SUSE Enterprise Storage 7

Man pages

Publication Date: 12 Dec 2024

Contents

1 ceph-volume -- Ceph OSD deployment and inspection tool

ceph-volume

1.1 Synopsis

```
ceph-volume [-h] [--cluster CLUSTER] [--log-level LOG_LEVEL]
                [--log-path LOG_PATH]
```

ceph-volume inventory

```
ceph-volume lvm [ trigger | create | activate | prepare
zap | list | batch]
```

ceph-volume simple [*trigger* | *scan* | *activate*]

1.2 Description

`ceph-volume` is a single purpose command line tool to deploy logical volumes as OSDs, trying to maintain a similar API to `ceph-disk` when preparing, activating, and creating OSDs.

It deviates from `ceph-disk` by not interacting or relying on the udev rules that come installed for Ceph. These rules allow automatic detection of previously setup devices that are in turn fed into `ceph-disk` to activate them.

1.3 Commands

1.3.1 inventory

This subcommand provides information about a host's physical disc inventory and reports metadata about these discs. Among this metadata one can find disc specific data items (like model, size, rotational or solid state) as well as data items specific to ceph using a device, such as if it is available for use with ceph or if logical volumes are present.

Examples:

```
ceph-volume inventory
```

```
ceph-volume inventory /dev/sda
ceph-volume inventory --format json-pretty
```

Optional arguments:

- [-h, --help] show the help message and exit
- **[--format] report format, valid values are** `plain` **(default),**
  `json` and `json-pretty`

1.3.2 lvm

By making use of LVM tags, the `lvm` sub-command is able to store and later re-discover and query devices associated with OSDs so that they can later activated.

Subcommands:

batch Creates OSDs from a list of devices using a `filestore` or `bluestore` (default) setup. It will create all necessary volume groups and logical volumes required to have a working OSD.

Example usage with three devices:

```
ceph-volume lvm batch --bluestore /dev/sda /dev/sdb /dev/sdc
```

Optional arguments:

- [-h, --help] show the help message and exit
- [--bluestore] Use the bluestore objectstore (default)
- [--filestore] Use the filestore objectstore
- [--yes] Skip the report and prompt to continue provisioning
- [--prepare] Only prepare OSDs, do not activate
- [--dmcrypt] Enable encryption for the underlying OSD devices
- [--crush-device-class] Define a CRUSH device class to assign the OSD to
- [--no-systemd] Do not enable or create any systemd units
- [--osds-per-device] Provision more than 1 (the default) OSD per device
- [--report] Report what the potential outcome would be for the current input (requires devices to be passed in)
- [--format] Output format when reporting (used along with --report), can be one of 'pretty' (default) or 'json'
- [--block-db-size] Set (or override) the "bluestore_block_db_size" value, in bytes
- [--journal-size] Override the "osd_journal_size" value, in megabytes

Required positional arguments:

- <DEVICE> Full path to a raw device, like `/dev/sda`. Multiple `<DEVICE>` paths can be passed in.

activate Enables a systemd unit that persists the OSD ID and its UUID (also called `fsid` in Ceph CLI tools), so that at boot time it can understand what OSD is enabled and needs to be mounted.

Usage:

```
ceph-volume lvm activate --bluestore <osd id> <osd fsid>
```

Optional Arguments:

- [-h, --help] show the help message and exit
- [--auto-detect-objectstore] Automatically detect the objectstore by inspecting the OSD
- [--bluestore] bluestore objectstore (default)
- [--filestore] filestore objectstore
- [--all] Activate all OSDs found in the system
- [--no-systemd] Skip creating and enabling systemd units and starting of OSD services

Multiple OSDs can be activated at once by using the (idempotent) `--all` flag:

```
ceph-volume lvm activate --all
```

prepare Prepares a logical volume to be used as an OSD and journal using a `filestore` or `bluestore` (default) setup. It will not create or modify the logical volumes except for adding extra metadata.

Usage:

```
ceph-volume lvm prepare --filestore --data <data lv> --journal <journal device>
```

Optional arguments:

- [-h, --help] show the help message and exit
- [--journal JOURNAL] A logical group name, path to a logical volume, or path to a device
- [--bluestore] Use the bluestore objectstore (default)
- [--block.wal] Path to a bluestore block.wal logical volume or partition
- [--block.db] Path to a bluestore block.db logical volume or partition
- [--filestore] Use the filestore objectstore
- [--dmcrypt] Enable encryption for the underlying OSD devices
- [--osd-id OSD_ID] Reuse an existing OSD id

- [--osd-fsid OSD_FSID] Reuse an existing OSD fsid
- [--crush-device-class] Define a CRUSH device class to assign the OSD to

Required arguments:

- --data A logical group name or a path to a logical volume

For encrypting an OSD, the `--dmcrypt` flag must be added when preparing (also supported in the `create` sub-command).

create Wraps the two-step process to provision a new osd (calling `prepare` first and then `activate`) into a single one. The reason to prefer `prepare` and then `activate` is to gradually introduce new OSDs into a cluster, and avoiding large amounts of data being rebalanced.

The single-call process unifies exactly what `prepare` and `activate` do, with the convenience of doing it all at once. Flags and general usage are equivalent to those of the `prepare` and `activate` subcommand.

trigger This subcommand is not meant to be used directly, and it is used by systemd so that it proxies input to `ceph-volume lvm activate` by parsing the input from systemd, detecting the UUID and ID associated with an OSD.

Usage:

```
ceph-volume lvm trigger <SYSTEMD-DATA>
```

The systemd "data" is expected to be in the format of:

```
<OSD ID>-<OSD UUID>
```

The lvs associated with the OSD need to have been prepared previously, so that all needed tags and metadata exist.

Positional arguments:

- <SYSTEMD_DATA> Data from a systemd unit containing ID and UUID of the OSD.

list List devices or logical volumes associated with Ceph. An association is determined if a device has information relating to an OSD. This is verified by querying LVM's metadata and correlating it with devices.

The lvs associated with the OSD need to have been prepared previously by ceph-volume so that all needed tags and metadata exist.

Usage:

```
ceph-volume lvm list
```

List a particular device, reporting all metadata about it:

```
ceph-volume lvm list /dev/sda1
```

List a logical volume, along with all its metadata (vg is a volume group, and lv the logical volume name):

```
ceph-volume lvm list {vg/lv}
```

Positional arguments:

- <DEVICE> Either in the form of `vg/lv` for logical volumes, `/path/to/sda1` or `/path/to/sda` for regular devices.

zap Zaps the given logical volume or partition. If given a path to a logical volume it must be in the format of vg/lv. Any file systems present on the given lv or partition will be removed and all data will be purged.

However, the lv or partition will be kept intact.

Usage, for logical volumes:

```
ceph-volume lvm zap {vg/lv}
```

Usage, for logical partitions:

```
ceph-volume lvm zap /dev/sdc1
```

For full removal of the device use the `--destroy` flag (allowed for all device types):

```
ceph-volume lvm zap --destroy /dev/sdc1
```

Multiple devices can be removed by specifying the OSD ID and/or the OSD FSID:

```
ceph-volume lvm zap --destroy --osd-id 1
ceph-volume lvm zap --destroy --osd-id 1 --osd-fsid C9605912-8395-4D76-AFC0-7DFDAC315D59
```

Positional arguments:

- <DEVICE> Either in the form of `vg/lv` for logical volumes, `/path/to/sda1` or `/path/to/sda` for regular devices.

1.3.3 simple

Scan legacy OSD directories or data devices that may have been created by ceph-disk, or manually.

Subcommands:

activate Enables a systemd unit that persists the OSD ID and its UUID (also called `fsid` in Ceph CLI tools), so that at boot time it can understand what OSD is enabled and needs to be mounted, while reading information that was previously created and persisted at `/etc/ceph/osd/` in JSON format.

Usage:

```
ceph-volume simple activate --bluestore <osd id> <osd fsid>
```

Optional Arguments:

- [-h, --help] show the help message and exit
- [--bluestore] bluestore objectstore (default)
- [--filestore] filestore objectstore

Note: It requires a matching JSON file with the following format:

```
/etc/ceph/osd/<osd id>-<osd fsid>.json
```

scan Scan a running OSD or data device for an OSD for metadata that can later be used to activate and manage the OSD with ceph-volume. The scan method will create a JSON file with the required information plus anything found in the OSD directory as well.

Optionally, the JSON blob can be sent to stdout for further inspection.

Usage on all running OSDs:

```
ceph-volume simple scan
```

Usage on data devices:

```
ceph-volume simple scan <data device>
```

Running OSD directories:

```
ceph-volume simple scan <path to osd dir>
```

Optional arguments:

- [-h, --help] show the help message and exit
- [--stdout] Send the JSON blob to stdout
- [--force] If the JSON file exists at destination, overwrite it

Optional Positional arguments:

- <DATA DEVICE or OSD DIR> Actual data partition or a path to the running OSD

trigger This subcommand is not meant to be used directly, and it is used by systemd so that it proxies input to `ceph-volume simple activate` by parsing the input from systemd, detecting the UUID and ID associated with an OSD.

Usage:

```
ceph-volume simple trigger <SYSTEMD-DATA>
```

The systemd "data" is expected to be in the format of:

```
<OSD ID>-<OSD UUID>
```

The JSON file associated with the OSD need to have been persisted previously by a scan (or manually), so that all needed metadata can be used.

Positional arguments:

- <SYSTEMD_DATA> Data from a systemd unit containing ID and UUID of the OSD.

1.4 Availability

`ceph-volume` is part of Ceph, a massively scalable, open-source, distributed storage system. Please refer to the documentation at http://docs.ceph.com/ for more information.

1.5 See also

`ceph-osd <ceph-osd>`(8),

2 ganesha-rgw-config -- NFS Ganesha RGW Configuration File

ganesha-rgw-config

2.1 SYNOPSIS

/etc/ganesha/rgw.conf

/etc/ganesha/rgw_bucket.conf

2.2 DESCRIPTION

NFS-Ganesha install two config examples for RGW FSAL:

/etc/ganesha/rgw.conf

/etc/ganesha/rgw_bucket.conf

This file lists RGW specific config options.

2.2.1 EXPORT { }

RGW supports exporting both the buckets and filesystem.

2.2.2 EXPORT { FSAL {} }

Name(string, "RGW")

Name of FSAL should always be RGW.

User_Id(string, no default)

Access_Key(string, no default)

Secret_Access_Key(string, no default)

2.2.3 RGW {}

The following configuration variables customize the startup of the FSAL's radosgw instance.

ceph_conf

optional full-path to the Ceph configuration file (equivalent to passing "-c /path/to/ceph.conf" to any Ceph binary

name

optional instance name (equivalent to passing "--name client.rgw.foohost" to the radosgw binary); the value provided here should be the same as the section name (sans brackets) of the radosgw facility in the Ceph configuration file (which must exist) cluster

optional cluster name (equivalent to passing "--cluster foo" to any Ceph binary); use of a non-default value for cluster name is uncommon, but can be verified by examining the startup options of Ceph binaries

init_args

additional argument strings which will be passed verbatim to the radosgw instance startup process as if they had been given on the radosgw command line provided for customization in uncommon setups

2.3 See also

`ganesha-log-config <ganesha-log-config>`(8) `ganesha-core-config <ganesha-core-config>`(8) `ganesha-export-config <ganesha-export-config>`(8)

3 ganesha-rados-grace -- manipulate the shared grace management database

3.1 SYNOPSIS

ganesha-rados-grace [--cephconf /path/to/ceph.conf] [--ns namespace] [--oid obj_id] [--pool pool_id] [--userid cephuser] dumpstartliftenforcemember [nodeid ...]

3.2 DESCRIPTION

This tool allows the administrator to directly manipulate the database used by the rados_cluster recovery backend. Cluster nodes use that database to indicate their current state in order to coordinate a cluster-wide grace period.

The first argument should be a command to execute against the database. Any remaining arguments represent the nodeids of nodes in the cluster that should be acted upon.

Most commands will just fail if the grace database is not present. The exception to this rule is the **add** command which will create the pool, database and namespace if they do not already exist.

Note that this program does not consult ganesha.conf. If you use non-default values for **ceph_conf**, **userid**, **grace_oid**, **namespace** or **pool** in your RADOS_KV config block, then they will need to passed in via command-line options.

3.3 OPTIONS

--cephconf

Specify the ceph.conf configuration that should be used (default is to use the normal search path to find one)

--ns

Set the RADOS namespace to use within the pool (default is NULL)

--oid

Set the object id of the grace database RADOS object (default is "grace")

--pool

Set the RADOS poolid in which the grace database object resides (default is "nfs-ganesha")

--userid

Set the cephx user ID to use when contacting the cluster (default is NULL)

3.4 COMMANDS

dump

Dump the current status of the grace period database to stdout. This will show the current and recovery epoch serial numbers, as well as a list of hosts currently in the cluster and what flags they have set in their individual records.

add

Add the specified hosts to the cluster. This must be done before the given hosts can take part in the cluster. Attempts to modify the database by cluster hosts that have not yet been added will generally fail. New hosts are added with the enforcing flag set, as they are unable to hand out new state until their own grace period has been lifted.

start

Start a new grace period. This will begin a new grace period in the cluster if one is not already active and set the record for the listed cluster hosts as both needing a grace period and enforcing the grace period. If a grace period is already active, then this is equivalent to **join**.

join

Attempt to join an existing grace period. This works like **start**, but only if there is already an existing grace period in force.

lift

Attempt to lift the current grace period. This will clear the need grace flags for the listed hosts. If there are no more hosts in the cluster that require a grace period, then it will be fully lifted and the cluster will transition to normal operations.

remove

Remove one or more existing hosts from the cluster. This will remove the listed hosts from the grace database, possibly lifting the current grace period if there are no more hosts that need one.

enforce

Set the flag for the given hosts that indicates that they are currently enforcing the grace period; not allowing the acquisition of new state by clients.

noenforce

Clear the enforcing flag for the given hosts, meaning that those hosts are now allowing clients to acquire new state.

member

Test whether the given hosts are members of the cluster. Returns an error if any of the hosts are not present in the grace db omap.

3.5 FLAGS

When the **dump** command is issued, ganesha-rados-grace will display a list of all of the nodes in the grace database, and any flags they have set. The flags are as follows:

E (Enforcing)

The node is currently enforcing the grace period by rejecting requests from clients to acquire new state.

N (Need Grace)

The node currently requires a grace period. Generally, this means that the node has clients that need to perform recovery.

3.6 NODEID ASSIGNMENT

Each running ganesha daemon requires a **nodeid** string that is unique within the cluster. This can be any value as ganesha treats it as an opaque string. By default, the ganesha daemon will use the hostname of the node where it is running.

This may not be suitable when running under certain HA clustering infrastructure, so it's generally recommended to manually assign nodeid values to the hosts in the **RADOS_KV** config block of **ganesha.conf**.

3.7 GANESHA CONFIGURATION

The ganesha daemon will need to be configured with the RecoveryBackend set to **rados_cluster**. If you use a non-default pool, namespace or oid, nodeid then those values will need to be set accordingly in the **RADOS_KV** config block as well.

3.8 STARTING A NEW CLUSTER

First, add the given cluster nodes to the grace database. Assuming that the nodes in our cluster will have nodeids ganesha-1 through ganesha-3:

ganesha-rados-grace add ganesha-1 ganesha-2 ganesha-3

Once this is done, you can start the daemons on each host and they will coordinate to start and lift the grace periods as-needed.

3.9 ADDING NODES TO A RUNNING CLUSTER

After this point, new nodes can then be added to the cluster as needed using the **add** command:

ganesha-rados-grace add ganesha-4

After the node has been added, ganesha.nfsd can then be started. It will then request a new grace period as-needed.

3.10 REMOVING A NODE FROM THE CLUSTER

To remove a node from the cluster, first unmount any clients that have that node mounted (possibly moving them to other servers). Then execute the remove command with the nodeids to be removed from the cluster. For example:

ganesha-rados-grace remove ganesha-4

This will remove the ganesha-4's record from the database, and possibly lift the current grace period if one is active and it was the last one to need it.

4 ganesha-rados-cluster-design -- Clustered RADOS Recovery Backend Design

4.1 Overview

This document aims to explain the theory and design behind the rados_cluster recovery backend, which coordinates grace period enforcement among multiple, independent NFS servers.

In order to understand the clustered recovery backend, it's first necessary to understand how recovery works with a single server:

4.2 Singleton Server Recovery

NFSv4 is a lease-based protocol. Clients set up a relationship to the server and must periodically renew their lease in order to maintain their ephemeral state (open files, locks, delegations or layouts).

When a singleton NFS server is restarted, any ephemeral state is lost. When the server comes comes back online, NFS clients detect that the server has been restarted and will reclaim the ephemeral state that they held at the time of their last contact with the server.

4.3 Singleton Grace Period

In order to ensure that we don't end up with conflicts, clients are barred from acquiring any new state while in the Recovery phase. Only reclaim operations are allowed.

This period of time is called the **grace period**. Most NFS servers have a grace period that lasts around two lease periods, however nfs-ganesha can and will lift the grace period early if it determines that no more clients will be allowed to recover.

Once the grace period ends, the server will move into its Normal operation state. During this period, no more recovery is allowed and new state can be acquired by NFS clients.

4.4 Reboot Epochs

The lifecycle of a singleton NFS server can be considered to be a series of transitions from the Recovery period to Normal operation and back. In the remainder of this document we'll consider such a period to be an **epoch**, and assign each a number beginning with 1.

Visually, we can represent it like this, such that each Normal -> Recovery transition is marked by a change in the epoch value:

```
+-------+-------+-------+---------------+-------+
| State | R | N | R | N | R | R | R | N | R | N |
+-------+-------+-------+---------------+-------+
| Epoch |   1   |   2   |       3       |   4   |
+-------+-------+-------+---------------+-------+
```

Note that it is possible to restart during the grace period (as shown above during epoch 3). That just serves to extend the recovery period and the epoch. A new epoch is only declared during a Recovery -> Normal transition.

4.5 Client Recovery Database

There are some potential edge cases that can occur involving network partitions and multiple reboots. In order to prevent those, the server must maintain a list of clients that hold state on the server at any given time. This list must be maintained on stable storage. If a client sends a request to reclaim some state, then the server must check to make sure it's on that list before allowing the request.

Thus when the server allows reclaim requests it must always gate it against the recovery database from the previous epoch. As clients come in to reclaim, we establish records for them in a new database associated with the current epoch.

The transition from recovery to normal operation should perform an atomic switch of recovery databases. A recovery database only becomes legitimate on a recovery to normal transition. Until that point, the recovery database from the previous epoch is the canonical one.

4.6 Exporting a Clustered Filesystem

Let us consider a set of independent NFS servers, all serving out the same content from a clustered backend filesystem of any flavor. Each NFS server in this case can itself be considered a clustered FS client. This means that the NFS server is really just a proxy for state on the clustered filesystem.

The filesystem must make some guarantees to the NFS server:

- The filesystem ensures that the NFS servers (aka the FS clients) cannot obtain state that conflicts with that of another NFS server.

  This is somewhat obvious and is what we expect from any clustered filesystem outside of any requirements of NFS. If the clustered filesystem can provide this, then we know that conflicting state during normal operations cannot be granted.

  The recovery period has a different set of rules. If an NFS server crashes and is restarted, then we have a window of time when that NFS server does not know what state was held by its clients.

  If the state held by the crashed NFS server is immediately released after the crash, another NFS server could hand out conflicting state before the original NFS client has a chance to recover it.

  This *must* be prevented.

- The filesystem must not release state held by a server during the previous epoch until all servers in the cluster are enforcing the grace period.

  In practical terms, we want the filesystem to provide a way for an NFS server to tell it when it's safe to release state held by a previous instance of itself. The server should do this once it knows that all of its siblings are enforcing the grace period.

  Note that we do not require that all servers restart and allow reclaim at that point. It's sufficient for them to simply begin grace period enforcement as soon as possible once one server needs it.

4.7 Clustered Grace Period Database

At this point the cluster siblings are no longer completely independent, and the grace period has become a cluster-wide property. This means that we must track the current epoch on some sort of shared storage that the servers can all access.

Additionally we must also keep track of whether a cluster-wide grace period is in effect. Any running nodes should all be informed when either of this info changes, so they can take appropriate steps when it occurs.

In the rados_cluster backend, we track these using two epoch values:

C: is the current epoch. This represents the current epoch value

of the cluster

R: is the recovery epoch. This represents the epoch from which

clients are allowed to recover. A non-zero value here means that a cluster-wide grace period is in effect. Setting this to 0 ends that grace period.

In order to decide when to make grace period transitions, each server must also advertise its state to the other nodes. Specifically, each server must be able to determine these two things about each of its siblings:

1. Does this server have clients from the previous epoch that will require recovery? (NEED)
2. Is this server enforcing the grace period by refusing non-reclaim locks? (ENFORCING)

We do this with a pair of flags per sibling (NEED and ENFORCING). Each server typically manages its own flags.

The rados_cluster backend stores all of this information in a single RADOS object that is modified using read/modify/write cycles. Typically we'll read the whole object, modify it, and then attept to write it back. If something changes between the read and write, we redo the read and try it again.

4.8 Clustered Client Recovery Databases

In rados_cluster the client recovery databases are stored as RADOS objects. Each NFS server has its own set of them and they are given names that have the current epoch (C) embedded in it. This ensures that recovery databases are specific to a particular epoch.

In general, it's safe to delete any recovery database that precedes R when R is non-zero, and safe to remove any recovery database except for the current one (the one with C in the name) when the grace period is not in effect ($R == 0$).

4.9 Establishing a New Grace Period

When a server restarts and wants to allow clients to reclaim their state, it must establish a new epoch by incrementing the current epoch to declare a new grace period ($R = C$; $C = C + 1$).

The exception to this rule is when the cluster is already in a grace period. Servers can just join an in-progress grace period instead of establishing a new one if one is already active.

In either case, the server should also set its NEED and ENFORCING flags at the same time.

The other surviving cluster siblings should take steps to begin grace period enforcement as soon as possible. This entails "draining off" any in-progress state morphing operations and then blocking the acquisition of any new state (usually with a return of NFS4ERR_GRACE to clients that attempt it). Again, there is no need for the survivors from the previous epoch to allow recovery here.

The surviving servers must however establish a new client recovery database at this point to ensure that their clients can do recovery in the event of a crash afterward.

Once all of the siblings are enforcing the grace period, the recovering server can then request that the filesystem release the old state, and allow clients to begin reclaiming their state. In the rados_cluster backend driver, we do this by stalling server startup until all hosts in the cluster are enforcing the grace period.

4.10 Lifting the Grace Period

Transitioning from recovery to normal operation really consists of two different steps:

1. the server decides that it no longer requires a grace period, either due to it timing out or there not being any clients that would be allowed to reclaim.
2. the server stops enforcing the grace period and transitions to normal operation

These concepts are often conflated in singleton servers, but in a cluster we must consider them independently.

When a server is finished with its own local recovery period, it should clear its NEED flag. That server should continue enforcing the grace period however until the grace period is fully lifted. The server must not permit reclaims after clearing its NEED flag, however.

If the servers' own NEED flag is the last one set, then it can lift the grace period (by setting $R = 0$). At that point, all servers in the cluster can end grace period enforcement, and communicate that fact to the others by clearing their ENFORCING flags.

5 ganesha-log-config -- NFS Ganesha Log Configuration File

ganesha-log-config

5.1 SYNOPSIS

/etc/ganesha/ganesha.conf

5.2 DESCRIPTION

NFS-Ganesha reads the configuration data from: | /etc/ganesha/ganesha.conf

This file lists NFS-Ganesha Log config options.

These options may be dynamically updated by issuing a SIGHUP to the ganesha.nfsd process.

5.2.1 LOG {}

Default_log_level(token,default EVENT)

```
If this option is NOT set, the fall back log level will be that specified in the -N
 option on the command line if that is set, otherwise the fallback level is EVENT.
```

The log levels are:

NULL, FATAL, MAJ, CRIT, WARN, EVENT, INFO, DEBUG, MID_DEBUG, M_DBG, FULL_DEBUG, F_DBG

RPC_Debug_Flags(uint32, range 0 to UINT32_MAX, default 7)

Debug flags for TIRPC (default 7 matches log level default EVENT).

These flags are only used if the TIRPC component is set to DEBUG

Display_UTC_Timestamp(bool, default false)

Flag to enable displaying UTC date/time in log messages instead of localtime.

5.2.2 LOG { COMPONENTS {} }

Default_log_level(token,default EVENT)

These entries are of the form:

COMPONENT = LEVEL;

The components are:

ALL, LOG, MEMLEAKS, FSAL, NFSPROTO, NFS_V4, EXPORT, FILEHANDLE, DISPATCH, CACHE_INODE, CACHE_INODE_LRU, HASHTABLE, HASHTABLE_CACHE, DUPREQ, INIT, MAIN, IDMAPPER, NFS_READDIR, NFS_V4_LOCK, CONFIG, CLIENTID, SESSIONS, PNFS, RW_LOCK, NLM, RPC, TIRPC, NFS_CB, THREAD, NFS_V4_ACL, STATE, 9P, 9P_DISPATCH, FSAL_UP, DBUS, NFS_MSK

Some synonyms are:

FH = FILEHANDLE HT = HASHTABLE INODE_LRU = CACHE_INODE_LRU INODE = CACHE_INODE DISP = DISPATCH LEAKS = MEMLEAKS NFS3 = NFSPROTO NFS4 = NFS_V4 HT_CACHE = HASHTABLE_CACHE NFS_STARTUP = INIT NFS4_LOCK = NFS_V4_LOCK NFS4_ACL = NFS_V4_ACL 9P_DISP = 9P_DISPATCH

The log levels are:

NULL, FATAL, MAJ, CRIT, WARN, EVENT, INFO, DEBUG, MID_DEBUG, M_DBG, FULL_DEBUG, F_DBG

default none

ALL is a special component that when set, sets all components to the specified value, overriding any that are explicitly set. Note that if ALL is then removed from the config and SIGHUP is issued, all components will revert to what is explicitly set, or Default_Log_Level if that is specified, or the original log level from the -N command line option if that was set, or the code default of EVENT.

TIRPC is a special component that also sets the active RPC_Debug_Flags. If the level for TIRPC is DEBUG or MID_DEBUG, the custom RPC_Debug_Flags set by that parameter will be used, otherwise flags will depend on the level the TIRPC component is set to:

```
NULL or FATAL: 0
```

5.2.3 LOG { FACILITY {} }

name(string, no default)

destination(string, no default, must be supplied)

max_level(token,default FULL_DEBUG)

The log levels are:

```
NULL, FATAL, MAJ, CRIT, WARN, EVENT, INFO, DEBUG, MID_DEBUG, M_DBG, FULL_DEBUG,
 F_DBG
```

headers(token, values [none, component, all] , default all)

enable(token, values [idle, active, default] , default idle)

5.2.4 LOG { FORMAT {} }

date_format(enum,default ganesha)

Possible values:

ganesha, true, local, 8601, ISO-8601, ISO 8601, ISO, syslog, syslog_usec, false, none, user_defined

time_format(enum,default ganesha)

Possible values:

ganesha, true, local, 8601, ISO-8601, ISO 8601, ISO, syslog, syslog_usec, false, none, user_defined

user_date_format(string, no default)

user_time_format(string, no default)

EPOCH(bool, default true)

CLIENTIP(bool, default false)

HOSTNAME(bool, default true)

PROGNAME(bool, default true)

PID(bool, default true)

THREAD_NAME(bool, default true)

FILE_NAME(bool, default true)

LINE_NUM(bool, default true)

FUNCTION_NAME(bool, default true)

COMPONENT(bool, default true)

LEVEL(bool, default true)

5.3 See also

`ganesha-config <ganesha-config>`(8)

6 ganesha-export-config -- NFS Ganesha Export Configuration File

ganesha-export-config

6.1 SYNOPSIS

```
/etc/ganesha/ganesha.conf
```

6.2 DESCRIPTION

NFS-Ganesha obtains configuration data from the configuration file:

```
/etc/ganesha/ganesha.conf
```

This file lists NFS-Ganesha Export block config options.

6.2.1 EXPORT_DEFAULTS {}

These options are all "export permissions" options, and will be repeated in the EXPORT {} and EXPORT { CLIENT {} } blocks.

These options will all be dynamically updateable.

Access_Type(enum, default None)

Possible values:

None, RW, RO, MDONLY, MDONLY_RO

Protocols(enum list, default [3,4])

Possible values:

3, 4, NFS3, NFS4, V3, V4, NFSv3, NFSv4, 9P

Transports(enum list, values [UDP, TCP, RDMA], default [UDP, TCP])

Anonymous_uid(anonid, range INT32_MIN to UINT32_MAX, default -2)

Anonymous_gid(anonid, range INT32_MIN to UINT32_MAX, default -2)

SecType(enum list, default [none, sys])

Possible values:
none, sys, krb5, krb5i, krb5p

PrivilegedPort(bool, default false)

Manage_Gids(bool, default false)

Squash(enum, default root_sqaush)

Possible values:
root, root_squash, rootsquash, rootid, root_id_squash, rootidsquash, all, all_squash, allsquash, all_anomnymous, allanonymous, no_root_squash, none, noidsquash

Each line of defaults above are synonyms

Security_Label(bool, default false)

NFS_Commit(bool, default false)

Delegations(enum, default None)

Possible values:
None, read, write, readwrite, r, w, rw

Attr_Expiration_Time(int32, range -1 to INT32_MAX, default 60)

6.2.2 EXPORT {}

All options below are dynamically changeable with config update unless specified below.

Export_id (required):

An identifier for the export, must be unique and betweem 0 and 65535. If Export_Id 0 is specified, Pseudo must be the root path (/).

Export_id is not dynamic per se, changing it essentially removes the old export and introduces a new export.

Path (required)

The directory in the exported file system this export is rooted on (may be ignored for some FSALs). It need not be unique if Pseudo and/or Tag are specified.

Note that if it is not unique, and the core option mount_path_pseudo is not set true, a v3 mount using the path will ONLY be able to access the first export configured. To access other exports the Tag option would need to be used.

This option is NOT dunamically updateable since it fundamentally changes the export. To change the path exported, export_id should be changed also.

Pseudo (required v4)

This option specifies the position in the Pseudo Filesystem this export occupies if this is an NFS v4 export. It must be unique. By using different Pseudo options, the same Path may be exported multiple times.

This option is used to place the export within the NFS v4 Pseudo Filesystem. This creates a single name space for NFS v4. Clients may mount the root of the Pseudo Filesystem and navigate to exports. Note that the Path and Tag options are not at all visible to NFS v4 clients.

Export id 0 is automatically created to provide the root and any directories necessary to navigate to exports if there is no other export specified with Pseudo = /;. Note that if an export is specified with Pseudo = /;, it need not be export id 0. Specifying such an export with FSAL { name = PSEUDO; } may be used to create a Pseudo Filesystem with specific options. Such an export may also use other FSALs (though directories to reach exports will ONLY be automatically created on FSAL PSEUDO exports).

This option is dynamically changeable and changing it will move the export within the pseudo filesystem. This may be disruptive to clients. Note that if the mount_path_pseudo NFS_CORE_PARAM option is true, the NFSv3 mount path will also change (that should not be disruptive to clients that have the export mounted).

Tag (no default)

This option allows an alternative access for NFS v3 mounts. The option MUST not have a leading /. Clients may not mount subdirectories (i.e. if Tag = foo, the client may not mount foo/baz). By using different Tag options, the same Path may be exported multiple times.

This option is not dynamically updatable.

MaxRead (64*1024*1024)

The maximum read size on this export

MaxWrite (64*1024*1024)

The maximum write size on this export

PrefRead (64*1024*1024)

The preferred read size on this export

PrefWrite (64*1024*1024)

The preferred write size on this export

PrefReaddir (16384)

The preferred readdir size on this export

MaxOffsetWrite (INT64_MAX)

Maximum file offset that may be written Range is 512 to UINT64_MAX

MaxOffsetRead (INT64_MAX)

Maximum file offset that may be read Range is 512 to UINT64_MAX

CLIENT (optional)

See the `EXPORT { CLIENT {} }` block.

FSAL (required)

See the `EXPORT { FSAL {} }` block.

The FSAL for an export can not be changed dynamically. In order to change the FSAL, a new export must be created.

At this time, no FSAL actually supports any updatable options.

6.2.3 EXPORT { CLIENT {} }

Take all the "export permissions" options from EXPORT_DEFAULTS. The client lists are dynamically updateable.

These blocks form an ordered "access control list" for the export. If no client block matches for a particular client, then the permissions in the EXPORT {} block will be used.

Even when a CLIENT block matches a client, if a particular export permission is not explicit in that CLIENT block, the permission specified in the EXPORT block will be used, or if not specified there, from the EXPORT_DEFAULTS block, and if not specified there, the permission will default to the default code value noted in the permission option descriptions above.

Note that when the CLIENT blocks are processed on config reload, a new client access list is constructed and atomically swapped in. This allows adding, removing, and re-arranging clients as well as changing the access for any give client.

Clients(client list, empty)

Client list entries can take on one of the following forms: Match any client:

```
@name       Netgroup name
```

```
x.x.x.x/y   IPv4 network address
wildcarded  If the string contains at least one ? or *
            character (and is not simply "*"), the string is
            used to pattern match host names. Note that [] may
            also be used, but the pattern MUST have at least one
            ? or *
hostname    Match a single client (match is by IP address, all
            addresses returned by getaddrinfo will match, the
            getaddrinfo call is made at config parsing time)
IP address  Match a single client
```

6.2.4 EXPORT { FSAL {} }

NFS-Ganesha supports the following FSALs: **Ceph Gluster GPFS PROXY_V3 PROXY_V4 RGW VFS XFS LUSTRE LIzardFS KVSFS**

Refer to individual FSAL config file for list of config options.

The FSAL blocks generally are less updatable

6.3 DISCUSSION

The EXPORT blocks define the file namespaces that are served by NFS-Ganesha.

In best practice, each underlying filesystem has a single EXPORT defining how that filesystem is to be shared, however, in some cases, it is desirable to sub-divide a filesystem into multiple exports. The danger when this is done is that rogue clients may be able to spoof file handles and access portions of the filesystem not intended to be accessible to that client.

Some FSALs (currently FSAL_VFS, FSAL_GPFS, FSAL_XFS, and FSAL_LUSTRE) are built to support nested filesystems, for example:

```
/export/fs1 /export/fs1/some/path/fs2
```

In this case, it is possible to create a single export that exports both filesystems. There is a lot of complexity of what can be done there.

In discussions of filesystems, btrfs filesystems exported by FSAL_VFS may have subvolumes. Starting in NFS-Ganesha V4.0 FSAL_VFS treats these as separate filesystems that are integrated with all the richness of FSAL_VFS exports.

Another significant FSAL from an export point of view is FSAL_PSEUDO. This is used to build glue exports to build the unified NFSv4 name space. This name space may also be used by NFSv3 by setting the NFS_CORE_PARAM option:

```
mount_path_pseudo = TRUE;
```

If no FSAL_PSEUDO export is explicitly defined, and there is no EXPORT with:

```
Pseudo = "/";
```

NFS-Ganesha will build a FSAL_PSEUDO EXPORT with this Pseudo Path using Export_Id = 0. This automatic EXPORT may be replaced with an explicit EXPORT which need not have Export_Id = 0, it just must have Pseudo = "/" and Protocols = 4.

In building the Pseudo Filesystem, there is a subtle gotcha. Since NFSv4 clients actually moount the root of the Pseudo Filesystem and then use LOOKUP to traverse into the actual directory the sysadmin has mounted from the client, any EXPORTs from "/" to the desired EXPORT MUST have Protocols = 4 specified either in EXPORT_DEFAULTS {}, EXPORT {}, or EXPORT { CLIENT {} }. This is to assure that the client is allowed to traverse each EXPORT.

If Mount_Path_Pseudo = TRUE is being used and an export is desired to be NFSv3 only, Protocols = 3 MUST be specified in the EXPORT {} block. If Protocols is not specified in the EXPORT {} block and is only specified in an EXPORT { CLIENT {} } block, then that export will still be mounted in the Pseudo Filesystem but might not be traversable. Thus if the following two filesystems are exported:

```
/export/fs1 /export/fs1/some/path/fs2
```

And the EXPORTs look something like this:

```
EXPORT { Export_Id = 1; Path = /export/fs1; Peudo = /export/fs1;
```

NFSv4 clients will not be able to access /export/fs1/some/path/fs2. The correct way to accomplish this is:

```
EXPORT { Export_Id = 1; Path = /export/fs1; Peudo = /export/fs1; Protocols=3;
```

Note that an EXPORT { CLIENT {} } block is not necessary if the default export pernmissions are workable.

Note that in order for an EXPORT to be usable with NSFv4 it MUST either have Protcols = 4 specified in the EXPORT block, or the EXPORT block must not have the Protocols option at all such that it defaults to 3,4,9P. Note though that if it is not set and EXPORT_DEFAULTS just has Protocols = 3; then even though the export is mounted in the Pseudo Filesystem, it will not be accessible and the gotcha discussed above may be in play.

6.4 CONFIGURATION RELOAD

In addition to the LOG {} configuration, EXPORT {} config is the main configuration that can be updated while NFS-Ganesha is running by issuing a SIGHUP.

This causes all EXPORT and EXPORT_DEFAULTS blocks to be reloaded. NFS-Ganesha V4.0 and later have some significant improvements to this since it was introduced in NFS-Ganesha V2.4.0. V2.8.0 introduced the ability to remove EXPORTs via SIGHUP configuration reload.

Significantly how things work now is:

On SIGHUP all the EXPORT and EXPORT_DEFAULTS blocks are re-read. There are three conditions that may occur:

```
An export may be added An export may be removed An export may be updated
```

A note on Export_Id and Path. These are the primary options that define an export. If Export_Id is changed, the change is treated as a remove of the old Export_Id and an addition of the new Export_Id. Path can not be changed without also changing Export_Id. The Tag and Pseudo options that also contribute to the uniqueness of an EXPORT may be changed.

Any removed exports are removed from the internal tables and if they are NFSv4 exports, unmounted from the Pseudo Filesystem, which will then be re-built as if those exports had not been present.

Any new exports are added to the internal tables, and if the export is an NFSv4 export, they are mounted into the Pseudo Filesystem.

Any updated exports will be modified with the least disruption possible. If the Pseduo option is changed, the export is unmounted from the Pseduo Filesystem in it's original location, and re-mounted in it's new location. Other options are updated atomically, though serially, so for a short period of time, the options may be mixed between old and new. In most cases this should not cause problems. Notably though, the CLIENT blocks are processed to form a new access control list and that list is atomically swapped with the old list. If the Protocols for an EXPORT are changed to include or remove NFSv4, the Pseduo Filesystem will also be updated.

Note that there is no pause in operations other than a lock being taken when the client list is being swapped out, however the export permissions are applied to an operation once. Notably for NFSv4, this is on a PUTFH or LOOKUP which changes the Current File Handle. As an example, if a write is in progress, having passed the permission check with the previous export permissions, the write will complete without interruption. If the write is part of an NFSv4 COMPOUND, the other operations in that COMPOUND that operate on the same file handle will also complete with the previous export permissions.

An update of EXPORT_DEFAULTS changes the export options atomically. These options are only used for those options not otherwise set in an EXPORT {} or CLIENT {} block and are applied when export permissions are evaluated when a new file handle is encountered.

The FSAL { Name } may not be changed and FSALs offer limited support for changing any options in the FSAL block. Some FSALs may validate and warn if any options in the FSAL block are changed when such a change is not supported.

6.5 SEE ALSO

`ganesha-config <ganesha-config>`(8) `ganesha-rgw-config <ganesha-rgw-config>`(8) `ganesha-vfs-config <ganesha-vfs-config>`(8) `ganesha-lustre-config <ganesha-lustre-config>`(8) `ganesha-xfs-config <ganesha-xfs-config>`(8) `ganesha-gpfs-config <ganesha-gpfs-config>`(8) `ganesha-9p-config <ganesha-9p-config>`(8) `ganesha-proxy-config <ganesha-proxy-config>`(8) `ganesha-ceph-config <ganesha-ceph-config>`(8)

7 ganesha-core-config -- NFS Ganesha Core Configuration File

ganesha-core-config

7.1 SYNOPSIS

/etc/ganesha/ganesha.conf

7.2 DESCRIPTION

NFS-Ganesha reads the configuration data from: | /etc/ganesha/ganesha.conf

This file lists NFS related core config options.

7.2.1 NFS_CORE_PARAM {}

Core parameters:

NFS_Port (uint16, range 0 to UINT16_MAX, default 2049)

Port number used by NFS Protocol.

MNT_Port (uint16, range 0 to UINT16_MAX, default 0)

Port number used by MNT Protocol.

NLM_Port (uint16, range 0 to UINT16_MAX, default 0)

Port number used by NLM Protocol.

Rquota_Port (uint16, range 0 to UINT16_MAX, default 875)

Port number used by Rquota Protocol.

Bind_addr(IPv4 or IPv6 addr, default 0.0.0.0)

The address to which to bind for our listening port.

NFS_Program(uint32, range 1 to INT32_MAX, default 100003)

RPC program number for NFS.

MNT_Program(uint32, range 1 to INT32_MAX, default 100005)

RPC program number for MNT.

NLM_Program(uint32, range 1 to INT32_MAX, default 100021)

RPC program number for NLM.

Drop_IO_Errors(bool, default false)

For NFSv3, whether to drop rather than reply to requests yielding I/O errors. It results in client retry.

Drop_Inval_Errors(bool, default false)

For NFSv3, whether to drop rather than reply to requests yielding invalid argument errors. False by default and settable with Drop_Inval_Errors.

Drop_Delay_Errors(bool, default false)

For NFSv3, whether to drop rather than reply to requests yielding delay errors. False by default and settable with Drop_Delay_Errors.

Plugins_Dir(path, default "/usr/lib64/ganesha")

Path to the directory containing server specific modules

Enable_NFS_Stats(bool, default true)

Whether to collect perfomance statistics. By default the perfomance counting is enabled. Enable_NFS_Stats can be enabled or disabled dynamically via ganesha_stats.

Enable_Fast_Stats(bool, default false)

Whether to use fast stats. If enabled this will skip statistics counters collection for per client and per export.

Enable_FSAL_Stats(bool, default false)

Whether to count and collect FSAL specific performance statistics. Enable_FSAL_Stats can be enabled or disabled dynamically via ganesha_stats

Enable_FULLV3_Stats(bool, default false)

Whether to count and collect "detailed statistics" for NFSv3. Enable_FULLV3_Stats can be enabled or disabled dynamically via ganesha_stats.

Enable_FULLV4_Stats(bool, default false)

Whether to count and collect "detailed statistics" for NFSv4. Enable_FULLV4_Stats can be enabled or disabled dynamically via ganesha_stats.

Enable_CLNT_AllOps_Stats(bool, default false)

Whether to count and collect statistics for all NFS operations requested by NFS clients. Enable_CLNT_AllOps_Stats can be enabled or disabled dynamically via ganesha_stats.

Short_File_Handle(bool, default false)

Whether to use short NFS file handle to accommodate VMware NFS client. Enable this if you have a VMware NFSv3 client. VMware NFSv3 client has a max limit of 56 byte file handles.

Manage_Gids_Expiration(int64, range 0 to 7*24*60*60, default 30*60)

How long the server will trust information it got by calling getgroups() when "Manage_Gids = TRUE" is used in a export entry.

heartbeat_freq(uint32, range 0 to 5000 default 1000)

Frequency of dbus health heartbeat in ms.

Enable_NLM(bool, default true)

Whether to support the Network Lock Manager protocol.

Blocked_Lock_Poller_Interval(int64, range 0 to 180, default 10)

Polling interval for blocked lock polling thread

Protocols(enum list, default [3,4,9P])

Possible values:

3, 4, NFS3, NFS4, V3, V4, NFSv3, NFSv4, 9P

The protocols that Ganesha will listen for. This is a hard limit, as this list determines which sockets are opened. This list can be restricted per export, but cannot be expanded.

NSM_Use_Caller_Name(bool, default false)

Whether to use the supplied name rather than the IP address in NSM operations.

Clustered(bool, default true)

Whether this Ganesha is part of a cluster of Ganeshas. Its vendor specific option.

fsid_device(bool, default false)

Whether to use device major/minor for fsid.

mount_path_pseudo(bool, default false)

Whether to use Pseudo (true) or Path (false) for NFS v3 and 9P mounts.

This option defaults to false for backward compatibility, however, for new setups, it's strongly recommended to be set true since it then means the same server path for the mount is used for both v3 and v4.x.

Note that as an export related option, it seems very desireable to be able to change this on config reload, unfortunately, at the moment it is NOT changeable on config reload. A restart is necessary to change this.

Dbus_Name_Prefix

DBus name prefix. Required if one wants to run multiple ganesha instances on single host. The prefix should be different for every ganesha instance. If this is set, the dbus name will be <prefix>.org.ganesha.nfsd

Enable_UDP(enum, values [False, True, Mount], default True)

Whether to create UDP listeners for Mount, NFS, NLM, RQUOTA, and register them with portmapper. Set to false, e.g., to run as non-root. Set to Mount to enable only Mount UDP listener.

Max_Uid_To_Group_Reqs(uint32, range 0 to INT32_MAX, default 0)

Maximum number of concurrent uid2grp requests that can be made by ganesha. In environments with a slow Directory Service Provider, where users are part of large number of groups, and Manage_Gids is set to True, uid2grp queries made by ganesha can fail if a large number of them are made in parallel. This option throttles the number of concurrent uid2grp queries that ganesha makes.

7.2.2 Parameters controlling TCP DRC behavior:

DRC_Disabled(bool, default false)

Whether to disable the DRC entirely.

TCP_Npart(uint32, range 1 to 20, default 1)

Number of partitions in the tree for the TCP DRC.

DRC_TCP_Size(uint32, range 1 to 32767, default 1024)

Maximum number of requests in a transport's DRC.

DRC_TCP_Cachesz(uint32, range 1 to 255, default 127)

Number of entries in the O(1) front-end cache to a TCP Duplicate Request Cache.

DRC_TCP_Hiwat(uint32, range 1 to 256, default 64)

High water mark for a TCP connection's DRC at which to start retiring entries if we can.

DRC_TCP_Recycle_Npart(uint32, range 1 to 20, default 7)

Number of partitions in the recycle tree that holds per-connection DRCs so they can be used on reconnection (or recycled.)

DRC_TCP_Recycle_Expire_S(uint32, range 0 to 60*60, default 600)

How long to wait (in seconds) before freeing the DRC of a disconnected client.

DRC_TCP_Checksum(bool, default true)

Whether to use a checksum to match requests as well as the XID

7.2.3 Parameters controlling UDP DRC behavior:

DRC_UDP_Npart(uint32, range 1 to 100, default 7)

Number of partitions in the tree for the UDP DRC.

DRC_UDP_Size(uint32, range 512, to 32768, default 32768)

Maximum number of requests in the UDP DRC.

DRC_UDP_Cachesz(uint32, range 1 to 2047, default 599)

Number of entries in the O(1) front-end cache to the UDP Duplicate Request Cache.

DRC_UDP_Hiwat(uint32, range 1 to 32768, default 16384)

High water mark for the UDP DRC at which to start retiring entries if we can

DRC_UDP_Checksum(bool, default true)

Whether to use a checksum to match requests as well as the XID.

7.2.4 Parameters affecting the relation with TIRPC:

RPC_Max_Connections(uint32, range 1 to 10000, default 1024)

Maximum number of connections for TIRPC.

RPC_Idle_Timeout_S(uint32, range 0 to 60*60, default 300)

Idle timeout (seconds). Default to 300 seconds.

MaxRPCSendBufferSize(uint32, range 1 to 1048576*9, default 1048576)

Size of RPC send buffer.

MaxRPCRecvBufferSize(uint32, range 1 to 1048576*9, default 1048576)

Size of RPC receive buffer.

RPC_Ioq_ThrdMax(uint32, range 1 to 1024*128 default 200)

TIRPC ioq max simultaneous io threads

RPC_GSS_Npart(uint32, range 1 to 1021, default 13)

Partitions in GSS ctx cache table

RPC_GSS_Max_Ctx(uint32, range 1 to 1048576, default 16384)

Max GSS contexts in cache. Default 16k

RPC_GSS_Max_Gc(uint32, range 1 to 1048576, default 200)

Max entries to expire in one idle check

7.2.5 Parameters for TCP:

Enable_TCP_keepalive(bool, default true)

Whether tcp sockets should use SO_KEEPALIVE

TCP_KEEPCNT(UINT32, range 0 to 255, default 0 -> use system defaults)

Maximum number of TCP probes before dropping the connection

TCP_KEEPIDLE(UINT32, range 0 to 65535, default 0 -> use system defautls)

Idle time before TCP starts to send keepalive probes

TCP_KEEPINTVL(INT32, range 0 to 65535, default 0 -> use system defaults)

Time between each keepalive probe

7.2.6 NFS_IP_NAME {}

Index_Size(uint32, range 1 to 51, default 17)

Configuration for hash table for NFS Name/IP map.

Expiration_Time(uint32, range 1 to 60*60*24, default 3600)

Expiration time for ip-name mappings.

7.2.7 NFS_KRB5 {}

PrincipalName(string, default "nfs")

KeytabPath(path, default "")

Kerberos keytab.

CCacheDir(path, default "/var/run/ganesha")

The ganesha credential cache.

Active_krb5(bool, default false)

Whether to activate Kerberos 5. Defaults to true (if Kerberos support is compiled in)

7.2.8 NFSv4 {}

Graceless(bool, default false)

Whether to disable the NFSv4 grace period.

Lease_Lifetime(uint32, range 0 to 120, default 60)

The NFSv4 lease lifetime.

Grace_Period(uint32, range 0 to 180, default 90)

The NFS grace period.

DomainName(string, default "localdomain")

Domain to use if we aren't using the nfsidmap.

IdmapConf(path, default "/etc/idmapd.conf")

Path to the idmap configuration file.

UseGetpwnam(bool, default false if using idmap, true otherwise)

Whether to use local password (PAM, on Linux) rather than nfsidmap.

Allow_Numeric_Owners(bool, default true)

Whether to allow bare numeric IDs in NFSv4 owner and group identifiers.

Only_Numeric_Owners(bool, default false)

Whether to ONLY use bare numeric IDs in NFSv4 owner and group identifiers.

Delegations(bool, default false)

Whether to allow delegations.

Deleg_Recall_Retry_Delay(uint32_t, range 0 to 10, default 1)

Delay after which server will retry a recall in case of failures

pnfs_mds(bool, default false)

Whether this a pNFS MDS server. For FSAL Gluster, if this is true, set pnfs_mds in gluster block as well.

pnfs_ds(bool, default false)

Whether this a pNFS DS server.

RecoveryBackend(enum, default "fs")

Use different backend for client info:

- fs : filesystem
- fs_ng: filesystem (better resiliency)
- rados_kv : rados key-value
- rados_ng : rados key-value (better resiliency)
- rados_cluster: clustered rados backend (active/active)

RecoveryRoot(path, default "/var/lib/nfs/ganesha")

Specify the root recovery directory for fs or fs_ng recovery backends.

RecoveryDir(path, default "v4recov")

Specify the recovery directory name for fs or fs_ng recovery backends.

RecoveryOldDir(path, "v4old")

Specify the recovery old directory name for fs recovery backend.

Minor_Versions(enum list, values [0, 1, 2], default [0, 1, 2])

List of supported NFSV4 minor version numbers.

Slot_Table_Size(uint32, range 1 to 1024, default 64)

Size of the NFSv4.1 slot table

Enforce_UTF8_Validation(bool, default false)

Set true to enforce valid UTF-8 for path components and compound tags

Max_Client_Ids(uint32, range 0 to UINT32_MAX, default 0)

Specify a max limit on number of NFS4 ClientIDs supported by the server. With filesystem recovery backend, each ClientID translates to one directory. With certain workloads, this could result in reaching inode limits of the filesystem that /var/lib/nfs/ganesha is part of. The above limit can be used as a guardrail to prevent getting into this situation.

7.2.9 RADOS_KV {}

ceph_conf(string, no default)

Connection to ceph cluster, should be file path for ceph configuration.

userid(path, no default)

User ID to ceph cluster.

namespace(string, default NULL)

RADOS Namespace in which to store objects

pool(string, default "nfs-ganesha")

Pool for client info.

grace_oid(string, default "grace")

Name of the object containing the rados_cluster grace DB

nodeid(string, default result of gethostname())

Unique node identifier within rados_cluster

7.2.10 RADOS_URLS {}

ceph_conf(string, no default)

Connection to ceph cluster, should be file path for ceph configuration.

userid(path, no default)

User ID to ceph cluster.

watch_url(url, no default)

rados:// URL to watch for notifications of config changes. When a notification is received, the server will issue a SIGHUP to itself.

8 ganesha-config -- NFS Ganesha Configuration File

ganesha-config

8.1 SYNOPSIS

/etc/ganesha/ganesha.conf

8.2 DESCRIPTION

NFS-Ganesha obtains configuration data from the configuration file:

```
/etc/ganesha/ganesha.conf
```

The configuration file consists of following parts:

8.2.1 Comments

Empty lines and lines starting with '#' are comments.:

```
# This whole line is a comment
Protocol = TCP; # The rest of this line is a comment
```

8.2.2 Blocks

Related options are grouped together into "blocks". A block is a name followed by parameters enclosed between "{" and "}". A block can contain other sub blocks as well.:

```
EXPORT
```

```
{
    Export_ID = 1;
    FSAL {
        Name = VFS:
    }
}
```

NOTE: FSAL is a sub block. Refer to `BLOCKS` section for list of blocks and options.

8.2.3 Options

Configuration options can be of following types.

1. **Numeric.** Numeric options can be defined in octal, decimal, or hexadecimal. The format follows ANSI C syntax. eg.:

```
mode = 0755;  # This is octal 0755, 493 (decimal)
```

Numeric values can also be negated or logical NOT'd. eg.:

```
anonymousuid = -2; # this is a negative
mask = ~0xff; # Equivalent to 0xffffff00 (for 32 bit integers)
```

2. **Boolean.** Possible values are true, false, yes and no. 1 and 0 are not acceptable.

3. **List.** The option can contain a list of possible applicable values. Protocols = 3, 4, 9p;

8.2.4 Including other config files

Additional files can be referenced in a configuration using '%include' and '%url' directives.:

```
%include <filename>
%url <url, e.g., rados://mypool/mynamespace/myobject>
```

The included file is inserted into the configuration text in place of the %include or %url line. Sub-inclusions may be to any depth. Filenames and URLs may optionally use "":

```
%include base.conf
%include "base.conf"
%url rados://mypool/mynamespace/myobject
%url "rados://mypool/mynamespace/myobject"
%url rados://mypool/myobject
%url "rados://mypool/myobject"
```

In the case of rados:// URLs, providing a two-component URL indicates that the default namespace should be used.

8.2.5 Reloading Config

A config reload can be triggered by sending the ganesha.nfsd process a SIGHUP. Not all config options may be changed with reload, those that can will be documented in the individual sections.

In general, currently dynamic config is supported for EXPORT and LOG options.

8.3 BLOCKS

NFS-Ganesha supports the following blocks:

8.3.1 EXPORT {}

Along with its configuration options, the **EXPORT** block supports **FSAL** and **CLIENT** sub-blocks. See `ganesha-export-config <ganesha-export-config>`(8) for usage of this block and its sub-blocks.

8.3.2 EXPORT_DEFAULTS {}

Refer to `ganesha-export-config <ganesha-export-config>`(8) for usage

8.3.3 MDCACHE {}

Refer to `ganesha-cache-config <ganesha-cache-config>`(8) for usage

8.3.4 NFS_CORE_PARAM {}

Refer to `ganesha-core-config <ganesha-core-config>`(8) for usage

8.3.5 NFS_IP_NAME {}

Refer to `ganesha-core-config <ganesha-core-config>`(8) for usage

8.3.6 NFS_KRB5 {}

Refer to `ganesha-core-config <ganesha-core-config>`(8) for usage

8.3.7 NFSv4 {}

Refer to `ganesha-core-config <ganesha-core-config>`(8) for usage

8.3.8 CEPH {}

Refer to `ganesha-ceph-config <ganesha-ceph-config>`(8) for usage

8.3.9 9P {}

Refer to `ganesha-9p-config <ganesha-9p-config>`(8) for usage

8.3.10 GLUSTER {}

Refer to `ganesha-gluster-config <ganesha-gluster-config>`(8) for usage

8.3.11 GPFS {}

Refer to `ganesha-gpfs-config <ganesha-gpfs-config>`(8) for usage

8.3.12 LOG {}

Refer to `ganesha-log-config <ganesha-log-config>`(8) for usage

1.**LOG { FACILITY {} }** 2.**LOG { FORMAT {} }**

8.3.13 PROXY_V4 {}

Refer to `ganesha-proxy-config <ganesha-proxy-v4-config>`(8) for usage

8.3.14 PROXY_V3 {}

Refer to `ganesha-proxy-v3-config <ganesha-proxy-v3-config>`(8) for usage

8.3.15 RGW {}

Refer to ganesha-rgw-config <ganesha-rgw-config>(8) for usage

8.3.16 VFS {}

Refer to ganesha-vfs-config <ganesha-vfs-config>(8) for usage

8.3.17 XFS {}

Refer to ganesha-xfs-config <ganesha-xfs-config>(8) for usage

8.4 EXAMPLE

Along with "ganesha.conf", for each installed FSAL, a sample config file is added at:

/etc/ganesha

8.5 See also

ganesha-log-config <ganesha-log-config>(8) ganesha-rgw-config <ganesha-rgw-config>(8) ganesha-vfs-config <ganesha-vfs-config>(8) ganesha-lustre-config <ganesha-lustre-config>(8) ganesha-xfs-config <ganesha-xfs-config>(8) ganesha-gpfs-config <ganesha-gpfs-config>(8) ganesha-gluster-config <ganesha-gluster-config>(8) ganesha-9p-config <ganesha-9p-config>(8) ganesha-proxy-config <ganesha-proxy-config>(8) ganesha-proxy-v3-config <ganesha-proxy-v3-config>(8) ganesha-ceph-config <ganesha-ceph-config>(8) ganesha-core-config <ganesha-core-config>(8) ganesha-export-config <ganesha-export-config>(8)

9 ganesha-ceph-config -- NFS Ganesha CEPH Configuration File

ganesha-ceph-config

9.1 SYNOPSIS

/etc/ganesha/ceph.conf

9.2 DESCRIPTION

NFS-Ganesha install config example for CEPH FSAL: | /etc/ganesha/ceph.conf

This file lists CEPH specific config options.

9.2.1 EXPORT { FSAL {} }

Name(string, "Ceph")

Name of FSAL should always be Ceph.

Filesystem(string, no default)

Ceph filesystem name string, for mounting an alternate filesystem within the cluster. The default is to mount the default filesystem in the cluster (usually, the first one created).

User_Id(string, no default)

cephx userid used to open the MDS session. This string is what gets appended to "client.". If not set, the ceph client libs will sort this out based on ceph configuration.

Secret_Access_Key(string, no default)

Key to use for the session (if any). If not set, then it uses the normal search path for cephx keyring files to find a key.

9.2.2 CEPH {}

Ceph_Conf(path, default "")

umask(mode, range 0 to 0777, default 0)

9.3 See also

`ganesha-config <ganesha-config>`(8) `ganesha-log-config <ganesha-log-config>`(8) `ganesha-core-config <ganesha-core-config>`(8) `ganesha-export-config <ganesha-export-config>`(8)

10 ganesha-cache-config -- NFS Ganesha Cache Configuration File

ganesha-cache-config

10.1 SYNOPSIS

/etc/ganesha/ganesha.conf

10.2 DESCRIPTION

NFS-Ganesha reads the configuration data from: | /etc/ganesha/ganesha.conf

This file lists NFS-Ganesha Cache config options. These options used to be configured in the CACHEINODE block. They may still be used in that block, but it is deprecated and will go away. The MDCACHE block takes precidence over the CACHEINODE block.

10.2.1 MDCACHE {}

NParts (uint32, range 1 to 32633, default 7)

Partitions in the Cache_Inode tree.

Cache_Size(uint32, range 1 to UINT32_MAX, default 32633)

Per-partition hash table size.

Use_Getattr_Directory_Invalidation(bool, default false)

Use getattr for directory invalidation.

Dir_Chunk(uint32, range 0 to UINT32_MAX, default 128)

Size of per-directory dirent cache chunks, 0 means directory chunking is not enabled.

Detached_Mult(uint32, range 1 to UINT32_MAX, default 1)

Max number of detached directory entries expressed as a multiple of the chunk size.

Entries_HWMark(uint32, range 1 to UINT32_MAX, default 100000)

The point at which object cache entries will start being reused.

Entries_Release_Size(uint32, range 0 to UINT32_MAX, default 100)

The number of entries attempted to release each time when the handle cache has exceeded the entries high water mark.

Chunks_HWMark(uint32, range 1 to UINT32_MAX, default 100000)

The point at which dirent cache chunks will start being reused.

LRU_Run_Interval(uint32, range 1 to 24 * 3600, default 90)

Base interval in seconds between runs of the LRU cleaner thread.

FD_Limit_Percent(uint32, range 0 to 100, default 99)

The percentage of the system-imposed maximum of file descriptors at which Ganesha will deny requests.

FD_HWMark_Percent(uint32, range 0 to 100, default 90)

The percentage of the system-imposed maximum of file descriptors above which Ganesha will make greater efforts at reaping.

FD_LWMark_Percent(uint32, range 0 to 100, default 50)

The percentage of the system-imposed maximum of file descriptors below which Ganesha will not reap file descriptors.

Reaper_Work(uint32, range 1 to 2000, default 0)

Roughly, the amount of work to do on each pass through the thread under normal conditions. (Ideally, a multiple of the number of lanes.) *This setting is deprecated. Please use Reaper_Work_Per_Lane*

Reaper_Work_Per_Lane(uint32, range 1 to UINT32_MAX, default 50)

This is the numer of handles per lane to scan when performing LRU maintenance. This task is performed by the Reaper thread.

Biggest_Window(uint32, range 1 to 100, default 40)

The largest window (as a percentage of the system-imposed limit on FDs) of work that we will do in extremis.

Required_Progress(uint32, range 1 to 50, default 5)

Percentage of progress toward the high water mark required in in a pass through the thread when in extremis

Futility_Count(uint32, range 1 to 50, default 8)

Number of failures to approach the high watermark before we disable caching, when in extremis.

Dirmap_HWMark(uint32, range 1 to UINT32_MAX, default 10000)

The point at which dirmap entries are reused. This puts a practical limit on the number of simultaneous readdirs that may be in progress on an export for a whence-is-name FSAL (currently only FSAL_RGW)

10.3 See also

`ganesha-config <ganesha-config>`(8)