SaunaFS Documentation

Version: 4.0.8

SaunaFS

Table of contents:

Version: 4.0.8

SaunaFS documentation overview

Introduction

About this document

This SaunaFS documentation, as of 29th December 2023, is in an early draft stage and primarily builds upon pre-existing documentation. This may lead to certain details being absent, incorrect, or outdated. In cases of confusion or questions, reaching out to the development team is advised.

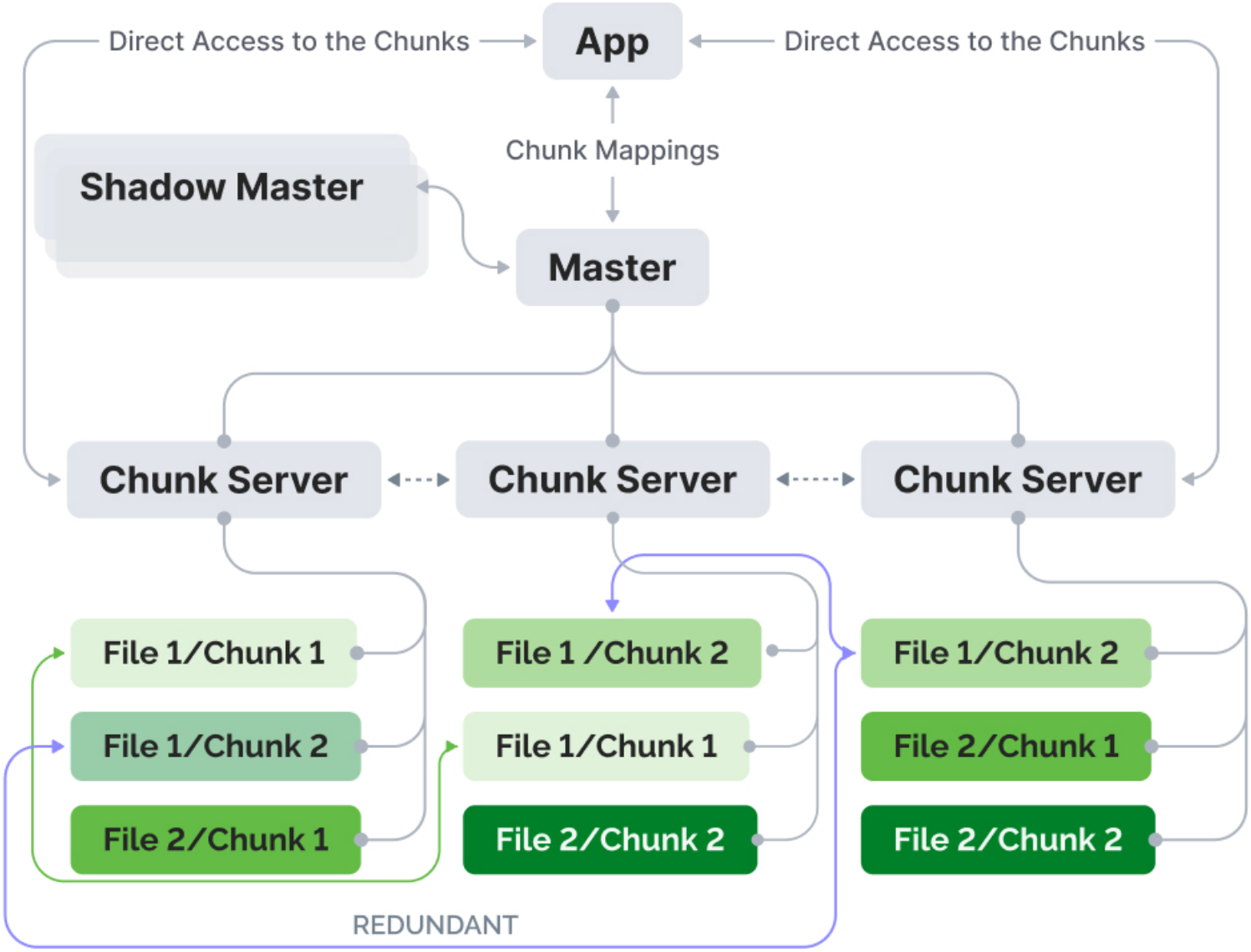
Despite its preliminary status, this document aims to provide sufficient information for the initial setup and operation of SaunaFS. It includes essential details and a quick start guide to facilitate basic configurations.

Feedback is highly valued at this stage. It not only aids in enhancing this document for future reference but also assists others in their SaunaFS setup process.

Please note that SaunaFS software, Windows Client software, and this documentation are all licensed separately under different licensing formats. For more information about the licensing terms for each component, please see the Licensing section of this documentation.

We appreciate your choice to use SaunaFS.

Architectural overview of SaunaFS



Important notes

Hardware recommendations

There are no fixed requirements for hardware, although, for better results it is recommended to have:

- 10 or 25 GbE networking
- Bonding (e.g., MC-LAG) across the switches for redundant setup
- Nodes with unified hardware configurations

Sample hardware configuration for node:

- 1x Intel® Xeon® Silver or higher CPU (or AMD equivalent)
- 4x 16GB DDR4 ECC Registered DIMM

- 2x 240GB Enterprise SSD
- 10x Enterprise HDD
- 1x Network Interface Card 25GbE Dual-Port

If unified nodes cannot be provided, at least Master/Shadow nodes should have hardware configuration like sample setup.

Alternatively, you can use the hardware sizer application to calculate hardware requirements for your specific needs: https://diaway.com/saunafs#calc

The suggestion for unified hardware configuration is drawn from the future updates that will introduce the setup with multiple master servers. It is our current effort to introduce distributed metadata enabled architecture to circumvent the limitations imposed by the RAM capacity of a single node within a namespace and also to introduce parallel access to metadata.

The following table will give the estimated amount of RAM occupied by the metadata correlated to the number of files in SaunaFS.

Number of files	all data structures overhead
1	500 B
1000	500 KB
1 000 000	500 MB
100 000 000	50 GB
1 000 000 000	500 GB

Manual pages

This document does not elaborate on every command or configuration option. For comprehensive information, manual (man) files are provided in the Debian packages for both commands and configuration files.

To view a man file for a command, e.g., saunafs-admin:

- man saunafs-admin

To view a man file for a configuration, e.g., sfsmaster.cfg:

- man sfsmaster.cfg

Systemd services

The Debian packages include systemd services for initiating various SaunaFS services. This document assumes the use of these services in its examples.

For systems without systemd, or those choosing not to use it, examining the service files for custom setup or direct command usage is recommended.

Quick overview of SaunaFS

SaunaFS is a distributed POSIX file system inspired by the Google File System, comprising Metadata Servers (Master, Shadows, Metaloggers), Data Servers (Chunkservers), and Clients (supporting multiple operating systems and NFS). It employs a chunk-based storage architecture, segmenting files into 64 MiB chunks subdivided into 64 KiB blocks, each with 4 bytes of CRC (Cyclic Redundancy Check) for data integrity.

The write process in SaunaFS involves clients requesting the Master server for suitable Chunkservers for file chunk storage. Data is transferred directly to Chunkservers in 64 KiB blocks, with CRC verification by Chunkservers and subsequent metadata updates. SaunaFS utilizes Reed-Solomon erasure coding for redundancy, enhancing data integrity and availability. For instance, a 65 MiB file is segmented into four 16 MiB data parts and two parity parts.

The system also prioritizes data resiliency through data scrubbing and CRC32 checksum verification. Additional features include instant copy-on-write snapshots, efficient metadata logging, and hardware integration without downtime.

Version: 4.0.8

Getting started

Get the Debian packages from the repository. See the installation guide for more details, then return here.

This quick-start will make some assumptions:

1. You want to quickly setup SaunaFS to test it.
2. You have a single machine to set it up.

The minimal setup required is a master server, a chunkserver and a client. We will set it up on a single machine. However note that this setup is **NOT SUPPORTED**, this is purely for testing it quickly with little effort. In actuality, you probably want a different setup, with master, client and chunkserver(s) on their own dedicated machines. Still, this guide should help you get an idea how to get started with your own setup. The admininstration guide and man pages will have more details.

Note that currently SaunaFS servers don't work on localhost (this might be changed in the future), so you'll need to do some network setup if you are planning on running both master and chunkserver on the same network address. This will be covered in the guide, however it can also useful for developers for setting up local testing environments.

Version: 4.0.8

Creating the required directories

For the purposes of this setup, we will set up everything in the localhost as a demonstration. We need three directories: Two to store data, and one to access it. In a real-world scenario, the two directories storing data would be dedicated drives, but for now we will set them up as simple directories.

```
mkdir /mnt/hd1 # Storage 1
mkdir /mnt/hd2 # Storage 2
mkdir /mnt/client # Client
```

You need to set the correct user and permissions for the storage directories. The user required can be changed in the configuration files, but the default is saunafs

```
chown saunafs:saunafs /mnt/hd* # NB: This assumes no other files that start with
hd exist in the directory
chmod 600 /mnt/hd*
```

You should also set the ownership of the client folder to the user you are using, so you can easily test if it’s working later:

```
chown $USER:$USER /mnt/client/
```

Version: 4.0.8

Setting up on localhost (Optional)

Chunkservers and master will not work with each other if running on the same IP address, so we need to make sure that they can (You can skip this if you are not running chunkservers on the same IP address as master.)

First, add an identifier to for master in the /etc/hosts file:

```
10.33.33.33 sfsmaster
```

It can be any IP address, but preferably it should be something private and unused.

Next, add the new IP address to your loopback device

```
sudo ip a a 10.33.33.33 dev lo
```

Note that a loopback will significantly impact performance. However, you shouldn't use localhost for production anyway. This will apply temporarily until the next boot.

Version: 4.0.8

Configuring and running master and chunkserver

Copy the configuration files

```
sudo cp /usr/share/doc/saunafs-master/examples/sfsmaster.cfg /etc/saunafs/
sudo cp /usr/share/doc/saunafs-chunkserver/examples/sfschunkserver.cfg
/etc/saunafs/
sudo cp /usr/share/doc/saunafs-chunkserver/examples/sfshdd.cfg /etc/saunafs/
```

In the new sfshdd.cfg, uncomment the lines with /mnt/hd1 and /mnt/hd2 This should be enough to work, but if you named your master host something other than sfsmaster (or if you are running master on another IP), you also need to uncomment this line in /etc/saunafs/sfschunkserver.cfg and change the value to something else:

`MASTER_HOST = <master ip address/name>`

Start both master server and chunkserver

For the first run of master, you will need an empty metadata file for master, otherwise it will not start.

```
cp -vib /var/lib/saunafs/metadata.sfs.empty /var/lib/saunafs/metadata.sfs
```

> **METADATA**
>
> Please note that usage of `cp -biv` is for avoiding potential existing metadata overwriting. There is a risk that one will execute this comand (eg. from history) on existing running instalation of SaunaFS.

```
sudo systemctl start saunafs-master
sudo systemctl start saunafs-chunkserver
```

They should start successfully.

Version: 4.0.8

Mounting the client

Finally, you need to mount the client. You can use sfsmount3 (sfsmount is deprecated and will be removed in a future release) to start using SaunaFS:

```
sudo sfsmount3 /mnt/client/
```

If you are not using the sfsmaster hostname IP address, you can specify the master with this:

```
sudo sfsmount3 <master IP> /mnt/client/
```

```
sudo chown $USER:$USER /mnt/client/
```

Version: 4.0.8

Wrapping up

If all goes well, you should be able to read and write to the /mnt/client/ (and thus to SaunaFS) with ease. If it's taking a long time to write and/or read the directory, then the chunkserver is likely not cooperating with the master. Check the logs for both master and chunkserver (using journalctl).

This was a short but simple setup of SaunaFS. While simple, this is not very redundant nor is it the setup you are looking for. For advanced setups, look at the Admin Guide next.

If you need help or advice, come, and join us at the SaunaFS Slack!

Version: 4.0.8

Windows client

The current SaunaFS Windows Client (1.0.0) consists of the Command Line Interface (CLI) client. Install our client from the provided exe file. The CLI client is in “C:\Program Files\SaunaFS\saunafscli.exe” on default installation path.

The CLI client works as an executable that receives commands passed by command line, like from CMD or PowerShell. After moving the console to the installation folder, you may start using the app.

Windows Client is licensed separately from the other parts of the SaunaFS software. For more information about the licensing terms for Windows Client, please see the Licensing section of this documentation.

General options

Option	Description
-H HOST | -o sfsmaster=HOST	define sfsmaster location (default: sfsmaster). It defines the IP of the master of the SaunaFS system, i.e. it defines to which system the client is connecting to.
-P PORT | -o sfsport=PORT	define sfsmaster port number (default: 9421).
-D LETTER | -o sfsdriveletter=LETTER	mount filesystem as drive with letter LETTER (default: Z).

For instance, the command:

```
saunafscli –H 192.168.56.1 –P 9521 –D S
```

should display a new local drive mounted in the letter S. The IP 192.168.56.1 must host a SaunaFS master server listening for incoming client communications on port 9521.

Extended general options

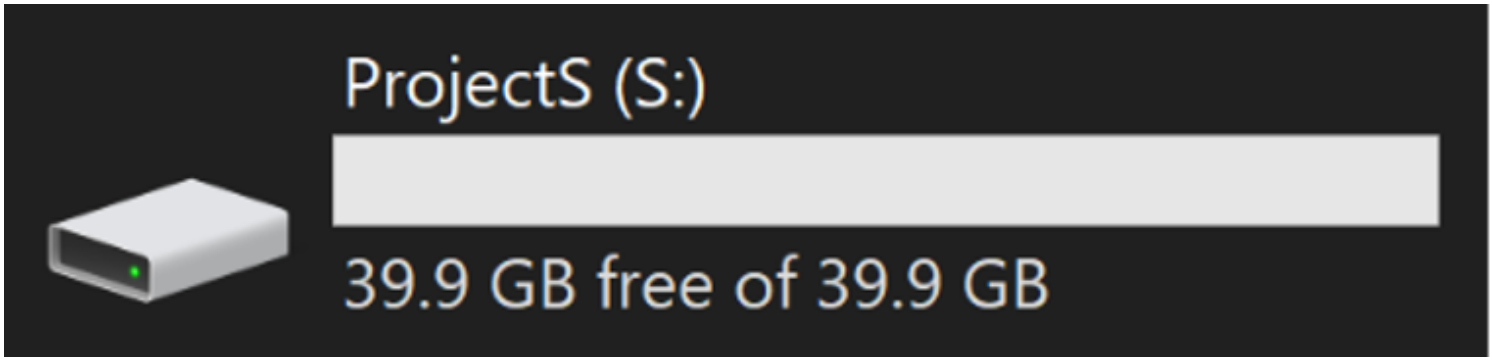
Option	Description
-S PATH | -o sfssubfolder=PATH	define subfolder to mount as root (default: /). The storage could have a number of folders, this option allows the user to display inside a mounted drive only the content of one of those specific folders.
-o sfsvolumelabel=NAME	mounted filesystem will appear with label NAME.
-o sfsuncpath=PATH	mount filesystem as a network drive with UNC path PATH. PATH format is SERVER/DRIVE.
-o sfsmountingsubfolder=PATH	mount filesystem at a specified PATH folder. When this option is set the drive letter option must not be set.

The pairs of options sfsvolumelabel-sfsuncpath and sfsuncpath-sfsmountingsubfolder and are mutually exclusive. Please only use one of the two while writing mount command because some options will be ignored.

Example 1

```
saunafscli –H 192.168.56.1 –P 9521 –D S –S /data/ProjectS –o
sfsvolumelabel=ProjectS
```

should display the following regular drive:

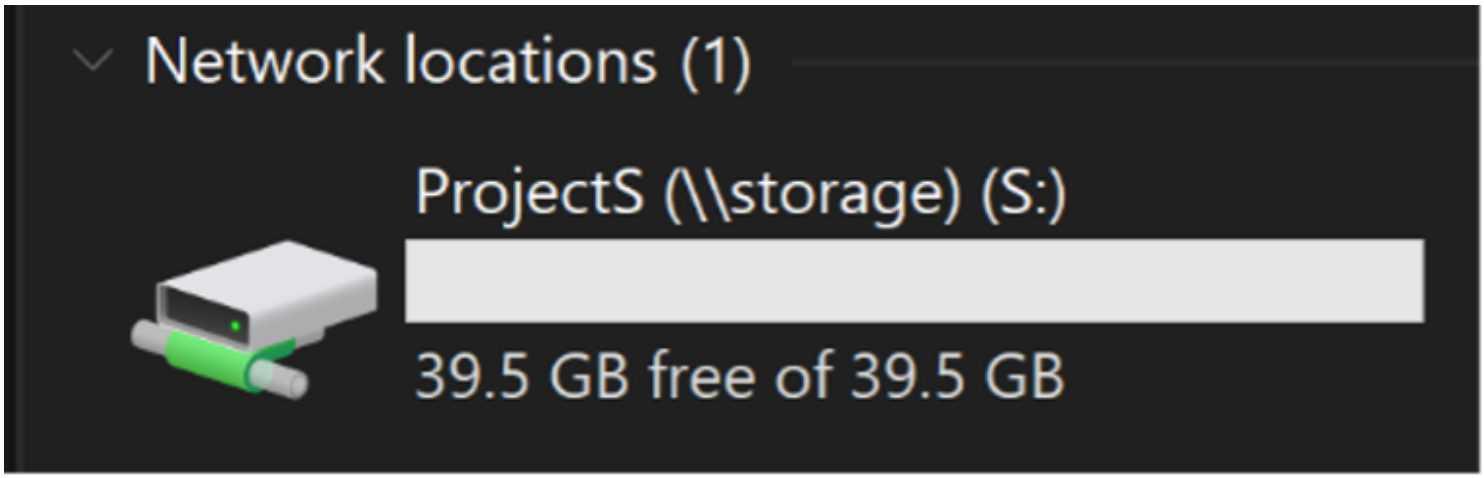


i.e. a regular drive labeled ProjectS and displaying only the contents of the folder “/data/ProjectS” of the storage.

Example 2

```
saunafscli -H 192.168.56.1 -P 9521 -D S -o sfsuncpath=storage/ProjectS
```

should display the following drive:



i.e. a network drive on the desired UNC path.

Example 3

```
saunafscli -H 192.168.56.1 -P 9521 -o sfsmountingsubfolder=C:\\mnt -o
sfsvolumelabel=ProjectS
```

should create the “ProjectS” folder in the “C:\mnt” folder displaying the content of the storage.

Security/permissions related options

Option	Description
-o sfspassword=PASSWORD	authenticate to sfsmaster with password. (Raw password option)
-o sfsmd5pass=MD5	authenticate to sfsmaster using directly given md5 (only if sfspassword is not defined).(Password after MD5 processing).
-p | --password | -o askpassword	similar to '-o sfspassword=PASSWORD’ but show prompt and ask user for password. Password won’t be displayed while typing it.

Option	Description
-o sfsdonotrememberpassword	do not remember password in memory - more secure, but when session is lost then new session is created without password.
-o sfsuid=UID	set user id of the mounting user as UID (default behavior uses WinFSP local mapping of Windows SID).
-o sfsgid=GID	set group id of the mounting user as GID (default behavior uses WinFSP "No group" mapping).
-o sfsumaskfile=MASK	set permissions for newly created file (octal) (default: 002). This option represents the unset file permission bits. So, for the default setting the resulting file permission mask is 775.
-o sfsumaskdir=MASK	set permissions for newly created folder (octal) (default: 002). This option is very much like sfsumaskfile option one.

Passwords can be set on the master side to provide layers of security to some specific paths of the storage, the first four options of this group provide the users the tools to fulfill those security requirements.

Current Windows Client does not support Active Directory and cannot provide a true translation from the Windows context of users (client side) to the Linux context of users (master side). The sfsuid and sfsgid options provide manual translation. Thus, the command

```
saunafscli -H 192.168.56.1 -P 9521 -D S -o sfsuid=999 -o sfsgid=999 -o
sfsumaskfile=111 -o sfsumaskdir=023
```

will mount the regular drive and all actions in the storage will be performed with the security clearance of the user with uid 999 and gid 999, the default permission mask of the new created files will be 666, which means full permissions to every user except execution ones and the default permission mask of the new created directories will be 754, which means full permissions to creator of the directory, no write permission for other users sharing group ID and no execution and write permissions for other users. Default behavior of the sfsuid and sfsgid options uses WinFSP local mappings of the Windows SID. It means that if this option is not set some weird values will be seen. For instance, local users are mapped to

UIDs higher than 3*2^16, Administrator user gets mapped to 544, System to 18 and the "No Group" is represented with GID 197121.

Readahead related options

Option	Description
-o cacheexpirationtime=MSEC	set timeout for read cache entries to be considered valid in milliseconds (0 disables cache) (default: 1000).
-o readaheadmaxwindowsize=KB	set max value of readahead window per single descriptor in kibibytes (default: 65536).
-o readworkers=N	define number of read workers (default: 30).
-o maxreadaheadrequests=N	define number of readahead requests per inode (default: 5).

The previous options configure the readahead mechanism of the client. This mechanism improves the client performance on read operations, especially on the sequential read operations. Default values should work for most cases, but it can be tuned.

The cacheexpirationtime option could be increased if the reliability of a read of the files is high, i.e., the files are not supposed to change shortly after a read. The read cache is kept in memory, so keep track of the used RAM by the client and reduce the cacheexpirationtime if necessary. The readworkers option configures the number of workers (threads) the client creates to read files, so depending on the number of files read at the same time this number can go up or down. The maxreadaheadrequests option configures how far the readahead mechanism should go when reading a file, so depending on the expected kind of reads the client is supposed to do this number can go up (sequential reads) or down (sparse reads). Please consider leaving default parameters if desired performance is achieved or there is little data about the type of reads the client is going to do.

Persistent mount options

Option	Description
-o sfsschedulestartmount=ID	mount the filesystem and schedule automount shortly after next boot. ID only refers to some identifier for the scheduled mount, for instance: Project1.
-o sfsstartscheduledmount=ID	start a scheduled mount with the given ID.
-o sfsstopscheduledmount=ID	stop the actual instance of the scheduled mount with the given ID.
-o sfsdeletescheduledmount=ID	delete/deschedule the scheduled mount with the given ID.
-o sfslistscheduledmounts	display ID, state, and original command line for scheduled mounts.

The persistent mount options allow client users to schedule automatic mount of the client shortly after the boot of the Windows system is finished (around a minute). It also allows to start, stop and delete/deschedule those configured mounts.

Current implementation requires running the commands using the first four of the previous options to use an elevated (Admin) prompt. Those options use the Windows OS task scheduler and require that number of permissions. Commands which do not involve those options can be run from regular prompt and must be run from regular prompt if it is intended to mount a drive for the current user. The options:

- sfsstartscheduledmount
- sfsstopscheduledmount
- sfsdeletescheduledmount
- sfslistscheduledmounts

will ignore all other options provided in the command line, so it is recommended to only use that option.

Example 1

Command

```
saunafscli –o sfsschedulestartmount=ProjectS –H 192.168.56.1 –P 9521 –D S –o
sfsuncpath=storage/ProjectS
```

on elevated prompt will mount the drive with the provided options (like in the previous example) and schedule to mount the same drive after boot. If that command is run on regular prompt will receive permission denied error.

Example 2

```
saunafscli –H 192.168.56.1 –P 9521 –D S –o sfsuncpath=storage/ProjectS
```

on elevated prompt appear to successfully mount the client but won't show the drive. If that command is run on regular prompt, it will get the already described effect.

Example 3

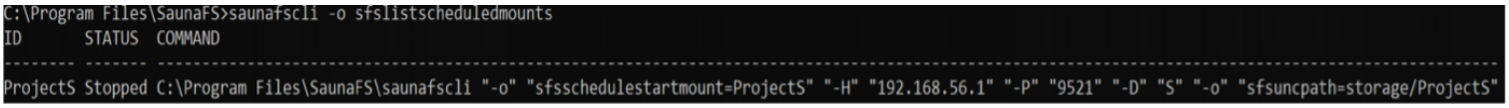
```
saunafscli –o sfsstopscheduledmount=ProjectS
```

will stop the already running mount.

Example 4

```
saunafscli –o sfslistscheduledmounts
```

if following the previous commands should show following:



Configuration file option

-c CFGFILE	-o sfscfgfile=CFGFILE	load some mount options from external file.

Example

```
saunafscli –c C:\\ProjectS.cfg
```

loads mount options from that file, if exists, and mounts the client using those options. Example configuration file:

ProjectS.cfg

```
# Here you can write some explanation
sfsdriveletter=S

# These are comment lines
sfsuncpath=storage/ProjectS

# You can write multiple entries in a single line
sfsmaster=192.168.56.1,sfsport=9521
```

The resulting mount of this example config file will be the same as shown in the Example 2 under section "Extended general options" shown as example.

Side note

The descriptions of previous options contain some italic and non-italic words. The italic description appears on the help command response

```
saunafscli –help
```

The rest of the description has been added especially for the documentation. The command

```
saunafscli –V
```

displays the version info.

FAQs

Why can't AJA see my S: drive, only C:?

AJA only recognizes drives mounted as network drives. To ensure AJA can see your drives, you should mount them using the `–o sfsuncpath=PATH` option. For example, use `–o sfsuncpath=server/drive` to mount your drive as a network drive. Note that AJA does not recognize drives mounted as regular drives from the SYSTEM user.

Version: 4.0.8

NFS client

SaunaFS implements a File System Abstraction Layer (FSAL) for NFS Ganesha to allow connecting NFS clients to the cluster.

NFS-Ganesha is an NFS v3, v4 and v4.1 fileserver that runs in user mode on most UNIX/Linux systems. NFS-Ganesha is used by SaunaFS for offering NFS v3 and v4 services.

SaunaFS FSAL is compatible with most of the features provided by Ganesha and is capable of managing classical goal replication and erasure coding. In the following sections will be covered the most important steps to setup SaunaFS FSAL and basic Ganesha settings.

More information and documentation to setup other specific options and advanced setups for NFS-Ganesha are available in the following link: https://github.com/nfs-ganesha/nfs-ganesha/wiki/Configurationfile

Installing NFS-Ganesha

SaunaFS FSAL was developed for NFS-Ganesha v4.3. In the future, the goal is to add support for NFS-Ganesha v5.5.

The package **nfs-ganesha v4.3** is available in Ubuntu 23.04 (lunar) repositories. The easiest way to install this package is to add (temporarily) Ubuntu 23.04 repositories to the file **/etc/apt/sources.list** and install this package. After installing NFS-Ganesha, we recommend removing Ubuntu 23.04 repositories to avoid conflicts with possibly outdated packages.

Below there is a list of Ubuntu 23.04 repositories that were tested in our environment to install **nfs-ganeshav4.3**.

```
# Repos Lunar Official
deb http://archive.ubuntu.com/ubuntu lunar main restricted universe multiverse
deb http://archive.ubuntu.com/ubuntu lunar-security main restricted universe
multiverse
deb http://archive.ubuntu.com/ubuntu lunar-updates main restricted universe
multiverse
deb http://archive.ubuntu.com/ubuntu lunar-proposed main restricted universe
multiverse
```

```
deb http://archive.ubuntu.com/ubuntu lunar-backports main restricted universe
multiverse
```

After updating the repositories, we update the packages of new repositories and install nfs-ganesha package with the following commands:

```
apt update
apt install nfs-ganesha
```

> **HINT**
>
> It is recommended to install another FSAL like VFS (**nfs-ganesha-vfs**) to create the library folder where the FSAL should be copied before starting **nfs-ganesha**.

In case it's not possible to install **nfs-ganesha** from Ubuntu repositories, we need to download the source code of Ganesha v4.3 from the official repository and build the binaries before installing. In that case, the file COMPILING_HOWTO.txt can be useful for the building process.

Installing and setup SaunaFS FSAL

The next step is to install the packages for SaunaFS FSAL and its dependencies:

```
apt install saunafs-lib-client saunafs-nfs-ganesha
```

> **HINT**
>
> SaunaFS FSAL library (**libfsalsaunafs.so**) should be installed in Ganesha library path (/usr/lib/x86_64-linux-gnu/ganesha/ in Ubuntu 22.04). Otherwise, when starting nfs-ganesha, SaunaFS FSAL will not be loaded and NFS clients will not be capable of connecting to SaunaFS cluster.

Below there is a basic **/etc/ganesha/ganesha.conf** example file to use SaunaFS FSAL:

```
##################################################
# The ganesha node connects to the saunafs master server
# with the ip address 192.168.99.100:
```

```
#
# To work correctly, all that is required is an EXPORT
################################################

EXPORT
{
    # Export Id (mandatory, each EXPORT must have a unique Export_Id)
    Export_Id = 77;

    # Exported path (mandatory)
    Path = "/";

    # Pseudo Path (required for NFS v4)
    Pseudo = "/";

    # Required for access (default is None)
    # Could use CLIENT blocks instead
    Access_Type = RW;
    Squash = None;
    Attr_Expiration_Time = 0;

    # Exporting FSAL
    FSAL {
        Name = SaunaFS;
        # The address of the SaunaFS Master Server or Floating IP if using uRaft
        hostname = "192.168.99.100";
        # The port to connect to on the Master Server
        port = "9421";
        # How often to retry to connect
        io_retries = 5;
        cache_expiration_time_ms = 2500;
    }

    # Which NFS protocols to provide
    Protocols = 3, 4;
}
```

One important aspect to consider is the **'Name'** value must be set to **SaunaFS**. Otherwise nfs-ganesha will not use SaunaFS FSAL.

After finishing the setup of SaunaFS FSAL, nfs-ganesha needs to be enabled and started:

```
systemctl enable nfs-ganesha
```

```
systemctl start nfs-ganesha
```

Connecting NFS clients to SaunaFS clusters

Before connecting NFS clients to SaunaFS clusters, make sure package **nfs-common** is already installed. This package contains programs like statd, showmount and mount.nfs that are needed for NFS clients to connect successfully.

For connecting NFS clients to one SaunaFS cluster, we can use the following command:

```
mount -vvvv ganesha_server_ip:/ganesha_export /local_mountpoint
```

- The option **vvvv** is optional and enables verbose mode to see whether the mount command was successful and other useful debug information.
- The second parameter is the IP address assigned to the Ganesha server, followed by the export (defined at ganesha.conf) to which we want to connect.
- The third parameter is the local mount point defined to access (locally) to the export defined at the **ganesha.conf** file.

Below, there is an example to connect a NFS client (locally) to NFS-Ganesha server installed at the same workstation:

```
mount -vvvv localhost:/ /mnt/export1/
```

The folder /mnt/export1/ is the directory under which the NFS share will be mounted on the client machine. This directory can be changed to any directory name.

Once NFS client is connected to the cluster, we can perform the following operations:

- stat
- readdir
- create folders and files
- remove dir and files
- cat

- symbolic links
- change permissions
- copy files and folders
- rename files and folders

NFS protocol version, client's authorization, and multiple exports

NFS-Ganesha supports NFS v3, 4.0, 4.1, and 4.2. When connecting NFS clients to the cluster, it’s possible to define a specific version of NFS to be used by the client. Below there are some examples to connect NFS clients with different NFS versions:

- NFS v3: mount -o nfsvers=3 localhost:/ /mnt/nfs3
- NFS v4.1: mount -o v4.1 localhost:/ /mnt/nfs41
- NFS v4.2 (default in Ubuntu 22.04): mount localhost:/ /mnt/nfs42

Another important step during configuration of an export is the list of clients authorized to access the export. The following example shows different ways to allow clients to connect to a given export:

```
EXPORT
{
    …
    CLIENT
    {
        #Clients = 192.168.208.236;
        #Clients = *;
        #Clients = 192.168.208.0/24;
        #Clients = mfsmaster;
    }
     …
}
```

The list of clients is usually defined inside a CLIENT section at the **ganesha.conf** file. The most frequent ways to authorize NFS clients are:

- Specific IP address for a single client: Clients = 192.168.208.236;
- All clients: Clients = *;

- Specific subnet: Clients = 192.168.208.0/24.
- Hostname of clients: Clients = nfsclient01;

NFS-Ganesha also allows to define multiple exports for the same namespace. The following example shows the definition of multiple exports in the same **ganesha.conf** file.

```
EXPORT
{
     Attr_Expiration_Time = 0;
     Export_Id = 1;
     Path = /export1;
     Pseudo = /e1;
     Access_Type = RW;

     FSAL
     {
          Name = SaunaFS;
          hostname = localhost;
          port = ${saunafs_info_[matocl]};
     }

     Protocols = 3, 4;
}

EXPORT
{
     Attr_Expiration_Time = 0;
     Export_Id = 2;
     Path = /export2;
     Pseudo = /e2;
     Access_Type = RW;

     FSAL
     {
          Name = SaunaFS;
          hostname = localhost;
          port = ${saunafs_info_[matocl]};
     }

     Protocols = 3, 4;
}

EXPORT
{
```

```
     Attr_Expiration_Time = 0;
     Export_Id = 97;
     Path = /;
     Pseudo = /e97;
     Access_Type = MDONLY;

     FSAL
     {
          Name = SaunaFS;
          hostname = localhost;
          port = ${saunafs_info_[matocl]};
     }

     Protocols = 4;
}

EXPORT
{
     Attr_Expiration_Time = 0;
     Export_Id = 99;
     Path = /;
     Pseudo = /e99;
     Access_Type = RO;

     FSAL
     {
          Name = SaunaFS;
          hostname = localhost;
          port = ${saunafs_info_[matocl]};
     }

     Protocols = 4;
}
```

The definition of multiple exports allows us to define different levels of access (RW, MDONLY, RO), which guarantee a fine-grained level of permissions for different folders in the namespace.

The way to connect to multiple exports is the same as before, we only need to define a NFS directory for each export.

Version: 4.0.8

Administration Guide

Under Development

Version: 4.0.8

Installation

Debian based distributions

Import the public key used to sign the packages

```
#optionally install dirmngr
#apt install dirmngr
#mkdir $HOME/.gnupg
#or sudo mkdir /root/.gnupg

gpg --no-default-keyring \
    --keyring /usr/share/keyrings/saunafs-archive-keyring.gpg \
    --keyserver hkps://keyserver.ubuntu.com \
    --receive-keys 0xA80B96E2C79457D4
```

It will create a new keyring file `/usr/share/keyrings/saunafs-archive-keyring.gpg` and import the public key used to sign the packages.

> **NOTE**
>
> At the time of writing, the use of **apt-key** is deprecated.

You can verify the keyring file by running the following command:

```
gpg --no-default-keyring \
    --keyring /usr/share/keyrings/saunafs-archive-keyring.gpg \
    --list-keys
```

Next, add our Debian/Ubuntu repository to the apt sources. Make sure that the command lsb_release is installed.

Ubuntu:

```
```

```
sudo tee /etc/apt/sources.list.d/saunafs.list <<EOF
deb [arch=amd64 signed-by=/usr/share/keyrings/saunafs-archive-keyring.gpg]
https://repo.saunafs.com/repository/saunafs-ubuntu-22.04/ jammy main
EOF
```

Update the package list

```
sudo apt update
```

These packages are available on the Debian/Ubuntu repository:

- saunafs-master – Master server
- saunafs-chunkserver – Chunkserver
- saunafs-client – Client (sfsmount)
- saunafs-adm – Administration tools `saunafs-admin`
- saunafs-cgi – SaunaFS CGI Monitor (deprecated)
- saunafs-cgiserv – Simple CGI-capable HTTP server to run SaunaFS CGI Monitor (deprecated)
- saunafs-metalogger – Metalogger server
- saunafs-common – SaunaFS shared library, required by saunafs-master, saunafs-chunkserver and saunafs-metalogger
- saunafs-dbg – Debugging symbols for all the SaunaFS binaries
- saunafs-uraft - High Availability solution based on RAFT algorithm (from version 3.13)

Source installation

Obtain the source

```
git clone https://github.com/leil-io/saunafs.git
```

Go into the saunafs directory and create a build directory

```
cd saunafs
mkdir build
```

SaunaFS uses CMake as its build system. For a complete list of options check the developer's guide for building, but these are the three most important options for installing:

- `DCMAKE_BUILD_TYPE=RelWithDebInfo` - Build for release with debug symbols
- `DCMAKE_INSTALL_PREFIX=/usr/local` - Where to install when `make install` is called (default is /usr/local)
- `DENABLE_DOCS=ON` - Build man docs

You might use the below commands to build SaunaFS:

```
cmake -B ./build \
  -DCMAKE_BUILD_TYPE=RelWithDebInfo \
  -DCMAKE_INSTALL_PREFIX=/usr/local
  -G 'Unix Makefiles' \
  -DENABLE_DOCS=ON \
  -DENABLE_CLIENT_LIB=ON \
  -DENABLE_TESTS=ON \
  -DENABLE_WERROR=ON

nice make -C ./build -j$(nproc)
```

Finally call **make install**:

```
sudo make install
```

> **NOTE**
>
> When building from source using this method, the version will default to `4.0.0-devel`. This default value is intended to avoid backward compatibility issues and to easily tag artifacts that are not officially built by our CI.

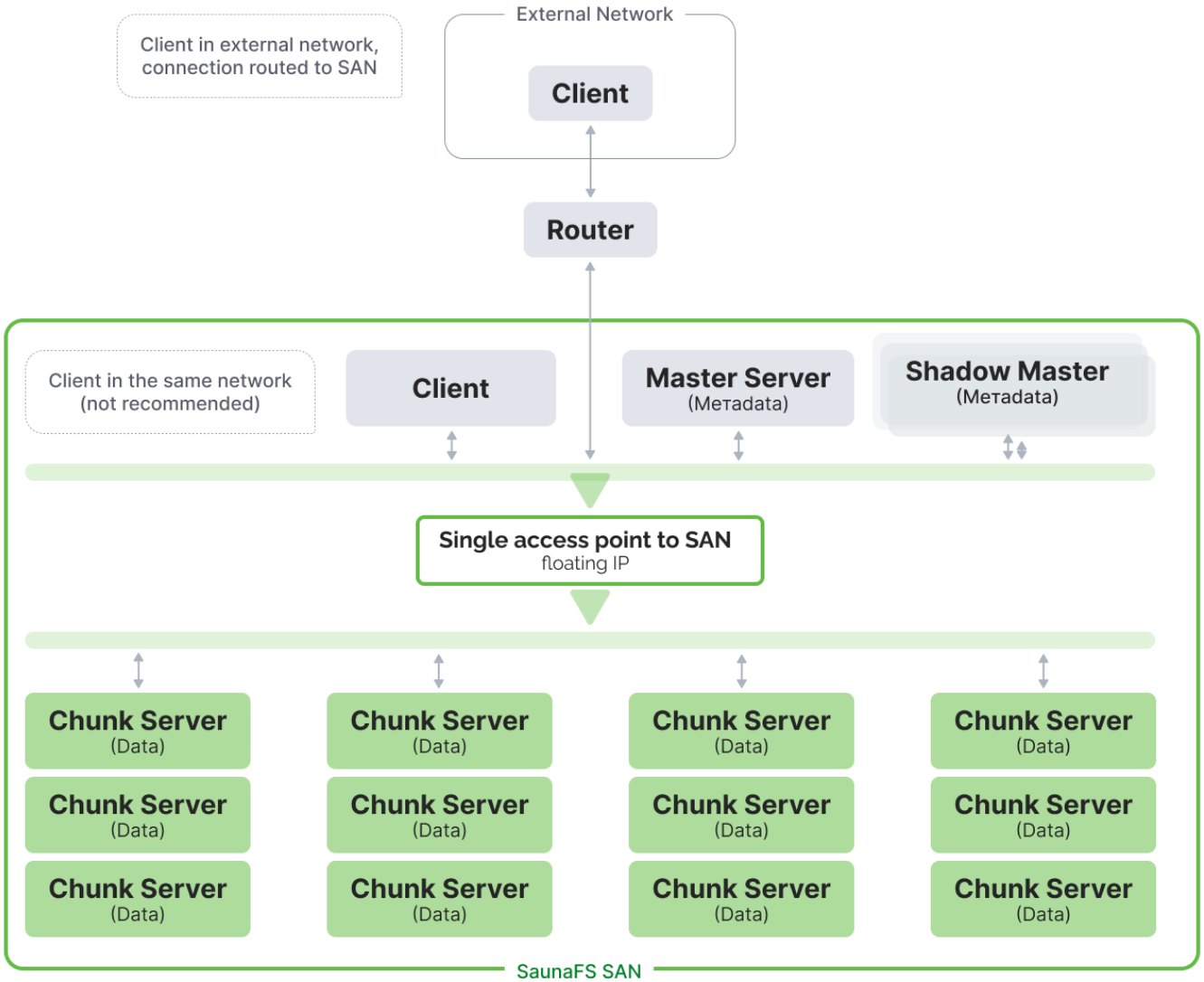
Version: 4.0.8

Network setup

We recommend setting up a static IP address for each SaunaFS server.

SaunaFS SAN (Storage Area Network) is a single IP access private network. This refers to the fact that there is only one floating IP address that clients use to access the SAN. Floating in this case means, this IP address can be assigned to different servers, which helps to improve performance and reliability.

Client connection to SaunaFS SAN



In general, there are two ways for a client to connect to SaunaFS SAN (both illustrated in the according diagram):

1. Client is in external network (outside SaunaFS's storage system's private network). In that case connection can be routed to SAN with VPN or physical router device.
2. Client is in the same network. Client computer is located within the same network as the SaunaFS storage system. However, this is not recommended for security reasons.

DNS

We do not recommend using DNS to resolve the IP addresses of the system. If something in the DNS breaks, it could cause some critical services to not work properly.

Instead, assign the IP addresses a name in /etc/hosts.

Network Topology

The configuration of rack awareness in a SaunaFS network involves setting up the network topology in the **sfstopology.cfg** file. This file specifies the topology using lines that include an ADDRESS and a SWITCH-NUMBER. ADDRESS can be defined in various ways, including as a wildcard for all addresses, a single IP address, an IP class with a network address and bits number or mask, or an IP range.

*	all addresses
n.n.n.n	single IP address
n.n.n.n/b	IP class specified by network address and bits number
n.n.n.n/m.m.m.m	IP class specified by network address and mask
f.f.f.f-t.t.t.t	IP range specified by from-to addresses (inclusive)

The switch number is a positive 32-bit integer. The distances calculated from this configuration are used to prioritize chunk servers during read/write operations based on their proximity to a client. Servers closer to a client are preferred.

However, new chunks are still created randomly to ensure equal distribution, and rebalancing procedures do not consider topology configuration. The distance between switches is categorized as 0 (same IP addresses), 1 (different IP addresses but same switch number), or 2 (different switch numbers).

This topology feature can be effectively combined with chunk server labeling to optimize client interactions with chunk servers, ensuring they read from or write to servers that are best suited for them, like those on the same network switch.

Version: 4.0.8

Service configuration

Here are some fundamental guidelines:

- The SaunaFS Master server is ideally run on a separate machine with an SSD.
- A Chunk server should have at least one dedicated disk.
- Avoid installing a metalogger on the same machine as the master server. Metaloggers are optional. However, they can coexist with a chunk server.
- Use shadow master servers to enhance data safety and allow for failover.

Before setting up SaunaFS, ensure each server has:

- Proper network settings.
- Suitable kernel settings.

Operating Systems

For the purposes of this documentation, we will assume you are on a Debian-based system, and specifically Ubuntu. However, most of what applies here also applies to most Linux distributions. If not, please contact us (See Contacts), we are also working on documentation for other OS-es.

Clients can be POSIX operating systems and Windows 11.

File systems

- For Metadata servers:
  - Use fast SSDs with a fast file system like XFS.
  - Along with HW RAID mirroring or better e.g. RAID 10, RAID 6 etc.
  - Avalible space for metadata should be minimum 2 x RAM planned RAM usage on Metadata server.
- For Chunk servers: If using XFS, certain mount options like:
  - `/dev/disk/by-id/ata-WDC_WUH721414ALE604_XXXXXX /mnt/sfschunkservers/data/ata-WDC_WUH721414ALE604_XXXXXX xfs`

```
rw,noexec,nofail,nodev,noatime,nodiratime,largeio,allocsize=16777216,inode64
0 0
```

Adjust the scheduler for your file systems based on your hardware and needs.

Master

The master server holds all critical file system information.

copy default configuration files:

```
cp -vi /usr/share/doc/saunafs-master/examples/* /etc/saunafs/
```

if this is a new installation create empty metadata by by copying `/var/lib/saunafs/metadata.sfs.empty` to `/var/lib/saunafs/metadata.sfs`

> **DANGER**
>
> This could be dangerous if it is NOT new/clean installation, since it is overriding potentially existing metadata.
>
> `cp -iva /var/lib/saunafs/metadata.sfs.empty /var/lib/saunafs/metadata.sfs`

And now you can configure the `sfsmaster.cfg` file with details like:

- server personality,
- listening addresses (for other services to connect to), ports,
- user/group,
- metadata storage location,
- access time recording
- ...

For network permissions and access rights, use the `sfsexports.cfg` file.

Shadow master

The shadow master mirrors the master server's settings and keeps its meta database synchronized. Set the shadow master's personality and master host address in the sfsmaster.cfg file:

```
PERSONALITY = shadow
MASTER_HOST = <master ip>
```

Note that the files sfsexports.cfg, sfsgoals.cfg and sfstopology.cfg need to be the same as in the actual master at all times.

Start the service like you would start master. See man sfsmaster.cfg and man sfsexports.cfg for more info.

Chunkserver

file `sfshdd.cfg.` must include paths/mountpoints to the disks you want to use in your storage system.

The chunkserver will assume these directories are dedicated drives and will calculate the total space and usage from that.

By default, the master will try to balance chunks evenly between the chunkservers. If some of the chunkservers are doing other non-SaunaFS related IO-operations, you may wish set the ENABLE_LOAD_FACTOR option in sfschunkserver.cfg.

Metalogger

Metalogger helps keep a backup of the (shadow) master servers in case anything happens to them. Without this, if all master and shadow's die, all data is lost.

These settings need to be set in the sfsmetalogger.cfg

```
MASTER_HOST=<master ip address/name>
MASTER_PORT=<master port>
```

Start the metalogger with systemctl:

```
systemctl enable --now saunafs-metalogger
```

See man sfsmetalogger[.cfg] for more details.

Version: 4.0.8

Replication

SaunaFS supports three replication modes.

- Simple Goal Setup: Specify the number of copies for each file or directory chunk across chunk servers.
- EC Mode: Advanced erasure coding with configurable data and parity copies. Clients write quasi-parallel to chunk servers, with up to 32 data and parity chunks.

> **NOTE**
>
> Replication settings are chunk-based, not node-based. For example, with five chunk servers in an EC3+1 setup, chunks are evenly distributed, ensuring active use of all servers and balanced distribution of data and parity chunks.

> **NOTE**
>
> To ensure repair procedures for broken servers, always have an extra chunk server beyond your configured goals.

Configuring Goals

Goals are set in 'sfsgoals.cfg' managed by the master server. The file syntax is:

```
id name : label ...
```

Comments start with '#'. Up to 40 goals can be configured, with IDs from 1 to 40. Each file in the system refers to a goal ID and replicates accordingly. 'sfsgoals.cfg' allows overriding default behaviors.

Goal Definitions

- **Id**: Redefines the goal ID. Changing an ID affects files already assigned to it.
- **Name**: A user-friendly name for interface tools like 'saunafs setgoal'. Names can be up to 32 alphanumeric characters.

- **List of Labels**: Defines chunk server labels, with up to 32 alphanumeric characters. Each label represents a chunk server where a file copy is maintained. The label '_' represents any chunk server.

Changing **sfsgoals.cfg** alters the replication behavior for files using that goal ID.

Example:

```
3 3 : _ _ _ # Three copies anywhere
8 not_important_file : _ # One copy
11 important_file : _ _
13 cached_on_ssd : ssd _
14 very_important_file : _ _ _ _
```

For more information:

```
man sfsgoals.cfg
```

Viewing and Setting Goals

- View current goals via command line: `saunafs-admin list-goals <master ip> <master port>` or web interface under 'Config' tab.
- Set goals with: `saunafs setgoal goal_name object`. Use (-r) for directories. Append '+' or '-' to the goal_name to increase or decrease “security” (i.e. goal_name id is higher or lower than id of the current goal), respectively.
- View goals with: `saunafs getgoal object` or `saunafs getgoal -r directory` for directories.

Setting up EC

EC goals are like standard goals but include EC ($ecM,K) definitions in 'sfsgoals.cfg'. EC supports up to 32 data or parity parts.

Examples in 'sfsgoals.cfg':

```
18 first_ec : $ec(3,1) # 3 data, 1 parity on all servers.
```

EC is the fastest replication mode, spreading writes across servers according to set goals.

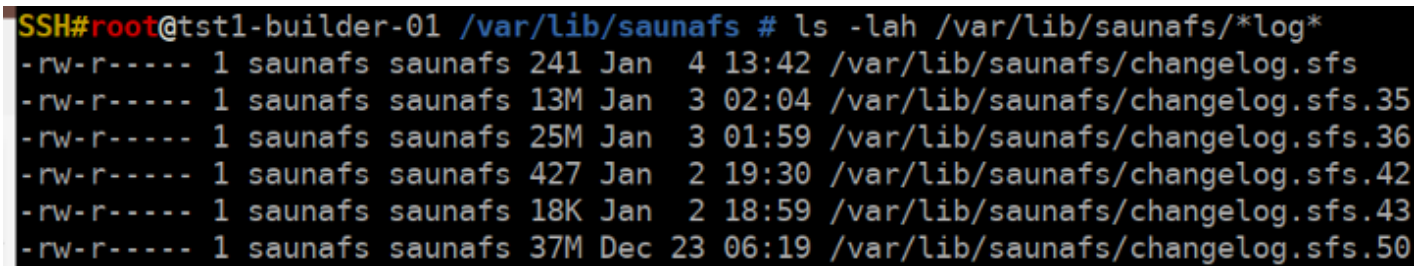
Version: 4.0.8

Logs and logging

There are 3 types of logs in SaunaFS

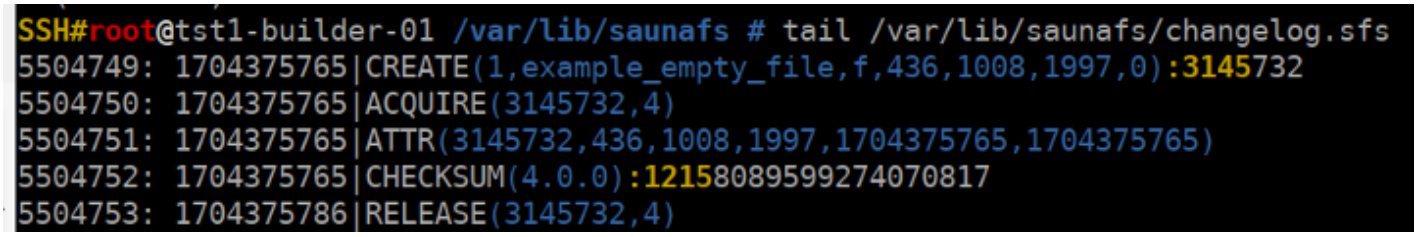
Metadata logs

Each change in the filesystem is being logged Default location for those loges is at /var/lib/saunafs/

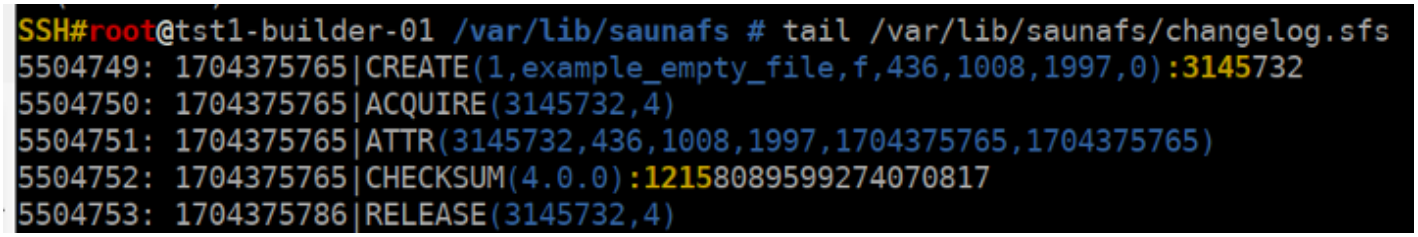



```
#root@tst1-builder-01 /var/lib/saunafs # ls -lah /var/lib/saunafs/*log*
-rw-r----- 1 saunafs saunafs 241 Jan  4 13:42 /var/lib/saunafs/changelog.sfs
-rw-r----- 1 saunafs saunafs 13M Jan  3 02:04 /var/lib/saunafs/changelog.sfs.35
-rw-r----- 1 saunafs saunafs 25M Jan  3 01:59 /var/lib/saunafs/changelog.sfs.36
-rw-r----- 1 saunafs saunafs 427 Jan  2 19:30 /var/lib/saunafs/changelog.sfs.42
-rw-r----- 1 saunafs saunafs 18K Jan  2 18:59 /var/lib/saunafs/changelog.sfs.43
-rw-r----- 1 saunafs saunafs 37M Dec 23 06:19 /var/lib/saunafs/changelog.sfs.50
```

Below we can se an example of empty file creation:



And followed by making snapshot of this emplty file:



tail /var/lib/saunafs/changelog.sfs

```
5504749: 1704375765|CREATE(1,example_empty_file,f,436,1008,1997,0):3145732
5504750: 1704375765|ACQUIRE(3145732,4)
5504751: 1704375765|ATTR(3145732,436,1008,1997,1704375765,1704375765)
5504752: 1704375765|CHECKSUM(4.0.0):12158089599274070817
5504753: 1704375786|RELEASE(3145732,4)
✓ (0.00080s) 13:46:11
```

tail /var/lib/saunafs/changelog.sfs

```
5504749: 1704375765|CREATE(1,example_empty_file,f,436,1008,1997,0):3145732
5504750: 1704375765|ACQUIRE(3145732,4)
5504751: 1704375765|ATTR(3145732,436,1008,1997,1704375765,1704375765)
5504752: 1704375765|CHECKSUM(4.0.0):12158089599274070817
5504753: 1704375786|RELEASE(3145732,4)
5504754: 1704376084|CLONE(3145732,1,3145733,example_empty_file.snapshot,0)
```

This logging is giving a powerful ability to apply Realtime or offline analysis (e.g security, usage, anomalies, etc)

Thosche changelog files also include checksum (which could be use to determine manipulation of logs data) used to detect potential errors in logs,

Copy of those changelogs can be stored in dedicated meta logger server in cluster for having a copy.

Regular syslog (journalctl)

In the confiuration file for every server/daemon/service

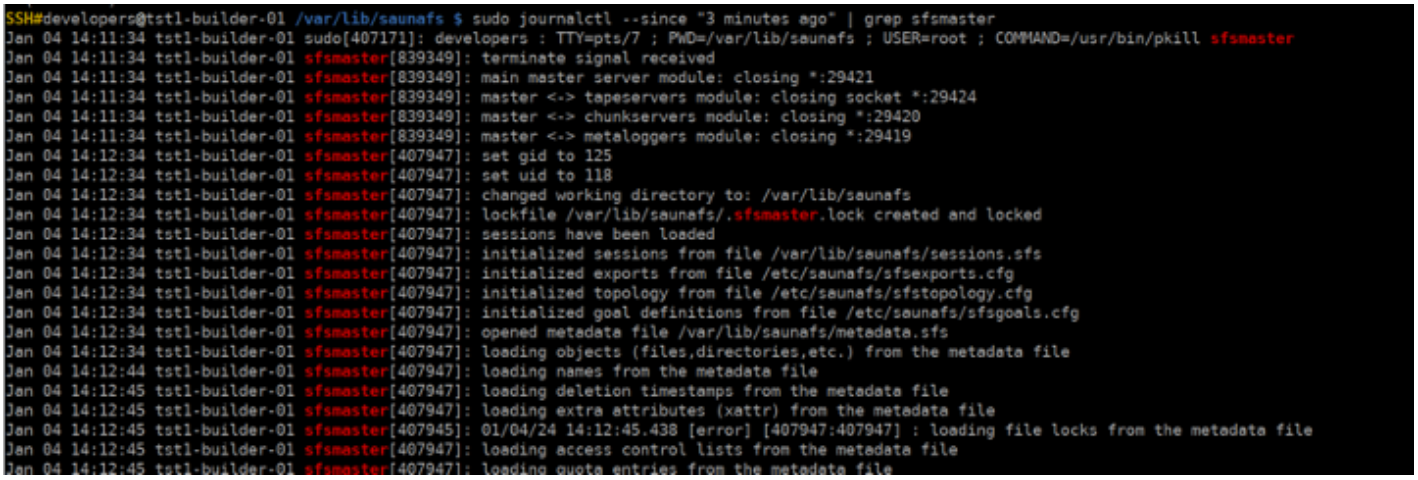
`/etc/saunafs/sfsmaster.cfg`

We have posibility to define syslog ID

Default for master server is:

```
# SYSLOG_IDENT = sfsmaster
```

```
sudo journalctl --since "3 minutes ago" | grep sfsmaster
```



Client site operation logs (oplog)

In case we need to determine/monitor whtat is happening in particular mountpoint we can access

```
sudo cat /<MOUNT_POINT>/.oplog
```

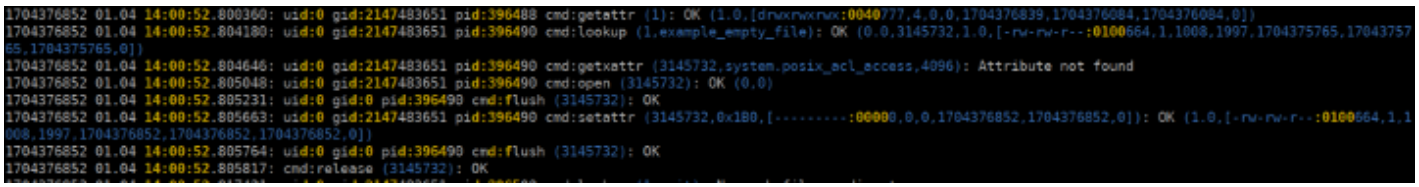
For example

```
sudo grc cat /mnt/sfs.208.29421/.oplog
1704376778 01.04 13:59:38.838828: uid:0 gid:2147483651 pid:395006 cmd:open
(4294967281) (internal node: OPLOG): OK (1,0)
1704376778 01.04 13:59:38.838934: uid:0 gid:2147483651 pid:395006 cmd:getattr
(4294967281) (internal node: OPLOG): OK (3600,[-r-------
-:0100400,1,0,0,0,0,0,0])
```

Example touch

```
touch example_empty_file
```

```
1704376852 01.04 14:00:52.800360: uid:0 gid:2147483651 pid:396488 cmd:getattr
(1): OK (1.0,[drwxrwxrwx:0040777,4,0,0,1704376839,1704376084,1704376084,0])
1704376852 01.04 14:00:52.804180: uid:0 gid:2147483651 pid:396490 cmd:lookup
(1,example_empty_file): OK (0.0,3145732,1.0,[-rw-rw-r-
-:0100664,1,1008,1997,1704375765,1704375765,1704375765,0])
1704376852 01.04 14:00:52.804646: uid:0 gid:2147483651 pid:396490 cmd:getxattr
(3145732,system.posix_acl_access,4096): Attribute not found
1704376852 01.04 14:00:52.805048: uid:0 gid:2147483651 pid:396490 cmd:open
(3145732): OK (0,0)
1704376852 01.04 14:00:52.805231: uid:0 gid:0 pid:396490 cmd:flush (3145732): OK
1704376852 01.04 14:00:52.805663: uid:0 gid:2147483651 pid:396490 cmd:setattr
(3145732,0x1B0,[---------:00000,0,0,1704376852,1704376852,0]): OK (1.0,[-rw-rw-
r--:0100664,1,1008,1997,1704376852,1704376852,1704376852,0])
1704376852 01.04 14:00:52.805764: uid:0 gid:0 pid:396490 cmd:flush (3145732): OK
1704376852 01.04 14:00:52.805817: cmd:release (3145732): OK
```



Version: 4.0.8

Basic checks

- Check if your nodes are all reachable by IP address as well as by hostname
- Check if your network has the right throughput
- Check if your disks are all working and do not report errors
- Check your Chunkservers for:
  - Broken Disks
  - Slow Disks
  - permissions on your directories (the user which is running the chunkserver must be the owner of the directories)
  - network performance to other chunkserver
  - network performance to master server
  - network performance to clients
- Check your license files for the correct name and location
- Check your log files for errors. SaunaFS is very talkative and reports a lot.

Version: 4.0.8

Check the speed of your network interface

To verify what your network interface is set to, you can just use the **ethtool** program:

```
ethtool <interface>
```

Version: 4.0.8

Checking the throughput of your network

The best tool to verify if your network throughput is according to what you think it is would be the **iperf** tool. **iperf** allows you to verify the throughput between two machines. It is available for all POSIX compliant systems and is quite easy to use.

For more information about iperf, please check out https://iperf.fr/.

Version: 4.0.8

Dev Guide

Development Environment

Currently, we target 22.04 Ubuntu LTS for releases. However, we don't recommend using your host machine for development (due to the fact that setting up tests modifies your host system and requires root privileges). Instead, we recommend using a virtual machine to develop and test SaunaFS. In the future, we will probably provide a Docker image for development instead.

You can use any virtualisation software you like.

Editors

You can use any editor you like. The core team uses various editors, including Visual Studio Code, CLion and Vim.

Building

Sharing the source code with the VM

If you want to use the editors on your host machine, you should share the source directory with the VM and do your building/testing/running on the VM. You'll need to check your virtualisation software's documentation for how to do this.

For example, in KVM you can add a filesystem passthrough in virt-manager or by editing the VM XML file directly. If using virt-manager, you should use the virtiofs driver, the source path should be the path to the SaunaFS source code on your host machine, and target path something like saunafs.

You can then mount the shared directory with the following command:

```
sudo mkdir /opt/saunafs
sudo mount -t virtiofs saunafs /opt/saunafs
```

To mount every time you start the VM, add the following line to /etc/fstab:

```
saunafs /opt/saunafs virtiofs defaults 0 0
```

Dependencies/Installing Tests

You can use the following script to both install the dependencies and the testing environment:

```
tests/setup_machine.sh /mnt/hda /mnt/hdb /mnt/hdc /mnt/hdd /mnth/hde /mnt/hdf
```

You can also run the script without arguments, and it will explain what it does.

Compiling

We use CMake for building. There are some useful options you can pass to CMake:

- `DCMAKE_COMPILE_COMMANDS=ON`: This will generate a compile_commands.json file which is useful for editors and tools to understand the build system. It should work fine on the host machine, as long as you have the necessary dependencies installed on the host (otherwise, it might show missing dependencies). Check the previous script code for the dependencies for Ubuntu LTS 22.04.
- `DENABLE_TESTS=1`: This will enable building the tests.
- `DENABLE_DOCS=1`: This will enable building the documentation.

In the source directory, you can run the following commands to build the project:

```
mkdir build && cd build
cmake -DCMAKE_COMPILE_COMMANDS=ON -DENABLE_TESTS=1 -DENABLE_DOCS=1 ..
make -j4 # Generally 4 is a safe option, see below
```

The number of jobs you should use for make depends on the number of cores your VM has (i.e if you have less than 4 cores, you should use less than 4 jobs) and the amount of RAM you have. If you use too many jobs, you could run out of RAM pretty quickly. For example, with 32 cores, you could use 32 jobs, but you'll need about 100GB of RAM.

After make finishes, you can install SaunaFS with the following command:

```
```

```
sudo make install
```

You can also `sudo make -j$(jobs) install` to both build and install in one command. However, you'll need to use `sudo` every time you want to build.

Configuring SaunaFS/tests

We use a mixture of unit and integration tests. The integration tests require some more setup. Assuming you ran the setup_machine.sh script, you need to edit the /etc/saunafs_tests.conf file, uncomment the part with SAUNAFS_ROOT, and set that to /usr/local/ (or wherever you installed SaunaFS).

You also need to setup networking a bit. Currently SaunaFS forbids communication with master on localhost. You need to add a new IP address to your loopback interface. You can do this with the following command:

```
sudo ip addr add 10.33.33.33 dev lo
```

To make this change permanent, you can add the following line under `ethernets` in /etc/network/00-installer-config.yaml (if you installed Ubuntu Server):

```
  lo:
    addresses:
    - 10.33.33.33/8
```

Generate the configuration with the following command:

```
sudo netplan --debug generate
```

Restart the network service with the following command:

```
sudo systemctl restart systemd-networkd
```

Verify localhost works with both 127.0.0.1 and 10.33.33.33 with ping.

You should then set `sfsmaster` in /etc/hosts to that IP address for ease of remembering the IP address.

```
10.33.33.33 sfsmaster
```

Running the tests

Running unit tests is easy, in the build directory after building, run the following command:

```
src/unittests/unittests
```

For the integration tests, there are test suites available. These are managed by `gtest` and are simple shell scripts. You can see all of the test suites in the tests/test_suites directory, but the most important one is the `SanityChecks` suite. This should be run to ensure nothing is broken and before submitting a pull request.

To run the `SanityChecks` suite, you can run the following command:

```
saunafs-tests --gtest_filter="SanityChecks.*"
```

Others of note are the `LongSystemTests` and the `ShortSystemTests`. These are run by the CI system. The `ShortSystemTests` take about an hour to run, and the `LongSystemTests` can take up to a day.

Submitting Pull Requests

See the CONTRIBUTING.md file for more information on how to submit pull requests.

Git specific settings

Code Style

We use clang-format to enforce a consistent code style. You can run something like git-clang-format to format your changes before committing.

```
git add <your changes>
git clang-format --style=file
```

Ignore revisions

Sometimes you want to ignore certain revisions when running git blame (i.e large rename commits). You can use the `.git-blame-ignore-revs` file to do this.

```
git config blame.ignoreRevsFile .git-blame-ignore-revs
```

Version: 4.0.8

Introduction

This section provides an overview of the licensing terms for SaunaFS software, Windows Client software and this documentation itself, all licensed separately under different licensing formats.

Documentation licensing information

SaunaFS Documentation is licensed separately from SaunaFS and is covered by the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License (CC BY-NC-ND 4.0).

The full text of the license can be found at https://creativecommons.org/licenses/by-nc-nd/4.0/.

This documentation may be used for non-commercial purposes only and may not be modified or distributed without prior permission from the copyright holder. When using this documentation, you must include the original licensing information and provide attribution to the copyright holder. When you redistribute this documentation, please maintain the original licensing information, and distribute this notice along with the documentation.

Version: 4.0.8

Windows Client licensing information

The software component "Windows Client" is not open source and is subject to a commercial license. To obtain a license to use Windows Client, please contact Leil Storage OÜ at contact@leil.io or using the contact information provided on our website at https://leil.io

Version: 4.0.8

SaunaFS (except its documentation and Windows Client) licensing information

This software is released under the terms of the GNU General Public License version 3 (GPLv3), which can be found at https://www.gnu.org/licenses/gpl-3.0.html. This software is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version. This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. When you redistribute this software, please maintain the original licensing information, and distribute this notice along with the software. (GPLv3): This subsection provides a detailed description of the remaining components of your software, which are licensed under the GPLv3 license. This subsection should include the full text of the GPLv3 license, as well as any additional terms or conditions that apply to the software as a whole.