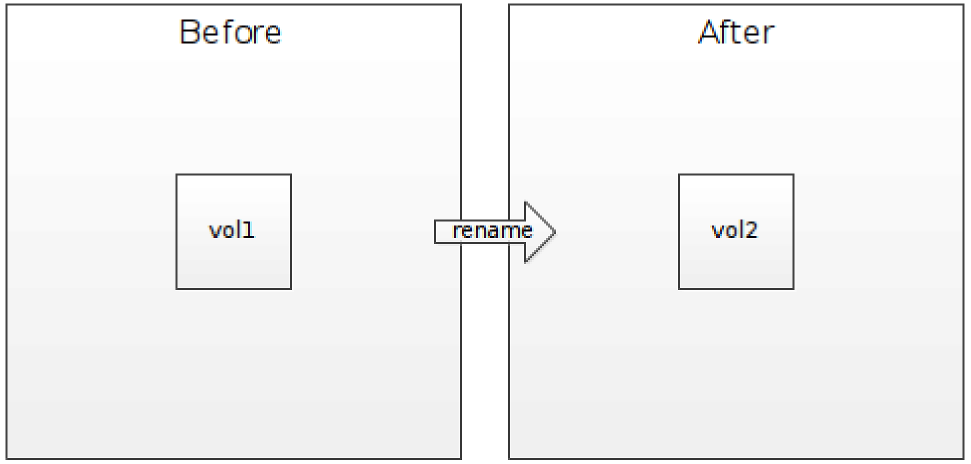
Creating a volume



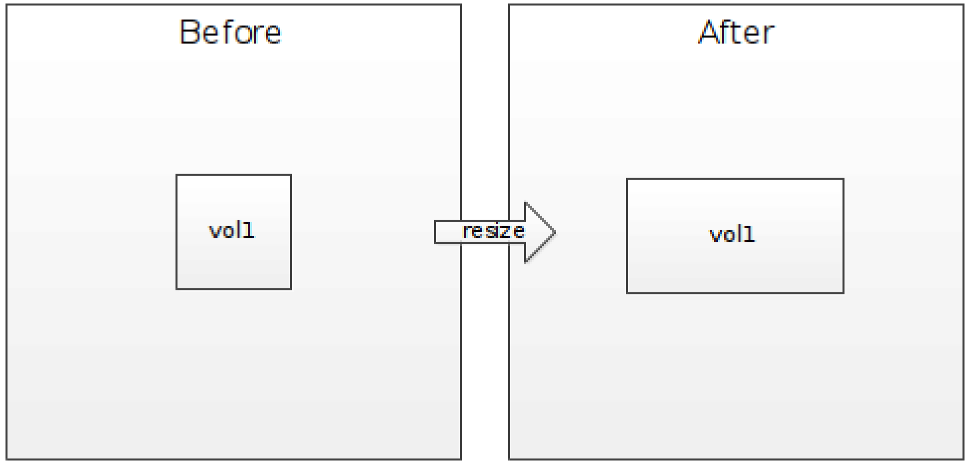
Deleting a volume



Renaming a volume

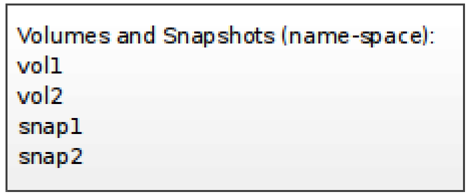


Resizing a volume



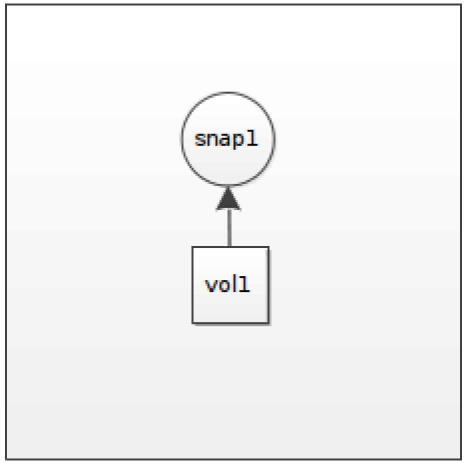
Snapshots

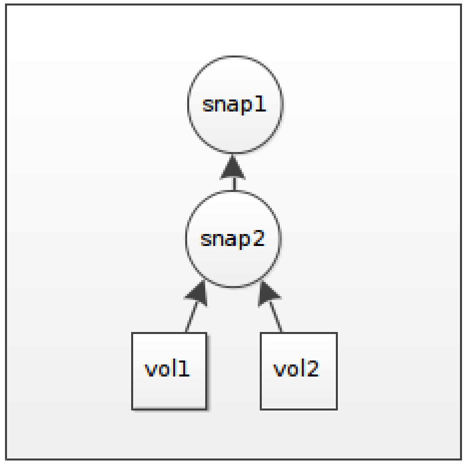
Snapshots are read-only point-in-time images of volumes. They are created once and cannot be changed. They can be attached to hosts as read-only block devices under `/dev/storpool`.



All volumes and snapshots share the same name-space. Names of volumes and snapshots are unique within a StorPool cluster. This diagram illustrates the relationship between a snapshot and a volume. Volume `vol1` is based on snapshot `snap1`. `vol1` contains only the changes since `snap1` was taken. In the common case this is a small amount of data. Arrows indicate a child-parent relationship. Each volume or snapshot may have exactly one parent which it is based upon. Writes to `vol1` are recorded within the volume. Reads from `vol1` may be served by `vol1` or by its parent snapshot - `snap1`, depending on whether `vol1` contains changed data for the read request or not.

Snapshots and volumes are completely independent. Each snapshot may have many children (volumes and snapshots). Volumes cannot have children.



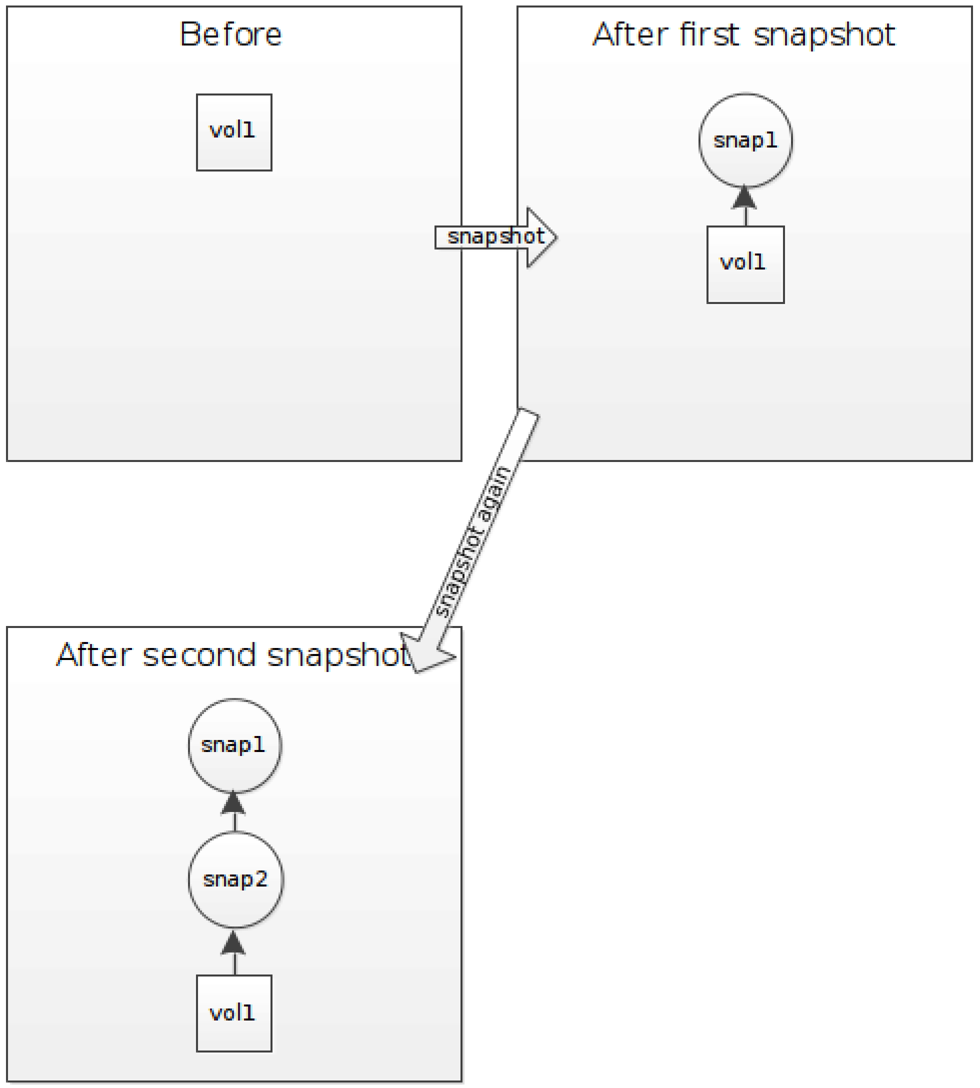


`snap1` contains a full image. `snap2` contains only the changes since `snap1` was taken. `vol1` and `vol2` contain only the changes since `snap2` was taken.

Creating a snapshot of a volume

There is a volume named `vol1`.

After the first snapshot the state of `vol1` is recorded in a new snapshot named `snap1`. `vol1` does not occupy any space now, but will record any new writes which come in after the creation of the snapshot. Reads from `vol1` may fall through to `snap1`.

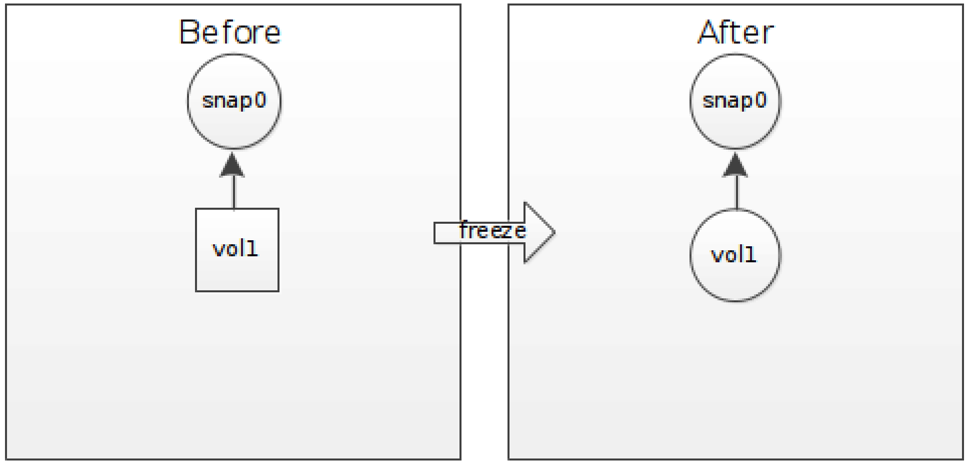


Then the state of `vol1` is recorded in a new snapshot named `snap2`. `snap2` contains the changes between the moment `snap1` was taken and the moment `snap2` was taken. `snap2`'s parent is the original parent of `vol1`.

Converting a volume to a snapshot (freeze)

There is a volume named `vol1`, based on a snapshot `snap1`. `vol1` contains only the changes since `snap0` was taken.

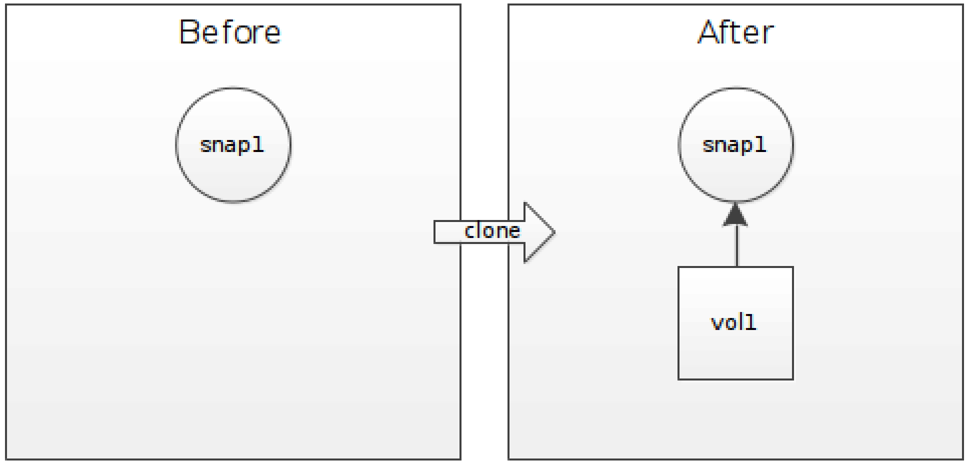
After the freeze operation the state of `vol1` is recorded in a new snapshot with the same name. The snapshot `vol` contains changes between the moment `snap0` was taken and the moment `vol1` was frozen.



Creating a volume based on an existing snapshot (a.k.a. clone)

Before the creation of `vol1` there is a snapshot named `snap1`.

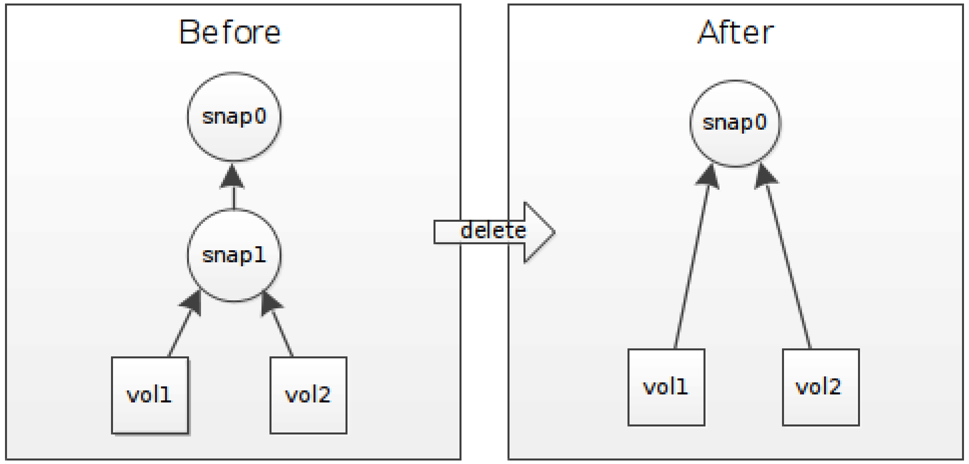
A new volume, named `vol1` is created. `vol1` is based on `snap1`. The newly created volume does not occupy any space initially. Reads from the `vol1` may fall through to `snap1` or to `snap1`'s parents (if any).



Deleting a snapshot

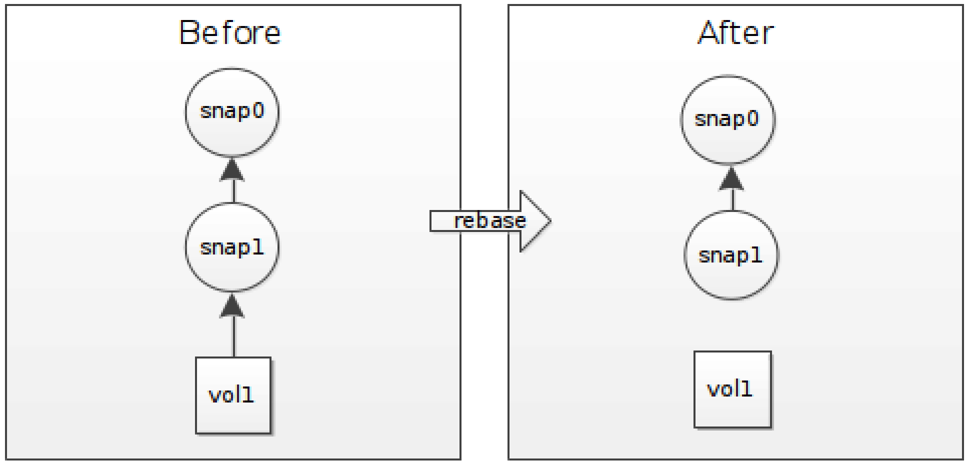
`vol1` and `vol2` are based on `snap1`. `snap1` is based on `snap0`. `snap1` contains the changes between the moment `snap0` was taken and when `snap1` was taken. `vol1` and `vol2` contain the changes since the moment `snap1` was taken.

After the deletion, `vol1` and `vol2` are based on `snap1`'s original parent (if any). In the example they are now based on `snap0`. When deleting a snapshot, the changes contained therein are propagated to all its children (if any). In this case `vol1` and `vol2` now contain all changes since `snap0` was taken, not just since `snap1`. Although StorPool tries to minimize the harmful effects of this, deleting a snapshot with many children may lead to an explosion of disk space usage.



Rebase to null (a.k.a. promote)

`vol1` is based on `snap1`. `snap1` is in turn based on `snap0`. `snap1` contains the changes between the moment `snap0` was taken and when `snap1` was taken. `vol1` contains the changes since the moment `snap1` was taken.

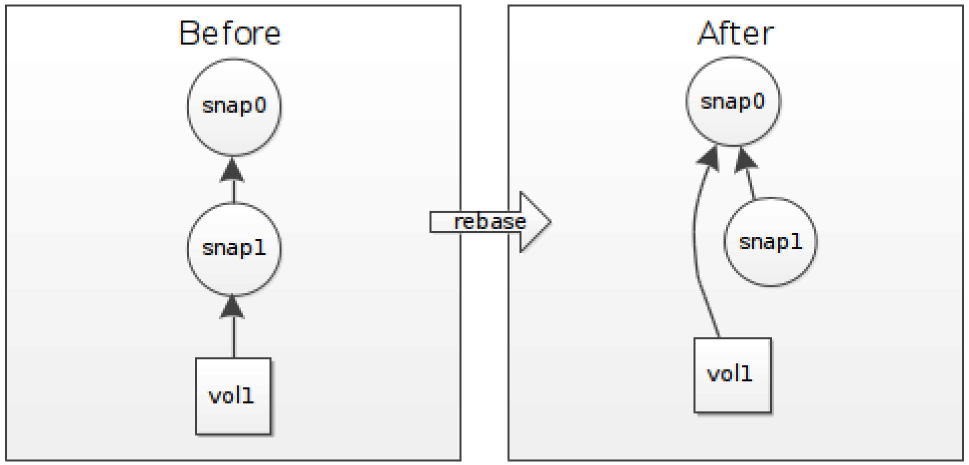


After promotion `vol1` is not based on a snapshot. `vol1` now contains all data, not just the changes since `snap1` was taken. Any relation between `snap1` and `snap0` is unaffected.

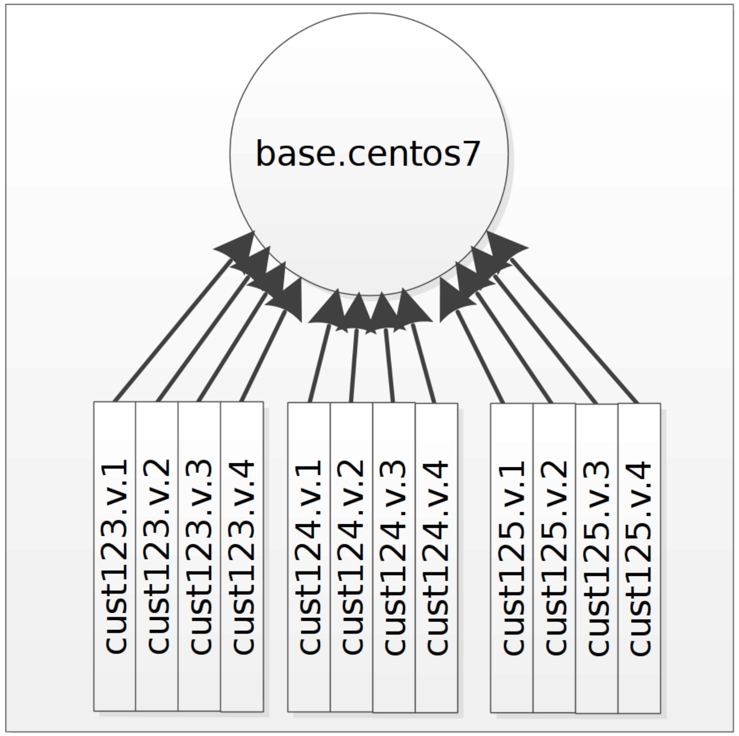
Rebase

`vol1` is based on `snap1`. `snap1` is in turn based on `snap0`. `snap1` contains the changes between the moment `snap0` was taken and when `snap1` was taken. `vol1` contains the changes since the moment `snap1` was taken.

After the rebase operation `vol1` is based on `snap0`. `vol1` now contains all changes since `snap0` was taken, not just since `snap1`. `snap1` is unchanged.

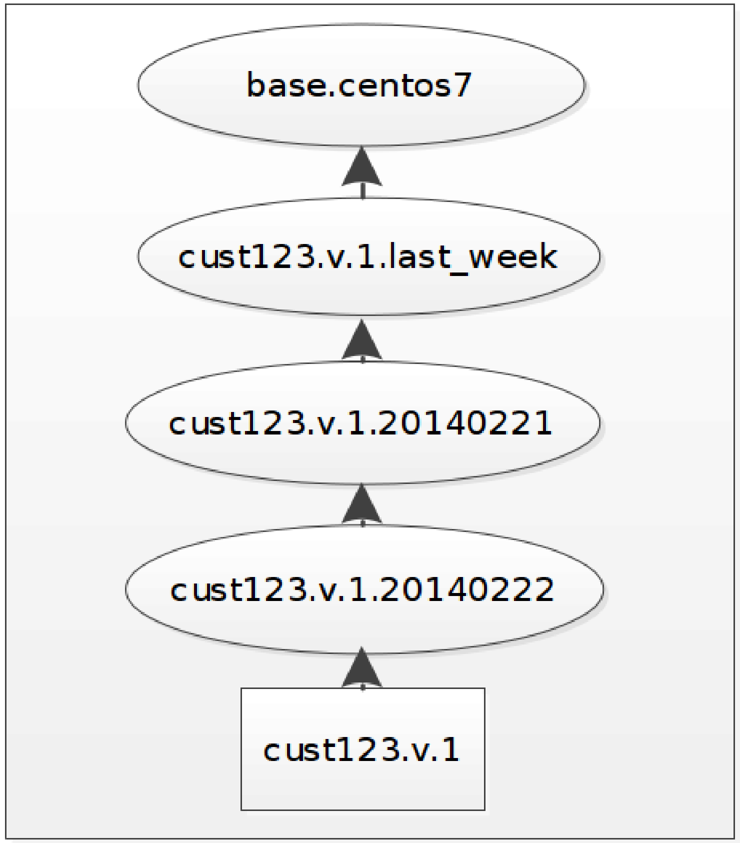


Example use of snapshots



This is a semi-realistic example of how volumes and snapshots may be used. There is a snapshot called `base.centos7`. This snapshot contains a base CentOS 7 VM image, which was prepared carefully by the service provider. There are 3 customers with 4 virtual machines each. All virtual machine images are based on CentOS 7, but may contain custom data, which is unique to each VM.

This example shows another typical use of snapshots - for restore points back in time for this volume. There is one base image for Centos 7, three snapshot restore points and one live volume `cust123.v.1`



StorPool iSCSI support

If StorPool volumes need to be accessed by hosts that cannot run the StorPool client service (e.g. VMware hypervisors), they may be exported using the iSCSI protocol.

A Quick Overview of iSCSI

The iSCSI remote block device access protocol, as implemented by the StorPool iSCSI service, is a client-server protocol allowing clients (referred to as “initiators”) to read and write data to disks (referred to as “targets”) exported by iSCSI servers. The iSCSI servers listen on portals (TCP ports, usually 3260, on specific IP addresses); these portals may be grouped into the so called portal groups to provide fine-grained access control or load balancing for the iSCSI connections.

An iSCSI Setup in a StorPool Cluster

The StorPool implementation of iSCSI provides a way to mark StorPool volumes as accessible to iSCSI initiators, define iSCSI portals where hosts running the StorPool iSCSI service listen for connections from initiators, define portal groups over these portals, and export StorPool volumes (iSCSI targets) to iSCSI initiators in the portal groups. To simplify the configuration of the iSCSI initiators, and also to provide load balancing and failover, each portal group has a floating IP address that is automatically brought up on only a single StorPool service at a given moment; the initiators are configured to connect to this floating address, authenticating if necessary, and are then redirected to the portal of the StorPool service that actually exports the target (volume) that they need to access.

In the simplest setup, there is a single portal group with a floating IP address, there is a single portal for each StorPool host that runs the iSCSI service, all the initiators connect to the floating IP address and are redirected to the correct host. For quality of service or fine-grained access control, more portal groups may be defined and some volumes may be exported via more than one portal group.

A trivial setup may be brought up by the following series of StorPool CLI commands; see the CLI tutorial for more information about the commands themselves:

```
# storpool iscsi config setBaseName iqn.2017-05.com.example:poc-cluster
OK

# storpool iscsi config portalGroup poc create
OK

# storpool iscsi config portalGroup poc create addNet 192.168.42.247/24
OK

# storpool iscsi config portal create portalGroup poc address 192.168.42.246 controller 1
OK

# storpool iscsi config portal create portalGroup poc address 192.168.42.202 controller 3
OK

# storpool iscsi portalGroup list
---------------------------------------
| name | networksCount | portalsCount |
---------------------------------------
| poc  |             1 |            2 |
---------------------------------------

# storpool iscsi portalGroup list portals
--------------------------------------------
| group | address             | controller |
--------------------------------------------
| poc   | 192.168.42.246:3260 |          1 |
| poc   | 192.168.42.202:3260 |          3 |
--------------------------------------------

# storpool iscsi config initiator iqn.2017-05.com.example:poc-cluster:hv1 create
OK

# storpool iscsi config target create tinyvolume
OK

# storpool iscsi config export volume tinyvolume portalGroup poc initiator iqn.2017-05.com.example:poc-cluster:hv1
OK

# storpool iscsi initiator list
-----------------------------------------------------------------------------------------------
| name                                     | username | secret | networksCount | exportsCount |
-----------------------------------------------------------------------------------------------
| iqn.2017-05.com.example:poc-cluster:hv1 |          |        |             0 |            1 |
-----------------------------------------------------------------------------------------------

# storpool iscsi initiator list exports
--------------------------------------------------------------------------------------------------------------------------------------
-----
| name                                             | volume     | currentControllerId | portalGroup | initiator
|
--------------------------------------------------------------------------------------------------------------------------------------
-----
```

```
| iqn.2017-05.com.example:poc-cluster:tinyvolume | tinyvolume |                   1 | poc        | iqn.2017-05.com.example:poc-
cluster:hv1 |
-------------------------------------------------------------------------------------------------------------------------------------
-----
```

Caveats with a Complex iSCSI Architecture

In iSCSI portal definitions, a TCP address/port pair must be unique; only a single portal within the whole cluster may be defined at a single IP address and port. Thus, if the same StorPool iSCSI service should be able to export volumes in more than one portal group, the portals should be placed either on different ports or on different IP addresses (although it is fine for these addresses to be brought up on the same network interface on the host).

The redirecting portal on the floating address of a portal group always listens on port 3260. Similarly to the above, different portal groups must have different floating IP addresses, although they are automatically brought up on the same network interfaces as the actual portals within the groups.

Some iSCSI initiator implementations (e.g. VMware vSphere) may only connect to TCP port 3260 for an iSCSI service. In a more complex setup where a StorPool service on a single host may export volumes in more than one portal group, this might mean that the different portals must reside on different IP addresses, since the port number is the same.

For technical reasons, currently a StorPool volume may only be exported by a single StorPool service (host), even though it may be exported in different portal groups. For this reason, some care should be taken in defining the portal groups so that they may have at least some StorPool services (hosts) in common.

Multi site

Initial setup

With StorPool different locations could be connected through the `storpool_bridge` service running on a public interface in each location for disaster recovery and remote backup purposes.

To configure a bridge on one of the nodes in the cluster, the following parameters would have to be configured in `/etc/storpool.conf`:

```
SP_CLUSTER_NAME=<Human readable name of the cluster>
SP_CLUSTER_ID=<location ID>.<cluster ID>
SP_BRIDGE_HOST=<IP address>
SP_BRIDGE_TEMPLATE=<template>
```

The `SP_CLUSTER_NAME` is optional and could be used as a name for the location/cluster in the remote.

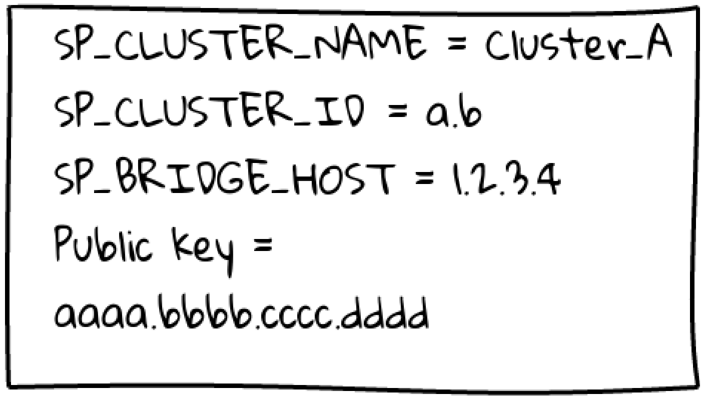
The `SP_CLUSTER_ID` is an unique ID assigned by StorPool for each existing cluster, for example `nmjc.b`. The cluster ID consists of two parts - a location ID (the first part before the `.` - `nmjc`) and a cluster ID (the second part after the `.` - `b`)

The `SP_BRIDGE_HOST` is the public IP address to listen for connections from other bridges. Note that `3749` port should be unblocked in the firewalls between the two locations.

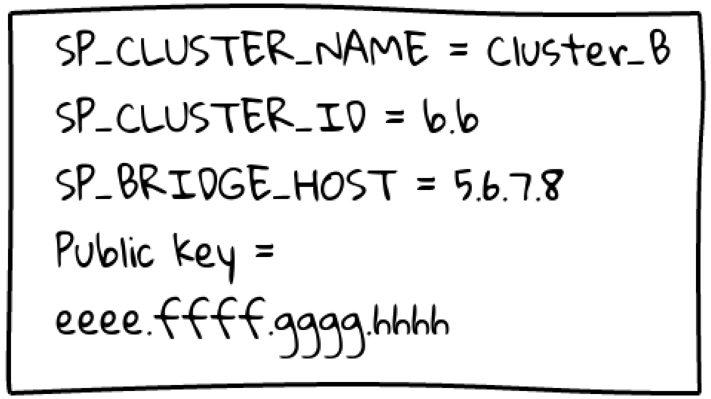
The `SP_BRIDGE_TEMPLATE` is needed to instruct the local bridge which template should be used for incoming snapshots from remote sites.

Connecting two locations

Lets have an example with two clusters named `Cluster_A` and `Cluster_B`. To have these two connected through their bridge services we would have to introduce each of them to the other.



SP_CLUSTER_NAME = Cluster_A
SP_CLUSTER_ID = a.b
SP_BRIDGE_HOST = 1.2.3.4
Public key =
aaaa.bbbb.cccc.dddd



SP_CLUSTER_NAME = Cluster_B
SP_CLUSTER_ID = b.b
SP_BRIDGE_HOST = 5.6.7.8
Public key =
eeee.ffff.gggg.hhhh

Cluster A

The following parameters from `Cluster_B` will be required:

- The `SP_CLUSTER_ID` - `b.b`
- The `SP_BRIDGE_HOST` IP address - `5.6.7.8`
- The public key in `/usr/lib/storpool/bridge/bridge.key.txt` in the remote bridge host in `Cluster_B` - `eeee.ffff.gggg.hhhh`
- The `SP_CLUSTER_NAME` - `Cluster_B`

By using the CLI we could add `Cluster_B`'s location with the following command in `Cluster_A`:

```
user@hostA # storpool location add b Cluster_B
```

Then add the cluster ID (which is `b` in this case) with:

```
user@hostA # storpool cluster add Cluster_B b
```

The last step in `Cluster_A` is to register the `Cluster_B`'s bridge. . The command will look like this:

```
user@hostA # storpool remoteBridge register Cluster_B 5.6.7.8 eeee.ffff.gggg.hhhh
```

Hint

The public key in `/usr/lib/storpool/bridge/bridge.key.txt` will be generated on the first run of the `storpool_bridge` service.

Cluster B

Similarly here the parameters from `Cluster_A` will be required:

- The `SP_CLUSTER_ID` - `a.b`
- The `SP_BRIDGE_HOST` IP address in `Cluster_A` - `1.2.3.4`
- The public key in `/usr/lib/storpool/bridge/bridge.key.txt` in the remote bridge host in `Cluster_A` - `aaaa.bbbb.cccc.dddd`
- The `SP_CLUSTER_NAME` - `Cluster_A`

Then to add and register the `Cluster_A`'s location and bridge into `Cluster_B`, the commands will be:

```
user@hostB # storpool location add a Cluster_A
```

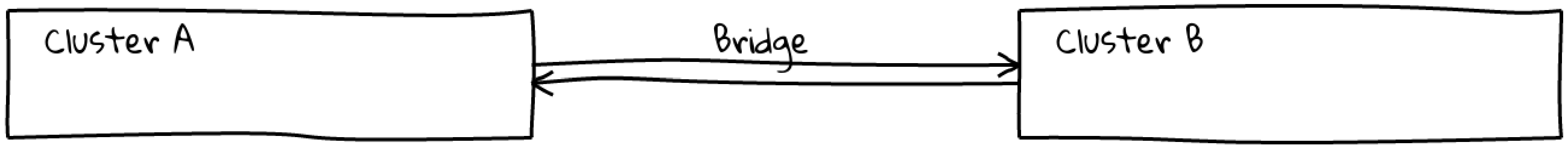
Then adding the cluster ID:

```
user@hostB # storpool cluster add Cluster_A b
```

And lastly register the `Cluster_A`'s bridge locally with:

```
user@hostB # storpool remoteBridge register Cluster_A 1.2.3.4 aaaa.bbbb.cccc.dddd
```

At this point, provided network connectivity is working, the two bridges will be connected.



Exports

When connected a snapshot in one of the clusters could be exported and become visible in the remote location, for example a snapshot called `snap1` could be exported with:

```
user@hostA # storpool snapshot snap1 export Cluster_B
```

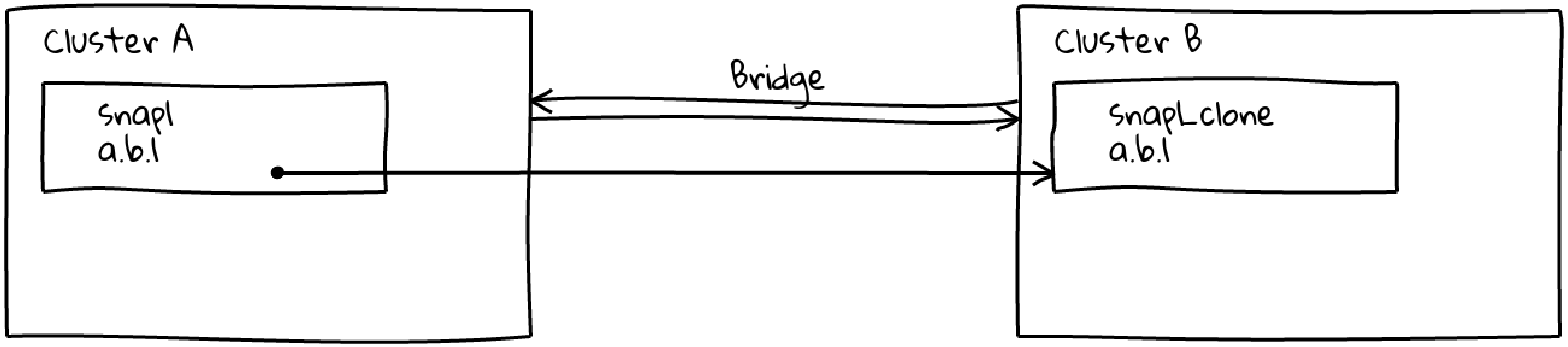
It becomes visible in `Cluster_B` and could be listed with:

```
user@hostB # storpool snapshot list remote
-----------------------------------------------------------------------------------
| location  | remoteId | name     | onVolume | size         | creationTimestamp   |
-----------------------------------------------------------------------------------
| Cluster_A | a.b.1    | snap1    |          | 107374182400 | 2017-02-03 15:18:02 |
----------------------------------------------------------------------------------
```

Remote clones

Any remote snapshot could be cloned locally. For example, to clone a remote snapshot with `globalId` of `a.b.1` locally we could use:

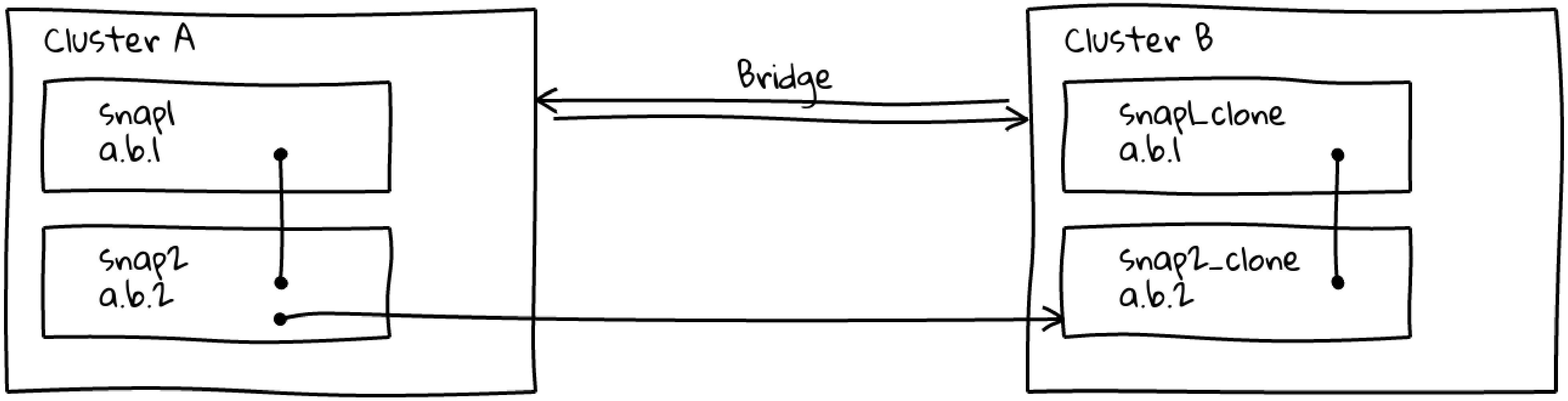
```
user@hostB # storpool snapshot snap1-copy template hybrid remote a.b.1
```



Where the name of the clone of the snapshot in `Cluster_B` will be `snap1-clone` with all parameters from the `hybrid` template. Please note that the name of the snapshot in `Cluster_B` is just for the example, the snapshot could be named `snap1` in both clusters.

The transfer will start immediately. Only written sectors from the snapshot will be transferred between the sites, for example if `snap1` has a size of 100GB, but only 1GB of data was ever written, only this data will be will be transferred between the two sites.

If another snapshot in the remote site is already based on `snap1` and then exported, the actual transfer will again include only the differences between `snap1` and `snap2`, since `snap1` is already cloned in `Cluster_B`.



The `globalId` for this snapshot will be the same for all sites it has been transferred to.

Creating a remote backup on a volume

The volume backup feature is in essence a set of steps that automate the backup procedure for a particular volume.

For example to backup a volume named `volume1` in `Cluster_A` to `Cluster_B` we will use:

```
user@hostA # storpool volume volume1 backup Cluster_B
```

The above command will actually trigger the following set of events:

1. Creates a local temporary snapshot of `volume1` in `Cluster_A` to be transferred to `Cluster_B`
2. Exports the temporary snapshot to `Cluster_B`
3. Instructs `Cluster_B` to initiate the transfer for this snapshot
4. Exports the transferred snapshot in `Cluster_B` to be visible from `Cluster_A`
5. Deletes the local temporary snapshot

For example if a backup operation has been initiated for a volume called `volume1` in `Cluster_A`, the progress of the operation could be followed with:

```
user@hostA # storpool snapshot list exports
---------------------------------------------------
| location  | snapshot     | globalId | backingUp |
---------------------------------------------------
| Cluster_B | volume1@1433 | a.b.p    | true      |
---------------------------------------------------
```

Once this operation completes the temporary snapshot will no longer be visible as an export and a snapshot with the same `globalId` will be visible remotely:

```
user@hostA # snapshot list remote
-------------------------------------------------------------------------------------
| location  | remoteId | name    | onVolume    | size         | creationTimestamp   |
-------------------------------------------------------------------------------------
| Cluster_B | a.b.p    | volume1 | volume1     | 107374182400 | 2017-02-13 16:27:03 |
-------------------------------------------------------------------------------------
```

Restoring a volume from remote snapshot

Restoring the volume to a previous state from a remote snapshot involves the following steps:

1. The first step is to create a local snapshot from the remotely exported one:

   ```
   user@hostA # storpool snapshot volume1-restore template hybrid remote Cluster_B a.b.p
   OK
   ```

There are some bits to explain in the above example - from left to right:

- `volume1-restore` is the name of the local snapshot that will be created.
- The `template hybrid` part instructs StorPool what will be the replication and placement for the locally created snapshot.
- The last part `remote Cluster_B a.b.p` is to instruct StorPool what are the remote location and the `globalId`, in this case `Cluster_B` and `a.b.p`.

If the bridges and the connection between the locations are operational, the transfer will begin immediately.

2. Next create a volume with the newly created snapshot as a parent:

```
user@hostA # storpool volume1-tmp parent volume1-restore
```

Note

If the transfer hasn’t completed yet some operations, e.g. read from a sector not yet completely transferred, might stale until the object being read is transfered. Request forwarding to the remote bridge is considered in our features list and will be available for future releases.

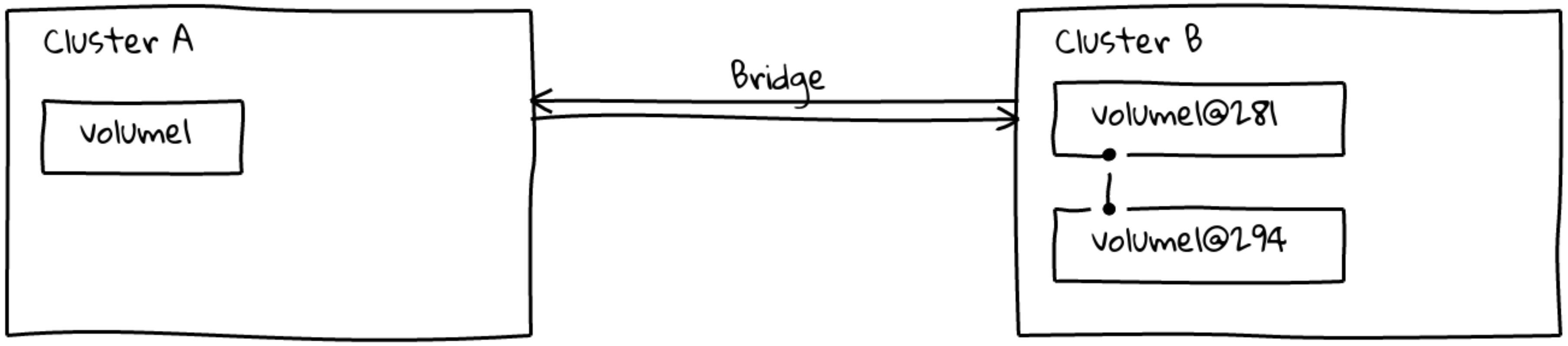
3. Finally the volume clone would have to be attached where it is needed.

The last step might involve reconfiguring a virtual machine to use this volume instead of the presently used one. This is handled differently in different orchestration systems.

Remote deferred deletion

The remote bridge could be registered with remote deferred deletion enabled. This feature will enable a user in `Cluster A` to unexport and set remote snapshots for deferred deletion in `Cluster B`.

An example for the case without deferred deletion enabled - `Cluster_A` and `Cluster_B` are two StorPool clusters connected with a bridge. There is a volume named `volume1` in `Cluster_A`. This volume has two backup snapshots in `Cluster_B` called `volume1@281` and `volume1@294`.



The remote snapshots could be unexported from `Cluster_A` with the `deleteAfter` flag, however it will be silently ignored in `Cluster_B`.

To enable this feature the following steps would have to be completed in the remote bridge for `Cluster_A`:

1. The bridge in `Cluster_A` should be registered with `minimumDeleteDelay` in `Cluster_B`.
2. Enable deferred snapshot deletion in `Cluster_B` (please check Management configuration for more)

This will enable setting up the `deleteAfter` parameter on unexport in `Cluster_B`.

With the above example volume and remote snapshots, if a user in `Cluster_A` unexports the `volume1@294` snapshot and set its `deleteAfter` flag for a week from now, e.g.:

```
user@hostA # storpool snapshot remote Cluster_B a.b.q unexport 7d
OK
```

After the completion of this operation the following events will occur:

- The `volume1@294` snapshot will immediately stop being visible in `Cluster_A`.
- The snapshot will get a `deleteAfter` flag with timestamp a week from this moment.
- A week later the snapshot will be deleted, however **only** if deferred snapshot deletion is turned on.