CernVM-FS Documentation

Release 2.10.0

CernVM Team

Jan 31, 2023

CONTENTS

1 What is CernVM-FS? **1**

2 Contents **3**

2.1 Release Notes for CernVM-FS 2.10.0 3
2.2 Overview 4
2.3 Getting Started 5
2.4 Client Configuration 10
2.5 Setting up a Local Squid Proxy 25
2.6 Creating a Repository (Stratum 0) 26
2.7 CernVM-FS Server Meta Information 47
2.8 Setting up a Replica Server (Stratum 1) 49
2.9 The CernVM-FS Repository Gateway and Publishers 53
2.10 The CernVM-FS Notification System (Experimental) 63
2.11 Container Images and CernVM-FS 64
2.12 Advanced Topics 69
2.13 Appendix 98

3 Contact and Authors **117**

Bibliography **119**

CHAPTER

ONE

WHAT IS CERNVM-FS?

The CernVM-File System (CernVM-FS) provides a scalable, reliable and low- maintenance software distribution service. It was developed to assist High Energy Physics (HEP) collaborations to deploy software on the worldwide-distributed computing infrastructure used to run data processing applications. CernVM-FS is implemented as a POSIX read-only file system in user space (a FUSE module). Files and directories are hosted on standard web servers and mounted in the universal namespace `/cvmfs`. Internally, CernVM-FS uses content-addressable storage and Merkle trees in order to maintain file data and meta-data. CernVM-FS uses outgoing HTTP connections only, thereby it avoids most of the firewall issues of other network file systems. It transfers data and meta-data on demand and verifies data integrity by cryptographic hashes.

By means of aggressive caching and reduction of latency, CernVM-FS focuses specifically on the software use case. Software usually comprises many small files that are frequently opened and read as a whole. Furthermore, the software use case includes frequent look-ups for files in multiple directories when search paths are examined.

CernVM-FS is actively used by small and large HEP collaborations. In many cases, it replaces package managers and shared software areas on cluster file systems as means to distribute the software used to process experiment data.

CHAPTER

TWO

CONTENTS

2.1 Release Notes for CernVM-FS 2.10.0

CernVM-FS 2.10.0 is a feature release containing new features, bug fixes and performance improvements. Highlights are:

- Support for proxy sharding with the new client option `CVMFS_PROXY_SHARD={yes|no}`
- Improved use of the kernel page cache resulting in significant client performance improvements in some scenarios (e.g., #2879)
- Fix for a long-standing open issue regarding the concurrent reading of changing files (CVM-2001)
- Support for unpacking container images through Harbor registry proxies in the container conversion tools
- Various bugfixes and smaller improvements

New platforms: EL 9 (x86_64 and AArch64), AArch64 on Ubuntu

As with previous releases, upgrading should be seamless just by installing the new package from the repository. As usual, we recommend to update only a few worker nodes first and gradually ramp up once the new version proves to work correctly. Please take special care when upgrading a client in NFS mode.

For Stratum 0 servers, all transactions must be closed before upgrading. For Stratum 1 servers, there should be no running snapshots during the upgrade. After the software upgrade, publisher nodes require doing cvmfs_server migrate for each repository.

Note: For gateway deployments, the cvmfs-server package on remote publishers needs to be updated in lockstep with the cvmfs-gateway package due to a backwards compatibility bug.

Note: The machine-readable output of `cvmfs_server tag -x -l` and `cvmfs_server tag -x -i` changed following the removal of the (unused) concept of "channels" from the CernVM-FS repository meta-data. In the output of these two commands, the second-last "channel" column has been removed.

Note: This release introduces a new base package, cvmfs-libs, that is now required by the cvmfs-server package. In future releases, more packages will depend on cvmfs-libs.

2.1.1 Bug fixes

- [client] Gracefully handle open, changing files (CVM-2001)
- [client] Fix race in the startup of the shared cache manager in debug mode (#2910)
- [client] Fix minor memory leak during reload (#2976)
- [client] Fix latency measurement of fuse callbacks (#3025)
- [server] Fix ingestion with a new nested catalog of an empty tarfile (#3055)
- [server] Fix creation of stratum 1 from HTTPS stratum 0 (#2974)
- [server] Avoid double-slash URLs in HTTP HEAD requests (#2989)
- [server] Fix check for open file descriptors before publishing
- [server] Catch unexpected errors in transaction command (#3004)
- [gw] Remove too strict repository name check (#2973)
- Fixes for compiling on macOS > 10.15

2.1.2 Improvements and changes

- [client] Change default visibility of synthetic extended attributes to `rootonly`
- [client] Cancel network fail-over cycle when fuse request is canceled (#2983)
- [client] Add catalog hash to catalog_counters xattr (#2900)
- [server] Remove partial support for "channels" from manifest (#2838)
- [server] Ignore size of directories in `cvmfs_server diff` output
- [server] Add OS version to meta.json (#2863)
- [server] Add /var/log/cvmfs to cvmfs-server rpm including SELinux label (CVM-2070)
- [container tools] Add support for container registry proxies through `DUCC_<REGISTRY_NAME>_PROXY` environment variable (#2893)
- [container tools] Support images with OCI manifest (#2851)

2.2 Overview

The CernVM File System (CernVM-FS) is a read-only file system designed to deliver scientific software onto virtual machines and physical worker nodes in a fast, scalable, and reliable way. Files and file metadata are downloaded on demand and aggressively cached. For the distribution of files, CernVM-FS uses a standard HTTP [BernersLee96] [Fielding99] transport, which allows exploitation of a variety of web caches, including commercial content delivery networks. CernVM-FS ensures data authenticity and integrity over these possibly untrusted caches and connections. The CernVM-FS software comprises client-side software to mount "CernVM-FS repositories" (similar to AFS volumes) as well as a server-side toolkit to create such distributable CernVM-FS repositories.

Fig. 1: A CernVM-FS client provides a virtual file system that loads data only on access. In this example, all releases of a sofware package (such as an HEP experiment framework) are hosted as a CernVM-FS repository on a web server.

The first implementation of CernVM-FS was based on grow-fs [Compostella10] [Thain05], which was originally provided as one of the private file system options available in Parrot. Ever since the design evolved and diverged, taking into account the works on HTTP- Fuse [Suzaki06] and content-delivery networks [Freedman03] [Nygren10] [Tolia03]. Its current implementation provides the following key features:

- Use of the the Fuse kernel module that comes with in-kernel caching of file attributes
- Cache quota management
- Use of a content addressable storage format resulting in immutable files and automatic file de-duplication
- Possibility to split a directory hierarchy into sub catalogs at user-defined levels
- Automatic updates of file catalogs controlled by a time to live stored inside file catalogs
- Digitally signed repositories
- Transparent file compression/decompression and transparent file chunking
- Capability to work in offline mode provided that all required files are cached
- File system data versioning
- File system client hotpatching
- Dynamic expansion of environment variables embedded in symbolic links
- Support for extended attributes, such as file capabilities and SElinux attributes
- Automatic mirror server selection based on geographic proximity
- Automatic load-balancing of proxy servers
- Support for WPAD/PAC auto-configuration of proxy servers
- Efficient replication of repositories
- Possibility to use S3 compatible storage instead of a file system as repository storage

In contrast to general purpose network file systems such as nfs or afs, CernVM-FS is particularly crafted for fast and scalable software distribution. Running and compiling software is a use case general purpose distributed file systems are not optimized for. In contrast to virtual machine images or Docker images, software installed in CernVM-FS does not need to be further packaged. Instead it is distributed and versioned file-by-file. In order to create and update a CernVM-FS repository, a distinguished machine, the so-called *Release Manager Machine*, is used. On such a release manager machine, a CernVM-FS repository is mounted in read/write mode by means of a union file system [Wright04]. The union file system overlays the CernVM-FS read-only mount point by a writable scratch area. The CernVM-FS server tool kit merges changes written to the scratch area into the CernVM-FS repository. Merging and publishing changes can be triggered at user-defined points in time; it is an atomic operation. As such, a CernVM-FS repository is similar to a repository in the sense of a versioning system.

On the client, only data and metadata of the software releases that are actually used are downloaded and cached.

Fig. 2: Opening a file on CernVM-FS. CernVM-FS resolves the name by means of an SQLite catalog. Downloaded files are verified against the cryptographic hash of the corresponding catalog entry. The `stat()` system call can be entirely served from the in-kernel file system buffers.

2.3 Getting Started

This section describes how to install the CernVM-FS client. The CernVM-FS client is supported on x86, x86_64, and ARM architectures running Linux and macOS ≥ 10.14 as well as on Windows Services for Linux (WSL2). There is experimental support for Power and RISC-V architectures.

2.3.1 Overview

The CernVM-FS repositories are located under /cvmfs. Each repository is identified by a *fully qualified repository name*. On Linux, mounting and un-mounting of the CernVM-FS is usually controlled by autofs and automount. That means that starting from the base directory /cvmfs different repositories are mounted automatically just by accessing them. A repository will be automatically unmounted after some automount-defined idle time. On macOS, mounting and un-mounting of the CernVM-FS is done by the user with `sudo mount -t cvmfs /cvmfs/...` commands.

2.3.2 Getting the Software

The CernVM-FS source code and binary packages are available from the CernVM website. However it is recommended to use the available package repositories that are also provided for the supported operating systems.

Scientific Linux/CentOS

To add the CVMFS repository and install CVMFS run

```
sudo yum install https://ecsft.cern.ch/dist/cvmfs/cvmfs-release/cvmfs-release-latest.
↪noarch.rpm
sudo yum install -y cvmfs
```

Debian/Ubuntu

To add the CVMFS repository and install CVMFS run

```
wget https://ecsft.cern.ch/dist/cvmfs/cvmfs-release/cvmfs-release-latest_all.deb
sudo dpkg -i cvmfs-release-latest_all.deb
rm -f cvmfs-release-latest_all.deb
sudo apt-get update
sudo apt-get install cvmfs
```

Fedora

To install the CVMFS package run

```
sudo dnf install https://ecsft.cern.ch/dist/cvmfs/cvmfs-2.10.0/cvmfs-2.10.0-1.fc34.
↪x86_64.rpm https://ecsft.cern.ch/dist/cvmfs/cvmfs-config/cvmfs-config-default-
↪latest.noarch.rpm
```

Docker Container

The CernVM-FS service container can expose the /cvmfs directory tree to the host. Import the container with

```
docker pull cvmfs/service
```

or with

```
curl https://ecsft.cern.ch/dist/cvmfs/cvmfs-2.10.0/cvmfs-service-2.10.0-1.x86_64.
↪docker.tar.gz | docker load
```

Run the container as a system service with

```
docker run -d --rm \
  -e CVMFS_CLIENT_PROFILE=single \
  -e CVMFS_REPOSITORIES=sft.cern.ch,... \
  --cap-add SYS_ADMIN \
  --device /dev/fuse \
  --volume /cvmfs:/cvmfs:shared \
  cvmfs/service:2.8.0-1
```

Use `docker stop` to unmount the /cvmfs tree. Note that if you run multiple nodes (a cluster), you should use `-e CVMFS_HTTP_PROXY` to set a proper site proxy as described further down.

Mac OS X

On Mac OS X, CernVM-FS is based on macFUSE. Note that as of macOS 11 Big Sur, kernel extensions need to be enabled to install macFUSE. Verify that fuse is available with

```
kextstat | grep -i fuse
```

Download the CernVM-FS client package in the terminal in order to avoid signature warnings

```
curl -o ~/Downloads/cvmfs-2.10.0.pkg https://ecsft.cern.ch/dist/cvmfs/cvmfs-2.10.0/
↪cvmfs-2.10.0.pkg
```

Install the CernVM-FS package by opening the .pkg file and reboot. Future releases will provide a signed and notarized package.

Windows / WSL2

Follow the Windows instructions to install the Windows Subsytem for Linux (WSL2). Install any of the Linux distributions and follow the instructions for the distribution in this guide. Whenever you open the Linux distribution, run

```
sudo cvmfs_config wsl2_start
```

to start the CernVM-FS service.

2.3.3 Setting up the Software

Configure AutoFS

For the basic setup, run `cvmfs_config setup`. This ensures that the file /etc/auto.master.d/cvmfs.autofs exists containing `/cvmfs /etc/auto.cvmfs` and that the autofs service is running. Reload the autofs service in order to apply an updated configuration.

NB: For OpenSUSE uncomment the line `#+dir:/etc/auto.master.d/` in the file /etc/auto.master and restart the autofs service.

```
sed -i 's%#+dir:/etc/auto.master.d%+dir:/etc/auto.master.d%' /etc/auto.master
systemctl restart autofs
```

Mac OS X

Due to the lack of autofs on macOS, mount the individual repositories manually like

```
sudo mkdir -p /cvmfs/cvmfs-config.cern.ch
sudo mount -t cvmfs cvmfs-config.cern.ch /cvmfs/cvmfs-config.cern.ch
```

For optimal configuration settings, mount the config repository before any other repositories.

Create default.local

Create `/etc/cvmfs/default.local` and open the file for editing. Select the desired repositories by setting `CVMFS_REPOSITORIES=repo1,repo2,...`. For ATLAS, for instance, set

```
CVMFS_REPOSITORIES=atlas.cern.ch,atlas-condb.cern.ch,grid.cern.ch
```

For an individual workstation or laptop, set

```
CVMFS_CLIENT_PROFILE=single
```

If you setup a cluster of cvmfs nodes, specify the HTTP proxy servers on your site with

```
CVMFS_HTTP_PROXY="http://myproxy1:port|http://myproxy2:port"
```

If you're unsure about the proxy names, set `CVMFS_HTTP_PROXY=DIRECT`. This should *only* be done for a small number of clients (< 5), because large numbers can put a heavy load on the Stratum 1 servers and result, amongst others, in poorer performance for the client. For the syntax of more complex HTTP proxy settings, see *Network Settings*. If there are no HTTP proxies yet at your site, see *Setting up a Local Squid Proxy* for instructions on how to set them up.

Verify the file system

Check if CernVM-FS mounts the specified repositories by `cvmfs_config probe`. If the probe fails, try to restart autofs with `sudo systemctl restart autofs`.

2.3.4 Building from source

The CernVM-FS client is not relocatable and needs to be installed under /usr. On Intel architectures, it needs a gcc $\geq$ 4.2 compiler, on ARMv7 a gcc $\geq$ 4.7 compiler. In order to compile and install from sources, use the following commands

```
cd <source directory>
mkdir build && cd build
cmake ../
make
sudo make install
```

Building with local libraries

For development purposes it might be useful to use locally installed libraries instead of using default system libraries. This can be done by defining variables during the `cmake` configuration step. The correct naming of the variables can be found in `cmake/Modules`. For example, in case of Fuse3 following variables must be set: `FUSE3_INCLUDE_DIR` and `FUSE3_LIBRARY`.

Furthermore, `CMAKE_INSTALL_RPATH_USE_LINK_PATH:BOOL=ON` must be set, otherwise will `sudo make install` strip all linked libraries that point to none-system libraries.

Example code for building CernVM-FS with locally built Fuse3 and including the CernVM-FS unittests and gateway:

```
cmake -DCMAKE_INSTALL_RPATH_USE_LINK_PATH:BOOL=ON \
      -D BUILD_UNITTESTS=ON -D BUILD_GATEWAY=ON \
      -D FUSE3_INCLUDE_DIR=/usr/local/include/ \
      -D FUSE3_LIBRARY=/usr/local/lib/x86_64-linux-gnu/libfuse3.so.3.10.5 \
      ../
make
sudo make install
```

2.3.5 Troubleshooting

- In order to check for common misconfigurations in the base setup, run

```
cvmfs_config chksetup
```

- CernVM-FS gathers its configuration parameter from various configuration files that can overwrite each others settings (default configuration, domain specific configuration, local setup, . . .). To show the effective configuration for *repository*.cern.ch, run

```
cvmfs_config showconfig repository.cern.ch
```

- In order to exclude autofs/automounter as a source of problems, you can try to mount *repository*.cern.ch manually with the following

```
mkdir -p /mnt/cvmfs
mount -t cvmfs repository.cern.ch /mnt/cvmfs
```

- In order to exclude SELinux as a source of problems, you can try mounting after SELinux has been disabled by

```
/usr/sbin/setenforce 0
```

- Once the issue has been identified, ensure that the changes are taken by restarting autofs

```
systemctl restart autofs
```

- If the problem is that a repository can be mounted and unmounted but later cannot be remounted, see *Remounting and Namespaces/Containers*.
- In order to exclude a corrupted local cache as a source of problems, run

```
cvmfs_config wipecache
```

- Finally running with debug logs enabled can provide additional information for bug reports. This can be done by specifying a log file path in the client settings, e.g: `CVMFS_DEBUGLOG=/tmp/cvmfs.log`. See *Debug Logs* for more details.

2.4 Client Configuration

2.4.1 Structure of /etc/cvmfs

The local configuration of CernVM-FS is controlled by several files in `/etc/cvmfs` listed in the table below. For every .conf file except for the files in /etc/cvmfs/default.d you can create a corresponding .local file having the same prefix in order to customize the configuration. The .local file will be sourced after the corresponding .conf file.

In a typical installation, a handful of parameters need to be set in /etc/cvmfs/default.local. Most likely, this is the list of repositories (`CVMFS_REPOSITORIES`), HTTP proxies (see *network settings*), and perhaps the cache directory and the cache quota (see *cache settings*). In a few cases, one might change a parameter for a specific domain or a specific repository, or provide an exclusive cache for a specific repository. For a list of all parameters, see Appendix "*Client parameters*".

The .conf and .local configuration files are key-value pairs in the form `PARAMETER=value`. They are sourced by /bin/sh. Hence, a limited set of shell commands can be used inside these files including comments, `if` clauses, parameter evaluation, and shell math (`$((...))`). Special characters have to be quoted. For instance, instead of `CVMFS_HTTP_PROXY=p1;p2`, write `CVMFS_HTTP_PROXY='p1;p2'` in order to avoid parsing errors. The shell commands in the configuration files can use the `CVMFS_FQRN` parameter, which contains the fully qualified repository names that is being mounted. The current working directory is set to the parent directory of the configuration file at hand.

File	Purpose
`config.sh`	Set of internal helper functions.
`default.conf`	Set of base parameters.
`default.d/$config.conf`	Adjustments to the default.conf configuration, usually installed by a cvmfs-config-... package. Read before default.local.
`domain.d/$domain.conf`	Domain-specific parameters and implementations of the functions in `config.sh`
`config.d/$repository.conf`	Repository-specific parameters and implementations of the functions in `config.sh`
`keys/`	Contains domain-specific sub directories with public keys used to verify the digital signature of file catalogs

The Config Repository

In addition to the local system configuration, a client can configure a dedicated config repository. A config repository is a standard mountable CernVM-FS repository that resembles the directory structure of /etc/cvmfs. It can be used to centrally maintain the public keys and configuration of repositories that should not be distributed with rather static packages, and also to centrally *blacklist* compromised repository keys. Configuration from the config repository is overwritten by the local configuration in case of conflicts; see the comments in /etc/cvmfs/default.conf for the precise ordering of processing the config files. The config repository is set by the `CVMFS_CONFIG_REPOSITORY` parameter. The default configuration rpm cvmfs-config-default sets this parameter to cvmfs-config.cern.ch.

The `CVMFS_CONFIG_REPO_REQUIRED` parameter can be used to force availability of the config repository in order for other repositories to get mounted.

The config repository is a very convenient method for updating the configuration on a lot of CernVM-FS clients at once. This also means that it is very easy to break configurations on a lot of clients at once. Also note that only one config repository may be used per client, and this is a technical limitation that is not expected to change. For these reasons, it makes the most sense to reserve the use of this feature for large groups of sites that share a common infrastructure with trusted people that maintain the configuration repository. In order to facilitate sharing of configurations between the infrastructures, a github repository has been set up. Infrastructure maintainers are invited to collaborate there.

Some large sites that prefer to maintain control over their own client configurations publish their own config repository but have automated processes to compare it to a repository from a larger infrastructure. They then quickly update their own config repository with whatever changes have been made to the infrastructure's config repository.

Exchanges of configurations between limited numbers of sites that are also depending separately on a configuration repository is encouraged to be done by making rpm and/or dpkg packages and distributing them through cvmfs-contrib package repositories. Keeping configurations up to date through packages is less convenient than the configuration repository but better than manually maintaining configuration files.

2.4.2 Mounting

Mounting of CernVM-FS repositories is typically handled by autofs. Just by accessing a repository directory under /cvmfs (/cvmfs/atlas.cern.ch), autofs will take care of mounting. autofs will also automatically unmount a repository if it is not used for a while.

Instead of using autofs, CernVM-FS repositories can be mounted manually with the system's `mount` command. In order to do so, use the `cvmfs` file system type, like

```
mount -t cvmfs atlas.cern.ch /cvmfs/atlas.cern.ch
```

Likewise, CernVM-FS repositories can be mounted through entries in /etc/fstab. A sample entry in /etc/fstab:

```
atlas.cern.ch /mnt/test cvmfs defaults,_netdev,nodev 0 0
```

Every mount point corresponds to a CernVM-FS process. Using autofs or the system's mount command, every repository can only be mounted once. Otherwise multiple CernVM-FS processes would collide in the same cache location. If a repository is needed under several paths, use a *bind mount* or use a *private file system mount point*.

If a configuration repository is required to mount other repositories, it will need to be mounted first. Since /etc/fstab mounts are done in parallel at boot time, the order in /etc/fstab is not sufficient to make sure that happens. On systemd-based systems this can be done by adding the option `x-systemd.requires-mounts-for=<configrepo>` on all the other mounts. For example:

```
config-egi.egi.eu /cvmfs/config-egi.egi.eu cvmfs defaults,_netdev,nodev 0 0
cms.cern.ch /cvmfs/cms.cern.ch cvmfs defaults,_netdev,nodev,x-systemd.requires-mounts-
↪for=/cvmfs/config-egi.egi.eu 0 0
```

Private Mount Points

In contrast to the system's `mount` command which requires root privileges, CernVM-FS can also be mounted like other Fuse file systems by normal users. In this case, CernVM-FS uses parameters from one or several user-provided config files instead of using the files under /etc/cvmfs. CernVM-FS private mount points do not appear as `cvmfs2` file systems but as `fuse` file systems. The `cvmfs_config` and `cvmfs_talk` commands ignore privately mounted CernVM-FS repositories. On an interactive machine, private mount points are for instance unaffected by an administrator unmounting all system's CernVM-FS mount points by `cvmfs_config umount`.

In order to mount CernVM-FS privately, use the `cvmfs2` command like

```
cvmfs2 -o config=myparams.conf atlas.cern.ch /home/user/myatlas
```

A minimal sample myparams.conf file could look like this:

```
CVMFS_CACHE_BASE=/home/user/mycache
CVMFS_RELOAD_SOCKETS=/home/user/mycache
CVMFS_USYSLOG=/home/user/cvmfs.log
CVMFS_CLAIM_OWNERSHIP=yes
CVMFS_SERVER_URL=http://cvmfs-stratum-one.cern.ch/cvmfs/atlas.cern.ch
CVMFS_KEYS_DIR=/etc/cvmfs/keys/cern.ch
CVMFS_HTTP_PROXY=DIRECT
```

Make sure to use absolute path names for the mount point and for the cache directory. Use `fusermount -u` in order to unmount a privately mounted CernVM-FS repository.

The private mount points can also be used to use the CernVM-FS Fuse module in case it has not been installed under /usr and /etc. If the public keys are not installed under /etc/cvmfs/keys, the directory of the keys needs to be specified in the config file by `CVMFS_KEYS_DIR=<directory>`. If the libcvmfs_fuse.so resp. libcvmfs_fuse3.so library is not installed in one of the standard search paths, the `CVMFS_LIBRARY_PATH` variable has to be set accordingly for the `cvmfs2` command.

The easiest way to make use of CernVM-FS private mount points is with the `cvmfsexec` package. Read about that in the Security *Running the client as a normal user* section.

Pre-mounting

In usual deployments, the `fusermount` utility from the system fuse package takes care of mounting a repository before handing of control to the CernVM-FS client. The `fusermount` utility is a suid binary because on older kernels and outside user name spaces, mounting is a privileged operation.

As of libfuse3, the task of mounting /dev/fuse can be performed by any utility. This functionality has been added, for instance, to Singularity 3.4.

An executable that pre-mounts /dev/fuse has to call the `mount()` system call in order to open a file descriptor. The file descriptor number is than passed as command line parameter to the CernVM-FS client. A working code example is available in the CernVM-FS tests.

In order to use the pre-mount functionality in Singularity, create a container that has the `cvmfs` package and configuration installed in it, and also the corresponding `cvmfs-fuse3` package. Bind-mount scratch space at `/var/run/cvmfs` and cache space at `/var/lib/cvmfs`. For each desired repository, add a `--fusemount` option with `container:cvmfs2` followed by the repository name and mountpoint, separated by whitespace. First mount the configuration repository if required. For example:

```
CONFIGREPO=config-osg.opensciencegrid.org
singularity exec -S /var/run/cvmfs -B $HOME/cvmfs_cache:/var/lib/cvmfs \
    --fusemount "container:cvmfs2 $CONFIGREPO /cvmfs/$CONFIGREPO" \
    --fusemount "container:cvmfs2 cms.cern.ch /cvmfs/cms.cern.ch" \
    docker://davedykstra/cvmfs-fuse3 bash
```

The `singcvmfs` command in the `cvmfsexec` package makes use of fuse pre-mounting. Read more about that package in the Security *Running the client as a normal user* section.

Remounting and Namespaces/Containers

It is common practice to use CernVM-FS from within containers, especially with Singularity. This sometimes results in a problem because the Linux kernel does not prevent unmounting a CernVM-FS repository if the only processes accessing it are in mount namespaces, even though the fuse processes managing the repository need to keep running until all processes using the repository exit. The problem in that case is that the repository cannot be remounted as long as the background processes keep running. This can be easily reproduced by interactively running a Singularity container out of CernVM-FS (without the `-p` option), running `sleep` in the background, and exiting Singularity. The repository can then be unmounted, but it cannot be remounted until the `sleep` process dies.

When this happens, `cvmfs_config fuser <repo>` can be used to identify all the processes using `<repo>`. The system administrator can then contact the owners of the processess to ask to change the application behavior to avoid this situation (for example by using Singularity `-p`), and the processes can be killed to enable the repository to be remounted.

Docker Containers

There are two options to mount CernVM-FS in docker containers. The first option is to bind mount a mounted repository as a volume into the container. This has the advantage that the CernVM-FS cache is shared among multiple containers. The second option is to mount a repository inside a container, which requires a *privileged* container.

Volume Driver

There is an external package that provides a Docker Volume Driver for CernVM-FS. This package provides management of repositories in Docker and Kubernetes. It provides a convenient interface to handle CernVM-FS volume definitions.

Bind mount from the host

On Docker >= 1.10, the autofs managed area /cvmfs can be directly mounted into the container as a shared mount point like

```
docker run -it -v /cvmfs:/cvmfs:shared centos /bin/bash
```

In order to bind mount an individual repository from the host, turn off autofs on the host and mount the repository manually, like:

```
service autofs stop  # systemd: systemctl stop autofs
chkconfig autofs off  # systemd: systemctl disable autofs
mkdir -p /cvmfs/sft.cern.ch
mount -t cvmfs sft.cern.ch /cvmfs/sft.cern.ch
```

Start the docker container with the `-v` option to mount the CernVM-FS repository inside, like

```
docker run -it -v /cvmfs/sft.cern.ch:/cvmfs/sft.cern.ch centos /bin/bash
```

The `-v` option can be used multiple times with different repositories.

Mount inside a container

In order to use `mount` inside a container, the container must be started in privileged mode, like

```
docker run --privileged -i -t centos /bin/bash
```

In such a container, CernVM-FS can be installed and used the usual way provided that autofs is turned off.

Parrot Connector to CernVM-FS

In case Fuse cannot be be installed, the parrot toolkit provides a means to "mount" CernVM-FS on Linux in pure user space. Parrot sandboxes an application in a similar way gdb sandboxes an application. But instead of debugging the application, parrot transparently rewrites file system calls and can effectively provide /cvmfs to an application. We recommend to use the latest precompiled parrot, which has CernVM-FS support built-in.

In order to sandbox a command `<CMD>` with options `<OPTIONS>` in parrot, use

```
export PARROT_ALLOW_SWITCHING_CVMFS_REPOSITORIES=yes
export PARROT_CVMFS_REPO="<default-repositories>"
export HTTP_PROXY='<SITE HTTP PROXY>'  # or 'DIRECT;' if not on a cluster or grid site
parrot_run <PARROT_OPTIONS> <CMD> <OPTIONS>
```

Repositories that are not available by default from the builtin `<default-repositories>` list can be explicitly added to `PARROT_CVMFS_REPO`. The repository name, a stratum 1 URL, and the public key of the repository need to be provided. For instance, in order to add alice-ocdb.cern.ch and ilc.desy.de to the list of repositories, one can write

```
export CERN_S1="http://cvmfs-stratum-one.cern.ch/cvmfs"
export DESY_S1="http://grid-cvmfs-one.desy.de:8000/cvmfs"
export PARROT_CVMFS_REPO="<default-repositories> \
  alice-ocdb.cern.ch:url=${CERN_S1}/alice-ocdb.cern.ch,pubkey=<PATH/key.pub> \
  ilc.desy.de:url=${DESY_S1}/ilc.desy.de,pubkey=<PATH/key.pub>"
```

given that the repository public keys are in the provided paths.

By default, parrot uses a shared CernVM-FS cache for all parrot instances of the same user stored under a temporary directory that is derived from the user id. In order to place the CernVM-FS cache into a different directory, use

```
export PARROT_CVMFS_ALIEN_CACHE=</path/to/cache>
```

In order to share this directory among multiple users, the users have to belong to the same UNIX group.

2.4.3 Network Settings

CernVM-FS uses HTTP for the data transfer. Repository data can be replicated to multiple web servers and cached by standard web proxies such as Squid [Guerrero99]. In a typical setup, repositories are replicated to a handful of web servers in different locations. These replicas form the CernVM-FS Stratum 1 service, whereas the replication source server is the CernVM-FS Stratum 0 server. In every cluster of client machines, there should be two or more web proxy servers that CernVM-FS can use (see *Setting up a Local Squid Proxy*). These site-local web proxies reduce the network latency for the CernVM-FS clients and they reduce the load for the Stratum 1 service. CernVM-FS supports WPAD/PAC proxy auto configuration [Gauthier99], choosing a random proxy for load-balancing, and automatic fail-over to other hosts and proxies in case of network errors. Roaming clients can connect directly to the Stratum 1 service.

IP Protocol Version

CernVM-FS can use both IPv4 and IPv6. For dual-stack stratum 1 hosts it will use the system default settings when connecting directly to the host. When connecting to a proxy, by default it will try on the IPv4 address unless the proxy only has IPv6 addresses configured. The `CVMFS_IPFAMILY_PREFER=[4|6]` parameter can be used to select the preferred IP protocol for dual-stack proxies.

Stratum 1 List

To specify the Stratum 1 servers, set `CVMFS_SERVER_URL` to a semicolon-separated list of known replica servers (enclose in quotes). The so defined URLs are organized as a ring buffer. Whenever download of files fails from a server, CernVM-FS automatically switches to the next mirror server. For repositories under the cern.ch domain, the Stratum 1 servers are specified in /etc/cvmfs/domain.d/cern.ch.conf.

It is recommended to adjust the order of Stratum 1 servers so that the closest servers are used with priority. This can be done automatically by *using geographic ordering*. Alternatively, for roaming clients (clients not using a proxy server), the Stratum 1 servers can be automatically sorted according to round trip time by `cvmfs_talk host probe` (see *Auxiliary Tools*). Otherwise, the proxy server would invalidate round trip time measurement.

The special sequence `@fqrn@` in the `CVMFS_SERVER_URL` string is replaced by fully qualified repository name (atlas.cern.cn, cms.cern.ch, . . .). That allows to use the same parameter for many repositories hosted under the same domain. For instance, http://cvmfs-stratum-one.cern.ch/cvmfs/@fqrn@ can resolve to http://cvmfs-stratum-one.cern.ch/cvmfs/atlas.cern.ch, http://cvmfs-stratum-one.cern.ch/cvmfs/cms.cern.ch, and so on depending on the repository that is being mounted. The same works for the sequence `@org@` which is replaced by the unqualified repository name (atlas, cms, . . .).

Proxy Lists

CernVM-FS uses a dedicated HTTP proxy configuration, independent from system-wide settings. Instead of a single proxy, CernVM-FS uses a *chain of load-balanced proxy groups*. The CernVM-FS proxies are set by the `CVMFS_HTTP_PROXY` parameter.

Proxy groups are used for load-balancing among several proxies of equal priority. Starting with the first group, one proxy within a group is selected at random. By default, this randomly selected proxy will used for all requests. If *proxy sharding* is enabled, then the proxy is instead selected on a per-request basis to distribute the requests across all proxies within the current group.

If a proxy fails, CernVM-FS automatically switches to another proxy from the current group. If all proxies in a group have failed, CernVM-FS switches to the next proxy group. After probing the last proxy group in the chain, the first is probed again. To avoid endless loops, for each file download the number of switches is limited by the total number of proxies.

Proxies within the same group are separated by a pipe character `|`, while groups are separated from each other by a semicolon character `;`[1]. Note that it is possible for a proxy group to consist of only one proxy. In the case of proxies that use a DNS *round-robin* entry, wherein a single host name resolves to multiple IP addresses, CVMFS automatically internally transforms the name into a load-balanced group, so you should use the host name and a semicolon. In order to limit the number of individual proxy servers used in a round-robin DNS entry, set `CVMFS_MAX_IPADDR_PER_PROXY`. This can also limit the perceived "hang duration" while CernVM-FS performs fail-overs.

The `DIRECT` keyword for a hostname avoids using a proxy altogether. Note that `CVMFS_HTTP_PROXY` must be defined in order to mount CVMFS, but to avoid using any proxies, you can set the parameter to `DIRECT`. However, note that this is not recommended for large numbers of clients accessing remote stratum servers, and stratum server administrators may ask you to deploy and use proxies.

`CVMFS_HTTP_PROXY` is typically configured with a primary proxy group listed first, and potentially other proxy groups listed after that for backup. In order to prevent CernVM-FS from permanently using the backup proxies after a fail-over, CernVM-FS will automatically retry the first proxy group in the list after some time. The delay for re-trying is set in seconds by `CVMFS_PROXY_RESET_AFTER`. This reset behaviour can be disabled by setting this parameter to 0.

Proxy List Examples

Suppose there are two proxy servers local to your site, `p1.site.example.org` and `p2.site.example.org`, and two regional proxy servers nearby available for backup use, `p3.region.example.org` and `p4.region.example.org`. In this example all proxy servers are configured to listen on port 3128. If the two local proxies are equally preferable to use and configured identically to each other, and the same applies for the two regional proxies, use

```
CVMFS_HTTP_PROXY="http://p1.site.example.org:3128|http://p2.site.example.org:3128;
↪http://p3.region.example.org:3128|http://p4.region.example.org:3128"
```

However, if `p1` should always be preferred over `p2` (for example if it has a faster network or larger cache), use

```
CVMFS_HTTP_PROXY="http://p1.site.example.org:3128;http://p2.site.example.org:3128;
↪http://p3.region.example.org:3128|http://p4.region.example.org:3128"
```

Moreover, if `p3` should always be preferred over `p4` (for example if it is significantly closer to your site), use

```
CVMFS_HTTP_PROXY="http://p1.site.example.org:3128;http://p2.site.example.org:3128;
↪http://p3.region.example.org:3128;http://p4.region.example.org:3128"
```

[1] The usual proxy notation rules apply, like `http://proxy1:8080|http://proxy2:8080;DIRECT`

Automatic Proxy Configuration

The proxy settings can be automatically gathered through WPAD. The special proxy server "auto" in CVMFS_HTTP_PROXY is resolved according to the proxy server specification loaded from a PAC file. PAC files can be on a file system or accessible via HTTP. CernVM-FS looks for PAC files in the order given by the semicolon separated URLs in the CVMFS_PAC_URLS environment variable. This variable defaults to http://wpad/wpad.dat. The auto keyword used as a URL in CVMFS_PAC_URLS is resolved to http://wpad/wpad.dat, too, in order to be compatible with Frontier [Blumenfeld08].

Fallback Proxy List

In addition to the regular proxy list set by CVMFS_HTTP_PROXY, a fallback proxy list is supported in CVMFS_FALLBACK_PROXY. The syntax of both lists is the same. The fallback proxy list is appended to the regular proxy list, and if the fallback proxy list is set, any DIRECT is removed from both lists. The automatic proxy configuration of the previous section only sets the regular proxy list, not the fallback proxy list. Also the fallback proxy list can be automatically reordered; see the next section.

Ordering of Servers according to Geographic Proximity

CernVM-FS Stratum 1 servers provide a RESTful service for geographic ordering. Clients can request *http://<HOST>/cvmfs/<FQRN>/api/v1.0/geo/<proxy_address>/<server_list>* The proxy address can be replaced by a UUID if no proxies are used, and the CernVM-FS client does that if there are no regular proxies. The server list is comma-separated. The result is an ordered list of indexes of the input host names. Use of this API can be enabled in a CernVM-FS client with CVMFS_USE_GEOAPI=yes. That will geographically sort both the servers set by CVMFS_SERVER_URL and the fallback proxies set by CVMFS_FALLBACK_PROXY.

Timeouts

CernVM-FS tries to gracefully recover from broken network links and temporarily overloaded paths. The timeout for connection attempts and for very slow downloads can be set by CVMFS_TIMEOUT and CVMFS_TIMEOUT_DIRECT. The two timeout parameters apply to a connection with a proxy server and to a direct connection to a Stratum 1 server, respectively. A download is considered to be "very slow" if the transfer rate is below for more than the timeout interval. The threshold can be adjusted with the CVMFS_LOW_SPEED_LIMIT parameter. A very slow download is treated like a broken connection.

On timeout errors and on connection failures (but not on name resolving failures), CernVM-FS will retry the path using an exponential backoff. This introduces a jitter in case there are many concurrent requests by a cluster of nodes, allowing a proxy server or web server to serve all the nodes consecutively. CVMFS_MAX_RETRIES sets the number of retries on a given path before CernVM-FS tries to switch to another proxy or host. The overall number of requests with a given proxy/host combination is $CVMFS_MAX_RETRIES+1. CVMFS_BACKOFF_INIT sets the maximum initial backoff in seconds. The actual initial backoff is picked with milliseconds precision randomly in the interval $[1, \$CVMFS_BACKOFF_INIT \cdot 1000]$. With every retry, the backoff is then doubled.

DNS Nameserver Changes

CernVM-FS can watch /etc/resolv.conf and automatically follow changes to the DNS servers. This behavior is controlled by the CVMFS_DNS_ROAMING client configuration. It is by default turned on on macOS and turned off on Linux.

Network Path Selection

This section summarizes the CernVM-FS mechanics to select a network path from the client through an HTTP forward proxy to an HTTP endpoint. At any given point in time, there is only one combination of web proxy and web host that a new request will utilize. In this section, it is this combination of proxy and host that is called "network path". The network path is chosen from the collection of web proxies and hosts in the CernVM-FS configuration according to the following rules.

Host Selection

The hosts specified as an ordered list. CernVM-FS will always start with the first host and fail-over one by one to the next hosts in the list.

Proxy Selection

Web proxies are treated as an ordered list of load-balance groups. Like the hosts, load-balance groups will be probed one after another. Within a load-balance group, a proxy is chosen at random. DNS proxy names that resolve to multiple IP addresses are automatically transformed into a proxy load-balance group, whose maximum size can be limited by `CVMFS_MAX_IPADDR_PER_PROXY`.

Proxy Sharding

In the default (non-sharded) configuration, each CernVM-FS client will independently choose a single proxy to be used for all requests. For sites with many clients that are likely to access the same content, this can result in unnecessary duplication of cached content across multiple proxies.

If proxy sharding is enabled via the `CVMFS_PROXY_SHARD` parameter, all proxies within a load-balancing group are used concurrently. Each proxy handles a subset of the requests. Proxies are selected using consistent hashing so that multiple clients will independently select the same proxy for a given request, to maximise cache efficiency. If any proxy fails, CernVM-FS automatically removes it from the load-balancing group and distributes its requests evenly across the remaining proxies.

Failover Rules

On download failures, CernVM-FS tries to figure out if the failure is caused by the host or by the proxy.

- Failures of host name resolution, HTTP 5XX and 404 return codes, and any connection/timeout error, partial file transfer, or non 2XX return code in case no proxy is in use are classified as host failure.
- Failures of proxy name resolution and any connection/timeout error, partial file transfer, or non 2XX return code (except 5XX and 404) are classified as proxy failure if a proxy server is used.
- Explicit proxy errors (indicated via the *X-Squid-Error* or *Proxy-Status* headers) will always be classified as proxy failure.

If CernVM-FS detects a host failure, it will fail-over to the next host in the list while keeping the proxy server untouched. If it detects a proxy failure, it will fail-over to to another proxy while keeping the host untouched. CernVM-FS will try all proxies of the current load-balance group in random order before trying proxies from the next load-balance group.

The change of host or proxy is a global change affecting all subsequent requests. In order to avoid concurrent requests changing the global network path at the same time, the actual change of path is only performed if the global host/proxy is equal to the currently used host/proxy of the request. Otherwise, the request assumes that another request already performed the fail-over and only the request's fail-over counter is increased.

In order to avoid endless loops, every request carries a host fail-over counter and a proxy fail-over counter. Once this counter reaches the number of host/proxies, CernVM-FS gives up and returns a failure.

The failure classification can mistakenly take a host failure for a proxy failure. Therefore, after all proxies have been probed, a connection/timeout error, partial file transfer, or non 2XX return code is treated like a host failure in any case and the proxy server as well as the proxy server failure counter of the request at hand is reset. This way, eventually all possible network paths are examined.

Network Path Reset Rules

On host or proxy fail-over, CernVM-FS will remember the timestamp of the failover. The first request after a given grace period (see *Default Values*) will reset the proxy to a random proxy of the first load-balance group or the host to the first host, respectively. If the default proxy/host is still unavailable, the fail-over routines again switch to a working network path.

Retry and Backoff

On connection and timeout errors, CernVM-FS retries a fixed, limitied number of times on the same network path before performing a fail-over. Retrying involves an exponential backoff with a minimum and maximum waiting time.

Default Values

- Network timeout for connections using a proxy: 5 seconds (adjustable by `CVMFS_TIMEOUT`)
- Network timeout for connections without a proxy: 10 seconds (adjustable by `CVMFS_TIMEOUT_DIRECT`)
- Grace period for proxy reset after fail-over: 5 minutes (adjustable by `CVMFS_PROXY_RESET_AFTER`)
- Grace period for host reset after fail-over: 30 minutes (adjustable by `CVMFS_HOST_RESET_AFTER`)
- Maximum number of retries on the same network path: 1 (adjustable by `CVMFS_MAX_RETRIES`)
- Minimum waiting time on a retry: 2 seconds (adjustable by CVMFS_BACKOFF_MIN)
- Maximum waiting time on a retry: 10 seconds (adjustable by CVMFS_BACKOFF_MAX)
- Minimum/Maximum DNS name cache: 1 minute / 1 day

Note: a continuous transfer rate below 1kB/s is treated like a network timeout.

2.4.4 Cache Settings

Downloaded files will be stored in a local cache directory. The CernVM-FS cache has a soft quota; as a safety margin, the partition hosting the cache should provide more space than the soft quota limit; we recommend to leave at least 20% + 1GB.

Once the quota limit is reached, CernVM-FS will automatically remove files from the cache according to the least recently used policy. Removal of files is performed bunch-wise until half of the maximum cache size has been freed. The quota limit can be set in Megabytes by `CVMFS_QUOTA_LIMIT`. For typical repositories, a few Gigabytes make a good quota limit.

The cache directory needs to be on a local file system in order to allow each host the accurate accounting of the cache contents; on a network file system, the cache can potentially be modified by other hosts. Furthermore, the cache directory is used to create (transient) sockets and pipes, which is usually only supported by a local file system. The location of the cache directory can be set by `CVMFS_CACHE_BASE`.

On SELinux enabled systems, the cache directory and its content need to be labeled as `cvmfs_cache_t`. During the installation of CernVM-FS RPMs, this label is set for the default cache directory /var/lib/cvmfs. For other directories, the label needs to be set manually by `chcon -Rv --type=cvmfs_cache_t $CVMFS_CACHE_BASE`.

Each repository can either have an exclusive cache or join the CernVM-FS shared cache. The shared cache enforces a common quota for all repositories used on the host. File duplicates across repositories are stored only once in the

shared cache. The quota limit of the shared directory should be at least the maximum of the recommended limits of its participating repositories. In order to have a repository not join the shared cache but use an exclusive cache, set CVMFS_SHARED_CACHE=no.

Alien Cache

An "alien cache" provides the possibility to use a data cache outside the control of CernVM-FS. This can be necessary, for instance, in HPC environments where local disk space is not available or scarce but powerful cluster file systems are available. The alien cache directory is a directory in addition to the ordinary cache directory. The ordinary cache directory is still used to store control files.

The alien cache directory is set by the CVMFS_ALIEN_CACHE option. It can be located anywhere including cluster and network file systems. If configured, all data chunks are stored there. CernVM-FS ensures atomic access to the cache directory. It is safe to have the alien directory shared by multiple CernVM-FS processes and it is safe to unlink files from the alien cache directory anytime. The contents of files, however, must not be touched by third-party programs.

In contrast to normal cache mode where files are store in mode 0600, in the alien cache files are stored in mode 0660. So all users being part of the alien cache directory's owner group can use it.

The skeleton of the alien cache directory should be created upfront. Otherwise, the first CernVM-FS process accessing the alien cache determines the ownership. The cvmfs2 binary can create such a skeleton using

```
cvmfs2 __MK_ALIEN_CACHE__ $alien_cachedir $owner_uid $owner_gid
```

Since the alien cache is unmanaged, there is no automatic quota management provided by CernVM-FS; the alien cache directory is ever-growing. The CVMFS_ALIEN_CACHE requires CVMFS_QUOTA_LIMIT=-1 and CVMFS_SHARED_CACHE=no.

The alien cache might be used in combination with a special repository replication mode that preloads a cache directory (Section *Setting up a Replica Server (Stratum 1)*). This allows to propagate an entire repository into the cache of a cluster file system for HPC setups that do not allow outgoing connectivity.

Advanced Cache Configuration

For exotic cache configurations, CernVM-FS supports specifying multiple, independent "cache manager instances" of different types. Such cache manager instances replace the local cache directory. Since the local cache directory is also used to store transient special files, CVMFS_WORKSPACE=$local_path must be used when advanced cache configuration is used.

A concrete cache manager instance has a user-defined name and it is specified like

```
CVMFS_CACHE_PRIMARY=myInstanceName
CVMFS_CACHE_myInstanceName_TYPE=posix
```

Multiple instances can thus be safely defined with different names but only one is selected when the client boots. The following table lists the valid cache manager instance types.

** Type**	**Behavior**
posix	Uses a cache directory with the standard cache implementation
tiered	Uses two other cache manager instances in a layered configuration
external	Uses an external cache plugin process (see Section *Client Plug-Ins*)

The instance name "default" is blocked because the regular cache configuration syntax is automatically mapped to CVMFS_CACHE_default_... parameters. The command sudo cvmfs_talk cache instance can be used to show the currently used cache manager instance.

Tiered Cache

The tiered cache manager combines two other cache manager instances as an upper layer and a lower layer into a single functional cache manager. Usually, a small and fast upper layer (SSD, memory) is combined with a larger and slower lower layer (HDD, network drive). The upper layer needs to be large enough to serve all currently open files. On an upper layer cache miss, CernVM-FS tries to copy the missing object from the lower into the upper layer. On a lower layer cache miss, CernVM-FS download and stores objects either in both layers or in the upper layer only, depending on the configuration.

The parameters `CVMFS_CACHE_$tieredInstanceName_UPPER` and `CVMFS_CACHE_$tieredInstanceName_LOWER` set the names of the upper and the lower instances. The parameter `CVMFS_CACHE_$tieredInstanceName_LOWER_READONLY=[yes|no]` controls whether the lower layer can be populated by the client or not.

External Cache Plugin

A CernVM-FS cache manager instance can be provided by an external process. The cache manager process and the CernVM-FS client are connected through a socket, whose address is called "locator". The locator can either address a UNIX domain socket on the local file system, or a TCP socket, as in the following examples

```
CVMFS_CACHE_instanceName_LOCATOR=unix=/var/lib/cvmfs/cache.socket
# or
CVMFS_CACHE_instanceName_LOCATOR=tcp=192.168.0.24:4242
```

If a UNIX domain socket is used, both the CernVM-FS client and the cache manager need to be able to access the socket file. Usually that means they have to run under the same user.

Instead of manually starting the cache manager, the CernVM-FS client can optionally automatically start and stop the cache manager process. This is called a "supervised cache manager". The first booting CernVM-FS client starts the cache manager process, the last terminating client stops the cache manager process. In order to start the cache manager in supervised mode, use `CVMFS_CACHE_instanceName_CMDLINE=<executable and arguments>`, using a comma (`,`) instead of a space to separate the command line parameters.

Example

The following example configures a tiered cache with an external cache plugin as an upper layer and a read-only, network drive as a lower layer. The cache plugin uses memory to cache data and is part of the CernVM-FS client. This configuration could be used in a data center with diskless nodes and a preloaded cache on a network drive (see Chapter *CernVM-FS on Supercomputers*)

```
CVMFS_WORKSPACE=/var/lib/cvmfs
CVMFS_CACHE_PRIMARY=hpc

CVMFS_CACHE_hpc_TYPE=tiered
CVMFS_CACHE_hpc_UPPER=memory
CVMFS_CACHE_hpc_LOWER=preloaded
CVMFS_CACHE_hpc_LOWER_READONLY=yes

CVMFS_CACHE_memory_TYPE=external
CVMFS_CACHE_memory_CMDLINE=/usr/libexec/cvmfs/cache/cvmfs_cache_ram,/etc/cvmfs/cache-
↪mem.conf
CVMFS_CACHE_memory_LOCATOR=unix=/var/lib/cvmfs/cvmfs-cache.socket

CVMFS_CACHE_preloaded_TYPE=posix
CVMFS_CACHE_preloaded_ALIEN=/gpfs/cvmfs/alien
CVMFS_CACHE_preloaded_SHARED=no
CVMFS_CACHE_preloaded_QUOTA_LIMIT=-1
```

The example configuration for the in-memory cache plugin in /etc/cvmfs/cache-mem.conf is

```
CVMFS_CACHE_PLUGIN_LOCATOR=unix=/var/lib/cvmfs/cvmfs-cache.socket
# 2G RAM
CVMFS_CACHE_PLUGIN_SIZE=2000
```

2.4.5 NFS Server Mode

In case there is no local hard disk space available on a cluster of worker nodes, a single CernVM-FS client can be exported via nfs [Callaghan95] [Shepler03] to these worker nodes.This mode of deployment will inevitably introduce a performance bottleneck and a single point of failure and should be only used if necessary.

NFS export requires Linux kernel >= 2.6.27 on the NFS server. For instance, exporting works for Scientific Linux 6 but not for Scientific Linux 5. The NFS server should run a lock server as well. For proper NFS support, set `CVMFS_NFS_SOURCE=yes`. On the client side, all available nfs implementations should work.

In the NFS mode, upon mount an additional directory nfs_maps.$repository_name appears in the CernVM-FS cache directory. These *NFS maps* use leveldb to store the virtual inode CernVM-FS issues for any accessed path. The virtual inode may be requested by NFS clients anytime later. As the NFS server has no control over the lifetime of client caches, entries in the NFS maps cannot be removed.

Typically, every entry in the NFS maps requires some 150-200 Bytes. A recursive `find` on /cvmfs/atlas.cern.ch with 50 million entries, for instance, would add up 8GB in the cache directory. For a CernVM-FS instance that is exported via NFS, the safety margin for the NFS maps needs be taken into account. It also might be necessary to monitor the actual space consumption.

Note: The NFS share should be mounted with the mount option `nordirplus`. Without this option, traversals of directories with large number of files can slow down significantly.

Tuning

The default settings in CernVM-FS are tailored to the normal, non-NFS use case. For decent performance in the NFS deployment, the amount of memory given to the meta-data cache should be increased. By default, this is 16M. It can be increased, for instance, to 256M by setting `CVMFS_MEMCACHE_SIZE` to 256. Furthermore, the maximum number of download retries should be increased to at least 2.

The number of NFS daemons should be increased as well. A value of 128 NFS daemons has shown perform well. In Scientific Linux, the number of NFS daemons is set by the `RPCNFSDCOUNT` parameter in /etc/sysconfig/nfs.

The performance will benefit from large RAM on the NFS server ($\geq$ 16GB) and CernVM-FS caches hosted on an SSD hard drive.

Export of /cvmfs with Cray DVS

On Cray DVS and possibly other systems that export /cvmfs as a whole instead of individual repositories as separate volumes, an additional effort is needed to ensure that inodes are distinct from each other across multiple repositories. The `CVMFS_NFS_INTERLEAVED_INODES` parameter can be used to configure repositories to only issue inodes of a particular residue class. To ensure pairwise distinct inodes across repositories, each repository should be configured with a different residue class. For instance, in order to avoid inode clashes between the atlas.cern.ch and the cms.cern.ch repositories, there can be a configuration file /etc/cvmfs/config.d/atlas.cern.ch.local with

```
CVMFS_NFS_INTERLEAVED_INODES=0%2 # issue inodes 0, 2, 4, ...
```

and a configuration file /etc/cvmfs/config.d/cms.cern.ch.local with

```
CVMFS_NFS_INTERLEAVED_INODES=1%2 # issue inodes 1, 3, 5, ...
```

The maximum number of possibly exported repositories needs to be known in advance. The CVMFS_NFS_INTERLEAVED_INODES only has an effect in NFS mode.

Shared NFS Maps (HA-NFS)

As an alternative to the existing, leveldb managed NFS maps, the NFS maps can optionally be managed out of the CernVM-FS cache directory by SQLite. This allows the NFS maps to be placed on shared storage and accessed by multiple CernVM-FS NFS export nodes simultaneously for clustering and active high-availablity setups. In order to enable shared NFS maps, set CVMFS_NFS_SHARED to the path that should be used to host the SQLite database. If the path is on shared storage, the shared storage has to support POSIX file locks. The drawback of the SQLite managed NFS maps is a significant performance penalty which in practice can be covered by the memory caches.

Example

An example entry /etc/exports (note: the fsid needs to be different for every exported CernVM-FS repository)

```
/cvmfs/atlas.cern.ch 172.16.192.0/24(ro,sync,no_root_squash,\
  no_subtree_check,fsid=101)
```

A sample entry /etc/fstab entry on a client:

```
172.16.192.210:/cvmfs/atlas.cern.ch /cvmfs/atlas.cern.ch nfs4 \
  ro,ac,actimeo=60,lookupcache=all,nolock,rsize=1048576,wsize=1048576 0 0
```

2.4.6 File Ownership

By default, cvmfs presents all files and directories as belonging to the mounting user, which for system mounts under /cvmfs is the user cvmfs. Alternatively, CernVM-FS can present the uid and gid of file owners as they have been at the time of publication by setting CVMFS_CLAIM_OWNERSHIP=no.

If the real uid and gid values are shown, stable uid and gid values across nodes are recommended; otherwise the owners shown on clients can be confusing. The client can also dynamically remap uid and gid values. To do so, the parameters CVMFS_UID_MAP and CVMFS_GID_MAP should provide the path to text files that specify the mapping. The format of the map files is identical to the map files used for *bulk changes of ownership on release manager machines*.

2.4.7 Hotpatching and Reloading

By hotpatching a running CernVM-FS instance, most of the code can be reloaded without unmounting the file system. The current active code is unloaded and the code from the currently installed binaries is loaded. Hotpatching is logged to syslog. Since CernVM-FS is re-initialized during hotpatching and configuration parameters are re-read, hotpatching can be also seen as a "reload".

Hotpatching has to be done for all repositories concurrently by

```
cvmfs_config [-c] reload
```

The optional parameter -c specifies if the CernVM-FS cache should be wiped out during the hotpatch. Reloading of the parameters of a specific repository can be done like

```
cvmfs_config reload atlas.cern.ch
```

In order to see the history of loaded CernVM-FS Fuse modules, run

```
cvmfs_talk hotpatch history
```

The currently loaded set of parameters can be shown by

```
cvmfs_talk parameters
```

The CernVM-FS packages use hotpatching in the package upgrade process.

2.4.8 Auxiliary Tools

cvmfs_fsck

CernVM-FS assumes that the local cache directory is trustworthy. However, it might happen that files get corrupted in the cache directory caused by errors outside the scope of CernVM-FS. CernVM-FS stores files in the local disk cache with their cryptographic content hash key as name, which makes it easy to verify file integrity. CernVM-FS contains the `cvmfs_fsck` utility to do so for a specific cache directory. Its return value is comparable to the system's `fsck`. For example,

```
cvmfs_fsck -j 8 /var/lib/cvmfs/shared
```

checks all the data files and catalogs in `/var/lib/cvmfs/shared` using 8 concurrent threads. Supported options are:

-v	Produce more verbose output.
-j #threads	Sets the number of concurrent threads that check files in the cache directory. Defaults to 4.
-p	Tries to automatically fix problems.
-f	Unlinks the cache database. The database will be automatically rebuilt by CernVM-FS on next mount.

The `cvmfs_config fsck` command can be used to verify all configured repositories.

cvmfs_config

The `cvmfs_config` utility provides commands in order to setup the system for use with CernVM-FS.

setup
: The `setup` command takes care of basic setup tasks, such as creating the cvmfs user and allowing access to CernVM-FS mount points by all users.

chksetup
: The `chksetup` command inspects the system and the CernVM-FS configuration in /etc/cvmfs for common problems.

showconfig
: The `showconfig` command prints the CernVM-FS parameters for all repositories or for the specific repository given as argument. With the *-s* option, only non-empty parameters are shown.

stat
: The `stat` command prints file system and network statistics for currently mounted repositories.

status
: The `status` command shows all currently mounted repositories and the process id (PID) of the CernVM-FS processes managing a mount point.

probe
: The `probe` command tries to access /cvmfs/$repository for all repositories specified in `CVMFS_REPOSITORIES` or the ones specified as a space separated list on the command line, respectively.

fsck
: Run `cvmfs_fsck` on all repositories specified in `CVMFS_REPOSITORIES`.

fuser
Identify all the processes that are accessing a cvmfs repository, preventing it from either being unmounted or mounted. See *Remounting and Namespaces/Containers*.

reload
The `reload` command is used to *reload or hotpatch CernVM-FS instances*.

umount
The `umount` command unmounts all currently mounted CernVM-FS repositories, which will only succeed if there are no open file handles on the repositories.

wipecache
The `wipecache` command is an alias for `reload -c`.

killall
The `killall` command immediately unmounts all repositories under /cvmfs and terminates the associated processes. It is meant to escape from a hung state without the need to reboot a machine. However, all processes that use CernVM-FS at the time will be terminated, too. The need to use this command very likely points to a network problem or a bug in cvmfs.

bugreport
The `bugreport` command creates a tarball with collected system information which can be attached to a bug report.

cvmfs_talk

The `cvmfs_talk` command provides a way to control a currently running CernVM-FS process and to extract information about the status of the corresponding mount point. Most of the commands are for special purposes only or covered by more convenient commands, such as `cvmfs_config showconfig` or `cvmfs_config stat`. Four commands might be of particular interest though.

```
cvmfs_talk cleanup 0
```

will, without interruption of service, immediately cleanup the cache from all files that are not currently pinned in the cache.

```
cvmfs_talk cleanup rate 120
```

shows the number of cache cleanups in the last two hours (120 minutes). If this value is larger than one or two, the cache size is probably two small and the client experiences cache thrashing.

```
cvmfs_talk internal affairs
```

prints the internal status information and performance counters. It can be helpful for performance engineering.

```
cvmfs_talk -i <repo> remount
```

starts the catalog update routine. When using `remount sync` the system waits for the new file system snapshot to be served (if there is a new one).

Other

Information about the current cache usage can be gathered using the `df` utility. For repositories created with the CernVM-FS 2.1 toolchain, information about the overall number of file system entries in the repository as well as the number of entries covered by currently loaded meta-data can be gathered by `df -i`.

For the Nagios monitoring system [Schubert08], a checker plugin is available on our website.

2.4.9 Debug Logs

The `cvmfs2` binary forks a watchdog process on start. Using this watchdog, CernVM-FS is able to create a stack trace in case certain signals (such as a segmentation fault) are received. The watchdog writes the stack trace into syslog as well as into a file `stacktrace` in the cache directory.

CernVM-FS can be started in debug mode. In the debug mode, CernVM-FS will log with high verbosity which makes the debug mode unsuitable for production use. In order to turn on the debug mode, set `CVMFS_DEBUGLOG=/tmp/cvmfs.log`.

2.5 Setting up a Local Squid Proxy

For clusters of nodes with CernVM-FS clients, we strongly recommend setting up two or more Squid forward proxy servers as well. The forward proxies will reduce the latency for the local worker nodes, which is critical for cold cache performance. They also reduce the load on the Stratum 1 servers.

From what we have seen, a Squid server on commodity hardware scales well for at least a couple of hundred worker nodes. The more RAM and hard disk you can devote for caching the better. We have good experience with memory cache and hard disk cache. We suggest setting up two identical Squid servers for reliability and load-balancing. Assuming the two servers are A and B, set

```
CVMFS_HTTP_PROXY="http://A:3128|http://B:3128"
```

Squid is very powerful and has lots of configuration and tuning options. For CernVM-FS we require only the very basic static content caching. If you already have a Frontier Squid installed you can use it as well for CernVM-FS.

One option that is particularly important when there are a lot of worker nodes and jobs that start close together is the *collapsed_forwarding* option. This combines multiple simultaneous requests for the same object into a single request to a Stratum 1 server. This did not work properly on squid versions prior to 3.5.28, which includes the default squid on EL7. This also works properly in Frontier Squid.

In any case, cache sizes and access control needs to be configured in order to use the Squid server with CernVM-FS. In order to do so, browse through your /etc/squid/squid.conf and make sure the following lines appear accordingly:

```
collapsed_forwarding on
minimum_expiry_time 0
maximum_object_size 1024 MB

cache_mem 128 MB
maximum_object_size_in_memory 128 KB
# 50 GB disk cache
cache_dir ufs /var/spool/squid 50000 16 256
```

Furthermore, Squid needs to allow access to all Stratum 1 servers. This is controlled through Squid ACLs. Most sites allow all of their IP addresses to connect to any destination address. By default squid allows that for the standard private IP addresses, but if you're not using a private network then add your public address ranges, with something like this:

```
acl localnet src A.B.C.D/NN
```

If you instead want to limit the destinations to major cvmfs Stratum 1s, it is better to use the list built in to Frontier Squid because the list is sometimes updated with new releases.

The Squid configuration can be verified by `squid -k parse`. Before the first service start, the cache space on the hard disk needs to be prepared by `squid -z`. In order to make enough file descriptors available to squid, execute `ulimit -n 8192` or some higher number prior to starting the squid service.

2.6 Creating a Repository (Stratum 0)

CernVM-FS is a file system with a single source of (new) data. This single source, the repository *Stratum 0*, is maintained by a dedicated *release manager machine* or *publisher*. A read-writable copy of the repository is accessible on the publisher. The CernVM-FS server tool kit is used to *publish* the current state of the repository on the release manager machine. Publishing is an atomic operation.

All data stored in CernVM-FS have to be converted into a CernVM-FS *repository* during the process of publishing. The CernVM-FS repository is a form of content-addressable storage. Conversion includes creating the file catalog(s), compressing new and updated files and calculating content hashes. Storing the data in a content-addressable format results in automatic file de-duplication. It furthermore simplifies data verification and it allows for file system snapshots.

In order to provide a writable CernVM-FS repository, CernVM-FS uses a union file system that combines a read-only CernVM-FS mount point with a writable scratch area. *This figure below* outlines the process of publishing a repository.

2.6.1 CernVM-FS Server Quick-Start Guide

System Requirements

- Apache HTTP server *or* S3 compatible storage service
- union file system in the kernel
  - AUFS
  - OverlayFS (as of kernel version 4.2.x or RHEL7.3)
- Officially supported platforms
  - CentOS/SL >= 7.3, provided that /var/spool/cvmfs is served by an ext4 file system.
  - Fedora 25 and above (with kernel $\geq$ 4.2.x)
  - Ubuntu 14.04 64 bit and above
    - Ubuntu < 15.10: with installed AUFS kernel module (cf. *linux-image-extra* package)
    - Ubuntu 15.10 and later (using upstream OverlayFS)

Installation

1. Install `cvmfs` and `cvmfs-server` packages
2. Ensure enough disk space in `/var/spool/cvmfs` (>50GiB)
3. For local storage: Ensure enough disk space in `/srv/cvmfs`
4. Create a repository with `cvmfs_server mkfs` (See *Repository Creation*)

Content Publishing

1. `cvmfs_server transaction <repository name>`
2. Install content into `/cvmfs/<repository name>`
3. Create nested catalogs at proper locations
   - Create `.cvmfscatalog` files (See *Managing Nested Catalogs*) or
   - Consider using a `.cvmfsdirtab` file (See *Managing Nested Catalogs with .cvmfsdirtab*)
4. `cvmfs_server publish <repository name>`

Backup Policy

- Create backups of signing key files in `/etc/cvmfs/keys`
- Entire repository content
  - For local storage: `/srv/cvmfs`
  - Stratum 1s can serve as last-ressort backup of repository content

2.6.2 Publishing a new Repository Revision

Fig. 3: Updating a mounted CernVM-FS repository by overlaying it with a copy-on-write union file system volume. Any changes will be accumulated in a writable volume (yellow) and can be synchronized into the CernVM-FS repository afterwards. The file catalog contains the directory structure as well as file metadata, symbolic links, and secure hash keys of regular files. Regular files are compressed and renamed to their cryptographic content hash before copied into the data store.

Since the repositories may contain many file system objects (i.e. ATLAS contains $70 * 10^6$ file system objects – February 2016), we cannot afford to generate an entire repository from scratch for every update. Instead, we add a writable file system layer on top of a mounted read-only CernVM-FS repository using a union file system. This renders a read-only CernVM-FS mount point writable to the user, while all performed changes are stored in a special writable scratch area managed by the union file system. A similar approach is used by Linux Live Distributions that are shipped on read-only media, but allow *virtual* editing of files where changes are stored on a RAM disk.

If a file in the CernVM-FS repository gets changed, the union file system first copies it to the writable volume and applies any changes to this copy (copy-on-write semantics). Also newly created files or directories will be stored in the writable volume. Additionally the union file system creates special hidden files (called *white-outs*) to keep track of file deletions in the CernVM-FS repository.

Eventually, all changes applied to the repository are stored in this scratch area and can be merged into the actual CernVM-FS repository by a subsequent synchronization step. Up until the actual synchronization step takes place, no changes are applied to the CernVM-FS repository. Therefore, any unsuccessful updates to a repository can be rolled back by simply clearing the writable file system layer of the union file system.

2.6.3 Requirements for a new Repository

In order to create a repository, the server and client part of CernVM-FS must be installed on the release manager machine. Furthermore you will need a kernel containing a union file system implementation as well as a running `Apache2` web server. Currently we support EL >= 7.3, Ubuntu 14.04+ and Fedora 25+ distributions.

CernVM-FS supports both OverlayFS and aufs as a union file system. At least a 4.2.x kernel is needed to use CernVM-FS with OverlayFS. (Red Hat) Enterprise Linux >= 7.3 works, too, provided that /var/spool/cvmfs is served by an ext3 or ext4 file system. Furthermore note that OverlayFS cannot fully comply with POSIX semantics, in particular hard links must be broken into individual files. That is usually not a problem but should be kept in mind when installing certain software distributions into a CernVM-FS repository.

2.6.4 Notable CernVM-FS Server Locations and Files

There are a number of possible customisations in the CernVM-FS server installation. The following table provides an overview of important configuration files and intrinsical paths together with some customisation hints. For an exhaustive description of the CernVM-FS server infrastructure please consult Appendix "*CernVM-FS Server Infrastructure*".

File Path	Description
`/cvmfs`	**Repository mount points** Contains read-only union file system mountpoints that become writable during repository updates. Do not symlink or manually mount anything here.
`/srv/cvmfs`	**Central repository storage location** Can be mounted or symlinked to another location *before* creating the first repository.
`/srv/cvmfs/<fqrn>`	**Storage location of a repository** Can be symlinked to another location *before* creating the repository <fqrn>.
`/var/spool/cvmfs`	**Internal states of repositories** Can be mounted or symlinked to another location *before* creating the first repository. Hosts the scratch area described *here*, thus might consume notable disk space during repository updates.
`/etc/cvmfs`	**Configuration files and keychains** Similar to the structure described in *this table*. Do not symlink this directory.
`/etc/cvmfs/cvmfs_server_hooks.sh`	**Customisable server behaviour** See "*Customizable Actions Using Server Hooks*" for further details
`/etc/cvmfs/repositories.d`	**Repository configuration location** Contains repository server specific configuration files.

2.6.5 CernVM-FS Repository Creation and Updating

The CernVM-FS server tool kit provides the `cvmfs_server` utility in order to perform all operations related to repository creation, updating, deletion, replication and inspection. Without any parameters it prints a short documentation of its commands.

Repository Creation

A new repository is created by `cvmfs_server mkfs`:

```
cvmfs_server mkfs my.repo.name
```

The utility will ask for a user that should act as the owner of the repository and afterwards create all the infrastructure for the new CernVM-FS repository. Additionally it will create a reasonable default configuration and generate a new release manager certificate and by default a new master key and corresponding public key (see more about that in the next section).

The `cvmfs_server` utility will use `/srv/cvmfs` as storage location by default. In case a separate hard disk should be used, a partition can be mounted on /srv/cvmfs or /srv/cvmfs can be symlinked to another location (see *Notable CernVM-FS Server Locations and Files*). Besides local storage it is possible to use an *S3 compatible storage service* as data backend.

Once created, the repository is mounted under `/cvmfs/my.repo.name` containing only a single file called `new_repository`. The next steps describe how to change the repository content.

The repository name resembles a DNS scheme but it does not need to reflect any real server name. It is supposed to be a globally unique name that indicates where/who the publishing of content takes place. A repository name must only contain alphanumeric characters plus `-`, `_`, and `.` and it is limited to a length of 60 characters.

Master keys

Each cvmfs repository uses two sets of keys, one for the individual repository and another called the "masterkey" which signs the repository key. The pub key that corresponds to the masterkey is what needs to be distributed to clients to verify the authenticity of the repository. It is usually most convenient to share the masterkey between all repositories in a domain so new repositories can be added without updating the client configurations. If the clients are maintained by multiple organizations it can be very difficult to quickly update the distributed pub key, so in that case it is important to keep the masterkey especially safe from being stolen. If only repository keys are stolen, they can be replaced without having to update client configurations.

By default, `cvmfs_server mkfs my.repo.name` creates a new `/etc/cvmfs/keys/my.repo.name.masterkey` and corresponding `/etc/cvmfs/keys/my.repo.name.pub` for every new repository. Additional user-written procedures can then be applied to replace those files with a common masterkey/pub pair, and then `cvmfs_server resign` must be run to update the corresponding signature (in `/srv/cvmfs/my.repo.name/.cvmfswhitelist`). Signatures are only good for 30 days by default, so `cvmfs_server resign` must be run again before they expire.

`cvmfs_server` also supports the ability to store the masterkey in a separate inexpensive smartcard, so that even if the computer hosting the repositories is compromised, the masterkey cannot be stolen. Smartcards allow writing keys into them and signing files but they never allow reading the keys back. Currently the supported hardware are the Yubikey 4 or Nano USB devices.

If one of those devices is plugged in to a release manager machine, this is how to use it:

1. Create a repository with `cvmfs_server mkfs my.repo.name`
2. Store its masterkey and pub into the smartcard with `cvmfs_server masterkeycard -s my.repo.name`
3. **Make a backup copy of `/etc/cvmfs/keys/my.repo.name.masterkey` on**

   at least one USB flash drive because the next step will irretrievably delete the file. Keep the flash drive offline in a safe place in case something happens to the smartcard.
4. Convert the repository to use the smartcard with `cvmfs_server masterkeycard -c my.repo.name`. This will delete the masterkey file. This command can also be applied to other repositories on the same machine; their pub file will be updated with what is stored in the card and they will be resigned.

From then on, every newly created repository on the same machine will automatically use the shared masterkey stored on the smartcard.

When using a masterkeycard, the default signature expiration reduces from 30 days to 7 days. `cvmfs_server resign` needs to be run to renew the signature. It is recommended to run that daily from cron.

Repositories for Volatile Files

Repositories can be flagged as containing *volatile* files using the `-v` option:

```
cvmfs_server mkfs -v my.repo.name
```

When CernVM-FS clients perform a cache cleanup, they treat files from volatile repositories with priority. Such volatile repositories can be useful, for instance, for experiment conditions data.

Compression and Hash Algorithms

Files in the CernVM-FS repository data store are compressed and named according to their compressed content hash. The default settings use DEFLATE (zlib) for compression and SHA-1 for hashing.

CernVM-FS can optionally skip compression of files. This can be beneficial, for instance, if the repository is known to contain already compressed content, such as JPG images or compressed ROOT files. In order to disable compression, set `CVMFS_COMPRESSION_ALGORITHM=none` in the `/etc/cvmfs/repositories.d/$repository/server.conf` file. Client version >= 2.2 is required in order to read uncompressed files.

Instead of SHA-1, CernVM-FS can use RIPEMD-160 or SHAKE-128 (a variant of SHA-3 with 160 output bits) as hash algorithm. In general, we advise not to change the default. In future versions, the default might change from SHA-1 to SHAKE-128. In order to enforce the use of a specific hash algorithm, set `CVMFS_HASH_ALGORITHM=sha1`, `CVMFS_HASH_ALGORITHM=rmd160`, or `CVMFS_HASH_ALGORITHM=shake128` in the `server.conf` file. Client version >= 2.1.18 is required for accessing repositories that use RIPEMD-160. Client version >= 2.2 is required for accessing repositories that use SHAKE-128.

Both compression and hash algorithm can be changed at any point during the repository life time. Existing content will remain untouched, new content will be processed with the new settings.

External Files

Files in a CernVM-FS repository can be marked as *external files*. Externals files are not expected to be served from the HTTP server(s) that provide the file catalogs but from an independent set of HTTP server(s). The idea is for CernVM-FS to be able to provide a directory of files that is already present on an HTTP service. External files are often *grafted*.

While regular files use their content hash as basis for the HTTP URL, external files are expected to be available under their file system name. For instance, a file `/foo/bar` with content hash `0x1234` would be addressed as

```
$HTTP_SERVER_URL/12/34      # as regular file
$HTTP_EXTERNAL_URL/foo/bar  # as external file
```

Note that the content hash of external files is still verified on download. Also note that CernVM-FS by itself does not know or store the location of external files but it must be explicitly set through the client configuration. On the clients, the `CVMFS_EXTERNAL_URL`, `CVMFS_EXTERNAL_HTTP_PROXY` and the other “external” parameters are used to configure the external HTTP servers (see *appendix*).

Files are marked as external data if the `CVMFS_EXTERNAL_DATA` server setting is enabled or if the `cvmfs_server publish -X` option is used. Conversely, if `CVMFS_EXTERNAL_DATA` is set and the `cvmfs_server publish -N` option is used, this particular publish operation will treat its files exceptionally as non-external files.

Confidential Repositories

Repositories can be created with the `-V` options or republished with the `-F` option with a `membership requirement`. Clients that mount repositories with a membership requirement will grant or deny access to the repository based on the decision made by an authorization helper. See Section *Authorization Helpers* for details on authorization helpers.

For instance, a repository can be configured to grant access to a repository only to those users that have a X.509 certificate with a certain DN. Note that the corresponding client-side X.509 authorization helper is not part of CernVM-FS but is provided as a third-party plugin by the Open Science Grid.

A membership requirement makes most sense if the repository is served by an HTTPS server that requires client-side authentication. Note that such repositories cannot be replicated to Stratum 1 servers. Such repositories also cannot benefit from site proxies. Instead, such repositories are either part of a (non CernVM-FS) HTTPS content distribution network or they might be installed for a small number of users that, for example, require access to licensed software.

S3 Compatible Storage Systems

CernVM-FS can store data directly to S3 compatible storage systems, such as Amazon S3, Azure Blob Storage or Ceph. The S3 target bucket needs to be created beforehand, for example with `s3cmd`. The bucket needs to be public for reading and require authorization for writing:

```
# The --configure is optional. For the CERN Ceph S3 instance, for example, use host␣
↪s3.cern.ch and the %(bucket).s3.cern.ch URL template.
s3cmd --configure
export AWS_ACCESS_KEY_ID=<ACCESS KEY>
export AWS_SECRET_ACCESS_KEY=<SECRET KEY>
s3cmd mb s3://<BUCKET NAME>
s3cmd --acl-public setacl s3://<BUCKET NAME>
```

Note: if you use the Minio client, the `download` bucket policy won't work as a bucket policy.

Once the bucket is available, the S3 storage settings are given as parameters to `cvmfs_server mkfs` or `cvmfs_server add-replica`:

```
cvmfs_server mkfs -s /etc/cvmfs/.../mys3.conf \
  -w http://mybucket.s3.amazonaws.com my.repo.name
```

The file "mys3.conf" contains the S3 settings (see :ref: *table below <tab_s3confparameters>*). The "-w" option is used define the S3 server URL, e.g. http://localhost:3128, which is used for accessing the repository's backend storage on S3.

Parameter	Meaning
CVMFS_S3_ACCESS_KEY	S3 account access key
CVMFS_S3_SECRET_KEY	S3 account secret key
CVMFS_S3_HOST	S3 server hostname, e.g. s3.amazonaws.com. The hostname should NOT be prefixed by "http://"
CVMFS_S3_FLAVOR	Set to "azure" if you store files in Microsoft Azure Blob Storage
CVMFS_S3_REGION	The S3 region, e.g. eu-central-1. If specified, AWSv4 authorization protocol is used.
CVMFS_S3_PORT	The port on which the S3 instance is running
CVMFS_S3_BUCKET	S3 bucket name. The repository name is used as a subdirectory inside the bucket.
CVMFS_S3_TIMEOUT	Timeout in seconds for the connection to the S3 server.
CVMFS_S3_MAX_RETRIES	Number of retries for the connection to the S3 server.
CVMFS_S3_MAX_NUMBER_OF_PARALLEL_CONNECTIONS	Number of parallel uploads to the S3 server, e.g. 400
CVMFS_S3_DNS_BUCKETS	Set to false to disable DNS-style bucket URLs (http://<bucket>.<host>/<object>). Enabled by default.
CVMFS_S3_PEEK_BEFORE_PUT	Make PUT requests conditional to a prior HEAD request. Enabled by default.
CVMFS_S3_USE_HTTPS	Allow to use S3 implementation over HTTPS and not over HTTP

Repository Update

Typically a repository publisher does the following steps in order to create a new revision of a repository:

1. Run `cvmfs_server transaction` to switch to a copy-on-write enabled CernVM-FS volume
2. Make the necessary changes to the repository, add new directories, patch certain binaries, …
3. Test the software installation
4. Do one of the following:
    - Run `cvmfs_server publish` to finalize the new repository revision *or*
    - Run `cvmfs_server abort` to clear all changes and start over again

In order to see the current set of staged changes, use the `cvmfs_server diff --worktree` command.

CernVM-FS supports having more than one repository on a single server machine. In case of a multi-repository host, the target repository of a command needs to be given as a parameter when running the `cvmfs_server` utility. Most `cvmfs_server` commands allow for wildcards to do manipulations on more than one repository at once, `cvmfs_server migrate *.cern.ch` would migrate all present repositories ending with `.cern.ch`.

Repository Update Propagation

Updates to repositories won't immediately appear on the clients. For scalability reasons, clients will only regularly check for updates. The frequency of update checks is stored in the repository itself and defaults to 4 minutes. The default can be changed by setting `CVMFS_REPOSITORY_TTL` in the `/etc/cvmfs/repositories.d/$repository/server.conf` file to a new value given in seconds. The value should not fall below 1 minute.

If the repository is replicated to a stratum 1 server (see Chapter *Setting up a Replica Server (Stratum 1)*), replication of the changes needs to finish before the repository time-to-live applies. The status of the replication can be checked by the cvmfs_info utility, like

```
cvmfs_info http://cvmfs-stratum-zero.cern.ch/cvmfs/cernvm-prod.cern.ch
```

The `cvmfs_info` utility can be downloaded as a stand-alone Perl script from the linked github repository.

The `cvmfs_info` utility relies on the repository meta-data as described in Chapter *CernVM-FS Server Meta Information*. It shows timestamp and revision number of the repository on the stratum 0 master server and all replicas, as well as the remaining life time of the repository whitelist and the catalog time-to-live.

Note: The `cvmfs_info` utility queries stratum servers without passing through web proxies. It is not meant to be used on a large-scale by all clients. On clients, the extended attribute `revision` can be used to check for the currently active repository state, like

```
attr -g revision /cvmfs/cernvm-prod.cern.ch
```

Tarball Publishing

Tarballs can be directly published in a repository without the need to extract them first. The `ingest` command can be used to publish the contents of a tarball at a given subdirectory:

```
cvmfs_server ingest --tar_file <tarball.tar> --base_dir <path/where/extract/>
↪<repository name>
```

The optional `--catalog` switch of the `ingest` command is used to automatically create a nested file catalog at the base directory where the tarball is extracted (see *Managing Nested Catalogs*). Note that currently the *.cvmfsdirtab file* does not apply to the ingest command.

The `ingest` command can also be used for the reverse operation of recursively removing a directory tree:

```
cvmfs_server ingest --delete <path/to/delete> <repository name>
```

The `ingest` command internally opens and closes a transaction. Therefore, it can only run if no other transactions are currently open.

Grafting Files

When a repository is updated, new files are checksummed and copied / uploaded to a directory exported to the web. There are situations where this is not optimal - particularly, when *"large-scale" repositories* are used, it may not be pragmatic to copy every file to a single host. In these cases, it is possible to "graft" files by creating a special file containing the necessary publication data. When a graft is encountered, the file is published as if it was present on the repository machine: the repository admin is responsible for making sure the file's data is distributed accordingly.

To graft a file, `foo` to a directory, one must: - Create an empty, zero-length file named `foo` in the directory. - Create a separate graft-file named `.cvmfsgraft-foo` in the same directory.

The `.cvmfsgraft` file must have the following format:

```
size=$SIZE
checksum=$CHECKSUM
chunk_offsets=$OFFSET_1,$OFFSET_2,$OFFSET_3,...
chunk_checksums=$CHECKSUM_1,$CHECKSUM_2,$CHECKSUM_3,...
```

Here, `$SIZE` is the entire file size and `$CHECKSUM` is the file's checksum; the checksums used by this file are assumed to correspond to the algorithm selected at publication time. The offsets `$OFFSET_X` and checksums `$CHECKSUM_X` correspond to the checksum and beginning offset of each chunk in the file. `$OFFSET_1` is always at `0`. Implicitly, the last chunk ends at the end of the file.

To help generate checksum files, the `cvmfs_swissknife graft` command is provided. The `graft` command takes the following options:

Option	Description
-i	Input file to process (- for reading from stdin)
-o	Output location for graft file (optional)
-v	Verbose output (optional)
-Z	Compression algorithm (default: none) (optional)
-c	Chunk size (in MB; default: 32) (optional)
-a	hash algorithm (default: SHA-1) (optional)

This command outputs both the `.cvmfsgraft` file and and zero-length "real" file if `-o` is used; otherwise, it prints the contents of the `.cvmfsgraft` file to `stdout`. A typical invocation would look like this:

```
cat /path/to/some/file | cvmfs_swissknife graft -i - -o /cvmfs/repo.example.com/my_
↪file
```

Template Transactions

In a "template transaction", an existing directory is used as a template for the changes to be published. Open a template transaction with the `-T` option like

```
cvmfs_server transaction -T /foo=/bar
```

The command clones the existing directory /foo to /bar before the transaction becomes available to writing. This can be useful to publish a new directory tree that is almost identical to an existing one, for instance to publish a patch release. Cloning the existing directory tree is a fast, meta-data only operation. Note that template transactions should be used with care – excessive use can quickly explode the repository size with negative consequences such as much increased garbage collection times.

Variant Symlinks

It may be convenient to have a symlink in the repository resolve based on the CVMFS client configuration; this is called a *variant symlink*. For example, in the `oasis.opensciencegrid.org` repository, the OSG provides a default set of CAs at `/cvmfs/oasis.opensciencegrid.org/mis/certificates` but would like to give the sysadmin the ability to override this with their own set of CA certificates.

To setup a variant symlink in your repository, create a symlink as follows inside a repository transaction:

```
ln -s '$(OSG_CERTIFICATES)' /cvmfs/oasis.opensciencegrid.org/mis/certificates
```

Here, the `certificates` symlink will evaluate to the value of the `OSG_CERTIFICATES` configuration variable in the client. If `OSG_CERTIFICATES` is not provided, the symlink resolution will be an empty string. To provide a server-side default value, you can instead do:

```
ln -s '$(OSG_CERTIFICATES:-/cvmfs/oasis.opensciencegrid.org/mis/certificates-real)' /
↪cvmfs/oasis.opensciencegrid.org/mis/certificates
```

Here, the symlink will evaluate to `/cvmfs/oasis.opensciencegrid.org/mis/certificates-real` by default unless the sysadmin sets `OSG_CERTIFICATES` in a configuration file (such as `/etc/cvmfs/config.d/oasis.opensciencegrid.org.local`.

Repository Import

The CernVM-FS server tools support the import of a CernVM-FS file storage together with its corresponding signing keychain. The import functionality is useful to bootstrap a release manager machine for a given file storage.

`cvmfs_server import` works similar to `cvmfs_server mkfs` (described in *Repository Creation*) except it uses the provided data storage instead of creating a fresh (and empty) storage.

During the import it might be necessary to resign the repository's whitelist. Usually because the whitelist's expiry date has exceeded. This operations requires the corresponding masterkey to be available in */etc/cvmfs/keys* or in a masterkeycard. Resigning is enabled by adding `-r` to `cvmfs_server import`.

An import can either use a provided repository keychain placed into */etc/cvmfs/keys* or generate a fresh repository key and certificate for the imported repository. The latter case requires an update of the repository's whitelist to incorporate the newly generated repository key. To generate a fresh repository key add `-t -r` to `cvmfs_server import`.

Refer to Section *Repository Signature* for a comprehensive description of the repository signature mechanics.

Customizable Actions Using Server Hooks

The `cvmfs_server` utility allows release managers to trigger custom actions before and after crucial repository manipulation steps. This can be useful for example for logging purposes, establishing backend storage connections automatically or other workflow triggers, depending on the application.

There are six designated server hooks that are potentially invoked during the *repository update procedure*:

- When running `cvmfs_server transaction`:
  - *before* the given repository is transitioned into transaction mode
  - *after* the transition was successful
- When running `cvmfs_server publish`:
  - *before* the publish procedure for the given repository is started
  - *after* it was published and remounted successfully
- When running `cvmfs_server abort`:
  - *before* the unpublished changes will be erased for the given repository
  - *after* the repository was successfully reverted to the last published state

All server hooks must be defined in a single shell script file called:

```
/etc/cvmfs/cvmfs_server_hooks.sh
```

The `cvmfs_server` utility will check the existence of this script and source it. To subscribe to the described hooks one needs to define one or more of the following shell script functions:

- `transaction_before_hook()`
- `transaction_after_hook()`
- `publish_before_hook()`
- `publish_after_hook()`
- `abort_before_hook()`
- `abort_after_hook()`

The defined functions get called at the specified positions in the repository update process and are provided with the fully qualified repository name as their only parameter (`$1`). Undefined functions automatically default to a NO-OP. An example script is located at `cvmfs/cvmfs_server_hooks.sh.demo` in the CernVM-FS sources.

2.6.6 Maintaining a CernVM-FS Repository

CernVM-FS is a versioning, snapshot-based file system. Similar to versioning systems, changes to /cvmfs/…are temporary until they are committed (`cvmfs_server publish`) or discarded (`cvmfs_server abort`). That allows you to test and verify changes, for instance to test a newly installed release before publishing it to clients. Whenever changes are published (committed), a new file system snapshot of the current state is created. These file system snapshots can be tagged with a name, which makes them *named snapshots*. A named snapshot is meant to stay in the file system. One can rollback to named snapshots and it is possible, on the client side, to mount any of the named snapshots in lieu of the newest available snapshot.

Two named snapshots are managed automatically by CernVM-FS, `trunk` and `trunk-previous`. This allows for easy unpublishing of a mistake, by rolling back to the `trunk-previous` tag.

Integrity Check

CernVM-FS provides an integrity checker for repositories. It is invoked by

```
cvmfs_server check
```

The integrity checker verifies the sanity of file catalogs and verifies that referenced data chunks are present. Ideally, the integrity checker is used after every publish operation. Where this is not affordable due to the size of the repositories, the integrity checker should run regularly.

The checker can also run on a nested catalog subtree. This is useful to follow up a specific issue where a check on the full tree would take a lot of time:

```
cvmfs_server check -s <path to nested catalog mountpoint>
```

Optionally `cvmfs_server check` can also verify the data integrity (command line flag `-i`) of each data object in the repository. This is a time consuming process and we recommend it only for diagnostic purposes.

Named Snapshots

Named snapshots or *tags* are an easy way to organise checkpoints in the file system history. CernVM-FS clients can explicitly mount a repository at a specific named snapshot to expose the file system content published with this tag. It also allows for rollbacks to previously created and tagged file system revisions. Tag names need to be unique for each repository and are not allowed to contain spaces or spacial characters. Besides the actual tag's name they can also contain a free descriptive text and store a creation timestamp.

To mount a specific named snapshot as a client use

```
CVMFS_REPOSITORY_TAG=$tagname
```

Named snapshots are best to use for larger modifications to the repository, for instance when a new major software release is installed. Named snapshots provide the ability to easily undo modifications and to preserve the state of the file system for the future. Nevertheless, named snapshots should not be used excessively. Less than 50 named snapshots are a good number of named snapshots in many cases.

Automatically Generated Tags

By default, new repositories will automatically create a generic tag if no explicit tag is given during publish. The automatic tagging can be turned off using the `-g` option during repository creation or by setting `CVMFS_AUTO_TAG=false` in the /etc/cvmfs/repositories.d/$repository/server.conf file.

The life time of automatic tags can be restriced by the `CVMFS_AUTO_TAG_TIMESPAN` parameter or by the `-G` option to `cvmfs_server mkfs`. The parameter takes a string that the `date` utility can parse, for instance `"4 weeks ago"`. On every publish, automatically generated tags older than the defined threshold are removed.

Creating a Named Snapshot

Tags can be added while publishing a new file system revision. To do so, the -a and -m options for `cvmfs_server publish` are used. The following command publishes a CernVM-FS revision with a new revision that is tagged as "release-1.0":

```
cvmfs_server transaction
# Changes
cvmfs_server publish -a release-1.0 -m "first stable release"
```

Managing Existing Named Snapshots

Management of existing tags is done by using the `cvmfs_server tag` command. Without any command line parameters, it will print all currently available named snapshots. Snapshots can be inspected (`-i <tag name>`), removed (`-r <tag name>`) or created (`-a <tag name> -m <tag description> -h <catalog root hash>`). Furthermore machine readable modes for both listing (`-l -x`) as well as inspection (`-i <tag name> -x`) is available.

Rollbacks

A repository can be rolled back to any of the named snapshots. Rolling back is achieved through the command `cvmfs_server rollback -t release-1.0` A rollback is, like restoring from backups, not something one would do often. Use caution, a rollback is irreversible.

Named Snapshot Diffs

The command `cvmfs_server diff` shows the difference in terms of added, deleted, and modified files and directories between any two named snapshots. It also shows the difference in total number of files and nested catalogs.

Unless named snapshots are provided by the `-s` and `-d` flags, the command shows the difference from the last snapshot ("trunk-previous") to the current one ("trunk").

Note that the command `cvmfs_server diff` shows the changes of the currently active transaction.

Instant Access to Named Snapshots

CernVM-FS can maintain a special directory

```
/cvmfs/${repository_name}/.cvmfs/snapshots
```

through which the contents of all named snapshots is accessible by clients. The directory is enabled and disabled by setting `CVMFS_VIRTUAL_DIR=[true,false]`. If enabled, for every named snapshot `$tag_name` a directory `/cvmfs/${repository_name}/.cvmfs/snapshots/${tag_name}` is maintained, which contains the contents of the repository in the state referenced by the snapshot.

To prevent accidental recursion, the top-level directory `.cvmfs` is hidden by CernVM-FS clients >= 2.4 even for operations that show dot-files like `ls -a`. Clients before version 2.4 will show the `.cvmfs` directory but they cannot recurse into the named snapshot directories.

Branching

In certain cases, one might need to publish a named snapshot based not on the latest revision but based on a previous named snapshot. This can be useful, for instance, if versioned data sets are stored in CernVM-FS and certain files in a past data set needs to be fixed.

In order to publish a branch, use `cvmfs_server checkout` in order to switch to the desired parent branch before starting a transaction. The following example publishes based on the existing snapshot "data-v201708" the new named snapshot "data-v201708-fix01" in the branch "fixes_data-v201708".

```
cvmfs_server checkout -b fixes_data-v201708 -t data-v201708
cvmfs_server transaction
# show that the repository is in a checked-out state
cvmfs_server list
# make changes to /cvmfs/${repository_name}
cvmfs_server publish -a data-v201708-fix01
# show all named snapshots and their branches
cvmfs_server tag -l
# verify that the repository is back on the trunk revision
cvmfs_server list
```

When publishing a checked out state, it is mandatory to specify a tag name. Later, it might be necessary to publish another set of fixes in the same branch. To do so, the command `cvmfs_server checkout -b fixes_data-v201708` checks out the latest named snapshot from the given branch. The command `cvmfs_server checkout` jumps back to the trunk of the repository.

The command `cvmfs_server tag -b` displays the tree of branches and their respective initial revisions. The `-x` switch triggers displaying of the tree in a machines-readable format.

Branching makes most sense for repositories that use the instant snapshot access (see Section *Branching*).

Please note that while CernVM-FS supports branching, it does not support merging of repository snapshots.

Managing Nested Catalogs

CernVM-FS stores meta-data (path names, file sizes, . . .) in file catalogs. When a client accesses a repository, it has to download the file catalog first and then it downloads the files as they are opened. A single file catalog for an entire repository can quickly become large and impractical. Also, clients typically do not need all of the repository's meta-data at the same time. For instance, clients using software release 1.0 do not need to know about the contents of software release 2.0.

With nested catalogs, CernVM-FS has a mechanism to partition the directory tree of a repository into many catalogs. Repository maintainers are responsible for sensible cutting of the directory trees into nested catalogs. They can do so by creating and removing magic files named `.cvmfscatalog`.

For example, in order to create a nested catalog for software release 1.0 in the hypothetical repository experiment.cern.ch, one would invoke

```
cvmfs_server transaction
touch /cvmfs/experiment.cern.ch/software/1.0/.cvmfscatalog
cvmfs_server publish
```

In order to merge a nested catalog with its parent catalog, the corresponding `.cvmfscatalog` file needs to be removed. Nested catalogs can be nested on arbitrary many levels.

Recommendations for Nested Catalogs

Nested catalogs should be created having in mind which files and directories are accessed together. This is typically the case for software releases, but can be also on the directory level that separates platforms. For instance, for a directory layout like

```
/cvmfs/experiment.cern.ch
  |- /software
  |    |- /i686
  |    |    |- 1.0
  |    |    |- 2.0
  |    `    |- common
  |    |- /x86_64
  |    |    |- 1.0
  |    `    |- common
  |- /grid-certificates
  |- /scripts
```

it makes sense to have nested catalogs at

```
/cvmfs/experiment.cern.ch/software/i686
/cvmfs/experiment.cern.ch/software/x86_64
/cvmfs/experiment.cern.ch/software/i686/1.0
/cvmfs/experiment.cern.ch/software/i686/2.0
/cvmfs/experiment.cern.ch/software/x86_64/1.0
```

A nested catalog at the top level of each software package release is generally the best approach because once package releases are installed they tend to never change, which reduces churn and garbage generated in the repository from old catalogs that have changed. In addition, each run only tends to access one version of any package so having a separate catalog per version avoids loading catalog information that will not be used. A nested catalog at the top level of each platform may make sense if there is a significant number of platform-specific files that aren't included in other catalogs.

It could also make sense to have a nested catalog under grid-certificates, if the certificates are updated much more frequently than the other directories. It would not make sense to create a nested catalog under /cvmfs/experiment.cern.ch/software/i686/common, because this directory needs to be accessed anyway whenever its parent directory is needed. As a rule of thumb, a single file catalog should contain more than 1000 files and directories but not contain more than ≈200000 files. See *Inspecting Nested Catalog Structure* how to find catalogs that do not satisfy this recommendation.

Restructuring the repository's directory tree is an expensive operation in CernVM-FS. Moreover, it can easily break client applications when they switch to a restructured file system snapshot. Therefore, the software directory tree layout should be relatively stable before filling the CernVM-FS repository.

Managing Nested Catalogs with .cvmfsdirtab

Rather than managing .cvmfscatalog files by hand, a repository administrator may create a file called .cvmfsdirtab, in the top directory of the repository, which contains a list of paths relative to the top of the repository where .cvmfscatalog files will be created. Those paths may contain shell wildcards such as asterisk (*) and question mark (?). This is useful for specifying patterns for creating nested catalogs as new files are installed. A very good use of the patterns is to identify directories where software releases will be installed. Manually-placed .cvmfscatalog files can still be used along with .cvmfsdirtab.

In addition, lines in .cvmfsdirtab that begin with an exclamation point (!) are shell patterns that will be excluded from those matched by lines without an exclamation point. Empty lines and comment lines starting with a pound sign (#) are ignored. For example a .cvmfsdirtab might contain these lines for the repository of the previous subsection:

```
# Nested catalogs for every platform
/software/*
# Nested catalogs for every version
/software/*/*
! */common
/grid-certificates
```

This will create nested catalogs at

```
/cvmfs/experiment.cern.ch/software/i686
/cvmfs/experiment.cern.ch/software/i686/1.0
/cvmfs/experiment.cern.ch/software/i686/2.0
/cvmfs/experiment.cern.ch/software/x86_64
/cvmfs/experiment.cern.ch/software/x86_64/1.0
/cvmfs/experiment.cern.ch/grid-certificates
```

Note that unlike the regular lines that add catalogs, asterisks in the exclamation point exclusion lines can span the slashes separating directory levels.

Automatic Management of Nested Catalogs

An alternative to .cvmfsdirtab is the automatic catalog generation. This feature automatically generates nested catalogs based on their weight (number of entries). It can be enabled by setting CVMFS_AUTOCATALOGS=true in the server configuration file.

Catalogs are split when their weight is greater than a specified maximum threshold, or removed if their weight is less than a minimum threshold. Automatically generated catalogs contain a .cvmfsautocatalog file (along with the .cvmfscatalog file) in its root directory. User-defined catalogs (containing only a .cvmfscatalog file) always remain untouched. Hence one can mix both manual and automatically managed directory sub-trees.

The following conditions are applied when processing a nested catalog:

- If the weight is greater than CVMFS_AUTOCATALOGS_MAX_WEIGHT, this catalog will be split in smaller catalogs that meet the maximum and minimum thresholds.
- If the weight is less than CVMFS_AUTOCATALOGS_MIN_WEIGHT, this catalog will be merged into its parent.

Both CVMFS_AUTOCATALOGS_MAX_WEIGHT and CVMFS_AUTOCATALOGS_MIN_WEIGHT have reasonable defaults and usually do not need to be defined by the user.

Inspecting Nested Catalog Structure

The following command visualizes the current nested file catalog layout of a repository.

```
cvmfs_server list-catalogs
```

This command also allows problematic nested catalogs to be identified. As stated *here* the recommended maximal file entry count of a single catalog should not exceed ≈200000. One can use the switch `list-catalogs -e` to inspect the current nested catalog entry counts in the repository. Furthermore `list-catalogs -s` will print the file sizes of the catalogs in bytes.

Repository Mount Point Management

CernVM-FS server maintains two mount points for each repository (see *CernVM-FS Server Infrastructure* for details) and needs to keep them in sync with *transactional operations* on the repository.

In rare occasions (for example at reboot of a release manager machine) CernVM-FS might need to perform repair operations on those mount points. As of CernVM-FS 2.2.0 those mount points are not automatically mounted on reboot of the release manager machine anymore. Usually the mount point handling happens automatically and transparently to the user when invoking arbitrary `cvmfs_server` commands.

Nevertheless `cvmfs_server mount <repo name>` allows users to explicitly trigger this repair operation anytime for individual repositories. Mounting all hosted repositories is possible with the `-a` parameter but requires root privileges. If you want to have all hosted repositories mounted after reboot then put `cvmfs_server mount -a` in a boot script, for example in `/etc/rc.local`.

```
# properly mount a specific repository
cvmfs_server mount test.cern.ch

# properly mount all hosted repositories (as root)
sudo cvmfs_server mount -a
```

Syncing files into a repository with cvmfs_rsync

A common method of publishing into CernVM-FS is to first install all the files into a convenient shared filesystem, mount the shared filesystem on the publishing machine, and then sync the files into the repository during a transaction. The most common tool to do the syncing is `rsync`, but `rsync` by itself doesn't have a convenient mechanism for avoiding generated `.cvmfscatalog` and `.cvmfsautocatalog` files in the CernVM-FS repository. Actually the `--exclude` option is good for avoiding the extra files, but the problem is that if a source directory tree is removed, then `rsync` will not remove the corresponding copy of the directory tree in the repository if it contains a catalog, because the extra file remains in the repository. For this reason, a tool called `cvmfs_rsync` is included in the `cvmfs-server` package. This is a small wrapper around `rsync` that adds the `--exclude` options and removes `.cvmfscatalog` and `.cvmfsautocatalog` files from a repository when the corresponding source directory is removed. This is the usage:

```
cvmfs_rsync [rsync_options] srcdir /cvmfs/reponame[/destsubdir]
```

This is an example use case:

```
$ cvmfs_rsync -av --delete /data/lhapdf /cvmfs/cms.cern.ch
```

Migrate File Catalogs

In rare cases the further development of CernVM-FS makes it necessary to change the internal structure of file catalogs. Updating the CernVM-FS installation on a Stratum 0 machine might require a migration of the file catalogs.

It is recommended that `cvmfs_server list` is issued after any CernVM-FS update to review if any of the maintained repositories need a migration. Outdated repositories will be marked as "INCOMPATIBLE" and `cvmfs_server` refuses all actions on these repositories until the file catalogs have been updated.

In order to run a file catalog migration use `cvmfs_server migrate` for each of the outdated repositories. This will essentially create a new repository revision that contains the exact same file structure as the current revision. However, all file catalogs will be recreated from scratch using the updated internal structure. Note that historic file catalogs of all previous repository revisions stay untouched and are not migrated.

After `cvmfs_server migrate` has successfully updated all file catalogs repository maintenance can continue as usual.

Change File Ownership on File Catalog Level

CernVM-FS tracks the UID and GID of all contained files and exposes them through the client to all using machines. Repository maintainers should keep this in mind and plan their UID and GID assignments accordingly.

Repository operation might occasionally require to bulk-change many or all UIDs/GIDs. While this is of course possible via `chmod -R` in a normal repository transaction, it is cumbersome for large repositories. We provide a tool to quickly do such adaption on *CernVM-FS catalog level* using UID and GID mapping files:

```
cvmfs_server catalog-chown -u <uid map> -g <gid map> <repo name>
```

Both the UID and GID map contain a list of rules to apply to each file meta data record in the CernVM-FS catalogs. This is an example of such a rules list:

```
# map root UID/GID to 1001
0 1001

# swap UID/GID 1002 and 1003
1002 1003
1003 1002

# map everything else to 1004
* 1004
```

Note that running `cvmfs_server catalog-chown` produces a new repository revision containing *CernVM-FS catalogs* with updated UIDs and GIDs according to the provided rules. Thus, previous revisions of the CernVM-FS repository will *not* be affected by this update.

2.6.7 Publisher Statistics

The CernVM-FS server tools now record a number of metrics related to the publication and garbage collection processes. By default, the database is located at `/var/spool/cvmfs/<REPOSITORY_NAME>/stats.db`, but the location can be changed through the `CVMFS_STATISTICS_DB` parameter.

At the end of each successful transaction, a new row is inserted into the `publish_statistics` table of the database, with the following columns:

Field	Type
publish_id	Integer
start_time	Text (timestamp format: *YYYY-MM-DD hh-mm-ss*)
finished_time	Text (timestamp format: *YYYY-MM-DD hh-mm-ss*)
files_added	Integer
files_removed	Integer
files_changed	Integer
duplicated_files	Integer
directories_added	Integer
directories_removed	Integer
directories_changed	Integer
sz_bytes_added	Integer
sz_bytes_removed	Integer
sz_bytes_uploaded	Integer

By setting `CVMFS_PRINT_STATISTICS=true`, in addition to being saved in the database, the metrics are printed to the console at the end of the `cvmfs_server publish` or `cvmfs_server ingest` commands.

When the garbage collector is run, a new row is inserted into the `gc_statistics` table, with the following columns:

Field	Type
gc_id	Integer
start_time	Text (timestamp format: *YYYY-MM-DD hh-mm-ss*)
finished_time	Text (timestamp format: *YYYY-MM-DD hh-mm-ss*)
n_preserved_catalogs	Integer
n_condemned_catalogs	Integer
n_condemned_objects	Integer
sz_condemned_bytes (*)	Integer

(*) Disabled by default due to the non-negligible computation cost. Can be enabled with `CVMFS_EXTENDED_GC_STATS=true`

Entries in the statistics database are kept, by default, for 1 year. This interval can be changed by the `CVMFS_STATS_DB_DAYS_TO_KEEP` parameter.

The contents of any table (`publish_statistics`, `gc_statistics`, or `properties`) in the database can be exported to text using:

```
# cvmfs_server print-stats [-t <TABLE_NAME>] <REPO_NAME>
```

If the `-t` argument is omitted, the `publish_statistics` table is exported.

Two database files can be merged as follows:

```
# cvmfs_server merge-stats [-o <OUTPUT_DB>] <DB_FILE_1> <DB_FILE_2>
```

The merge can only take place if the two database files come from the same repository and have the same schema version.

By setting `CVMFS_UPLOAD_STATS_DB=true`, the statistics database together with a web page with relevant plots will be published to the stratum 0 `/stats` location. This provides a lightweight monitoring for repository maintainers.

2.6.8 Repository Garbage Collection

Since CernVM-FS is a versioning file system it is following an insert-only policy regarding its backend storage. When files are deleted from a CernVM-FS repository, they are not automatically deleted from the underlying storage. Therefore legacy revisions stay intact and usable forever (cf. *Named Snapshots*) at the expense of an ever-growing storage volume both on the Stratum 0 and the Stratum 1s.

For this reason, applications that frequently install files into a repository and delete older ones - for example the output from nightly software builds - might quickly fill up the repository's backend storage. Furthermore these applications might actually never make use of the aforementioned long-term revision preservation rendering most of the stored objects "garbage".

CernVM-FS supports garbage-collected repositories that automatically remove unreferenced data objects and free storage space. This feature needs to be enabled on the Stratum 0 and automatically scans the repository's catalog structure for unreferenced objects both on the Stratum 0 and the Stratum 1 installations on every publish respectively snapshot operation.

Garbage Sweeping Policy

The garbage collector of CernVM-FS is using a mark-and-sweep algorithm to detect unused files in the internal catalog graph. Revisions that are referenced by named snapshots (cf. *Named Snapshots*) or that are recent enough are preserved while all other revisions are condemned to be removed. By default this time-based threshold is *three days* but can be changed using the configuration variable `CVMFS_AUTO_GC_TIMESPAN` both on Stratum 0 and Stratum 1. The value of this variable is expected to be parseable by the `date` command, for example `3 days ago` or `1 week ago`.

Enabling Garbage Collection

Creating a Garbage Collectable Repository

Repositories can be created as *garbage-collectable* from the start by adding `-z` to the `cvmfs_server mkfs` command (cf. *Repository Creation*). It is generally recommended to also add `-g` to switch off automatic tagging in a garbage collectable repository. For debugging or bookkeeping it is possible to log deleted objects into a file by setting `CVMFS_GC_DELETION_LOG` to a writable file path.

Enabling Garbage Collection on an Existing Repository (Stratum 0)

Existing repositories can be reconfigured to be garbage collectable by adding `CVMFS_GARBAGE_COLLECTION=true` and `CVMFS_AUTO_GC=true` to the `server.conf` of the repository. Furthermore it is recommended to switch off automatic tagging by setting `CVMFS_AUTO_TAG=false` for a garbage collectable repository. The garbage collection will be enabled with the next published transaction and will run every once in a while after a publish operation. Alternatively, `CVMFS_AUTO_GC=false` may be set and `cvmfs_server gc` run from cron at a time when no publish operations will be happening; garbage collection and publish operations cannot happen at the same time.

Enabling Garbage Collection on an Existing Replication (Stratum 1)

In order to use automatic garbage collection on a stratum 1 replica, set `CVMFS_AUTO_GC=true` in the `server.conf` file of the stratum 1 installation. This will run the garbage collection every once in a while after a snapshot. It will only work if the upstream stratum 0 repository has garbage collection enabled.

Alternatively, all garbage collectable repositories can be automatically collected in turn separately from snapshots. See *Maintenance processes*.

Frequency of the Automatic Garbage Collection

If `CVMFS_AUTO_GC=true` is set, the parameter `CVMFS_AUTO_GC_LAPSE` controls how frequently automatic garbage collection is executed. By default, `CVMFS_AUTO_GC_LAPSE` is set to `1 day ago`. If, on publish or snapshot, the last manual or automatic garbage collection is farther in the past then the given threshold, garbage collection will run. Otherwise it is skipped.

2.6.9 Limitations on Repository Content

Because CernVM-FS provides what appears to be a POSIX filesystem to clients, it is easy to think that it is a general purpose filesystem and that it will work well with all kinds of files. That is not the case, however, because CernVM-FS is optimized for particular types of files and usage. This section contains guidelines for limitations on the content of repositories for best operation.

Data files

First and foremost, CernVM-FS is designed to distribute executable code that is shared between a large number of jobs that run together at grid sites, clouds, or clusters. Worker node cache sizes and web proxy bandwidth are generally engineered to accommodate that application. The total amount read per job is expected to be roughly limited by the amount of RAM per job slot. The same files are also expected to be read from the worker node cache multiple times by similar jobs, and read from a caching web proxy by multiple worker nodes.

If there are data files distributed by CernVM-FS that follow similar access patterns and size limits as executable code, it will probably work fine. In addition, if there are files that are larger but read slowly throughout long jobs, as opposed to all at once at the beginning, that can also work well if the same files are read by many jobs. That is because web proxies have to be engineered for handling bursts at the beginning of jobs and so they tend to be lightly loaded a majority of the time.

As a general rule of thumb, calculate the maximum rate at which jobs typically start and limit the amount of data that might be read from a web proxy to around 100 MB/s per thousand jobs, assuming a reasonable amount of overlap of jobs on the same worker nodes. Also, limit the amount of data that will be put into any one worker node cache to around 5 GB. Of course, if you have a special arrangement with particular sites to have large caches and bandwidths available, these limits can be made higher at those sites. Web proxies may also need to be engineered with faster disks if the data causes their cache hit ratios to be reduced.

If you need to publish files with much larger working set sizes than a typical software environment, refer to *large-scale repositories* and alien caches. Using an alien cache is a good way to distribute large data sets when multiple users on the cluster are accessing the same data files.

Also, keep in mind that the total amount of data distributed is not unlimited. The files are stored and distributed compressed, and files with the same content stored in multiple places in the same repository are collapsed to the same file in storage, but the storage space is used not only on the original repository server, it is also replicated onto multiple Stratum 1 servers. Generally if only executable code is distributed, there is no problem with the space taken on Stratum 1s, but if many large data files are distributed they may exceed the Stratum 1 storage capacity. Data files also tend to not compress as well, and that is especially the case of course if they are already compressed before installation.

Tarballs, zip files, and other archive files

If the contents of a tarball, zip file, or some other type of archive file is desired to be distributed by CernVM-FS, it is usually better to first unpack it into its separate pieces first. This is because it allows better sharing of content between multiple releases of the file; some pieces inside the archive file might change and other pieces might not in the next release, and pieces that don't change will be stored as the same file in the repository. CernVM-FS will compress the content of the individual pieces, so even if there's no sharing between releases it shouldn't take much more space.

File permissions

Care should be taken to make all the files in a repository readable by "other". This is because permissions on files in the original repository are generally the same as those seen by end clients, except the files are owned by the "cvmfs" user and group. The write permissions are ignored by the client since it is a read-only filesystem. However, unless the client has set

```
CVMFS_CHECK_PERMISSIONS=no
```

(and most do not), unprivileged users will not be able to read files unless they are readable by "other" and all their parent directories have at least "execute" permissions. It makes little sense to publish files in CernVM-FS if they won't be able to be read by anyone.

Hardlinks

CernVM-FS breaks hardlinks on publishing into multiple, independent regular files.

2.6.10 Configuration Recommendation by Use Case

The default configuration of a fresh CernVM-FS repository are tuned for production software repositories and maximum compatibility and safety. For other typical use cases, the configuration should be adapted.

General Recommendations

Unless an older client base needs to be supported, we recommend to the following configuration changes:

```
CVMFS_AUTO_TAG_TIMESPAN="2 weeks ago"
CVMFS_HASH_ALGORITHM=shake128
```

These changes make unreferenced objects older than two weeks subject to garbage collection (without enabling garbage collection) and uses the more future-proof SHA-3 derived content hash algorithm.

Multi-Tenant Repositories

For repositories that are edited by several, possibly unexperienced users, we suggest the following configuration settings:

```
CVMFS_AUTOCATALOGS=true
CVMFS_ENFORCE_LIMITS=true
CVMFS_FORCE_REMOUNT_WARNING=false
```

This will, in addition to manually created nested catalogs, keep the maximum file catalog size small and enforce the limit on maximum file sizes. It will also prevent forced remounts from sending a broadcast message to all users.

Repositories for Software "Nightly Builds"

Repositories containing the result of "nightly builds" are usually subject to a lot of churn and accumulate unreferenced objects quickly. We recommend to set

```
CVMFS_AUTO_TAG=false
CVMFS_GARBAGE_COLLECTION=true
CVMFS_AUTO_GC=true
```

in order to activate garbage collection and to turn off CernVM-FS' versioning (provided that the content on such repositories is ephemeral). Instead of autmatic garbage collection, one can also install a regular cron job running `cvmfs_server gc -af`, or the nightly build script should be updated to invoke `cvmfs_server gc <repo name>`.

Repositories for (Conditions) Data

Repositories containing data sets (cf. *Large-Scale Data CernVM-FS*) should start with the following base configuration

```
CVMFS_COMPRESSION_ALGORITHM=none
CVMFS_FILE_MBYTE_LIMIT= >> larger than expected maximum file size
CVMFS_VIRTUAL_DIR=true
```

provided that data files are already compressed and that access to previous file system revisions on client-side is desired.

Repositories for Container Images

Repositories containing Linux container image contents (that is: container root file systems) should use overlayfs as a union file system and have the following configuration:

```
CVMFS_INCLUDE_XATTRS=true
CVMFS_VIRTUAL_DIR=true
```

Extended attributes of files, such as file capabilities and SElinux attributes, are recorded. And previous file system revisions can be accessed from the clients.

2.7 CernVM-FS Server Meta Information

The CernVM-FS server automatically maintains both global and repository-specific meta information as JSON data. Release manager machines keep a list of hosted Stratum0 and Stratum1 repositories and user-defined administrative meta information.

Furthermore each repository contains user-maintained signed meta-information that gets replicated to Stratum1 servers automatically.

2.7.1 Global Meta Information

This JSON data provides information about the CernVM-FS server itself. A list of all repositories (both Stratum0 and Stratum1) hosted at this specific server is automatically generated and can be accessed here:

```
http://<server base URL>/cvmfs/info/v1/repositories.json
```

Furthermore there might be user-defined information like the administrator's name, contact information and an arbitrary user-defined JSON portion here:

```
http://<server base URL>/cvmfs/info/v1/meta.json
```

Using the `cvmfs_server` utility, an administrator can edit the user-defined portion of the data with a text editor (cf. `$EDITOR`)s:

```
cvmfs_server update-info
```

Note that the `cvmfs_server` package requires the `jq` utility, which validates CVMFS JSON data.

Below are *examples* of both the repository list and user-defined JSON files.

2.7.2 Repository Specific Meta Information

Each repository contains a JSON object with repository-specific meta data. The information is maintained by the repository's owner on the Stratum0 release manager machine. It contains the maintainer's contact information, a description of the repository's content, the recommended Stratum 0 URL and a list of recommended Stratum 1 replica URLs. Furthermore it provides a custom JSON region for arbitrary information.

Note that this JSON file is stored inside CernVM-FS's backend data structure and gets replicated to Stratum1 servers automatically.

Editing is done per repository using the `cvmfs_server` utility. As with the *global meta information* `cvmfs_server` uses `jq` to validate edited JSON information before storing it:

```
cvmfs_server update-repoinfo <repo name>
```

Besides the interactive editing (cf. `$EDITOR`) one can specify a file path that should be stored as the repository's meta information:

```
cvmfs_server update-repoinfo -f <path to JSON file> <repo name>
```

An example of a repository-specific meta information file can be found in *the section below*.

2.7.3 Examples

/cvmfs/info/v1/meta.json

```
{
  "administrator" : "Your Name",
  "email"         : "you@organisation.org",
  "organisation"  : "Your Organisation",

  "custom" : {
    "_comment" : "Put arbitrary structured data here"
  }
}
```

/cvmfs/info/v1/repositories.json

```
{
  "schema"       : 1,
  "repositories" : [
    {
      "name"  : "atlas.cern.ch",
      "url"   : "/cvmfs/atlas.cern.ch"
    },
    {
      "name"  : "cms.cern.ch",
      "url"   : "/cvmfs/cms.cern.ch"
    }
  ],
  "replicas" : [
    {
      "name"  : "lhcb.cern.ch",
      "url"   : "/cvmfs/lhcb.cern.ch"
    }
```

(continues on next page)

(continued from previous page)

```
  ]
}
```

Repository Specific Meta Information

```
{
  "administrator" : "Your Name",
  "email"         : "you@organisation.org",
  "organisation"  : "Your Organisation",
  "description"   : "Repository content",
  "url"           : "https://www.example.com/",
  "recommended-stratum0"  : "http://cvmfs-s0.example.com/cvmfs/repo.example.com",
  "recommended-stratum1s" : [ "http://cvmfs-s1-a.example.com/cvmfs/repo.example.com",
↪"http://cvmfs-s1-b.example.com/cvmfs/repo.example.com" ],

  "custom" : {
    "_comment" : "Put arbitrary structured data here"
  }
}
```

2.8 Setting up a Replica Server (Stratum 1)

While a CernVM-FS Stratum 0 repository server is able to serve clients directly, a large number of clients is better be served by a set of Stratum 1 replica servers. Multiple Stratum 1 servers improve the reliability, reduce the load, and protect the Stratum 0 master copy of the repository from direct accesses. Stratum 0 server, Stratum 1 servers and the site-local proxy servers can be seen as content distribution network. The *figure below* shows the situation for the repositories hosted in the cern.ch domain.

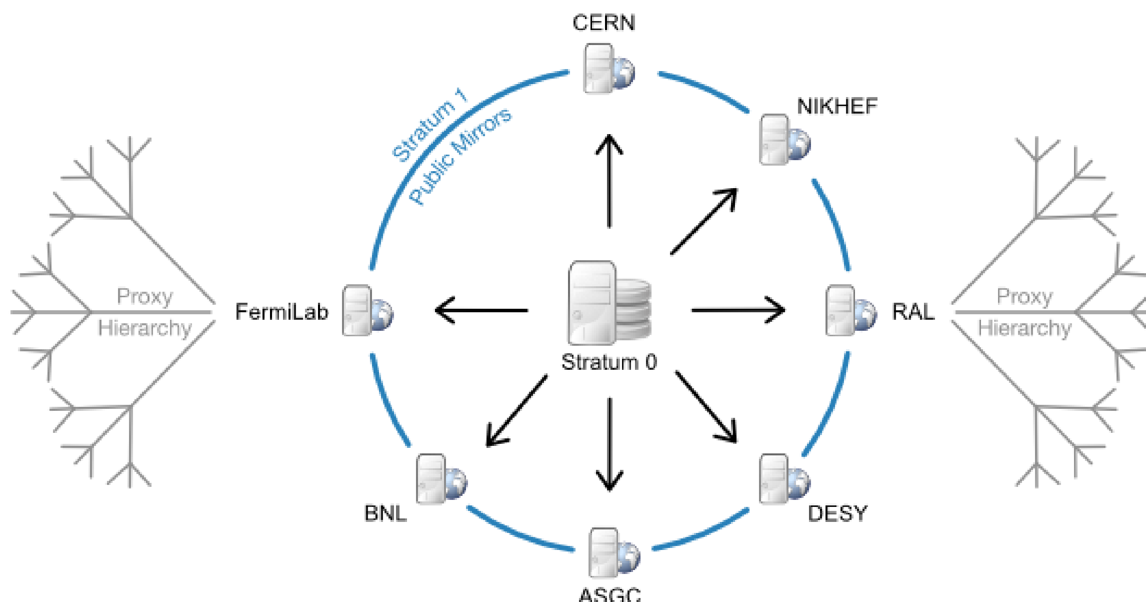

Fig. 4: CernVM-FS content distribution network for the cern.ch domain: Stratum1 replica servers are located in Europe, the U.S. and Asia. One protected read/write instance (Stratum 0) is feeding up the public, distributed mirror servers. A distributed hierarchy of proxy servers fetches content from the closest public mirror server.

A Stratum 1 server is a standard web server that uses the CernVM-FS server toolkit to create and maintain a mirror of a CernVM-FS repository served by a Stratum 0 server. To this end, the `cvmfs_server` utility provides the `add-replica` command. This command will register the Stratum 0 URL and prepare the local web server. Periodical synchronization has to be scheduled, for instance with `cron`, using the `cvmfs_server snapshot -a` command. The advantage over general purpose mirroring tools such as rSync is that all CernVM-FS file integrity verifications mechanisms from the Fuse client are reused. Additionally, by the aid of the CernVM-FS file catalogs, the `cvmfs_server` utility knows beforehand (without remote listing) which files to transfer.

In order to prevent accidental synchronization from a repository, the Stratum 0 repository maintainer has to create a `.cvmfs_master_replica` file in the HTTP root directory. This file is created by default when a new repository is created. Note that replication can thrash caches that might exist between Stratum 1 and Stratum 0. A direct connection is therefore preferable.

2.8.1 Recommended Setup

The vast majority of HTTP requests will be served by the site's local proxy servers. Being a publicly available service, however, we recommend to install a Squid frontend in front of the Stratum 1 web server.

We suggest the following key parameters:

Storage
RAID-protected storage. The `cvmfs_server` utility should have low latency to the storage because it runs a large number of system calls (`stat()`) against it. For the local storage backends ext3/4 filesystems are preferred (rather than XFS).

Web server
A standard Apache server. Directory listing is not required. In addition, it is a good practice to exclude search engines from the replica web server by an appropriate robots.txt. The webserver should be close to the storage in terms of latency.

Squid frontend
Squid should be used as a frontend to Apache, configured as a reverse proxy. It is recommended to run it on the same machine as Apache instead of a separate machine, to reduce the number of points of failure. In that case caching can be disabled for the data (since there's no need to store it again on the same disk), but caching is helpful for the responses to geo api calls. Using a squid is also helpful for participating in shared monitoring such as the WLCG Squid Monitor.

Alternatively, separate Squid server machines may be configured in a round-robin DNS and each forward to the Apache server, but note that if any of them are down the entire service will be considered down by CernVM-FS clients. A front end hardware load balancer that quickly takes a machine that is down out of service would help reduce the impact.

High availability
On the subject of availability, note that it is not advised to use two separate complete Stratum 1 servers in a single round-robin service because they will be updated at different rates. That would cause errors when a client sees an updated catalog from one Stratum 1 but tries to read corresponding data files from the other that does not yet have the files. Different Stratum 1s should either be separately configured on the clients, or a pair can be configured as a high availability active/standby pair using the cvmfs-contrib cvmfs-hastratum1 package. An active/standby pair can also be managed by switching a DNS name between two different servers.

DNS cache
The geo api on a Stratum 1 does DNS lookups. It caches lookups for 5 minutes so the DNS server load does not tend to be severe, but we still recommend installing a DNS caching mechanism on the machine such as `dnsmasq` or `bind`. We do not recommend `nscd` since it does not honor the DNS Time-To-Live protocol.

2.8.2 Apache Configuration

In general the `cvmfs_server` utility automatically manages the Apache configuration. However, for systems based on Red Hat Enterprise Linux 7 it is recommended that heavily used Stratum 1s disable the "prefork" Multi-Process Module (MPM) and instead use the "worker" or "event" MPM which perform much better under heavy load because they work with multiple threads per process. That can be done by changing which module is uncommented in `/etc/httpd/conf.modules.d/00-mpm.conf`. The "event" MPM is the default on Red Hat Enterprise Linux 8.

2.8.3 Squid Configuration

If you participate in the Open Science Grid (OSG) or the European Grid Infrastructure (EGI), you are encouraged to use their distribution of squid called frontier-squid. It is kept up to date with the latest squid bug fixes and has features for easier upgrading and monitoring. Step-by-step instructions for setting it up with a Stratum 1 is available in the **`OSG documentation https://opensciencegrid.org/docs/other/install-cvmfs-stratum1/#configuring-frontier-squid`**_.

Otherwise, a *squid* package is available in most Linux operating systems. The Squid configuration differs from the site-local Squids because the Stratum 1 Squid servers are transparent to the clients (*reverse proxy*). As the expiry rules are set by the web server, Squid cache expiry rules remain unchanged.

The following lines should appear accordingly in `/etc/squid/squid.conf`:

```
http_port 8000 accel
http_access allow all
cache_peer <APACHE_HOSTNAME> parent <APACHE_PORT> 0 no-query originserver

cache_mem <MEM_CACHE_SIZE> MB
cache_dir ufs /var/spool/squid <DISK_CACHE_SIZE in MB> 16 256
maximum_object_size 1024 MB
maximum_object_size_in_memory 128 KB
```

Note that `http_access allow all` has to be inserted before (or instead of) the line `http_access deny all`. If Apache is running on the same host, the `APACHE_HOSTNAME` will be `localhost`. Also, in that case there is not a performance advantage for squid to cache files that came from the same machine, so you can configure squid to not cache files. Do that with the following lines:

```
acl CVMFSAPI urlpath_regex ^/cvmfs/[^/]*/api/
cache deny !CVMFSAPI
```

Then the squid will only cache API calls. You can then set `MEM_CACHE_SIZE` and `DISK_CACHE_SIZE` quite small. Even if squid is configured to cache everything it is best to keep `MEM_CACHE_SIZE` small, because it is generally better to leave as much RAM to the operating system for file system caching as possible.

Check the configuration syntax by `squid -k parse`. Create the hard disk cache area with `squid -z`. In order to make the increased number of file descriptors effective for Squid, execute `ulimit -n 8192` prior to starting the squid service.

The Squid also needs to respond to port 80, but Squid might not have the ability to directly listen there if it is run unprivileged, plus Apache listens on port 80 by default. Direct external port 80 traffic to port 8000 with the following command:

```
iptables -t nat -A PREROUTING -p tcp -m tcp --dport 80 -j REDIRECT --to-ports 8000
```

If IPv6 is supported, do the same command with `ip6tables`. This will leave localhost traffic to port 80 going directly to Apache, which is good because `cvmfs_server` uses that and it doesn't need to go through squid.

Note: Port 8000 might be assigned to `soundd`. On SElinux systems, this assignment must be changed to the HTTP service by `semanage port -m -t http_port_t -p tcp 8000`. The `cvmfs-server` RPM for EL7 executes this command as a post-installation script.

2.8.4 Geo API Setup

One of the essential services supplied by Stratum 1s to CernVM-FS clients is the Geo API. This enables clients to share configurations worldwide while automatically sorting Stratum 1s geographically to prioritize connecting to the closest ones. This makes use of a GeoIP database from Maxmind that translates IP addresses of clients to longitude and latitude.

The database is free, but the Maxmind End User License Agreement requires that each user of the database sign up for an account and promise to update the database to the latest version within 30 days of when they issue a new version. The signup process will end with giving you a License Key. The `cvmfs_server add-replica` and `snapshot` commands will take care of automatically updating the database if you put a line like the following in `/etc/cvmfs/server.local`, replacing `<license key>` with the key you get from the signup process:

```
CVMFS_GEO_LICENSE_KEY=<license key>
```

To keep the key secret, set the mode of `/etc/cvmfs/server.local` to 600. You can test that it works by running `cvmfs_server update-geodb`.

Alternatively, if you have a separate mechanism of installing and updating the Geolite2 City database file, you can instead set `CVMFS_GEO_DB_FILE` to the full path where you have installed it. If the path is `NONE`, then no database will be required, but note that this will break the client Geo API so only use it for testing, when the server is not used by production clients. If the database is installed in the default directory used by Maxmind's own geoipupdate tool, `/usr/share/GeoIP`, then `cvmfs_server` will use it from there and neither variable needs to be set.

Normally repositories on Stratum 1s are created owned by root, and the `cvmfs_server snapshot` command is run by root. If you want to use a different user id while still using the builtin mechanism for updating the geo database, change the owner of `/var/lib/cvmfs-server/geo` and `/etc/cvmfs/server.local` to the user id.

The builtin geo database update mechanism normally checks for updates once a week on Tuesdays but can be controlled through a set of variables defined in `cvmfs_server` beginning with `CVMFS_UPDATEGEO_`. Look in the `cvmfs_server` script for the details. An update can also be forced at any time by running `cvmfs_server update-geodb`.

2.8.5 Monitoring

The `cvmfs_server` utility reports status and problems to `stdout` and `stderr`.

For the web server infrastructure, we recommend cvmfs-servermon which watches for problems in every repository's `.cvmfs_status.json` status file.

In order to tune the hardware and cache sizes, keep an eye on the Squid server's CPU and I/O load.

Keep an eye on HTTP 404 errors. For normal CernVM-FS traffic, such failures should not occur. Traffic from CernVM-FS clients is marked by an `X-CVMFS2` header.

2.8.6 Maintenance processes

If any replicated repositories have Garbage Collection enabled, the Stratum 1 also needs to run garbage collection in order to prevent the disk space usage from growing rapidly. Run `cvmfs_server gc -af` periodically (e.g daily or weekly) from cron to run garbage collection on all repositories that have garbage collection enabled. Logs will go into /var/log/cvmfs/gc.log.

In addition, over time problems can show up with a small percentage of files stored on a large Stratum 1. Run `cvmfs_server check -a` daily from cron to start a check process. On a large Stratum 1 it will run for many days, but only with a single thread so it is not very intrusive. If another check is still in process a new one will not start. Each repository by default will only be checked at most once every 30 days. Logs will go into /var/log/cvmfs/checks.log and problems will be recorded in a repository's `.cvmfs_status.json`.

2.9 The CernVM-FS Repository Gateway and Publishers

This page describes the distributed CernVM-FS publication architecture, composed of a repository gateway machine and separate publisher machines.

2.9.1 Glossary

Publisher

A machine running the CernVM-FS server tools which can publish to a number of repositories, using a repository gateway as mediator.

The resource-intensive parts of the publication operation take place here: compressing and hashing the files which are to be added or modified. The processed files are then packed together and sent to the gateway to be inserted into the repository and made available to clients.

Repository gateway

This machine runs the `cvmfs-gateway` application. It is the sole entity able to write to the authoritative storage of the managed repositories, either by mounting the storage volume or through an S3 API.

The role of the gateway is to mediate access to a set of repositories by assigning exclusive leases for specific repository sub-paths to different publisher machines. The gateway receives payloads from publishers, in the form of object packs, which it processes and writes to the repository storage. Its final task is to rebuild the catalogs and repository manifest of the modified repositories at the end of a successful publication transaction.

2.9.2 Repository gateway configuration

Install the `cvmfs-gateway` package on the gateway machine. Packages for various platforms are available for download here.

When the CernVM-FS client and server packages are also installed and set up as a stratum 0, it's possible to use the gateway machine as a master publisher (for example to perform some initialization operations on a repository, before a separate publisher machine is set up). To avoid any possible repository corruption, the gateway application should always be stopped before starting a local repository transaction on the gateway machine.

With the gateway application installed, create the repository which will be used for the rest of this guide:

```
# cvmfs_server mkfs -o root test.cern.ch
```

Create an API key file for the new repo (replace <KEY_ID> and <SECRET> with actual values):

```
# cat <<EOF > /etc/cvmfs/keys/test.cern.ch.gw
plain_text <KEY_ID> <SECRET>
EOF
# chmod 600 /etc/cvmfs/keys/test.cern.ch.gw
```

Since version 1.0 of `cvmfs-gateway`, the repository and key configuration have been greatly simplified. If an API key file is present at the conventional location (`/etc/cvmfs/keys/<REPOSITORY_NAME>.gw`), it will be used by default as the key for that repository. The repository configuration file only needs to specify which repositories are to be handled by the application:

```
# cat <<EOF > /etc/cvmfs/gateway/repo.json
{
  "version": 2,

  "repos": [
    "test.cern.ch"
  ]
}
EOF
```

The `"version": 2` property enables the use of the improved configuration syntax. If this property is omitted, the parser will interpret the file using the legacy configuration syntax, maintaining compatibility with existing configuration files (see *Legacy repository configuration syntax*). The *Advanced repository configuration* section shows how to implement more complex key setups.

In addition to `repo.json`, the `user.json` configuration file contains runtime parameters for the gateway application. The most important are:

- `max_lease_time` - the maximum duration, in seconds, of an acquired lease
- `port` - the TCP port on which the gateway application listens, 4929 by default (the legacy name for this option is "fe_tcp_port")
- `num_receivers` - the number of parallel `cvmfs_receiver` worker processes to be spawned. Default value is 1, and it should not be increased beyond the number of available CPU cores (the legacy name of this option is the `size` entry in the `receiver_config` map).

To access the gateway service API, the specified `port` needs to be open in the firewall. If the gateway machine also serves as a repository stratum 0 (i.e. the repository is created with "local" upstream), then the port on which httpd listens (80 by default) also needs to be open for TCP.

Note: The gateway service receives data from publishers via HTTP transport. However, since the gateway and publisher have a shared secret (the API key), it is not strictly necessary to use TLS certificates and HTTPS to secure the connection to the gateway. Instead, to ensure the integrity and authenticity of content during the publishing process, a hash-based message authentication code (HMAC) is produced by a publisher, and verified by the gateway.

Finally, to start the gateway application, use `systemctl` if systemd is available:

```
# systemctl start cvmfs-gateway.service
```

otherwise use the service command:

```
# service cvmfs-gateway start
```

Note that in order to apply any gateway configuration changes, including changes to the API keys, the gateway service must be restarted.

If systemd is available, the application logs can be consulted with:

```
# journalctl -u cvmfs-gateway
```

Additional log files may also be found in `/var/log/cvmfs-gateway` and `/var/log/cvmfs-gateway-runner`.

Running under a different user

By default, the `cvmfs-gateway` application is run as root. An included systemd service template file allows running it as an arbitrary user:

```
# systemctl start cvmfs-gateway@<USER>
```

To consult the logs of the application instance running as *<USER>*, run:

```
# journalctl -u cvmfs-gateway@<USER>
```

2.9.3 Publisher configuration

This section describes how to set up a publisher for a specific CVMFS repository. The precondition is a working gateway machine where the repository has been created as a Stratum 0.

Example procedure

- The gateway machine is `gateway.cern.ch`.
- The publisher is `publisher.cern.ch`.
- The new repository's fully qualified name is `test.cern.ch`.
- The repository's public key (RSA) is `test.cern.ch.pub`.
- The repository's public key (encoded as a X.509 certificate) is `test.cern.ch.crt`.
- The gateway API key is `test.cern.ch.gw`.
- The gateway application is running on port 4929 at the URL `http://gateway.cern.ch:4929/api/v1`.
- The three key files for the repository (.pub, .crt, and .gw) have been copied from the gateway machine onto the publisher machine, in the directory `/tmp/test.cern.ch_keys/`.

To make the repository available for writing on `publisher.cern.ch`, run the following command on that machine as a non-root user with sudo access:

```
$ sudo cvmfs_server mkfs -w http://gateway.cern.ch/cvmfs/test.cern.ch \
                         -u gw,/srv/cvmfs/test.cern.ch/data/txn,http://gateway.cern.
↪ch:4929/api/v1 \
                         -k /tmp/test.cern.ch_keys -o `whoami` test.cern.ch
```

At this point, it's possible to start writing into the repository from the publisher machine:

```
$ cvmfs_server transaction test.cern.ch
```

Alternatively, to take advantage of the gateway functionality which allows concurrent transactions on different paths of a repository, or fine-grained permission to only publish changes in certain paths, you can request a publishing lease that is scoped to a subdirectory of the repository by starting a transaction like this:

```
$ cvmfs_server transaction test.cern.ch/example/path
```

Then to commit the changes to the repository and publish:

```
$ cvmfs_server publish
```

2.9.4 Querying the gateway machine

The configuration and current state of the gateway application can be queried using standard HTTP requests. A "GET" request to the "repos" endpoint returns the key configuration for all the repositories:

```
$ curl http://example.gateway.org:4929/api/v1/repos | jq

{
  "data": {
    "example.repo.org": {
      "key1": "/"
    }
  },
  "status": "ok"
}
```

The configuration of a single repository can also be obtained:

```
$ curl http://example.gateway.org:4929/api/v1/repos/example.repo.org | jq

{
  "data": {
    "key1": "/"
  },
  "status": "ok"
}
```

The list of current active leases can be obtained as follows:

```
$ curl http://example.gateway.org:4929/api/v1/leases | jq

{
  "data": {
    "example.repo.org/sub/dir/1": {
      "key_id": "key1",
      "expires": "2019-05-09 23:10:31.730136676 +0200 CEST"
    },
    "example.repo.org/sub/dir/2": {
      "key_id": "key1",
      "expires": "2019-05-09 23:10:32.497061458 +0200 CEST"
    },
    "example.repo.org/sub/dir/3": {
      "key_id": "key1",
      "expires": "2019-05-09 23:10:31.935336579 +0200 CEST"
    }
  },
  "status": "ok"
}
```

2.9.5 Advanced repository configuration

It's possible to register multiple API keys with each repository, and each key can be restricted to a specific subpath of the repository. When there are multiple keys for the same repository, and they are defined as files, naturally they can not all have the same filename, so at least some of them will be in a location not automatically imported by the gateway. For this reason, all the key file names need to be explicitly enumerated. Keys can also be declared inline. The `"version": 2` property needs to be specified for this configuration format to be accepted:

```
{
  "version": 2,
  "repos": [
    {
      "domain": "test.cern.ch",
      "keys": [
        {
          "id": "keyid1",
          "path": "/"
        },
        {
          "id": "keyid2",
          "path": "/restricted/to/subdir"
        }
      ]
    }
  ],
  "keys": [
    {
      "type": "file",
      "file_name": "/etc/cvmfs/keys/test.cern.ch.gw"
    },
    {
      "type": "plain_text",
      "id": "keyid2",
      "secret": "<SECRET>"
    }
  ]
}
```

It should be noted that when keys are loaded from a file, an `id` field does not need to be specified in the configuration file. The public id of the loaded key is the one specified in the key file itself.

2.9.6 Legacy repository configuration syntax

In the legacy repository configuration format, subpath restrictions are given with the key declaration, not when associating the keys with the repository:

```
{
  "repos": [
    {
      "domain": "test.cern.ch",
      "keys": ["<KEY_ID>"]
    }
  ],
  "keys": [
    {
      "type": "file",
      "file_name": "/etc/cvmfs/keys/test.cern.ch.gw",
```

(continues on next page)

(continued from previous page)

```
      "repo_subpath": "/"
    }
  ]
}
```

2.9.7 Updating from cvmfs-gateway-0.2.5

In the first published version, `cvmfs-gateway-0.2.5`, the application files were installed under `/opt/cvmfs-gateway` and the database files under `/opt/cvmfs-mnesia`. Starting with version 0.2.6, the application is installed under `/usr/libexec/cvmfs-gateway`, while the database files are under `/var/lib/cvmfs-gateway`.

When updating from 0.2.5, please make sure that the application is stopped:

```
# systemctl stop cvmfs-gateway
```

and rerun the setup script:

```
# /usr/libexec/cvmfs-gateway/scripts/setup.sh
```

At this point, the new version of the application can be started. If the old directories are still present, they can be deleted:

```
# rm -r /opt/cvmfs-{gateway,mnesia}
```

2.9.8 API reference

This sections describes the HTTP API exposed by the gateway application.

Repositories

GET /repos

Retrieve the list of all configured repositories

Response

```
{
  "data": {
    "test1.cern.ch": {
      "keys": {
        "k1": "/"
      },
      "enabled": true
    }
  },
  "status": "ok"
}
```

GET /repos/<REPO_NAME>

Retrieve the configuration for a repository

Response

```
{
  "data": {
    "keys": {
      "k1": "/"
    },
    "enabled": true
  },
  "status": "ok"
}
```

Leases

GET /leases

Retrieve the current list of leases

Response

```
{
  "data": {
    "test1.cern.ch/": {
      "key_id": "k1",
      "expires": "2021-10-25 22:02:12.688703553 +0000 UTC"
    }
  },
  "status": "ok"
}
```

GET /leases/<TOKEN>

Retrieve information about the lease identified by the given token

Response

```
{
  "data": {
    "key_id": "k1",
    "path": "test1.cern.ch/",
    "expires": "2021-10-25 22:14:12.695939889 +0000 UTC"
  }
}
```

POST /leases

Request a new lease

Headers

Header	Value	Description
Authorization	"<KEY_ID> <HMAC>"	"<KEY_ID>" identifies a gateway key used to sign the message and "<HMAC>" is the keyed-hash message authentication code (HMAC) of the request body.

Request parameters

Parameter	Example value	Description
api_version	"3"	API version requested by the client (passed as a string)
path	"test1.cern.ch/path/to/lease"	Repository subpath on which a lease is requested

Response

Outcome	Field	Value	Description
Success	status	"ok"	Response status
	session_token	"<TOKEN>"	String containing the session token associated with the new lease
	max_api_version	3	Max API version usable for the remainder of the session
Path busy	status	"path_busy"	There is a conflicting lease for the requested path
	"time_remaining"	1234	Remaining lease time in seconds
Error	status	"error"	An error occurred
	reason	"Something went wrong"	Description text of the error

POST /leases/<TOKEN>

Commit all changes associated with a lease

Headers

Header	Value	Description
Authorization	"<KEY_ID> <HMAC>"	"<KEY_ID>" identifies a gateway key used to sign the message and "<HMAC>" is the keyed-hash message authentication code (HMAC) of the request's path component (/lease/<TOKEN>).

Request parameters

Parameter	Example value	Description
old_root_hash	"abcd3f"	Initial root hash of the repository
new_root_hash	"bfa42b"	New root hash of the repository
tag name	"Monday"	Tag associated with the publication
tag_channel	"Nightlies"	Name of the publication channel
tag_description	"Nightly builds, Monday's batch"	Description of the tag

Response

Outcome	Field	Value	Description
Success	status	"ok"	Response status
	final_revision	1234	New revision of the repository after committing the changes associated with a lease
Error	status	"error"	An error occurred
	reason	"Something went wrong"	Description text of the error

DELETE /leases/<TOKEN>

Cancel a lease

Headers

Header	Value	Description
Authorization	"<KEY_ID> <HMAC>"	"<KEY_ID>" identifies a gateway key used to sign the message and "<HMAC>" is the keyed-hash message authentication code (HMAC) of the request's path component (/lease/<TOKEN>).

Response

Outcome	Field	Value	Description
Success	status	"ok"	Response status
Error	status	"error"	An error occurred
	reason	"Something went wrong"	Description text of the error

Payload submission

POST /payloads (deprecated)

Upload an object pack payload

Headers

Header	Value	Description
Authorization	"<KEY_ID> <HMAC>"	"<KEY_ID>" identifies a gateway key used to sign the message and "<HMAC>" is the keyed-hash message authentication code (HMAC) of the JSON message at the start of the request body.
message-size	1234	Total length of the JSON message at the start of the request body

Request parameters

Parameter	Example value	Description
session_token	"<SESSION_TOKEN>"	Session token associated with the lease
payload_digest	"bfa42b"	Digest of the payload part (serialized object pack) of the request
header_size	1234	Size of the payload header (the header of the serialized object pack)
api_version	"3"	API version tag (unused)

The upload payload (the serialized object pack) comes after the JSON part of the message.

Response

Outcome	Field	Value	Description
Success	`status`	"ok"	Response status
Error	`status`	"error"	An error occurred
	`reason`	"Something went wrong"	Description text of the error

POST /payloads/<TOKEN>

Upload an object pack payload

Headers

Header	Value	Description
`Authorization`	"<KEY_ID> <HMAC>"	"<KEY_ID>" identifies a gateway key used to sign the message and "<HMAC>" is the keyed-hash message authentication code (HMAC) of the session token.
`message-size`	1234	Total length of the JSON message at the start of the request body

Request parameters

Parameter	Example value	Description
`payload_digest`	"bfa42b"	Digest of the payload part (serialized object pack) of the request
`header_size`	1234	Size of the payload header (the header of the serialized object pack)
`api_version`	"3"	API version tag (unused)

The upload payload (the serialized object pack) comes after the JSON part of the message.

Response

Outcome	Field	Value	Description
Success	`status`	"ok"	Response status
Error	`status`	"error"	An error occurred
	`reason`	"Something went wrong"	Description text of the error

Notifications

POST /notifications/publish

Publish a notification

Request parameters

Parameter	Example value	Description
`version`	1	API version tag (unused)
`timestamp`	"26 Oct 2021 15:00:00"	Timestamp
`type`	"activity"	Message type (no other values are currently used)
`repository`	"test.cern.ch"	Repository name
`manifest`	"<MANIFEST STRING>"	The serialized signed repository manifest

Response

Outcome	Field	Value	Description
Success	`status`	"ok"	Response status
Error	`status`	"error"	An error occurred
	`reason`	"Something went wrong"	Description text of the error

GET /notifications/subscribe

Subscribe to notifications

Request parameters

Parameter	Example value	Description
`version`	1	API version tag (unused)
`repository`	"test.cern.ch"	Target repository name

This request opens a long-running connection to the notification server. Messages are delivered as server-sent events (SSE), one per line:

```
data: <JSON MESSAGE>
```

Messages

Parameter	Example value	Description
`version`	1	API version tag (unused)
`timestamp`	"26 Oct 2021 15:00:00"	Timestamp
`type`	"activity"	Message type (no other values are currently used)
`repository`	"test.cern.ch"	Repository name
`manifest`	"<MANIFEST STRING>"	The serialized signed repository manifest

2.9.9 Publication workflow

2.10 The CernVM-FS Notification System (Experimental)

This page describes the CernVM-FS notification system, a reactive repository change propagation system, complementary to the default, pull-based, approach based on the time-to-live value of cached repository manifests. This new system is used when a more precise propagation method is needed. One such use case is the distribution of conditions databases, which during data taking change at a much higher rate than software repositories. In a conditions data workflow, it is desired to process new data samples as soon as they are available, to avoid the pileup of new samples. Another case is the construction of a complex software build and test pipeline, where later stages of the pipeline depend on artifacts published at earlier stages of the pipeline already being available in replicas of the repository.

The main components of the notification system are a message broker, part of the CernVM-FS repository gateway application, and a command-line tool to publish new messages and subscribe to notifications. CernVM-FS clients can also be configured to receive and react to notifications. Communication between the notification system clients and the broker is done with standard HTTP. The message broker does not require any specific configuration. Please consult the relevant documentation (*The CernVM-FS Repository Gateway and Publishers*) for setting up a gateway.

2.10.1 Command-line tool for the notification system

There is a new `notify` sub-command in the `cvmfs_swissknife` command, which is used to publish and subscribe to activity messages for a specific repository.

Example:

- The CernVM-FS repository is located at `http://stratum-zero.cern.ch/cvmfs/test.repo.ch`
- The repository gateway is located at `http://gateway.cern.ch:4929/api/v1`

To publish the current manifest of the repository to the notification system, simply run:

```
# cvmfs_swissknife notify -p \
  -u http://gateway.cern.ch:4929/api/v1 \
  -r http://stratum-zero.cern.ch/cvmfs/test.cern.ch
```

To subscribe to the stream of messages concerning the repository, run:

```
# cvmfs_swissknife notify -s \
  -u http://gateway.cern.ch:4929/api/v1 \
  -t test.cern.ch
```

By default, once a message is received, the command will exit.

The subscription command has two optional flags:

- `-c` enables "continuous" mode. When messages are received, the command will output the message but will not exit.
- `-m NUM` specifies of minimum repository revision number to react to. For messages with a revision number smaller than or equal to `NUM`, no output is printed and the command will not exit (when the `-c` flag is not given).

2.10.2 CernVM-FS client configuration

A CernVM-FS client can also be connected to a notification server, allowing the client to react to activity messages by triggering a remount of the repository.

This functionality is enabled with the following client configuration option:

```
CVMFS_NOTIFICATION_SERVER=http://gateway.cern.ch:4929/api/v1
```

2.11 Container Images and CernVM-FS

CernVM-FS interacts with container technologies in two main ways:

1. CernVM-FS application repositories (e.g. /cvmfs/atlas.cern.ch) can be mounted into a stock container (e.g. CentOS 8)
2. The container root filesystem (e.g. the root file system "/" of CentOS 8) itself can be served directly from CernVM-FS

Both ways have a similar goal, that is to give users access to a reproducible, ready-to-use environment while retaining the advantages of CernVM-FS regarding data distribution, content de-duplication, software preservation and ease of operations.

2.11.1 Mounting /cvmfs inside a container

The simplest way to access /cvmfs from inside a container is to bind-mount the /cvmfs host directory inside the container.

Using this approach will allow using small images to create a basic operating system environment, and to access all the necessary application software through /cvmfs.

This is supported by all the common containers runtimes, including:

1. Docker
2. Podman
3. Apptainer
4. Kubernetes

Examples

To bind-mount CVMFS inside a docker container, it is sufficient to use the --volume/-v flag.

For instance:

```
docker run -it --volume /cvmfs:/cvmfs:shared ubuntu ls -lna /cvmfs/atlas.cern.ch
```

Of course, it is also possible to limit the bind-mount to only one repository, or a few repositories:

```
$ docker run -it -v /cvmfs/alice.cern.ch:/cvmfs/alice.cern.ch \
                 -v /cvmfs/sft.cern.ch:/cvmfs/sft.cern.ch ubuntu
root@808d42605e97:/# ll /cvmfs/
total 17
drwxr-xr-x 17  125  130 4096 Nov 27  2012 alice.cern.ch/
drwxr-xr-x  8  125  130 4096 Oct 15  2018 sft.cern.ch/
```

Podman has the same interface as docker, but it requires the ro options when mounting a single repository.

```
$ podman run -it -v /cvmfs/alice.cern.ch:/cvmfs/alice.cern.ch:ro ubuntu ls -lna /
↪cvmfs/
total 13
drwxr-xr-x  3     0     0 4096 Apr 20 11:34 .
drwxr-xr-x 22     0     0 4096 Apr 20 11:34 ..
drwxr-xr-x 17 65534 65534 4096 Nov 27  2012 alice.cern.ch
```

A similar approach is possible with apptainer, but the syntax is a little different.

```
$ apptainer exec --bind /cvmfs docker://library/ubuntu ls -l /cvmfs/lhcb.cern.ch
total 2
drwxrwxr-x.  3 cvmfs cvmfs  3 Jan  6  2011 etc
lrwxrwxrwx.  1 cvmfs cvmfs 16 Aug  6  2011 group_login.csh -> lib/etc/LHCb.csh
lrwxrwxrwx.  1 cvmfs cvmfs 15 Aug  6  2011 group_login.sh -> lib/etc/LHCb.sh
drwxrwxr-x. 20 cvmfs cvmfs  3 Apr 24 12:39 lib
```

Also in apptainer it is possible to use the syntax host_directory:container_directory and it is possible to mount multiple paths at the same time separating the --bind arguments with a comma.

```
$ apptainer exec --bind /cvmfs/alice.cern.ch:/cvmfs/alice.cern.ch,/cvmfs/lhcb.cern.ch␣
↪\
    docker://library/ubuntu ls -l /cvmfs/
total 5
drwxr-xr-x 17      125      130 4096 Nov 27  2012 alice.cern.ch/
drwxrwxr-x  4      125      130    6 Nov 16  2010 lhcb.cern.ch/
```

For Kubernetes, the approach is more heterogeneous and it depends on the cluster settings. A recommended approach is creating a DaemonSet so that on every node one pod exposes /cvmfs to other pods. This pod may use the cvmfs service container.

Alternatively, a CSI-plugin makes it simple to mount a repository inside a Kubernetes managed container. The plugin is distributed and available to the CERN Kubernetes managed clusters.

2.11.2 Distributing container images on CernVM-FS

Image distribution on CernVM-FS works with *unpacked* layers or image root file systems. Any CernVM-FS repository can store container images.

A number of images are already provided in `/cvmfs/unpacked.cern.ch`, a repository managed at CERN to host container images for various purposes and groups. The repository is managed using the CernVM-FS container tools to publish images from registries on CernVM-FS.

Every container image is stored in two forms on CernVM-FS

1. All the unpacked layers of the image
2. The whole unpacked root filesystem of the image

With the whole filesystem root directory in /cvmfs, `apptainer` can directly start a container.

```
apptainer exec /cvmfs/unpacked.cern.ch/registry.hub.docker.com/library/centos\
↪:centos7 /bin/bash
```

The layers can be used, e.g., with containerd and the CernVM-FS snapshotter. In addition, the container tools create the *chains* of an image. Chains are partial root filesystem directores where layers are applied one after another. This is used internally to incrementally publish image updates if only a subset of layers changed.

Using unpacked.cern.ch

The `unpacked.cern.ch` repository provides a centrally managed container image hub without burdening users with managing their CernVM-FS repositories or conversion of images. It also enables saving storage space because of cvmfs deduplication of files that are common between different images. The repository is publicly available.

To add your image to `unpacked.cern.ch` you can add the image name to any one of the following two files, the so-called *wishlists*.

1. https://gitlab.cern.ch/unpacked/sync/-/blob/master/recipe.yaml
2. https://github.com/cvmfs/images-unpacked.cern.ch/blob/master/recipe.yaml

The first file is accessible from CERN infrastructure, while the second is on Github open to everybody.

A simple pull request against one of those files is sufficient, the image is vetted, and the pull request merged. Soon after the pull request is merged DUCC publishes the image to /cvmfs/unpacked.cern.ch. Depending on the size of the image, ingesting an image in unpacked.cern.ch takes ~15 minutes.

The images are continuously checked for updates. If you push another version of the image with the same tag, the updated propagates to CernVM-FS usually within ~15 minutes of delay.

Image wishlist syntax

The image must be specified like the following examples:

```
https://registry.hub.docker.com/library/centos:latest
https://registry.hub.docker.com/cmssw/cc8:latest
https://gitlab-registry.cern.ch/clange/jetmetanalysis:latest
```

The first two refer to images in Docker Hub, the standard `centos` using the latest tag and the `cms` version of centos8, again using the latest tag. The third image refers to an image hosted on CERN GitLab that contains the code for an analysis by a CERN user.

It is possible to use the * wildcard to specify multiple tags.

For instance:

```
https://registry.hub.docker.com/atlas/analysisbase:21.2.1*
```

is a valid image specification, and triggers conversion of all the `atlas/analysisbase` images whose tags start with `21.2.1`, including:

```
atlas/analysisbase:21.2.10
atlas/analysisbase:21.2.100-20191127
atlas/analysisbase:21.2.15-20180118
```

But **not** `atlas/analysisbase:21.3.10`.

The * wildcard can also be used to specify all the tags of an image, like in this example:

```
https://registry.hub.docker.com/pyhf/pyhf:*
```

All the tags of the image `pyhf/pyhf` that are published on Docker Hub will be published in unpacked.cern.ch.

Updated images and new tags

The unpacked.cern.ch service polls the upstream registries continuously. As soon as a new or modified container image is detected it starts the conversion process.

`containerd` snapshotter plugin (pre-production)

CernVM-FS integration with `containerd` is achieved by the cvmfs snapshotter plugin, a specialized component responsible for assembling all the layers of container images into a stacked filesystem that `containerd` can use. The snapshotter takes as input the list of required layers and outputs a directory containing the final filesystem. It is also responsible to clean-up the output directory when containers using it are stopped.

How to use the CernVM-FS Snapshotter

The CernVM-FS snapshotter runs alongside the containerd service. The snapshotter communicates with `containerd` via gRPC over a UNIX domain socket. The default socket is `/run/containerd-cvmfs-grpc/containerd-cvmfs-grpc.sock`. This socket is created automatically by the snapshotter if it does not exist.

The containerd snapshotter is available from http://ecsft.cern.ch/dist/cvmfs/snapshotter/. Packages will be made available in future.

The binary accepts the following command line options:

- `--address`: address for the snapshotter's GRPC server. The default one is `/run/containerd-cvmfs-grpc/containerd-cvmfs-grpc.sock`

- `--config`: path to the configuration file. Creating a configuration file is useful to customize the default values.
- `--log-level`: logging level [trace, debug, info, warn, error, fatal, panic]. The default value is `info`.
- `--root`: path to the root directory for this snapshotter. The default one is `/var/lib/containerd-cvmfs-grpc`.

By default, the repository used to search for the layers is `unpacked.cern.ch`. The default values can be overwritten in the `config.toml` file using the `--config` option. A template `config.toml` file looks like this:

```
version = 2

# Source of image layers
repository = "unpacked.cern.ch"
absolute-mountpoint = "/cvmfs/unpacked.cern.ch"

# Ask containerd to use this particular snapshotter
[plugins."io.containerd.grpc.v1.cri".containerd]
    snapshotter = "cvmfs-snapshotter"
    disable_snapshot_annotations = false

# Set the communication endpoint between containerd and the snapshotter
[proxy_plugins]
    [proxy_plugins.cvmfs]
        type = "snapshot"
        address = "/run/containerd-cvmfs-grpc/containerd-cvmfs-grpc.sock"
```

Note that if only the repository is specified under the key value `repository`, the mountpoint (under the key value `absolute-mountpoint`) is by default constructed as `/cvmfs/<repo_name>`.

`podman` integration (pre-production)

In order to use images from unpacked.cern.ch with podman, the podman client needs to point to an *image store* that references the images on /cvmfs. The image store is a directory is a directory with a a certain file structure that provides an index of images and layers. The CernVM-FS container tools by default create a podman image store for published images.

In order to set the image store, edit `/etc/containers/storage.conf` or `${HOME}/.config/containers/storage.conf` like in this example:

```
[storage]
driver = "overlay"

[storage.options]
additionalimagestores = [ "/cvmfs/unpacked.cern.ch/podmanStore" ]
# mount_program = "/usr/bin/fuse-overlayfs"

[storage.options.overlay]
mount_program = "/usr/bin/fuse-overlayfs"
```

The configuration can be checked with the `podman images` command.

Note: the image store in the unpacked.cern.ch repository currently provides access only to test images. This is due to poor performance in the image conversion when the image store is updated. This will be fixed in a future version.

2.12 Advanced Topics

2.12.1 Client Plug-Ins

The CernVM-FS client's functionality can be extended through plug-ins. CernVM-FS plug-ins are binaries (processes) that communicate with the main client process through IPC. Currently there are two plug-in interfaces: cache manager plugins and authorization helpers.

Cache Plugins

A cache plugin provides the functionality of the client's local cache directory: it maintains a set of content-addressed objects. Clients can read from these objects. Depending on its capabilities, a cache plugin might also support addition of new objects, listing objects and eviction of objects from the cache.

Note: The CernVM-FS client trusts the contents of the cache. Cache plugins that store data in untrusted locations need to perform their own content verification before data is provided to the clients.

Cache plugins and clients exchange messages through a socket. The messages are serialized by the Google protobuf library. A description of the wire protocol can be found in the `cvmfs/cache.proto` source file, although the cache plugins should not directly implement the protocol. Instead, plugins are supposed to use the `libcvmfs_cache` library (part of the CernVM-FS development package), which takes care of the low-level protocol handling.

Good entry points into the development of a cache plugin are the demo plugin `cvmfs/cache_plugin/cvmfs_cache_null.cc` and the production in-memory cache plugin `cvmfs/cache_plugin/cvmfs_cache_ram.cc`. The CernVM-FS unit test suite has a unit test driver, `cvmfs_test_cache`, with a number of tests that are helpful for the development and debugging of a cache plugin.

Broadly speaking, a cache plugin process performs the following steps

```
#include <libcvmfs_cache.h>

cvmcache_init_global();
// Option parsing, which can use cvmcache_options_... functions to parse
// CernVM-FS client configuration files

// Optionally: spawning the watchdog to create stack traces when the cache
// plugin crashes
cvmcache_spawn_watchdog(NULL);

// Create a plugin context by passing function pointers to callbacks
struct cvmcache_context *ctx = cvmcache_init(&callbacks);

// Connect to the socket defined by the locator string
cvmcache_listen(ctx, locator);

// Spawn an I/O thread in which the callback functions are called
cvmcache_process_requests(ctx, 0);

// Depending on whether the plugin is started independently or by the
// CernVM-FS client, cvmcache_process_requests() termination behaves
// differently

if (!cvmcache_is_supervised()) {
  // Decide when the plugin should be terminated, e.g. wait for a signal
  cvmcache_terminate(ctx);
}

// Cleanup
```

(continues on next page)

(continued from previous page)

```
cvmcache_wait_for(ctx);
cvmcache_terminate_watchdog();
cvmcache_cleanup_global();
```

The core of the cache plugin is the implementation of the callback functions provided to `cvmcache_init()`. Not all callback functions need to be implemented. Some can be set to `NULL`, which needs to correspond to the indicated plugin capabilities specified in the `capabilities` bit vector.

Basic Capabilities

Objects maintained by the cache plugin are identified by their content hash. Every cache plugin must be able to check whether a certain object is available or not and, if it is available, provide data from the object. This functionality is provided by the `cvmcache_chrefcnt()`, `cvmcache_obj_info()`, and `cvmcache_pread()` callbacks. With only this functionality, the cache plugin can be used as a read-only lower layer in a tiered cache but not as a stand-alone cache manager.

For a proper stand-alone cache manager, the plugin must keep reference counting for its objects. The concept of reference counting is borrowed from link counts in UNIX file systems. Every object in a cache plugin has a reference counter that indicates how many times the object is being in use by CernVM-FS clients. For objects in use, clients expect that reading succeeds, i.e. objects in use must not be deleted.

Adding Objects

On a cache miss, clients need to populate the cache with the missing object. To do so, cache plugins provide a transactional write interface. The upload of an object results in the following call chain:

1. A call to `cvmcache_start_txn()` with a given transaction id
2. Zero, one, or multiple calls to `cvmcache_write_txn()` that append data
3. A call to `cvmcache_commit_txn()` ro `cvmcache_abort_txn()`

Only after commit the object must be accessible for reading. Multiple concurrent transactions on the same object are possible. After commit, the reference counter of the object needs to be equal to the number of transactions that committed the object (usually 1).

Listing and Cache Space Management

Listing of the objects in the cache and the ability to evict objects from the cache are optional capabilities. Only objects whose reference counter is zero may be evicted. Clients can keep file catalogs open for a long time, thereby preventing them from being evicted. To mitigate that fact, cache plugins can at any time send a notification to clients using `cvmcache_ask_detach()`, asking them to close as many nested catalogs as they can.

Authorization Helpers

Client authorization helpers (*authz helper*) can be used to grant or deny read access to a mounted repository. To do so, authorization helpers can verify the local UNIX user (uid/gid) and the process id (pid) that is issuing a file system request.

An authz helper is spawned by CernVM-FS if the root file catalog contains *membership requirement* (see below). The binary to be spawned is derived from the membership requirement but it can be overwritten with the `CVMFS_AUTHZ_HELPER` parameter. The authz helper listens for commands on `stdin` and it replies on `stdout`.

Grant/deny decisions are typically cached for a while by the client. Note that replies are cached for the entire session (session id) that contains the calling process id.

Membership Requirement

The root file catalog of a repository determines if and which authz helper should be used by a client. The membership requirement (also called *VOMS authorization*) can be set, unset, and changed when creating a repository and on every publish operation. It has the form

```
<helper>%<membership string>
```

The `<helper>` component helps the client find an authz helper. The client searches for a binary `${CVMFS_AUTHZ_SEARCH_PATH}/cvmfs_<helper>_helper`. By default, the search path is `/usr/libexec/cvmfs/authz`. CernVM-FS comes with two helpers: `cvmfs_helper_allow` and `cvmfs_helper_deny`. Both helpers make static decisions and disregard the membership string. Other helpers can use the membership string to specify user groups that are allowed to access a repository.

Authz Helper Protocol

The authz helper gets spawned by the CernVM-FS client with `stdin` and `stdout` connected. There is a command/reply style of messages. Messages have a 4 byte version (=1), a 4 byte length, and then a JSON text that needs to contain the top-level struct `cvmfs_authz_v1 { ... }`. Communication starts with a handshake where the client passes logging parameters to the authz helper. The client then sends zero or more authorization requests, each of which is answered by a positive or negative permit. A positive permit can include an access token that should be used to download data. The permits are cached by the client with a TTL that the helper can chose. On unmount, the client sends a quit command to the helper.

When spawned, the authz helper's environment is prepopulated with all `CVMFS_AUTHZ_...` environment variables that are in the CernVM-FS client's environment. Furthermore the parameter `CVMFS_AUTHZ_HELPER=yes` is set.

The JSON snippet of every message contains `msgid` and `revision` integer fields. The revision is currently 0 and unused. Message ids indicate certain other fields that can or should be present. Additional JSON text is ignored. The message id can be one of the following

Code	Meaning
0	Cvmfs: "Hello, helper, are you there?" (handshake)
1	Helper: "Yes, cvmfs, I'm here" (handshake reply)
2	Cvmfs: "Please verify, helper" (verification request)
3	Helper: "I verified, cvmfs, here's the result" (permit)
4	Cvmfs: "Please shutdown, helper" (termination)

Handshake and Termination

In the JSON snippet of the hand shake, the CernVM-FS client transmits the fully qualified repository name (`fqrn` string field) and the syslog facility and syslog level the helper is supposed to use (`syslog_facility`, `syslog_level` integer fields). The handshake reply as well as the termination have no additional payload.

Verification Requests

A verification request contains the uid, gid, and pid of the calling process (`uid`, `gid`, `pid` integer fields). It furthermore contains the Base64 encoded membership string from the membership requirement (`membership` string field).

The permit has to contain a status indicating success or failure (`status` integer field) and a time to live for this reply in seconds (`ttl` integer field). The status can be one of the following

Code	Meaning
0	Success (allow access)
1	Authentication token of the user not found (deny access)
2	Invalid authentication token (deny access)
3	User is not member of the required groups (deny access)

On success, the permit can optionally conatain a Base64 encoded version of either an X.509 proxy certificate (`x509_proxy` string field) or a bearer token (`bearer_token` string field). These credentials are used by the CernVM-FS client when downloading nested catalogs files as client-side HTTPS authentication information.

2.12.2 Tracing File System Accesses

The CernVM-FS Fuse client comes with a built-in tracer that can be used to record file system accesses to repositories. The tracer produces a CSV file. Every file system call, such as opening a file or listing a directory, is written as another line into the log file.

In order to activate the tracer, set

```
CVMFS_TRACEFILE=/tmp/cvmfs-trace-@fqrn@.log  # the cvmfs user must have write␣
↪permission to the target directory
```

The `@fqrn@` syntax ensures that the trace file is different for every repository.

The trace is internally buffered. Therefore, it is important to either unmount the CernVM-FS client or to call `cvmfs_talk tracebuffer flush` at the end of a tracing session in order to produce a complete record.

By default, the trace buffer can keep 8192 recorded calls, and it will start to flush on disk at 7000 recorded system calls. The buffer parameters can be adjusted with the two parameters `CVMFS_TRACEBUFFER` and `CVMFS_TRACEBUFFER_THRESHOLD`.

Trace Log Format

The generated trace log is a CSV file with the following fields

Field	Description
Timestamp	Seconds since the UNIX epoch, miliseconds precision
Event code	Numerical ID for the system call. Negative numbers indicate internal events, such as mounting and unmounting.
Path	The repository relative target path of the system call
Event name	A string literal corresponding to the event code.

The following events are known:

Event ID	Description
1	Open file
2	List directory contents
3	Read symbolic link
4	Lookup path
5	Get file system meta-data (e.g. df call)
6	Get file/directory meta-data
7	List extended attributes of a file/directory
8	Read extended attributes of a file/directory

2.12.3 Ephemeral Writable Container

Note: this feature is still considered experimental.

The CernVM-FS ephemeral writable container can provide a short-lived shell with writable access to a regular, read-only CernVM-FS repository. A writable CernVM-FS mountpoint is normally a functionality that only publisher nodes provide. With the ephemeral writable container, this capability becomes available to every regular client.

The ephemeral writable container requires the `cvmfs-server` package to be installed. Provided that the target repository is already mounted, a writable shell is opened with

```
cvmfs_server enter <repository name> [-- <command>]
```

Changes to the writable mountpoint are only stored locally. The changes are discarded when the shell is closed. In a future release it will be possible to publish changes directly to a gateway.

Repository changes in the writable shell can be shown with

```
cvmfs_server diff --worktree
```

Before closing the shell, changes can be manually copied to a publisher node for publication. This helps with building and deploying non-relocatable packages to CernVM-FS.

The ephemeral writable container uses Linux user namespaces and fuse-overlayfs in order to construct the writable repository mountpoint. Therefore, it requires a recent enough kernel. The vanilla kernel >= 4.18 and the EL 8 kernel are known to work.

The container creates a session directory in `$HOME/.cvmfs` to store temporary files and changes to the repository. By default, the session directory is removed when exiting the shell. It can be preserved with the `--keep-session` parameter. If only the logs should be preserved, use the `--keep-logs` parameter instead.

If necessary, the container can be opened as fake root user using the `root` option.

Note that by default a dedicated CernVM-FS cache directory is created for the lifetime of the ephemeral container. It can be desireable to use a shared cache directory across several invocations of the `cvmfs_server enter` command. To do so, use the `--cvmfs-config <config file>` parameter and set `CVMFS_CACHE_BASE=/common/path` in the passed configuration file.

2.12.4 CernVM-FS on Supercomputers

There are several characteristics in which supercomputers can differ from other nodes with respect to CernVM-FS

1. Fuse is not allowed on the individual nodes
2. Individual nodes do not have Internet connectivity
3. Nodes have no local hard disk to store the CernVM-FS cache

These problems can be overcome as described in the following sections.

Running CernVM-FS as an unprivileged user

CernVM-FS can be run as an unprivileged user under several different scenarios. See documentation about that in the Security *Running the client as a normal user* section.

Parrot-Mounted CernVM-FS in lieu of Fuse Module

Instead of accessing /cvmfs through a Fuse module, processes can use the Parrot connector. The parrot connector works on x86_64 Linux if the `ptrace` system call is not disabled. In contrast to a plain copy of a CernVM-FS repository to a shared file system, this approach has the following advantages:

- Millions of synchronized meta-data operations per node (path lookups, in particular) will not drown the shared cluster file system but resolve locally in the parrot-cvmfs clients.
- The file system is always consistent; applications never see half-synchronized directories.
- After initial preloading, only change sets need to be transfered to the shared file system. This is much faster than *rsync*, which always has to browse the entire name space.
- Identical files are internally de-duplicated. While space of the order of terabytes is usually not an issue for HPC shared file systems, file system caches benefit from deduplication. It is also possible to preload only specific parts of a repository namespace.
- Support for extra functionality implemented by CernVM-FS such as versioning and variant symlinks (symlinks resolved according to environment variables).

Downloading complete snapshots of CernVM-FS repositories

When there is no possible way to run the CernVM-FS client, an option that has been used on some HPC systems is to download entire or partial snapshots of CernVM-FS repositories using the *cvmfs_shrinkwrap utility*. These snapshots are also sometimes called "HPC fat container images". This has many disadvantages compared to running a CernVM-FS client so it is typically a last resort.

NFS Export with Cray DVS

Some HPC sites have tried running the cvmfs client on just one server and exporting to worker nodes over *NFS*. These installations can be made to work but it is very inefficient and they often run into operational problems. If you want to try it using the Cray DVS please see the *workaround* on inode handling and DVS export.

Preloading the CernVM-FS Cache

When the CernVM-FS client can be installed on the worker node but for whatever reason on-demand downloading to a local cache is difficult, the cvmfs_preload utility can be used to preload a CernVM-FS cache onto the shared cluster file system. Internally it uses the same code that is used to replicate between CernVM-FS stratum 0 and stratum 1. The `cvmfs_preload` command is a self-extracting binary with no further dependencies and should work on a majority of x86_64 Linux hosts. Note however that this method can significantly strain the cluster file system's meta-data server(s) and that many HPC systems have had better results with *loopback filesystems* as node caches as discussed below.

The `cvmfs_preload` command replicates from a stratum 0 (not from a stratum 1). Because this induces significant load on the source server, stratum 0 administrators should be informed before using their server as a source. As an example, in order to preload the ALICE repository into /shared/cache, one could run from a login node

```
cvmfs_preload -u http://cvmfs-stratum-zero-hpc.cern.ch:8000/cvmfs/alice.cern.ch -r /
↪shared/cache
```

This will preload the entire repository. In order to preload only specific parts of the namespace, you can create a _dirtab_ file with path prefixes. The path prefixes must not involve symbolic links. An example dirtab file for ALICE could look like

```
/example/etc
/example/x86_64-2.6-gnu-4.8.3/Modules
/example/x86_64-2.6-gnu-4.8.3/Packages/GEANT3
/example/x86_64-2.6-gnu-4.8.3/Packages/ROOT
```

(continues on next page)

(continued from previous page)

```
/example/x86_64-2.6-gnu-4.8.3/Packages/gcc
/example/x86_64-2.6-gnu-4.8.3/Packages/AliRoot/v5*
```

The corresponding invokation of `cvmfs_preload` is

```
cvmfs_preload -u http://cvmfs-stratum-zero-hpc.cern.ch:8000/cvmfs/alice.cern.ch -r /
↪shared/cache \
  -d </path/to/dirtab>
```

The initial preloading can take several hours to a few days. Subsequent invokations of the same command only transfer a change set and typically finish within seconds or minutes. These subsequent invokations need to be either done manually when necessary or scheduled for instance with a cron job.

The `cvmfs_preload` command can preload files from multiple repositories into the same cache directory.

Access from the Nodes

In order to access a preloaded cache from the nodes, set the path to the directory as an *Alien Cache*. Since there won't be cache misses, parrot or fuse clients do not need to download additional files from the network.

If clients do have network access, they might find a repository version online that is newer than the preloaded version in the cache. This results in conflicts with `cvmfs_preload` or in errors if the cache directory is read-only. Therefore, we recommend to explicitly disable network access for the parrot process on the nodes, for instance by setting

```
HTTP_PROXY='INVALID-PROXY'
```

before the invocation of `parrot_run`.

Compiling `cvmfs_preload` from Sources

In order to compile `cvmfs_preload` from sources, use the `-DBUILD_PRELOADER=yes` cmake option.

Loopback File Systems for Nodes' Caches

If nodes have Internet access but no local hard disk, it is preferable to provide the CernVM-FS caches as loopback file systems on the cluster file system. This way, CernVM-FS automatically populates the cache with the latest upstream content. A Fuse mounted CernVM-FS will also automatically manage the cache quota.

This approach requires a separate file for every node (not every mountpoint) on the cluster file system. The file should be 15% larger than the configured CernVM-FS cache size on the nodes, and it should be formatted with an ext3/4 or an xfs file system. These files can be created with the `dd` and `mkfs` utilities. Nodes can mount these files as loopback file systems from the shared file system.

Because there is only a single file for every node, the parallelism of the cluster file system can be exploited and all the requests from CernVM-FS circumvent the cluster file system's meta-data server(s). That can be a very large advantage because very often the meta-data server is the bottleneck under typical workloads.

Tiered Cache and Cache Plugins

Diskless compute nodes can also combine an in-memory cache with a preloaded directory on the shared cluster file system. An example configuration can be found in Section *Example*.

2.12.5 CernVM-FS Graph Driver Plugin for Docker

The CernVM-FS graph driver plugin for Docker provides a dockerized CernVM-FS client that can be used by the Docker daemon to access and store container images that reside in an extracted form on a CernVM-FS repository. Because CernVM-FS downloads the files of a container image only when accessed and because typically very little of a container image is accessed at runtime, the CernVM-FS graph driver can remove the bottleneck of distributing (large) container images to (many) nodes.

The CernVM-FS graph driver can run any normal image from a Docker registry. Additionally, it can run so called *Thin Images*. A thin image is like a symbolic link for container images. It is a regular, very small image in the registry. It contains a single file, the *thin image descriptor*, that specifies where in a CernVM-FS repository the actual image contents can be found. The `docker2cvmfs` utility can be used to convert a regular image to a thin image.

Requirements

The graph driver plugin requires Docker version > 17 and a host kernel with either aufs or overlay2 support, which includes RHEL >= 7.3. Please note that on RHEL 7, Docker's data root should reside either on an ext file system or on an xfs file system that is formatted with the `ftype=1` mount option.

The Docker graph driver plugin receives its CernVM-FS configuration by default from the Docker host's /etc/cvmfs directory. The easiest way to populate /etc/cvmfs is to install the `cvmfs-config-default` package (or any other `cvmfs-config-...` package) on the Docker host. Alternatively, a directory structure resembling the /etc/cvmfs hierarchy can by manually created and linked to the graph driver plugin.

Installation

The following steps install and activate the CernVM-FS graph driver plugin.

1. Install the plugin with `docker plugin install cvmfs/graphdriver`. The command `docker plugin ls` should now show the new plugin as being activated.
2. Create or edit the file `/etc/docker/daemon.json` so that it contains the following content

```
{
  "experimental": true,
  "storage-driver": "cvmfs/graphdriver",

  // To change the docker data root to an ext formatted location (remove this␣
↪line)
  "data-root": "/path/to/ext/mountpoint",

  // Add the following storage option on RHEL 7 (remove this line)
  "storage-opts": [
    "overlay2.override_kernel_check=true"
  ]
}
```

3. Restart the Docker daemon with `systemctl restart docker`.
4. Test the new plugin with a normal image

```
docker run -it --rm ubuntu /bin/bash
```

and with a thin image

```
docker run -it --rm cvmfs/thin_ubuntu /bin/bash
```

In order to get debugging output, add `"debug": true` to the /etc/docker/daemon.json file.

Location of the Plugin Configuration

By default, the plugin tries to bind mount the host's /etc/cvmfs directory as a source of configuration. Other locations can be linked to the container by running

```
docker plugin set cvmfs/graphdriver cvmfs_ext_config="/alternative/location"
docker plugin set cvmfs/graphdriver minio_ext_config="/alternative/location"
```

Installation from a Plugin Tarball

Instead of installing the plugin from the Docker registry, it can be installed directly from a tarball. To do so, download and untar a graph driver plugin tarball. Run

```
docker plugin create my-graphdriver cvmfs-graphdriver-plugin-$VERSION
docker plugin enable my-graphdriver
```

Note: currently, the graph driver name (`my-graphdriver`) must not contain a colon (`:`) nor a comma (`,`). This issue will be fixed in a later version.

Conversion of Images

A simple way to ingest docker images inside a cvmfs repository is available through a small utility `docker2cvmfs`.

At the moment it is possible to directly download the executable: docker2cvmfs v0.3

`docker2cvmfs` provides different commands to manipulate docker images but the simplest way is to use the `make-thin` sub-command.

This sub-command expects to find on the host machine a recent version of `cvmfs_server` that supports the `ingest` command.

Invoking the help of the subcommand `docker2cvmfs make-thin --help` explains what options are available and how to use them.

Below we provide a complete example on how to use `docker2cvmfs` to convert the docker image of `Redis` into a thin image.

Assuming a cvmfs repository called `example.cern.ch` is already in place:

```
./docker2cvmfs make-thin --input-reference library/redis:4 --output-reference thin/
↪redis:4 --repository example.cern.ch
```

The utility takes as input the reference (`library/redis:4`) to the image to ingest into `cvmfs` along with the reference to associate to the new thin image (`thin/redis:4`) and the repository where we want to store the several layers (`example.cern.ch`).

The utility downloads every layer that composes the image, stores them into the repository, creates the new thin image and imports that into docker.

By default the layers are stored into the `layers/` subdirectory of the repository; this can be modified using the `--subdirectory` parameters.

The images are downloaded, by default, from the official docker hub registry, this can be modified as well using the `--registry` parameter.

2.12.6 Working with DUCC and Docker Images (Experimental)

DUCC (Daemon that Unpacks Container Images into CernVM-FS) helps in publishing container images in CernVM-FS. The daemon publishes images in their extracted form in order for clients to benefit from CernVM-FS' on-demand loading of files. The DUCC service is deployed as an extra package and supposed to be co-located with a publisher node having the `cvmfs-server` package installed.

Converted images are usable with Docker through the *CernVM-FS docker graph driver* and with container engines that can use a flat root file system from CernVM-FS such as Singularity and runc. For use with Docker, DUCC will upload a so-called "thin image" to the registry for every converted image. Only the thin image makes an image available through CernVM-FS.

Vocabulary

The following section introduces the terms used in the context of DUCC publishing container images.

Registry A Docker image registry such as:

- https://registry.hub.docker.com
- https://gitlab-registry.cern.ch

Image Repository This specifies a group of images. Each image in an image repository is addressed by tag or by digest. Examples are:

- library/redis
- library/ubuntu

The term **image repository** is unrelated to a CernVM-FS repository.

Image Tag An image tag identifies an image inside an image repository. Tags are mutable and may refer to different container images over time. Examples are:

- 4
- 3-alpine

Image Digest A digest is an immutable identifier for a container image. Digests are calculated based on the result of a hash function to the content of the image. Examples are:

- sha256:2aa24e8248d5c6483c99b6ce5e905040474c424965ec866f7decd87cb316b541
- sha256:d582aa10c3355604d4133d6ff3530a35571bd95f97aadc5623355e66d92b6d2c

To uniquely identify an image, we need to provide: 1. registry 2. image repository 3. image tag or image digest (or both)

We use a slash (/) to separate the *registry* from the *repository*, a colon (:) to separate the *repository* from the *tag* and the at (@) to separate the *digest* from the tag or from the *repository*. The syntax is

```
REGISTRY/REPOSITORY[:TAG][@DIGEST]
```

Examples of fully identified images are:

- https://registry.hub.docker.com/library/redis:4
- https://registry.hub.docker.com/minio/minio@sha256:b1e5dd4a7be831107822243a0675ceb5eabe124356a9815f2519fe02beb
- https://registry.hub.docker.com/wurstmeister/kafka:1.1.0@sha256:3a63b48894bce633fb2f0d2579e162163367113d79ea12ca

Thin Image A Docker image that contains only a reference to the image contents in CernVM-FS. Requires the CernVM-FS Docker graph driver in order to start.

Image Wish List

The user specifices the set of images supposed to be published on CernVM-FS in the form of a wish list. The wish list consists of triplets of input image, the output thin image and the cvmfs destination repository for the unpacked data.

```
wish => (input_image, output_thin_image, cvmfs_repository)
```

The input image in your wish should unambigously specify an image as decribed above.

Wish List Syntax v1

The wish list is provided as YAML file. An example of a wish list containing four images is show below.

```
version: 1
user: smosciat
cvmfs_repo: unpacked.cern.ch
output_format: '$(scheme)://registry.gitlab.cern.ch/thin/$(image)'
input:
    - 'https://registry.hub.docker.com/econtal/numpy-mkl:latest'
    - 'https://registry.hub.docker.com/agladstein/simprily:version1'
    - 'https://registry.hub.docker.com/library/fedora:latest'
    - 'https://registry.hub.docker.com/library/debian:stable'
```

version: wish list version; at the moment only *1* is supported.

user: the account that will push the thin images into the docker registry. The password must be stored in the DOCKER2CVMFS_DOCKER_REGISTRY_PASS environment variable.

cvmfs_repo: the target CernVM-FS repository to store the layers and the flat root file systems.

output_format: how to name the thin images. It accepts a few variables that refer to the input image.

- $(scheme), the image url protocol, most likely *http* or *https*
- $(registry), the Docker registry of the input image, in the case of the example it would be *registry.hub.docker.com*
- $(repository), the image repository of the input image, like *library/ubuntu* or *atlas/athena*
- $(tag), the tag of the image, which could be *latest*, *stable* or *v0.1.4*
- $(image), combines $(repository) and $(tag)

input: list of docker images to convert

The current wish list format requires all the images to be stored in the same CernVM-FS repository and have the same thin output image format.

DUCC Commands

DUCC supports the following commands.

convert

The *convert* command provides the core functionality of DUCC:

```
cvmfs_ducc convert wishlist.yaml
```

where *wishlist.yaml* is the path of a wish list file.

This command will try to ingest all the specified images into CernVM-FS.

The process consists of downloading the manifest of the image, downloading and ingesting the layers that compose each image, uploading the thin image, creating the flat root file system necessary to work with Singularity and writing DUCC specific metadata in the CernVM-FS repository next to the unpacked image data.

The layers are stored in the *.layer* subdirectory in the CernVM-FS repository, while the flat root file systems are stored in the *.flat* subdirectory.

loop

The *loop* command continously executes the *convert* command. On each iteration, the wish list file is read again in order to pick up changes.

```
cvmfs_ducc loop recipe.yaml
```

convert-single-image

The *convert-single-image* command is useful when only a single image need to be converted and pushed into a CernVM-FS repository.

```
cvmfs_ducc convert-single-image image-to-convert repository.cern.ch
```

The command takes two arguments as input, the image to convert and the CernVM-FS repository where to store it.

The *image-to-convert* argument follow the same syntax of the wishlist, for instance it could be something like *https://registry.hub.docker.com/library/fedora:latest*.

Incremental Conversion

The *convert* command will extract image contents into CernVM-FS only where necessary. In general, some parts of the wish list will be already converted while others will need to be converted ex-novo.

An image that has been already unpacked in CernVM-FS will be skipped. For unconverted images, only the missing layers will be unpacked.

Layer Aware

DUCC is now aware that containers images are build incrementally on top of smaller layers.

Converting an image based on an image already inside the repository will skip most of the work.

As long as the lower layers of an image don't change this allows a very fast ingestion of software images, irrespectively of their size.

Notification

DUCC provides a basic notification system to alert external services of updates in the filesystem.

The notifications are appended to a simple text file as JSON objects.

Human operator or software can follow the file and react on notification of interest.

The notification file, eventually can grow large. The suggestion is to treat it as a standard log file with tools like *logrotate*.

Multiple DUCC processes can write on the same notification file at the same time, multiple consumer can read from it.

The notification are activated if and only if the user ask for them providing a file where to write them. To provide a notification file the flag *-n/–notification-file* is available.

Multiprocess

DUCC is able to run multiprocess against the same CernVM-FS repository.

Before to interact with the CernVM-FS repository, DUCC takes a filesystem level lock against */tmp/DUCC.lock*.

This allows to run multiple instances of DUCC at the same time, one instance could listen to a web socket, while one could be doing wishlist conversion.

2.12.7 Setting up an Xcache reverse proxy

This page describes how to set up an experimental HTTP reverse proxy layer for CernVM-FS based on Xcache.

NOTE: This is not a replacement for a general site forward proxy. Forwarding needs to be defined separately in the Xcache configuration for each destination Stratum 1, and the client CVMFS_SERVER_URL configuration has to point to a separate forwarder URL for each server. This document is for the convenience of people who want to experiment with this configuration.

Requirements

- A machine (labeled **Machine A**) to serve the contents of the CernVM-FS repository. Should have CernVM-FS server tools installed, as well as XRootD.
- A second machine (labeled **Machine B**) to use as a reverse proxy. Only XRootD is needed on this machine.
- A CernVM-FS client to mount the repository, for testing.

Instructions

XRootD is a high-performance, scalable file distribution solution. It has a plugin-based architecture and can be configured to suit various use cases. In the Xcache configuration, an XRootD daemon functions as a reverse proxy, serving the contents of a data repository over HTTP.

The following diagram shows how Xcache can be deployed as a cache layer between a CernVM-FS repository and client machines:

Machine A contains a CernVM-FS repository, served by default over HTTP. An Xcache instance is running on a second machine. By default Xcache can only ingest files from another XRootD instance - we start an

instance of XRootD on the same machine as the CernVM-FS repository, configured to export the repository using the XRootD protocol. The following configuration can be used for this instance of XRootD, replacing <CVMFS_REPOSITORY_NAME> with the actually name of the repository:

```
oss.localroot /srv
all.export /cvmfs/<CVMFS_REPOSITORY_NAME> r/o

all.adminpath /var/spool/xrootd
all.pidpath   /var/run/xrootd

xrd.trace all
```

The Xcache instance running on the second machine can be pointed to the XRootD daemon started on the first one (<CVMFS_REPOSITORY_NAME> should be replaced with the actual repository name and MACHINE_A_HOSTNAME with the actual host name of the first machine):

```
all.adminpath /var/spool/xrootd
all.pidpath   /var/run/xrootd

oss.localroot  /data/namespace

all.export /cvmfs/<CVMFS_REPOSITORY_NAME>

oss.space meta /data/xrdcinfos
oss.space data /data/datafiles

xrd.protocol http:3000 /usr/lib64/libXrdHttp.so
xrd.trace all

ofs.osslib   /usr/lib64/libXrdPss.so
pss.cachelib /usr/lib64/libXrdFileCache.so
pss.config streams 32
pss.origin = <MACHINE_A_HOSTNAME>:1094

pfc.ram 4g
pfc.diskusage 0.5 0.6
pfc.spaces data meta
pfc.blocksize 1M
pfc.prefetch 0
pfc.trace info
```

With this configuration, Xcache re-exports the contents of the repository over HTTP, on port 3000. Interested CernVM-FS clients can be configured to use the Xcache instance by modifying the CVMFS_SERVER_URL variable:

```
CVMFS_SERVER_URL=http://<MACHINE_B_HOSTNAME>:3000/cvmfs/<CVMFS_REPOSITORY_NAME>
```

Cache invalidation

A current limitation of Xcache is that cached files are never invalidated. In the context of CernVM-FS, this means that newly published root catalogs are not picked up automatically. An Xcache plugin is being developed to address this limitation.

Ingestion over HTTP

A new XRootD client plugin is being developed to allow the Xcache instance to ingest files over HTTP:

This set up is non-intrusive, as the machine serving the CernVM-FS repository no longer needs to be modified in any way. Xcache could thus be deployed as a reverse proxy layer for existing CernVM-FS stratum servers.

2.12.8 Large-Scale Data CernVM-FS

CernVM-FS primarily is developed for distributing large software stacks. However, by combining several extensions to the base software, one can use CVMFS to distribute large, non-public datasets. While there are several ways to deploy a the service, in this section we outline one potential path to achieve secure distribution of terabytes-to-petabytes of data.

To deploy large-scale CVMFS, a few design decisions are needed:

- **How is data distributed?** For the majority of repositories, data is replicated from a repository server to an existing content distribution network tuned for the object size common to software repositories. The CDNs currently in use are tuned for working set size on the order of tens of gigabytes; they are not appropriately sized for terabytes of data. You will need to put together a mechanism for delivering data at the rates your clients will need.
  - For example, `ligo.osgstorage.org` has about 20TB of data; each scientific workflow utilizes about 2TB of data and each running core averages 1Mbps of input data. So, to support the expected workflows at 10,000 running cores, several 10TB caches were deployed that could export a total of 40Gbps.
  - The `cms.osgstorage.org` repository publishes 3PB of data. Each analysis will read around 20TB and several hundred analyses will run simultaneously. Given the large working set size, there is no caching layer and data is read directly from large repositories.
- **How is data published?** By default, CVMFS publication will calculate checksums on its contents, compresses the data, and serves it from the Apache web server. Implicitly, this means all data must be _copied_ to and _stored_ on the repository host; at larger scales, this is prohibitively expensive. The `cvmfs_swissknife graft` tool provides a mechanism to publish files directly if the checksum is known ahead of time; see *Grafting Files*.
  - For `ligo.osgstorage.org`, a cronjob *copies* all new data to the repository from a cache, creates the checksum file, and immediately deletes the downloaded file. Hence, the LIGO data is copied but not stored.
  - The `cms.osgstorage.org`, a cronjob queries the underlying filesystem for the relevant checksum information and published the checksum. The data is neither copied nor stored on the repository

  On publication, the files may be marked as *non-compressed* and *externally stored*. This allows the CVMFS client to be configured to be pointed at a non-CVMFS data (stored as the "logical name", not the "content addressed" form). CVMFS clients can thus use existing data sources without change.
- **How is data secured?** CVMFS was originally designed to distribute open-source software with strong data integrity guarantees. More recently, read-access authorization has been added to the software. An access

control list is added to the repository (at creation time or publication time) and clients are configured to invoke a plugin for new process sessions. The plugin enforces the ACLs *and* forwards the user's credential back to the CVMFS process. This allows the authorization to be enforced for worker node cache access and the CDN to enforce authorization on the CVMFS process for downloading new files to the cache.

The entire ACL is passed to the external plugin and not interpreted by CVMFS; the semantics are defined by the plugin. The existing plugin is based on GSI / X509 proxies and authorization can be added based on DN or VOMS FQANs.

In order to perform mounts, the root catalog must be accessible without authorization. However, the repository server (or CDN) can be configured to require authorization for the remaining data in the namespace.

Creating Large, Secure Repositories

For large-scale repositories, a few tweaks are useful at creation time. Here is the command used to create the `cms.osgstorage.org`:

```
cvmfs_server mkfs -V cms:/cms -X -Z none -o cmsuser cms.osgstorage.org
```

- The `-V cms:/cms` option indicates that only clients with an X509 proxy with a VOMS extension from CMS are allowed to access the mounted proxy. If multiple VOMS extensions are needed, it's easiest to add this at publication time.
- `-X` indicates that, by default, files published to this repository are served at an "external URL". The clients will attempt to access the file by *name*, not content hash, and look for the server as specified by the client's setting of `CVMFS_EXTERNAL_URL`.
- `-Z none` indicates that, by default, files published to this repository will not be marked as compressed.

By combining the `-X` and `-Z` options, files at an HTTP endpoint can be published in-place: no compression or copying into a different endpoint is necessary to publish.

2.12.9 CernVM-FS Shrinkwrap Utility

The CernVM-FS Shrinkwrap utility provides a means of exporting CVMFS repositories. These exports may consist of the complete repository or contain a curated subset of the repository.

The CernVM-FS shrinkwrap utility uses `libcvmfs` to export repositories to a POSIX file tree. This file tree can then be packaged and exported in several different ways, such as SquashFS, Docker layers, or TAR file. The `cvmfs_shrinkwrap` utility supports multithreaded copying to increase throughput and a file specification to create a subset of a repository.

Installation

The `cvmfs_shrinkwrap` utility is packaged for Red Hat based and Debian based platforms in the `cvmfs-shrinkwrap` package.

In order to compile `cvmfs_shrinkwrap` from sources, use the `-DBUILD_SHRINKWRAP=on` cmake option.

CernVM-FS Shrinkwrap Layout

The structure used in the Shrinkwrap output mirrors that used internally by CernVM-FS. The visible files are hardlinked to a hidden data directory. By default `cvmfs_shrinkwrap` builds in a base directory (`/tmp/cvmfs`) where a directory exists for each repository and a `.data` directory containing the content-addressed files for deduplication.

The shrinkwrap output directory should be formatted with XFS. The ext file systems limit the number of hard links to 64k.

File Path	Description
`/tmp/cvmfs`	**Default base directory** Single mount point that can be used to package repositories, containing both the directory tree and the data directory.
`<base>/<fqrn>`	**Repository file tree** Directory containing the visible structure and file names for a repository.
`<base>/.data`	**File storage location for repositories** Content-addressed files in a hidden directory.
`<base>/.provenance`	**Storage location for provenance** Hidden directory that stores the provenance information, including `libcvmfs` configurations and specification files.

Specification File

The specification file allows for both positive entries and exlusion statements. Inclusion can be specified directly for each file, can use wildcards for directories trees, and an anchor to limit to only the specified directory. Directly specify file :

```
/lcg/releases/gcc/7.1.0/x86_64-centos7/setup.sh
```

Specify directory tree :

```
/lcg/releases/ROOT/6.10.04-4c60e/x86_64-cenots7-gcc7-opt/*
```

Specify only directory contents :

```
^/lcg/releases/*
```

Negative entries will be left out of the traversal :

```
!/lcg/releases/uuid
```

Creating an image for ROOT

Start out with either building `cvmfs_shrinkwrap`, adding it to your path, or locating it in your working directory.

Create a file specification to limit the files subject to being shrinkwrapped. Here is an example for ROOT version 6.10 (~8.3 GB). For our example put this in a file named `sft.cern.ch.spec`.

```
/lcg/releases/ROOT/6.10.04-4c60e/x86_64-centos7-gcc7-opt/*
/lcg/contrib/binutils/2.28/x86_64-centos7/lib/*
/lcg/contrib/gcc/*
/lcg/releases/gcc/*
/lcg/releases/lcgenv/*
```

Write the `libcvmfs` configuration file that will be used for `cvmfs_shrinkwrap`. `cvmfs_shrinkwrap` puts a heavy load on servers, so please do not configure it to read from production Stratum 1s. CERN provides a separate server at http://cvmfs-stratum-zero-hpc.cern.ch and OSG provides one at http://cvmfs-s1goc.opensciencegrid.org:8001. Here is an example that uses the CERN server, written to `sft.cern.ch.config`.

```
CVMFS_REPOSITORIES=sft.cern.ch
CVMFS_REPOSITORY_NAME=sft.cern.ch
CVMFS_CONFIG_REPOSITORY=cvmfs-config.cern.ch
CVMFS_SERVER_URL='http://cvmfs-stratum-zero-hpc.cern.ch/cvmfs/sft.cern.ch'
CVMFS_HTTP_PROXY=DIRECT # Avoid filling up any local squid's cache
CVMFS_CACHE_BASE=/var/lib/cvmfs/shrinkwrap
CVMFS_KEYS_DIR=/etc/cvmfs/keys/cern.ch # Need to be provided for shrinkwrap
CVMFS_SHARED_CACHE=no # Important as libcvmfs does not support shared caches
CVMFS_USER=cvmfs
```

Note: Keys will need to be provided. The location in this configuration is the default used for CVMFS with FUSE.

Using the cvmfs repository `sft.cern.ch` :

```
sudo cvmfs_shrinkwrap -r sft.cern.ch -f sft.cern.ch.config -t sft.cern.ch.spec --dest-
↪base /tmp/cvmfs -j 16
```

Creating an image in userspace

Start by using the above setup.

Alternatively, shrinkwrap images can be created in user space. This is achieved using the UID and GID mapping feature of `libcvmfs`. First mapping files need to be written.

Example (Assuming UID 1000). Write `* 1000` into `uid.map` at `/tmp/cvmfs`. Add this rule `sft.cern.ch.config`. :

```
CVMFS_UID_MAP=/tmp/cvmfs/uid.map
```

The same is done with GID into `gid.map`.

Using the cvmfs repository `sft.cern.ch` :

```
cvmfs_shrinkwrap -r sft.cern.ch -f sft.cern.ch.config -t sft.cern.ch.spec --dest-base␣
↪/tmp/cvmfs -j 16
```

Note on CernVM-FS Variant Symlinks

CernVM-FS variant symlinks that are used in the organization of repositories are evaluated at the time of image creation. As such, the OS the image is created on should be the expected OS the image will be used with. Specification rules can be written to include other OS compatible version, but symlinks will resolve to the original OS.

Using a shrinkwrap image

Shrinkwrap was developed to address similar restrictions as the CVMFS Preloader. Having created an image from your specification there are a number of ways this can be used and moved around.

Exporting image

Having a fully loaded repository, including the hardlinked data, the image can be exported to a number of different formats and packages. Some examples of this could be ZIP, tarballs, or squashfs. The recommendation is to use squashfs as it provides a great amount of portability and is supported for directly mounting on most OS.

If tools for creating squashfs are not already available try :

```
apt-get install squashfs-tools
```

– or –

```
yum install squashfs-tools
```

After this has been install a squashfs image can be created using the above image :

```
mksquashfs /tmp/cvmfs root-sft-image.sqsh
```

This process may take time to create depending on the size of the shrinkwrapped image. The squashfs image can now be moved around and mounted using :

```
mount -t squashfs /PATH/TO/IMAGE/root-sft-image.sqsh /cvmfs
```

Bind mounting an image

The shrinkwrap image can also be directly moved and mounted using bind mounts.

```
mount --bind /tmp/cvmfs /cvmfs
```

This provides a quick method for testing created images and verifying the contents will run your expected workload.

Important note on use

Shrinkwrap images mirror the data organization of CVMFS. As such it is important that the data and the filesystem tree be co-located in the filesystem/mountpoint. If the data is separated from the filesystem tree you are likely to encounter an error.

2.12.10 Implementation Notes

CernVM-FS has a modular structure and relies on several open source libraries. Figure *below* shows the internal building blocks of CernVM-FS. Most of these libraries are shipped with the CernVM-FS sources and are linked statically in order to facilitate debugging and to keep the system dependencies minimal.

File Catalog

A CernVM-FS repository is defined by its *file catalog*. The file catalog is an SQLite database [Allen10] having a single table that lists files and directories together with its metadata. The table layout is shown in the table below:

Field	Type
Path MD5	128Bit Integer
Parent Path MD5	128Bit Integer
Hardlinks	Integer
Content Hash	BLOB
Size	Integer
Mode	Integer
Last Modified	Timestamp
Flags	Integer
Name	String
Symlink	String
uid	Integer
gid	Integer
xattr	BLOB

In order to save space we do not store absolute paths. Instead we store MD5 [Rivest92], [Turner11] hash values of the absolute path names. Symbolic links are kept in the catalog. Symbolic links may contain environment variables in the form `$(VAR_NAME)` or `$(VAR_NAME:-/default/path)` that will be dynamically resolved by CernVM-FS on access. Hardlinks are emulated by CernVM-FS. The hardlink count is stored in the lower 32 bits of the hardlinks field, and a *hardlink group* is stored in the higher 32 bits. If the hardlink group is greater than zero, all files with the same hardlink group will get the same inode issued by the CernVM-FS Fuse client. The emulated hardlinks work within the same directory, only. The cryptographic content hash refers to the zlib-compressed [Deutsch96] version of the file. Flags indicate the type of an directory entry (see table *below*).

Extended attributes are either NULL or stored as a BLOB of key-value pairs. It starts with 8 bytes for the data structure's version (currently 1) followed by 8 bytes for the number of extended attributes. This is followed by the list of pairs, which start with two 8 byte values for the length of the key/value followed by the concatenated strings of the key and the value.

Flags	Meaning
1	Directory
2	Transition point to a nested catalog
33	Root directory of a nested catalog
4	Regular file
8	Symbolic link
68	Chunked file
132	External file (stored under path name)

As of bit 8, the flags store the cryptographic content hash algorithm used to process the given file. Bit 11 is 1 if the file is stored uncompressed.

A file catalog contains a *time to live* (TTL), stored in seconds. The catalog TTL advises clients to check for a new version of the catalog, when expired. Checking for a new catalog version takes place with the first file system operation on a CernVM-FS volume after the TTL has expired. The default TTL is 4 minutes. If a new catalog is available, CernVM-FS delays the loading for the period of the CernVM-FS kernel cache life time (default: 1 minute). During this drain-out period, the kernel caching is turned off. The first file system operation on a CernVM-FS volume after that additional delay will apply a new file catalog and kernel caching is turned back on.

Content Hashes

CernVM-FS can use SHA-1 [Jones01], RIPEMD-160 [Dobbertin96] and SHAKE-128 [Bertoni09] as cryptographic hash function. The hash function can be changed on the Stratum 0 during the lifetime of repositories. On a change, new and updated files will use the new cryptographic hash while existing files remain unchanged. This is transparent to the clients since the hash function is stored in the flags field of file catalogs for each and every file. The default hash function is SHA-1. New software versions might introduce support for further cryptographic hash functions.

Nested Catalogs

In order to keep catalog sizes reasonable[1], repository subtrees may be cut and stored as separate *nested catalogs*. There is no limit on the level of nesting. A reasonable approach is to store separate software versions as separate nested catalogs. The figure *below* shows the simplified directory structure which we use for the ATLAS repository.

Fig. 5: Directory structure used for the ATLAS repository (simplified).

When a subtree is moved into a nested catalog, its entry directory serves as *transition point* for nested catalogs. This directory appears as empty directory in the parent catalog with flags set to 2. The same path appears as root-directory in the nested catalog with flags set to 33. Because the MD5 hash values refer to full absolute paths, nested catalogs store the root path prefix. This prefix is prepended transparently by CernVM-FS. The cryptographic hash of nested catalogs is stored in the parent catalog. Therefore, the root catalog fully defines an entire repository.

Loading of nested catalogs happens on demand by CernVM-FS on the first attempt to access of anything inside, a user won't see the difference between a single large catalog and several nested catalogs. While this usually avoids unnecessary catalogs to be loaded, recursive operations like `find` can easily bypass this optimization.

Catalog Statistics

A CernVM-FS file catalog maintains several counters about its contents and the contents of all of its nested catalogs. The idea is that the catalogs know how many entries there are in their sub catalogs even without opening them. This way, one can immediately tell how many entries, for instance, the entire ATLAS repository has. Some of the numbers are shown using the number of inodes in `statvfs`. So `df -i` shows the overall number of entries in the repository and (as number of used inodes) the number of entries of currently loaded catalogs. Nested catalogs create an additional entry (the transition directory is stored in both the parent and the child catalog). File hardlinks are still individual entries (inodes) in the cvmfs catalogs. The following counters are maintained for both a catalog itself and for the subtree this catalog is root of:

- Number of regular files
- Number of symbolic links
- Number of directories
- Number of nested catalogs
- Number of external files
- Number of chunked files
- Number of individual file chunks
- Overall file content size
- File content size stored in chunked files

[1] As a rule of thumb, file catalogs (when compressed) are reasonably small.

Repository Manifest (.cvmfspublished)

Every CernVM-FS repository contains a repository manifest file that serves as entry point into the repository's catalog structure. The repository manifest is the first file accessed by the CernVM-FS client at mount time and therefore must be accessible via HTTP on the repository root URL. It is always called **.cvmfspublished** and contains fundamental repository meta data like the root catalog's cryptographic hash and the repository revision number as a key-value list.

Internal Manifest Structure

Below is an example of a typical manifest file. Each line starts with a capital letter specifying the meta data field, followed by the actual data string. The list of meta information is ended by a separator line (`--`) followed by signature information further described *here*.

```
C64551dccfbe0a48de7618dd7deb290200b474759
B1442336
Rd41d8cd98f00b204e9800998ecf8427e
D900
S42
Nexample.cern.ch
X731cca9476eb882f5a3f24aaa38001105a0e35eb
T1390301299
--
edde5308e502dd5e8fe405c56f5700f7477dc319
[...]
```

Please refer to table below for detailed information about each of the meta data fields.

Field	Meta Data Description
`C`	Cryptographic hash of the repository's current root catalog
`B`	Size of the root file catalog in bytes
`A`	"yes" if the catalog should be fetched under its alternative name (outside servers /data directory)
`R`	MD5 hash of the repository's root path (usually always `d41d8cd98f00b204e9800998ecf8427e`)
`X`	Cryptographic hash of the signing certificate
`G`	"yes" if the repository is garbage-collectable
`H`	Cryptographic hash of the repository's named tag history database
`T`	Unix timestamp of this particular revision
`D`	Time To Live (TTL) of the root catalog
`S`	Revision number of this published revision
`N`	The full name of the manifested repository
`M`	Cryptographic hash of the repository JSON metadata
`Y`	Cryptographic hash of the reflog checksum
`L`	currently unused (reserved for micro catalogs)

Repository Signature

In order to provide authoritative information about a repository publisher, the repository manifest is signed by an X.509 certificate together with its private key.

Signing a Repository

It is important to note that it is sufficient to sign just the manifest file itself to gain a secure chain of the whole repository. The manifest refers to the cryptographic content hash of the root catalog which in turn recursively references all sub-catalogs with their cryptographic content hashes. Each catalog lists its files along with their cryptographic content hashes. This concept is called a merkle tree and eventually provides a single hash that depends on the *complete* content of the repository.

The top level hash used for the repository signature can be found in the repository manifest right below the separator line (`--` / *see above*). It is the cryptographic hash of the manifest's meta data lines excluding the separator line. Following the top level hash is the actual signature produced by the X.509 certificate signing procedure in binary form.

Signature Validation

In order to validate repository manifest signatures, CernVM-FS uses a white-list of valid publisher certificates. The white-list contains the cryptographic fingerprints of known publisher certificates and a timestamp. A white-list is valid for 30 days. It is signed by a private RSA key, which we refer to as *master key*. The public RSA key that corresponds to the master key is distributed with the `cvmfs-config-...` RPMs as well as with every instance of CernVM.

As crypto engine, CernVM-FS uses libcrypto from the OpenSSL project.

Blacklisting

In addition to validating the white-list, CernVM-FS checks certificate fingerprints against the local black-list /etc/cvmfs/blacklist and the blacklist in an optional *"Config Repository"*. The blacklisted fingerprints have to be in the same format as the fingerprints on the white-list. The black-list has precedence over the white-list.

Blacklisted fingerprints prevent clients from loading future repository publications by a corresponding compromised repository key, but they do not prevent mounting a repository revision that had previously been mounted on a client, because the catalog for that revision is already in the cache. However, the same blacklist files also support another format that actively blocks revisions associated with a compromised repository key from being mounted and even forces them to be unmounted if they are mounted. The format for that is a less-than sign followed by the repository name followed by a blank and a repository revision number:

```
<repository.name NNN
```

This will prevent all revisions of a repository called repository.name less than the number NNN from being mounted or staying mounted. An effective protection against a compromised repository key will use both this format to prevent mounts and the fingerprint format to prevent accepting future untrustworthy publications signed by the compromised key.

Use of HTTP

The particular way of using the HTTP protocol has significant impact on the performance and usability of CernVM-FS. If possible, CernVM-FS tries to benefit from the HTTP/1.1 features keep-alive and cache-control. Internally, CernVM-FS uses the libcurl library.

The HTTP behaviour affects a system with cold caches only. As soon as all necessary files are cached, there is only network traffic when a catalog TTL expires. The CernVM-FS download manager runs as a separate thread that handles download requests asynchronously in parallel. Concurrent download requests for the same URL are collapsed into a single request.

DoS Protection

A subtle denial of service attack (DoS) can occur when CernVM-FS is successfully able to download a file but fails to store it in the local cache. This situation escalates into a DoS when the application using CernVM-FS remains in an endless loop and tries to open a file over and over again. Such a situation is prevented by CernVM-FS by re-trying with an exponential backoff. The backoff is triggered by consecutive failures to cache a downloaded file within 10 seconds.

Keep-Alive

Although the HTTP protocol overhead is small in terms of data volume, in high latency networks we suffer from the bare number of requests: Each request-response cycle has a penalty of at least the network round trip time. Using plain HTTP/1.0, this results in at least $3 \cdot$ round trip time additional running time per file download for TCP handshake, HTTP GET, and TCP connection finalisation. By including the `Connection: Keep-Alive` header into HTTP requests, we advise the HTTP server end to keep the underlying TCP connection opened. This way, overhead ideally drops to just round trip time for a single HTTP GET. The impact of the keep-alive feature is shown in here.

This feature, of course, somewhat sabotages a server-side load-balancing. However, exploiting the HTTP keep-alive feature does not affect scalability per se. The servers and proxies may safely close idle connections anytime, in particular if they run out of resources.

Cache Control

In a limited way, CernVM-FS advises intermediate web caches how to handle its requests. Therefore it uses the `Pragma: no-cache` and the `Cache-Control: no-cache` headers in certain cases. These cache control headers apply to both, forward proxies as well as reverse proxies. This is not a guarantee that intermediate proxies fetch a fresh original copy (though they should).

By including these headers, CernVM-FS tries to not fetch outdated cache copies. Only in case CernVM-FS downloads a corrupted file from a proxy server, it retries having the HTTP `no-cache` header set. This way, the corrupted file gets replaced in the proxy server by a fresh copy from the backend.

Identification Header

CernVM-FS sends the `User-Agent` header set to either of `libcvmfs` or `Fuse` depending on how it was compiled, plus the current `VERSION` value. If the `CERNVM_UUID` environment variable is set, that's also included in the `User-Agent` field.

Redirects

Normally, the Stratum-1 servers directly respond to HTTP requests so CernVM-FS has no need to support HTTP redirect response codes. However, there are some high-bandwidth applications where HTTP redirects are used to transfer requests to multiple data servers. To enable support for redirects in the CernVM-FS client, set `CVMFS_FOLLOW_REDIRECTS=yes`.

Name Resolving

Round-robin DNS entries for proxy servers are treated specially by CernVM-FS. Multiple IP addresses for the same proxy name are automatically transformed into multiple proxy servers within the same load-balance group. So the usual rules for load-balancing and fail-over apply to the different servers in a round-robin entry. CernVM-FS resolves all the proxy servers at once (and in parallel) at mount time. From that point on, proxy server names are resolved on demand, when a download takes place and the TTL of the active proxy expired. CernVM-FS resolves using /etc/host (resp. the file referenced in the `HOST_ALIASES` environment variable) or, if a host name is not resolvable locally, it uses the c-ares resolver. Proxy servers given in IP notation remain unchanged.

CernVM-FS uses the TTLs that come from DNS servers. However, there is a cutoff at 1 minute minimum TTL and 1 day maximum TTL. Locally resolved host names get a TTL of 1 minute. The host alias file is re-read with every attempt to resolve a name. Failed attempts to resolve a name remain cached for 1 minute, too. If a name has been successfully resolved previously, this result stays active until another successful attempt is done. If the DNS entries change for a host name, CernVM-FS adjust the corresponding load-balance group and picks a new server from the group at random.

The name resolving silently ignores errors in individual records. Only if no valid IP address is returned at all it counts as an error. IPv4 addresses have precedence if available. If the `CVMFS_IPV4_ONLY` environment variable is set, CernVM-FS does not try to resolve IPv6 records.

The timeout for name resolving is hard-coded to 2 attempts with a timeout of 3 seconds each. This is independent from the `CVMFS_TIMEOUT` and `CVMFS_TIMEOUT_DIRECT` settings. The effective timeout can be a bit longer than 6 seconds because of a backoff.

The name server used by CernVM-FS is looked up only once on start. If the name server changes during the life time of a CernVM-FS mount point, this change needs to be manually advertised to CernVM-FS using `cvmfs_talk nameserver set`.

Disk Cache

Each running CernVM-FS instance requires a local cache directory. Data are downloaded into a temporary files. Only at the very latest point they are renamed into their content-addressable names atomically by `rename()`.

The hard disk cache is managed, CernVM-FS maintains cache size restrictions and replaces files according to the least recently used (LRU) strategy [Panagiotou06]. In order to keep track of files sizes and relative file access times, CernVM-FS sets up another SQLite database in the cache directory, the *cache catalog*. The cache catalog contains a single table; its structure is shown here:

Field	Type
Hash	String (hex notation)
Size	Integer
Access Sequence	Integer
Pinned	Integer
File type (chunk or file catalog)	Integer

CernVM-FS does not strictly enforce the cache limit. Instead CernVM-FS works with two customizable soft limits, the *cache quota* and the *cache threshold*. When exceeding the cache quota, files are deleted until the overall cache size is less than or equal to the cache threshold. The cache threshold is currently hard-wired to half of the cache quota. The cache quota is for data files as well as file catalogs. Currently loaded catalogs are pinned in the cache, they will not be deleted until unmount or until a new repository revision is applied. On unmount, pinned file catalogs are updated with the highest sequence number. As a pre-caution against a cache that is blocked by pinned catalogs, all catalogs except the root catalog are unpinned when the volume of pinned catalogs exceeds the overall cache volume.

The cache catalog can be re-constructed from scratch on mount. Re-constructing the cache catalog is necessary when the managed cache is used for the first time and every time when “unmanaged” changes occurred to the cache directory, when CernVM-FS was terminated unexpectedly.

In case of an exclusive cache, the cache manager runs as a separate thread of the `cvmfs2` process. This thread gets notified by the Fuse module whenever a file is opened or inserted. Notification is done through a pipe. The shared cache uses the very same code, except that the thread becomes a separate process (see Figure *below*). This cache manager process is not another binary but `cvmfs2` forks to itself with special arguments, indicating that it is supposed to run as a cache manager. The cache manager does not need to be started as a service. The first CernVM-FS instance that uses a shared cache will automatically spawn the cache manager process. Subsequent CernVM-FS instances will connect to the pipe of this cache manager. Once the last CernVM-FS instance that uses the shared cache is unmounted, the communication pipe is left without any writers and the cache manager automatically quits.

The CernVM-FS cache supports two classes of files with respect to the cache replacement strategy: *normal* files and *volatile* files. The sequence numbers of volatile files have bit 63 set. Hence they are interpreted as negative numbers and have precedence over normal files when it comes to cache cleanup. On automatic rebuild the volatile property of entries in the cache database is lost.

NFS Maps

In normal mode, CernVM-FS issues inodes based on the row number of an entry in the file catalog. When exported via NFS, this scheme can result in inconsistencies because CernVM-FS does not control the cache lifetime of NFS clients. A once issued inode can be asked for anytime later by a client. To be able to reply to such client queries even after reloading catalogs or remounts of CernVM-FS, the CernVM-FS *NFS maps* implement a persistent store of the path names $\mapsto$ inode mappings. Storing them on hard disk allows for control of the CernVM-FS memory consumption (currently $\approx$ 45 MB extra) and ensures consistency between remounts of CernVM-FS. The performance penalty for doing so is small. CernVM-FS uses Google's leveldb, a fast, local key value store. Reads and writes are only performed when meta-data are looked up in SQLite, in which case the SQLite query supposedly dominates the running time.

A drawback of the NFS maps is that there is no easy way to account for them by the cache quota. They sum up to some 150-200 Bytes per path name that has been accessed. A recursive `find` on /cvmfs/atlas.cern.ch with 50 million entries, for instance, would add up 8GB in the cache directory. This is mitigated by the fact that the NFS mode will be only used on few servers that can be given large enough spare space on hard disk.

Loader

The CernVM-FS Fuse module comprises a minimal *loader* loader process (the `cvmfs2` binary) and a shared library containing the actual Fuse module (`libcvmfs_fuse.so`, `libcvmfs_fuse3.so`). This structure makes it possible to reload CernVM-FS code and parameters without unmounting the file system. Loader and library don't share any symbols except for two global structs `cvmfs_exports` and `loader_exports` used to call each others functions. The loader process opens the Fuse channel and implements stub Fuse callbacks that redirect all calls to the CernVM-FS shared library. Hotpatch is implemented as unloading and reloading of the shared library, while the loader temporarily queues all file system calls in-between. Among file system calls, the Fuse module has to keep very little state. The kernel caches are drained out before reloading. Open file handles are just file descriptors that are held open by the process. Open directory listings are stored in a Google dense_hash that is saved and restored.

File System Interface

CernVM-FS implements the following read-only file system call-backs.

mount

On mount, the file catalog has to be loaded. First, the file catalog *manifest* `.cvmfspublished` is loaded. The manifest is only accepted on successful validation of the signature. In order to validate the signature, the certificate and the white-list are downloaded in addition if not found in cache. If the download fails for whatever reason, CernVM-FS tries to load a local file catalog copy. As long as all requested files are in the disk cache as well, CernVM-FS continues to operate even without network access (*offline mode*). If there is no local copy of the manifest or the downloaded manifest and the cache copy differ, CernVM-FS downloads a fresh copy of the file catalog.

getattr and lookup

Requests for file attributes are entirely served from the mounted catalogs, there is no network traffic involved. This function is called as pre-requisite to other file system operations and therefore the most frequently called Fuse callback. In order to minimize relatively expensive SQLite queries, CernVM-FS uses a hash table to store negative and positive query results. The default size for this memory cache is determined according to benchmarks with LHC experiment software.

Additionally, the callback takes care of the catalog TTL. If the TTL is expired, the catalog is re-mounted on the fly. Note that a re-mount might possibly break running programs. We rely on careful repository publishers that produce more or less immutable directory trees, new repository versions just add files.

If a directory with a nested catalog is accessed for the first time, the respective catalog is mounted in addition to the already mounted catalogs. Loading nested catalogs is transparent to the user.

readlink

A symbolic link is served from the file catalog. As a special extension, CernVM-FS detects environment variables in symlink strings written as `$(VARIABLE)` or `$(VARIABLE:-/default/path)`. These variables are expanded by CernVM-FS dynamically on access (in the context of the `cvmfs2` process). This way, a single symlink can point to different locations depending on the environment. This is helpful, for instance, to dynamically select software package versions residing in different directories.

readdir

A directory listing is served by a query on the file catalog. Although the "parent"-column is indexed (see *Catalog table schema*), this is a relatively slow function. We expect directory listing to happen rather seldom.

open / read

The `open()` call has to provide a file descriptor for a given path name. In CernVM-FS file requests are always served from the disk cache. The Fuse file handle is a file descriptor valid in the context of the CernVM-FS process. It points into the disk cache directory. Read requests are translated into the `pread()` system call.

getxattr

CernVM-FS uses synthetic extended attributes to display additional repository information. In general they can be displayed with a command like

```
attr -g <attributename> /cvmfs/<repository>
```

There are the following supported magic attributes:

catalog_counters
: Like `repo_counters` but only for the nested catalog that hosts the given path.

chunks
: Number of chunks of a regular file.

chunk_list
: Hashes and sizes of the chunks of a regular (large) file.

compression
: Compression algorithm, for regular files only. Either “zlib” or “none”.

expires
: Shows the remaining life time of the mounted root file catalog in minutes.

external_file
: Indicates if a regular file is an external file or not. Either 0 or 1.

external_host
: Like `host` but for the host settings to fetch external files.

external_timeout
: Like `timeout` but for the host settings to fetch external files.

fqrn
: Shows the fully qualified repository name of the mounted repository.

hash
: Shows the cryptographic hash of a regular file as listed in the file catalog.

host
: Shows the currently active HTTP server.

host_list
: Shows the ordered list of HTTP servers.

inode_max
: Shows the highest possible inode with the current set of loaded catalogs.

lhash
: Shows the cryptographic hash of a regular file as stored in the local cache, if available.

logbuffer
: Shows system log messages for the repository.

maxfd
: Shows the maximum number of file descriptors available to file system clients.

ncleanup24
: Shows the number of cache cleanups in the last 24 hours.

nclg
: Shows the number of currently loaded nested catalogs.

ndiropen
: Shows the overall number of opened directories.

ndownload
: Shows the overall number of downloaded files since mounting.

nioerr
Shows the total number of I/O errors encoutered since mounting.

nopen
Shows the overall number of `open()` calls since mounting.

pid
Shows the process id of the CernVM-FS Fuse process.

proxy
Shows the currently active HTTP proxy.

pubkeys
The loaded public RSA keys used for repository whitelist verification.

rawlink
Shows unresolved variant symbolic links; only accessible from the root attribute namespace (use *attr -Rg rawlink*).

repo_counters
Shows the aggregate counters of the repository contents (number of files etc.)

repo_metainfo
Shows the *repository meta info* file, if available

revision
Shows the file catalog revision of the mounted root catalog, an auto-increment counter increased on every repository publish.

root_hash
Shows the cryptographic hash of the root file catalog.

rx
Shows the overall amount of downloaded kilobytes.

speed
Shows the average download speed.

tag
The configured repository tag.

timeout
Shows the timeout for proxied connections in seconds.

timeout_direct
Shows the timeout for direct connections in seconds.

uptime
Shows the time passed since mounting in minutes.

usedfd
Shows the number of file descriptors currently issued to file system clients.

version
Shows the version of the loaded CernVM-FS binary.

Extended attributes can be queried using the `attr` command. For instance, `attr -g hash /cvmfs/atlas.cern.ch/ChangeLog` returns the cryptographic hash of the file at hand. The extended attributes are used by the `cvmfs_config stat` command in order to show a current overview of health and performance numbers.

Repository Publishing

Repositories are not immutable, every now and then they get updated. This might be installation of a new release or a patch for an existing release. But, of course, each time only a small portion of the repository is touched, say out of . In order not to re-process an entire repository on every update, we create a read-write file system interface to a CernVM-FS repository where all changes are written into a distinct scratch area.

Read-write Interface using a Union File System

Union file systems combine several directories into one virtual file system that provides the view of merging these directories. These underlying directories are often called *branches*. Branches are ordered; in the case of operations on paths that exist in multiple branches, the branch selection is well-defined. By stacking a read-write branch on top of a read-only branch, union file systems can provide the illusion of a read-write file system for a read-only file system. All changes are in fact written to the read-write branch.

Preserving POSIX semantics in union file systems is non-trivial; the first fully functional implementation has been presented by Wright et al. [Wright04]. By now, union file systems are well established for "Live CD" builders, which use a RAM disk overlay on top of the read-only system partition in order to provide the illusion of a fully read-writable system. CernVM-FS supports both aufs and OverlayFS union file systems.

Union file systems can be used to track changes on CernVM-FS repositories (Figure *below*). In this case, the read-only file system interface of CernVM-FS is used in conjunction with a writable scratch area for changes.

Fig. 6: A union file system combines a CernVM-FS read-only mount point and a writable scratch area. It provides the illusion of a writable CernVM-FS mount point, tracking changes on the scratch area.

Based on the read-write interface to CernVM-FS, we create a feed-back loop that represents the addition of new software releases to a CernVM-FS repository. A repository in base revision r is mounted in read-write mode on the publisher's end. Changes are written to the scratch area and, once published, are re-mounted as repository revision $r + 1$. In this way, CernVM-FS provides snapshots. In case of errors, one can safely resume from a previously committed revision.

2.13 Appendix

2.13.1 Security Considerations

CernVM-FS provides end-to-end data integrity and authenticity using a signed Merkle Tree. CernVM-FS clients verify the signature and the content hashes of all downloaded data. Once a particular revision of a file system is stored in a client's local cache, the client will not apply an older revision anymore.

The public key used to ultimately verify a repository's signature needs to be distributed to clients through a channel different from CernVM-FS content distribution. In practice, these public keys are distributed as part of the source code or through `cvmfs-config-...` packages. One or multiple public keys can be configured for a repository (the *fully qualified repository name*), all repositories within a specific domain (like `*.cern.ch`) or all repositories (`*`). If multiple keys are configured, it is sufficient if any of them validates a signature.

Besides the client, data is also verified by the replication code (Stratum 1 or preloaded cache) and by the release manager machine in case the repository is stored in S3 and not on a local file system.

CernVM-FS does **not** provide data confidentiality out of the box. By default data is transferred through HTTP and thus only public data should be stored on CernVM-FS. However, CernVM-FS can be operated with HTTPS data transport. In combination with client-authentication using an authz helper (see Section *Authorization Helpers*), CernVM-FS can be configured for end-to-end data confidentiality.

Once downloaded and stored in a cache, the CernVM-FS client fully trusts the cache. Data in the cache can be checked for silent corruption but no integrity re-check takes place.

Signature Details

Creating and validating a repository signature is a two-step process. The *repository manifest* (the file `.cvmfspublished`) is signed by a private RSA key whose public part is stored in the form of an X.509 certificate in the repository. The fingerprint of all certificates that are allowed to sign a repository is stored on a *repository whitelist* (the file `.cvmfswhitelist`). The whitelist is signed with a different RSA key, the *repository master key*. Only the public part of this master key needs to be distributed to clients.

The X.509 certificate currently only serves as an envelope for the public part of a repository key. No further certificate validation takes place.

The repository manifest contains, among other information, the content hash of the root file catalog, the content hash of the signing certificate, the fully qualified repository name, and a timestamp. In order to sign the manifest, the content of the manifest is hashed and encrypted with a private repository key. The timestamp and repository name are used prevent replay attacks.

The whitelist contains the fully qualified repository name, a creation timestamp, an expiry timestamp, and the certificate fingerprints. Since the whitelist expires, it needs to be regularly resigned.

The private part of the repository key needs to be accessible on the release manager machine. The private part of the repository master key used to sign the whitelist *can* be maintained on a file on the release manager machine. We recommend, however, to use a smart card to store this private key. See section *Master keys* for more details.

Content Hashes

CernVM-FS supports multiple content hash algorithms: SHA-1 (default), RIPEMD-160, and SHAKE-128 with 160 output bits. The content hash algorithm can be changed with every repository publish operation. Files and file catalogs hashed with different content hash algorithms can co-exist. On changing the algorithm, new and changed files are hashed with the new algorithm, existing data remains unchanged. That allows seamless migration from one algorithm to another.

Local UNIX Permissions

Most parts of CernVM-FS do not require root privileges. On the server side, only creating and deleting a repository (or replica) requires root privileges. Repository transactions and snapshots can be performed with an unprivileged user account. In order to remount a new file system revision after publishing a transaction, the release manager machines uses a custom suid binary.

On client side, the CernVM-FS fuse module is normally started as root. It drops root privileges and changes the persona to the `cvmfs` user early in the file system initialization. The client RPM package installs SElinux rules for RHEL6 and RHEL7. The cache directory should be labeled as `cvmfs_cache_t`.

Running the client as a normal user

The client can also be started as a normal user. In this case, the user needs to have access to /dev/fuse. On Linux kernels < 4.18, mounting /dev/fuse is either performed by fuse's `fusermount` utility or through a pre-mounted file descriptor. On newer Linux kernels, the client can mount as an unprivileged user in a user namespace with a detached mount namespace.

The easiest way to run the client as a normal user is with the cvmfsexec package. It supports four ways to run cvmfs as an unprivileged user, depending on the capabilities available on the host. See the README there for details.

SETUID bit and file capabilities

By default, CernVM-FS repositories are mounted with the `nosuid` option. Therefore, file capabilities and the setuid bit of files in the repository are ignored. The root user can decide to mount a CernVM-FS repository with the `cvmfs_suid` option, in which case the original behavior of the suid flag and file capabilities is restored.

CernVM-FS Software Distribution

CernVM-FS software is distributed through HTTPS in packages. There are yum and apt repositories for Linux and `pkg` packages for OS X. Software is available from HTTPS servers. The Linux packages and repositories are signed with a GPG key.

2.13.2 CernVM-FS Parameters

Client parameters

Parameters recognized in configuration files under /etc/cvmfs:

Parameter	Meaning
CVMFS_ALIEN_CACHE	If set, use an alien cache at the given location
CVMFS_ALT_ROOT_PATH	If set to *yes*, use alternative root catalog path. Only required for fixed catalogs (tag / hash) under the alternative path.
CVMFS_AUTO_UPDATE	If set to *no*, disables the automatic update of file catalogs.
CVMFS_AUTHZ_HELPER	Full path to an authz helper, overwrites the helper hint in the catalog.
CVMFS_AUTHZ_SEARCH_PATH	Full path to the directory that contains the authz helpers.
CVMFS_BACKOFF_INIT	Seconds for the maximum initial backoff when retrying to download data.
CVMFS_BACKOFF_MAX	Maximum backoff in seconds when retrying to download data.
CVMFS_CATALOG_WATERMARK	Try to release pinned catalogs when their number surpasses the given watermark. Defaults to 1/4 CVMFS_NFILES; explicitly set by shrinkwrap.
CVMFS_CACHE_BASE	Location (directory) of the CernVM-FS cache.
CVMFS_CHECK_PERMISSIONS	If set to *no*, disable checking of file ownership and permissions (open all files).
CVMFS_CLAIM_OWNERSHIP	If set to *yes*, allows CernVM-FS to claim ownership of files and directories.
CVMFS_DEBUGLOG	If set, run CernVM-FS in debug mode and write a verbose log the the specified file.
CVMFS_DEFAULT_DOMAIN	The default domain will be automatically appended to repository names when given without a domain.

continues on next page

Table 1 – continued from previous page

Parameter	Meaning
CVMFS_DNS_MIN_TTL	Minimum effective TTL in seconds for DNS queries of proxy server names (not Stratum 1s). Defaults to 1 minute.
CVMFS_DNS_MAX_TTL	Maximum effective TTL in seconds for DNS queries of proxy server names (not Stratum 1s). Defaults to 1 day.
CVMFS_DNS_RETRIES	Number of retries when resolving proxy names
CVMFS_DNS_TIMEOUT	Timeout in seconds when resolving proxy names
CVMFS_DNS_ROAMING	If true, watch /etc/resolv.conf for nameserver changes
CVMFS_ENFORCE_ACLS	Enforce POSIX ACLs stored in the repository. Requires libfuse 3.
CVMFS_EXTERNAL_FALLBACK_PROXY	List of HTTP proxies similar to `CVMFS_EXTERNAL_HTTP_PROXY`. The fallback proxies are added to the end of the normal proxies, and disable DIRECT connections.
CVMFS_EXTERNAL_HTTP_PROXY	Chain of HTTP proxy groups to be used when CernVM-FS is accessing external data
CVMFS_EXTERNAL_MAX_SERVERS	Caps the list of external hosts to the given number (after geo-sorting them)
CVMFS_EXTERNAL_TIMEOUT	Timeout in seconds for HTTP requests to an external-data server with a proxy server
CVMFS_EXTERNAL_TIMEOUT_DIRECT	Timeout in seconds for HTTP requests to an external-data server without a proxy server
CVMFS_EXTERNAL_URL	Semicolon-separated chain of webservers serving external data chunks.
CVMFS_FALLBACK_PROXY	List of HTTP proxies similar to `CVMFS_HTTP_PROXY`. The fallback proxies are added to the end of the normal proxies, and disable DIRECT connections.
CVMFS_FOLLOW_REDIRECTS	When set to *yes*, follow up to 4 HTTP redirects in requests.
CVMFS_HIDE_MAGIC_XATTRS	If set to *yes* the client will not expose CernVM-FS specific extended attributes
CVMFS_HOST_RESET_AFTER	See `CVMFS_PROXY_RESET_AFTER`.
CVMFS_HTTP_PROXY	Chain of HTTP proxy groups used by CernVM-FS. Necessary. Set to `DIRECT` if you don't use proxies.
CVMFS_IGNORE_SIGNATURE	When set to *yes*, don't verify CernVM-FS file catalog signatures.

continues on next page

Table 1 – continued from previous page

Parameter	Meaning
CVMFS_INITIAL_GENERATION	Initial inode generation. Used for testing.
CVMFS_INSTRUMENT_FUSE	When set to *true* gather performance statistics about the FUSE callbacks. The results are displayed with *cvmfs_talk internal affairs*.
CVMFS_NFS_INTERLEAVED_INODES	In NFS mode, use only inodes of the form $an + b$, specified as "b%a".
CVMFS_IPFAMILY_PREFER	Which IP protocol to prefer when connecting to proxies. Can be either 4 or 6.
CVMFS_KCACHE_TIMEOUT	Timeout in seconds for path names and file attributes in the kernel file system buffers.
CVMFS_KEYS_DIR	Directory containing *.pub files used as repository signing keys. If set, this parameter has precedence over `CVMFS_PUBLIC_KEY`.
CVMFS_LOW_SPEED_LIMIT	Minimum transfer rate in bytes/second a server or proxy must provide.
CVMFS_MAX_EXTERNAL_SERVERS	Limit the number of (geo sorted) stratum 1 servers for external data that are effectively used.
CVMFS_MAX_IPADDR_PER_PROXY	Limit the number of IP addresses a proxy names resolves into. From all registered addresses, up to the limit are randomly selected.
CVMFS_MAX_RETRIES	Maximum number of retries for a given proxy/host combination.
CVMFS_MAX_SERVERS	Limit the number of (geo sorted) stratum 1 servers that are effectively used.
CVMFS_MAX_TTL	Maximum file catalog TTL in minutes. Can overwrite the TTL stored in the catalog.
CVMFS_MEMCACHE_SIZE	Size of the CernVM-FS meta-data memory cache in Megabyte.
CVMFS_MOUNT_RW	Mount CernVM-FS as a read/write file system. Write operations will fail but this option can workaround faulty `open()` flags.
CVMFS_NFILES	Maximum number of open file descriptors that can be used by the CernVM-FS process.
CVMFS_NFS_SOURCE	If set to *yes*, act as a source for the NFS daemon (NFS export).

continues on next page

Table 1 – continued from previous page

Parameter	Meaning
CVMFS_NFS_SHARED	If set a path, used to store the NFS maps in an SQlite database, instead of the usual LevelDB storage in the cache directory.
CVMFS_PAC_URLS	Chain of URLs pointing to PAC files with HTTP proxy configuration information.
CVMFS_OOM_SCORE_ADJ	Set the Linux kernel's out-of-memory killer priority for the CernVM-FS client [-1000 - 1000].
CVMFS_PROXY_RESET_AFTER	Delay in seconds after which CernVM-FS will retry the primary proxy group in case of a fail-over to another group.
CVMFS_PROXY_SHARD	If set to *yes*, shard requests across all proxies within the current load-balancing group using consistent hashing.
CVMFS_PROXY_TEMPLATE	Overwrite the default proxy template in Geo-API calls. Only needed for debugging.
CVMFS_PUBLIC_KEY	Colon-separated list of repository signing keys.
CVMFS_QUOTA_LIMIT	Soft-limit of the cache in Megabyte.
CVMFS_RELOAD_SOCKETS	Directory of the sockets used by the CernVM-FS loader to trigger hotpatching/reloading.
CVMFS_REPOSITORIES	Comma-separated list of fully qualified repository names to include in use of client utilities such as `cvmfs_talk` and `cvmfs_config`. Does not limit which repositories may be mounted, unless `CVMFS_STRICT_MOUNT` is set to *yes*.
CVMFS_REPOSITORY_DATE	A timestamp in ISO format (e.g. `2007-03-01T13:00:00Z`). Selects the repository state as of the given date.
CVMFS_REPOSITORY_TAG	Select a named repository snapshot that should be mounted instead of `trunk`.
CVMFS_CONFIG_REPO_REQUIRED	If set to *yes*, no repository can be mounted unless the config repository is available.
CVMFS_ROOT_HASH	Hash of the root file catalog, implies `CVMFS_AUTO_UPDATE=no`.
CVMFS_SEND_INFO_HEADER	If set to *yes*, include the cvmfs path of downloaded data in HTTP headers.
CVMFS_SERVER_CACHE_MODE	Enable special cache semantics for a client used as a publisher's repository base line.
CVMFS_SERVER_URL	Semicolon-separated chain of Stratum~1 servers.

continues on next page

Table 1 – continued from previous page

Parameter	Meaning
CVMFS_SHARED_CACHE	If set to *no*, makes a repository use an exclusive cache.
CVMFS_STRICT_MOUNT	If set to *yes*, mount only repositories that are listed in `CVMFS_REPOSITORIES`.
CVMFS_SUID	If set to *yes*, enable suid magic on the mounted repository. Requires mounting as root.
CVMFS_SYSLOG_FACILITY	If set to a number between 0 and 7, uses the corresponding LOCALn facility for syslog messages.
CVMFS_SYSLOG_LEVEL	If set to 1 or 2, sets the syslog level for CernVM-FS messages to LOG_DEBUG or LOG_INFO respectively.
CVMFS_SYSTEMD_NOKILL	If set to *yes*, modify the command line to `@vmfs2 ...` in order to act as a systemd lowlevel storage manager.
CVMFS_TIMEOUT	Timeout in seconds for HTTP requests with a proxy server.
CVMFS_TIMEOUT_DIRECT	Timeout in seconds for HTTP requests without a proxy server.
CVMFS_TRACEFILE	If set, enables the tracer and trace file system calls to the given file.
CVMFS_USE_GEOAPI	Request order of Stratum 1 servers and fallback proxies via Geo-API.
CVMFS_USER	Sets the `gid` and `uid` mount options. Don't touch or overwrite.
CVMFS_USYSLOG	All messages that normally are logged to syslog are re-directed to the given file. This file can grow up to 500kB and there is one step of log rotation. Required for muCernVM.
CVMFS_WORKSPACE	Set the local directory for storing special files (defaults to the cache directory).
CVMFS_USE_SSL_SYSTEM_CA	When connecting to an HTTPS endpoints, it will load the certificates provided by the system.

Server parameters

Parameter	Meaning
CVMFS_AUFS_WARNING	Set to *false* to silence AUFS kernel deadlock warning.
CVMFS_AUTO_GC	Enables the automatic garbage collection on *publish* and *snapshot*
CVMFS_AUTO_GC_TIMESPAN	Date-threshold for automatic garbage collection (For example: *3 days ago*, *1 week ago*, ...)
CVMFS_AUTO_GC_LAPSE	Frequency of auto garbage collection, only garbage collect if last GC is before the given threshold (For example: *1 day ago*)
CVMFS_AUTO_REPAIR_MOUNTPOINT	Set to *true* to enable automatic recovery from bogus server mount states.
CVMFS_AUTO_TAG	Creates a generic revision tag for each published revision (if set to *true*).
CVMFS_AUTO_TAG_TIMESPAN	Date-threshold for automatic tags, after which auto tags get removed (For example: *4 days ago*)
CVMFS_AUTOCATALOGS	Enable/disable automatic catalog management using autocatalogs.
CVMFS_AUTOCATALOGS_MAX_WEIGHT	Maximum number of entries in an autocatalog to be considered overflowed. Default value: 100000 (see also *CVMFS_AUTOCATALOGS*)
CVMFS_AUTOCATALOGS_MIN_WEIGHT	Minimum number of entries in an autocatalog to be considered underflowed. Default value: 1000 (see also *CVMFS_AUTOCATALOGS*)
CVMFS_AVG_CHUNK_SIZE	Desired Average size of a file chunk in bytes (see also *CVMFS_USE_FILE_CHUNKING*)
CVMFS_CATALOG_ALT_PATHS	Enable/disable generation of catalog bootstrapping shortcuts during publishing. (Useful when backend directory */data* is not publicly accessible)
CVMFS_CHECK_ALL_MIN_DAYS	Minimum number of days between checking each repository with `cvmfs_server check -a` Default value: 30

continues on next page

Table 2 – continued from previous page

Parameter	Meaning
CVMFS_COMPRESSION_ALGORITHM	Compression algorithm to be used during publishing (currently either 'default' or 'none')
CVMFS_CREATOR_VERSION	The CernVM-FS version that was used to create this repository (do not change manually).
CVMFS_DONT_CHECK_OVERLAYFS_VERSION	Disable checking of OverlayFS version before usage. (see *Requirements for a new Repository*)
CVMFS_ENFORCE_LIMITS	Set to *true* to cause exceeding *LIMIT variables to be fatal to a publish instead of a warning
CVMFS_EXTENDED_GC_STATS	Set to *true* to keep track of the volume of garbage collected files (increases GC running time)
CVMFS_EXTERNAL_DATA	Set to *true* to mark repository to contain external data that is served from an external HTTP server
CVMFS_FILE_MBYTE_LIMIT	Maximum number of megabytes for a published file, default value: 1024 (see also *CVMFS_ENFORCE_LIMITS*)
CVMFS_FORCE_REMOUNT_WARNING	Enable/disable warning through `wall` and grace period before forcefully remounting a CernVM-FS repository on the release managere machine.
CVMFS_GARBAGE_COLLECTION	Enables repository garbage collection (Stratum~0 only | if set to *true*)
CVMFS_GC_DELETION_LOG	Log file path to track all garbage collected objects during sweeping for bookkeeping or debugging
CVMFS_GEO_DB_FILE	Path to externally updated location of geolite2 city database, or 'None' for no database.
CVMFS_GEO_LICENSE_KEY	A license key for downloading the geolite2 city database from maxmind.
CVMFS_GID_MAP	Path of a file for the mapping of file owner group ids.

continues on next page

Table 2 – continued from previous page

Parameter	Meaning
CVMFS_HASH_ALGORITHM	Define which secure hash algorithm should be used by CernVM-FS for CAS objects (supported are: *sha1*, *rmd160* and *shake128*)
CVMFS_IGNORE_SPECIAL_FILES	Set to *true* to skip special files (pipes, sockets, block device and character device files) during publish without aborting.
CVMFS_INCLUDE_XATTRS	Set to *true* to process extended attributes
CVMFS_MAX_CHUNK_SIZE	Maximal size of a file chunk in bytes (see also *CVMFS_USE_FILE_CHUNKING*)
CVMFS_MAXIMAL_CONCURRENT_WRITES	Maximal number of concurrently processed files during publishing.
CVMFS_MIN_CHUNK_SIZE	Minimal size of a file chunk in bytes (see also *CVMFS_USE_FILE_CHUNKING*)
CVMFS_NESTED_KCATALOG_LIMIT	Maximum thousands of files allowed in nested catalogs, default 500 (see also *CVMFS_ROOT_KCATALOG_LIMIT* and *CVMFS_ENFORCE_LIMITS*)
CVMFS_NUM_UPLOAD_TASKS	Number of threads used to commit data to storage during publication. Currently only used by the local backend.
CVMFS_NUM_WORKERS	Maximal number of concurrently downloaded files during a Stratum1 pull operation (Stratum~1 only).
CVMFS_PUBLIC_KEY	Colon-separated path to the public key file(s) or directory(ies) of the repository to be replicated. (Stratum 1 only).
CVMFS_PRINT_STATISTICS	Set to *true* to show publisher statistics on the console
CVMFS_REPLICA_ACTIVE	Stratum1-only: Set to *no* to skip this repository when executing `cvmfs_server snapshot -a`
CVMFS_REPOSITORY_NAME	The fully qualified name of the specific repository.
CVMFS_REPOSITORY_TYPE	Defines if the repository is a master copy (*stratum0*) or a replica (*stratum1*).
CVMFS_REPOSITORY_TTL	The frequency in seconds of client lookups for changes in the repository. Defaults to 4 minutes.

continues on next page

Table 2 – continued from previous page

Parameter	Meaning
CVMFS_ROOT_KCATALOG_LIMIT	Maximum thousands of files allowed in root catalogs, default 200 (see also *CVMFS_NESTED_KCATALOG_LIMIT* and *CVMFS_ENFORCE_LIMITS*
CVMFS_SNAPSHOT_GROUP	Group name for subset of repositories used with `cvmfs_server snapshot -a -g`. Added with `cvmfs_server add-replica -g`.
CVMFS_SPOOL_DIR	Location of the upstream spooler scratch directories; the read-only CernVM-FS moint point and copy-on-write storage reside here.
CVMFS_STATISTICS_DB	Set a custom path for the publisher statistics database
CVMFS_STATS_DB_DAYS_TO_KEEP	Sets the pruning interval for the publisher statistics database (365 by default)
CVMFS_STRATUM0	URL of the master copy (*stratum0*) of this specific repository.
CVMFS_STRATUM1	URL of the Stratum1 HTTP server for this specific repository.
CVMFS_SYNCFS_LEVEL	Controls how often `sync` will by called by `cvmfs_server` operations. Possible levels are 'none', 'default', 'cautious'.
CVMFS_UID_MAP	Path of a file for the mapping of file owner user ids.
CVMFS_UNION_DIR	Mount point of the union file system for copy-on-write semantics of CernVM-FS. Here, changes to the repository are performed (see *CernVM-FS Repository Creation and Updating*).
CVMFS_UNION_FS_TYPE	Defines the union file system to be used for the repository. (currently *aufs* and *overlayfs* are fully supported)
CVMFS_UPLOAD_STATS_DB	Publish repository statistics data file to the Stratum 0 /stats location
CVMFS_UPLOAD_STATS_PLOTS	Publish repository statistics plots and webpage to the Stratum 0 /stats location (requires ROOT)

continues on next page

Table 2 – continued from previous page

Parameter	Meaning
CVMFS_UPSTREAM_STORAGE	Upstream spooler description defining the basic upstream storage type and configuration (see below).
CVMFS_USE_FILE_CHUNKING	Allows backend to split big files into small chunks (*true* | *false*)
CVMFS_USER	The user name that owns and manipulates the files inside the repository.
CVMFS_VIRTUAL_DIR	Set to *true* to enable the hidden, virtual `.cvmfs/snapshots` directory containing entry points to all named tags.
CVMFS_VOMS_AUTHZ	Membership requirement (e.g. VOMS authentication) to be added into the file catalogs
CVMFS_STATISTICS_DB	SQLite file path to store the statistics. Default is `/var/spool/cvmfs/<REPO_NAME>/stats.db` .
CVMFS_PRINT_STATISTICS	Set to *true* to enable statistics printing to the standard output.
X509_CERT_BUNDLE	Bundle file with CA certificates for HTTPS connections (see *Large-Scale Data CernVM-FS*)
X509_CERT_DIR	Directory file with CA certificates for HTTPS connections, defaults to /etc/grid-security/certificates (see *Large-Scale Data CernVM-FS*)

Deprecated parameters

Will be removed in future versions.

Parameter	Meaning
CVMFS_GENERATE_LEGACY_BULK_CHUNKS	Deprecated, set to *true* to enable generation of whole-file objects for large files.
CVMFS_IGNORE_XDIR_HARDLINKS	Deprecated, defaults to *true* hardlinks are found. Instead automatically break the hardlinks across directories.

Format of CVMFS_UPSTREAM_STORAGE

The format of the `CVMFS_UPSTREAM_STORAGE` parameter depends on the storage backend. Note that this parameter is initialized by `cvmfs_server mkfs` resp. `cvmfs_server add-replica`. The internals of the parameter are only relevant if the configuration is maintained by a configuration management system.

For the local storage backend, the parameter specifies the storage directory (to be served by Apache) and a temporary directory in the form `local,<path for temporary files>,<path to storage>`, e.g.

```
CVMFS_UPSTREAM_STORAGE=local,/srv/cvmfs/sw.cvmfs.io/data/txn,/srv/cvmfs/sw.cvmfs.io
```

For the S3 backend, the parameter specifies a temporary directory and the location of the S3 config file in the form `s3,<path for temporary files>,<repository entry URL on the S3 server>@<S3 config file>`, e.g.

```
CVMFS_UPSTREAM_STORAGE=S3,/var/spool/cvmfs/sw.cvmfs.io/tmp,cvmfs/sw.cvmfs.io@/etc/
↪cvmfs/s3.conf
```

The gateway backend can only be used on a remote publisher (not on a stratum 1). The parameter specifies a temporary directory and the endpoint of the gateway service, e.g.

```
CVMFS_UPSTREAM_STORAGE=gw,/var/spool/cvmfs/sw.cvmfs.io/tmp,http://cvmfs-gw.cvmfs.
↪io:4929/api/v1
```

Tiered Cache Parameters

The following parameters are used to configure a tiered cache manager instance.

Parameter	Meaning
CVMFS_CACHE_$name_UPPER	Name of the upper layer cache instance
CVMFS_CACHE_$name_LOWER	Name of the lower layer cache instance
CVMFS_CACHE_LOWER_READONLY	Set to *true* to avoid populating the lower layer

External Cache Plugin Parameters

The following parameters are used to configure an external cache plugin as a cache manager instance.

Parameter	Meaning
CVMFS_CACHE_$name_CMDLINE	If the client should start the plugin, the executable and command line parameters of the plugin, separated by comma.
CVMFS_CACHE_$name_LOCATOR	The address of the socket used for communication with the plugin.

In-memory Cache Plugin Parameters

The following parameters are interpreted from the configuration file provided to the in-memory cache plugin (see Section *Example*).

Parameter	Meaning
CVMFS_CACHE_PLUGIN_DEBUGLOG	If set, run CernVM-FS in debug mode and write a verbose log the the specified file.
CVMFS_CACHE_PLUGIN_LOCATOR	The address of the socket used for client communication
CVMFS_CACHE_PLUGIN_SIZE	The amount of RAM in megabyte used by the plugin for caching.

2.13.3 CernVM-FS Server Infrastructure

This section provides technical details on the CernVM-FS server setup including the infrastructure necessary for an individual repository. It is highly recommended to first consult "*Notable CernVM-FS Server Locations and Files*" for a more general overview of the involved directory structure.

Prerequisites

A CernVM-FS server installation depends on the following environment setup and tools to be in place:

- Appropriate kernel version. You must have ONE of the following:
  - kernel 4.2.x or later.
  - RHEL7.3 kernel (for OverlayFS)
- Backend storage location available through HTTP
- Backend storage accessible at `/srv/cvmfs/...` (unless stored on S3)
- **cvmfs** and **cvmfs-server** packages installed

Local Backend Storage Infrastructure

CernVM-FS stores the entire repository content (file content and meta-data catalogs) into a content addressable storage (CAS). This storage can either be a file system at `/srv/cvmfs` or an S3 compatible object storage system (see "*S3 Compatible Storage Systems*" for details). In the former case the contents of `/srv/cvmfs` are as follows:

File Path	Description
`/srv/cvmfs`	**Central repository storage location** Can be mounted or symlinked to another location *before* creating the first repository.
`/srv/cvmfs/<fqrn>`	**Storage location of a specific repository** Can be symlinked to another location *before* creating the repository `<fqrn>`. This location needs to be both writable by the repository owner and accessible through an HTTP server.
`/srv/cvmfs/<fqrn>/.cvmfspublished`	**Manifest file of the repository** The manifest provides the entry point into the repository. It is the only file that needs to be signed by the repository's private key.
`/srv/cvmfs/<fqrn>/.cvmfswhitelist`	**List of trusted repository certificates** Contains a list of certificate fingerprints that should be allowed to sign a repository manifest (see .cvmfspublished). The whitelist needs to be signed by a globally trusted private key.
`/srv/cvmfs/<fqrn>/data`	**CAS location of the repository** Data storage of the repository. Contains catalogs, files, file chunks, certificates and history databases in a content addressable file format. This directory and all its contents need to be writable by the repository owner.
`/srv/cvmfs/<fqrn>/data/00..ff`	**Second CAS level directories** Splits the flat CAS namespace into multiple directories. First two digits of the file content hash defines the directory the remainder is used as file name inside the corresponding directory.
`/srv/cvmfs/<fqrn>/data/txn`	**CAS transaction directory** Stores partial files during creation. Once writing has completed, the file is committed into the CAS using an atomic rename operation.

Server Spool Area of a Repository (Stratum0)

The spool area of a repository contains transaction infrastructure and scratch area of a Stratum0 or specifically a release manager machine installation. It is always located inside `/var/spool/cvmfs` with directories for individual repositories. Note that the data volume of the spool area can grow very large for massive repository updates since it contains the writable union file system branch and a CernVM-FS client cache directory.

File Path	Description
/var/spool/cvmfs	**CernVM-FS server spool area** Contains administrative and scratch space for CernVM-FS repositories. This directory should only contain directories corresponding to individual CernVM-FS repositories.
/var/spool/cvmfs/<fqrn>	**Individual repository spool area** Contains the spool area of an individual repository and might temporarily contain large data volumes during massive repository updates. This location can be mounted or symlinked to other locations. Furthermore it must be writable by the repository owner.
/var/spool/cvmfs/<fqrn>/cache	**CernVM-FS client cache directory** Contains the cache of the CernVM-FS client mounting the r/o branch (i.e. `/var/spool/cvmfs/<fqrn>/rdonly`) of the union file system mount point located at `/cvmfs/<fqrn>`. The content of this directory is fully managed by the CernVM-FS client and hence must be configured as a CernVM-FS cache and writable for the repository owner.
/var/spool/cvmfs/<fqrn>/rdonly	**CernVM-FS client mount point** Serves as the mount point of the CernVM-FS client exposing the latest published state of the CernVM-FS repository. It needs to be owned by the repository owner and should be empty if CernVM-FS is not mounted to it.
/var/spool/cvmfs/<fqrn>/scratch	**Writable union file system scratch area** All file system changes applied to `/cvmfs/<fqrn>` during a transaction will be stored in this directory. Hence, it potentially needs to accommodate a large data volume during massive repository updates. Furthermore it needs to be writable by the repository owner.
/var/spool/cvmfs/<fqrn>/tmp	**Temporary scratch location** Some CernVM-FS server operations like publishing store temporary data files here, hence it needs to be writable by the repository owner. If the repository is idle this directory should be empty.
/var/spool/cvmfs/<fqrn>/client.config	**CernVM-FS client configuration** This contains client configuration variables for the CernVM-FS client mounted to `/var/spool/cvmfs/<fqrn>/rdonly`. Most notibly it needs to contain `CVMFS_ROOT_HASH` configured to the latest revision published in the corresponding repository. This file needs to be writable by the repository owner.

Repository Configuration Directory

The authoritative configuration of a CernVM-FS repository is located in `/etc/cvmfs/repositories.d` and should only be writable by the administrator. Furthermore the repository's keychain is located in `/etc/cvmfs/keys` and follows the naming convention `<fqrn>.crt` for the certificate, `<fqrn>.key` for the repository's private key and `<fqrn>.pub` for the public key. All of those files can be symlinked somewhere else if necessary.

File Path	Description
/etc/cvmfs/repositories.d	**CernVM-FS server config directory** This contains the configuration directories for individual CernVM-FS repositories. Note that this path is shortened using `/.../repos.d/` in the rest of this table.
/.../repos.d/<fqrn>	**Config directory for specific repo** This contains the configuration files for one specific CernVM-FS repository server.
/.../repos.d/<fqrn>/server.conf	**Server configuration file** Authoriative configuration file for the CernVM-FS server tools. This file should only contain *valid server configuration variables* as it controls the behaviour of the CernVM-FS server operations like publishing, pulling and so forth.
/.../repos.d/<fqrn>/client.conf	**Client configuration file** Authoriative configuration file for the CernVM-FS client used to mount the latest revision of a Stratum 0 release manager machine. This file should only contain *valid client configuration variables*. This file must not exist for Stratum 1 repositories.
/.../repos.d/<fqrn>/replica.conf	**Replication configuration file** Contains configuration variables for Stratum 1 specific repositories. This file must not exist for Stratum 0 repositories.

Environment Setup

Apart from file and directory locations a CernVM-FS server installation depends on a few environment configurations. Most notably the possibility to access the backend storage through HTTP and to allow for mounting of both the CernVM-FS client at `/var/spool/cvmfs/<fqrn>/rdonly` and a union file system on `/cvmfs/<fqrn>`.

Granting HTTP access can happen in various ways and depends on the chosen backend storage type. For an S3 hosted backend storage, the CernVM-FS client can usually be directly pointed to the S3 bucket used for storage (see "*S3 Compatible Storage Systems*" for details). In case of a local file system backend any web server can be used for this purpose. By default CernVM-FS assumes Apache and uses that automatically.

Internally the CernVM-FS server uses a SUID binary (i.e. `cvmfs_suid_helper`) to manipulate its mount points. This is necessary since transactional CernVM-FS commands must be accessible to the repository owner that is usually different from root. Both the mount directives for `/var/spool/cvmfs/<fqrn>/rdonly` and `/cvmfs/<fqrn>` must be placed into `/etc/fstab` for this reason. By default CernVM-FS uses the following entries for these mount points:

```
cvmfs2#<fqrn> /var/spool/cvmfs/<fqrn>/rdonly fuse \
allow_other,config=/etc/cvmfs/repositories.d/<fqrn>/client.conf: \
/var/spool/cvmfs/<fqrn>/client.local,cvmfs_suid 0 0

aufs_<fqrn> /cvmfs/<fqrn> aufs br=/var/spool/cvmfs/<fqrn>/scratch=rw: \
/var/spool/cvmfs/<fqrn>/rdonly=rr,udba=none,ro 0 0
```

2.13.4 Available Packages

The CernVM-FS software is available in form of several packages:

cvmfs-release
Adds the CernVM-FS yum/apt repository.

cvmfs-config-default
Contains a configuration and public keys suitable for nodes in the Worldwide LHC Computing Grid. Provides access to repositories in the cern.ch, egi.eu, and opensciencegrid.org domains.

cvmfs-config-none
Empty package to satisfy the cvmfs-config requirement of the cvmfs package without actually installing any configuration.

cvmfs
Contains the Fuse module and additional client tools. It has dependencies to at least one of the cvmfs-config-··· packages.

cvmfs-fuse3
Contains the additional client libraries necessary to mount with the libfuse3 system libraries.

cvmfs-devel
Contains the `libcvmfs.a` static library and the `libcvmfs.h` header file for use of CernVM-FS with Parrot [Thain05] as well as the `libcvmfs_cache.a` static library and `libcvmfs_cache.h` header in order to develop cache plugins.

cvmfs-auto-setup
Only available through yum. This is a wrapper for `cvmfs_config setup`. This is supposed to provide automatic configuration for the ATLAS Tier3s. Depends on cvmfs.

cvmfs-server
Contains the CernVM-FS server tool kit for maintaining publishers and Stratum 1 servers.

cvmfs-gateway
The publishing gateway services are installed on a node with access to the authoritative storage.

cvmfs-ducc
Daemon that unpacks container images into a repository. Supposed to run on a publisher node.

cvmfs-notify
Websockets frontend for used for repository update notifications. Supposed to be co-located with a RabbitMQ service.

kernel-...-.aufs21
Scientific Linux 6 kernel with aufs. Required for SL6 based Stratum 0 servers.

cvmfs-shrinkwrap
Stand-alone utility to export file system trees into containers for HPC use cases.

cvmfs-unittests
Contains the `cvmfs_unittests` binary. Only required for testing.

2.13.5 Known Issues

Publisher nodes with AUFS and XFS

If the /tmp file system is on xfs, the publisher node cannot be used with AUFS. On such systems, adding the mount option `xino=/dev/shm/aufs.xino` can be a workaround. In general, new repositories should use OverlayFS if available.

2.13.6 Contact Information

For support requests and bug reports, please submit a GitHub issue in our issue tracker.

Together with bug reports, please attach a "bugreport tarball", which is created with `sudo cvmfs_config bugreport`.

Discourse Forum

For announcements, discussions, and support please join us in the CernVM Forum.

2.13.7 References

CHAPTER

THREE

CONTACT AND AUTHORS

Visit our website on cernvm.cern.ch.

Authors of this documentation:

- Jakob Blomer
- Brian Bockelman
- Daniel-Florin Dosaru
- Dave Dykstra
- Nikola Hardi
- Nick Hazekamp
- René Meusel
- Simone Mosciatti
- Radu Popescu
- Laura Promberger

BIBLIOGRAPHY

[Blumenfeld08] Blumenfeld, B. et al. 2008. CMS conditions data access using FroNTier. *Journal of Physics: Conference Series*. 119, (2008).

[Callaghan95] Callaghan, B. et al. 1995. *NFS Version 3 Protocol Specification*. Technical Report #1813. Internet Engineering Task Force.

[Gauthier99] Gauthier, P. et al. 1999. *Web proxy auto-discovery protocol*. IETF Secretariat.

[Guerrero99] Guerrero, D. 1999. Caching the web, part 2. *Linux Journal*. 58 (Feburary 1999).

[Panagiotou06] Panagiotou, K. and Souza, A. 2006. On adequate performance measures for paging. *Annual ACM Symposium on Theory Of Computing*. 38, (2006), 487-496.

[Schubert08] Schubert, M. et al. 2008. *Nagios 3 enterprise network monitoring*. Syngress.

[Shepler03] Shepler, S. et al. 2003. *Network File System (NFS) version 4 Protocol*. Technical Report #3530. Internet Engineering Task Force.

[Jones01] 3rd, D.E. and Jones, P. 2001. *US Secure Hash Algorithm 1 (SHA1)*. Technical Report #3174. Internet Engineering Task Force.

[Dobbertin96] Dobbertin, H. et al. 1996. RIPEMD-160: A strengthened version of RIPEMD. Springer. 71-82.

[Bertoni09] Bertoni, G., Daemen, J., Peeters, M. and Van Assche, G., 2009. Keccak sponge function family main document. Submission to NIST (Round 2), 3, p.30.

[Rivest92] Rivest, R. 1992. *The MD5 Message-Digest Algorithm*. Technical Report #1321. Internet Engineering Task Force.

[Turner11] Turner, S. and Chen, L. 2011. *Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms*. Technical Report #6151. Internet Engineering Task Force.

[Deutsch96] Deutsch, P. and Gailly, J.-L. 1996. *ZLIB Compressed Data Format Specification version 3.3*. Technical Report #1950. Internet Engineering Task Force.

[Allen10] Allen, G. and Owens, M. 2010. *The definitive guide to SQLite*. Apress.

[Wright04] Wright, C.P. et al. 2004. *Versatility and unix semantics in a fan-out unification file system*. Technical Report #FSL-04-01b. Stony Brook University.

[BernersLee96] Berners-Lee, T. et al. 1996. *Hypertext Transfer Protocol - HTTP/1.0*. Technical Report #1945. Internet Engineering Task Force.

[Fielding99] Fielding, R. et al. 1999. *Hypertext Transfer Protocol - HTTP/1.1*. Technical Report #2616. Internet Engineering Task Force.

[Compostella10] Compostella, G. et al. 2010. CDF software distribution on the Grid using Parrot. *Journal of Physics: Conference Series*. 219, (2010).

[Thain05] Thain, D. and Livny, M. 2005. Parrot: an application environment for data-intensive computing. *Scalable Computing: Practice and Experience*. 6, 3 (18 2005), 9.

[Suzaki06] Suzaki, K. et al. 2006. HTTP-FUSE Xenoppix. *Proc. of the 2006 linux symposium* (2006), 379-392.

[Freedman03] Freedman, M.J. and Mazières, D. 2003. Sloppy hashing and self-organizing clusters. M.F. Kaashoek and I. Stoica, eds. Springer. 45-55.

[Nygren10] Nygren, E. et al. 2010. The Akamai network: A platform for high-performance internet applications. *ACM SIGOPS Operating Systems Review*. 44, 3 (2010), 2-19.

[Tolia03] Tolia, N. et al. 2003. Opportunistic use of content addressable storage for distributed file systems. *Proc. of the uSENIX annual technical conference* (2003).

[Dykstra10] Dykstra, D. and Lueking, L. 2010. Greatly improved cache update times for conditions data with frontier/Squid. *Journal of Physics: Conference Series*. 219, (2010).