LeilFS Documentation

Version: 5.0.0

Table of contents:

SaunaFS documentation overview

- Introduction
  - Architectural overview of SaunaFS
- Important notes
  - Hardware recommendations
  - Manual pages
  - Systemd services
  - Versioning
    - Breaking changes
    - Non-breaking changes

Getting started

Creating the required directories

Setting up network on localhost (Optional)

Configuring and running master and chunkserver

Mounting the client

Wrapping up

Container-based Deployment (Docker/Podman)

- Requirements
- Get the container repo
- Build and run with Docker
- Build and run with Podman
- Access the CGI monitor
- Data persistence and initialization
- Cleaning up data

Migrations

- Downgrade scripts
- 5.0.0
  - Upgrade to 5.0.0
  - Downgrade from 5.0.0
    - Instructions

Administration Guide

Windows client

- General options
  - Extended general options
    - Example 1
    - Example 2

        - Example 3
        - Example 4
- Security/permissions related options
- Read related options
    - Example
- Write related options
    - Example
- Persistent mount options
    - Example 1
    - Example 2
    - Example 3
    - Example 4
- Configuration file option
    - Example
    - Further examples
- Other options
    - Side note
- SFSMount Manager
    - Usage
        - Example 1
        - Example 2
        - Example 3
- SaunaFS Admin Tool
    - Usage
        - Example 1
        - Example 2
- SaunaFS Command Tool
    - Usage
    - SaunaFS Tools Overview
        - Extra info for tool commands extra options
        - Deprecated Tools
        - Example 1
        - Example 2
        - Example 3
- SaunaFS Right-click Context Menu
    - Example 1
    - Example 2
    - Example 3

- User-wise behavior
- FAQs
  - Why can't AJA see my S: drive, only C:?
  - Why the client is failing to mount some subfolder of the storage?
  - Why my IIS server is not able to load the data from storage?
  - Why is the client using considerable disk resources if I am not performing any operation on the mountpoint?
  - Why is the Windows client UNC path option not working?
  - Why can't I write data to a file while the client is running?
  - How can I adjust the size of the usable space on a drive according to my needs?

NFS client
- Installing NFS-Ganesha
  - Ubuntu LTS 22.04
  - Ubuntu LTS 24.04
- Installing and setup SaunaFS FSAL
- Connecting NFS clients to SaunaFS clusters
- NFS protocol version, client's authorization, and multiple exports

Common Gateway Interface (CGI) Monitor
- How to run it
- Q&A Section

Installation
- Ubuntu
- Source installation

Network setup
- Client connection to SaunaFS SAN
- DNS
- Network Topology

Service configuration
- Operating Systems
- File systems
- Master
- Shadow master
- Chunkserver
- Metalogger

Storage device setup/removal
- Recommended Filesystems
- Adding storage devices
- Replacing storage devices (non-damaged)

- Removing/replacing storage devices

Replication

- Configuring Goals
- Goal Definitions
- Viewing and Setting Goals
- Setting up EC

Logs and logging

- Metadata logs
- Regular syslog (journalctl)
- Client site operation logs (oplog)

Basic checks

Check the speed of your network interface

Checking the throughput of your network

Dev Guide

- Development Environment
- Editors
- Building
  - Sharing the source code with the VM
  - Dependencies/Installing Tests
  - Compiling
  - Configuring SaunaFS/tests
  - Running the tests
- Submitting Pull Requests
- Git specific settings
  - Code Style
  - Ignore revisions

Introduction

- Documentation licensing information

Windows Client licensing information

SaunaFS (except its documentation and Windows Client) licensing information

Version: 5.0.0

SaunaFS documentation overview

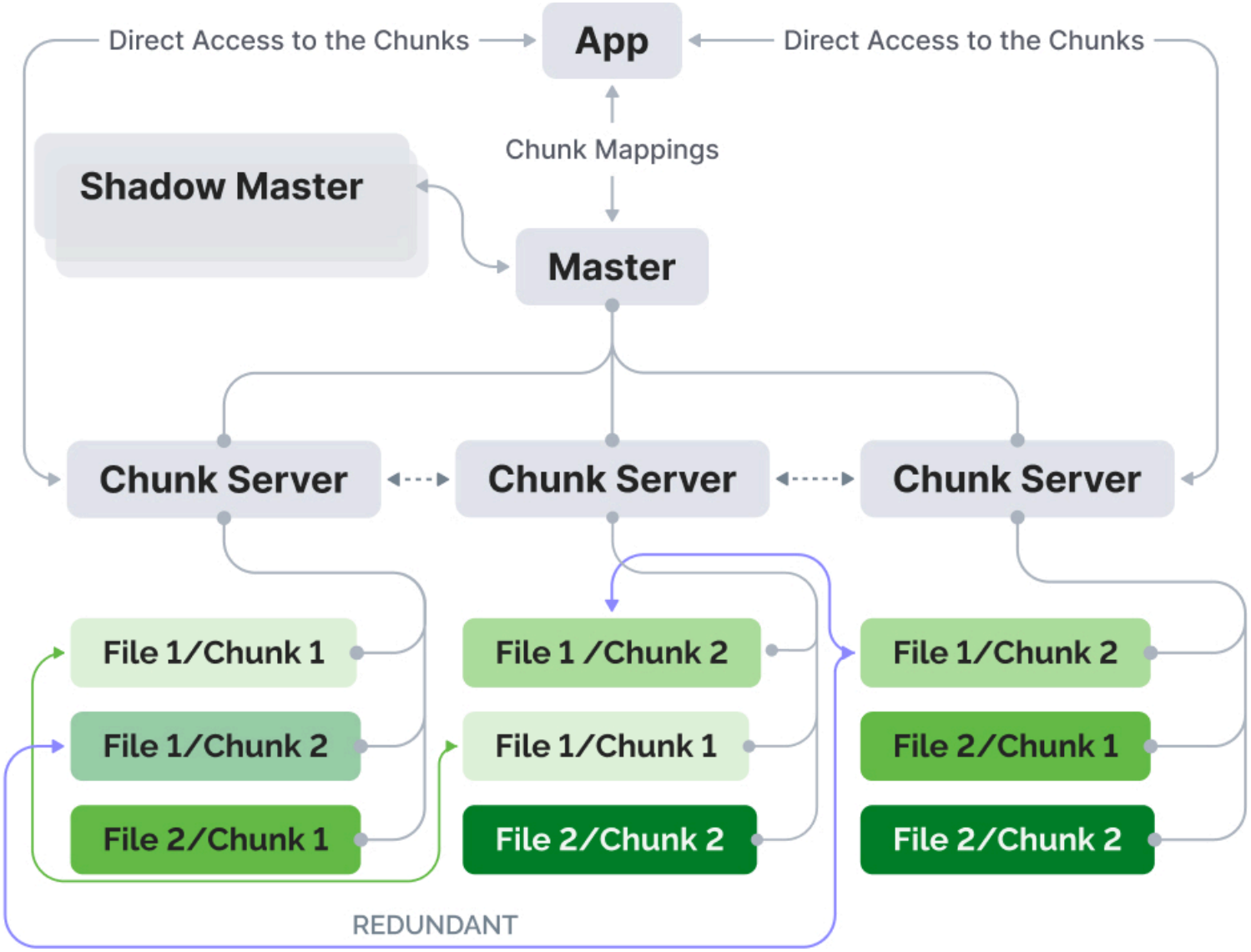
Introduction

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 replication goal for specific files and directories can be configured to use Erasure Coding based on Reed-Solomon for redundancy purposes. For instance, a file with 64 MiB of data, with replication goal EC(4,2), will divide the chunk into 4 data parts of 16 MiB and 2 parity parts of 16 MiB. This way, there is no data loss even if two Chunkservers are down or any two parts are lost.

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.

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.

Versioning

We use semantic versioning for SaunaFS, where the versions are described as:

MAJ.MIN.PAT

Where:

MAJ - Major version

MIN - Minor version

PAT - Patch version

We guarantee seamless upgrades/downgrades between Minor and Patch versions. Between these kind of versions, no breaking changes will be introduced.

Downgrading is supported by one major version down.

See Migrations on how to handle downgrades/upgrades between major versions

Breaking changes

Changes to these are considered breaking changes, and will be a major version bump

- Changes to behavior in the software as described in the man pages and docs.saunafs.com
- Command-line interface options (as described in the documentation/man pages)

- Default configuration values
- Clients, metadata servers and/or chunkservers must be able to communicate with each other in the same minor and/or patch versions.
- Upgrading/downgrading from any minor version to another minor version in the (for example, from 4.0.1 to 4.2.0 or from 4.2.0 to 4.0.1)

Currently non-breaking changes

Not currently considered as breaking changes (unless it affects the behavior of the points above), but may change in the future

- Binary API interfaces that our clients (sfsmount and saunafs-admin) use to communicate with chunkserver/master.

Non-breaking changes

Will not be considered breaking changes in minor/patch version bumps:

- Internal protocol used by metadata servers, chunkservers and metaloggers to communicate with each other between **MAJOR** versions.
- Testing related code
- Output logs
- Output of command line executables.
- Improvements to documentation

Version: 5.0.0

Getting started

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.

Note that while SaunaFS master does work on localhost, this is experimental. You should probably setup a dedicated IP address for master, otherwise strange behaviours can occur (see Setting up on localhost).

Version: 5.0.0

Creating the required directories

For the purposes of this setup, we will set up everything on one machine. We need three directories: Two to store data, and one for client. In a production scenario, the two directories storing data would be dedicated drives, but for now we will set them up as simple directories as that works as well for testing purposes.

```
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. Note that if the directory is usually mounted, it's usually a good practice to make sure the unmounted directory is owned by any other user other than saunafs (root is a good choice). This is a extra layer of protection to ensure that SaunaFS does not write to unmounted directories.

```
chown saunafs:saunafs /mnt/hd1
chmod 755 /mnt/hd1

chown saunafs:saunafs /mnt/hd2
chmod 755 /mnt/hd2
```

Version: 5.0.0

Setting up network on localhost (Optional)

While master/chunkserver will work with each other if running on the same IP address, it's best to assign master it's own dedicated IP.

First, add an identifier for `sfsmaster` 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: 5.0.0

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: 5.0.0

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 sfsmount /mnt/client/
```

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

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

Version: 5.0.0

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: 5.0.0

Container-based Deployment (Docker/Podman)

This section adapts the saunafs-container repository README for easy reference within the SaunaFS docs. It helps you spin up a demo cluster using Docker or Podman. It is intended for testing, demos, and education—not for production.

> ⚠ **TESTING AND EDUCATION ONLY**
>
> This container setup is designed for demos and playgrounds. It should NOT be used for production data!

Requirements

- Docker and Docker Compose (v2) or Podman and podman-compose
- ~1 GB free disk space recommended for simulating storage replication

Get the container repo

Clone the saunafs-container repository:

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

Builds use the public SaunaFS APT repository and do not require credentials.

Build and run with Docker

> Note: On some systems, the `buildx` plugin may need to be installed first (e.g., Ubuntu):

```
sudo apt install -y docker-buildx
```

```
# Build the shared base image (no credentials required)
docker build \
  -f saunafs-base/Dockerfile \
  -t saunafs-base saunafs-base/

# Build and start all services
docker compose up --build
```

Build and run with Podman

If you previously created a `./volumes` folder while using Docker, delete it before deploying with Podman to avoid permission issues.

```
# Build the shared base image (no credentials required)
podman build \
  -f saunafs-base/Dockerfile \
  -t saunafs-base saunafs-base/

# Build and start all services
podman-compose up --build
```

> Note: `docker compose` (v2) or `podman-compose` is recommended.

Access the CGI monitor

Open the CGI monitor in your browser:

http://localhost:29425/sfs.cgi?masterhost=master&masterport=9421

Data persistence and initialization

This deployment is designed for convenience:

- No pre-committed data. The `volumes/` directory is not part of the repo.

- Automatic initialization on first startup: each service will create default configuration files (from `/usr/share/doc/saunafs-*/examples/` in the containers) and initialize data directories.
- Persistent data: If you map volumes to standard SaunaFS paths (e.g., `/var/lib/saunafs/`, `/etc/saunafs/`), your data and configs persist across restarts. If these are empty initially, they are auto-initialized.
- Chunkserver storage:
  - Chunkservers look for mount points at `/mnt/hdd001`, `/mnt/hdd002`, etc.
  - If you provide external volumes mounted to these paths in your `docker-compose.yml`, they will be used.
  - If not provided, the startup script creates these directories inside the container (volatile storage) and logs a warning. Suitable for testing, not for production.

Cleaning up data

If you used Docker named volumes or host-mounted directories, your data persists after containers stop.

To reset the environment:

- Named volumes:
  - List: `docker volume ls`
  - Remove: `docker volume rm <volume_name>`
- Host-mounted directories:
  - If you created a local `volumes` directory and mapped paths (e.g., `./volumes/master/data:/var/lib/saunafs`), delete those directories on your host.

Example cleanup (host-mounted `./volumes/`):

```
# WARNING: Permanently deletes data. Verify the path!
sudo rm -r ./volumes
```

> 🔥 **BE CAREFUL WITH DESTRUCTIVE COMMANDS**
>
> Using `rm -r` on the wrong path can cause irreversible data loss. Double-check the directory.

Version: 5.0.0

Migrations

This is a high level guide to how to handle migrations between major versions with the official packages. For more technical details, see saunafs-migrations 7

Downgrade scripts

The official packages include saunafs-common, which includes helper scripts to help with downgrades/upgrades. They can be found `/usr/lib/saunafs/package-scripts/migrations`. This package is automatically installed when any other SaunaFS package is installed. These scripts handle the downgrade/upgrade automatically between the major versions. In particular, it will automatically install old/newer packages using apt and then apply the correct migrations. Please read the README located in the specific versions to understand the specifics and run `<script> --help` for more details.

5.0.0

Upgrade to 5.0.0

No action should be needed. Simply updating to the latest versions should work fine. However, note that newer chunkservers will not work with older metadata servers.

Downgrade from 5.0.0

You will find the main downgrade script in `/usr/lib/saunafs/migration-scripts/5.0.0/downgrade-master.sh`. This script is responsible for orchestrating the downgrade process for master, metalogger, and uraft components.

It performs the following steps:

1. **Service Control:** It intelligently stops the relevant service. If `saunafs-uraft` is installed and active, it will stop `saunafs-uraft`. Otherwise, it will stop `saunafs-master` (if installed and active). It also stops `saunafs-metalogger` (if installed and active).

2. **Data Migration:** It calls a separate data migration script located at `/usr/lib/saunafs/migrations/5.0.0/rollback/downgrade-changelogs.sh`. This script handles the necessary data fixes and migrations. It also creates a backup in `/usr/lib/`
3. **Package Downgrade:** It downgrades the `saunafs-master`, `saunafs-metalogger`, and `saunafs-uraft` packages (if installed) to version `4.11.0-1` using `apt`.
4. **Service Restart:** Finally, it restarts the services that were stopped in step 1.

Logging for the data migration process is handled by the migration script itself and is saved to `/var/log/saunafs/downgrade/changelog.log`.

Instructions

1. First, downgrade your chunkservers manually with `apt upgrade saunafs-chunkserver=4.11.0-2`. This is not done by the script above and must be done by hand. This is needed because newer chunkservers do not work with 4.11.0 or earlier.
2. Execute `bash /usr/lib/saunafs/migration-scripts/5.0.0/downgrade-master.sh --help` and read the help thoroughly, it contains important information!
3. Execute `sudo bash /usr/lib/saunafs/migration-scripts/5.0.0/downgrade-master.sh` first on the master, and then shadow(s). If you are on uraft, note that older shadows (followers) cannot connect to newer masters (leaders), so you need to downgrade all or most of them before the cluster can start.
4. After everything works, you may delete the backup in /var/lib/.

Version: 5.0.0

Administration Guide

Under Development

Version: 5.0.0

Windows client

The current SaunaFS Windows Client (5.6.0-2) consists of the Command Line Interface (CLI) client, the SaunaFS Admin tool, the SaunaFS command tool and the helper of the CLI client. Install our client from the provided exe file. The CLI client is in “C:\Program Files\SaunaFS\sfsmount.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, as well as the saunafs-admin and sfsmount-manager tools.

For saunafs-admin help, please check the output of the `saunafs-admin --help`. This article will describe most of the options of the CLI client and the ones of the sfsmount-manager.

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.

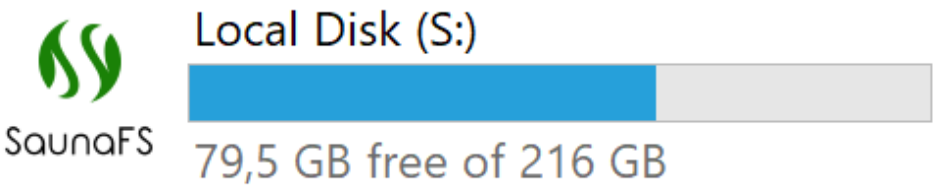
General options

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 behavior takes last one not used.

For instance, the command:

```
sfsmount -H 192.168.1.158 -P 9421 -D S
```

should display a new local drive mounted in the letter S. The IP 192.168.1.158 must host a SaunaFS master server listening for incoming client communications on port 9421.



Extended general options

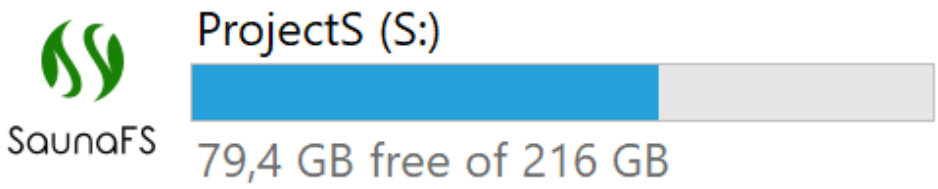
Option	Description
-S PATH | -o sfssubfolder=PATH	define subfolder to mount as root (default: /). The storage could have a number of folders, this option allows the user to display inside a mounted drive only the content of one of those specific folders.
-o sfsvolumelabel=NAME	mounted filesystem will 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` are mutually exclusive. Please only use one of the two while writing mount command because some options will be ignored.

Example 1

```
sfsmount -H 192.168.1.158 -P 9421 -D S -S /data/ProjectS -o
sfsvolumelabel=ProjectS
```

should display the following regular drive:

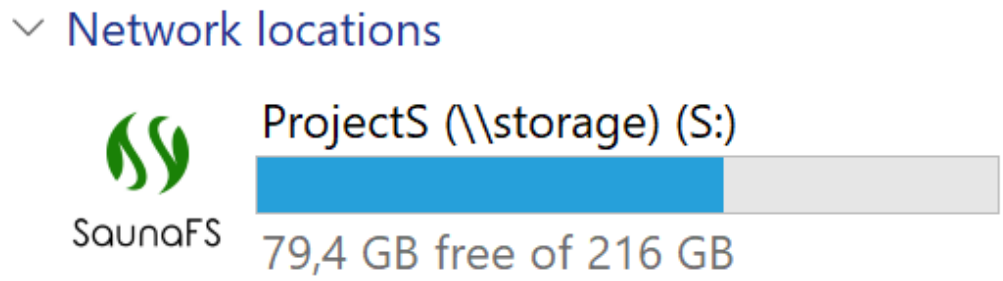


i.e. a regular drive labeled ProjectS and displaying only the contents of the folder “/data/ProjectS” of the storage.

Example 2

```
sfsmount -H 192.168.1.158 -P 9421 -D S -o sfsuncpath=storage/ProjectS
```

should display the following drive:

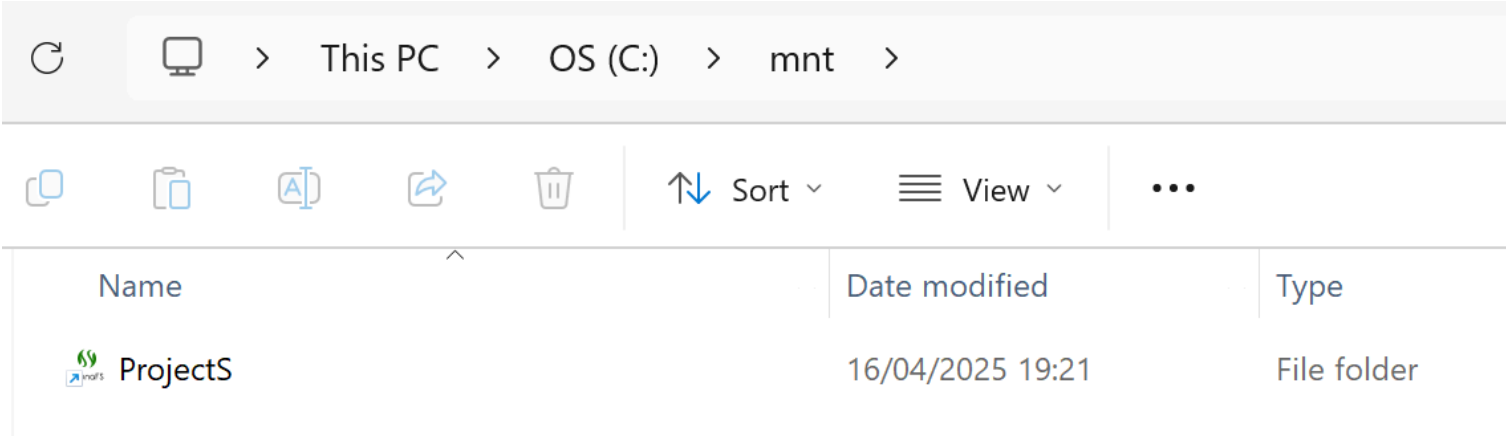


i.e. a network drive on the desired UNC path.

Example 3

```
sfsmount -H 192.168.1.158 -P 9421 -o sfsmountingsubfolder=C:\\mnt -o
sfsvolumelabel=ProjectS
```

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



Example 4

```
sfsmount -H 192.168.1.158 -P 9421 -o sfssubfolder=/storage/volume01 -o
sfsdriveletter=S
```

should display the content of the subfolder "/storage/volume01" in the drive S.

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.
-o sfsdonotrememberpassword	do not remember password in memory - more secure, but when session is lost then new session is created without password.

Option	Description
--uid UID | -o sfsuid=UID	set user id of the mounting user as UID (default behavior uses WinFSP local mapping of Windows SID).
--gid GID | -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.
-o sfsallowedusers=UID1:UID2:...:UIDk	treat users provided as regular local users (default: none). This option hacks some inner client workarounds to make it able to support specific opertions.

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

```
sfsmount -H 192.168.1.158 -P 9421 -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 196608, Administrator user gets mapped to 544, System to 18 and the "No Group" is represented with GID 197121.

Read 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 readcachemaxsizepercentage=P	specifies the maximum percentage of system memory for the read cache (default: 60).
-o readworkers=N	define number of read workers (default: 30).
-o maxreadaheadrequests=N	define number of readahead requests per inode (default: 5).
-o sfschunkserverconnectreadto=MSEC	set timeout for connecting with chunkservers during read operation in milliseconds (default: 2000).
-o sfschunkserverwavereadto=MSEC	set timeout for executing each wave of a read operation in milliseconds (default: 500).
-o sfschunkservertotalreadto=MSEC	set timeout for the whole communication with chunkservers during a read operation in milliseconds (default: 2000).

Option	Description
-o sfsprefetchxorstripes	prefetch full xor stripe on every first read of a xor chunk.
-o bandwidthoveruse=N	define ratio of allowed bandwidth overuse when fetching data (default: 1.00).
-o readbuffersexpirationtime=MSEC	set timeout for read buffers to expire (default: 1000).

The first four of 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 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 `readcachemaxsize` option enforces a hard limit on the amount of data that can be cached when reading, specifying the percentage of the total memory capacity of the system to be used for this purpose. 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.

There is not much to add on the timeout options than to explain the meaning of "read operation" and "wave of read operation". A read operation is the procedure to get the data of a specific chunk, so a read syscall or request to the client may consist of more than one read operation. Meanwhile, a read operation may consist of several waves, depending on the type of the chunk (goal) and `bandwidthoveruse` parameter set. In this regard, a high `bandwidthoveruse` parameter would decrease the number of waves planned to perform the read, posibly increasing the number of chunkservers being read at the same time. The `sfschunkserverconnectreadto` refers to the timeout for the start of connection with a chunkserver at the begining of a read operation.

The client recycles the buffers used in reads to reduce memory allocations and deallocations. The option `readbuffersexpirationtime` allows to configure this feature by defining how long the read buffers are kept in memory before being released.

Example

```
sfsmount -H 192.168.1.158 -P 9421 -D S -o cacheexpirationtime=10000 -o
readaheadmaxwindowsize=32768 -o readworkers=16 -o maxreadaheadrequests=16 -o
sfschunkserverwavereadto=150 -o bandwidthoveruse=0.5
```

Previous command would start a regular mount on S:, with the parameters:

- the data read using the readahead mechanism would last 10s in cache.
- the maximum size of the window to be read ahead is 32MB.
- the number of read workers and maximum number of read requests in advance (readahead mechanism) is set to 16.
- the timeout for each wave is 150ms.
- it is intended to use at most 50% extra of the expected bandwidth in each read operation.

Write related options

Option	Description
-o sfschunkserverwriteto=MSEC	set chunkserver response timeout during write operation in milliseconds (default: 5000).
-o sfswritecachesize=N	define size of write cache in MiB (default: 50).
-o sfscacheperinodepercentage=P	define what part of the write cache non occupied by other inodes can a single inode occupy (in %, default: 25).
-o sfswriteworkers=N	define number of write workers (default: 10).
-o sfswritewindowsize=N	define write window size (in blocks) for each chunk (default: 15).

Option	Description
-o sfsignoreflush=0|1	Advanced: use with caution. Ignore flush usual behavior by replying SUCCESS to it immediately. Targets fast creation of small files, but may cause data loss during crashes (default: 0).
-o sfsuseinodebasedwritealgorithm=0|1	use inode based write algorithm when set to 1; use chunk based write algorithm when set to 0 (default: 0).
-o sfschunkserverwavewriteto=MSEC	set timeout for executing each wave of a write operation in milliseconds (default: 50).

Much like the `sfschunkserverconnectreadto` option, the `sfschunkserverwriteto` sets the timeout for the connection with a chunkserver at the start of a write operation. A write operation is analogous to a read operation: it is the sequence of pending updates of data inside the same chunk. The write requests are always stored in memory and processed by a pool of threads, the size of this buffer is determined by the `sfswritecachesize` parameter and the number of threads in the processing pool is determined by `sfswriteworkers` option. The `sfscacheperinodepercentage` option caps the amount of data regarding a single file which can be in the write buffer at a time. The `sfswritewindowsize` is the number of blocks each write worker will try to process at the same time.

The `sfsignoreflush` option disables the flush syscall, which is typically invoked at the end of a write sequence to ensure that data is successfully written to disk. By disabling this syscall, the additional layer of confirmation is removed. However, this may cause certain file copying applications, which normally copy files sequentially, to instead perform these operations in parallel behind the scenes.

The `sfsignoreutimensupdate` option disables the update of timestamps during file creation and writing operations. This can improve performance for operations involving the creation, copying, or moving of small files by avoiding the overhead of updating timestamp metadata. However, it will result in the loss of accurate timestamp information for these files. Use this option with caution, as it may impact applications that rely on precise timestamp metadata.

`SaunaFS v5.0.1-1` introduces a chunk-based write algorithm as one of its major features. The `sfsuseinodebasedwritealgorithm` option ensures by default (0) that chunk-based write

algorithm is used in all write operations. In case you would like to set the inode-based write algorithm used before `5.0.1-1` version, you should set `sfsuseinodebasedwritealgorithm=1`. It is recommended to use the chunk-based one as it improves writing performance on most cases.

Example

```
sfsmount -H 192.168.1.158 -P 9421 -D S -o sfschunkserverwriteto=3000 -o
sfswritecachesize=1024 -o sfscacheperinodepercentage=100 -o sfswriteworkers=30 -
o sfsignoreflush=1
```

Previous command would start a regular mount on S:, with the parameters:

- the timeout for the connection with the chunkservers during writes is set to 3s.
- the size of the write buffer is 1GB.
- the write operations on any file could take over the whole write buffer.
- the number of write workers is set to 30.
- the flushes are ignored.

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.

Option	Description
-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. The mounted drives behave like SYSTEM user drives. 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

```
sfsmount -o sfsschedulestartmount=ProjectS -H 192.168.1.158 -P 9421 -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

```
sfsmount -H 192.168.1.158 -P 9421 -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

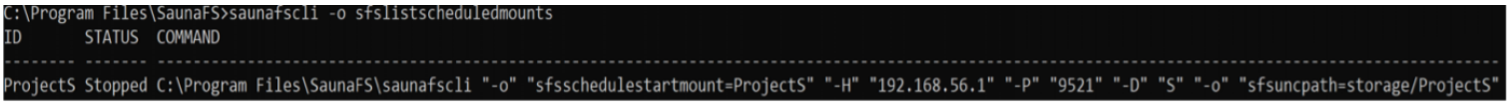
```
sfsmount -o sfsstopscheduledmount=ProjectS
```

will stop the already running mount.

Example 4

```
sfsmount -o sfslistscheduledmounts
```

if following the previous commands should show following:



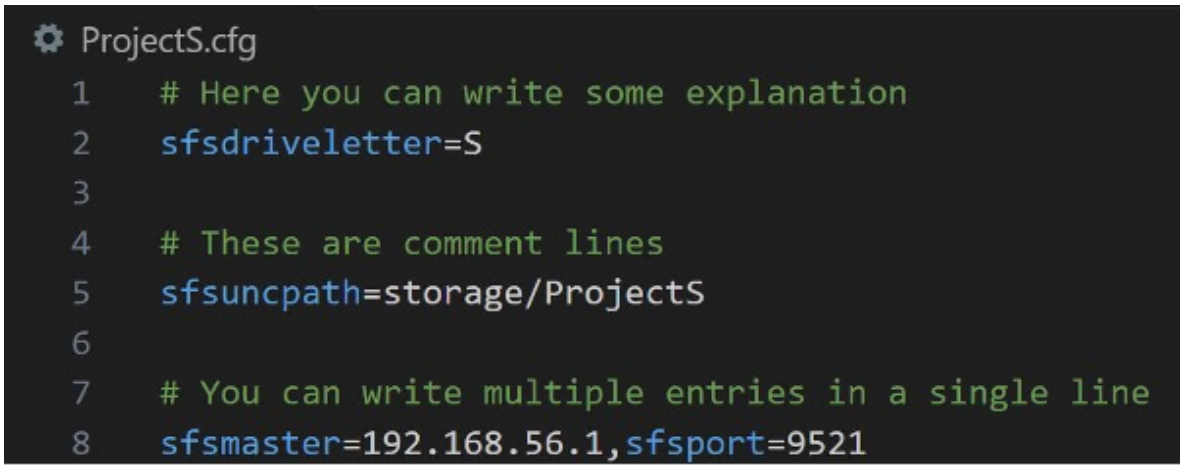
Configuration file option

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

Example

```
sfsmount -c C:\\ProjectS.cfg
```

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



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.

Further examples

These last examples are present to show some other ways to write arguments for the mount command line. The following commands are equivalent (note it is a merge of some of the previous commands):

```
sfsmount -H 192.168.1.158 -P 9421 -D S -S /data/ProjectS -o
cacheexpirationtime=10000 -o readaheadmaxwindowsize=32768 -o readworkers=16 -o
maxreadaheadrequests=16 -o sfschunkserverwavereadto=150 -o bandwidthoveruse=0.5
--uid 999 --gid 999 -o sfsumaskfile=111 -o sfsumaskdir=023
```

```
sfsmount -H 192.168.1.158 -P 9421 -D S -S /data/ProjectS -o
cacheexpirationtime=10000,readaheadmaxwindowsize=32768,readworkers=16 -o
maxreadaheadrequests=16,sfschunkserverwavereadto=150 -obandwidthoveruse=0.5 --
uid 999 --gid 999 -o sfsumaskfile=111,sfsumaskdir=023
```

```
sfsmount -H 192.168.1.158 -P 9421 -D S -S /data/ProjectS -o
CacheExpirationTime=10000 -o ReadaheadMaxWindowSize=32768 -o ReadWorkers=16 -o
MaxReadaheadRequests=16 -o SFSChunkserverWaveReadTO=150 -o BandwidthOveruse=0.5
--UiD 999 --gId 999 -o SfSuMaSkFiLe=111 -o sfsumaskdir=023
```

```
sfsmount -H 192.168.1.158 -P 9421 -D S -S /data/ProjectS -o
CacheExpirationTime=10000 -o
ReadaheadMaxWindowSize=32768,ReadWorkers=16,MaxReadaheadRequests=16 -o
```

```
SFSChunkserverWaveReadTO=150,BandwidthOveruse=0.5 --UiD 999 --gId 999 -o
SfSuMaSkFiLe=111 -osfsumaskdir=023
```

In other words, the "-o" particle could be disposed and replaced by a comma, and the arguments that should be preceded by it can be joined. Also, the space in between the "-o" and the option itself could be removed, if the option is not a sequence of options comma separated. The names of the options can be written using any case, note it does not apply to:

- keywords ("-H", "-P", "-D")
- the "-o" particle
- the parameters ("/data/ProjectS")

Both of these advanced manners to write options cann be merged, such as the last example. Any command line that does not suffice the expected pattern will display an error message with the wrong/misspelled option, and prevent the mount start.

Other options

Option	Description
-o sfscleanacquiredfilesperiod=SEC	defines the interval, in seconds, at which the checker function runs to release unused acquired files. (default: 0).
-o sfscleanacquiredfilestimeout=SEC	defines the lifetime limit, in seconds, for an acquired file. If a file is not used within this time frame, it is considered unusedand eligible for release. (default: 0).
-o sfsloglevel=LEVEL	set logging level to LEVEL. Levels are: trace, debug, info, warn, err, critical, off. (default: warn).
-o sfslogflushlevel=LEVEL	set level of the automatic flush of the logs to LEVEL (default: err). Log levels are the same of `sfsloglevel` option.

Option	Description
-o sfsignoreutimensupdate=0|1	Advanced: use with caution. Ignore timestamps updates during file creation and writing operations. Targets fast creation of small files, but causes timestamp metadata loss (default: 0).

The previously shown options are some of our actual "Other options" batch. Remaining ones may be present in other parts of this documentation or in some FAQ.

Side note

The descriptions of previous options contain the default and extra comments. The default description appears on the help command response

```
sfsmount -—help
```

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

```
sfsmount -V
```

displays the version info.

SFSMount Manager

SFSMount Manager is a command line application that is used for monitoring latest statuses from Windows client instances on the same Windows machine environment.

This tool provides as main functionalities listing and disconnecting current Windows client instances with some additional options.

When executing complete SaunaFS installation on Windows OS, this binary is also added as part of it.

Usage

Option	Description
`--list`	List client instances with current statuses.
`--help`	Show help information.
`--disconnect <ID>`	Disconnect a specific client instance identified by `<ID>`.
`--disconnect --all`	Disconnect all client instances.
`--disconnect --invalid`	Disconnect invalid client instances (clients not connected to a master server).

These are current possible actions that can be performed by the SFSMount Manager. All of these options request info from status files generated by Windows client instances. These files contains following info:

- `MOUNTID`: The clients mount point identifier, which has format `<drive>:<mountpoint>`.
- `PID`: Denotes the process id associated with the client process. It is used for stopping client when needed.
- `MDS_ON`: Has value 'YES' if metadata server associated to client is connected, 'NO' otherwise.
- `MDS`: Metadata server ip address.
- `SCHEDULED`: Has value 'YES' if client process is scheduled, 'NO' otherwise.
- `MOUNTPOINT`: Represents client's mountpoint path.
- `TIME`: To represent time elapsed in milliseconds since the epoch .
- `STATUS`: This value can be of four different forms:
  - `INVALID`: When client is running but no master server is connected to it.
  - `SCHEDULED`: Client has been scheduled
  - `DISCONNECTED`: Client has been disconnected more than 5 seconds since last time its status was updated on its corresponding status file.

- `WORKING`: When client is running and none of above cases happend.

Example 1

```
.\sfsmount -H 192.168.43.180 -D S
```

```
.\sfsmount -H 192.168.43.180 -D T
```

After mounting two separate client instances we can list them and receive the corresponding info associated with them:

```
PS C:\Program Files\SaunaFS> .\sfsmount-manager.exe --list
Listing connected devices...
MOUNTID:S, PID: 22416, MDS_ON: Yes, MDS: 192.168.43.180, SCHEDULED: No, MOUNTPOINT: S/, Time: 1719890945061 ms STATUS: WORKING
MOUNTID:T, PID: 2164, MDS_ON: Yes, MDS: 192.168.43.180, SCHEDULED: No, MOUNTPOINT: T/, Time: 1719890945091 ms STATUS: WORKING
```

Example 2

```
PS C:\Program Files\SaunaFS> .\sfsmount-manager.exe --disconnect 22416
Disconnecting device with ID 22416...
```

If we try to disconnect a client process by `ID` and considering there were the same two instances from last example running, then the result should be:

```
PS C:\Program Files\SaunaFS> .\sfsmount-manager.exe --list
Listing connected devices...
MOUNTID:T, PID: 2164, MDS_ON: Yes, MDS: 192.168.43.180, SCHEDULED: No, MOUNTPOINT: T/, Time: 1719891070181 ms STATUS: WORKING
```

Example 3

```
PS C:\Program Files\SaunaFS> .\sfsmount-manager.exe --disconnect --all
Disconnecting all connections...
```

It also possible to disconnect all current client processes and considering there is the same instance from last example running, then the result should be:

```
PS C:\Program Files\SaunaFS> .\sfsmount-manager.exe --list
Listing connected devices...
```

SaunaFS Admin Tool

The SaunaFS Admin Tool (saunafs-admin.exe) is a command-line utility designed for monitoring, managing, and administering various aspects of a SaunaFS installation. It provides a unified interface for inspecting system health, managing server components, controlling system behavior, and retrieving operational statistics. Suitable for both routine diagnostics and advanced administrative tasks.

Usage

`saunafs-admin COMMAND [OPTIONS...] [ARGUMENTS...]`

Command	Arguments	Description	Extra Options
chunks-health	`<master ip>` `<master port>`	Returns chunks health reports in the installation. By default, all reports are shown. In replication and deletion states, the column means the number of chunks with number of copies specified in the label to replicate/delete.	• `--porcelain`: Make the output parsing-friendly. • `--availability`: Print report about availability of chunks. • `--replication`: Print report about number of chunks that need replication. • `--deletion`: Print report about number of chunks that need deletion.
info	`<master ip>` `<master port>`	Prints statistics concerning the SaunaFS installation.	• `--porcelain`: Make the output parsing-friendly.

Command	Arguments	Description	Extra Options
iolimits-status	`<master ip>` `<master port>`	Prints current configuration of global I/O limiting.	• `--porcelain`: Make the output parsing-friendly.
list-chunkservers	`<master ip>` `<master port>`	Prints information about all connected chunkservers.	• `--porcelain`: Make the output parsing-friendly.
list-defective-files	`<master ip>` `<master port>`	Lists files which currently have defective chunks.	• `--porcelain`: Make the output parsing-friendly. • `--unavailable`: Print unavailable files. • `--undergoal`: Print files with undergoal chunks. • `--structure-error`: Print files/directories with structure error. • `--limit=`: Limit maximum number of printed files.
list-disks	`<master ip>` `<master port>`	Prints information about all connected chunkservers.	• `--porcelain`: Make the output parsing-friendly. • `--verbose`: Show operations

Command	Arguments	Description	Extra Options
			statistics.
list-disk-groups	`<master ip>` `<master port>`	Prints disk groups configuration in chunkservers.	
list-goals	`<master ip>` `<master port>`	List goal definitions.	• `--porcelain`: Make the output parsing-friendly. • `--pretty`: Print nice table.
list-mounts	`<master ip>` `<master port>`	Prints information about all connected mounts.	• `--porcelain`: Make the output parsing-friendly. • `--verbose`: Show goal and trash time limits.
list-metadataservers	`<master ip>` `<master port>`	Prints status of active metadata servers.	• `--porcelain`: Make the output parsing-friendly.
list-tasks	`<master ip>` `<master port>`	Lists tasks which are currently executed by master.	
manage-locks	`<master ip>` `<master port>` `[list/unlock]` `[flock/posix/all]`	Manages file locks.	• `--porcelain`: Make the output parsing-friendly. • `--active`: Print only active locks.

Command	Arguments	Description	Extra Options
			• `--pending`: Print only pending locks. • `--inode=`: Specify an inode for operation. • `--owner=`: Specify an owner for operation. • `--sessionid=`: Specify a sessionid for operation. • `--start=`: Specify a range start for operation. • `--end=`: Specify a range end for operation.
metadataserver-status	`<master ip>` `<master port>`	Prints status of a master or shadow master server.	• `--porcelain`: Make the output parsing-friendly.
ready-chunkservers-count	`<master ip>` `<master port>`	Prints number of chunkservers ready to be written to.	
promote-shadow	`<shadow ip>` `<shadow port>`	Promotes metadata server. Works only if personality 'ha-cluster-managed' is	

Command	Arguments	Description	Extra Options
		used. Authentication with the admin password is required.	
stop-master-without-saving-metadata	`<master ip>` `<master port>`	Stop the master server without saving metadata in the metadata.sfs file. Used to quickly migrate a metadata server (works for all personalities). Authentication with the admin password is required.	
reload-config	`<master ip>` `<master port>`	Requests reloading configuration from the config file. This is synchronous (waits for reload to finish). Authentication with the admin password is required.	
save-metadata	`<metadataserver ip>` `<metadataserver port>`	Requests saving the current state of metadata into the metadata.sfs file. With `--async`, fails if the process cannot be started (e.g., already in progress). Without `--async`, fails if the process cannot be started or	• `--async`: Don't wait for the task to finish.

Command	Arguments	Description	Extra Options
		if it finishes with an error. Authentication with the admin password is required.	
stop-task	`<master ip>` `<master port>` `<task id>`	Stop execution of task with the given id.	
list-sessions	`<master ip>` `<master port>`	Lists all currently open sessions.	
delete-session	`<master ip>` `<master port>` `<session_id>`	Deletes the specified session.	
dump-config	`<master ip>` `<master port>`	Dumps the configuration files of the master server. Authentication with the admin password is required.	• `--defaults`: Return default values as well. This is informational and may not be correct in all cases.

Example 1

Assuming that we have a master server running at IP address 192.168.1.158 and listening for client communication requests on port 9421, let's say we also have a client mounted with the following command:

```
C:\> sfsmount -H 192.168.1.158 -D T
```

To check the number of active sessions, you can use the `saunafs-admin` command to request this information from the master server:

```
C:\> saunafs-admin list-sessions 192.168.1.158 9421
```

The output should look like this:

```
Session ID: 5
IP: 192.168.1.158
Open files: 0
```

This output shows the list of active sessions, which in this case is one because we only started one client. This output includes the session ID assigned by the master server to identify each client, the client's IP address, and the number of currently open files in that session.

Example 2

To check the list of currently mounted drives, you can use the `saunafs-admin` command to request this information from the master server:

```
C:\> saunafs-admin list-mounts 192.168.1.158 9421
```

The output should look like this:

```
session 5:
        ip: 192.168.1.158
        mount point: T:
        version: 4.10.0
        root dir: /
        root uid: 0
        root gid: 0
        users uid: 999
        users gid: 999
        read only: no
        restricted ip: yes
        ignore gid: no
        all can change quota: no
        map all users: no
```

This output provides detailed information about the mounted session, including the session ID, client IP address, mount point (drive letter), client version, root directory, user and group IDs,

read-only status, IP restrictions, and other mount-related options.

SaunaFS Command Tool

The SaunaFS Command Tool is an open prompt that can be used in a SaunaFS mounted file system in order to perform SaunaFS-specific operations.

Usage

```
saunafs <tool name> [options]
```

SaunaFS Tools Overview

Tool Name	Extra options	Description
getgoal	[-nhHr] name [name ...]	Get objects goal (desired number of copies).
setgoal	GOAL name [name ...]	Set objects goal (desired number of copies).
gettrashtime	[-nhHr] name [name ...]	Get objects trashtime (how many seconds file should be left in trash).
settrashtime	[-nhHr] SECONDS[-|+] name [name ...]	Set objects trashtime (how many seconds file should be left in trash).
geteattr	[-nhHr] name [name ...]	Get objects extra attributes.
seteattr	[-nhHr] -f attrname [-f attrname ...] name [name ...]	Set objects extra attributes.
deleattr	[-nhHr] -f attrname [-f attrname ...] name [name ...]	Delete objects extra attributes.
checkfile	[-nhH] name [name ...]	Checks and prints number of chunks and number of chunk copies belonging

Tool Name	Extra options	Description
		to specified file(s).
fileinfo	name [name ...]	Show files info (shows detailed info of each file chunk).
appendchunks	dstfile name [name ...]	Append file chunks to another file. If destination file doesn't exist then it's created as empty file and then chunks are appended.
dirinfo	[-nhH] name [name ...]	Show directories stats.
filerepair	[-nhHc] name [name ...]	Repair given file. Use it with caution. It forces file to be readable, so it could erase (fill with zeros) file when chunkservers are not currently connected.
makesnapshot	[-of] src [src ...] dst	Makes a "real" snapshot (lazy copy, like in case of *appendchunks*) of some object(s) or subtree (similarly to *cp -r* command).
repquota	[-nhH] (-u <uid>|-g <gid>)+ <mountpoint-root-path>; [-nhH] -a <mountpoint-root-path>; [-nhH] -d <directory-path>	Summarize quotas for a user/group/inode or all users and groups.
setquota	(-u <uid>|-g <gid> |-d) <soft-limit-size> <hard-limit-size> <soft-limit-inodes> <hard-limit-inodes> <directory-path>	Set quotas for a user/group/inode.
rremove	name [name ...]	Recursive remove.

Tool Name	Extra options	Description
help	[tool]	Can be used for getting usage description for an specific tool name.

Extra info for tool commands extra options

Tool Name Extra Option	Description
-n	Show numbers in plain format.
-h	Show "human-readable" numbers using base 2 prefixes (IEC 60027).
-H	Show "human-readable" numbers using base 10 prefixes (SI).
-r	Do it recursively.
-GOAL	Set goal to given goal name.
SECONDS+	If trashtime is smaller than given value, increase trashtime to given value.
SECONDS-	if trashtime is bigger than given value, decrease trashtime to given value.
SECONDS	Just set trashtime to given value.
-f attrname	Specify attribute to set.
-c	Restore to previous version if applicable, never erase.
-o, -f	Allow to overwrite existing objects.

Deprecated Tools

Deprecated Tool	Equivalent	Description
rgetgoal	getgoal -r	Recursive version of getgoal tool.
rsetgoal	setgoal -r	Recursive version of setgoal tool.
rgettrashtime	gettrashtime -r	Recursive version of gettrashtime tool.
rsettrashtime	settrashtime -r	Recursive version of settrashtime tool.

To execute its operations, this tool, when run in a SaunaFS filesystem, connects to the corresponding master server and requests the required information accordingly. If executed outside a SaunaFS-mounted filesystem, it will retry communicating with the master server until a 15-second timeout is reached.

For the following examples, let's assume there is a default mounted client with the following command:

```
sfsmount -H 192.168.1.158 -P 9421 -D S
```

This command creates a drive mounted on the letter S: with the SaunaFS filesystem. The IP address 192.168.1.158 must correspond to a SaunaFS master server that is actively listening for incoming client connections on port 9421.

Example 1

It is possible to use the `saunafs` command for requesting the corresponding goal for an specific file:

```
S:\> saunafs getgoal .\file1.txt
```

Should produce the following output specifying the file and its goal:

```
.\file1.txt: 1
```

Example 2

It is also possible to make snapshots of existing files and folders:

```
S:\> saunafs makesnapshot .\file1.txt .\file1_snapshot.txt
```

Which should produce the following output:

```
Creating snapshot: .\file1.txt -> S:\/file1_snapshot.txt ...
Snapshot .\file1.txt -> S:\/file1_snapshot.txt completed
```

In the last case, the snapshot should produce a file with the same size and chunks than the source file or folder. This can be checked by using the fileinfo functionality from the `saunafs` command:

```
S:\> saunafs fileinfo .\file1.txt
```

```
.\file1.txt:
        chunk 0: 00000000000000C6_00000002 / (id:198 ver:2)
                copy 1: 192.168.1.158:9422:_
```

```
S:\> saunafs fileinfo .\file1_snapshot.txt
```

```
.\file1_snapshot.txt:
        chunk 0: 00000000000000C6_00000002 / (id:198 ver:2)
                copy 1: 192.168.1.158:9422:_
```

Example 3

As mentioned before, it is not possible to use the `saunafs` command on a file system outside SaunaFS. This could lead the client to wait for communicating to master, which never happens and should output the following output after *15* seconds:

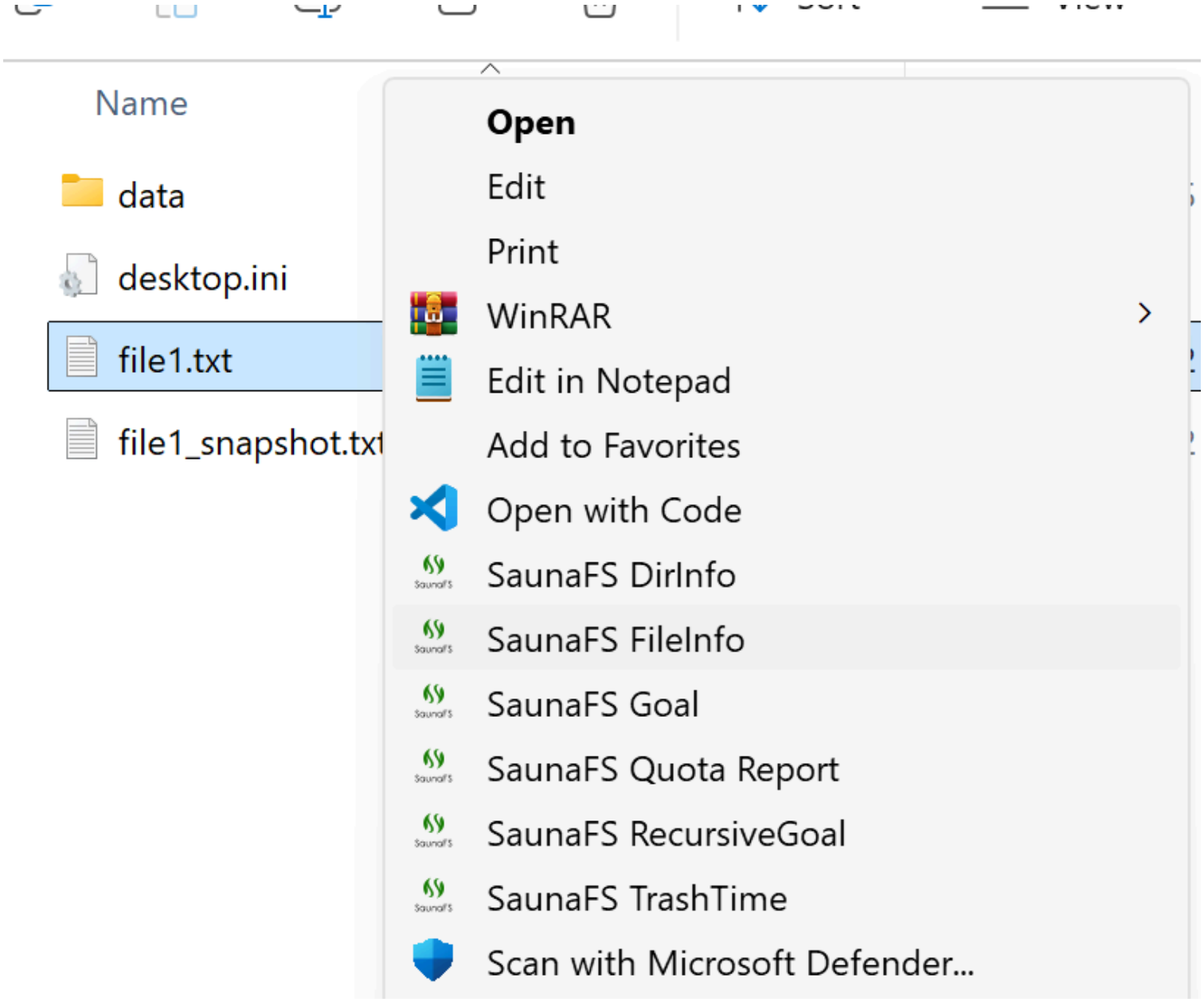
```
.: exceeded master connection max retries: not SaunaFS object
```

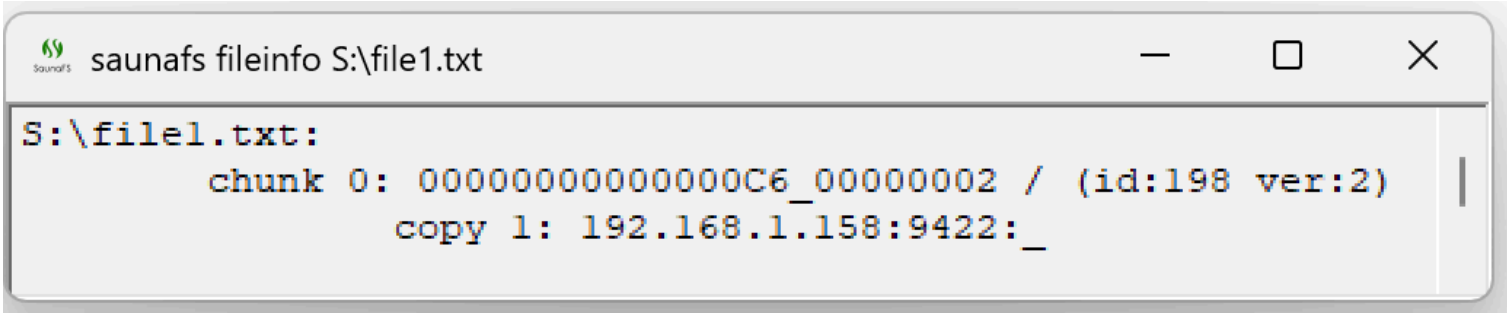
SaunaFS Right-click Context Menu

The SaunaFS rick-click context menu options is another way to use the functionalities from the `saunafs` command with the context menu that is display on Windows when right-clicking a file, folder, drive, or folder background.

Example 1

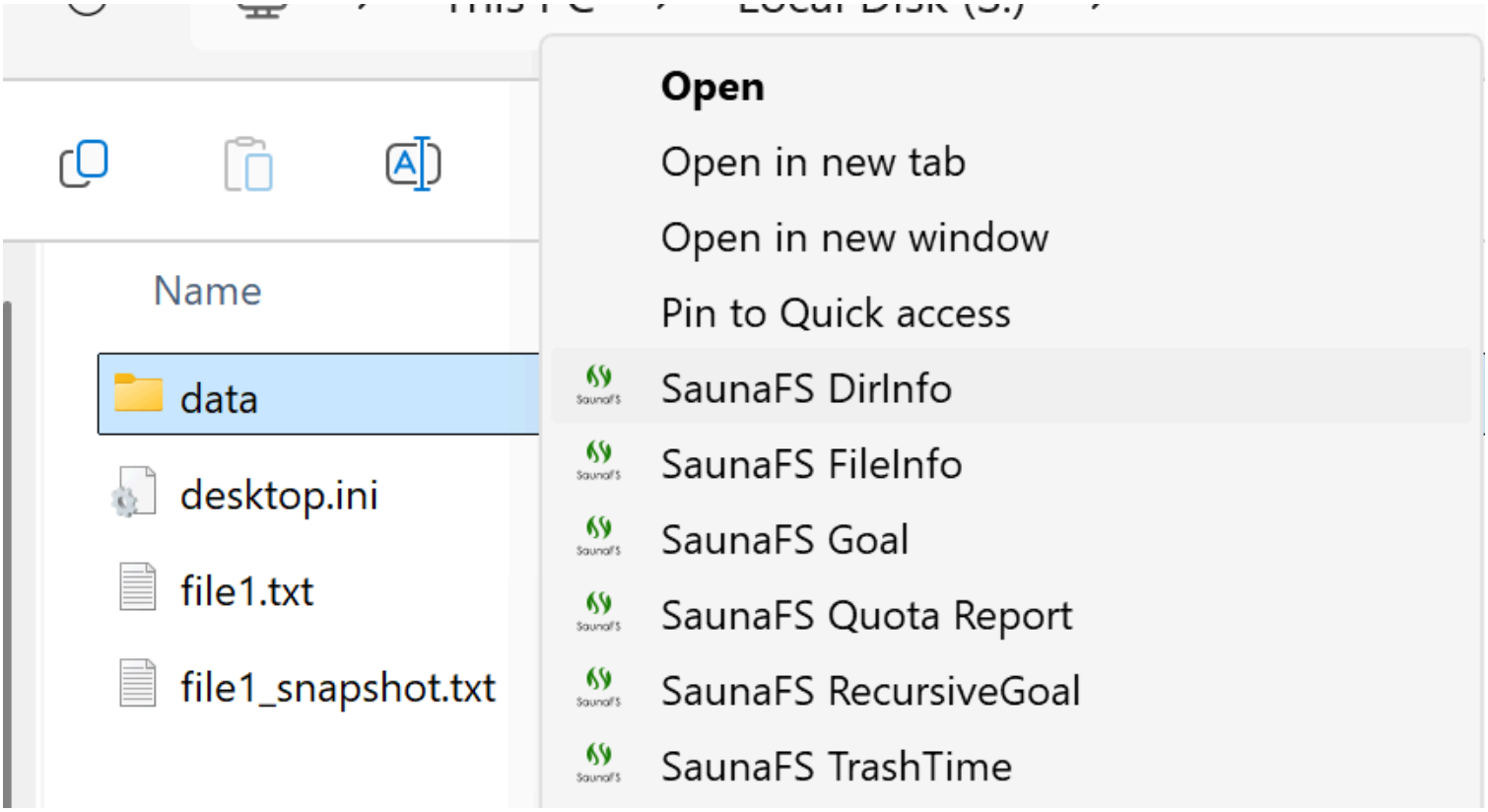
When right-clicking on a file, it could display for instance, the corresponding `fileinfo` result.

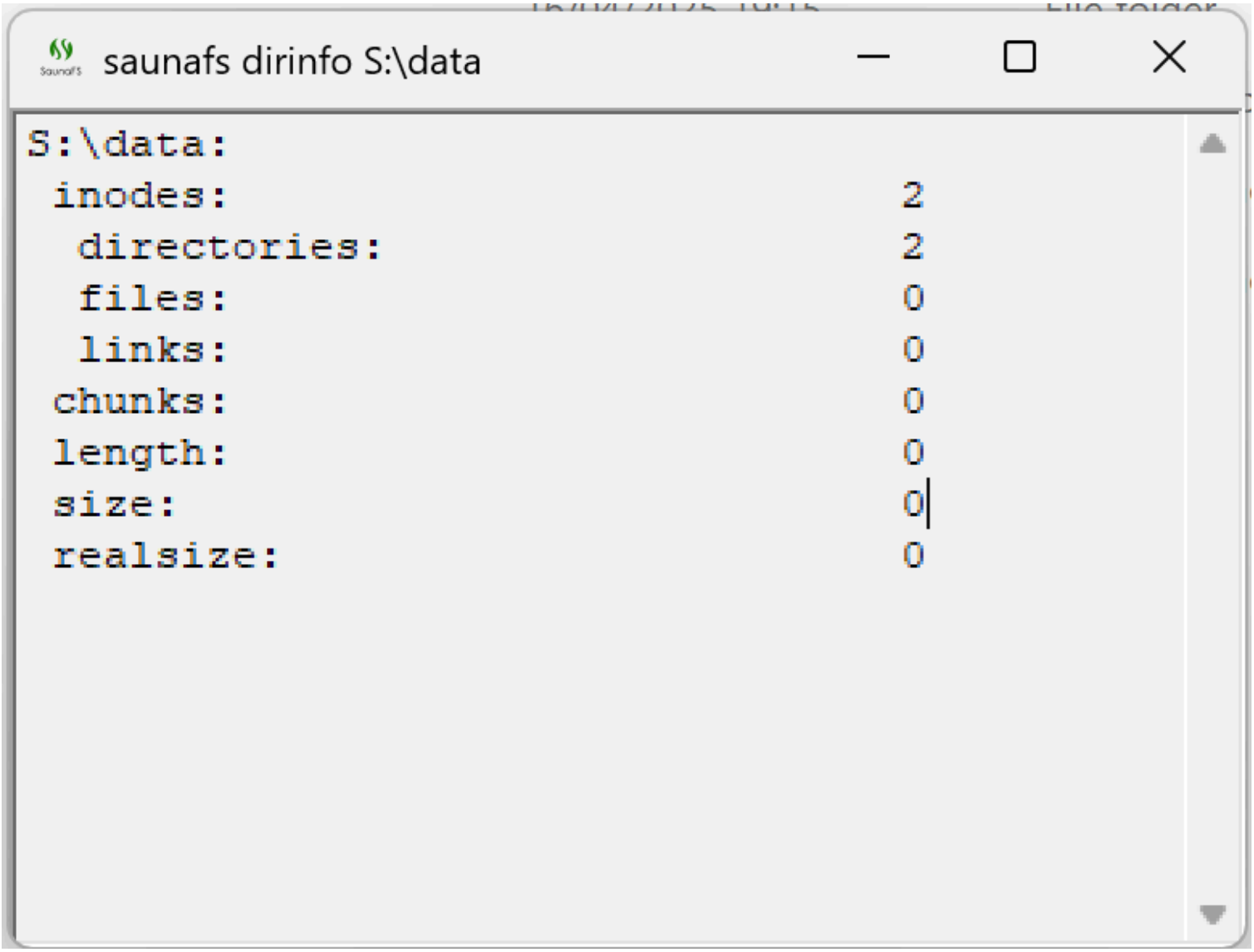




Example 2

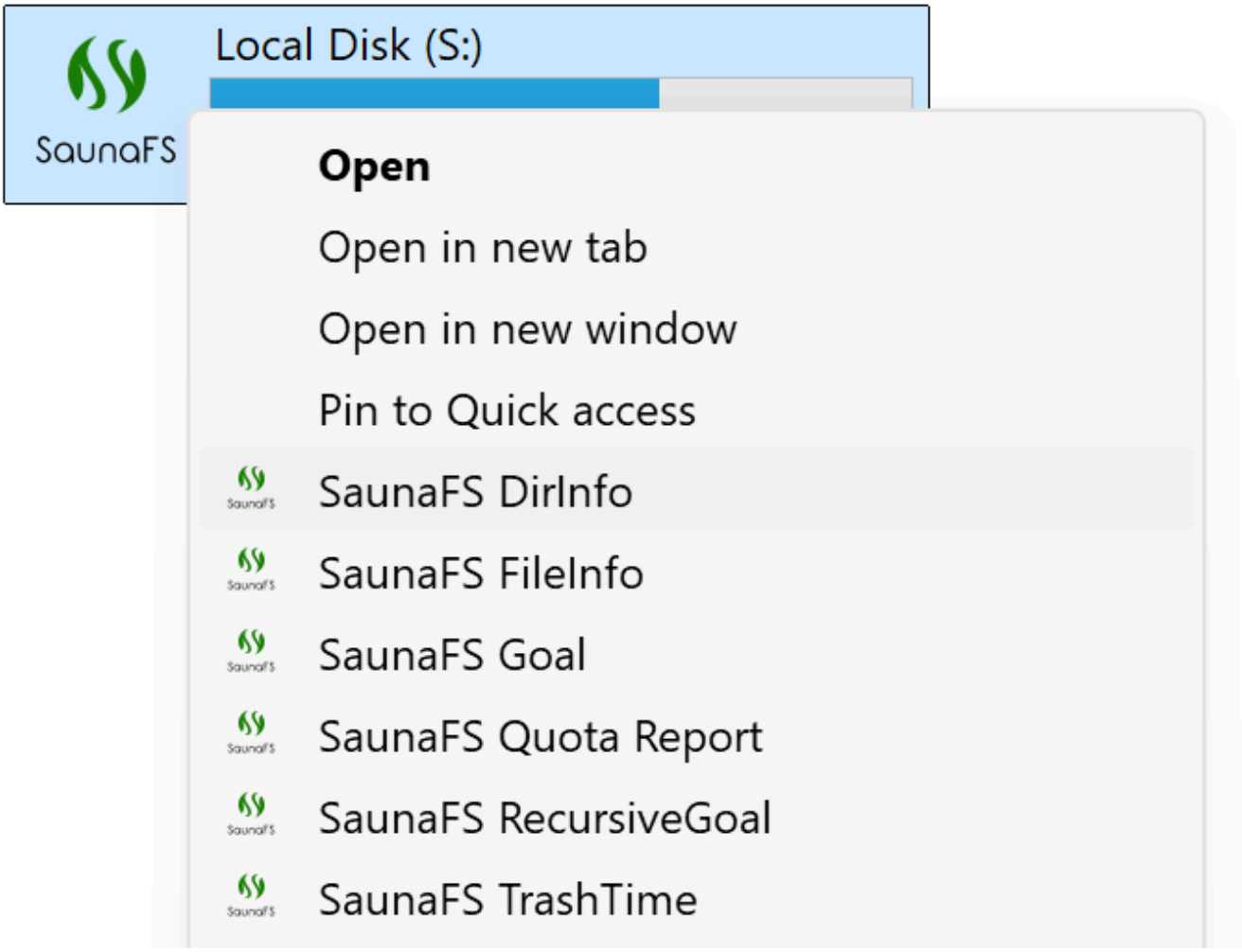
It is also possible to display the `dirinfo` of a folder by using this feature.

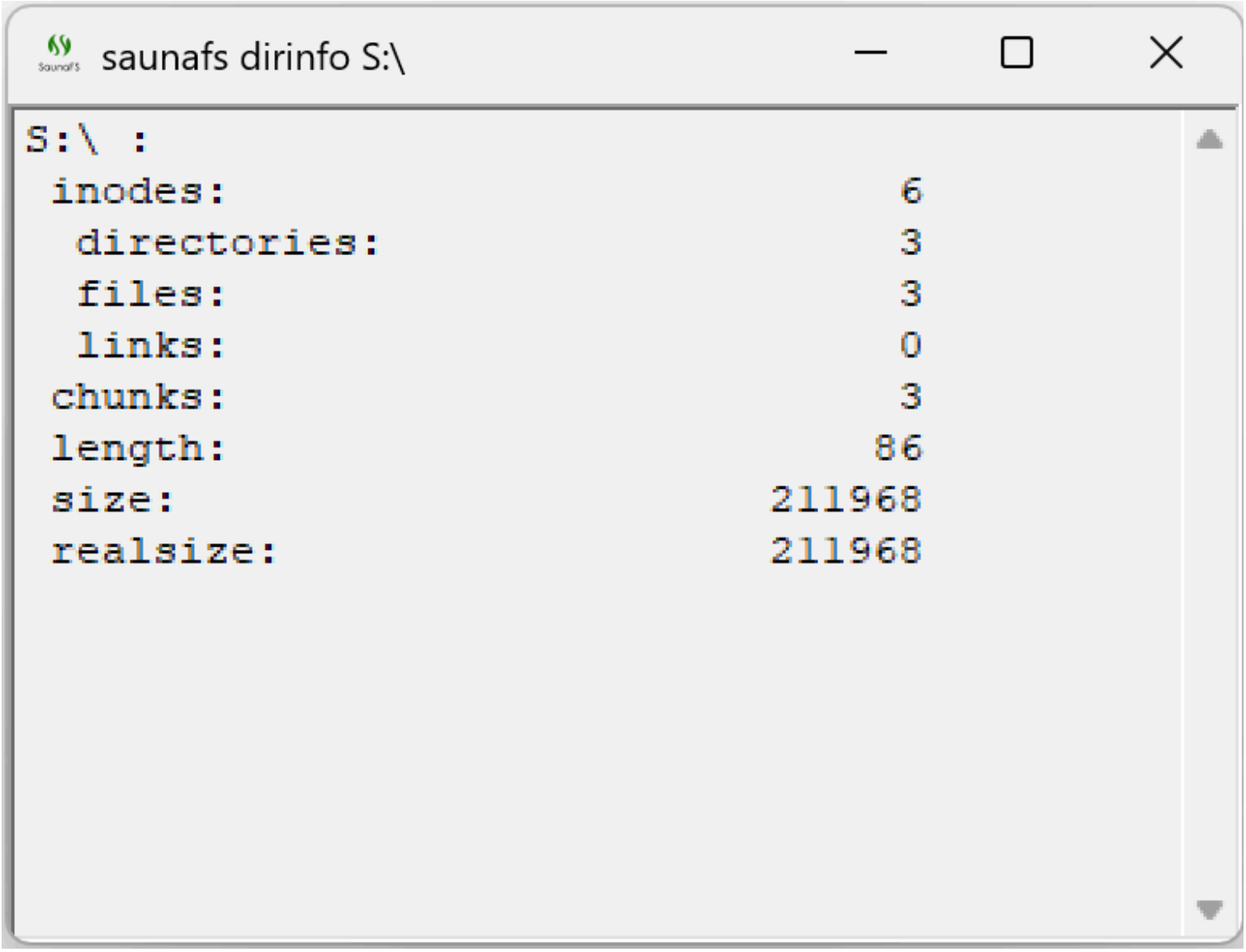




Example 3

This is an example on how to get the `dirinfo` of a mounted SaunaFS drive.


SaunaFS
Local Disk (S:)
Open
Open in new tab
Open in new window
Pin to Quick access
SaunaFS DirInfo
SaunaFS FileInfo
SaunaFS Goal
SaunaFS Quota Report
SaunaFS RecursiveGoal
SaunaFS TrashTime



User-wise behavior

It may be troublesome in some cases the expected behavior of the Windows Client when mounted using different users. For instance, consider the case of been able to launch commands from the current user (non-elevated prompt), current user in Admin mode (elevated prompt) and SYSTEM user.

The behavior should be the following:

- a letter mounted by current user would only be visible to the current user (including moving to the mounted path, accessing it through Explorer or else).
- a letter mounted by current user in Admin mode would only be visible to the current user in Admin mode, which means that won't be accesible by regular Explorer and won't be reachable through non-elevated prompt.

- a letter mounted by SYSTEM user would be visible by every user.
- if the current user has already mounted a letter and the SYSTEM user mounts the same one, then the letter would continue displaying the first mount content. If the first mount process ends, then the displayed content would be the SYSTEM user mount.
- if the SYSTEM user has already mounted a letter and the current user tries to mount the same one, you should receive the following error: `Cannot create WinFsp-FUSE file system: mount point in use.`.
- two different users should be able to mount the same letter at the same time in the same machine, unless some other conflict exists.

The mounts using persistent mount options are SYSTEM user ones (as mentioned in its section) though could be run from elevated prompt.

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.

Why the client is failing to mount some subfolder of the storage?

When using `sfssubfolder` option, it needs to be guaranteed that the subfolder represents a valid namespace, i.e, the folder exists.

Otherwise the client will not be able to export the folder. Additionally, the permissions for the folder need to match the `sfsuid` used to start the client.

Summarizing, before exporting a subfolder, the following steps should verified:

1. The subfolder exists, otherwise it needs to be created.
2. Assign correct permissions to the user `UID` (default 1000) provided (parameter `sfsuid=UID`) that will write/read the folder, i.e., update folder permissions using `chown UID`.

3. Start the Windows Client.

Windows Client can also be used to create new subfolders by following the steps below:

1. Start the Windows Client using options `sfsuid=UID` and `sfsgid=GID` to mount the root folder ("/") or an ancestor folder containing the path where the new subfolder will be created. For instance, to create folder "/storage/volume01", the client can mount "/storage" subfolder.
2. Create the desired new folder, for example, "/storage/volume01".
3. Stop the client and start it using option `sfssubfolder=/storage/volume01`.

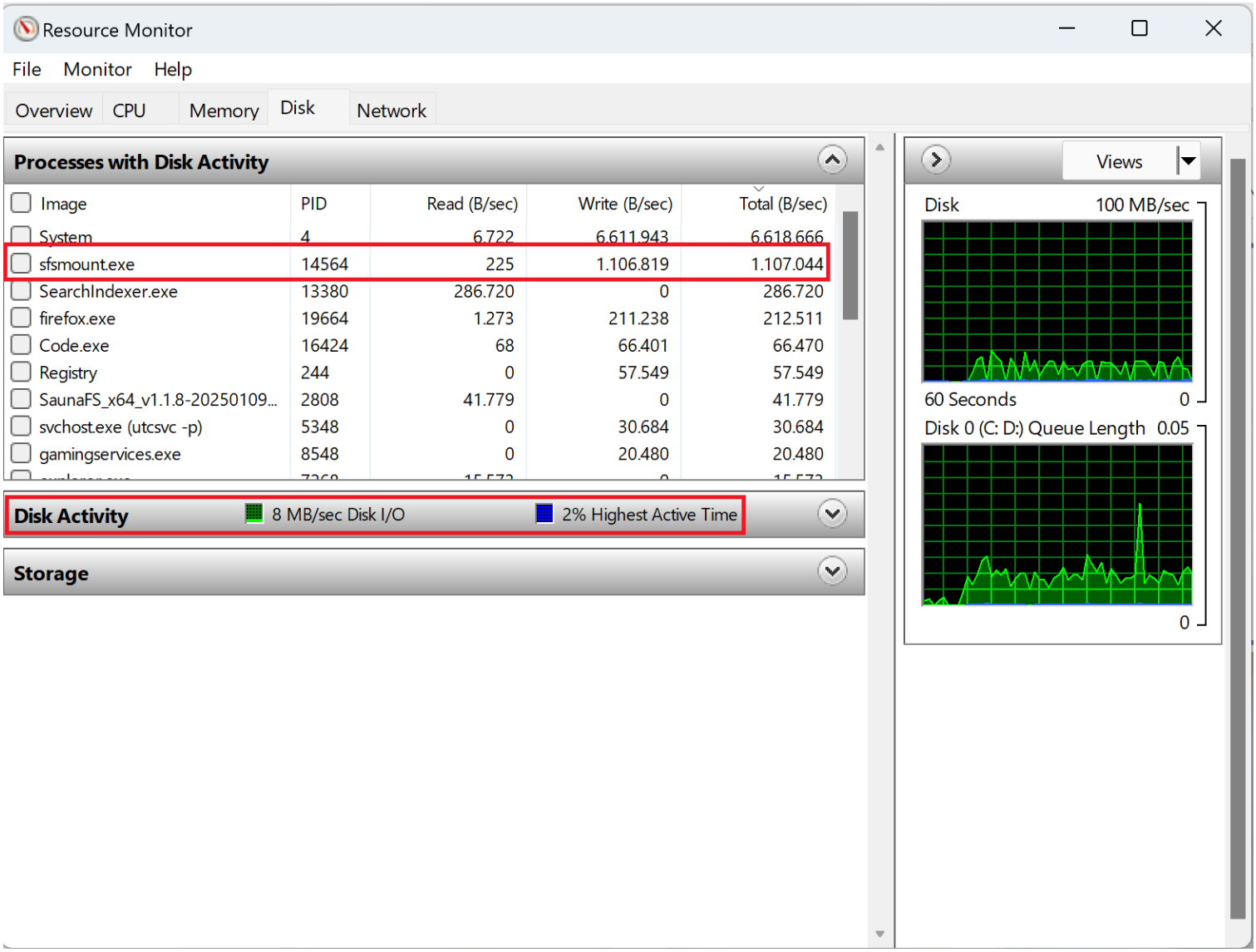
Note: This works if the ancestor folder, i.e., "/storage" folder has write permissions for *other* group.

Why my IIS server is not able to load the data from storage?

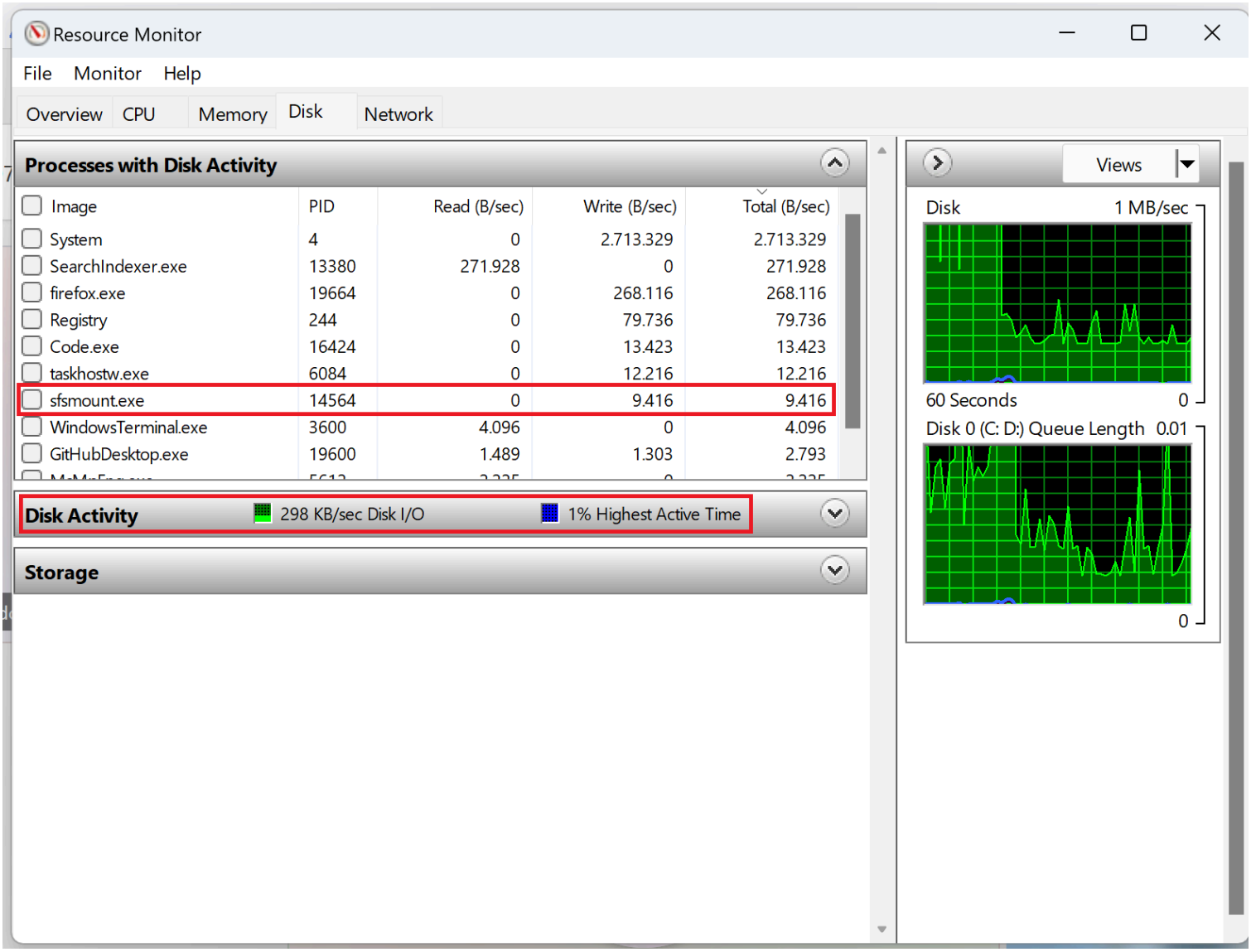
The IIS server performs its queries for data through the Windows Client with their very own specific user. Until now, that user has been translated to the uid 17. To confirm this, please check the output of the ".oplog" hidden file (`type .oplog` at the root of the filesystem) while trying to load the page. That user is from the range of users which are not local (less than 196608) and is not treated like them, which most likely is the cause of the issue. In order to fix the issue, you should provide the `sfsallowedusers=17` option (if IIS user is 17).

Why is the client using considerable disk resources if I am not performing any operation on the mountpoint?

In case you check the Resource Monitor tool on your Windows machine and notice a situation of unexpected disk resources consumption similar to the image below:



You should check if, when the client was mounted, the `enableupdaterstatusthread` option was set like this: `enableupdaterstatusthread=1`. In that case, consider mounting the client again without specifying that option, which is disabled by default. Then you could see a disk usage consumption reduction like this:



The reason behind this situation is that `enableupdaterstatusthread` option enables a thread to periodically update the mount status in a file. Enabling this option can cause unexpected and continuous logging to `C:\\$LogFile` and `:\\$Extend\\$UsnJrnl:$J` files, which may affect performance on machines with lower capabilities.

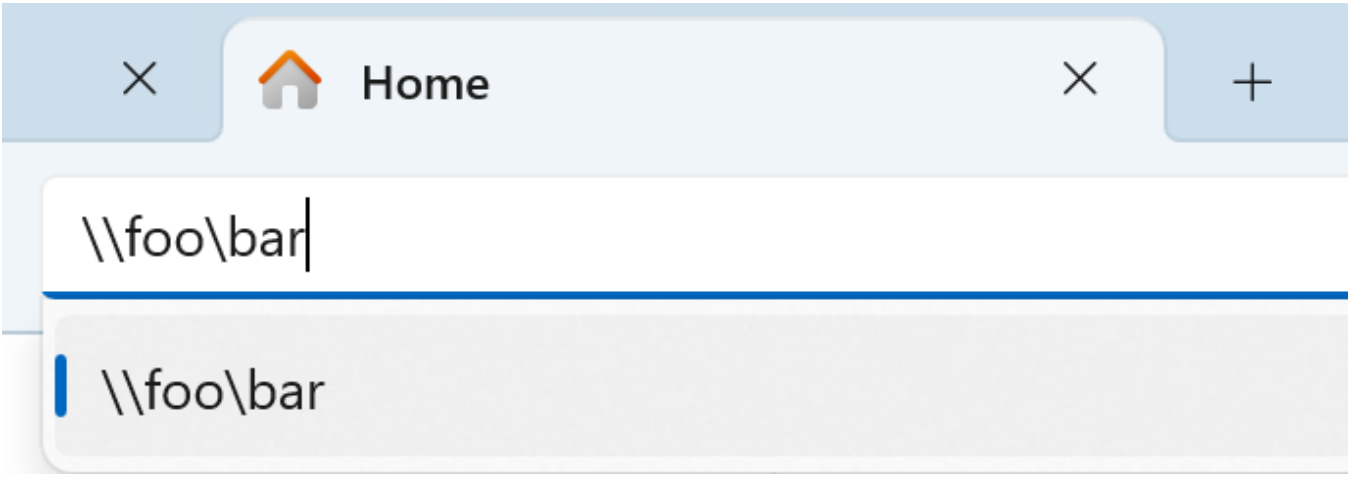
This option is only available from SaunaFS Windows Client version v1.1.8. If you are using an older version, consider updating it for avoiding this issue.

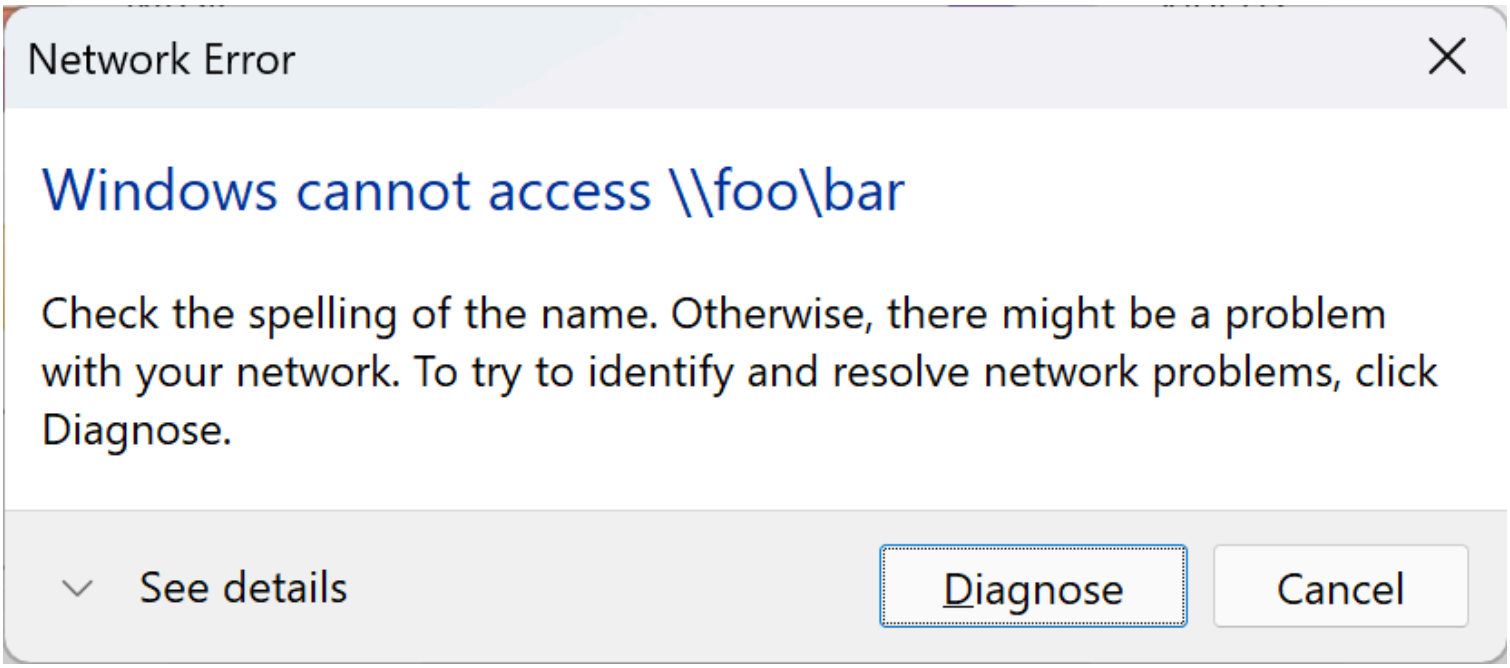
Why is the Windows client UNC path option not working?

If you want to start a client as a network drive using the UNC path feature, you might encounter an issue like this:

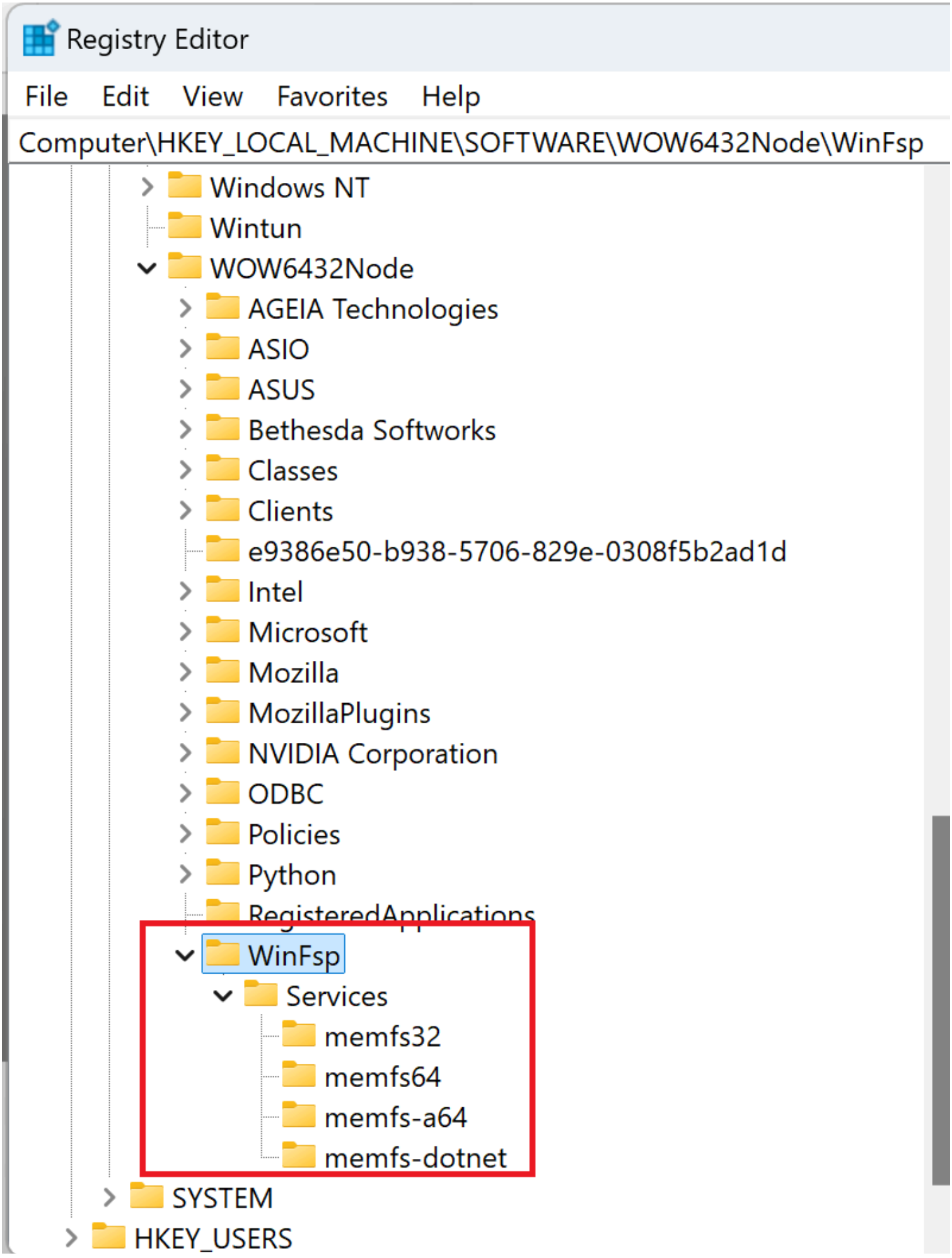
```
PS C:\Program Files\SaunaFS> ./sfsmount.exe -H 192.168.1.158 -D T -o sfsuncpath=foo/bar
Operation failed with error: 5. Message: Access is denied.

Failed to open parent key: 0
sfsmaster accepted connection with parameters: read-write,restricted_ip ; root mapped to 0:0 ; users mapped to local IDs
The service sfsmount has been started.
```

After this, you might find that it is not possible to access the mounted network drive like this:

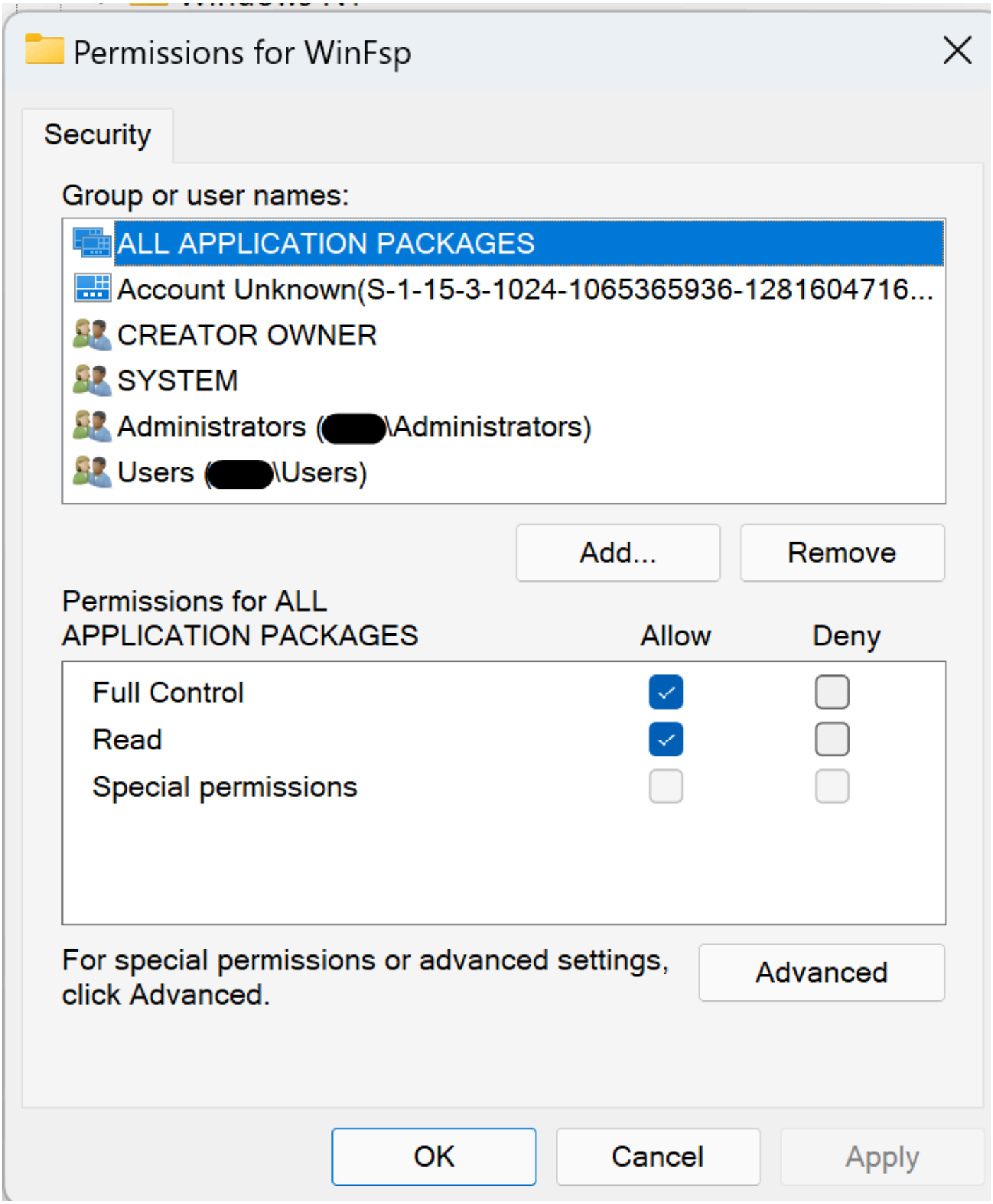




The reason for this situation to happen is that you probably haven't granted enough permissions to the WinFSP registry in the Windows Registry. You can go to the search tab and type "Registry Editor" then you should see something like this:


Registry Editor
File Edit View Favorites Help
Computer\HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\WinFsp
Windows NT
Wintun
WOW6432Node
AGEIA Technologies
ASIO
ASUS
Bethesda Softworks
Classes
Clients
e9386e50-b938-5706-829e-0308f5b2ad1d
Intel
Microsoft
Mozilla
MozillaPlugins
NVIDIA Corporation
ODBC
Policies
Python
WinFsp
Services
memfs32
memfs64
memfs-a64
memfs-dotnet
SYSTEM
HKEY_USERS

You should then right-click on WinFSP and grant full access to the users on the machine where you are trying to mount the client using the UNC path option.



Then, when you try to mount the client with the UNC path feature, the permission error message should be gone:

```
PS C:\Program Files\SaunaFS> ./sfsmount.exe -H 192.168.1.158 -D T -o sfsuncpath=foo/bar
Service registered successfully: foo
sfsmaster accepted connection with parameters: read-write,restricted_ip ; root mapped to 0:0 ; users mapped to local IDs
The service sfsmount has been started.
```

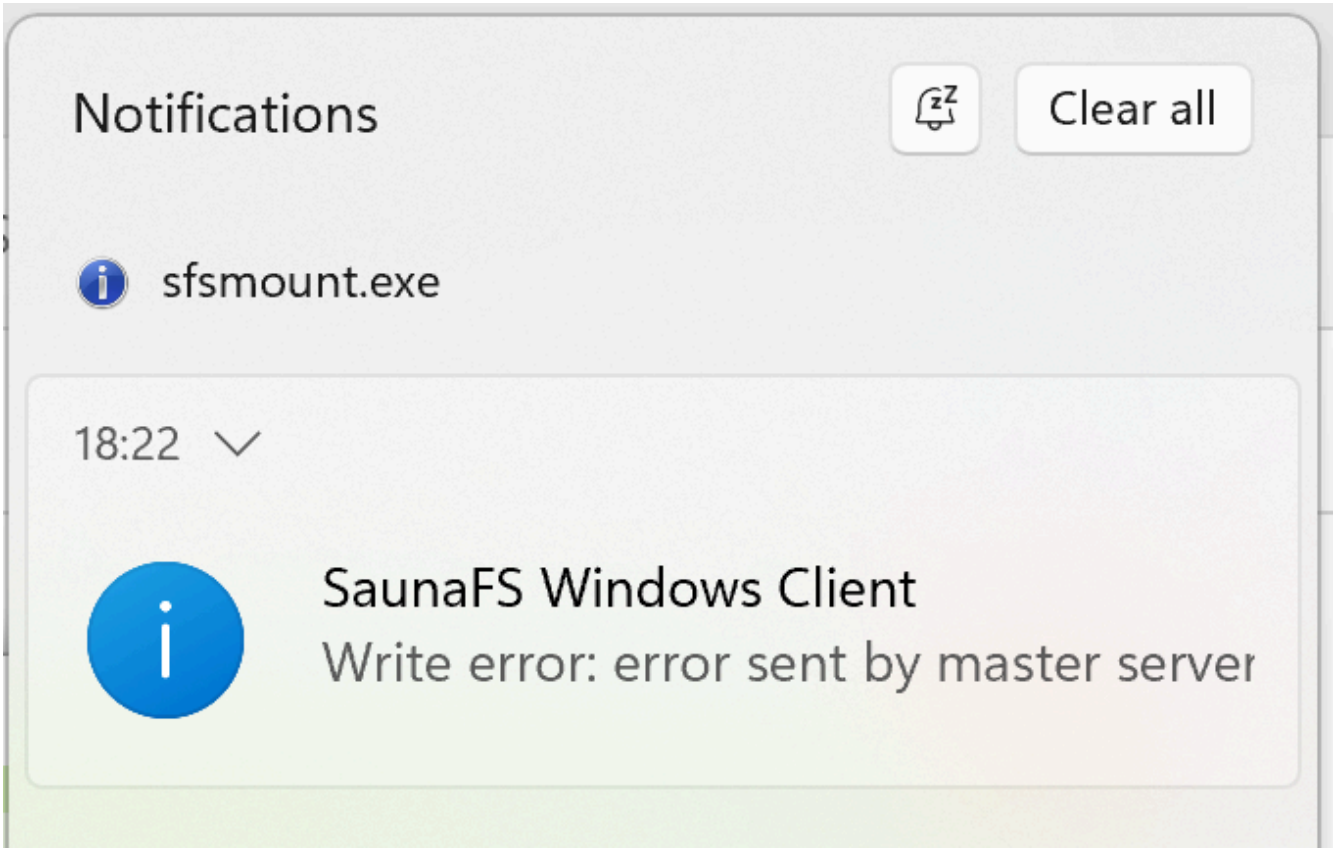
Finally, when you try to check the resolution of the UNC path, it should work as expected, showing the content of the mount point after typing the UNC path in the Windows Explorer bar.

This issue should have been fixed after installing v1.2.0.

Why can't I write data to a file while the client is running?

When using the Windows client created mount point, you might experience the inability to write data to a file, such as a newly created Notepad file. In this situation, you should first verify that the client is correctly connected to a master server and that there are enough chunkserver disks available to ensure data creation.

For better feedback on the possible error reason for writing operations, you can enable the `sfslogwindowsnotificationarea=1` option. This option is disabled by default, but when enabled, it shows the write error reason, the inode of the issue, and the corresponding path of that inode in the Windows notification area. Additionally, you can use the `sfsmessagesuppressionperiod` option to set the period of message suppression in seconds for logging in the notification area. The following images show a sample of this notification in the Windows environment.



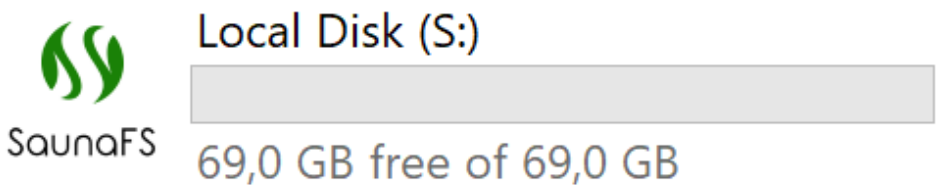
How can I adjust the size of the usable space on a drive according to my needs?

Thanks to changes introduced in version v1.4.2, it is now possible to mount the client in a way that allows filesystem quotas to be considered when determining the current used and total space of a given drive. To do this, you can enable the `usequotainvolumesize` option by setting it to 1, and then use the saunafs command to configure the quota for the specified drive.

For example, you can mount a client like this:

```
C:\> sfsmount -H 192.168.1.158 -P 9421 -D S -o usequotainvolumesize=1
```

This will display a mounted drive like this on Windows:

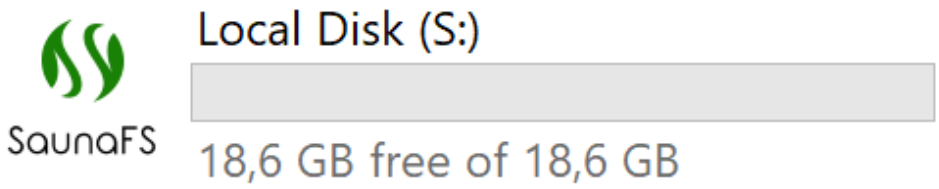


With a total of 60 GB of usable space.

If you want to limit the mounted drive to support at most 20 GB of usable space, you can run:

```
S:\> saunafs setquota -d 0 20G 0 0 S:
```

After applying the quota, the updated drive will show a reduced maximum usable space like this:



As shown, the approximate maximum usable space is now around 20 GB.

Version: 5.0.0

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

Ubuntu LTS 22.04

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 #REMOVE it after installing nfs-ganesha & nfs-ganesha-vfs
packages
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
```

> **NOTE**
>
> We recommend removing Ubuntu 23.04 repositories from **/etc/apt/sources.list** to avoid conflicts with possibly outdated packages or accidental unwanted system upgrade.

> **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**.

Ubuntu LTS 24.04

```
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: 5.0.0

Common Gateway Interface (CGI) Monitor

A **Common Gateway Interface (CGI)** is a standard that facilitates communication between web servers and external databases or information sources. It acts as middleware, allowing web servers to interact with applications that process data and send back responses. The CGI standard was defined by the World Wide Web Consortium (W3C) and specifies how a program interacts with a Hyper Text Transfer Protocol (https://www.geeksforgeeks.org/understanding-http-using-browsers/) server.

SaunaFS has its own monitoring CGI-based server for cheking some of main statistics from a given cluster of our filesystem.

How to run it

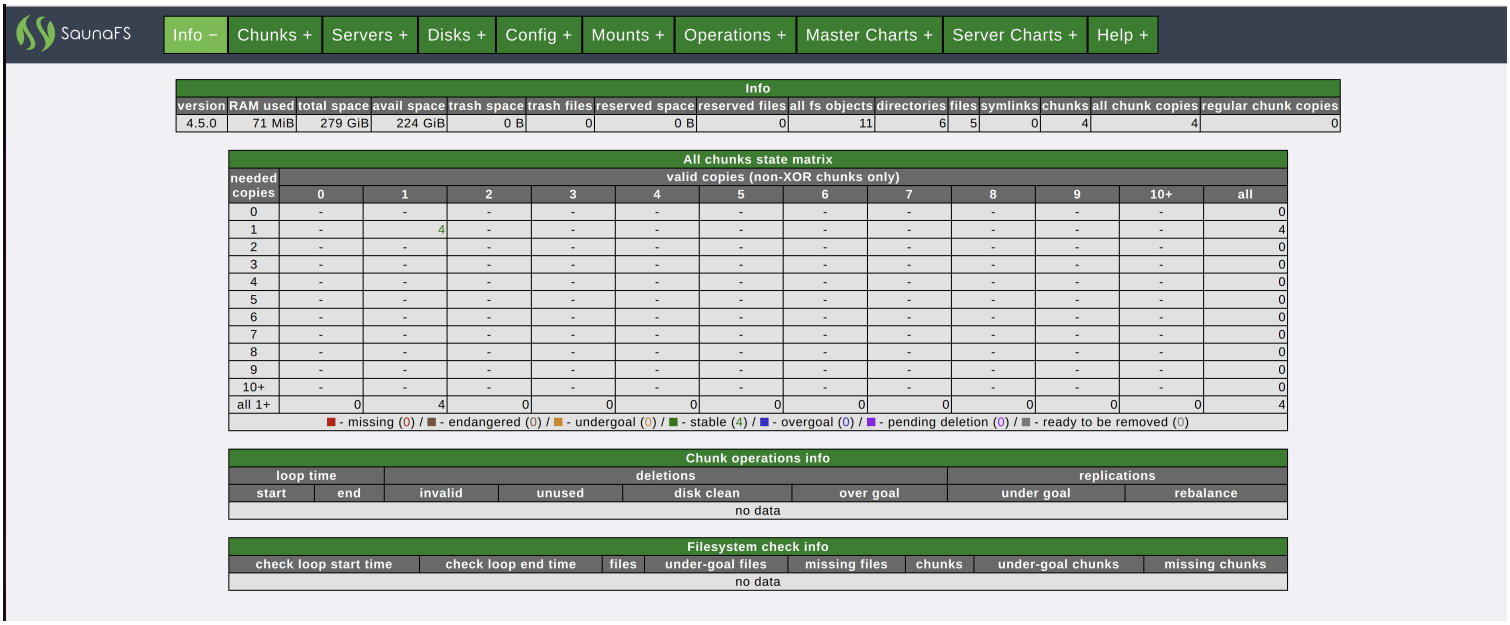
After complete installation of SaunaFS filesystem, there should be available a binary denoted as **saunafs-cgiserver** which is the one related to our monitor system.

For starting this server, it would be enough to run command:

```
$ saunafs-cgiserver
```

As result of this command, the CGI will start as an asynchronous HTTP server running on port `9425` by default and on localhost.

Then, for accessing this tool, you can go to any local browser of you preference and access http://localhost:9425/sfs.cgi



Last show the result of what should be shown when CGI Monitor is running.

Q&A Section

1. **What is reserved space?**

   The reserved space of our filesystem is part of the Filesystem Metadata managed by the master. This variable refers to leght (the size of data space) of all files in our system that are marked as **kReserved** (reserved files).

2. **What are reserved files?**

   The reserved files are an special kind of file managed by our filesystem and are marked as File system nodes of type **kReserved**. There are other filesystem nodes like ordinary files, folders, symlinks etc...

   During the use of our filesystem, some ordinary files could be created. All files in our system have a reference to a sessionId, which is an identifier for a communication session with a client referring to it with its corresponding master servers.

3. **When reserved files are created?**

   At some point of using our filesystem, it could be possible that we try to delete a file. In this case, functions like **PURGE** and **UNLINK** are called in order to successfully complete file deletion. During those functions execution, it is checked at some point if the file to be unlinked and/or purged still has an active **sessionid**, which is some way to express that

current file to be deleted is currently being used. When this happen, this ordinary file is marked as a **kReserved** filesystem node, which is a way to express it has to be a reserved file. It is important to note that, besides the condition of an active session for setting a file as a **kReserved** during **PURGE** operation, it is also necessary that it was selected to trash before.

Reserved files are, in conclusion, files that our filesystem keeps tracks on when they are still being used or implied on a communication process between master and client. Because of this, our filesystem does not complete deletion process and keeps this data until it could be release by another process using it during a session.

4. **When reserved files become not reserved?**

   This happen when a **PURGED** operation is performed and a file to be purged is a **kReserved** file already. Ordinary files first are added to trash, then if they have an active session using it, the system marks it as reserved. Finally, when the purge operation is executed and this file was a reserved file before, then it should be completely deleted from system and not be reserved anymore.

Version: 5.0.0

Installation

> **TIP**
>
> Looking to build the source? Head over to https://github.com/leil-io/saunafs/ and check the README below, or see the section on building from source down below.

Currently we package for Ubuntu only, but you can always compile from source (see below)

Current supported Ubuntu versions are:

- 22.04
- 24.04

Ubuntu

First check that you have GPG setup correctly. You can use this command:

```
gpg2 --list-keys
```

This will make sure that the .gnupg exists in your home directory, creating it if it doesn't exist.

Import the public key used to sign the packages

```
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 program `lsb_release` is installed.

Ubuntu 24.04:

```
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-24.04/ noble main
EOF
```

Ubuntu 22.04:

```
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
```

HINT

For some version of `apt` using `Sub-process /usr/bin/sqv` ASCII format of keyring is required. Please use this command to generate it:

```
gpg --no-default-keyring --keyring /usr/share/keyrings/saunafs-archive-
keyring.gpg --export --armor > /usr/share/keyrings/saunafs-archive-
keyring.asc
```

and use `/usr/share/keyrings/saunafs-archive-keyring.asc` as `[arch=amd64 signed-by=/usr/share/keyrings/saunafs-archive-keyring.asc]` in `/etc/apt/sources.list.d/saunafs.list` file.

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.

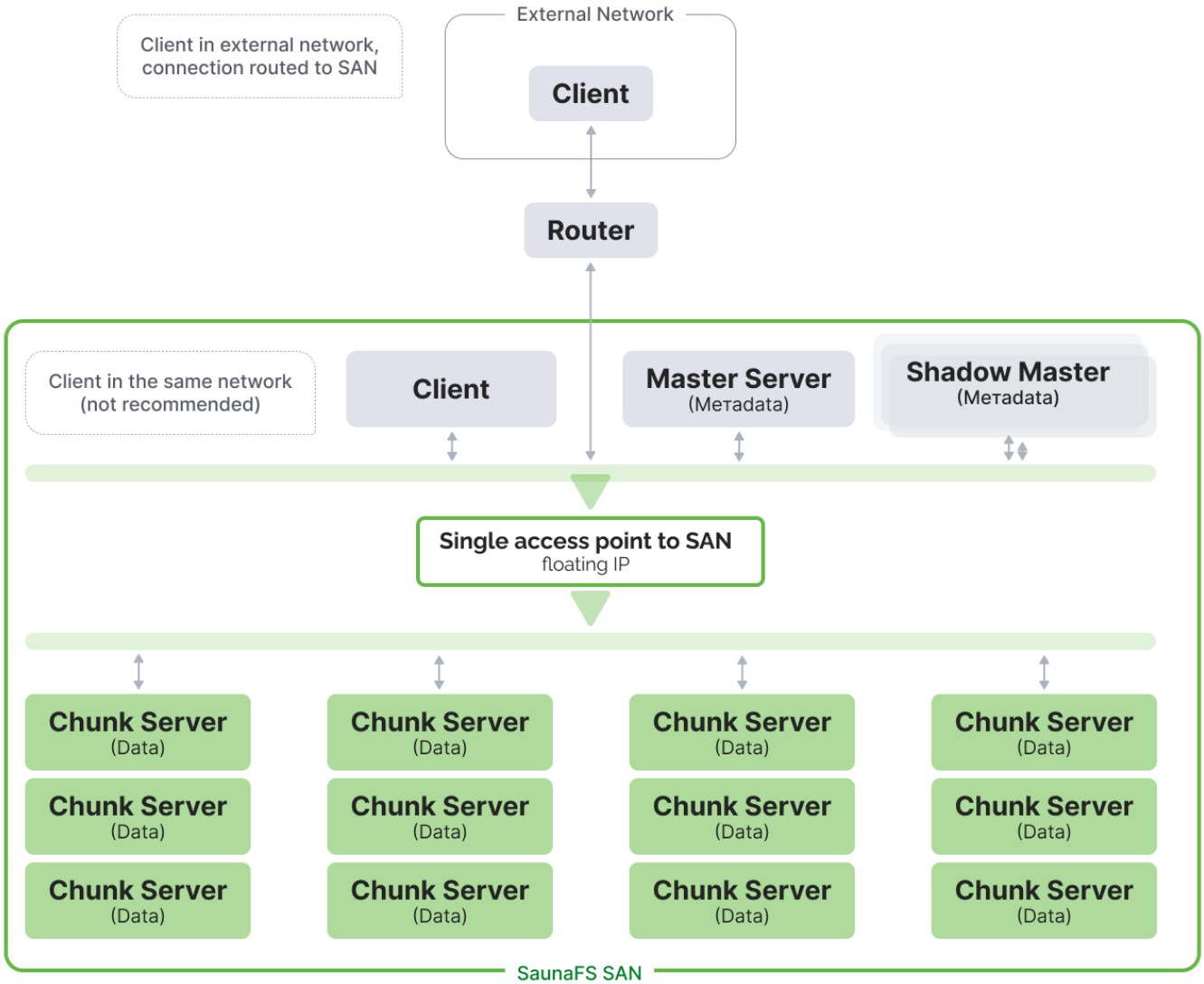
Version: 5.0.0

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: 5.0.0

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: 5.0.0

Storage device setup/removal

All storage devices used by chunkserver are defined in sfshdd.cfg (see `man sfshdd.cfg` for more details). They need to be mounted to a specific directory and defined in the file. The chunkserver will assume it's dedicated, but it's not strictly necessary for it to be (but highly recommended).

After any modification to the sfshdd.cfg files, the chunkserver needs to be reloaded, by:

`systemctl reload saunafs-chunkserver`

OR

`sfschunkserver reload`

OR sending a SIGHUP signal to the running process.

Recommended Filesystems

While any filesystem can be used, we recommend XFS for CMR drives with the following options:

```
/dev/sda /mnt/sfschunkservers/data/sda xfs
rw,noexec,nofail,nodev,noatime,nodiratime,largeio,allocsize=16777216,inode64 0 0
```

`allocsize` can vary depending on the goal replication. For simple goals it should be 64 MiB, but for EC this will depend on the number of data parts. For example, with EC(4,3) (4 data parts and 3 parities), the alloc size should be (64 MiB / 4) 16 MiB (16777216). Or with EC(3,1), it should be (64 MiB / 3) 22.4 MiB (22369621)

Adding storage devices

Adding a storage device is to simply point to the directory where it's mounted. For example, if you had /dev/sda mounted in /mnt/hdd1/, you need to simply add `/mnt/hdd1` to the file.

Replacing storage devices (non-damaged)

If you wish to replace the storage devices (provided they are still working), it may be best to mark them for removal with *. From the previous example, to replace `/dev/sda`, add `*` to the beginning:

```
*/mnt/hdd1
```

It will start copying files to other devices. Once the process finishes (all the chunks are copied to other drives), it can safely be removed by deleting or commenting the line.

Removing/replacing storage devices

Removing the devices is achieved by simply deleting/commenting out the line. In this case the device will be treated as missing and, after some delay (defined by `OPERATIONS_DELAY_INIT` in sfsmaster.cfg), will start replicating from other devices to restore the chunks.

With bigger data, this is much faster than moving the data out of a device, but will put the entire system under strain as it will need to replicate data to elsewhere.

Version: 5.0.0

Replication

SaunaFS supports two 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.

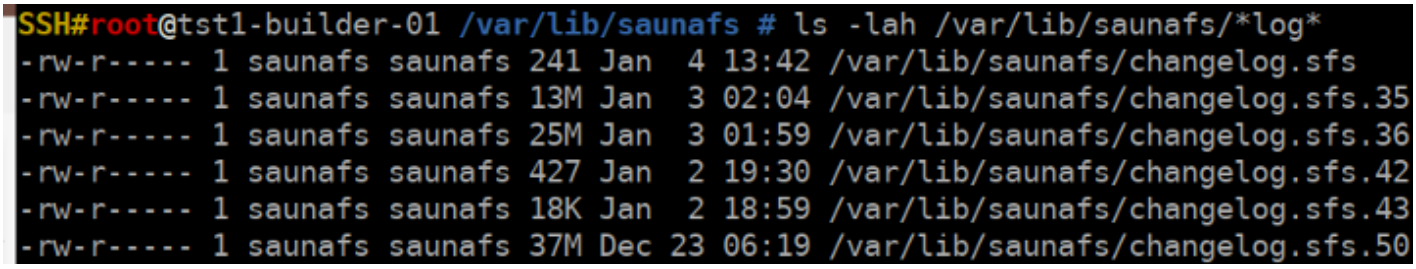
Version: 5.0.0

Logs and logging

There are 3 types of logs in SaunaFS

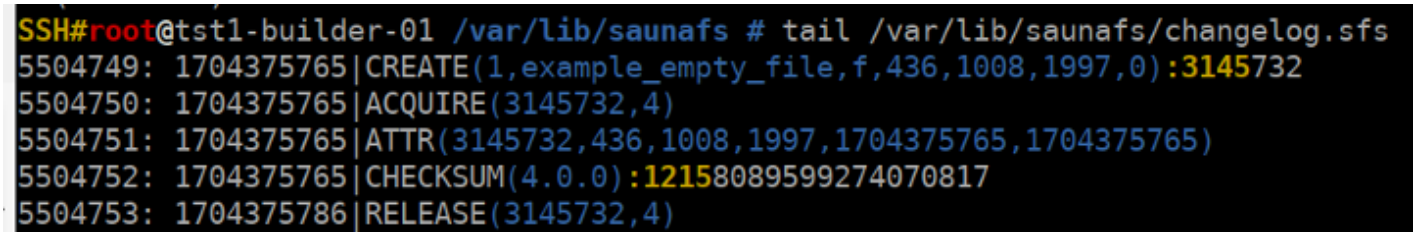
Metadata logs

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

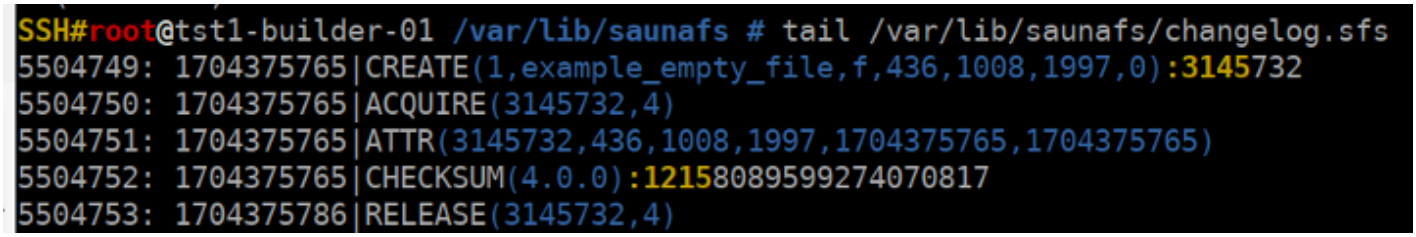



```
#root@tst1-builder-01 /var/lib/saunafs # ls -lah /var/lib/saunafs/*log*
-rw-r----- 1 saunafs saunafs 241 Jan  4 13:42 /var/lib/saunafs/changelog.sfs
-rw-r----- 1 saunafs saunafs 13M Jan  3 02:04 /var/lib/saunafs/changelog.sfs.35
-rw-r----- 1 saunafs saunafs 25M Jan  3 01:59 /var/lib/saunafs/changelog.sfs.36
-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

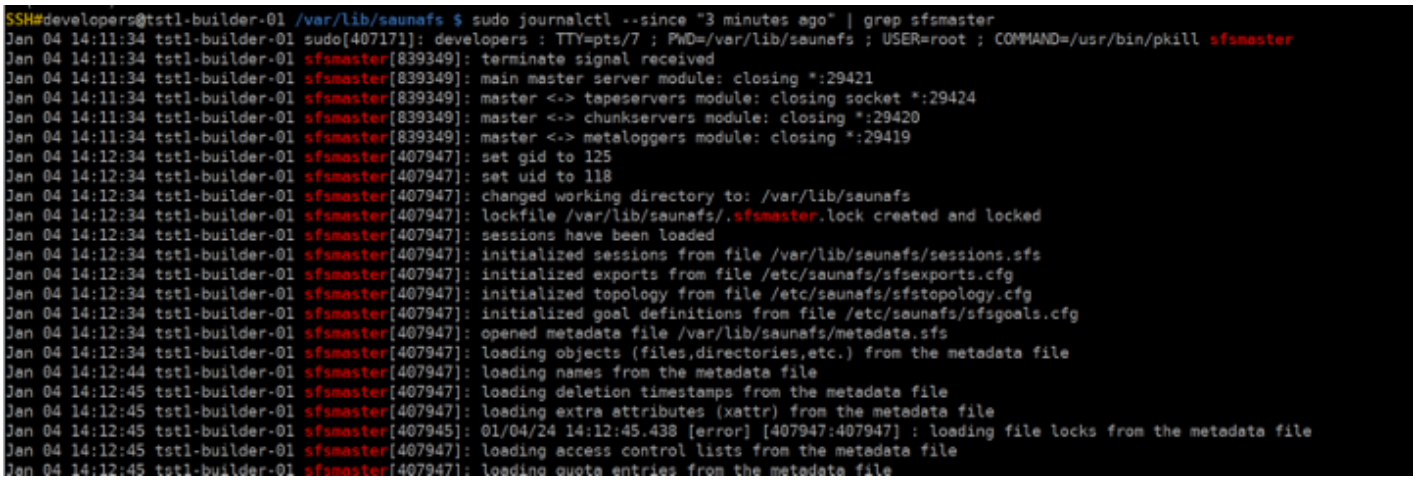
`/etc/saunafs/sfsmaster.cfg`

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
```

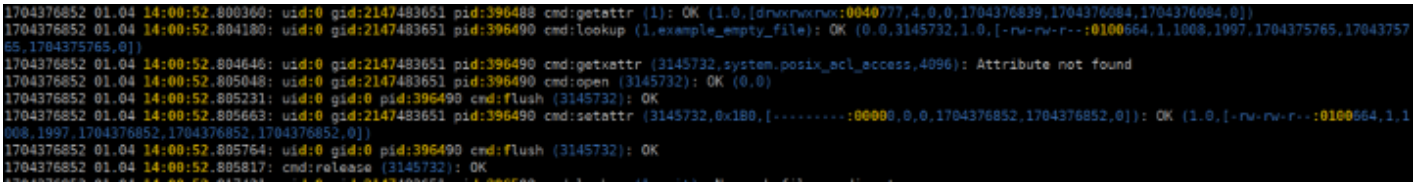
For example

```
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: 5.0.0

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: 5.0.0

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: 5.0.0

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: 5.0.0

Dev Guide

Development Environment

Currently, we target 24.04 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 setup /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: 5.0.0

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: 5.0.0

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: 5.0.0

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.