Object Mount

User Guide

Version 26.11.6 · November 2026

Contents

Object Mount 4

macOS 6

- Installation Guide 7
- User Guide 13
- Feature Guide 23

Windows 27

- Installation Guide 28
- User Guide 29
- Feature Guide 37

Linux 42

- Getting Started Guides 45
  - POSIX Options 46
  - Deployment Modes 49
- Installation Guides 50
  - glibc 51
  - musl 54
  - Debian 57
  - Red Hat 60
  - Alpine 63
  - Windows (WSL) 66
- User Guides for Linux 67
  - Validate Functionality & Access Object Storage 68
  - Activate a License Key 75
  - Deployment Mode Details 76
  - Direct Interception 77
  - Object Mount on FUSE 81
  - Object Mount FlexMount 87
  - Object Mount Fusion 90
  - Enable Enforced POSIX File Access 100
  - Cloud Path Details 105
  - Application Tips 108
  - Media & HPC Modes 111
- Advanced Guides 112
  - Advanced Credential Options 113
  - Advanced Loading Options 119
  - Advanced Configuration Options 126
  - Server-Side Encryption 134
  - Kubernetes CSI Driver 137
  - Machine Learning Accelerator 147
  - Technical Limitations 149
  - Uninstalling Object Mount 151

Media Workflow Guides 153

Adobe Premiere Pro . . . 155
Avid Media Composer . . . 157
DaVinci Resolve . . . 159

Appendix . . . 161

Core Concepts . . . 162
POSIX Explained . . . 164
Moving POSIX attributes to the .om folder . . . 167
Libuplink mode (Storj only) . . . 171
S3 Credential Management . . . 179
Supported S3 Providers . . . 186
Performance Tuning . . . 187
Troubleshooting Guide . . . 190
FAQ . . . 196
Release Notes . . . 198

Object Mount

Welcome to the Storj Object Mount Documentation!

Storj's Object Mount makes working with object storage as fast and seamless as using a local drive.

Whether you're working in media production, research & development, or backup workflows, Object Mount combines the convenience of local storage with the scalability of the cloud.

What is Object Mount?

Object Mount is a high-performance mount client that provides seamless, native access to object storage as if it were a fast, local file system. It allows existing applications to interact with object storage content without needing to recode apps or change your workflow — just mount and go.

Feature	Benefit
Native Clients	Object Mount offers native clients for Linux, Windows and macOS. Installs in minutes and can run inside unmodified containers.
Compatible and Seamless	Run your same, unmodified tools and applications — now with fast access to object storage data. Files and directories on object storage behave the same as they do on a local disk. Fast, transparent, user-friendly.
POSIX Compliant	Run your Linux tools, applications and workloads as expected — with all the same user and group permissions, symbolic & hard links, etc.
High Performance	Intelligent caching and high-throughput make editing large files in real-time a breeze. Adobe Premiere Pro, DaVinci Resolve and any other industry standard toolkit all work "out-of-the-box." Cloud storage is now just as responsive as local storage.
No Proprietary Formatting	Object Mount keeps your files in their original format without any proprietary conversions. Your data is accessible across workflows, teams, tools, apps and dashboards. No risk of vendor lock-in.
Cost-efficient and Scalable	By only accessing the bytes you need, Object Mount minimizes download bandwidth and reduces the need for expansive (and expensive) local storage arrays, lowering egress costs and saving you money on every project.
Eco-friendly Collaboration	Reduce the energy and resource footprint of your Media and Entertainment workflows. Global teams access the same 'single source of truth' content without moving, copying or replicating media across multiple, disparate platforms.

Bin Locking

Bin locking prevents two editors from changing the same bin or project at the same time on Object Mount. The first person to open it gets write access and others are read only until the lock clears. Object Mount propagates lock state quickly and enforces read only behavior, but it does not create or remove locks. Some workflows also require additional third party software to coordinate locking with your editing application.

Where to Start

Learn more about how Object Mount works in the many **Appendix** articles:

- Object Storage Core Concepts
- POSIX Explained

Follow the **Installation Procedure** and **User Guides** for your operating system:

- macOS
- Windows
- Linux

Optimize your workflows with Media and Application tips:

- Premiere Pro from Adobe
- Media Composer from Avid
- DaVinci Resolve from Blackmagic Productions

macOS Installation and Usage

This section contains everything you need to install and use the native **Object Mount** app on an Apple Mac.

Whether you're setting up for creative workflows, remote storage access, or team-based collaboration, these pages will walk you through installation, configuration and day-to-day use.

What's Covered

- Native installation using a `.dmg` package
- System requirements and FUSE dependencies
- Setup guidance for Macs using either Apple Silicon and Intel chips
- User guides for mounting, configuring, and managing your object storage
- Tips and recommendations for unique macOS features, media applications, and performance improvements

Next Steps

Start with the **Installation Guide**:

- macOS Installation Guide

Move on to the **User Guide** for instructions on creating mounts and working with object storage in your daily workflows:

- macOS User Guide

Lastly, review the macOS OS-specific **Feature Guide** and the appropriate **Media Workflow Guide**:

- macOS Feature Guide
- Media Workflow Guides

Installation Guide: macOS Native App

This section covers how to install the Object Mount native application for macOS using a standard `.dmg` image.

The steps below work for Macs with either Intel processors or Apple Silicon (M1-M5) chips.

System Requirements

Before you begin, make sure your Mac meets the following system requirements:

- macOS 13 Ventura or later (macOS 13.5 or later for the Finder extension)
- At least 300MB of free disk space
- Internet connection for mounting cloud-hosted buckets, or LAN connection for mounting local S3 buckets
- macFUSE must be installed (see instructions below), at least the version for your macOS:

macOS	Minimum macFUSE
13 Ventura	4.4.0
14 Sonoma	4.5.0
15 Sequoia	4.8.0
26 Tahoe	5.0.6

> **NOTE**
>
> **S3 Provider Support:**
>
> Object Mount for Mac fully supports S3 container services from Amazon Web Services, Storj, and other fully S3-compatible Object Storage providers.
>
> - See the Supported S3 Providers article in the Appendix for a complete S3 provider list.
>
> **Azure & GCS Support:**
>
> - Support for *Native* Microsoft Azure Storage and *Native* Google Cloud Storage services are still under development in Object Mount for Mac.
> - For Google Cloud access: Using GCS *S3* is recommended instead of using GCS *S3 Gateways*.
> - Native Azure and Native GCS (S3 Gateways) should be considered “experimental” at this time.

Step 1: Download the latest Installation Image

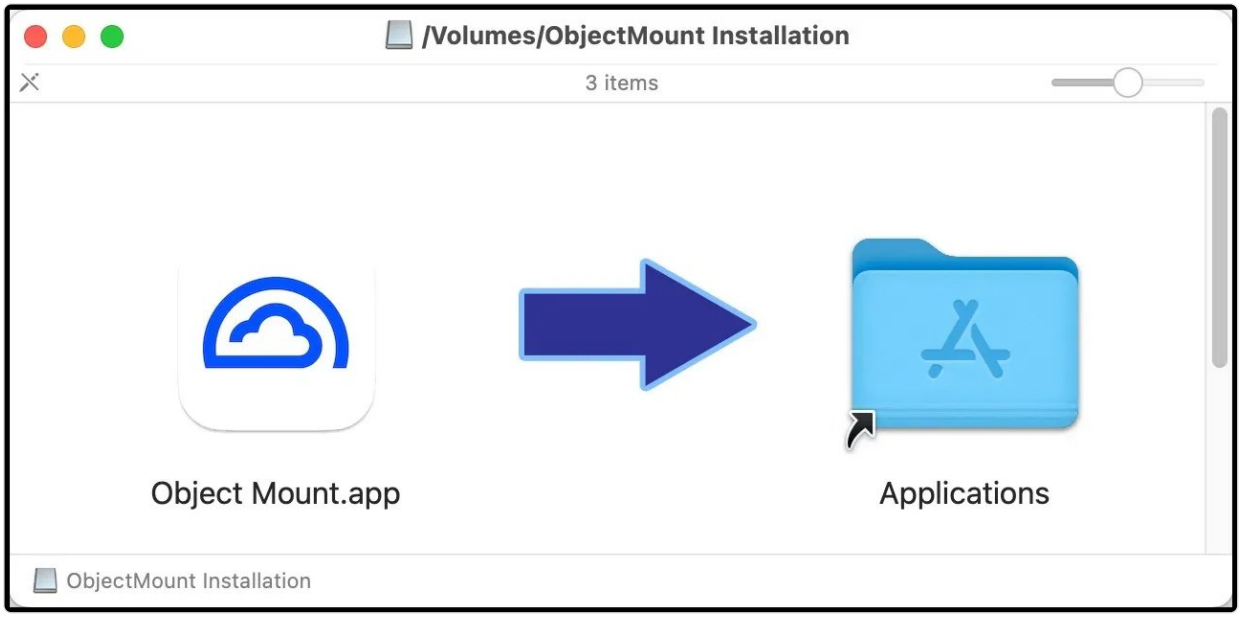
Download the correct macOS `.dmg` disk image file for your system.

Note: Storj customers have 2 free licenses included. To obtain additional licenses or if you are not a Storj customer, reach out to your 🌐 Storj Representative.

> **NOTE**
>
> **Mac Chip Types**
>
> Be sure to choose the disk image that matches your chip architecture:
>
> - **x86** for Intel processors
> - **arm64** for Apple Silicon/chips (M1-M5).
>
> To determine which chip you have:
>
> - Click the Apple menu in the top left
> - Select **About This Mac**
> - Look for **Processor** (Intel) or **Chip** (Apple M1, M2, etc.)

Step 2: Install the Object Mount App

1. Double-click the `.dmg` file to mount it
2. Drag the Object Mount app into your **Applications** folder



Step 3: Install macFUSE (Required)

The native macOS Object Mount app requires the **macFUSE** system extension.

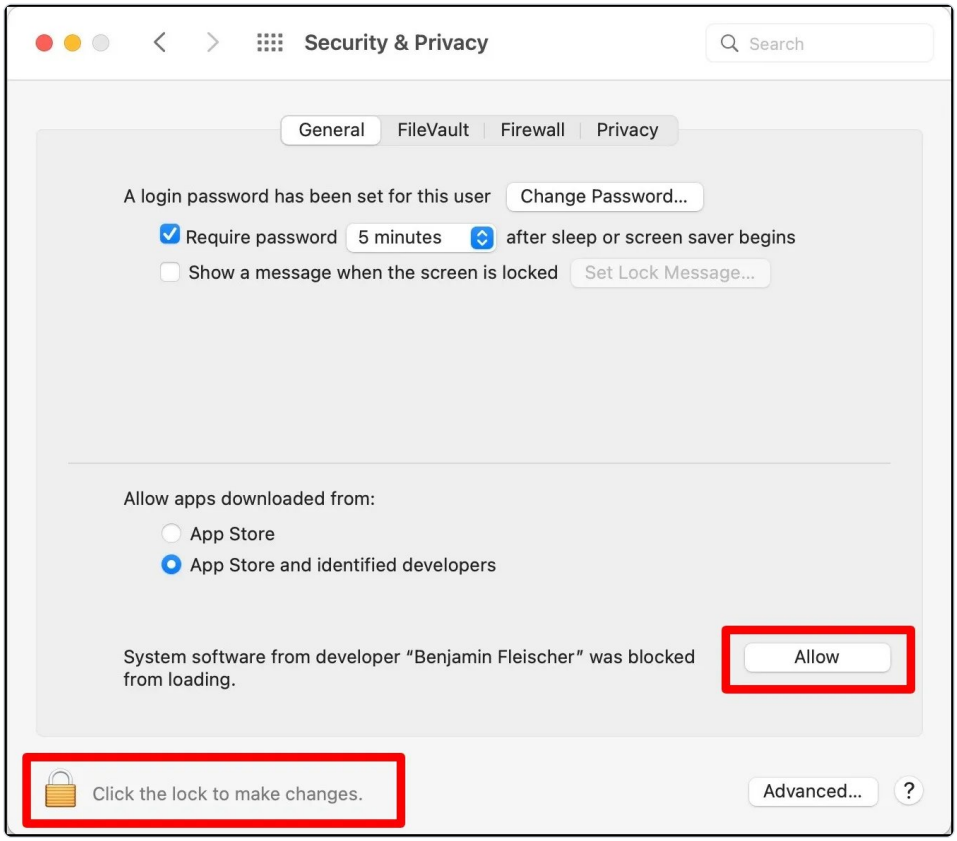
macFuse, on both Intel and Apple silicon Macs, provides system-level volume mounting extensions that Object Mount leverages.

Download and install macFUSE following the steps below for your Mac hardware type:

Intel Processor

1. Download the **latest** version (not pre-release) `.dmg` image from the macFUSE releases page.
2. Open the macFUSE `.dmg` and run the **Install macFUSE.pkg**.
    - As prompted, click **Allow**, **Continue** and/or **Agree** to install the package
3. As a new System Extension, the package will generate a “System Extension Blocked” message.
    - Open **System Settings**
    - Open **Security & Privacy**

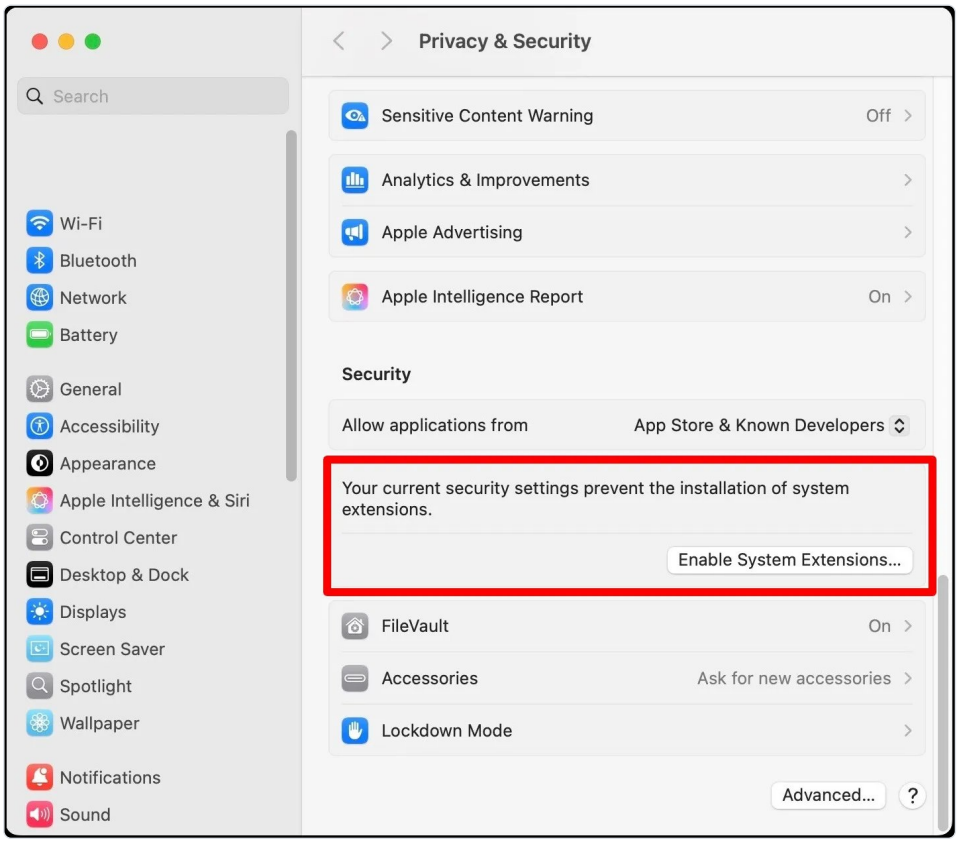
- Click the **lock** 🔒 icon to allow changes
- Click the **Allow** button to permit the installation



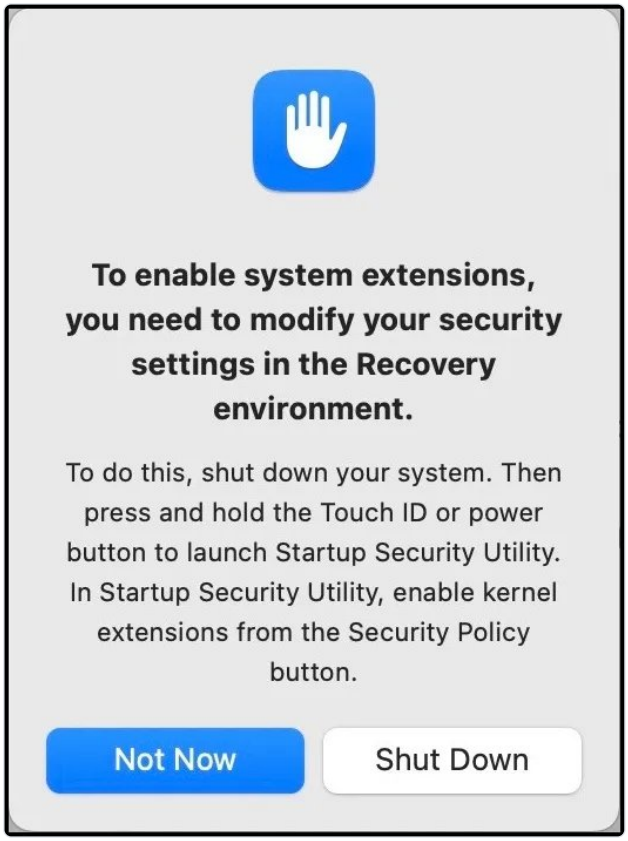
4. Restart your Mac when prompted.

Apple Silicon (M Chip)

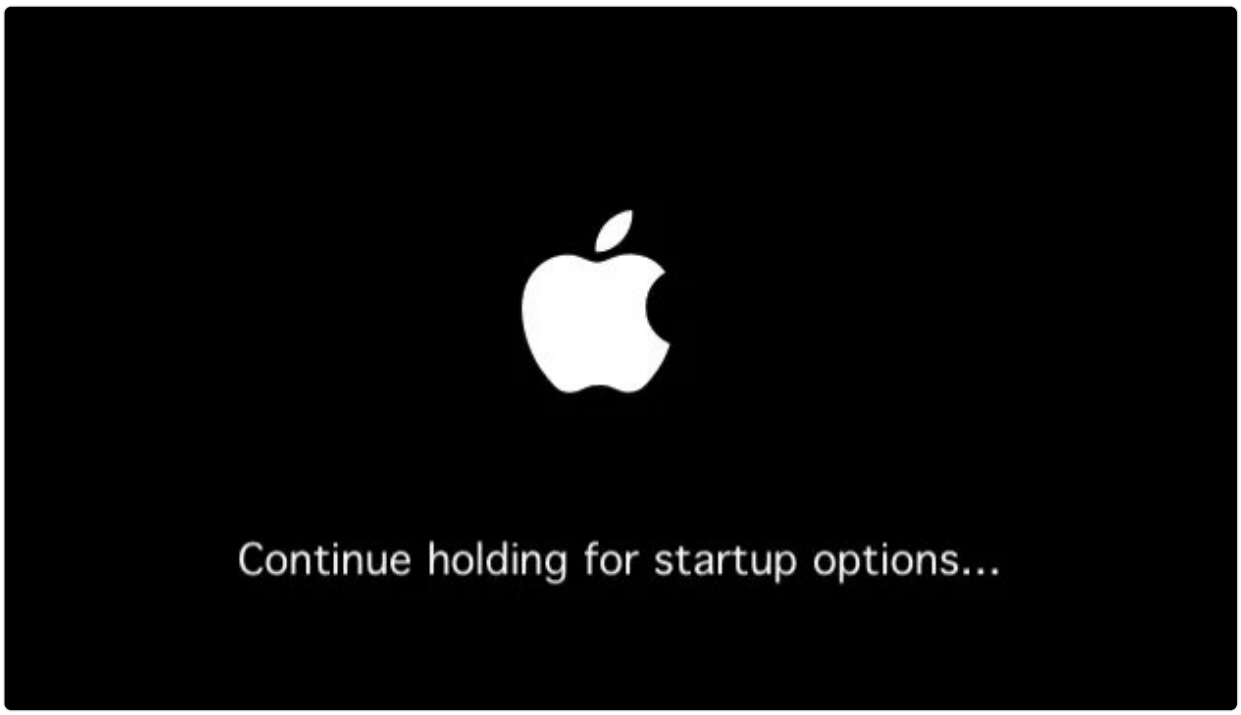
1. Download the **latest** version (not pre-release) `.dmg` image from the macFUSE releases page.
2. Open the macFUSE `.dmg` and run the **Install macFUSE.pkg**.
   - As prompted, click **Allow**, **Continue** and/or **Agree** to install the package
3. As a new System Extension, the package will generate a "System Extension Blocked" message.
   - Open **System Settings**
   - Open **Privacy & Security**
   - Scroll down to the **Security** section
   - Find and click the **Enable System Extensions...** button



4. You will then be prompted to **Shut Down** and restart your Mac into its **Recovery Mode** -Save any work you have in progress
    - Click **Shut Down**



5. With your Mac powered off, **PRESS AND HOLD** your power button (or Touch ID button) for 3-5+ seconds.
    - You should see an Apple message that says **Continue holding for startup options...**



    - Continue holding the power button until the message changes to **Loading startup options...**
6. In the modified boot screen, click the gear icon labeled **Options** then click **Continue**



    - Select your admin user, click **Next** and enter your password to view the **Recovery Mode** screen and its menu options
7. From the **Recovery** menu, select the **Utilities** drop-down menu and choose **Startup Security Utility**
8. Select your primary hard drive and click **Unlock...**
    - Enter your Admin password and click **Unlock**
9. Once unlocked, click the **Security Policy...** button
10. Click the **Reduced Security** radio button
    - Then, tick the first box for **Allow user management of kernel extensions from identified developers**



- Click **OK** and enter your admin password one last time

11. Lastly, from the **Startup Disk** Menu, choose **Shut Down**

Restart your Mac as normal.

Next Steps

Proceed to the macOS User Guide to **activate** your Object Mount installation, **configure** your object storage credentials, and **create** your first mount.

Then review the macOS Feature Guide.

Finally, read through the usage tips for your specific Media Workflows and Applications.

User Guide: macOS Native App

This guide walks you through the core steps to get started with **Object Mount on macOS**, including: activation, adding credentials, and mounting your cloud-based object storage as a local volume.

Step 1. Launch Object Mount

Once Object Mount and macFUSE have been installed on your Mac (see the macOS), follow the steps below to launch and activate Object Mount:

Step 2. Configure S3 Credentials & Create a Mount

Object Mount can connect to object storage from Storj as well as other major cloud storage providers. Standard S3-compatible credentials from your cloud storage provider are required.

Obtain Your Object Storage Credentials

- See the Appendix article for instructions.

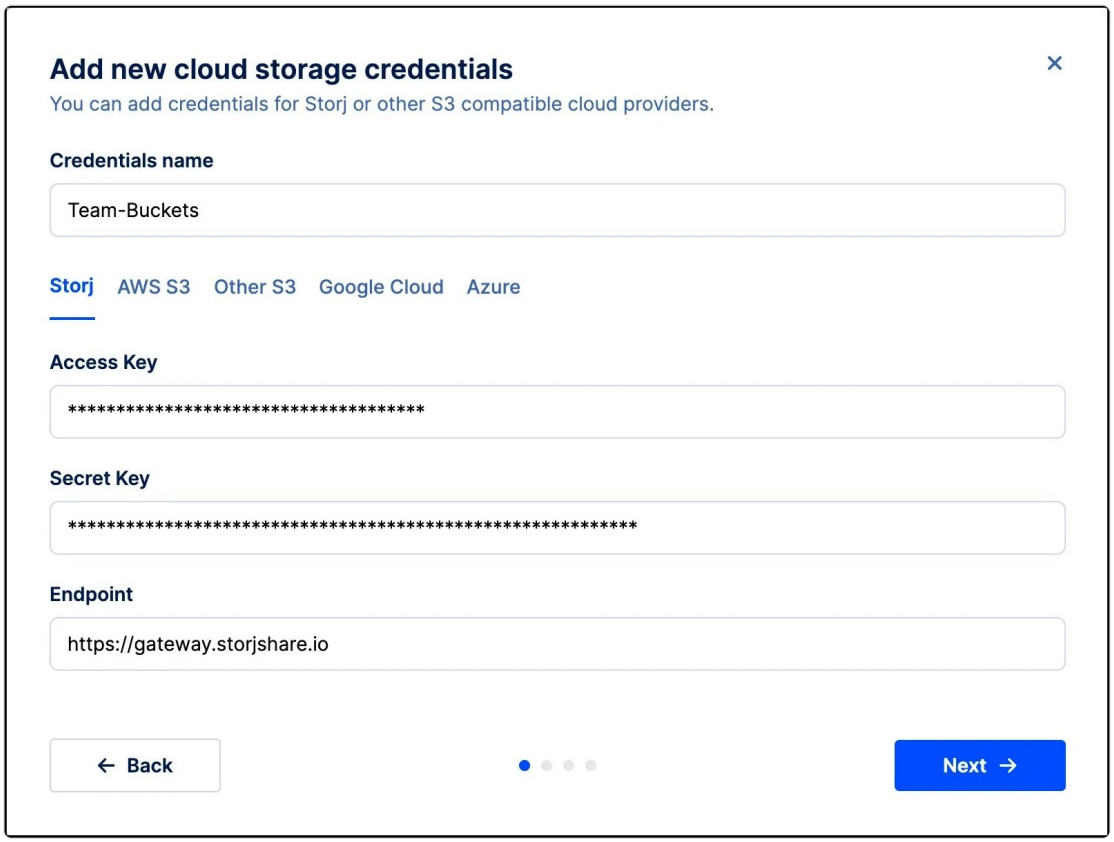
Enter Your Credentials and Create a Mount

There are two easy paths to create a mount — both paths achieve identical results and both require your object storage credentials:

- Click **Create New Mount** from the **Mounts** tab, or
- Click **Add New Credentials** from the **Credentials** tab
- Then select and configure one or more buckets to mount as a local drive

Follow the steps below to create a new mount using the **Add New Credentials** path:

1. Select the **Credentials** tab in Object Mount.
2. Click the **Add New Credentials** button.
    - Give the credential set a clear name.
    - Choose your provider tab (e.g.: AWS, Storj, etc.).
    - Enter the **Access key ID**, **Secret key**, and **Endpoint** (or Region, if applicable) that you obtained earlier.


Add new cloud storage credentials
You can add credentials for Storj or other S3 compatible cloud providers.
Credentials name
Team-Buckets
Storj
AWS S3
Other S3
Google Cloud
Azure
Access Key
****
Secret Key
****
Endpoint
https://gateway.storjshare.io
Back
Next

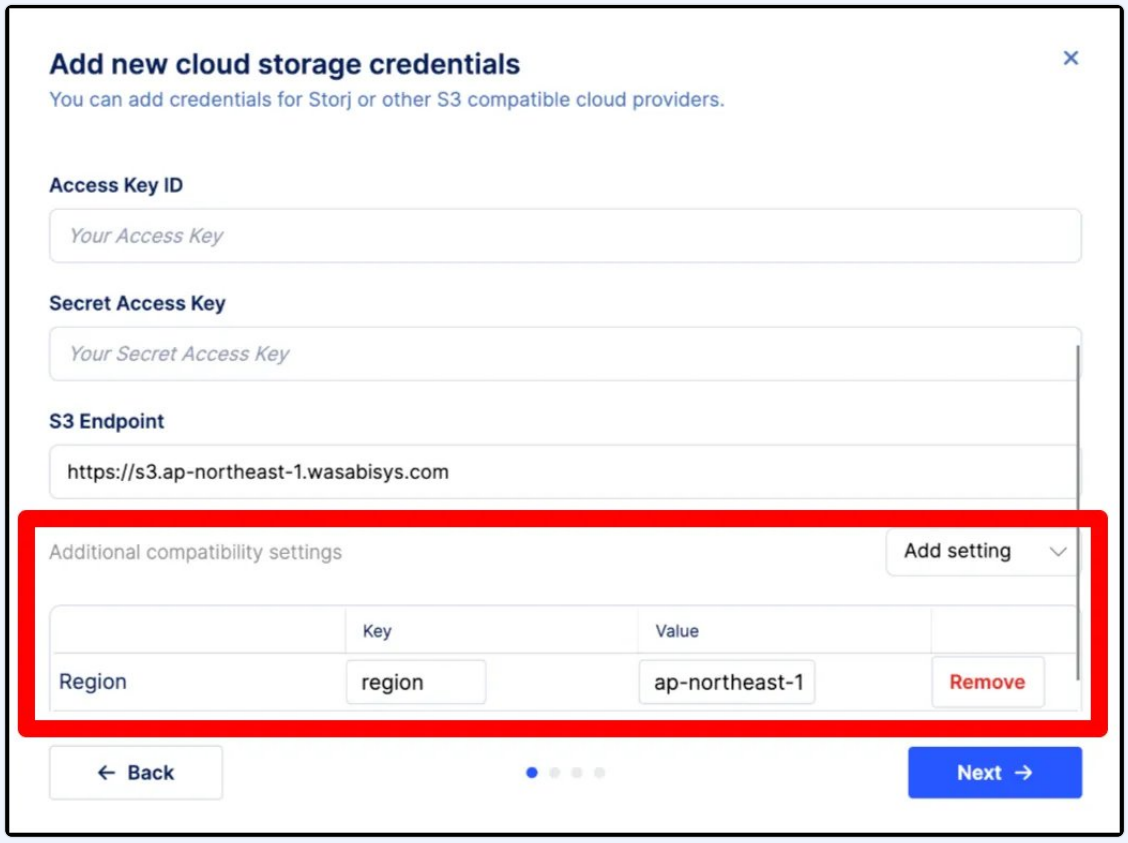
NOTE

S3 Compatible Providers

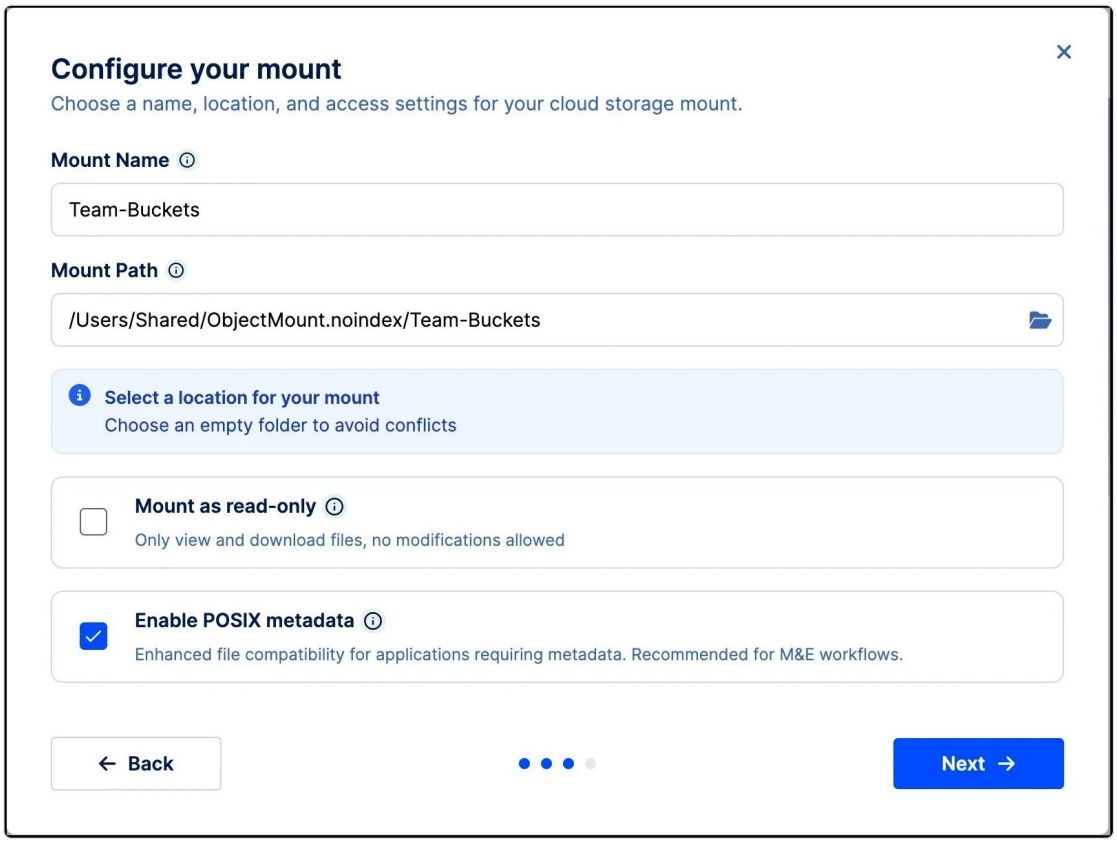
If your provider is not listed, select **Other S3** and choose the provider from the dropdown.

- Ensure the endpoint is accessible from the Internet.
- Some providers may require extra compatibility settings like a defined region.

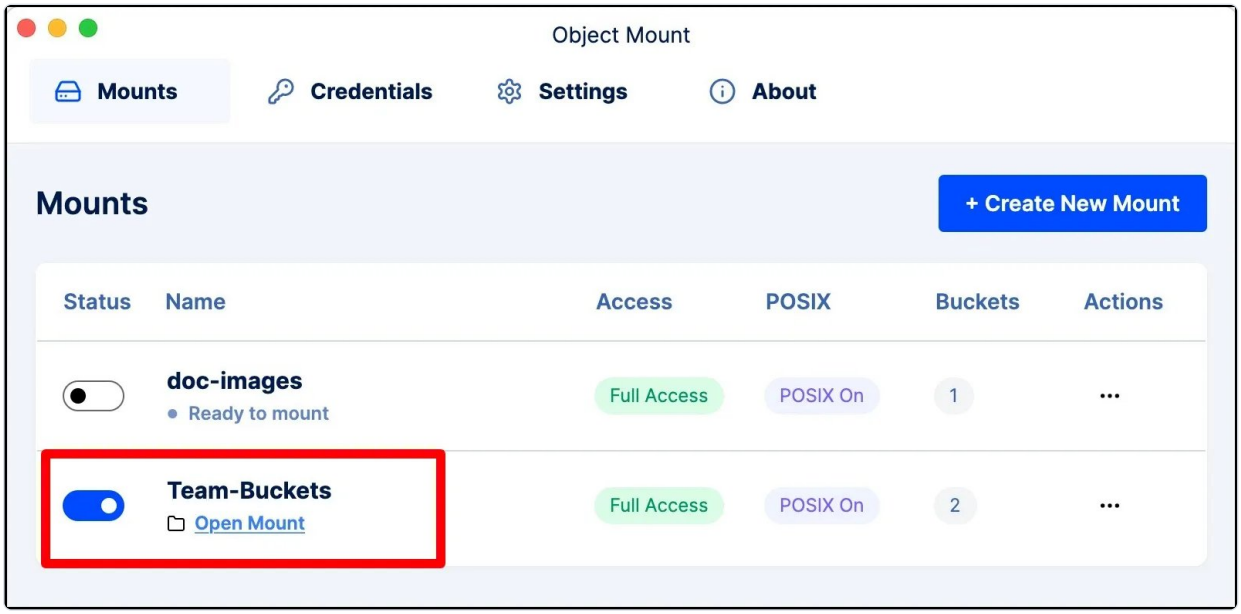
Below is an example of providing an additional S3 Compatibility Setting for region:



3. Click **Next**.
   - Object Mount will authenticate to your cloud storage provider and attempt to discover any buckets your credentials have access to.
4. Select **Mount all buckets** or choose one or more specific buckets to create a mounted drive for, then click **Next**.
5. Enter a **Mount name** to use and, if necessary, alter the path location for the local mount drive.
6. Tick the **Mount as read-only** box if all you need is to browse and copy content.
   - Uncheck the box to enable **full read/write access**.
7. Tick the box to **Enable POSIX metadata** (recommended).
   - For more details, see the Appendix article: POSIX Explained.



8. Click **Next** to create the mount. Then enable the mount from the **Mounts** tab.



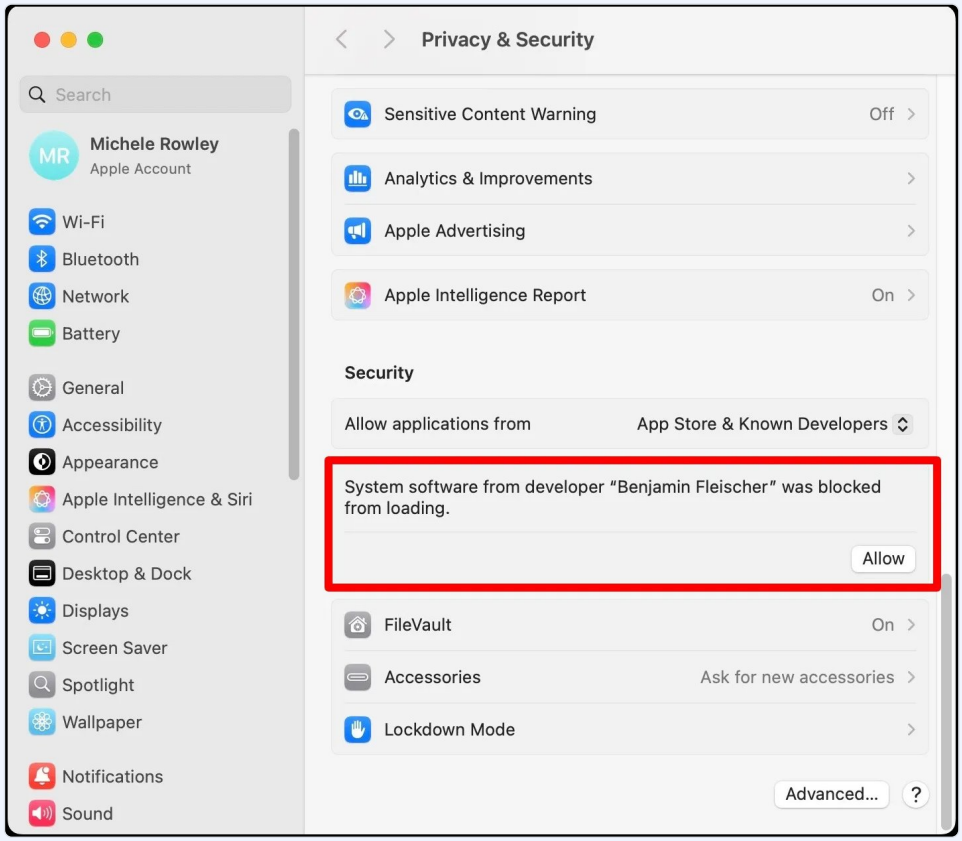
NOTE

"Extension Blocked" Message

Depending on your Mac hardware and OS version, an **"Extension Blocked"** message may be presented when Object Mount attempts to mount your new drive.

If so, perform the following steps:

- Open **System Settings**
- Navigate to **Privacy & Security**
- Scroll down to the **Security** section
- Click **Allow** to permit the macFUSE extension (from developer "Benjamin Fleischer") to be loaded



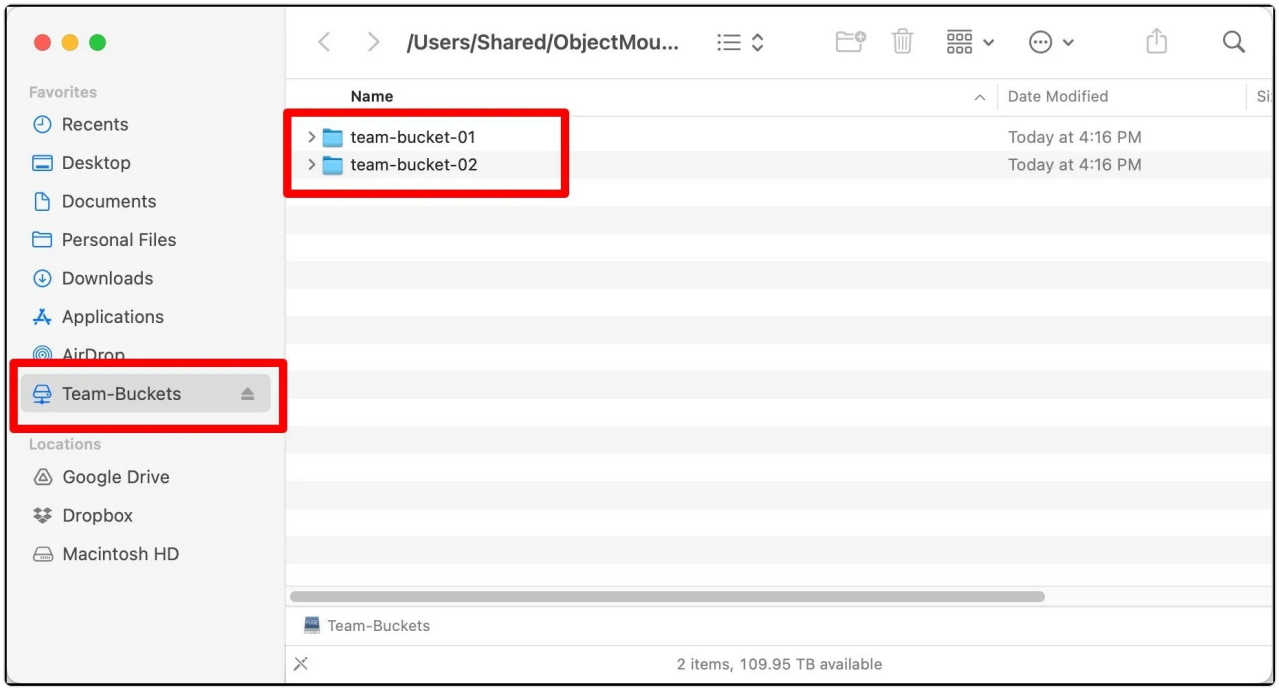
Note: First-time access to a new mount may take a few moments if the bucket contains many files — performance improves after the initial mount.

Newly Mounted Drive & Buckets

Your newly mounted drive will appear on your Mac using a **FUSE** drive icon:



If there are multiple buckets accessible using the same credential set, those buckets will each appear as separate **folders** within the single mounted drive icon:



Preferences & Advanced Settings

Object Mount for macOS includes a flexible **Settings** panel where you can fine-tune caching, logging, and performance settings.

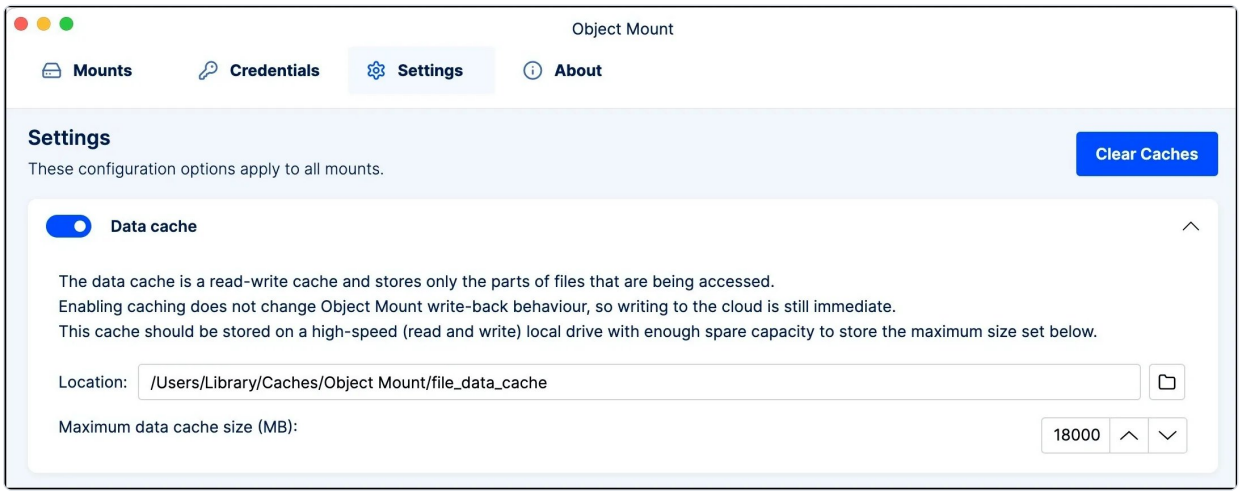
> **NOTE**
> **Settings**
>
> - These settings are global and apply to all mounts.
> - These settings cannot be edited while a mount is active. Disable all mounts before making changes.

Refer to the following sections for details on each setting:

Data Cache

The local **Data Cache** temporarily stores small portions of accessed cloud files on your local disk — useful for repeated or random access operations.

- **Location:** The local path where data cache files are stored
- **Maximum cache size:** The upper limit (in MB) for total cached data
  - **Note:** This feature does *not* delay cloud writes — writing to the cloud happens immediately.

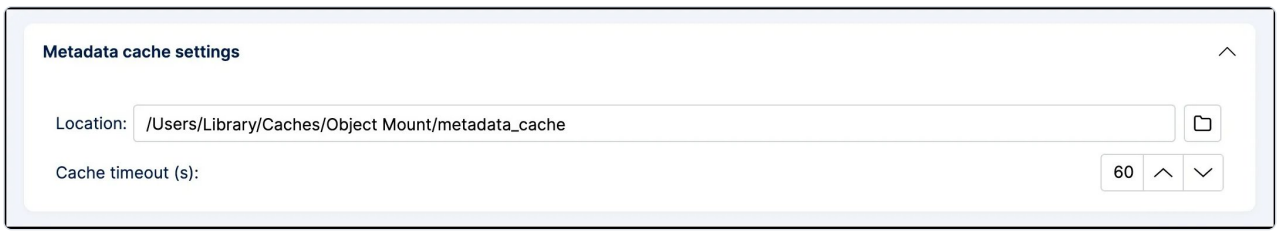


> **NOTE**
>
> **Data Cache Recommendations**
>
> - Use a fast local drive (SSD or NVMe) for best caching performance.
> - If the connection to your object storage is on a high-speed, low-latency network, you may see improved performance by turning caching *off*.

Metadata Cache Settings

Metadata caching helps Object Mount load and retain file structure, timestamps, and permissions more efficiently.

- **Location:** The local folder for storing metadata information
- **Cache timeout:** The duration, in seconds, that metadata remains valid before being refreshing from the cloud (default: 60 seconds)

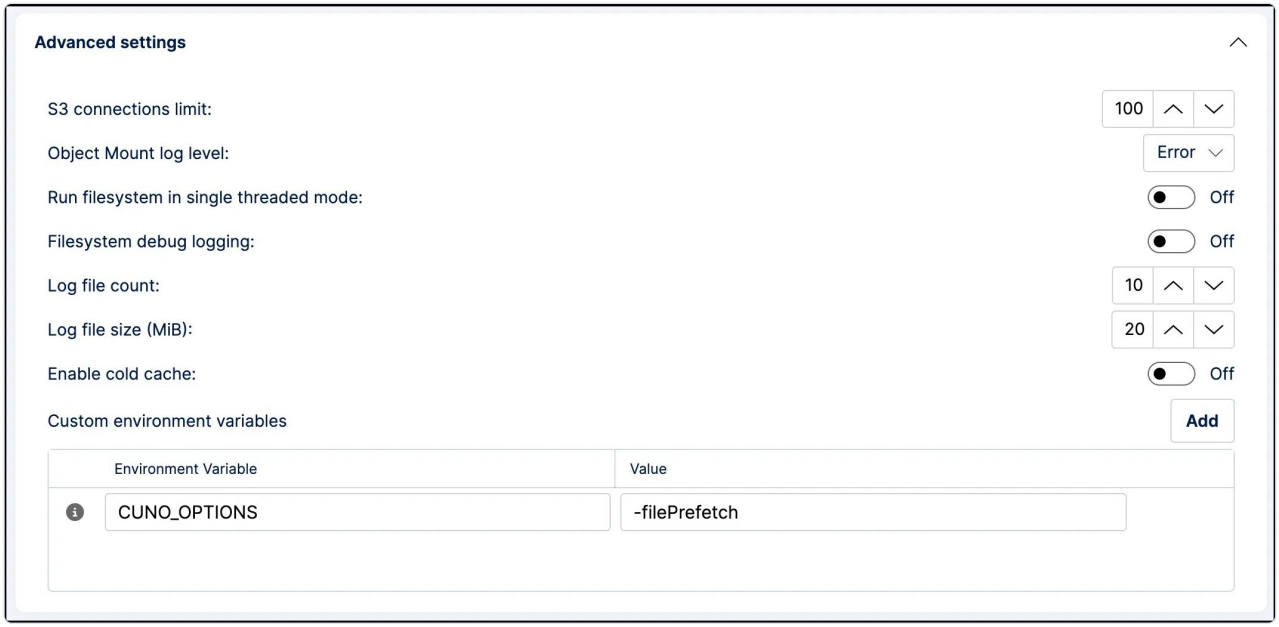


Advanced Settings

These options let you further customize Object Mount's performance and diagnostic behavior.

Setting	Description
S3 connections limit	Controls the number of concurrent S3 operations (default: 100).
Log level	Adjusts logging detail. Options: `error`, `warning`, `debug`, or `trace`.

Setting	Description
Single threaded mode	Runs the filesystem in single-threaded mode (rarely needed; leave `off` unless instructed by Storj Support).
Debug logging	Enables extensive debug output to assist with support issues.
Custom Environment Variables	Custom variables to modify Object Mount's behavior, add variables when recommended by Storj Support.



NOTE

Log File Location

On macOS the default location for storing the log file is:

`~/Library/Application Support/Object Mount/cunoFS.log`.

Click **Apply** to save any changes. Then re-enable your mount.

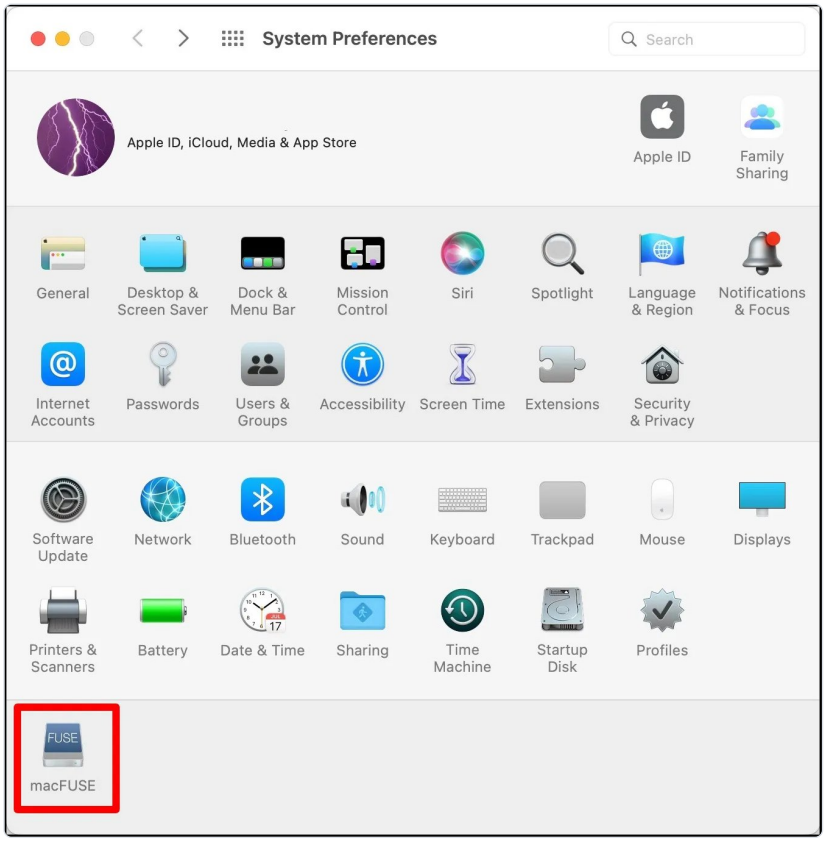
Click **Reset** to revert all settings back to their default values.

Updating macFUSE

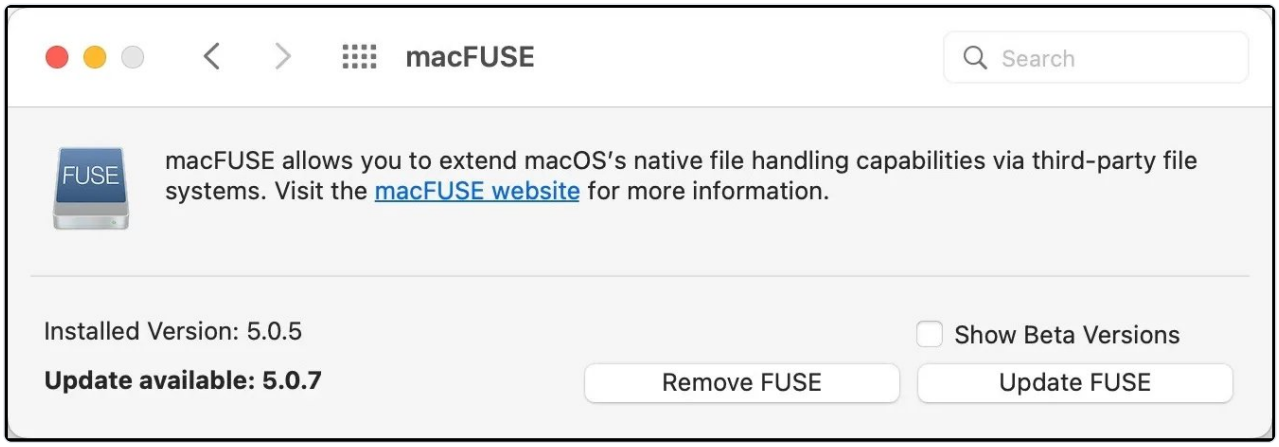
You can update and/or uninstall macFUSE within the Mac's System Preference.

Monterey (12) & Older

For macOS versions Monterey and prior, you will find the macFUSE extension at the bottom of the **System Preferences** window:

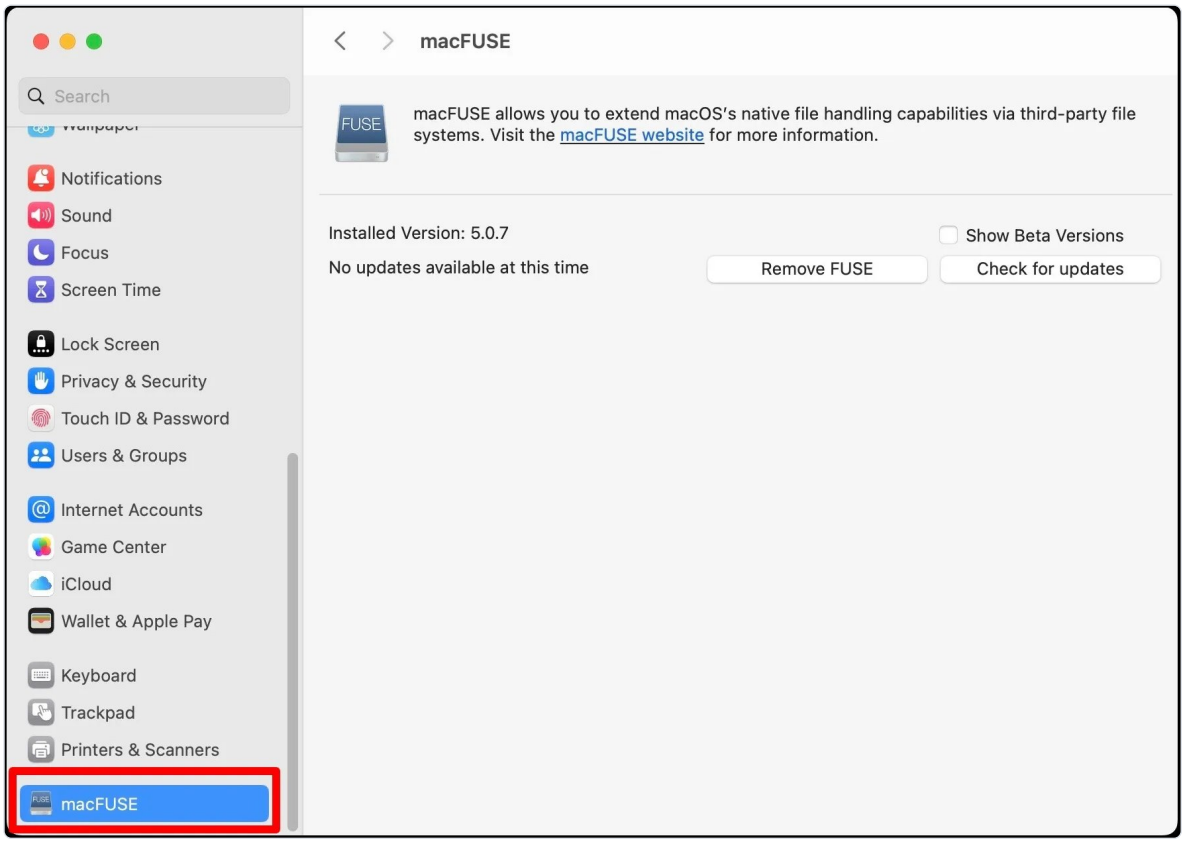


Click the macFUSE icon to see options to **update** and/or **uninstall** macFUSE:



Ventura (13) & Newer

For **newer macOS versions**, you will find the macFUSE extension listed at the very bottom within **System Settings**:


Search
Notifications
Sound
Focus
Screen Time
Lock Screen
Privacy & Security
Touch ID & Password
Users & Groups
Internet Accounts
Game Center
iCloud
Wallet & Apple Pay
Keyboard
Trackpad
Printers & Scanners
macFUSE
macFUSE
FUSE
macFUSE allows you to extend macOS's native file handling capabilities via third-party file systems. Visit the macFUSE website for more information.
Installed Version: 5.0.7
No updates available at this time
Show Beta Versions
Remove FUSE
Check for updates

Feature Guide: macOS Native App

This guide provides details on unique and noteworthy features that are specific to the **macOS Native App** of Object Mount.

Features that are global and apply to **all OS versions** of Object Mount (macOS, Windows, Linux) can be found in the Global Features section below.

Fast Copy and Fast Move

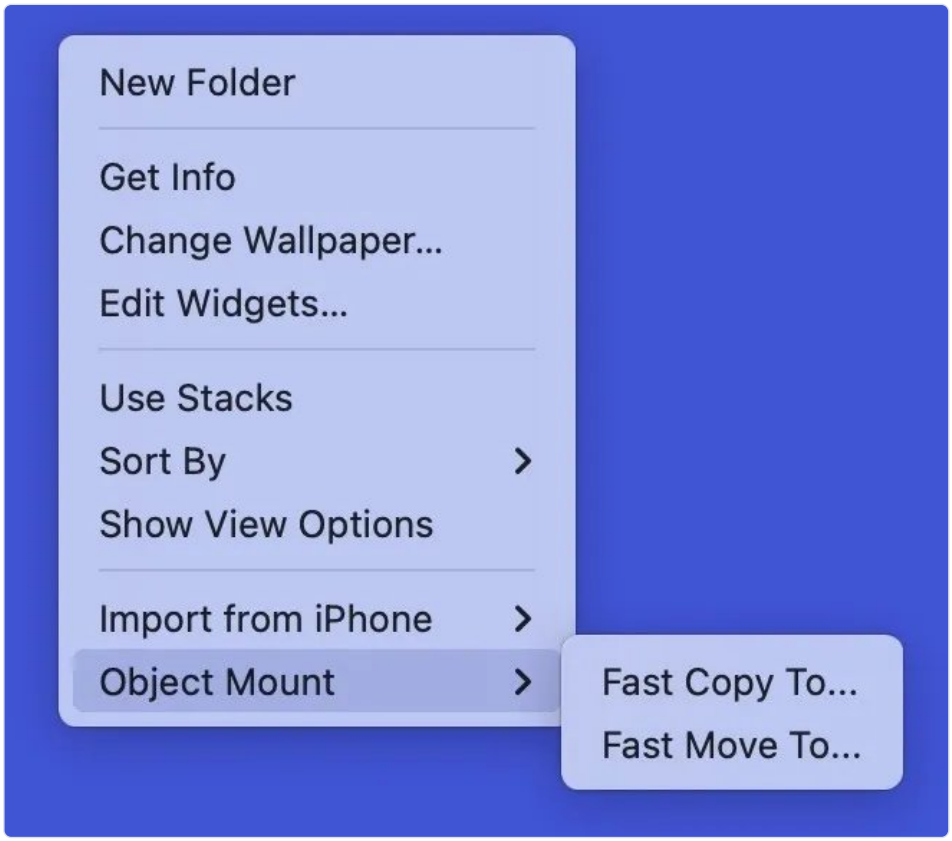
Starting with **v1.0.4**, Object Mount for macOS now supports **Fast Copy** and **Fast Move** options directly from the Mac Finder.

- **Note:** macOS 13 Ventura (or later) is required to use Fast Copy and Fast Move.

Fast Copy and **Fast Move** provide a much faster way to upload files to your object storage — ideal when working with large media assets or entire folder trees of content.

Once enabled, two new context-sensitive menu options will be displayed when you right-click on a file or folder on the Mac's Desktop or within the Finder:

- **Object Mount > Fast Copy To...**
- **Object Mount > Fast Move To...**

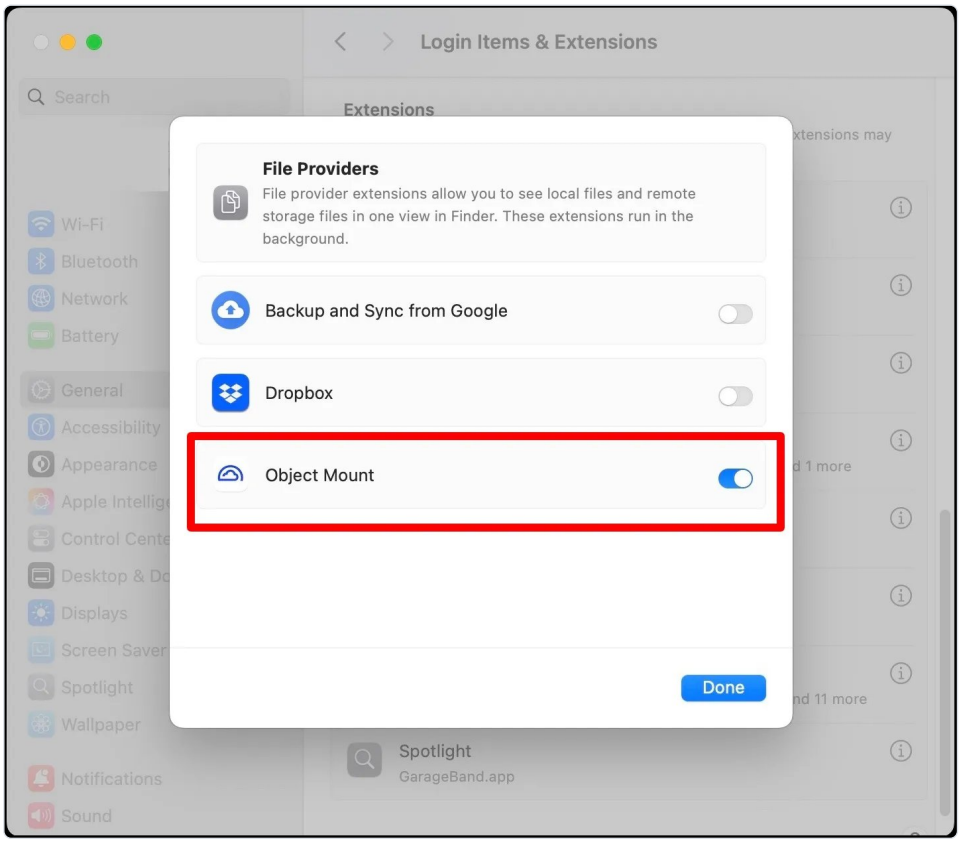


Enabling Fast Copy and Fast Move

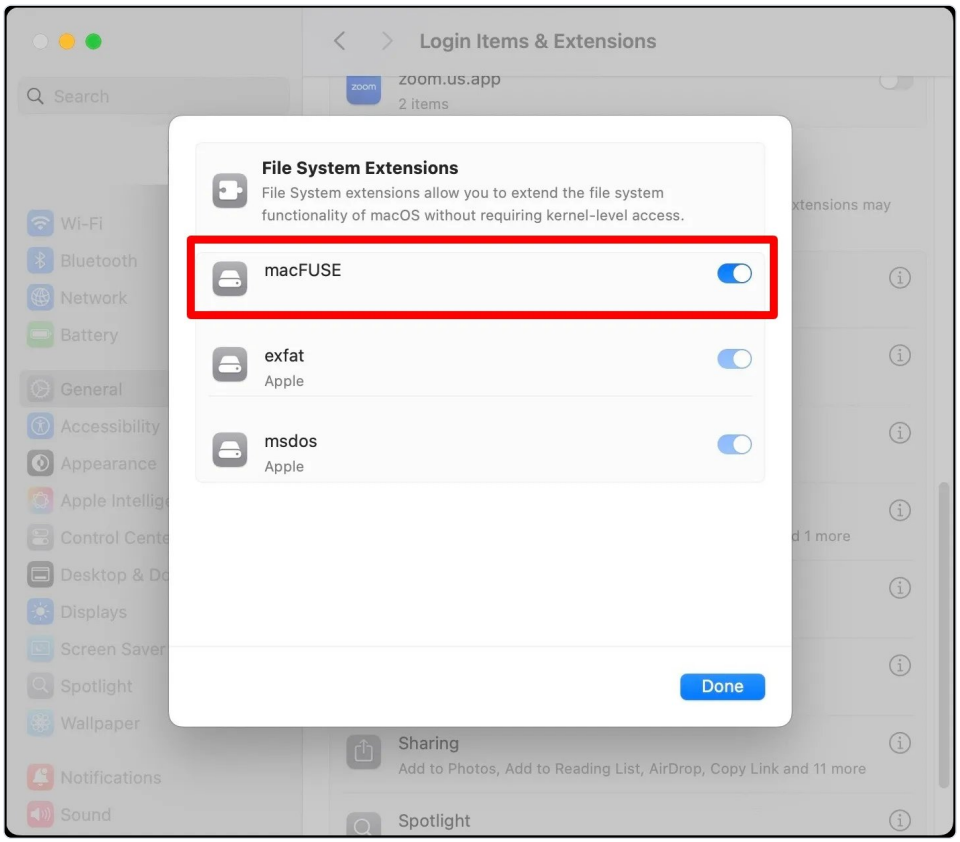
Fast Copy and Fast Move are enabled through a Finder Extension. The extension must be manually activated.

Follow the steps below to enable Fast Copy and Fast Move in **two** places within the Mac's System Settings:

1. Open **System Settings**.
2. Navigate to **General → Login Items & Extensions**.
3. Scroll down to the **Extensions** section.
4. Click the **Info Circle** `(i)` next to **File Providers**.
    - Activate the slider for **Object Mount** and click **Done**.



5. Click the **Info Circle** `(i)` next to **File System Extensions**.
    - Activate the slider for **macFuse** and click **Done**.



Once enabled, you will see the new “right-click” options in the Finder.

> **NOTE**
>
> **“Right-Click” Options**
>
> You may need to right-click more than once to activate the extension for the first time. If necessary, reboot your Mac to see the new menu items.

When to Use Fast Copy or Fast Move

Using **Fast Copy** or **Fast Move** is ideal when you want to:

- Move large files (media footage, projects, renders, etc.) or large folder trees from a local disk to a mounted S3 bucket
- Copy large files or folders from one Object Mount volume to another Object Mount volume
- Improve performance on high-latency networks (or slow remote S3 endpoints)
- Avoid the bottlenecks of the Finder’s standard file operations

How Fast Copy and Fast Move Work

Standard **copy and move** actions performed by the Finder use the native macOS file system’s I/O calls. These standard file operations copy data **one block** at a time, without any awareness of how cloud storage works. This can lead to slow performance — especially for large files or nested folder structures.

Fast Copy and **Fast Move** bypass these standard OS-based read/write calls and instead:

- Use Object Mount’s **optimized S3 transfer logic**
- Support **multipart and parallel uploads**
- Automatically manage metadata and cloud layout

- Integrate directly with your mount's caching settings

Behind the scenes, Fast Copy and Fast Move:

- Leverage both the Finder Extension and Object Mount's core engine
- Use **direct S3 transfer** logic, not shell scripting or AppleScript
- Respect your mount settings (e.g.: POSIX mode, cache location)
- Log operations to the standard Object Mount log file

Use Cases for Using Fast Copy and Fast Move

- Store original media locally, then **Fast Move** it to the cloud when ready.
- For large archive folders, use **Fast Copy** to quickly duplicate them into your S3 bucket.
- Use **Fast Move** to free up local storage when archiving to the cloud.

Global Features & Concepts

See the many articles in the Appendix for additional details on features that apply to Object Mount on *all* operating systems (macOS, Windows, and Linux):

- Object Storage Core Concepts
- POSIX explained
- S3 Credential Management
- Supported S3 Providers
- Performance Tuning
- Troubleshooting Guide
- FAQ
- Release Notes

Windows Installation and Usage

This section contains everything you need to install and use the native **Object Mount** app on Microsoft Windows.

Whether you're setting up a single workstation or deploying across a team, these pages will walk you through installation, configuration and day-to-day use.

What's Covered

- Native installer and system requirements
- User guides for mounting, configuring, and managing your object storage
- Tips and recommendations for unique Windows features, media applications, and performance improvements

Next Steps

Start with the **Installation Guide**:

- Windows Installation Guide

Move on to the **User Guide** for instructions on creating mounts and working with object storage in your daily workflows:

- Windows User Guide

Lastly, review the Windows OS-specific **Feature Guide** and the appropriate **Media Workflow Guide**:

- Windows Feature Guide
- Media Workflow Guides

Installation Guide: Windows Native App

This section covers how to install the the Object Mount native application for Windows using a standard `.exe` installer.

System Requirements

Before you begin, make sure your PC meets the following system requirements:

- Windows 10 or later (or Windows Server 2016 or newer)
- Intel x86_64 or AMD processor (no ARM CPU support at this time)
- At least 300MB of free disk space
- Internet connection for mounting cloud-hosted buckets, or LAN connection for mounting local buckets

> **NOTE**
>
> **S3 Provider Support:**
>
> Object Mount for Windows fully supports S3 container services from Amazon Web Services, Storj, and other fully S3-compatible Object Storage providers.
>
> - See the Supported S3 Providers article in the Appendix for a complete S3 provider list.
>
> **Azure & GCS Support:**
>
> - Support for *Native* Microsoft Azure Storage and *Native* Google Cloud Storage services are still under development in Object Mount for Windows.
> - For Google Cloud access: Using GCS *S3* is recommended instead of using GCS *S3 Gateways*.
> - Native Azure and Native GCS (S3 Gateways) should be considered “experimental” at this time.

Step 1: Download the Latest Installer

Download the latest Object Mount `.exe` Windows installer program.

Note: Storj customers have 2 free licenses included. To obtain additional licenses or if you are not a Storj customer, reach out to your 🌐 Storj Representative.

Step 2: Install the Object Mount App

Note: Administrative permissions are required to install some of the Object Mount components.

1. Double-click the setup `.exe` program to start the installation.
2. Follow the prompts to complete the installation process.

Next Steps

Proceed to the Windows User Guide to **activate** your Object Mount installation, **configure** your object storage credentials, and **create** your first mount.

Then review the Windows OS-specific Feature Guide.

Finally, read through usage tips for your specific Media Workflows and Applications.

User Guide: Windows Native App

This guide walks you through the core steps to get started with **Object Mount** on Windows, including: activation, adding credentials, and mounting your cloud-based object storage as a local drive.

Step 1. Launch Object Mount

- Run Object Mount from the Start Menu or Desktop shortcut.
- If prompted, click **Allow Access** to permit firewall changes.

Step 2. Configure S3 Credentials & Create a Mount

Object Mount can connect to object storage from Storj as well as other major cloud storage providers. Standard S3-compatible credentials from your cloud storage provider are required.

Obtain your Object Storage Credentials

- See the Appendix article for instructions.

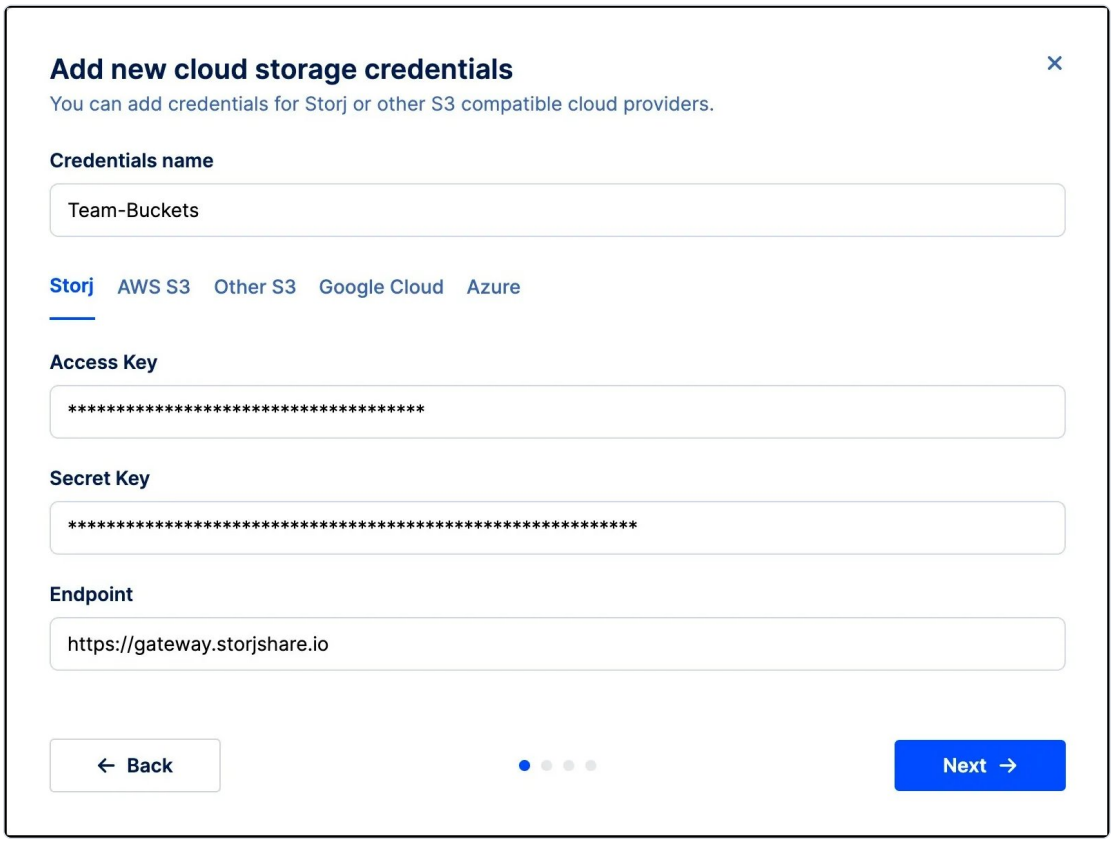
Enter Your Credentials and Create a Mount

There are two easy paths to create a mount — both paths achieve identical results and both require your object storage credentials:

- Click **Create New Mount** from the **Mounts** tab, or
- Click **Add New Credentials** from the **Credentials** tab
- Then select and configure one or more buckets to mount as a local drive

Follow the steps below to create a new mount using the **Add New Credentials** path:

1. Select the **Credentials** tab in Object Mount.
2. Click the **Add New Credentials** button.
    - Give the credential set a clear name
    - Choose your provider tab (e.g.: Storj, AWS, etc.).
    - Enter the **Access key ID**, **Secret key**, and **Endpoint** (or Region, if applicable) that you obtained earlier.


Add new cloud storage credentials
You can add credentials for Storj or other S3 compatible cloud providers.
Credentials name
Team-Buckets
Storj
AWS S3
Other S3
Google Cloud
Azure
Access Key
****
Secret Key
****
Endpoint
https://gateway.storjshare.io
Back
Next

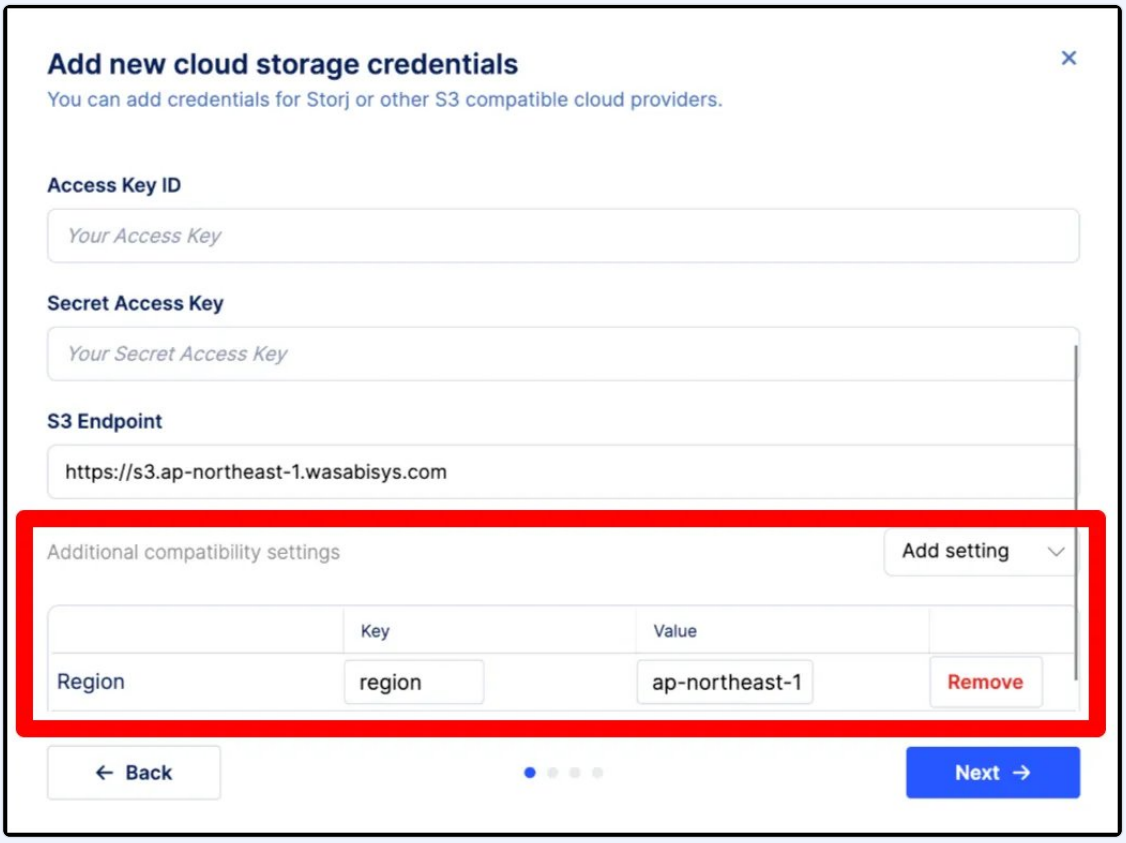
NOTE

S3 Compatible Providers

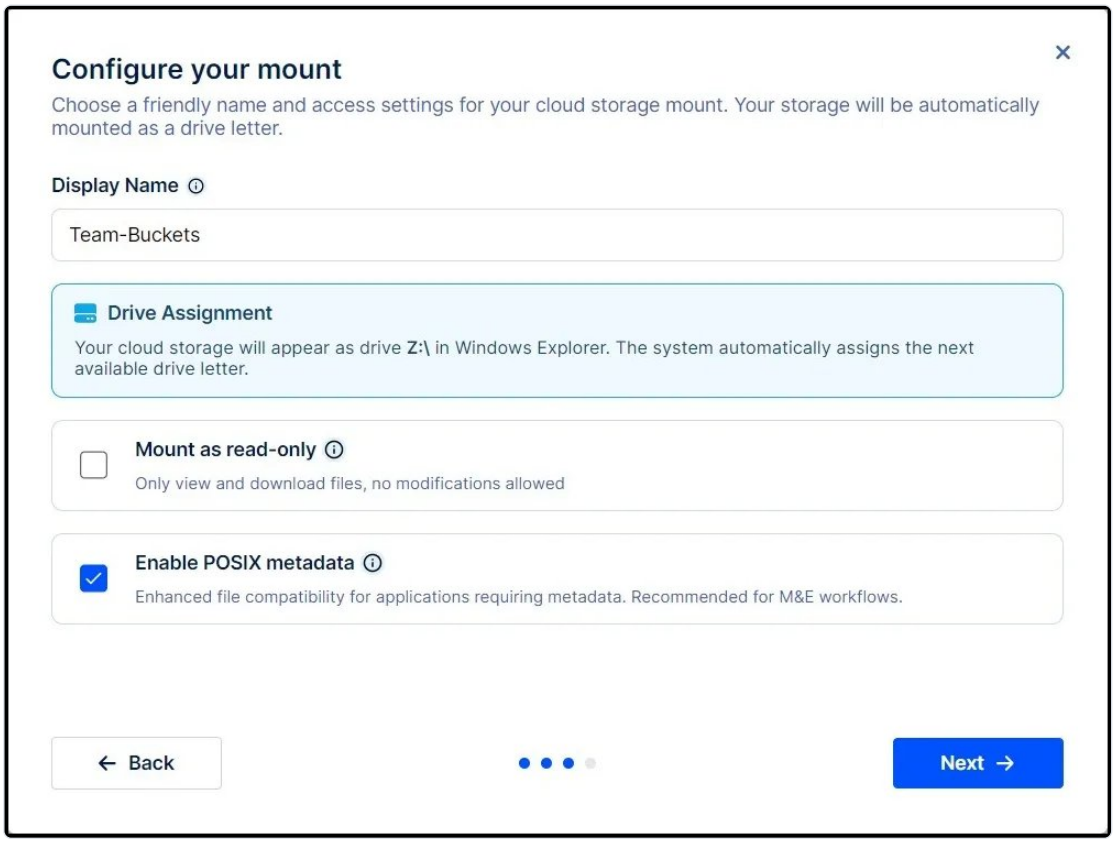
If your provider is not listed, select **Other S3** and choose the provider from the dropdown.

- Ensure the endpoint is accessible from the Internet.
- Some providers may require extra compatibility settings like a defined region.

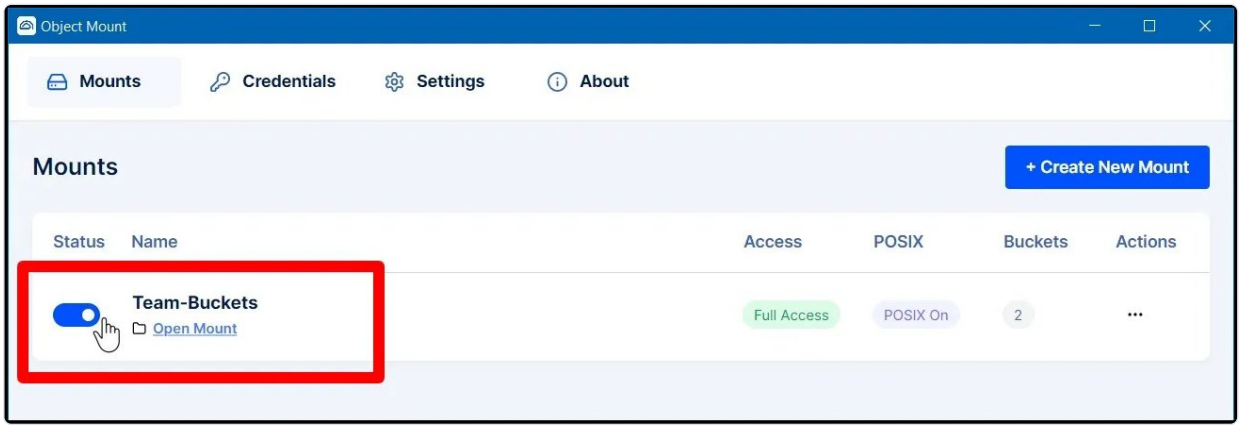
Below is an example of providing an additional S3 Compatibility Setting for region:



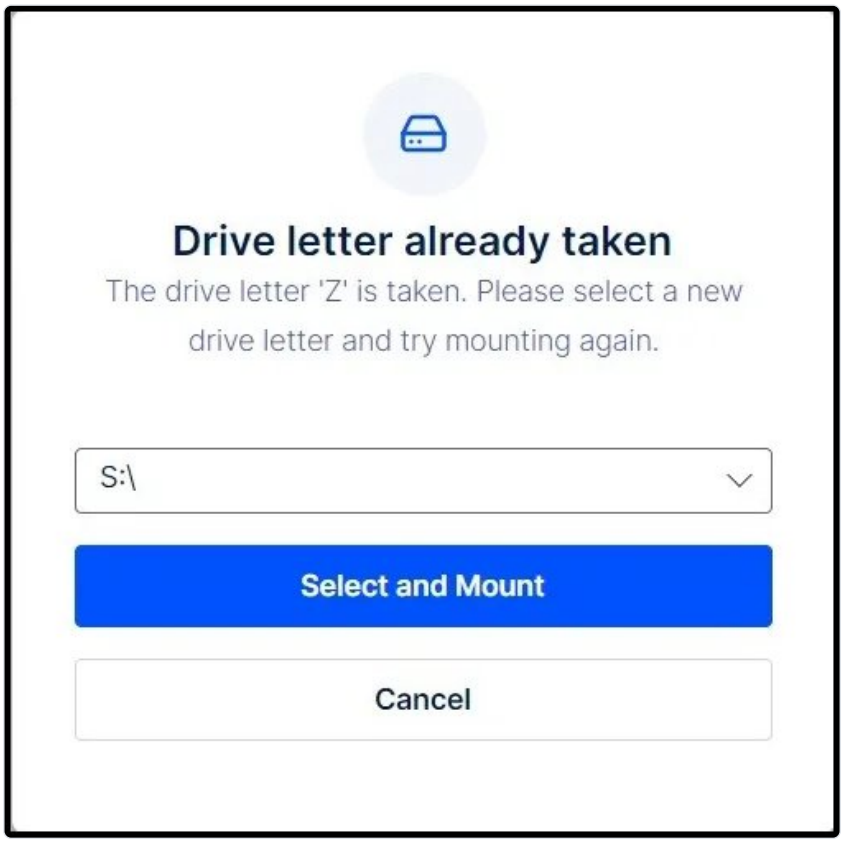
3. Click **Next**.
   - Object Mount will authenticate to your cloud storage provider and attempt to discover any buckets your credentials have access to.
4. Select **Mount all buckets** or choose one or more specific buckets to create a mounted drive for. Click **Next**.
5. Enter a **Display name** for your mount.
   - Choose a **Drive Letter** for the mount. The default, **Automatically assign**, picks a free drive letter each time the mount is enabled.
   - If you pick a specific letter and it is in use when you enable your mount, you will be prompted to select a new drive letter (see step 8, below).
6. Tick the **Mount as read-only** box if all you need is to browse and copy content.
   - Uncheck the box to enable **full read/write access**.
7. Tick the box to **Enable POSIX metadata** (recommended).
   - For more details, see the Appendix article: POSIX Explained.



8. Click **Next** to create the mount. Then enable the mount from the **Mounts** tab.



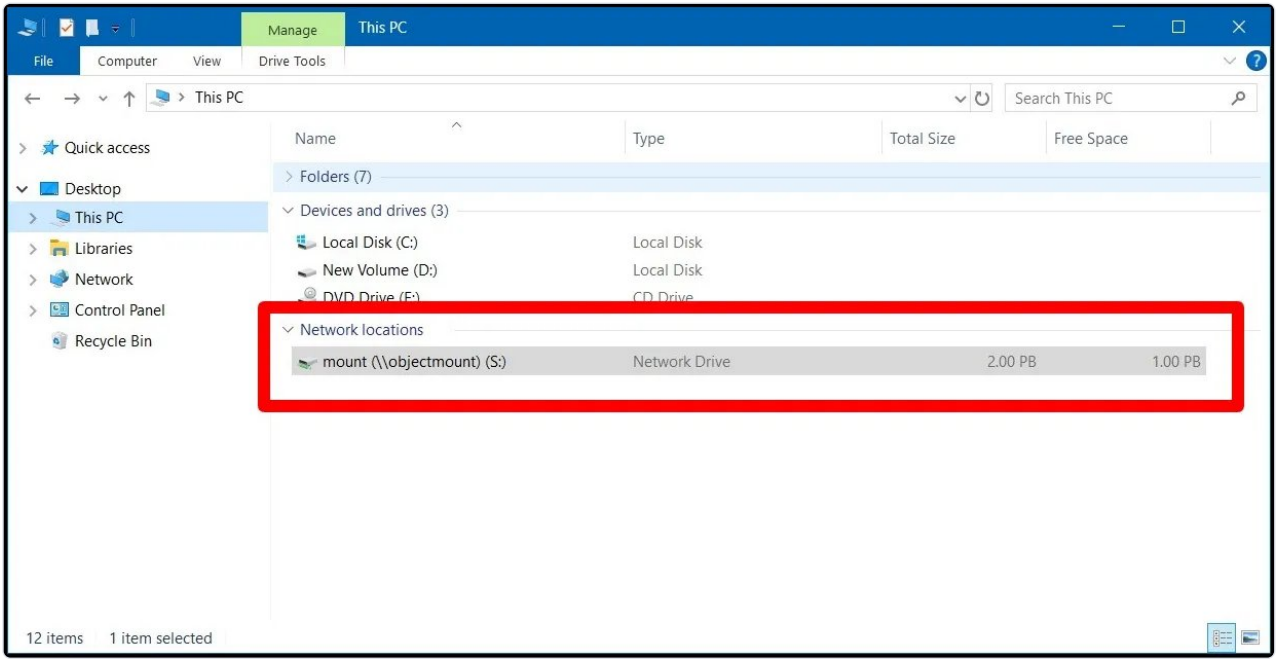
   - If you picked a specific drive letter and it is already in use, you will be prompted to select a new drive letter:



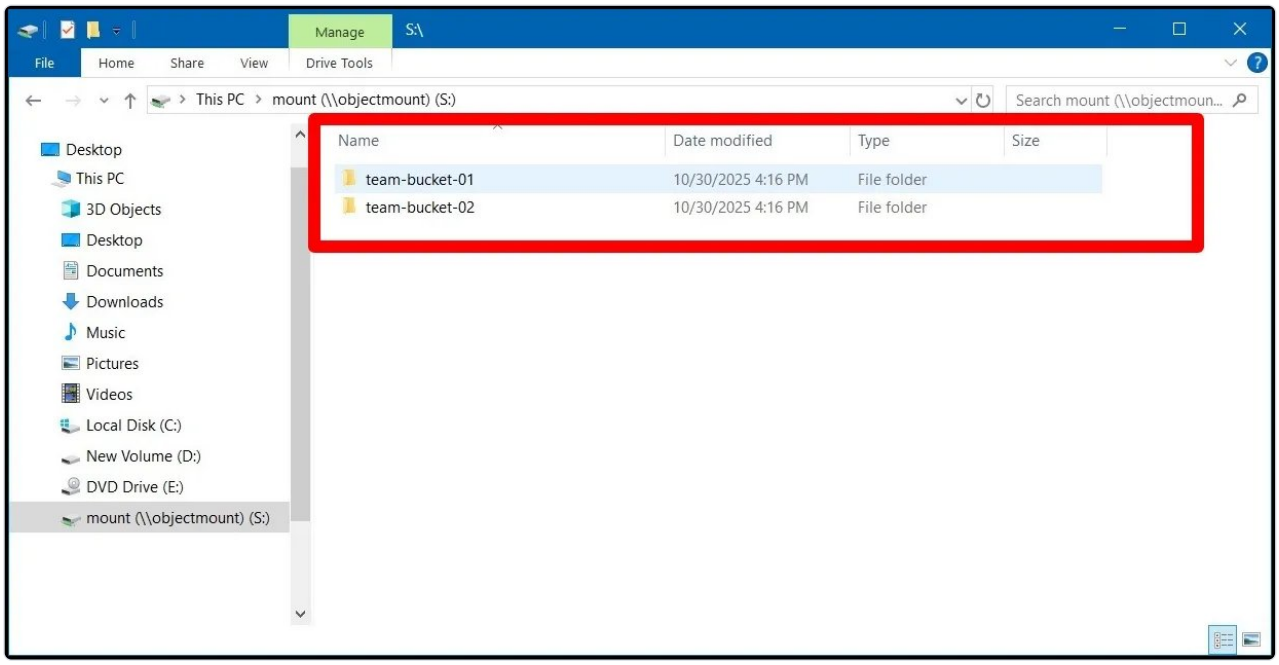
Note: First-time access to a new mount may take a few moments if the bucket contains many files — performance improves after the initial mount.

Newly Mounted Drive & Buckets

Your newly mounted drive will appear on your PC using the assigned drive letter:



If there are multiple buckets accessible using the same credential set, those buckets will each appear as separate **folders** within the single mounted drive letter:



Preferences & Advanced Settings

Object Mount includes a flexible **Settings** panel where you can fine-tune caching, logging, and performance settings across all mounts.

> **NOTE**
>
> **Settings**
>
> - These settings are global and apply to all mounts.
> - These settings cannot be edited while a mount is active. Disable all mounts before making changes.

Refer to the following sections for details on each setting:

General

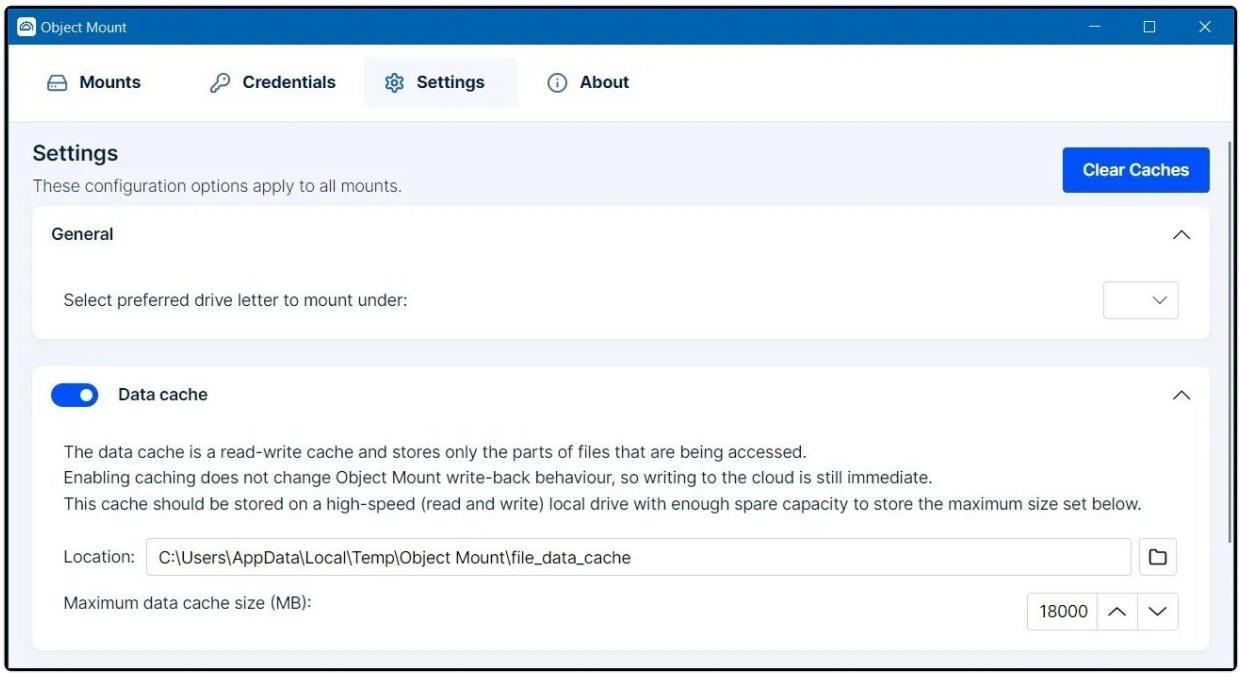
- **Confirm before exiting if mount is active:** Shows a confirmation dialog when you exit Object Mount while a mount is active.

> **NOTE**
>
> **Multiple Mounts**
>
> Several mounts can be active at the same time. A mount cannot be enabled while another active mount uses the same drive letter or an overlapping cloud path, such as the same bucket.

Data Cache

The local **Data Cache** temporarily stores small portions of accessed cloud files on your local disk — useful for repeated or random access operations.

- **Location:** The local path where data cache files are stored
- **Maximum cache size:** The upper limit (in MB) for total cached data
  - **Note:** This feature does *not* delay cloud writes — writing to the cloud happens immediately.



NOTE

Data Cache Recommendations

- Use a fast local drive (SSD or NVMe) for best caching performance.
- If the connection to your object storage is on a high-speed, low-latency network, you may see improved performance by turning caching *off*.

Metadata Cache Settings

Metadata caching helps Object Mount load and retain file structure, timestamps, and permissions more efficiently.

- **Location:** The local directory for storing metadata information
- **Cache timeout:** The duration, in seconds, that metadata remains valid before being refreshing from the cloud (default: 60 seconds)

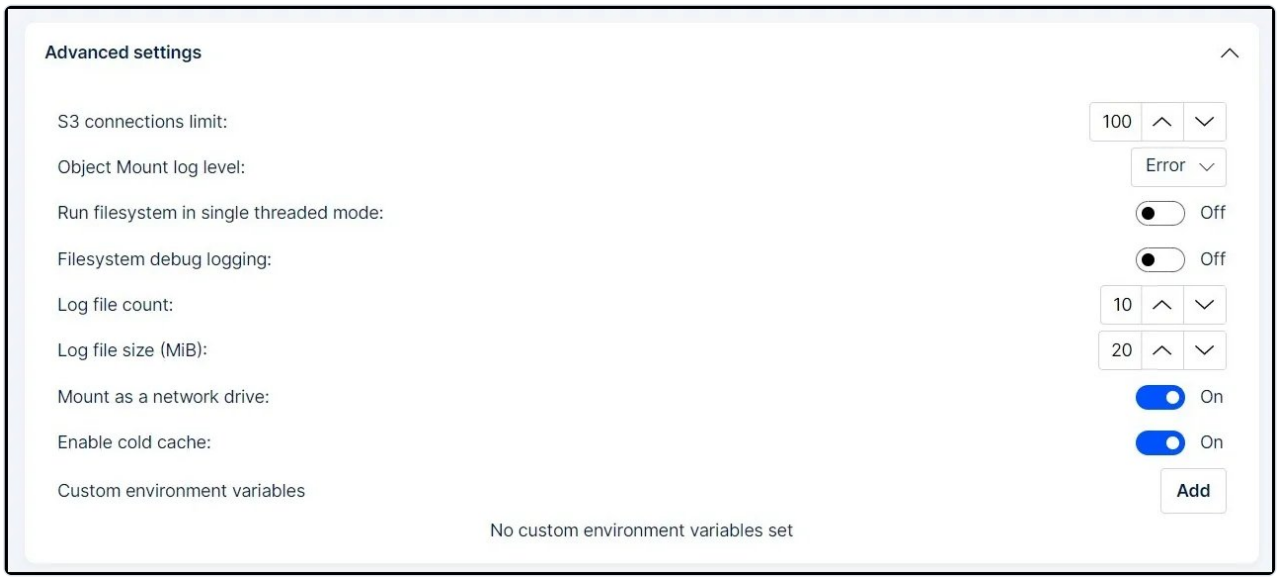


Advanced Settings

These options let you further customize Object Mount's performance and diagnostic behavior.

Setting	Description
S3 connections limit	Controls the number of concurrent S3 operations (default: 100).
Log level	Adjusts logging detail. Options: `error`, `warning`, `debug`, or `trace`.
Single threaded mode	Runs the filesystem in single-threaded mode (rarely needed; leave `off` unless instructed by Storj Support).

Setting	Description
Debug logging	Enables extensive debug output to assist with support issues.
Custom Environment Variables	Custom variables to modify Object Mount's behavior, add variables when recommended by Storj Support



NOTE

Log File Location

On Windows the default location for storing the log file is:

`C:\Users\%username%\AppData\Local\Object Mount\cunofs.log`.

Click **Apply** to save any changes. Then re-enable your mount.

Click **Reset** to revert all settings back to their default values.

Feature Guide: Windows Native App

This guide provides details on unique and noteworthy features that are specific to the **Windows Native App** of Object Mount.

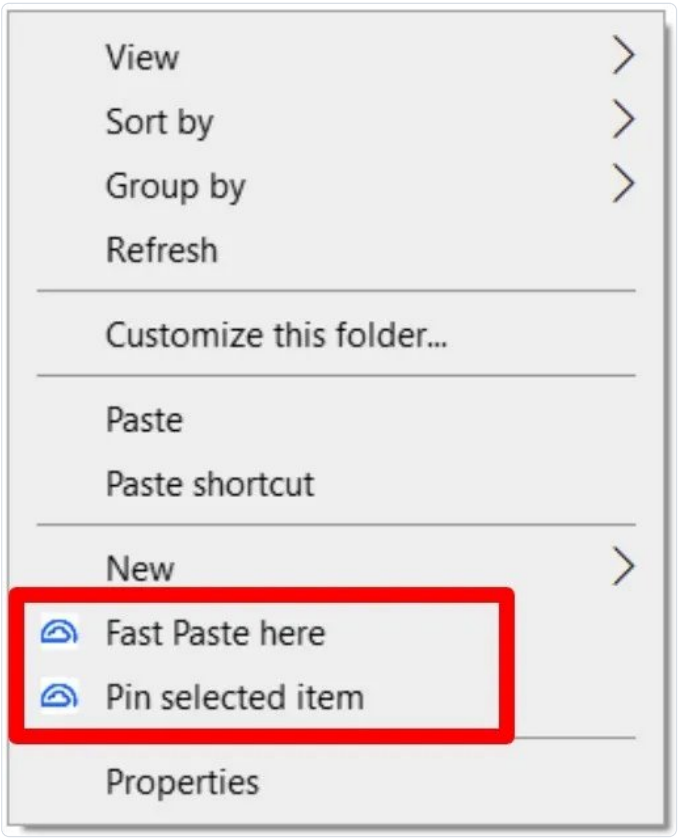
Features that are global and apply to **all OS versions** of Object Mount (macOS, Windows, Linux) can be found in the Global Features section below.

Fast Paste on Windows

Fast Paste provides a much faster way to copy or move files to and from your object storage — ideal when working with large media assets or entire folder trees of content.

To use **Fast Paste**:

1. Copy a file or folder as normal, using **CTRL+C** or **right-click + Copy**. To move it instead, use **CTRL+X** or **right-click + Cut**.
2. Navigate to the destination directory. Either the source or the destination must be on an Object Mount drive.
3. Right-click and choose **Object Mount > Copy items here** (or **Move items here** after a Cut). The menu is shown with a white and blue Object Mount icon.



When to Use Fast Paste

Using Fast Paste is ideal when you want to:

- Move large files (media footage, projects, renders, etc.) or large folder trees from a local disk to a mounted S3 bucket
- Copy large files or folders from one Object Mount volume to another Object Mount volume

- Improve performance on high-latency networks (or slow remote S3 endpoints)
- Avoid the bottlenecks of Windows' standard file operations

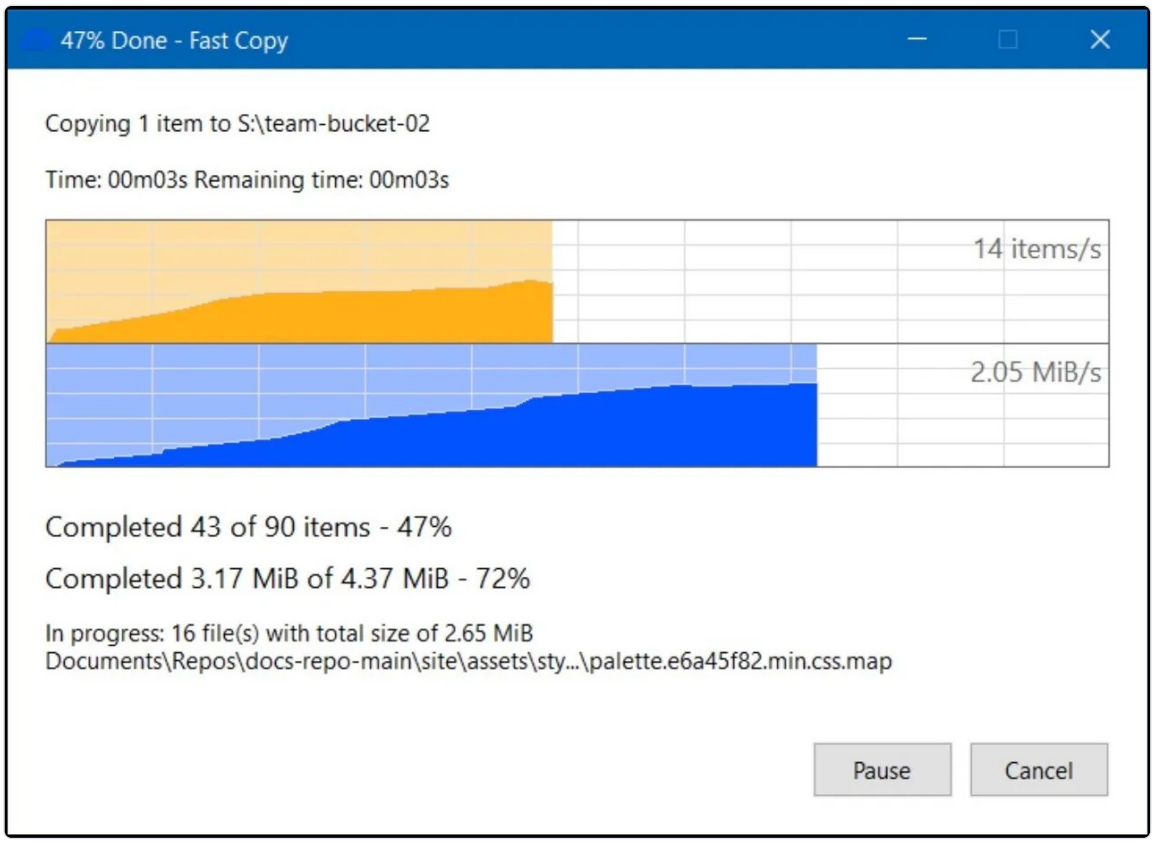
How Fast Paste Works

Standard **copy and paste** actions performed by Windows (Ctrl+C/Ctrl+V, or right-click + Copy/Paste) invokes Windows' built-in file I/O calls. These standard file operations copy data **one block** at a time, without any awareness of how cloud storage works. This can lead to slow performance — especially for large files or nested folder structures.

Fast Paste bypasses the standard OS-based write calls and instead:

- Reads the files from the Windows clipboard using native Windows I/O calls
- Streams files up to the mount using **multipart-aware** and **concurrent** upload operations tuned for S3-compatible storage

A live **progress indicator** will be displayed when you use Fast Paste:



Additionally, **Fast Paste**:

- Respects your mount settings (e.g.: POSIX mode, cache location)
- Applies performance and tuning options (cache settings & location, multipart thresholds, etc.)
- Logs operations to the standard Object Mount log file

Use Cases for using Fast Paste

- Store original media locally, then copy and **Fast Paste** it to cloud when ready.
- For large archive folders, use copy then **Fast Paste** to quickly duplicate them into your S3 bucket.
- Use **Fast Paste** to free up local storage when archiving to the cloud.

File Pinning on Windows

Starting with **Object Mount v1.0.5**, you can now **pin files and folders** in Windows to ensure they remain in the **local Object Mount cache**.

This improves performance and is especially useful for:

- Preventing delays when reopening large media assets
- Ensuring critical project files stay cached locally

What Does File Pinning Do?

When you **pin a file**, Object Mount:

- Downloads and stores the file in Object Mount's local data cache
- Prevents the file from being *evicted* due to cache size limitation settings
- Flags the file for *persistent caching*, even across restarts

> **NOTE**
>
> **Windows File Pinning**
>
> Pinning does *not* duplicate or move the file elsewhere. It simply marks the file as *persistent* in the Object Mount local cache.
>
> **Important:** Pinned files *do not remain accessible* when Object Mount is off-line, or when your mount is disabled or disconnected. An active, live connection is still required for access, even when a file is pinned.

Pinning Files

To Pin a file or folder:

1. Right-click any file or folder *stored within* an **Object Mount volume**.
2. Select **Object Mount > Pin selected item** (shown with a white and blue Object Mount icon).
3. A small icon overlay in File Explorer will indicate the item is pinned: 🟢

Pinning a folder also pins the files in it, including files added to the folder later.

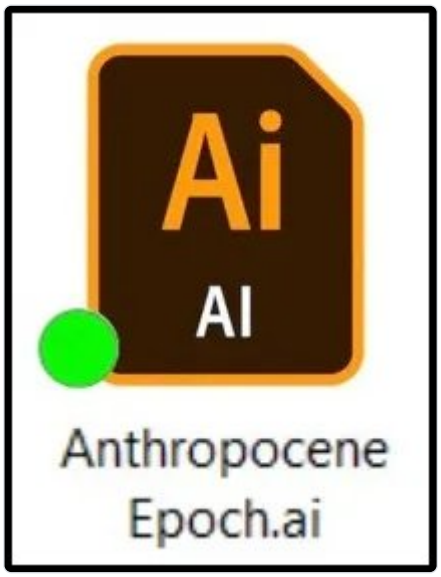
To Unpin a file or folder:

1. Right-click the item and choose **Object Mount > Unpin selected item**.

For an item inside a pinned folder, **Object Mount > Reset pinned status** returns the item to the folder's pinned status.

Pinned items can be identified by:

- Overlay icons in **File Explorer**: one for pinned items (the "green dot" 🟢), one while pinning is in progress, and one for items pinned through a pinned parent folder
- Their status in the **Object Mount logs**



Note: A restart may be necessary before you will see the “pinned file” indicator.

Pinned Files & Cache Space

- Pinned files do *not* count toward the maximum data cache size set in the Settings tab of Object Mount.
- However, pinning requires that the **Data Cache** be enabled.
  - If the Data Cache is disabled, the pinning menu items will no longer be displayed.
  - When the Data Cache is re-enabled, any previously pinned files will retain their pinned status.

> **WARNING**
>
> **Clearing the Cache**
>
> The **Clear Cache** button in the Settings tab of Object Mount opens a dialog with several options:
>
> - **Clear Disk Cache** removes cached data but keeps pinned items.
> - **Clear Everything** removes the metadata cache, the disk cache and all pinned items. Pinned items revert to an “unpinned” status.
> - **Manage Pinned Items** lets you unpin all items or selected items.

Best Practices for Using Pinning

Use Pinning to:

- Keep critical assets readily available without re-downloading
- Avoid cache eviction for large files you frequently revisit
- Preload media for use in slow-bandwidth environments

Global Features & Concepts

See the many articles in the Appendix for additional details on features that apply to Object Mount on *all* operating systems (macOS, Windows, and Linux):

- Object Storage Core Concepts
- POSIX explained
- S3 Credential Management
- Supported S3 Providers

- Performance Tuning
- Troubleshooting Guide
- FAQ
- Release Notes

Linux Installation and Usage

This section contains everything you need to understand, install and use **Object Mount** on a Linux machine.

Whether you're setting up for creative workflows, remote storage access, or team-based collaboration, these pages will walk you through installation, setup, and day-to-day use.

Linux Package

Object Mount is a scalable, high-performance POSIX compatibility layer that lets you interact with files stored on object storage such as Amazon S3, Storj Object Storage, Azure Blob Storage, Google Cloud Storage, or any S3-compatible object store hosted in the cloud or locally.

Object Mount for Linux retains much of its original look and feel from its **cunoFS** heritage.

Object Mount for Linux includes an Object Mount Command Line Interface (CLI), `cuno`, providing the highest performance and most straightforward way to interact with object storage. You will encounter many `cuno`-related commands, settings, and custom variables throughout your configuration and daily use of Object Mount for Linux.

Object Mount functions via a user-mode library, `cuno.so`, which acts as a 🌐 dynamic linker/loader intercepting applications (both dynamic and static) via fast, dynamic binary instrumentation.

Object Mount: OS Version Differences

Storj's Object Mount for Linux, with its CLI-heavy user interface, is in stark contrast to the Graphical User Interface (GUI) experience that Object Mount presents to Windows and macOS users.

Other OS differences include:

Object Mount Feature	For Linux	For Windows & macOS
CLI-based	100%	Limited CLI
GUI-based	No GUI available	100%
POSIX Mode Features	Wide array of settings	Either `on` or `off`
`cuno`-based commands, settings, and variables	Large quantity of options	Limited options

What's Covered

The many articles in this Linux section will cover the installation, configuration, and use of Object Mount on multiple flavors of Linux. Navigate to what you need, when you need it.

Object Mount's Linux-specific content includes:

- Getting Started Guides
- Installation Guides
- User Guides
- Advanced Guides

Click below to access the content you are looking for:

Getting Started Guides

Prior to installing Object Mount for Linux, view the Linux-specific Getting Started Guide articles to better understand the capabilities, options, and deployment choices available.

Getting Started topics include:

- Understanding POSIX Compatibility and Options in Object Mount for Linux
- Understanding Deployment Modes for Object Mount for Linux

Installation Guides

When *ready to install*, view the Object Mount for Linux Installation Guides for your particular Linux variant, including:

- Alpine
- Debian & Ubuntu
- Red Hat
- RHEL & CentOS
- glibc (standard C library)
- musl (standard C library)
- Windows - via Windows Subsystem for Linux (WSL)

User and Configuration Guides

After installation, read through the Linux User Guides to learn more about using and configuring Object Mount for Linux, including details on:

- Validating Functionality & Access Object Storage
- Activating License Keys
- The different Linux Deployment Modes
  - Direct Interception Mode
  - Object Mount on FUSE
  - Object Mount FlexMount
  - Object Mount Fusion
- Enabling Enforced POSIX File Access
- Cloud Path Details
- Application Tips
- Media & Entertainment and High Performance COmputing Modes

Advanced Guides

Once up and running, explore the Linux-specific *advanced configuration tips* found in the Object Mount for Linux Advanced Guides. These articles will help you fine-tune your deployment, improve Object Mount's performance, and increase your productivity.

Topics include:

- Advanced Credential Options
- Advanced Loading Options

- Advanced Configuration Options
- Server-Side Encryption
- Kubernetes CSI Driver
- Machine Learning Accelerator
- Technical Limitations
- Uninstalling Object Mount

Additional Reference Material

Once your Linux deployment of Object Mount is up and running, there are several **Global Features** and **Media Workflow Guides** that you can reference, including:

Global Features & Concepts:

See the many articles in the Appendix for additional details on features that apply to Object Mount on *all* operating systems (macOS, Windows, and Linux):

- Object Storage Core Concepts
- POSIX explained
- S3 Credential Management
- Supported S3 Providers
- Performance Tuning
- Troubleshooting Guide
- FAQ
- Release Notes

Media Application & Workflow Guides:

Detailed usage tips and recommendations for Media Workflows, including:

- Using Object Mount with Adobe Premiere Pro
- Using Object Mount with Avid Media Composer
- Using Object Mount with DaVinci Resolve

Getting Started Guides for Linux

The pages in these **Getting Started Guides** for Linux will introduce you to topics that should be understood *before* installing and using Object Mount.

Deployment & Configuration Options on Linux

When configuring Object Mount on Linux, it is important to understand the needs of the **tools** and **workloads** you plan to run.

You can adapt settings for each workflow or tool, or you can set options at the bucket-level to impose a particular usage mode for all users.

Deployment Considerations

There are two important, but independent, aspects to consider when configuring Object Mount for Linux:

1. The requirements for your tools and applications to work *correctly*:
    - For **correct operation**, you need to know what level of **POSIX Compatibility** your tools require in order to function.
    - Object Mount for Linux offers multiple POSIX compatibility options giving the end user a wide array of tools to fine-tune and tailor the functionality and performance of Object Mount for their specific use case and environment.
    - See the article on POSIX Options to understand the many different POSIX compatibility choices available to you.
2. Ensuring your specific tools and environment work *efficiently*:
    - For **efficient operation**, the bandwidth and latency between your compute resources and your object storage resources will help guide your selection of the best **Deployment Mode** for Object Mount.
    - Object Mount for Linux supports multiple Deployment Modes supporting a variety of unique bandwidth, user requirements, and environmental situations.
    - See the Deployment Modes article to understand the different deployment choices available to you.

POSIX Options

POSIX Compatibility

For Object Mount to function correctly, it needs to comply with the level of POSIX compatibility that your tools require.

Some Linux applications have only limited or minimal requirements from a POSIX perspective, and may only use the file system for basic operations like renaming, reading, or writing new data.

Other applications may require a fully POSIX-compatible interface, which invokes additional services within Object Mount's compatibility layer.

- **Note:** Enabling full POSIX compatibility requires extra metadata to be written to object storage and may have a slight effect on performance.

POSIX Modes

The levels of POSIX compatibility provided in Object Mount for Linux are:

- Core File Access — no users/groups/symlinks/hardlinks/permissions control/file modes.
- POSIX File Access — POSIX-compatible access with users/groups/symlinks/hardlinks/permission control/file modes.
- POSIX Enforced File Access — Same as "POSIX File Access" above, but POSIX access control *is enforced*.

> **NOTE**
>
> **How to check if you need to enable "POSIX File Access"**
>
> - Run your workload using a local file system volume. Then run the same workload using an Object Mount volume. Compare the outputs and results. If you find discrepancies, it is likely that POSIX File Access needs to be enabled.
> - If Object Mount running with only **Core File Access** fails with an error `(95) ENOTSUP` (not supported) or `Operation not permitted` then it is likely that POSIX File Access needs to be enabled.

Core File Access

Core File Access mode disables all POSIX compatibility features in Object Mount for linux. Core File Access mode does not support the persistence or modification of POSIX users, groups, symlinks, hard links, permissions control or file modes attributes on objects. This is the default mode.

- Use "Core File Access" when you don't have any metadata requirements and your tools don't need any POSIX metadata persistence to function correctly.
- Use "Core File Access" when you don't have write access to the bucket in question, or you don't want to create any Object Mount-generated objects there.

Example use-cases:

- Use "Core File Access" when interacting with data you have already moved up to object storage and when you only require read-only access to the filenames and content of those objects. For example, if you have machine learning datasets in the cloud and you have previously configured your libraries to read them directly from object storage.

How to enable:

- "Core File Access" is enabled as the default mode. No additional action is required.

POSIX File Access

POSIX File Access mode will generate and maintain POSIX metadata for your objects, but it *will not* enforce any permissions or modes set on the objects. This means any user can use Object Mount to read or write any file that their object storage credentials give them access to.

- Use "POSIX File Access" when you need POSIX metadata but do *not* need to enforce POSIX read/write/execute file permissions.

Example use-cases:

- Use "POSIX File Access" when your applications are dependent on the preservation of POSIX metadata (owner/group permissions, change/modify times, etc.) or POSIX "links" (symlinks or hardlinks).
- Use "POSIX File Access" if you are moving workflows from POSIX to object storage, such as workloads that were previously run on AWS EC2 with EFS.

  Note: Object Mount does not currently support POSIX ACLs or extended attributes in cloud object storage. Contact the 🌐 Storj Support team if you need assistance in providing these features.

How to enable:

There are two main ways to enable "POSIX File Access":

- If the object storage provider supports setting **tags** at the bucket level, then you can enable POSIX File Access mode for *all users* using the command `cuno creds setposix s3://your_bucket_name permissive` from a command line/shell interface. This will affect *everyone* using the bucket and will force all Object Mount users of that bucket into POSIX File Access mode. (`setposix ... true` selects the enforced mode instead; see POSIX Enforced File Access.)
- Otherwise, to enable POSIX File Access on an *individual workstation-basis* you can set a local environment variable. Enter `export CUNO_POSIX=1` from a command line/shell interface. This setting is valid per-session.

POSIX Enforced File Access

POSIX *Enforced* File Access mode will generate and maintain POSIX metadata for your objects (like "POSIX File Access" mode, above) but *will enforce* POSIX access controls on those objects.

- Use "POSIX Enforced File Access" when you want to manage what users have access to, based on the UID/GID of their UNIX user and the corresponding POSIX metadata (owner, group, mode) on files.
- This means users will encounter `access denied` errors if they try to read or write to a file/directory they haven't been given permission to (by a suitably privileged user/admin using: `chown`, `chgrp` or `chmod`).

Example use-cases:

- Use "POSIX Enforced File Access" to host a website (using NGINX or other server technologies) entirely from object storage, without any attached storage device(s) (such as EFS). "POSIX Enforced File Access" mode lets you maintain control over which files/directories the web server process (e.g.: the `nginx` user) can access.
- Use "POSIX Enforced File Access" to host an organization's user filesystem in the cloud.

  Note: Object Mount does not currently support POSIX ACLs or extended attributes in cloud object storage. Contact the 🌐 Storj Support team if you need assistance in providing these features.

How to enable:

- To enable and configure “POSIX Enforced File Access” see the detailed steps in the Configuration Guide article: .

Deployment Modes

Deployment Modes

The purpose of Object Mount is to make working with object storage as fast and seamless as using a local drive.

But each user's environment is unique — comprising different applications, workflows, teams, geography, bandwidth, access to local and remote storage options, etc.

Therefore, Object Mount for Linux is fully configurable and can be deployed in a variety of different modes to best fit your unique situation.

These deployment modes include:

1. **Direct Interception Mode**

   In Direct Interception Mode, Object Mount inserts itself between your standard Linux applications and the host Operating System. Object Mount detects and intercepts file I/O calls, then translates and redirects them, as needed, to your object storage provider.

   The Object Mount Command Line Interface (CLI), `cuno`, is used to configure Direct Interception Mode.

2. **Object Mount on FUSE**

   If Object Mount's CLI cannot be used, or you require broader support for non-standard Linux applications, Object Mount on FUSE uses Linux FUSE (Filesystem in Userspace) to mount object storage within the Linux file system hierarchy.

   This increases compatibility but can reduce performance.

3. **Object Mount FlexMount**

   Object Mount FlexMount combines the speed and performance of Direct Interception (#1, above) with the widest compatibility and application support of Object Mount on FUSE (#2, above).

4. **Object Mount Fusion**

   Object Mount Fusion is used to enhance high-performance, LAN-based *attached* storage solutions (like Amazon's Elastic File System - EFS) with the throughput of object storage.

   Examples include:

   - The client is a cloud-hosted EC2 instance or virtual machine within the same **region** as the cloud data.
   - The client is in the same LAN network as an on-premises object storage cluster and has a high-speed, low-latency connection linking the client to the storage.

Configuring Deployment Modes

After you have installed Object Mount for Linux, read through the detailed configuration and usage guidelines for these different deployment modes in the Linux User Guide article: .

Linux Installation Overview

Storj's Object Mount supports multiple flavors of Linux and can also run inside Docker and other virtual environments, such as Microsoft's Hyper-V and Oracle's VirtualBox.

Object Mount offers both **Scripted Installers** (which do *not* require root privileges to install for a single user) as well as **Package Manager** installation files for different Linux platforms.

General Installation Procedure

The general steps to install and activate Object Mount are the *same* regardless of your specific Linux distribution.

A outline of these steps include:

1. **Obtain** an Object Mount license key.
   - **Note:** If you do not yet have a license key, ask your 🌐 Storj Representative for a trial key.
   - **Note:** Storj customers have 2 free licenses included. To obtain additional licenses or if you are not a Storj customer, reach out to your 🌐 Storj Representative.
2. **Download** the latest scripted installer or installation package for your Linux distribution.
3. **Install & Activate** Object Mount.

Once Object Mount is installed, proceed to the User Guide article: to ensure you can access both public and private Object Storage buckets.

OS-Specific Installation Procedures

Equipped with your installation download link and your Object Mount license key, proceed to the installation instructions for your specific OS and preferred installation method:

Scripted Installers	Installation Guide
glibc (C standard library)	glibc Scripted Installer
musl (C standard library)	musl Scripted Installer

Package Managers	Installation Guide
Alpine (and APM, etc.)	APK Package Installer
Debian (and Ubuntu, etc.)	Debian Package Installer
Red Hat (and RPM, RHEL, CentOS, etc.)	Red Hat Package Installer

Other	Installation Guide
Windows (WSL)	Windows (via Windows Subsystem for Linux [WSL])

Other Distro? Not Sure?

Most Linux distributions use either the GNU Project's C standard library 🌐 glibc or the 🌐 musl C standard library.

Check your distribution, then follow instructions for either our glibc Scripted Installer or our musl Scripted Installer.

glibc Scripted Installer

Many Linux distributions use the GNU Project's C standard library: 🌐 glibc.

This includes: Debian, Ubuntu, RedHat and their derivatives.

Storj's Object Mount **glibc scripted installer** (a `.run` file) is compatible with all such Linux distributions.

This installer shell script allows you to install Object Mount (for the current user) *without* needing root privileges — no `sudo` required.

> **NOTE**
> **Optional: Package Installer**
>
> If you prefer a **full installer** for use with package managers, refer to the (Debian, Ubuntu, etc.) or installation instructions.

Follow the steps below to install and configure Object Mount for **glibc** platforms.

Step 1. Download the Install Script

Download the latest scripted installer.

- **Note:** If you do not have a download link, reach out to your 🌐 Storj Representative.

The glibc scripted installer is named `cuno-<version>-x86_64-glibc-installer.run`. The file `cuno-<version>-amd64-glibc-installer.run` is the same installer under the Debian architecture name; either one works.

- **Note:** Object Mount for Linux runs on 64-bit Intel and AMD (x86_64) processors. ARM-based processors are not supported at this time.

Step 2. Install & Activate Object Mount

- **Note:** If you do not yet have a license key, ask your 🌐 Storj Representative for a trial key.

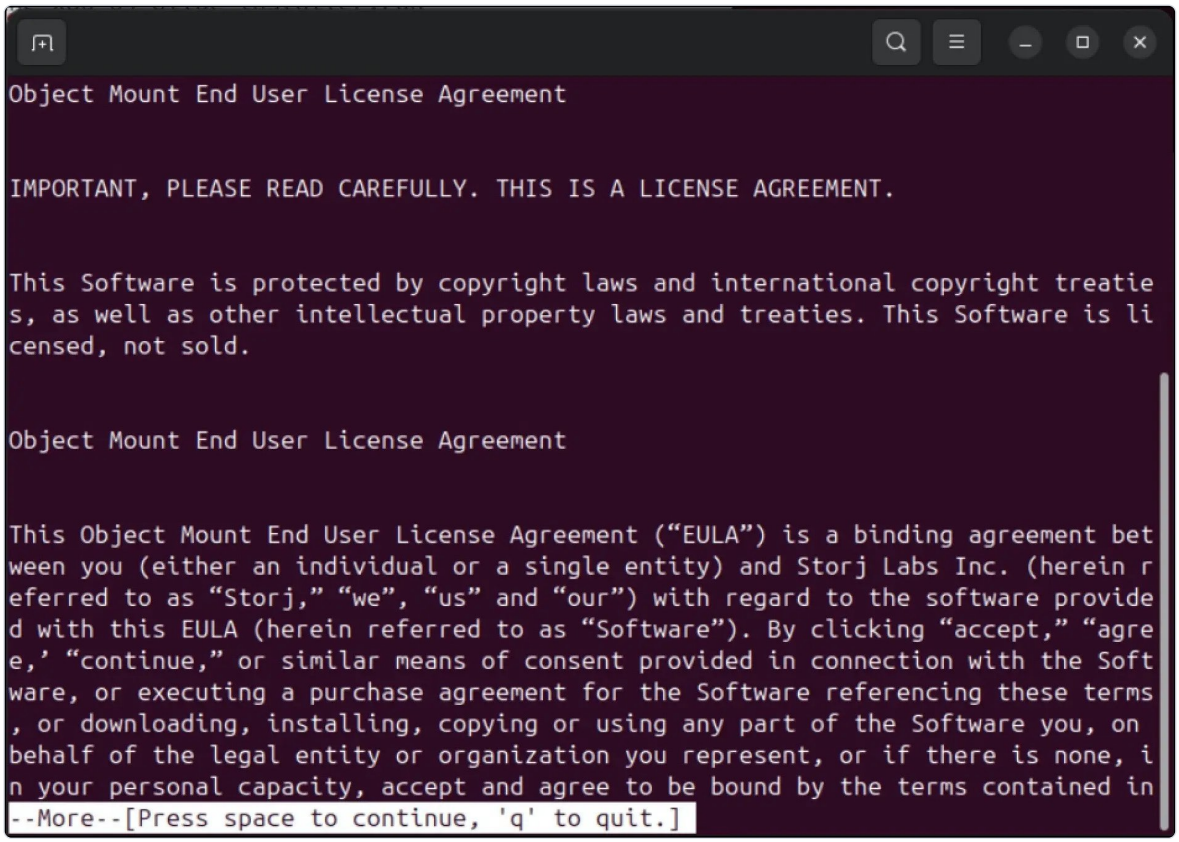
2a. To install Object Mount for use *only* by the **current user**, run the following command from a terminal window:

```
# terminal
sh cuno-<version>-x86_64-glibc-installer.run
```

2b. To install Object Mount to be used by **all users** system-wide, add `sudo` to the *front* of the command:

```
# terminal
sudo sh cuno-<version>-x86_64-glibc-installer.run
```

2c. The installation process will prompt you to read and agree to the Object Mount **end-user license agreement (EULA)**.



- **Note:** You can automate this step by setting the environment variable `CUNO_INSTALL_ACCEPT_EULA="yes"`. (This is equivalent to accepting the EULA).

2d. The installer then activates Object Mount if the environment variable `CUNO_INSTALL_LICENSE` is set to your license key or to the full path of a license key file. Set it to `none`, or leave it unset, to skip activation. With `sudo`, set `CUNO_INSTALL_LICENSE` and `CUNO_INSTALL_ACCEPT_EULA` after `sudo` (for example `sudo CUNO_INSTALL_LICENSE=/path/to/license.txt <install command>`); `sudo` drops variables set before it.

If Object Mount is not activated, the installer prints the next steps. Activate it afterwards with `cuno creds activate`; see .

Installation Locations: Scripted Installers

When Object Mount is installed via a **Scripted Installer** (e.g.: glibc or musl), components are installed to different locations depending on whether it was installed by a root, or by a non-root user.

Install Method	Install directory	Symlinks created
Non-Root User	`~/.local/opt/cuno/`	`~/.local/bin/cuno`
		`~/.local/lib/x86_64-linux-gnu/cuno.so`
		bash completions in `~/.local/share/bash-completion/completions/`
Root User (`sudo`)	`/opt/cuno/`	`/usr/bin/cuno`
		`/usr/lib/cuno.so`
		bash completions in the system bash-completion directory
		zsh completions in `/usr/share/zsh/vendor-completions/`

Next Steps

Once Object Mount is installed, proceed to the Linux User Guide article: to ensure you can access both public and private Object Storage buckets.

musl Scripted Installer

Some Linux distributions (including Alpine) use the musl C standard library: 🌐 musl.

Storj's Object Mount **musl scripted installer** (a `.run` file) is compatible with all such Linux distributions.

This installer shell script allows you to install Object Mount (for the current user) *without* needing root privileges — no `sudo` required.

> **NOTE**
> **Optional: Package Installer**
>
> If you prefer a **full installer** for use with package managers (i.e.: Alpine, etc.) refer to the installation instructions.

Follow the steps below to install and configure Object Mount for **musl** platforms.

> **WARNING**
> **`libgcc` Dependency**
>
> The `musl` Scripted Installer instructions below assume you have the `libgcc` package installed (due to a dependency on `libgcc_s.so`).
>
> If necessary, run the following command on your Alpine distro to add the `libgcc` package (may require `sudo`):
>
> ```
> apk add libgcc
> ```

Step 1. Download the Install Script

Download the latest scripted installer.

- **Note:** If you do not have a download link, reach out to your 🌐 Storj Representative.
- **Note:** The musl scripted installer is named `cuno-<version>-x86_64-musl-installer.run`. It runs on 64-bit Intel and AMD (x86_64) processors, not on ARM-based CPUs.

Step 2. Install & Activate Object Mount

- **Note:** If you do not yet have a license key, ask your 🌐 Storj Representative for a trial key.

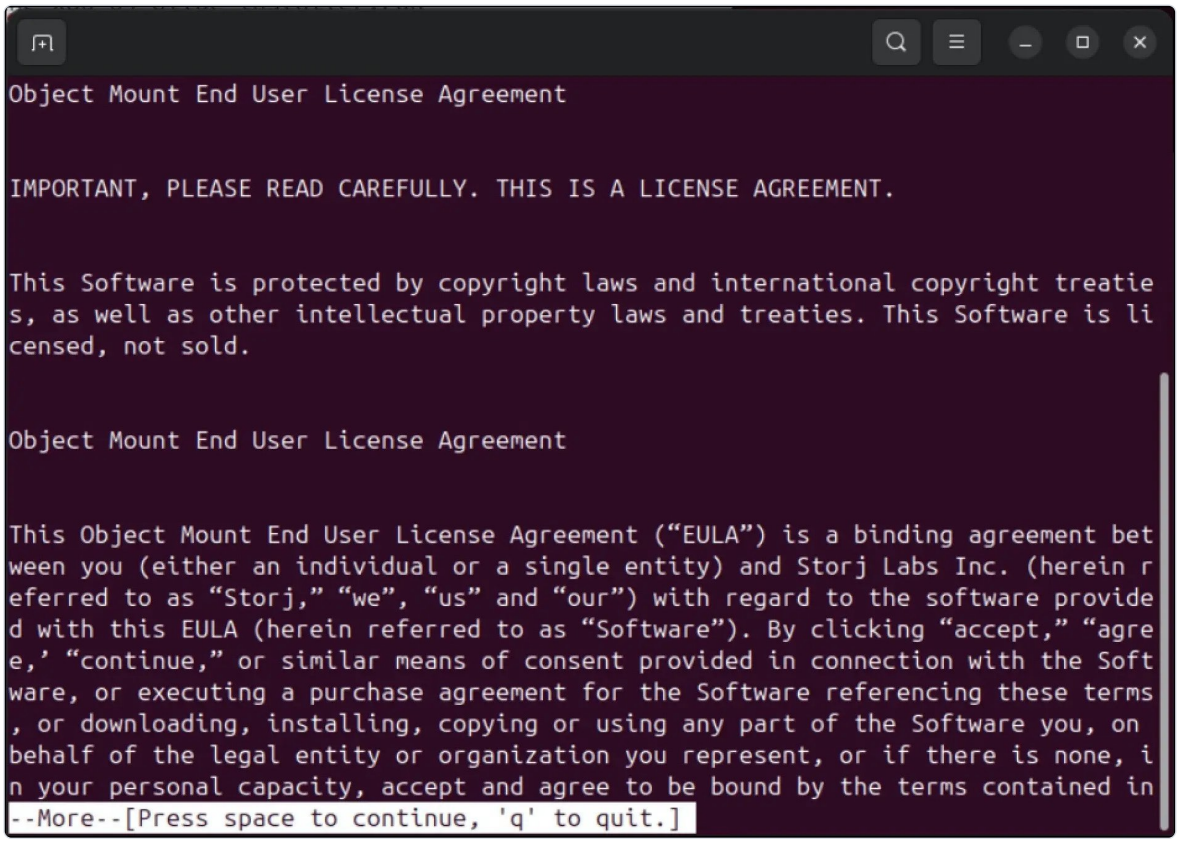
2a. To install Object Mount for use *only* by the **current user**, run the following command from a terminal window:

```
# terminal
sh cuno-<version>-x86_64-musl-installer.run
```

2b. To install Object Mount to be used by **all users** system-wide, add `sudo` to the *front* of the command:

```
# terminal
sudo sh cuno-<version>-x86_64-musl-installer.run
```

2c. The installation process will prompt you to read and agree to the Object Mount **end-user license agreement (EULA)**.



- **Note:** You can automate this step by setting the environment variable `CUNO_INSTALL_ACCEPT_EULA="yes"`. (This is equivalent to accepting the EULA).

2d. The installer then activates Object Mount if the environment variable `CUNO_INSTALL_LICENSE` is set to your license key or to the full path of a license key file. Set it to `none`, or leave it unset, to skip activation. With `sudo`, set `CUNO_INSTALL_LICENSE` and `CUNO_INSTALL_ACCEPT_EULA` after `sudo` (for example `sudo CUNO_INSTALL_LICENSE=/path/to/license.txt <install command>`); `sudo` drops variables set before it.

If Object Mount is not activated, the installer prints the next steps. Activate it afterwards with `cuno creds activate`; see .

Installation Locations: Scripted Installers

When Object Mount is installed via a **Scripted Installer** (e.g.: glibc or musl), components are installed to different locations depending on whether it was installed by a root, or by a non-root user.

Install Method	Install directory	Symlinks created
Non-Root User	`~/.local/opt/cuno/`	`~/.local/bin/cuno`
		`~/.local/lib/x86_64-linux-gnu/cuno.so`
		bash completions in `~/.local/share/bash-completion/completions/`
Root User (`sudo`)	`/opt/cuno/`	`/usr/bin/cuno`
		`/usr/lib/cuno.so`
		bash completions in the system bash-completion directory
		zsh completions in `/usr/share/zsh/vendor-completions/`

Next Steps

Once Object Mount is installed, proceed to the Linux User Guide article: to ensure you can access both public and private Object Storage buckets.

Debian Package Installer

Storj provides an Object Mount package installer for Debian (and Debian-based) Linux distributions.

This installer can be used with 🌐 Debian-based distributions such as Ubuntu, etc.

> **NOTE**
> **Optional: Scripted Installer**
>
> If you prefer to use a glibc-based **scripted installer**, refer to the installation instructions.

Follow the steps below to install and configure Object Mount for Debian-based Linux platforms.

Step 1. Download the Installer Package

Download the latest package installer.

- **Note:** If you do not have a download link, reach out to your 🌐 Storj Representative.

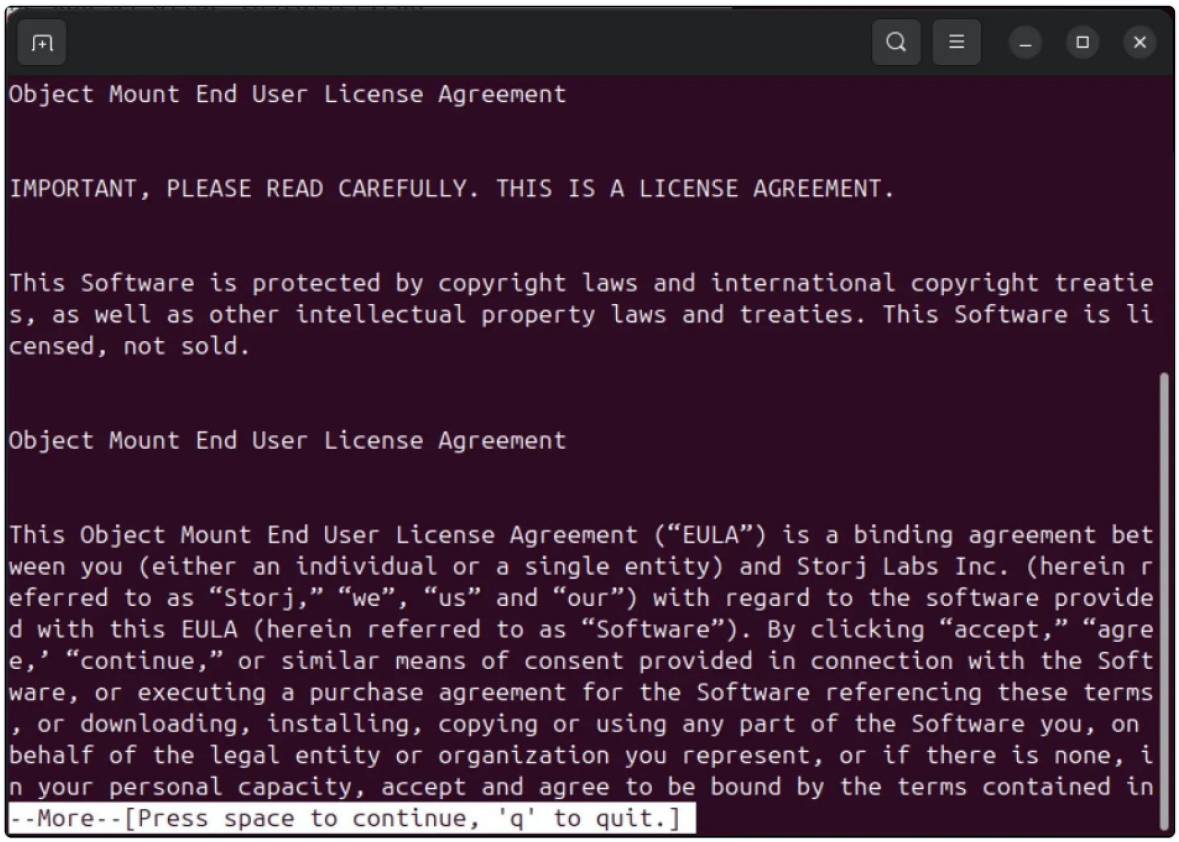
Step 2. Unpack the Archive

Use the installer's `.run` script to unpack the archive by entering the following command from a terminal window:

```
sh cuno_<version>_amd64_glibc_deb.run
```

> **NOTE**
> **Processor Compatibility**
>
> Although the Debian distro file naming convention is to only specify `AMD64` (ex: `cuno_<version>_amd64_glibc_deb.run`), the Object Mount DEB/APT Installer Package is designed to run on *both* AMD64 and Intel x86_64 processors (although not ARM-based CPUs).
>
> **Note:** ARM-based processors are not supported at this time.

The unpacking process will prompt you to read and agree to the Object Mount **end-user license agreement (EULA)**.



- **Note:** You can automate this step by setting the environment variable `CUNO_INSTALL_ACCEPT_EULA="yes"`. (This is equivalent to accepting the EULA).

The archive and its contents will be unpacked into a directory named `cuno_<version>_amd64_glibc_deb` containing the package itself plus some additional files.

View the unpacked contents with the following `ls` command (substituting your version number):

```
$ ls cuno_26.11.3_amd64_glibc_deb
cuno_26.11.3_amd64_glibc.deb
Object-Mount-User-Guide.pdf
```

Step 3. Install Object Mount and its dependencies using apt

3a. Use `sudo` to update `apt`:

```
sudo apt update
```

3b. Then install the Object Mount package (substituting your version number):

```
sudo apt install ./cuno_26.11.3_amd64_glibc_deb/cuno_26.11.3_amd64_glibc.deb
```

The installer will fetch and install any necessary additional packages (such as `libfuse`, etc.).

3c. The installer then activates Object Mount if the environment variable `CUNO_INSTALL_LICENSE` is set to your license key or to the full path of a license key file. Set it to `none`, or leave it unset, to skip activation. With `sudo`, set `CUNO_INSTALL_LICENSE` and `CUNO_INSTALL_ACCEPT_EULA` after `sudo` (for example `sudo CUNO_INSTALL_LICENSE=/path/to/license.txt <install command>`); `sudo` drops variables set before it.

If Object Mount is not activated, the installer prints the next steps. Activate it afterwards with `cuno creds activate`; see .

> **NOTE**
>
> **Access for: All-Users**
>
> By using `sudo`, all package manager installations of Object Mount install the app for use by *all users* of the system, not just the account that initiated the install process.

Next Steps

Once Object Mount is installed, proceed to the Linux User Guide article: to ensure you can access both public and private Object Storage buckets.

Red Hat Package Installer

Storj provides an Object Mount package installer for Red Hat Linux (and 🌐 RHEL-based Linux distributions)

> **NOTE**
> **Optional: Scripted Installer**
>
> If you prefer to use a glibc-based **scripted installer**, refer to the installation instructions.

Follow the steps below to install and configure Object Mount for Red Hat-based Linux platforms.

Step 1. Download the Installer Package

Download the latest package installer.

- **Note:** If you do not have a download link, reach out to your 🌐 Storj Representative.

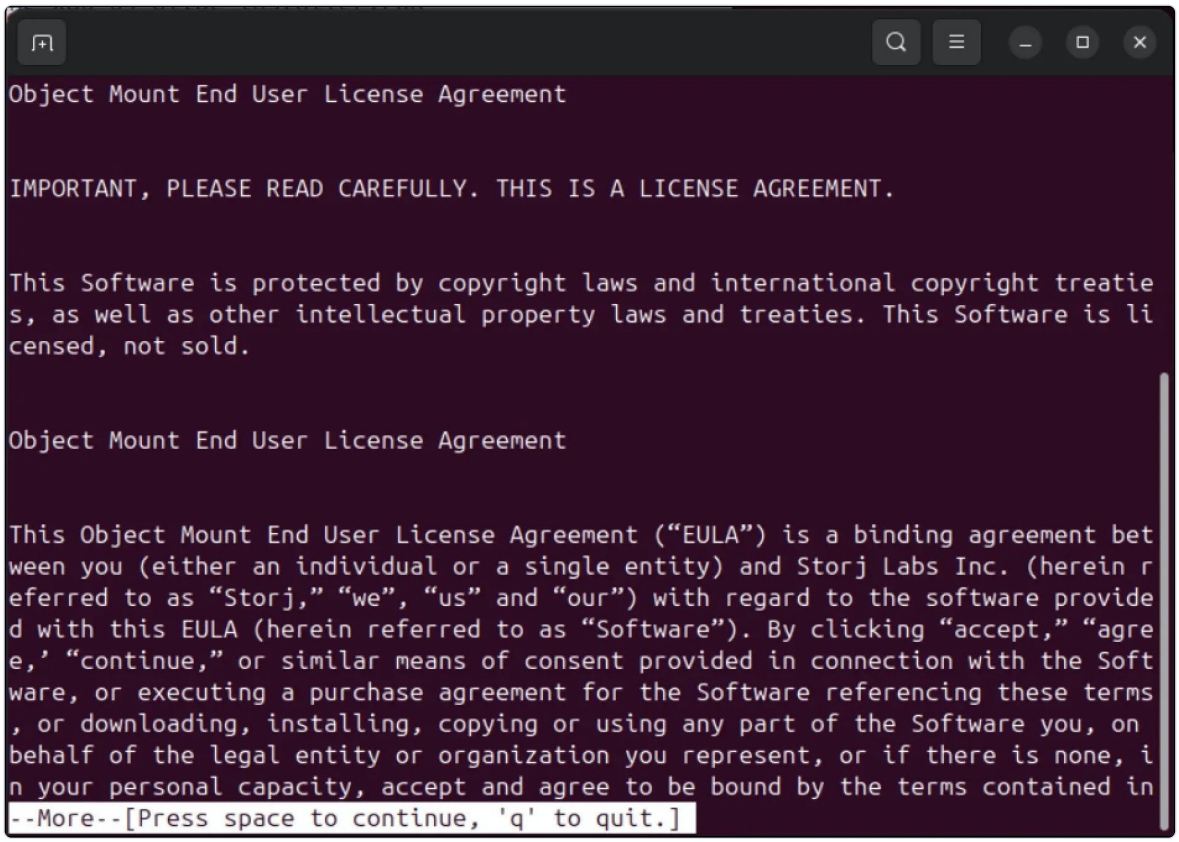
Step 2. Unpack the Archive

Use the installer's `.run` script to unpack the archive by entering the following command from a terminal window:

```
sh cuno_<version>_x86_64_glibc_rpm.run
```

> **NOTE**
> **Processor Compatibility**
>
> Although the Red Hat distro file naming convention is to only specify `X86_64` (ex: `cuno_<version>_x86_64_glibc_rpm.run`), the Object Mount YUM/RPM Installer Package is designed to run on *both* AMD64 and Intel x86_64 processors (although not ARM-based CPUs).
>
> **Note:** ARM-based processors are not supported at this time.

The unpacking process will prompt you to read and agree to the Object Mount **end-user license agreement (EULA)**.



- **Note:** You can automate this step by setting the environment variable `CUNO_INSTALL_ACCEPT_EULA="yes"`. (This is equivalent to accepting the EULA).

The archive and its contents will be unpacked into a directory named `cuno_<version>_x86_64_glibc_rpm` containing the package itself plus some additional files.

View the unpacked contents with the following `ls` command (substituting your version number):

```
$ ls cuno_26.11.3_x86_64_glibc_rpm
cuno_26.11.3_x86_64_glibc.rpm
Object-Mount-User-Guide.pdf
```

Step 3. Install Object Mount and its dependencies using yum

3a. Use the following commands to extract and install Object Mount (substituting your version number):

```
sudo yum makecache
sudo yum install ./cuno_26.11.3_x86_64_glibc_rpm/cuno_26.11.3_x86_64_glibc.rpm
```

The installer will fetch and install any necessary additional packages (such as `libfuse`, etc.).

3b. The installer then activates Object Mount if the environment variable `CUNO_INSTALL_LICENSE` is set to your license key or to the full path of a license key file. Set it to `none`, or leave it unset, to skip activation. With `sudo`, set `CUNO_INSTALL_LICENSE` and `CUNO_INSTALL_ACCEPT_EULA` after `sudo` (for example `sudo CUNO_INSTALL_LICENSE=/path/to/license.txt <install command>`); `sudo` drops variables set before it.

If Object Mount is not activated, the installer prints the next steps. Activate it afterwards with `cuno creds activate`; see .

> **NOTE**
>
> **Access for: All-Users**
>
> By using `sudo`, all package manager installations of Object Mount install the app for use by *all users* of the system, not just the account that initiated the install process.

Next Steps

Once Object Mount is installed, proceed to the Linux User Guide article: to ensure you can access both public and private Object Storage buckets.

Alpine Package Installer

Storj provides an Object Mount package installer for the Alpine Linux distribution.

This installer can be used with any musl-based Linux distribution.

> **NOTE**
> **Optional: Scripted Installer**
>
> If you prefer to use a musl-based **scripted installer**, refer to the installation instructions.

Follow the steps below to install and configure Object Mount for Alpine-based Linux platforms.

Step 1. Download the Installer Package

Download the latest package installer.

- **Note:** If you do not have a download link, reach out to your 🌐 Storj Representative.

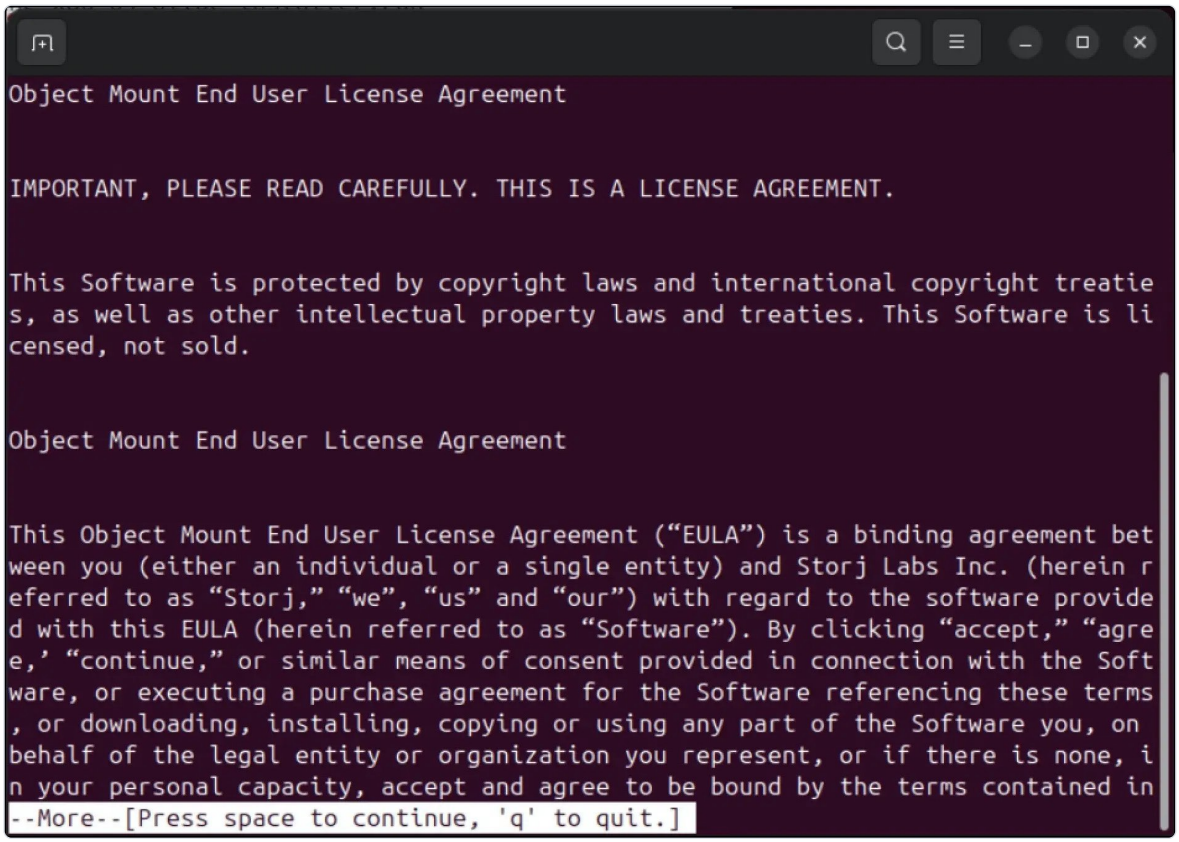
Step 2. Unpack the Archive

Use the installer's `.run` script to unpack the archive by entering the following command from a terminal window:

```
sh cuno_<version>_x86_64_musl_apk.run
```

> **NOTE**
> **Processor Compatibility**
>
> Although the Alpine distro file naming convention is to only specify `X86_64` (ex: `cuno_<version>_x86_64_musl_apk.run`), the Object Mount APK Installer Package is designed to run on *both* AMD64 and Intel x86_64 processors (although not ARM-based CPUs).
>
> **Note:** ARM-based processors are not supported at this time.

The unpacking process will prompt you to read and agree to the Object Mount **end-user license agreement (EULA)**.



- **Note:** You can automate this step by setting the environment variable `CUNO_INSTALL_ACCEPT_EULA="yes"`. (This is equivalent to accepting the EULA).

The archive and its contents will be unpacked into a directory named `cuno_<version>_x86_64_musl_apk` containing the package itself plus some additional files.

View the unpacked contents with the following `ls` command (substituting your version number):

```
$ ls cuno_26.11.3_x86_64_musl_apk
cuno_26.11.3_x86_64_musl.apk
Object-Mount-User-Guide.pdf
```

Step 3. Install Object Mount and its dependencies using `apk`

3a. Install the Object Mount package (substituting your version number):

```
sudo apk add ./cuno_26.11.3_x86_64_musl_apk/cuno_26.11.3_x86_64_musl.apk
```

The installer will fetch and install any necessary additional packages (such as `libfuse`, etc.).

3b. The installer then activates Object Mount if the environment variable `CUNO_INSTALL_LICENSE` is set to your license key or to the full path of a license key file. Set it to `none`, or leave it unset, to skip activation. With `sudo`, set `CUNO_INSTALL_LICENSE` and `CUNO_INSTALL_ACCEPT_EULA` after `sudo` (for example `sudo CUNO_INSTALL_LICENSE=/path/to/license.txt <install command>`); `sudo` drops variables set before it.

If Object Mount is not activated, the installer prints the next steps. Activate it afterwards with `cuno creds activate`; see .

> **NOTE**
>
> **Access for: All-Users**
>
> By using `sudo`, all package manager installations of Object Mount install the app for use by *all users* of the system, not just the account that initiated the install process.

Next Steps

Once Object Mount is installed, proceed to the Linux User Guide article: to ensure you can access both public and private Object Storage buckets.

Windows Subsystem for Linux Installation

Storj provides a full-featured, native version of Object Mount for Windows.

You are **strongly encouraged** to use that native Windows version.

However, if you would like to install the Linux version of Object Mount within your Windows environment, you can leverage 🌐 Windows Subsystem for Linux (WSL).

Installation of Object Mount for Linux within a Linux virtual machine or container is also possible.

Windows Subsystem for Linux

WSL installs Ubuntu by default.

For Ubuntu, follow either the glibc Scripted Installer instructions or the Debian Package Installer instructions.

Once completed, you will be able to use Object Mount with any Linux applications also installed within the Linux subsystem.

User Guides for Linux

Once Object Mount for Linux is installed and running on your computer, browse through these Linux **User Guides** articles for additional usage and configuration settings.

Articles Include:

- Validate Functionality & Access Object Storage
- Activate a License Key
- Understand Deployment Modes
- Enable Enforced POSIX File Access
- Cloud Path Details
- Application Tips
- Media & Entertainment Mode
- High-Performance Computing Mode

Validate Functionality & Access Object Storage

After installing Object Mount for Linux, you should:

1. Run and Validate global functionality by accessing public buckets
2. Import your private S3 credentials into Object Mount
3. Connect to your private Object Storage buckets

Details for each step is outlined below.

Step 1. Validate Object Mount's Public Connectivity

The following steps will help you verify that Object Mount is running correctly and can connect to publicly accessible S3 storage buckets.

1a. Enter Object Mount's CLI Console (aka "cunoFS")

- From a Linux shell prompt enter the `cuno` command:

```
user:~$ cuno
```

- You should see the `(cuno)` CLI prefix to the left of your user prompt. This indicates that a new shell has been launched, wrapped in Object Mount's `cuno` wrapper:

```
user:~$ cuno
(cuno) user:~$
```

> **NOTE**
>
> **Adding `cuno` to the PATH**
>
> If the Object Mount `cuno` application cannot be found (`cuno: command not found`), add the install location to your `$PATH` variable.

- Enter `exit` at any time to close the Object Mount CLI console and return to the unwrapped shell prompt:

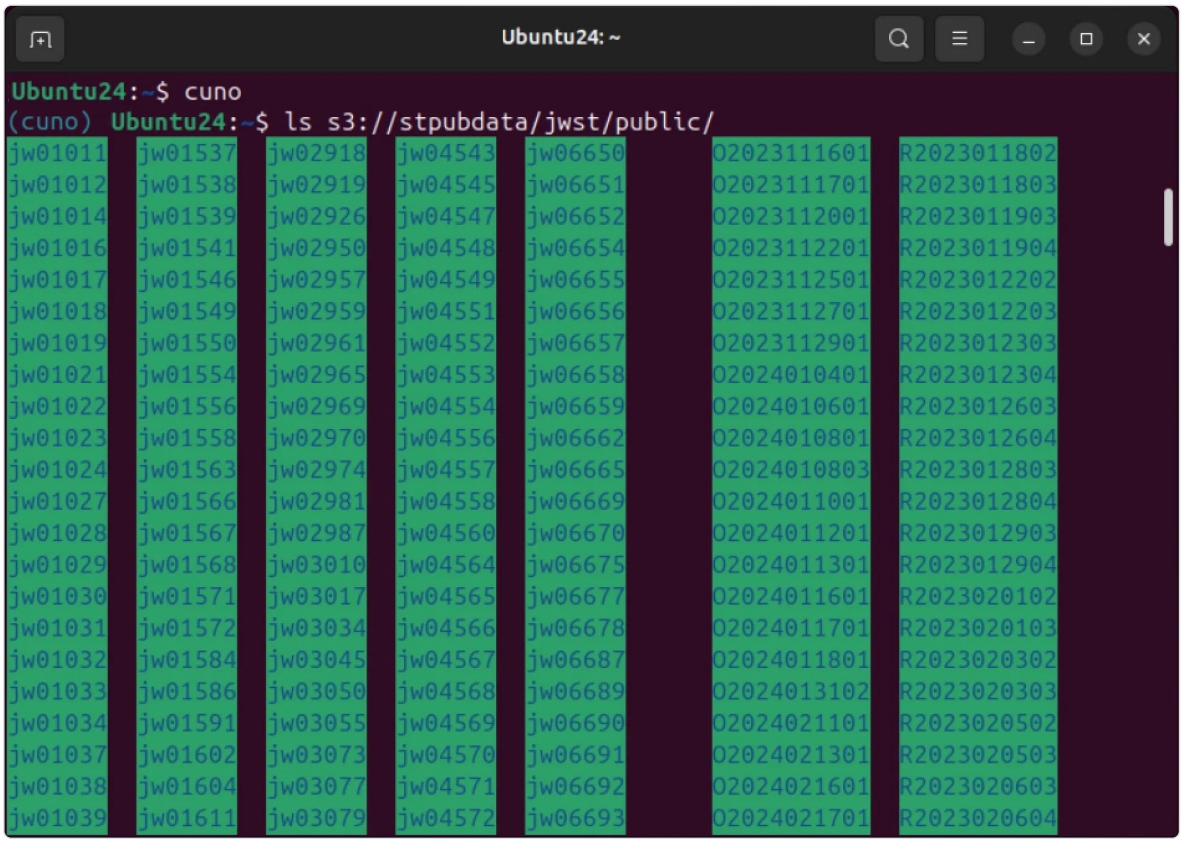
```
(cuno) user:~$ exit
user:~$
```

1b. Explore Public Datasets

Validate that Object Mount can reach Internet-based object storage buckets by listing the files in several public S3 repositories:

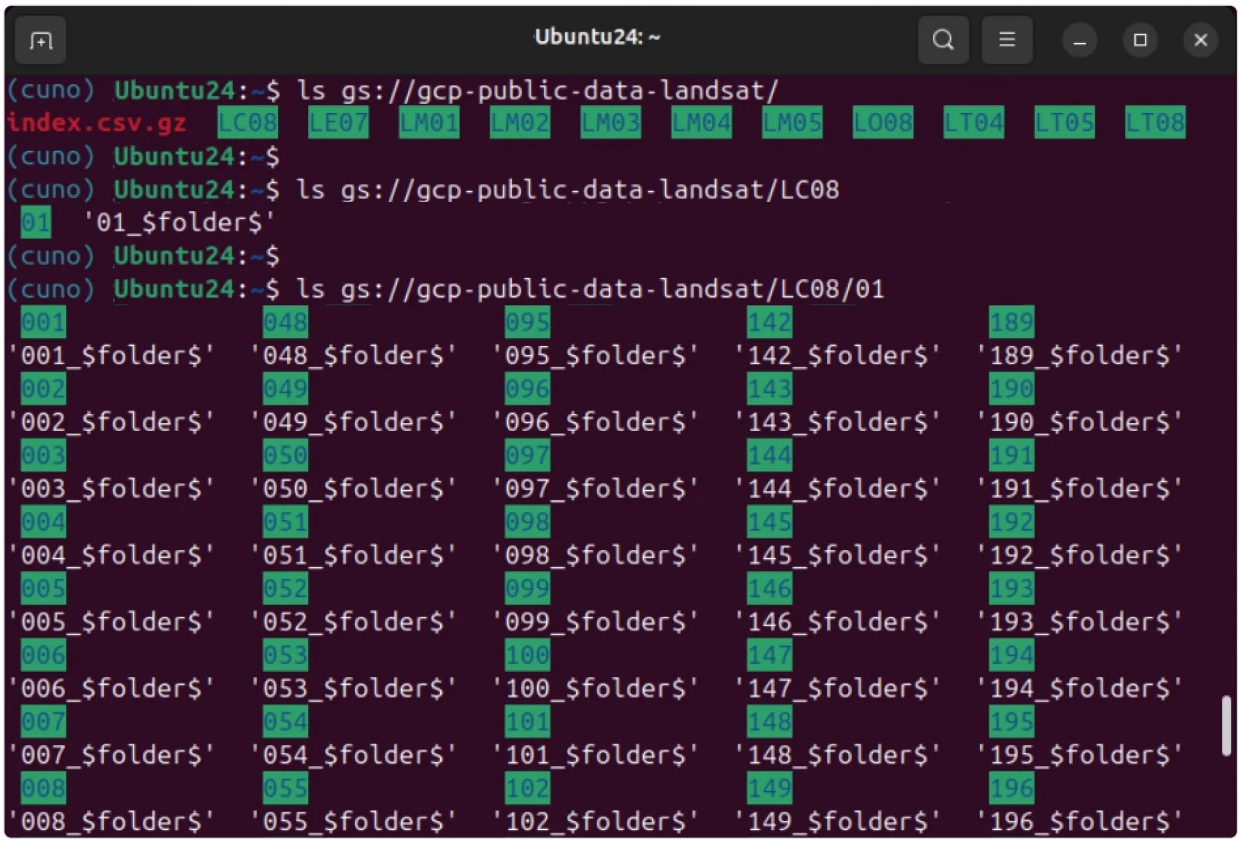
- View image file listings from the James Webb Space Telescope on AWS S3:

```
(cuno) $ ls s3://stpubdata/jwst/public/
```



- Browse satellite images on Google Cloud Storage:

```
(cuno) $ ls gs://gcp-public-data-landsat/
```



Step 2. Import S3 Credentials

In order to access and mount your private Object Storage buckets you will need to save your S3 credentials to a file.

Instructions on obtaining your credentials and saving them to a file can be found in the Appendix article: .

Once your credentials have been saved to a file on your Linux drive, proceed with the following steps:

2a. Import Your Credential File into Object Mount

Assuming you have saved your credentials to a file named `credentials.txt`, run the following command to add these credentials to the local set of Object Mount managed credentials:

```
cuno creds import credentials.txt
```

This command will attempt to discover all the buckets that your credentials have access to, as well as the settings, limitations, and compatibility of these buckets. This may take a while if you have *many* buckets associated with the credentials you are importing.

```
cuno creds import credentials.txt
- Examining credential file..
- Importing credential file into CUNO's credentials store..
- Detecting available buckets/containers..
Detected buckets: 2
- Attempting to pair 2 buckets/containers..
Paired 2/2 buckets:
 [+] s3://bucket01
 [+] s3://bucket02
```

You can display your imported credentials with the `cuno creds list` command:

```
+-------------+
| credentials |
+-------------+
 [*] credentials.s3c

+----------+
| pairings |
+----------+
 [*] s3://bucket01 -> credentials.s3c
 [*] s3://bucket02 -> credentials.s3c
```

Notice that a new credential file has been created with the `.s3c` extension, and that two buckets were found using the credentials.

> **NOTE**
>
> **Generated `.s3c` Credential File Location**
>
> The `cuno creds import` command creates an `.s3c` file with the corresponding bucket entries and adds appropriate configuration settings such as: region, URL path style, etc.
>
> The default location for storing these imported credentials is the directory `$XDG_CONFIG_HOME/cuno/creds` (if unset, `$XDG_CONFIG_HOME` defaults to `~/.config`).
>
> To use an alternative location, set the `CUNO_CREDENTIALS` environment variable to point to your preferred path.
>
> For example: `export CUNO_CREDENTIALS=/home/user/my-cloud-credentials`.
>
> **Note:** You should *not* manually copy your credentials directly into these locations.

2b. Troubleshooting Credential Import

Purge and Re-Import:

If you encounter an issue and need to re-import your credential file, you must first remove the previously generated (and unwanted) `.s3c` credential file using the `purge` command:

```
cuno creds purge credentials.s3c
```

See the article for additional `cuno creds` details.

S3 Compatibility Check:

If you are using an S3-compatible service and are having problems, you can run a compatibility check using the `detectfeatures` command:

```
cuno creds detectfeatures s3://bucket-to-test credentials.s3c
```

This command will iterate through multiple S3-compatibility tests, settings, and limitations, and then adjust the imported `.s3c` credential file based on its findings. Give the imported name shown by `cuno creds list` (`credentials.s3c`), not the original `credentials.txt`: any other argument is treated as a file path, and the results are appended to that file instead.

Note: The bucket specified must permit **write access** to allow for the creation of temporary files for testing purposes.

> **WARNING**
>
> **Feature Detection: Bandwidth & Time Usage**
>
> Running feature detection may use up to a few gigabytes of bandwidth and may take a several minutes to complete depending on the machine's connection speed and the S3 storage provider.

Step 3. Test Connectivity to Your Private Object Storage Bucket(s)

Once your credentials have been successfully imported you can test access to your private buckets.

3a. Enter Object Mount's CLI Console (aka "cunoFS")

- From a Linux shell prompt enter the `cuno` command:

```
user:~$ cuno
```

- You should see the `(cuno)` CLI prefix to the left of your user prompt. This indicates that a new shell has been launched, wrapped in Object Mount's `cuno` wrapper:

```
user:~$ cuno
(cuno) user:~$
```

3b. Access Your Bucket's Contents

Select your Object Storage Provider below for the appropriate commands to:

- View buckets
- View files
- Create files
- Display files
- Delete files

AWS S3

List your accessible buckets:

```
(cuno) $ ls s3://
```

List files and folders in a bucket:

```
(cuno) $ ls s3://bucket01/
```

Create a new object in your bucket

```
(cuno) $ echo "hello world" > s3://bucket01/helloworld.txt
```

Display that new file back to the console:

```
(cuno) $ cat s3://bucket01/helloworld.txt
```

Delete that new file:

```
(cuno) $ rm s3://bucket01/helloworld.txt
```

Storj

List your accessible buckets:

```
(cuno) $ ls s3://
```

List files and folders in a bucket:

```
(cuno) $ ls s3://bucket01/
```

Create a new object in your bucket

```
(cuno) $ echo "hello world" > s3://bucket01/helloworld.txt
```

Display that new file back to the console:

```
(cuno) $ cat s3://bucket01/helloworld.txt
```

Delete that new file:

```
(cuno) $ rm s3://bucket01/helloworld.txt
```

Microsoft Azure

List your accessible buckets:

```
(cuno) $ ls az://
```

List files and folders in a bucket:

```
(cuno) $ ls az://your-azure-storage-account/bucket01/
```

Create a new object in your bucket

```
(cuno) $ echo "hello world" > az://your-azure-storage-account/bucket01/helloworld.txt
```

Display that new file back to the console:

```
(cuno) $ cat az://your-azure-storage-account/bucket01/helloworld.txt
```

Delete that new file:

```
(cuno) $ rm az://your-azure-storage-account/bucket01/helloworld.txt
```

Google Cloud

List your accessible buckets:

```
(cuno) $ ls gs://
```

List files and folders in a bucket:

```
(cuno) $ ls gs://bucket01/
```

Create a new object in your bucket

```
(cuno) $ echo "hello world" > gs://bucket01/helloworld.txt
```

Display that new file back to the console:

```
(cuno) $ cat gs://bucket01/helloworld.txt
```

Delete that new file:

```
(cuno) $ rm gs://bucket01/helloworld.txt
```

Other S3 Compatible

List your accessible buckets:

```
(cuno) $ ls s3://
```

List files and folders in a bucket:

```
(cuno) $ ls s3://bucket01/
```

Create a new object in your bucket

```
(cuno) $ echo "hello world" > s3://bucket01/helloworld.txt
```

Display that new file back to the console:

```
(cuno) $ cat s3://bucket01/helloworld.txt
```

Delete that new file:

```
(cuno) $ rm s3://bucket01/helloworld.txt
```

3c. Exit Object Mount's CLI

Enter `exit` at any time to close the Object Mount CLI console and return to the unwrapped shell prompt:

```
(cuno) user:~$ exit
user:~$
```

Activate a License Key

Activate using an Object Mount License Key

When you first installed Object Mount for Linux, you may have already added your license key with the `CUNO_INSTALL_LICENSE` environment variable.

- If not, the installer told you to activate Object Mount with `cuno creds activate`.
- Or you may have renewed an expired license key.

Note: If you do not yet have a license key, reach out to your 🌐 Storj Representative.

In any of the above situations, you can activate (or re-activate) Object Mount with your new license key by entering the `cuno creds activate` command and following the interactive steps:

```
# terminal
cuno creds activate
```

Non-Interactive Activation

Alternatively, for non-interactive license key activation, you can point the command to a license key file. The file's permissions must be `0600`, `0400` or `0500`, otherwise the command rejects it:

```
# terminal
cuno creds activate "<path/filename>"
```

You can also pipe the license key into the command as its input:

```
# terminal
echo "<your license key>" | cuno creds activate
```

> **NOTE**
>
> **License File Location**
>
> Once activated, the license file is `etc/license` in the install directory: `/opt/cuno/etc/license` for a system-wide install, or `~/.local/opt/cuno/etc/license` for a single-user install.
>
> By default, the permissions on this file are set to `0600`. If needed, Administrators can manually modify the group and permissions to allow other system users access to the `cuno` CLI command.

Activation for Multiple Users

Normally, activation means that access to Object Mount is **limited to the one user** who runs the `cuno creds activate` command.

However, if the user that runs `cuno creds activate` is `root` and runs it in a terminal, the command asks whether to give access to the root group and to all other users. When the key is piped in, or set with `CUNO_INSTALL_LICENSE` during a package install, it does not ask and the license stays readable by `root` only.

Deployment Mode Details

The articles in this section provide additional details on the different Modes that Object Mount for Linux can be deployed in.

As was described in the **Getting Started Guide** article , Object Mount for Linux can be used in several different modes:

1. Direct Interception Mode: In Direct Interception Mode, Object Mount inserts itself between your standard Linux applications and the host Operating System. Object Mount detects and intercepts file I/O calls, then translates and redirects them, as needed, to your object storage provider. The Object Mount Command Line Interface, `cuno`, is used to configure Direct Interception Mode.
2. Object Mount on FUSE: If Object Mount's CLI cannot be used, or you require broader support for non-standard Linux applications, Object Mount on FUSE uses Linux FUSE (Filesystem in Userspace) to mount object storage within the native Linux file system environment. This increases compatibility but can reduce performance.
3. Object Mount FlexMount combines the speed and performance of Direct Interception (#1) with the wide application support of Object Mount on FUSE (#2).
4. Object Mount Fusion can be used to enhance high-performance *attached* storage solutions (like Amazon's Elastic File System - EFS) with the throughput of object storage.

Enabling Object Mount in Other Environments

For information on setting up Docker containers with Object Mount pre-loaded, automatically setting up Object Mount on FUSE at boot, and other containerization and High-Performance Computing (HPC) use-cases, see Linux User Guide article: .

Direct Interception

This article dives into the configuration and use of **Direct Interception Mode** in Object Mount for Linux.

Overview

When running in **Direct Interception Mode**, Object Mount will insert itself between your applications and the operating system. It will dynamically redirect relevant file system calls made by shell commands and applications to the appropriate APIs of your object storage provider. This dynamic "mapping" between *file system concepts* and *object storage interfaces* gives your tools instant access to objects as files.

This direct interception means *any* applications that can access file storage can now work with object storage. Object Mount intercepts I/O calls from any user binaries, including those written in C/C++, Rust, Go, Java, Python, etc.

It's built to work everywhere: It works with dynamic binaries and static binaries. It works inside unprivileged containerized environments.

> **NOTE**
>
> **Support for Any Application**
>
> This includes applications you've written yourself — just treat the paths to object storage as local file system paths. Object Mount handles the rest.

Direct interception Mode uses the `LD_PRELOAD` environment variable so that Object Mount can capture and redirect storage access library and system calls to object storage APIs.

- If a static binary is intercepted, a JIT ELF binary translator will redirect relevant calls when the binary is loaded into memory.

Highlights

- Direct Interception offers the highest performance access to object storage that Object Mount provides.
- Direct Interception Mode is the *default mode* and the most common mode for Object Mount to be deployed in.
- Unless otherwise noted, all documentation pages and installation instructions are written assuming you are using "Direct Interception Mode".

Example Use Cases

- Direct Interception Mode is best suited for situations in which installed app compatibility is not an issue.
- For example: When setting up a fixed workflow with Object Mount that can be tested and verified before putting it into production.

> **WARNING**
>
> **Direct Interception vs. Object Mount on Fuse**
>
> Direct Interception Mode does *not* currently support SUID binaries, or certain packaged apps like 🌐 Snap, 🌐 AppImage, or 🌐 Flatpak applications.
>
> See Object Mount on FUSE and/or Object Mount FlexMount to support these tools.

Advantages and Disadvantages

Advantages	Disadvantages
Speed: Dynamic interception offers the best performance.	**Compatibility:** Direct interception does not currently support SUID binaries, Snaps, AppArmor, or Flatpak applications.
Set up time: This is the default mode — just launch an Object Mount shell with the `cuno` CLI command.	**Activation:** In some cases, it is difficult to enable direct interception or keep it enabled (because of environment variable clobbering or lack of privileges to set/use LD_PRELOAD).
Linux native: For users who are skilled with Linux, using the `cuno` CLI is quick, easy, and non-intrusive.	**Non-technical users:** If Object Mount is being used by team members with only minimal Linux skills, teaching them how to use the Object Mount shell and diagnose if it's working correctly may be less desirable than setting up a Object Mount on FUSE on their behalf.

How to Enable

Direct Interception Mode can be enabled per-session or per-command.

Direct Interception: Per-Session

Enable Object Mount for a **single shell session** by calling `cuno` from a terminal command line:

```
# terminal
cuno
```

This will launch a new interactive "wrapped shell" with Object Mount acting in Direct Interception Mode.

The wrapped shell itself has Object Mount monitoring all its I/O calls — so every command entered or application launched from within the wrapped shell will be monitored and intercepted by Object Mount

> **NOTE**
>
> **Object Mount CLI `cuno` as a Wrapper**
>
> When `cuno` is used to launch a new shell, the shell is launched with `LD_PRELOAD` set to point at `cuno.so`.
>
> - There is no Object Mount shell binary.
> - `cuno` only "wraps" existing shells (bash, zsh, etc.) with Object Mount pre-loaded.

Direct Interception: Per-Command

To enable direct interception for a **single command**, use:

```
cuno run bash -c "<your command and arguments>"
```

> **NOTE**
>
> **Best Practice for Running Single Commands**
>
> It is recommended to execute one-time commands in this fashion to maintain support for Wildcard expansion (`*`) of filesystem and remote paths.

Path Usage

Inside a `cuno`-wrapped shell session, object storage buckets can be accessed using either **path** or **URI** formats.

The commands below provide examples of both path and URI usage within a `cuno`-launched shell:

AWS S3

```
# terminal
cuno
(cuno)$ ls s3://<bucket>/<path>
(cuno)$ ls /cuno/s3/<bucket>/<path>
```

Storj

```
# terminal
cuno
(cuno)$ ls s3://<bucket>/<path>
(cuno)$ ls /cuno/s3/<bucket>/<path>
```

Microsoft Azure

```
# terminal
cuno
(cuno)$ ls az://<account name>/<bucket>/<path>
(cuno)$ ls /cuno/az/<account name>/<bucket>/<path>
```

Google Cloud

```
  # terminal
  cuno
  (cuno)$ ls gs://<bucket>/<path>
  (cuno)$ ls /cuno/gs/<bucket>/<path>
```

Other S3 Compatible

```
# terminal
cuno
(cuno)$ ls s3://<bucket>/<path>
(cuno)$ ls /cuno/s3/<bucket>/<path>
```

> **NOTE**
>
> **Cloud Paths**
>
> For additional information on specifying cloud paths using the Object Mount CLI, see the Linux User Guide article: .

Shell Interaction

`bash` and `zsh` are fully supported when using the Object Mount CLI.

This includes support for:

- Tab auto-completion of remote paths
- Wildcard expansion (`*`) of filesystem and remote paths

In either of these fully-supported shells, the prompt will be prefixed with `(cuno)` to indicate that Direct Interception is enabled and that you are using an Object Mount CLI-wrapped shell::

```
(cuno) user@host:~$
```

Detecting Shells

To select between the two shells, Object Mount determines if either is present using the following methods (in order):

1. The shell used to launch Object Mount
2. The user's preferred login shell
3. All installed shells

If neither `bash` nor `zsh` are found, then the first shell in this list is used. If no shell is set, then `/bin/sh` is used.

> **WARNING**
>
> **Processor Snapshot (`ps`) usage**
>
> The behavior described here relies on `ps`.
>
> Depending on your method of installation, you may need to install `ps` manually. (For RedHat/RHEL derivatives, install package `procps`).

Specifying Shells

To launch a **specific shell** with Object Mount enabled, use `cuno run`:

```
cuno run <shell>
```

Object Mount on FUSE

This article dives into the configuration and use of **Object Mount on FUSE** for Linux.

Overview

Object Mount on FUSE allows you to mount an object storage path as a *directory* within the local file system hierarchy.

This allows you (and any other user with access to the mount) to access object storage as if it were just another local directory.

Highlights & Advantages

Object Mount on FUSE is a FUSE file system. FUSE routes calls through the Object Mount object storage back-end.

- **Compatibility:** Use Object Mount on FUSE when compatibility is a primary concern (e.g.: when using a variety of applications across multiple different operating systems).
- **Interoperable:** Object Mount on FUSE can be used when launching the `cuno` binary is not possible (e.g. an automation engine without the required features).
- **Simple Usage:** After a mount is set up, scripts only need to be changed to point at the new directory. No changes are needed to the workflow and no environment variables need to be set.
- **Greater Administrative Control:** Object Mount on FUSE can be setup once, by an admin, without giving users any credentials. This removes the need to train users on object storage concepts.

> **NOTE**
>
> **Performance vs. Compatibility**
>
> Due to the nature of FUSE file systems, Object Mount on FUSE can be slightly less performant than Object Mount in Direct Interception Mode.
>
> If speed is your primary objective, consider using Object Mount in Direct Intercept Mode or via the User-Mode Library.

How to Enable

Create a FUSE-based object storage mount by entering:

```
# terminal
mkdir ~/my-object-storage
# terminal
cuno mount ~/my-object-storage
```

Any paired object storage buckets will now be accessible through the mount.

Choose your storage provider below to see the commands to list files on your S3 mount:

AWS S3

```
# terminal
ls ~/my-object-storage/s3/<bucket>/<path>
```

Storj

```
# terminal
ls ~/my-object-storage/s3/<bucket>/<path>
```

Microsoft Azure

```
# terminal
ls ~/my-object-storage/az/<account-name>/<bucket>/<path>
```

Google Cloud Storage

```
# terminal
ls ~/my-object-storage/gs/<bucket>/<path>
```

Other S3 Compatible

```
# terminal
ls ~/my-object-storage/s3/<bucket>/<path>
```

> **NOTE**
>
> **Mount Paths vs URI Paths**
>
> You cannot use URI paths like "s3://bucket1/foo" when using Object Mount on FUSE.
>
> To be usable, replace the URI prefix (`s3://`) with the path to your mount.

> **NOTE**
>
> **POSIX Options**
>
> The `--posix` option requires that a FUSE package be installed on the system.

Configuration Options

There are various options that can be specified from the command line:

```
cuno [subsys-options] mount [mount-options] <mount-path>
```

These options include:

- Object Mount ⟶ Mount Options
- Object Mount ⟶ Subsystem Options
- FUSE Options

Object Mount: Mount Options

To *right* of the `mount` verb, you can specify options that are specific to the mount operation:

```
cuno mount [option] ...
```

Noteworthy Mount Options:

Mount Option	Description
`--root <cloud path>`	Specifies the root object storage path to be mounted. For example, to mount a single bucket, you would use `--root s3://<bucket>`.

Mount Option	Description
`--posix`	Enables setting and enforcing access permissions, symbolic and hard links, users, groups, etc. Can be used with default permissions. Implicitly sets `-o allow_other`. This option is ideally used with a mount that is only on a single bucket, e.g. `cuno mount --root s3://<bucket> --posix <mount-path>`. You should also be using `cuno creds setposix s3://examplebucket true` to enable POSIX handling of the bucket consistently when Direct Interception or a FlexMount is used.
`<FUSE option>`	Options that are specific to the FUSE mount operation. These options are passed on to `fum` (FUSE User Mount). Not to be confused with Object Mount subsystem options which use the same `-o` syntax. See below for some of these. See user-guide-cuno-mount-fuse-options.
`--no-allow-root`	Do not allow root to access the mount (allowed by default). Disables support for SUID binaries, Snap, AppArmor, and Flatpak applications.
`--auto-restart`	Automatically restarts Object Mount on FUSE if problems occur during execution.
`--mkdir`	Automatically creates the mount point directory if it does not exist.
`--debug`	Enables debug output (same as `cuno mount -o debug`).
`--dev-logs`	Enabled debug logging to `/tmp/fuse.logs`
`--verbose`	Enables verbose output.

Object Mount: Subsystem Options

To *left* of the `mount` verb, you can specify options that are specific to the Object Mount subsystem:

```
cuno -o <Object Mount Subsystem Option> mount ...
```

Alternatively, these can be specific using a **system variable**:

```
export CUNO_OPTIONS="<Object Mount Subsystem Option>"
```

Noteworthy Subsystem Options:

Some relevant Object Mount Subsystem options are given in the table below.

Refer to the Linux Advanced Guide article for additional details.

Subsystem Option	Description
`cachehome=<directory>` (default: `/cunodb/;/dev/shm/`)	This option allows you to set a semicolon-delimited list of directories to consider for caching. The first existing directory is used. If none exists, Object Mount uses `$XDG_CACHE_HOME/cuno` (or `~/.cache/cuno`), and then the system temporary directory. This is for the internal metadata cache of the Object Mount process behind the mount. Note that the FUSE cache is separate from this cache. Quote the value, because the shell treats `;` as a command separator. Example: `cuno -o 'cachehome=/mnt/cache;/cunodb;/dev/shm' mount <mount-path>`

Subsystem Option	Description
`uid=<integer>`	Define the default user ownership of files and directories within a bucket. These are the UIDs that the Object Mount subsystem will feed to the FUSE mount for non-POSIX (core file access) buckets. Not recommended for most use cases, but may be useful if mounting non-POSIX buckets in the same mount as POSIX buckets. **Ignored** if the bucket has an enabled POSIX tag. Example: `cuno -o uid=$(id -u <username>) mount <mount-path>` To enforce these settings, use the FUSE mount option `-o default_permissions`.
`gid=<integer>`	Define the default group ownership of files and directories within a bucket. These are the GIDs that the Object Mount subsystem will feed to the FUSE mount for non-POSIX (core file access) buckets. Not recommended for most use cases, prefer to use POSIX file access. Not recommended for most use cases, but may be useful if mounting non-POSIX buckets in the same mount as POSIX buckets. **Ignored** if the bucket has an enabled POSIX tag. Example: `cuno -o gid=$(id -g <username>) mount <mount-path>` To enforce these settings, use the FUSE mount option `-o default_permissions`.
`filemode=<octal>`	Define the default file access permission bits of files within a bucket. Supply the octal (numeric) representation of the permissions you want to apply. These are the permissions that the Object Mount subsystem will feed to the FUSE mount for non-POSIX (core file access) buckets. Not recommended for most use cases, but may be useful if mounting non-POSIX buckets in the same mount as POSIX buckets. **Ignored** if the bucket has an enabled POSIX tag. For example, to set the default file access permission bits to 0770 (`-rwxrwx---`), use the following command: `cuno -o filemode=0770 mount <mount-path>` To enforce these settings, use the FUSE mount option `-o default_permissions`.
`dirmode=<octal>`	Define the default file access permission bits of directories within a bucket. Supply the octal (numeric) representation of the permissions you want to apply. These are the permissions that the Object Mount subsystem will feed to the FUSE mount for non-POSIX (core file access) buckets. Not recommended for most use cases, but may be useful if mounting non-POSIX buckets in the same mount as POSIX buckets. **Ignored** if the bucket has an enabled POSIX tag. For example, to set the default file access permission bits to 0770 (`drwx

FUSE Options

Also on the *right hand-side* of the `mount` verb, you can specify options that are specific to the FUSE mount operation.

These options are passed to the FUSE User Mount process: `fum`. (Refer to the 🌐 FUSE Mount documentation.

```
cuno mount [FUSE option] ...
```

> **NOTE**
>
> **Object Mount Options vs. FUSE Options**
>
> As with the Mount Options, these FUSE options must come *after* the verb `mount`.
>
> **Note:** Some of these FUSE options use a similar `-o` syntax as the Object Mount Subsystem options (e.g.: `cuno -o`), but they are *not* the same.

Noteworthy FUSE Options:

Option	Description
`-o allow_root`	Allows root access to the mount; required for SUID permissions.
`-o allow_other`	Allows other users to access the mount; requires `user_allow_other` in `/etc/fuse.conf`.
`-o default_permissions`	Enable permission checking by the kernel. To use Enforced POSIX, use `--posix`, which will set this internally.
`-o auto_unmount`	Automatically unmounts the mount when the process terminates.
`-o ro`	Mounts the file system read-only. To achieve a read-only mount through Object Mount, you can alternatively use `cuno mount --posix` and change the permissions on dirs/files in the mount manually to read-only using `chmod`. See here for more on standard mount options like `ro`, `rw`, etc.
`-o rw` (default)	Mounts the file system read-write.
`-o exec` (default)	Allow execution of binaries on the file system.
`-o noexec`	Disallow execution of binaries on the file system.
`-o clone_fd`	Uses a separate fuse device fd for each thread (may improve performance).
`-o max_idle_threads`	The maximum number of idle worker threads allowed (default: 10).
`-s`	Run in single-threaded mode.
`-f`	Run in the foreground.
`-o debug`	Enable debug output (implies `-f`).

FUSE Kernel Default Attributes:

Attribute	Description
`-o uid=N`	Sets the file owner of all mounted files/dirs to the specified user ID. These are the UIDs that the FUSE mount will expose to the user. This will ignore any cuonFS POSIX settings including the bucket tag. To enforce these settings, use the FUSE mount option `-o default_permissions`.
`-o gid=N`	Sets the file group of all mounted files/dirs to the specified group ID. These are the GIDs that the FUSE mount will expose to the user. This will ignore any cuonFS POSIX settings including the bucket tag. To enforce these settings, use the FUSE mount option `-o default_permissions`.
`-o umask=M`	Sets the file permissions (octal) of all mounted files/dirs. These are the permissions that the FUSE mount will expose to the user. This will ignore any cuonFS POSIX settings including the bucket tag. To enforce these settings, use the FUSE mount option `-o default_permissions`.

FUSE Kernel Cache Configuration:

Cache Config	Description
`-o kernel_cache`	This method caches file data across `open`; i.e. disables flushing of the file data cache on every `open`. Without this option (and neither `direct_io`), data is cached before the next `open` so a `read` syscall may not initiate a `read` operation.
`-o auto_cache`	Enables automatic flushing of the data cache on open(). The cache is only flushed if the modification based on modification times (off).
`-o entry_timeout=T`	Sets the kernel cache timeout for names (1.0s).
`-o negative_timeout=T`	Sets the kernel cache timeout for a negative lookup (file not found) (0.0s).

Cache Config	Description
`-o attr_timeout=T`	Sets the kernel cache timeout for attributes (1.0s).
`-o ac_attr_timeout=T`	Sets the kernel cache timeout for checking if `-o auto_cache` should flush file data on `open`.

Object Mount on FUSE - Commands

Once you have configured a mount using Object Mount on FUSE, you can use the following commands to manage it.

```
cuno mount [command]
```

Noteworthy Object Mount on FUSE Commands:

List

- `--list`
- Lists your active Object Mount on FUSE.
- You can also use the Linux utility `mount` directly.
- Example:

```
cuno mount --list
```

Unmount

- `--unmount` / `-u`
- Unmounts a mount from the specified mount path.
- You can also use the Linux utility `umount` directly.
- Example:

```
cuno mount --unmount <path to mount>
```

Unmount and end the `fum` process

- `--unmount-kill` / `-U`
- Unmounts a mount from the specified mount path and kills the `fum` process.
- You can also use the Linux utility `umount` directly.
- Example:

```
cuno mount --unmount-kill <path to mount>
```

Mount on Boot

You can add mount commands to `/etc/fstab` to automatically mount an object storage path on start up.

Object Mount FlexMount

This article dives into the configuration and use of **Object Mount FlexMount** for Linux.

Overview

Object Mount FlexMount combines the wide compatibility and support for Linux Apps via FUSE (including SUID binaries, Snap, AppImage and FlatPak applications) plus all the speed & benefits of Direct Interception Mode whenever possible.

With Object Mount in FlexMount Mode, Object Mount will recognize that the mount path is an Object Storage mount and use **Direct Interception Mode** whenever possible for the fastest access. Object Mount will fall back to using **Object Mount on FUSE** for anything that cannot be directly intercepted. THis provides both the **best performance** and the **broadest compatibility**.

Object Mount's FlexMount benefits are leveraged by:

1. Configuring **Object Mount on FUSE**, and
2. Accessing your cloud storage via the **Object Mount CLI** by specifying your mount path via the `--flex` **parameter**.

Advantages

1. **Speed:** Where possible, Object Mount in FlexMount Mode will operate as fast as possible, using Direct Interception Mode.
2. **Compatibility:** Support for all POSIX applications: Anything that cannot be directly intercepted will pass through and be handled by Object Mount on FUSE.

Disadvantages

1. **Set up time:** Object Mount on FUSE needs to be configured.
2. **Shell launch is more complicated:** Each time an Object Mount shell is launched it must be configured to use the mount. This can be worked around by setting up a Object Mount on FUSE on boot, and setting an alias to launch a Object Mount shell with the correct parameters.

How to Enable

FlexMount is set up as follows:

1. First, set up a mount using Object Mount on FUSE:

   ```
   $ mkdir "$HOME/my-object-storage"
   $ cuno mount "$HOME/my-object-storage"
   ```

2. Always use the Object Mount CLI to interact with your object storage.

   Every time the Object Mount CLI is launched, use the `--flex <full_path_to_mount>` parameter.

   For example:

   ```
   $ cuno --flex "$HOME/my-object-storage"
   ```

3. Then reference your mount using the mount's path:

   ```
   (cuno) $ ls $HOME/my-object-storage/s3/<bucket>/<path>
   ```

> **NOTE**
>
> **FlexMount Tilde (~) Use**
>
> You cannot use a tilde `~` in your `CUNO_OPTIONS` or `cuno -o` CLI options when setting up a FlexMount, as this is something that the shell needs to resolve.
>
> You may still use it for your `cuno mount` commands, and subsequent FlexMount operations (such as `ls ~/my-object-storage`).

The same FlexMount can be re-used across multiple Object Mount-wrapped shells.

The `--flex` option can also be used with `cuno run` to run a single command or script with Object Mount enabled. Put `--flex` before `run`:

For example:

```
# terminal
cuno --flex "$HOME/my-object-storage" run bash -c "touch $HOME/my-object-storage/s3/<bucket>/
newfile"
```

FlexMount Interception Details

The `--flex <mount point>` option adds `cloudrootover=<mount point>` to `CUNO_OPTIONS`. With a path as its value, `cloudrootover` resolves each path and redirects paths inside the mount point to Direct Interception instead of the FUSE mount. If the path does not exist, `cuno` ignores `--flex` with a warning.

When using Object Mount Direct Interception in FlexMount Mode, most “local” paths are intercepted but not acted upon as they are handled by the local file system.

The `cloudrootover` setting tells Object Mount to intercept local paths that match the `cloudroot` setting, and to prioritize itself handling them via the mount. This means that whenever a path is recognized as the `cloudroot`, it will be more efficiently handled in user-space by the Direct Interception/`LD_PRELOAD` library.

Instead of `--flex`, you can set `cloudrootover` and `cloudroot` yourself. Path recognition can then be done in two ways:

- **Exact String Matching:** (`exact`) This method will match the `cloudroot` setting exactly. This is faster, and will end up relying on the FUSE mount whenever the paths don’t match the `cloudroot` — for example with symbolic links located outside the mount pointing into the mount.

  Exact String Matching is used when `cloudrootover` is given without a value: `-o cloudrootover -o cloudroot="<full path to mount point>"`.
- **Resolved Path Matching:** (`resolve`) This method will resolve the full path given in a file system call, including symbolic links, to check if the file is ultimately located inside the mount point.

  This requires more calls for every path-based file system call, so is slower when many files are being accessed. It is useful when symbolic links are pointed to the mount, and other cases where the path ultimately is inside the mount.

  This can be more efficient in some special cases (few files, large transfers, complex relationships between files) as `cuno.so` will intercept more calls without requiring them to go through the slower FUSE mount.

  To use Resolved Path Matching Mode, use the parameters `-o cloudrootover=resolve -o cloudroot="<full path to mount point>"`.

  For example:

```
$ cuno -o cloudrootover=resolve -o cloudroot="$HOME/my-object-storage"
(cuno) $ ls $HOME/my-object-storage/s3/<bucket>/<path>
```

Object Mount Fusion

> **WARNING**
>
> **Fusion Beta**
>
> Object Mount Fusion is currently in **Beta**.
>
> Your feedback via our 🌐 Support Team will help us improve it!

Overview

Object Mount Fusion is designed to enhance local, LAN-based, high-performance, attached storage solutions (like Amazon's Elastic File System, EFS) with the throughput of object storage.

It is a **cheaper** and **faster** solution when compared to using EFS alone.

Object Mount Fusion takes an attached storage filesystem and an (initially) empty object storage bucket/directory and exposes a single interface for both. Object Mount Fusion will migrate files between the object storage location and the local filesystem location depending on their best fit for both performance and cost.

How it Works

Object Mount Fusion combines both the **local** file storage and the **cloud** object storage into a single, virtual mount (a FUSE mount).

The files on object storage are represented as hidden links from the host filesystem to the object store.

Unlike other solutions, Object Mount enables cloud-based object storage to be used as a first-class, high-throughput tier, rather than a traditional slow archival tier.

Object Mount Fusion automatically and dynamically migrates files between the two locations according to your application behavior and usage. New files may be written into either location depending on predicted and observed access properties.

Object Mount Fusion supports multiple users, each simultaneously accessing files on multiple nodes, by combining and sharing the attached storage and mount locations.

Setting up Object Mount Fusion

Object Mount Fusion should be used *within* the same high-speed LAN as your object storage.

For example: If you use AWS S3, Object Mount Fusion should *only* be set up on EC2 nodes within the same AWS Region as the bucket to to be accessed.

If you are using an S3-compatible on-premises object storage solution, Object Mount Fusion should be set up on a computer on the same high-speed local area network (LAN).

Note: *If you have already set up and are using an attached storage system (such as EFS), you may skip ahead to* *Mounting a Object Mount Fusion filesystem*.

Otherwise, follow the steps below to configure Object Mount Fusion:

Setup an Empty Bucket on Object Storage

To use the object storage location when running Object Mount Fusion mount, you will need a location on object storage that is **empty**.

This will be used as the location that the Fusion filesystem will dynamically migrate data to when it's more suitably stored on object storage.

This can either be an **entirely empty bucket**, or an **empty directory** on an existing bucket. This location *must not* be modified by anything but Object Mount Fusion's filesystems.

Review the configuration steps below for your object storage provider:

AWS S3
AWS S3

To create a new AWS empty bucket, follow the instructions for 🌐 Creating a general purpose bucket.

Storj
Storj

To create a new Storj empty bucket, follow the instructions to Create buckets.

Microsoft Azure
Microsoft Azure

For a new empty container on Azure, follow the instructions to 🌐 Create a container.

Google Cloud
Google Cloud

For a new empty Google Cloud bucket, follow the instructions for 🌐 Create a bucket.

Other S3 Compatible
Other S3 Compatible

For a new empty bucket, follow your storage provider's instructions for setting up a new bucket.

Setup a New Compute Node

Object Mount Fusion needs to be set up on a compute node *within* the same LAN/region as the object storage bucket to be used.

You will also need to attach a high-speed file storage solution to the instance. It is recommended to configure this while creating the new instance/image.

To set up a compute node in the same region as your bucket, follow the relevant steps below for your provider:

AWS S3
AWS S3

Follow the AWS tutorial: 🌐 Get started with Amazon EC2.

Storj
Storj

Typically, you will need a physical or virtualized computer connected to the same LAN as both your object storage location and your file storage device.

Microsoft Azure
Microsoft Azure

Follow the Azure Virtual Machine Guide: 🌐 Quickstart: Create a Linux virtual machine in the Azure portal.

Google Cloud
Google Cloud

Follow the Google Cloud Compute Engine Guide: 🌐 Instance templates.

Other S3 Compatible

Other S3 Compatible

Typically, you will need a physical or virtualized computer connected to the same LAN as both your object storage location and your file storage device.

Create an Empty Directory on Attached Storage

You will need an empty directory on your attached file storage that is writable by your user.

If you don't have this set up yet, follow the relevant steps for your platform to attach a writable storage device to your compute node:

AWS S3

AWS S3

Follow the AWS guide: 🌐 Getting started with Amazon EFS.

Storj

Storj

You can use local storage, or attach an NFS/SMB mount to a file storage device in the same LAN.

Microsoft Azure

Microsoft Azure

Follow the Azure documentation to 🌐 Use the portal to attach a data disk to a Linux VM.

Additional information on Azure's fully managed file shares can be found in the 🌐 What is Azure Files? article.

Google Cloud

Google Cloud

Choose an appropriate Google Cloud attached disk option be reading through the information in the 🌐 Choose a disk type article. Then, attach your instance to it using the relevant guide.

Follow the instructions enable to Filestore in the 🌐 Install and enable components article.

Other S3 Compatible

Other S3 Compatible

You can use local storage, or attach an NFS/SMB mount to a file storage device in the same LAN.

Mounting an Object Mount Fusion Filesystem

To set up Object Mount Fusion, you need Object Mount installed on the compute node, and you need to Import S3 Credentials so that the bucket/container is accessible.

Your next steps depend on the filesystem you are using and the options used when mounting/attaching it.

Refer to the relevant section below:

- Filesystems *without* extended attribute support (e.g. EFS)
- Filesystems *with* extended attribute support (e.g. NFSv4 with Linux Kernel 5.9+)

Filesystems *Without* Extended Attribute Support (e.g. EFS)

Object Mount Fusion is set up and accessed through Object Mount on FUSE.

To do this, additional options must be passed to `cuno mount` when it is run.

You will need to:

- Use the `--fusion` option to specify the path to the associated attached storage directory
- Use the `--root` option to specify the object storage location as a URI, such as `s3://<bucket>/<path>`

Use a command similar to the one below to enable both these options:

```
cuno mount \
    --fusion "<path to attached-storage backing directory>" \
    --root "<URI of object storage backing directory>" \
    "<mount location>"
```

Example:

```
`cuno mount --fusion /dev/sdf/fusion-store --root s3://bucket/fusion-store $HOME/my-fusion-
        filesystem`
```

By running the above command, the `<mount location>` becomes the proper way to access the Fusion filesystem.

All operations, workflows, pipelines, etc. should be pointed to `<mount location>`.

Mounting the Filesystem

Refer to your environment for specific commands:

AWS S3

AWS S3

If the bucket is not empty, create a new empty directory on your bucket:

```
#terminal
cuno run mkdir "/cuno/s3/<bucket>/fusion-store"
```

Create a directory on the attached-storage device (assuming you've mounted it at `/mnt/fast`) to use for this purpose:

```
#terminal
mkdir "/mnt/fast/fusion-store"
```

Mount the Object Mount Fusion volume:

```
#terminal
cuno mount --fusion "/mnt/fast/fusion-store" --root "s3://<bucket>/fusion-store" "~/my-fusion-
filesystem"
```

Storj

Storj

If the bucket is not empty, create a new empty directory on your bucket:

```
#terminal
cuno run mkdir "/cuno/s3/<bucket>/fusion-store"
```

Create a directory on the attached-storage device (assuming you've mounted it at `/mnt/fast`) to use for this purpose:

```
#terminal
mkdir "/mnt/fast/fusion-store"
```

Mount the Object Mount Fusion volume:

```
#terminal
cuno mount --fusion "/mnt/fast/fusion-store" --root "s3://<bucket>/fusion-store" "~/my-fusion-filesystem"
```

Microsoft Azure

Microsoft Azure

If the bucket is not empty, create a new empty directory on your bucket:

```
#terminal
cuno run mkdir "/cuno/az/<account-name>/<bucket>/fusion-store"
```

Create a directory on the attached-storage device (assuming you've mounted it at `/mnt/fast`) to use for this purpose:

```
#terminal
mkdir "/mnt/fast/fusion-store"
```

Mount the Object Mount Fusion volume:

```
#terminal
cuno mount --fusion "/mnt/fast/fusion-store" --root "az://<account-name>/<bucket>/fusion-store" "~/my-fusion-filesystem"
```

Google Cloud

Google Cloud

If the bucket is not empty, create a new empty directory on your bucket:

```
#terminal
cuno run mkdir "/cuno/gs/<bucket>/fusion-store"
```

Create a directory on the attached-storage device (assuming you've mounted it at `/mnt/fast`) to use for this purpose:

```
#terminal
mkdir "/mnt/fast/fusion-store"
```

Mount the Object Mount Fusion volume:

```
#terminal
cuno mount --fusion "/mnt/fast/fusion-store" --root "gs://<bucket>/fusion-store" "~/my-fusion-filesystem"
```

Other S3 Compatible

Other S3 Compatible

If the bucket is not empty, create a new empty directory on your bucket:

```
#terminal
cuno run mkdir "/cuno/s3/<bucket>/fusion-store"
```

Create a directory on the attached-storage device (assuming you've mounted it at `/mnt/fast`) to use for this purpose:

```
#terminal
mkdir "/mnt/fast/fusion-store"
```

Mount the Object Mount Fusion volume:

```
#terminal
cuno mount --fusion "/mnt/fast/fusion-store" --root "s3://<bucket>/fusion-store" "~/my-fusion-
filesystem"
```

Note: *See tables below for additional* `cuno mount` *parameters.*

Filesystems *With* Extended Attribute Support (e.g. NFSv4 with Linux Kernel 5.9+)

When using a filesystem that supports extended attributes you should "bind" it to a cloud location first, as this does additional error checking and makes for a simpler mount/unmount procedure.

Bind a Directory to a Cloud Location:

To bind a local Fusion directory to a cloud location, use `cuno fusion`.

This command will configure a local pointer targeted at the cloud mount we will make later. It will also save any relevant options used at this stage as metadata; to be used by default by any mounts that use this fusion binding.

```
#terminal
cuno fusion "<path to attached storage backing directory>" "<path to object storage backing
directory>"
```

Example:

```
#terminal
cuno fusion /dev/sdf/fusion-store s3://bucket/fusion-store
```

Mounting the Filesystem (After Binding)

Object Mount Fusion is set up and accessed through Object Mount on FUSE.

To do this, additional options must be used when configuring Object Mount on FUSE.

If you have already bound a directory to a cloud location, you only need to use the `--root` option to specify the path to the associated attached-storage directory in order to use the mount as a Fusion mount:

```
# terminal
cuno mount \
    --root "<path to attached-storage backing directory>" \
    "<mount location>"
```

Refer to your environment for specific commands:

AWS S3

AWS S3

If the bucket is not empty, create a new empty directory on your bucket:

```
# terminal
cuno run mkdir "s3://<bucket>/fusion-store"
```

Create a directory on the attached-storage device (assuming you've mounted it at `/mnt/fast`) to use for this purpose:

```
#terminal
mkdir "/mnt/fast/fusion-store"
```

Create and mount the Object Mount Fusion volume:

```
#terminal
cuno fusion "/mnt/fast/fusion-store" "s3://<bucket>/fusion-store"
#terminal
cuno mount --root "/mnt/fast/fusion-store" "$HOME/my-fusion-filesystem"
```

Storj

Storj

If the bucket is not empty, create a new empty directory on your bucket:

```
#terminal
cuno run mkdir "s3://<bucket>/fusion-store"
```

Create a directory on the attached-storage device (assuming you've mounted it at `/mnt/fast`) to use for this purpose:

```
#terminal
mkdir "/mnt/fast/fusion-store"
```

Create and mount the Object Mount Fusion volume:

```
#terminal
cuno fusion "/mnt/fast/fusion-store" "s3://<bucket>/fusion-store"
#terminal
cuno mount --root "/mnt/fast/fusion-store" "$HOME/my-fusion-filesystem"
```

Microsoft Azure

Microsoft Azure

If the bucket is not empty, create a new empty directory on your bucket:

```
#terminal
cuno run mkdir "az://<account-name>/<bucket>/fusion-store"
```

Create a directory on the attached-storage device (assuming you've mounted it at `/mnt/fast`) to use for this purpose:

```
#terminal
mkdir "/mnt/fast/fusion-store"
```

Create and mount the Object Mount Fusion volume:

```
#terminal
cuno fusion "/mnt/fast/fusion-store" "az://<account-name>/<bucket>/fusion-store"
#terminal
cuno mount --root "/mnt/fast/fusion-store" "$HOME/my-fusion-filesystem"
```

Google Cloud

Google Cloud

If the bucket is not empty, create a new empty directory on your bucket:

```
#terminal
cuno run mkdir "gs://<bucket>/fusion-store"
```

Create a directory on the attached-storage device (assuming you've mounted it at `/mnt/fast`) to use for this purpose:

```
#terminal
mkdir "/mnt/fast/fusion-store"
```

Create and mount the Object Mount Fusion volume:

```
#terminal
cuno fusion "/mnt/fast/fusion-store" "gs://<bucket>/fusion-store"
#terminal
cuno mount --root "/mnt/fast/fusion-store" "$HOME/my-fusion-filesystem"
```

Other S3 Compatible

Other S3 Compatible

If the bucket is not empty, create a new empty directory on your bucket:

```
#terminal
cuno run mkdir "s3://<bucket>/fusion-store"
```

Create a directory on the attached-storage device (assuming you've mounted it at `/mnt/fast`) to use for this purpose:

```
#terminal
mkdir "/mnt/fast/fusion-store"
```

Create and mount the Object Mount Fusion volume:

```
#terminal
cuno fusion "/mnt/fast/fusion-store" "s3://<bucket>/fusion-store"
#terminal
cuno mount --root "/mnt/fast/fusion-store" "$HOME/my-fusion-filesystem"
```

Note: *See tables below for additional* `cuno mount` *and* `cuno fusion` *parameters.*

Using & Testing the Object Mount Fusion filesystem

All operations, workflows, pipelines, etc. should be pointed to your mount location.

Usage Examples

The following examples assume you have mounted a Fusion filesystem at `$HOME/my-fusion-filesystem`.

- Download data from the web into the Fusion filesystem:

```
#terminal
cd $HOME/my-fusion-filesystem
#terminal
wget http://vision.stanford.edu/aditya86/ImageNetDogs/images.tar
```

- List the files in the Fusion filesystem:

```
#terminal
ls $HOME/my-fusion-filesystem/
```

- Unpack a tar archive:

```
#terminal
cd $HOME/my-fusion-filesystem
#terminal
tar -xf images.tar
```

Testing

Check that access, modification and migration are working as expected.

We recommend measuring cost differences, and the overall time taken for equivalent jobs run on attached storage alone.

Note: You can set the `CUNO_LOG` environment variable to `trace`, or `debug` at the time of mounting for more details logging information.

Not seeing what you expect? Contact us at our 🌐 Support Team.

APPENDIX TABLES: Parameters, Arguments & Options

TABLE: `cuno fusion` Arguments

Argument	Description
`<path to attached storage backing directory>`	Sets the location specified as the place to store file-storage data. This is where files that are migrated off the object store into local storage go. This can happen when access patterns suggest the file would benefit from higher IOPS, or when a file doesn't meet the minimum size or age thresholds.
`<path to object storage backing directory>`	Text

TABLE: `cuno fusion` Options

Option	Description
`--fusion-size-threshold <size: default 10M>` (optional)	Define the minimum size that a file needs to be for it to be migrated from file storage to object storage. The argument value is of the form `<INTEGER>[UNIT]`. If no unit is given, the value is assumed to be in *bytes*. Valid units are `K` (Kilobytes), `M` (Megabytes), `G` (Gigabytes), `T` (Terabytes).

Option	Description
`--fusion-age-threshold <age: default 1h>` (optional)	Defines the minimum age requirement for files to be considered for migration from file storage to object storage. The age is measured as the time since the most recent of the POSIX creation time, modify time, access time, and change time - in other words, the time since each of these must be greater than the set threshold. The argument value is of the form `<INTEGER>[UNIT]`. If no unit is given, the value is assumed to be in seconds. Valid units are `s` (seconds), `m` (minutes), `h` (hours), `d` (days).

TABLE: `cuno fusion` Subcommands

Subcommand	Description
`cuno fusion bind <local> <cloud>`	Bind a local directory to a cloud directory.
`cuno fusion unbind <local>`	Unbind a local directory from a cloud directory.
`cuno fusion rebind <local> <cloud>`	Rebind a local directory to a cloud directory.
`cuno fusion info <local>`	Get information about the binding at the location.

TABLE: Relevant `cuno mount` Options for Object Mount Fusion:

Argument/Option	Description	Example
`<mount location>`	The location from which the Fusion filesystem will be accessed.	`$HOME/my-fusion-filesystem`
`--fusion "<path to attached storage backing directory>"`	Enables Fusion for this mount, and sets it to use the location specified to store file-storage data. This is where files that are migrated off the object store into local storage go. This can happen when access patterns suggest the file would benefit from higher IOPS, or when a file doesn't meet the minimum size or age thresholds.	`/dev/sdf/fusion-store`
`--root "<URI of object storage backing directory>"`	Sets the object storage location to be used as the place to store data that is better suited to be stored on object storage rather than on the file storage. Must be a URI such as `s3://bucket/path`.	`s3://bucket/fusion-store`
`--fusion-size-threshold <size: default 10M>` (optional)	Defines the minimum size that a file needs to be for it to be migrated from file storage to object storage. The argument value is of the form `<INTEGER>[UNIT]`. If no unit is given, the value is assumed to be in *bytes*. Valid units are `K` (Kilobytes), `M` (Megabytes), `G` (Gigabytes), `T` (Terabytes).	
`--fusion-age-threshold <age: default 1h>` (optional)	Defines the minimum age requirement for files to be considered for migration from file storage to object storage. The age is measured as the time since the most recent of the POSIX creation time, modify time, access time, and change time - in other words, the time since each of these must be greater than the set threshold. The argument value is of the form `<INTEGER>[UNIT]`. If no unit is given, the value is assumed to be in seconds. Valid units are `s` (seconds), `m` (minutes), `h` (hours), `d` (days).	

Enable Enforced POSIX File Access

This guide represents the general process for setting up “Enforced POSIX File Access” mode in Object Mount for Linux.

Enforced POSIX File Access mode will maintain POSIX metadata for your objects, and will enforce POSIX access controls on those objects. Use this when you want to manage what users have access to based on the UID/GID of their UNIX user and the corresponding POSIX metadata (owner, group, mode) on files.

Users will encounter `access denied` errors if they try to read or write to a file/directory they haven’t been given permission to (by a suitably privileged user doing `chown`, `chgrp` or `chmod`). Note that this is *client-side* rather than *server-side* enforcement, and is not enforced using server-side IAM roles or ACL lists.

The specific steps may vary depending on your object storage provider. Contact the Storj 🌐 Support Team if this is a use-case that you are interested in and these instructions don’t match your expectations or preferences.

> **NOTE**
> **Hybrid Deployment for ACL Support**
>
> We can also provide a hybrid-approach, which does *both* client-side access management and server-side access management through ACLs.
>
> If you need server-side ACL policies to reflect POSIX access controls, contact the Storj 🌐 Support Team for assistance.

Configuration Overview

From your object storage provider’s dashboard, generate access credentials with the most complete set of access permissions that any user or admin could need, including permissions to *edit bucket tags*.

- On a public cloud supporting IAM, you should set up an `admin` IAM user with such credentials.
- The admin credentials are stored privately and are used to set up Object Mount on FUSE in an accessible location.
- Users are only told the path to the mount — they are not given access to `cuno` nor to the admin credentials.

> **WARNING**
> **Security Considerations**
>
> The approach given here is **not suitable** if users have the ability to create their own VMs/instances where they can set any POSIX uid/gid.
>
> In that case, consider using server-side IAM roles or ACL lists to enforce access controls. Contact our 🌐 Support Team for advice on how this integrates with Object Mount.

Key Considerations

1. If multiple machines or locations are being used, it is important that user and group IDs are controlled and consistent across all machines.

   This is because the POSIX metadata is stored in the object storage, and Object Mount will use the same user and group IDs to present the files on all machines. If you require assistance with managing user and group IDs across multiple machines, contact our 🌐 Support Team.

2. Independent IAM roles or credentials for each user are not required. The admin credentials are used to set up the mount, and the users are only given access to the mount (not the credentials).

   If the user has access to object storage credentials with server-side privileges beyond this, then the user can potentially access or modify objects outside of these POSIX access controls.
3. This guide assumes that the *bucket to use is empty* initially. If you are switching from an ACL-based approach to access management, then you will need to map existing rules to POSIX uids/gids and "apply" those ACLs through POSIX commands like `chown` and `chmod`.

Note: Object Mount does not currently support POSIX ACLs or extended attributes in cloud object storage. Contact 🌐 Storj Support if you need these features.

How to Enable POSIX Enforced File Access

Assuming you have already installed Object Mount for Linux, the following steps will guide you through setting up enforced POSIX access for the users in a generic POSIX-based system.

Step 1. Import your Admin Credentials

Follow the steps in to obtain, save and import your admin credentials.

You will need to have run `cuno creds import <file containing admin credentials>`.

After importing, you should ensure that any credentials files (usually in `${XDG_CONFIG_HOME}"/cuno/creds` or `~/.config/cuno/creds`) have been created with, or assigned the, appropriately strict permissions so that non-admin users cannot read them.

Step 2. Tag the Bucket

We check a bucket tags when Object Mount tries to access a bucket to assess the POSIX enforcement settings.

If your bucket or object storage provider does *not support tags* on buckets, you can skip this step and proceed to Step 3. Mount the Bucket.

Note: You will need to set the `--posix` flag every time you run `cuno mount`.

Use `cuno creds setposix <uri of bucket> true` to set the bucket tag which will automatically enable POSIX File Access whenever this bucket is accessed by any Object Mount installation.

This will affect *everyone* using the bucket and force all Object Mount users of that bucket into "Enforced POSIX File Access" mode.

Example:

```
# terminal
cuno creds setposix s3://mybucket true
```

By setting this to `true`, Object Mount stores hidden subdirectories inside your object storage directories describing the POSIX metadata (owner/group permissions, change/modify times, etc.) for each file in the directory. This allows Object Mount to present the files in a way that is compatible with POSIX file access semantics.

NOTE

File-Based Metadata

If you are using S3 buckets, it is possible to set `cuno creds setposix <uri of bucket> metadata`. This will store POSIX metadata as object metadata *on each individual file* in the bucket,

We do not normally recommend this, as it will slow down Object Mount. For more information, see .

WARNING

Inability to Enforce File Access

When a bucket tag enabling POSIX File Access is set, Object Mount Direct Interception, along with any Object Mount on FUSEs/FlexMounts, are compelled to operate in POSIX mode while accessing the bucket.

However, the mounts will *not be able to enforce POSIX access* without the additional `--posix` flag at mount time.

Object Mount Direct Interception Mode necessitates read access to object storage credentials (or some other user-accessible way to authenticate with the object storage (e.g. an IAM role on EC2)].

This implies that Direct Interception Mode *is not appropriate for enforcing POSIX*, as users can easily circumvent this by using the accessible credentials or IAM role with a tool other than Object Mount (such as AWS CLI, or directly through an S3 API library).

Since Object Mount FlexMounts are dependent on Object Mount Direct Interception Mode, they are also unsuitable for use-cases that require POSIX enforcement.

Step 3. Mount the Bucket

Mount a bucket in a location that is accessible to the users.

In this guide, we will use a shared location of `/mnt/cloud/bucket` as the mount point.

`--posix` sets the FUSE option `-o allow_other`, so **other users**, including root, can access the mount as themselves:

```
$ cuno mount --posix --root s3://mybucket /mnt/cloud/bucket
```

Do not add `-o allow_root`: FUSE refuses to mount with both `allow_root` and `allow_other`. If you mount as a non-root user, `allow_other` requires `user_allow_other` in `/etc/fuse.conf`.

Users will now be able to see the files in the bucket at `/mnt/cloud/bucket`, and POSIX permissions persisted by Object Mount for files in that bucket will be respected.

You should treat this as any other POSIX file system and set permissions as you need to control access.

Common Patterns

Below are some common behaviors and recommendations that you might want to implement when using enforced POSIX access.

For more examples, see the **Usage Examples** section in the Advanced Guide article .

Setting Default Permissions for New Files

On POSIX systems, you would normally set default permissions using `umask`.

To determine the umask value you want to set, subtract the value of the permissions you want from `666` (for a file) or `777` (for a directory).

For example, if `user1` wants the default permissions on new files to be `rw-r--r-- 644`, they would subtract `644` from `666`, giving `022`.

```
$ umask 022
$ touch /mnt/cloud/bucket/user1/newfile
$ ls -l /mnt/cloud/bucket/user1/newfile
-rw-r--r-- 1 user1 group1 0 Mar  1 12:00 /mnt/cloud/bucket/user1/newfile
```

If you want to set the umask for **all users**, you can set this in the system-wide profile, or in the user's profile.

For example, to set it for all users, you could add the umask command to `/etc/profile`.

See the **Ownership and Permissions** section in the Advanced Guides article for additional details on file and directory ownership and permissions.

Converting a Bucket to "POSIX Enforced File Access" Mode

If you have a pre-existing storage location for which you now need to implement POSIX-based access controls, you can simply set the bucket tag as shown above. This will ensure that any *new files* are stored with POSIX metadata.

Note: This will *not* affect any *existing files* until they (or their directories) are accessed.

To control access to any particular files or directories, proceed with the use of `chown`, `chgrp` and `chmod` as you would normally on a POSIX file system.

For example, to limit access to `/mnt/cloud/bucket/directory` to only allow `user1`, you could do the following:

```
$ chown -R user1 /mnt/cloud/bucket/directory
$ chgrp -R group1 /mnt/cloud/bucket/directory
$ chmod -R 700 /mnt/cloud/bucket/directory
```

As a result of the above commands, only `user1` will be able to access the directory and its contents across all systems using Object Mount.

You can check the POSIX metadata for a file or directory using `ls -l` or `stat`.

For example:

```
$ ls -l /mnt/cloud/bucket/directory
drwx------ 1 user1 group1 0 Mar  1 12:00 /mnt/cloud/bucket/directory
$ stat /mnt/cloud/bucket/directory
File: /mnt/cloud/bucket/directory
Size: 0             Blocks: 0          IO Block: 4096   directory
Device: 1ch/28d     Inode: 123456      Links: 1
Access: (0700/drwx------)  Uid: ( 1000/  user1)   Gid: ( 1000/  group1)
Access: 2022-03-01 12:00:00.000000000 +0000
Modify: 2022-03-01 12:00:00.000000000 +0000
Change: 2022-03-01 12:00:00.000000000 +0000
 Birth: -
```

Setting Inherited Permissions on a Directory

> **WARNING**
>
> **POSIX ACL Support**
>
> Object Mount does not support POSIX ACLs, so default ACLs set with `setfacl` cannot be used to set inherited permissions.
>
> If this is a feature you need, please contact our 🌐 Support Team.

To control the permissions of new files, set a `umask` as described in Setting Default Permissions for New Files.

Cloud Path Details

When using Object Mount for linux from the Object Mount CLI

Specifying Cloud Paths

When accessing cloud-based, object storage volumes using the Object Mount CLI, there are two primary ways of interacting with the remote files and directories:

- The URI format, e.g.: `s3://my-bucket/dir/file.txt`
- The directory format, e.g.: `/cuno/s3/my-bucket/dir/file.txt`

URI Format

The following table describes **URI prefixes** for supported cloud providers:

Cloud Provider	URI Format
AWS S3	`s3://<bucket_name>`
Storj	`s3://<bucket_name>`
Google Cloud Storage	`gs://<bucket_name>`
Azure Blob Storage	`az://<account_name>/<container_name>`
Other S3 Compatible	`s3://<bucket_name>`

Directory Format

The following table describes **directory format** for supported cloud providers:

Cloud Provider	Directory Format
AWS S3	`/cuno/s3/<bucket_name>`
Storj	`/cuno/s3/<bucket_name>`
Google Cloud Storage	`/cuno/gs/<bucket_name>`
Azure Blob Storage	`/cuno/az/<account_name>/<container_name>`
Other S3 Compatible	`/cuno/s3/<bucket_name>`

> **WARNING**
> **URI Format Support**
>
> Some software tools cannot process the URI format. Use the directory format instead in such cases.

Examples

Some examples of using Object Mount are given below:

1. List the contents of a remote container `my-bucket` on AWS S3:

```
(cuno) $ ls s3://my-bucket
```

or

```
(cuno) $ ls /cuno/s3/my-bucket
```

2. Move a text file `sample.txt` from the local filesystem to a remote Google Cloud Storage container `my-bucket`:

```
(cuno) $ mv ./sample.txt gs://my-bucket/dir/sample.txt
```

or

```
(cuno) $ mv ./sample.txt /cuno/gs/my-bucket/dir/sample.txt
```

3. List the contents of a subdirectory `dir` in a remote Google Cloud Storage container `my-bucket`:

```
(cuno) $ ls gs://my-bucket/dir
```

or

```
(cuno) $ ls /cuno/gs/my-bucket/dir
```

4. Use UNIX coreutils (e.g. `head`) on a remote file `sample.txt` on AWS S3:

```
(cuno) $ head s3://my-bucket/dir/sample.txt
```

Microsoft Azure Blob Storage Usage

When using Microsoft's Azure Storage, the associated **Account Name** must be included in both the URI and directory format:

```
(cuno) $ ls az://my-account-name/my-container
```

or:

```
(cuno) $ ls /cuno/az/my-account-name/my-container
```

AWS S3 Access Point Support

Object Mount supports using an **AWS S3 Access Point** instead of a bucket as an endpoint.

Access Points are unique addresses that customers can create to enforce distinct permissions and network controls for requests made through that Access Point.

To use an Access Point instead of a container, provide the full **Amazon Resource Name (ARN)** in place of the bucket name:

```
s3://arn:aws:s3:us-east-1:999999999999:accesspoint:my-access-point-name
```

This applies to all places a container can be specified, such as within `cuno creds` or console commands:

```
(cuno) $ cuno creds -i pair s3://arn:aws:s3:us-east-1:999999999999:accesspoint:my-access-point-
         name
(cuno) $ ls s3://arn:aws:s3:us-east-1:999999999999:accesspoint:my-access-point-name/<remote-path>
(cuno) $ ls /cuno/s3/arn:aws:s3:us-east-1:999999999999:accesspoint:my-access-point-name/<remote-
         path>
```

File System Behaviors When Using Object Mount

Symbolic Links

WHen not using POSIX Mode, symbolic links can **only** be created (via `ln --symbolic` or `ln -s`) to point **from** a local system **to** a cloud location. Either URI or directory format can be used.

Hard Links and Server-Side Copy

Conversely, hard links **cannot** be created from the local system to a remote cloud location.

Use hard-linking to trigger a more efficient cloud-only copy mechanism within a cloud region.

`mv` or `cp` within a single cloud vendor will trigger a **server-side copy**, which is more efficient than a download followed by an upload.

POSIX Mode

Symbolic links can be created (via `ln --symbolic` or `ln -s`) between local and cloud objects in **either direction** when POSIX mode in enabled.

Hard links can be created (using `ln`) between cloud objects in the `same` bucket when POSIX mode in enabled.

See the Getting Started Guide article POSIX File Access for more details.

> **WARNING**
>
> **HardLinks and SymLink on File Deletion**
>
> If the target file/object is deleted or moved, both symbolic and hard links will break.

File Metadata

`ls --long` (or `ls -l`) will list **the current user** as the owner of the remote objects, and the permissions will be `777`.

The creation date of a remote directory is not always available to system calls.

Performance Optimizations

Object Mount uses multiple concurrent connections to remote objects to achieve the highest performance.

It can also execute operations on objects entirely remotely where possible.

Some optimized operations provided by Object Mount are listed below:

- `mv` commands between locations **within the same object storage provider** will invoke a server-side remote-copy operation to avoid streaming through the client.
- `ln` hard links between locations **within the same object storage provider** will invoke a server-side remote-copy operation to avoid streaming through the client.
- Hard links between objects **in the same bucket** will work like local hard links when POSIX mode in enabled.

Application Tips

This article provides additional recommendations and guidance when using Object Mount for Linux with the following applications:

Python

Python's `os.path.realpath(path)` does *not* support URI formatted cloud paths (e.g.: `s3://<bucket_name>/path`).

Use directory formatted cloud paths instead (e.g.: `/cuno/s3/<bucket_name>/path`).

Rsync

We strongly recommend running `rsync` with the options `--inplace -W`. This makes Rsync work more efficiently with object storage.

To use Rsync options that preserve permissions (`-p`) and modification times (`-t`), such as when you want to update files *only* when the source has been updated, you must enable POSIX File Access.

See the Getting Started Guide: POSIX File Access for details.

Fpsync

When using `fpsync`, use the `-o` option to pass the options recommended for `rsync` down to the worker processes (e.g.: `-o "--inplace -W"`).

Further, because Object Mount is already parallelized, we recommend limiting the number of Fpsync worker processes using the `-n` option.

FFmpeg

While `ffmpeg` works, it may be slow in the following use-cases:

- Writing to object storage: If the task requires many randomly-placed writes during an upload. For example, for a large mp4 file being written with the flags `-movflags +faststart`.
- Reading from object storage: A complex filter chain, where multiple subtitle streams are being read from the same input file.
  - Improvements in this area are being developed.
- Reading from object storage: Files that contain title screen and extra credit scenes can be slow to start.

Using sudo with Direct Interception

Using Direct Interception requires the `LD_PRELOAD` environment variable to be set and maintained for all executed child processes.

Since `sudo` usage, by default, does not preserve the environment variables set, the following requirements apply:

- `sudo` needs to be run with `--preserve-env` to preserve `CUNO_OPTIONS`.
- `sudo` needs to launch a child shell that will then run the command, so that the `LD_PRELOAD` environment variable can be set *before* running the command to be intercepted.
- `LD_PRELOAD` needs to be manually set inside the child shell launched.

To use `sudo` with Direct Interception, please do the following:

1. Start a wrapped-shell using the Object Mount CLI: `cuno`
2. Run `sudo` in the following way:

```
sudo --preserve-env /bin/bash -c "export LD_PRELOAD=$LD_PRELOAD && <YOUR COMMAND HERE>"
```

Locate and UpdateDB

The `locate` command requires some heightened privileges to create the file path database. It also has some incompatibilities with Object Mount Direct Interception Mode.

When using Direct Interception Mode, both `locate` and `updatedb` will not work with URI formatted cloud paths (e.g.: `s3://<bucket_name>/path`).

Use directory formatted cloud paths instead (e.g.: `/cuno/s3/<bucket_name>/path`).

Furthermore, Direct Interception (even when using the Object Mount CLI) requires the `LD_PRELOAD` environment variable to be set and maintained for all executed child processes.

Note: Since `updatedb` typically needs to be run with `sudo`, the limitations specified for `sudo` **with Direct Interception** (above) apply here as well.

Instructions for Running UpdateDB

To help work around these limitations, follow the guidance below when using `locate` and `updatedb`:

1. **Create a New Database**

Direct Interception Mode

Run the Object Mount CLI by entering the `cuno` command:

```
# terminal
cuno
```

Then run `updatedb` inside the Object Mount-wrapped shell, using the `sudo` recommendations with cloud paths specified in the directory format. (We use an example database name of: `cunoloc.db`).

```
(cuno) $ sudo --preserve-env /bin/bash -c "export LD_PRELOAD=$LD_PRELOAD && updatedb -U /cuno/<s3/
az/gs>/<bucket> -o cunoloc.db"
```

Object Mount on FUSE

Assuming you have abject Mount on FUSE set up at `~/my-object-storage`, you can use `updatedb` *directly* to crawl all paired buckets from all your object storage providers:

```
sudo updatedb -U ~/my-object-storage -o cunoloc.db
```

> **NOTE**
>
> **Mount Location Consistency**
>
> The mount location (`~/my-object-storage` in the example above) should not be change after running `updatedb` as the mount details are written into the file path database.

1. **Change the Database Ownership**

   The newly created database needs to have its ownership changed back to the current user:

```
sudo chown $(whoami):$(whoami) cunoloc.db
```

2. **Add the Database to Your `LOCATE_PATH`**

   Update the `LOCATE_PATH` environment variable to include the database.

   Then use locate normally, or search within the database, as follows:

   ```
   locate -d cunoloc.db myfile
   ```

Modify the Locate cron Job

By default, the global locate database is periodically updated by a cron job.

To setup the cron job to work properly with Object Mount, you need to edit the file `/etc/cron.daily/mlocate`.

The last line of that file is used to update the global database:

```
flock --nonblock /run/mlocate.daily.lock $NOCACHE $IONICE nice /usr/bin/updatedb.mlocate
```

Modify that line, replacing it with something similar to the line below:

```
LD_PRELOAD='/usr/lib/cuno.so' CUNO_OPTIONS='<your options>' CUNO_CREDENTIALS='<path to credentials usable by the root user>' flock --nonblock /run/mlocate.daily.lock $NOCACHE $IONICE nice /usr/bin/updatedb.mlocate
```

If you prefer not to index *all* of your object storage files, you can specify where `updatedb` does *not* look for files by adding paths to `PRUNEPATHS` in the file `/etc/updatedb.conf`.

Media & Entertainment and HPC Modes

Object Mount for Linux reads the environment variable `CUNO_PROFILE` to choose how it downloads data from object storage. It does not change any other behavior.

CUNO Profile Setting	Download Behavior
unset (default)	The default reader, suited to interactive and Media & Entertainment (M&E) workflows
`CUNO_PROFILE=hpc`	A reader tuned for High-Performance Computing (HPC) throughput
`CUNO_PROFILE=direct`	Direct reads from object storage

The value is not case-sensitive. Any other value, including `M&E`, uses the default reader.

Choosing the Right Profile

Leave `CUNO_PROFILE` unset for interactive and creative workloads, such as desktop video editing and media previews.

Set `CUNO_PROFILE=hpc` for bulk, high-throughput workloads, such as automated ingest pipelines, render farm nodes and batch jobs.

Measure your own workload with each setting if you are unsure.

Setting the Profile

Define the environment variable *before* launching Object Mount.

For example:

```
export CUNO_PROFILE=hpc
```

For permanent use, add the environment variable setting to:

- Your shell profile (`~/.bashrc`, `~/.zshrc`)
- A systemd service or login script
- Docker or container entrypoints (if applicable)

Advanced Guides for Linux

The articles in this Advanced Guides section provide expert-level configuration and deployment advise regarding Object Mount for Linux.

Articles include:

- Advanced Guides
- Advanced Credential Options
- Advanced Loading Options
- Advanced Configuration Options
- Server-Side Encryption
- Kubernetes CSI Driver
- Object Mount's Machine Learning Accelerator
- Technical Limitations

Advanced Credential Options

Obtaining and Saving your S3 Credentials

Users will need to obtain the credentials for their S3 cloud storage accounts and save those credentials to a text file.

Refer to the Appendix article for instructions on how to obtain and save your S3 credentials.

Credential File Location

Object Mount for Linux uses the `cuno creds` command to import and manage your S3 credentials.

Linux credentials are stored in `"${XDG_CONFIG_HOME}"/cuno/creds` by default (or `~/.config/cuno/creds` if `XDG_CONFIG_HOME` if not set).

You can override the defaults by setting the `CUNO_CREDENTIALS` environment variable.

> **NOTE**
>
> **Credential Storage**
>
> `cuno creds` stores credentials in `$CUNO_CREDENTIALS` if it is set, and otherwise in `"${XDG_CONFIG_HOME}"/cuno/creds` (or `~/.config/cuno/creds`).
>
> When Object Mount reads credentials, it uses the first of these directories that exists:
>
> 1. `$CUNO_CREDENTIALS`
> 2. `"${XDG_CONFIG_HOME}"/cuno/creds` (or `~/.config/cuno/creds` if `XDG_CONFIG_HOME` is not set)
> 3. `cuno-config/creds` next to the installation directory, e.g. `/opt/cuno-config/creds` for an installation in `/opt/cuno`

Endpoint Details

Endpoints can be IP addresses or domain URIs (optionally with port numbers).

Note that a schema is required (e.g. `http://` or `https://`).

The following are examples of valid endpoint key-value pairs:

```
endpoint = http://127.0.0.1:8080
endpoint = https://s3objectstorage.example.com
```

Global S3 Configuration File Options

Depending on the capabilities and behaviors of your object storage provider's S3 API, you may find the additional keys useful:

Option	Description
`skipssl`	Skip SSL certificate verification. Useful for private storage solutions with self-signed certificates.
`nossl`	Use HTTP protocol instead of HTTPS. Use when the endpoint starts with `http://`.
`pathstyle`	Switch from virtual host style to path style syntax for S3-compatible storage solutions. Enable for private storage solutions supporting only path style syntax (e.g. MinIO, Ceph).
`no-delete-objects`	Do not use `DeleteObjects` to remove multiple objects. Instead, use `DeleteObject` for single object removal.

Option	Description
`no-remote-copy`	Do not copy objects within the same bucket via the server — download and upload instead.
`no-object-tagging`	Do not attempt to get/put object tags.
`no-object-acl`	Do not attempt to get/put object ACLs.
`directory-marker`	Create a placeholder object when creating an empty prefix (e.g., `mkdir`). Remove the placeholder object when removing an empty directory.
`ignore-empty-prefixes`	Ignore directories that were removed but still reported as existing prefixes by the server.
`posix=<true/false>`	Set POSIX mode for all buckets associated with this credential. This option takes precedence over the POSIX mode runtime environment variable, but may be overridden by a bucket-wide POSIX setting (set via `cuno creds`).

NOTE

Detect Features

For S3 buckets, `no-delete-objects` and `no-remote-copy` can be detected and added by running `cuno creds detectfeatures <URI> <imported credential file>`. Give the name of an imported credential file as shown by `cuno creds list` (e.g. `cuno creds detectfeatures s3://test-bucket test_credentials.s3c`); any other argument is treated as a file path, and the results are appended to that file instead.

This command needs a bucket to be specified that it can write temporary files to for testing purposes. The command will run a series of tests against the specified bucket, checking the availability of S3 features that Object Mount uses and updating the credentials file accordingly.

Note: Running feature detection will use up to a few gigabytes of bandwidth and may take a few minutes to complete depending on the machine's connection speed and the S3-compatible storage provider.

Microsoft Azure Storage

Azure Blob Storage credential files have the following format:

```
AZURE_STORAGE_ACCOUNT    = <account_name>
AZURE_STORAGE_ACCESS_KEY = <access_key>
```

If you need the access key, see Microsoft's 🌐 View account access keys article.

Microsoft Azure Storage with Shared Access Signatures

Object Mount also supports using **Shared Access Signatures (SAS)** to access remote containers in Azure.

Shared Access Signatures permit access to storage resources *without* sharing access keys with users.

An SAS gives user's access to a container for only specified intervals and with specified permissions.

To provide a SAS to Object Mount use the following command:

```
cuno creds pair <container_remote_uri> "?<SAS>"
```

Note: The question mark `?` prefix and quotation marks `"` are required.

Google Cloud Storage

You will need key-based access to be associated with your Google Cloud service account.

If you need to set this up, you can have Google generate a new key by following Google's guide, Create and delete service account keys. Alternatively, you can generate your own key pair and upload it on the service account's key management page, by following the guide Upload service account keys.

If you've generated a new key, store the JSON file you have downloaded in a safe place with appropriate access permissions, ready for the next steps.

Importing Credentials

Every cloud account requires a unique credential file.

Import a credential file into Object Mount using the following command:

```
cuno creds import <credential_file>
```

Object Mount performs the following actions when importing a credential file:

1. Detect the credential file format, and connect to the remote server.
2. Discover all possible containers that can be accessed using the provided credentials.
3. Create pairings between the imported credential file and remote containers.

> **NOTE**
> **Listing Permission are Required**
>
> Discovery only completes when the provided credentials include bucket listing permissions.
>
> If listing permissions cannot be provided, manually pair the imported credential file to remote buckets using `cuno creds pair` (see below)`.

Pairing Containers and Credentials

Manually pair additional containers with previously-imported credential files using `pair`:

```
cuno creds pair <container_remote_uri> [imported_credential_file]
```

- `<container_remote_uri>` is the remote URI of the container to be associated with the credential file.
- `[imported_credential_file]` is optional and corresponds to the credential file that has already been imported.

If the latter is not provided, all imported credential files are listed.

To re-associate a previously paired remote container with a different credential file, use `pair` with the same remote URI and a different credential file.

> **NOTE**
> **Automatic Verification**
>
> Object Mount verifies that a remote container is accessible when pairing it with a credential file.
>
> Object Mount auto-detects an AWS region if it is not specified, or is invalid.

Public Access Buckets

To pair a remote URI that has public access, use the following command:

```
cuno creds pair <bucket_remote_uri> public
```

Notice `public` is used in the place of the imported credential file name.

"Requester Pays" buckets

Amazon AWS S3 and Google Cloud Storage offer "Requester Pays" features for containers.

In these cases the requesters, not the bucket owners, are billed when remote objects are accessed.

By default, Object Mount skips "Requester Pays" containers when importing a credential file to avoid unexpected charges. To enable access to such containers with a previously-imported credential file, run the following command:

```
cuno creds --interactive pair <bucket_remote_uri>
```

The `--interactive` (`-i`) option will prompt the user to select from credentials in Object Mount's database.

"Requester Pays" with Google Cloud Storage

When you run the run the `cuno creds --interactive pair <bucket_remote_uri>` command, it will list all available project IDs associated with the selected credential file and available for billing.

Select and confirm a "Requester Pays" container to pair as prompted.

To run the command non-interactively, enter:

```
cuno creds pair --billing <billing_project_ID> <bucket_remote_uri> <imported_credential_file>
```

"Requester Pays" with Amazon AWS S3

To force a pairing between a "Requester Pays" container with an imported credential file use the following:

```
cuno creds --force pair <bucket_remote_uri> <imported_credential_file>
```

Listing Credentials

`list` displays information about imported credential files and paired buckets.

The command accepts additional options:

Command	Description
`cuno creds list`	Display all imported credential files and all paired buckets
`cuno creds list creds`	Display all imported credential files **only**
`cuno creds list pairings`	Display all paired buckets **only**
`cuno creds list pairings <provider>`	Display all paired buckets for the specified provider **only**

`pairings <provider>` accepts a comma-separated list of providers.

For example: `cuno creds list pairings gs,s3` will display information about all paired buckets in Google Cloud Storage and AWS S3.

Unpair and Purge Credentials

To dissociate a container from a credential file, use `unpair`:

```
cuno creds unpair [bucket_remote_uri]
```

If the second argument is missing, then existing pairings are listed.

To completely remove a credential file and delete all its associations with remote containers, use `purge`:

```
cuno creds purge [imported_credential_file]
```

If the second argument is missing, then existing credential files are listed.

Additional Flags

Object Mount"s `cuno creds` command accepts additional flags:

```
cuno creds [flags] <rest_of_command>
```

These flags are described in the following table:

Flag	Description
`--interactive/-i`	Asks for permission to replace existing credential files, pairings, SAS, etc.
`--force/-f`	Always replaces existing credential files, pairings, SAS, etc.
`--color/-c`	Displays using colored output.

Alternative Authentication Methods

> **WARNING**
>
> **Native Object Mount Credential Management is Recommended**
>
> For improved security, it is highly recommended to use the `cuno cred` features as described above for all credential administration and management.
>
> Only use the following methods if additional flexibility is required.

Amazon AWS S3: Using Native Credentials

Object Mount can also use the file supplied by Amazon AWS S3:

```
~/.aws/credentials
```

With the profile optionally specified with the environment variable:

```
AWS_PROFILE
```

Set the AWS region with the environment variable:

```
AWS_REGION
```

Alternatively, directly set the following two environment variables before launching Object Mount:

```
AWS_ACCESS_KEY_ID
AWS_SECRET_ACCESS_KEY
```

Amazon AWS S3: Using IAM Roles for EC2

IAM Roles provide an alternative method to securely distribute AWS credentials across EC2 instances. These credentials can be used for accessing AWS S3. Refer to the AWS article 🌐 IAM roles for Amazon EC2 for more information.

Object Mount will automatically detect IAM roles on an EC2 instance, so no further configuration is necessary.

> **WARNING**
>
> **Do Not Combine Methods**
>
> Combined use of Object Mount managed credentials with IAM Roles in EC2 is not currently supported.

Google Cloud Storage: Using Environment Variable

Object Mount can directly use service account credentials, which are commonly used by applications to access Google Cloud Storage.

To use a service account credential file (in JSON or PKCS12 format), specify a value for the `GOOGLE_APPLICATION_CREDENTIALS` environment variable.

For example: A credential file located in `/home/user/gckey.json` can be loaded to Object Mount by using:

```
export GOOGLE_APPLICATION_CREDENTIALS="/home/user/gckey.json"
```

Azure Storage: Using Environment Variables

To access Azure Storage without using Object Mount credentials management, provide a storage access key associated with a Microsoft Azure Storage account.

To enable this access scheme, set the `AZURE_STORAGE_ACCOUNT` and `AZURE_STORAGE_ACCESS_KEY` environment variables:

```
export AZURE_STORAGE_ACCOUNT="<account-name>"
export AZURE_STORAGE_ACCESS_KEY="<account-key>"
```

> **WARNING**
>
> **Not Recommended**
>
> Microsoft recommends that storage access keys **not** be shared.
>
> To permit access to storage resources *without* sharing access keys, see above for how to use a Shared Access Signature (SAS) instead.
>
> An SAS gives users access to a container for a limited time period with a fixed set of permissions.

Advanced Loading Options

Loading Object Mount

This section describes advanced usage of Object Mount, including the user-mode library, scripting, containerization, and high-performance computing (HPC) integration.

Via CLI

You can use the `cuno` command to launch a new default Linux shell that is "wrapped" by Object Mount running in Direct Interception Mode:

```
user:~$ cuno
```

You should see the `(cuno)` CLI prefix to the left of your user prompt. This indicates that a new shell has been launched, wrapped in Object Mount's `cuno` wrapper:

```
user:~$ cuno
(cuno) user:~$
```

The shell, and any command or application executed from within this shell, will be monitored by Object Mount. Any I/O calls will be intercepted, transformed and redirected to object storage as needed.

Enter `exit` at any time to close the Object Mount CLI console and return to the unwrapped shell prompt:

```
(cuno) user:~$ exit
user:~$
```

Via User-Mode library: LD_PRELOAD

As mentioned in the Object Mount for Linux Introduction, Object Mount provides a User-mode Library, `cuno.so`. This dynamic loader can be pre-loaded into a running process using the `LD_PRELOAD` environment variable.

Set the environment variable `LD_PRELOAD=/opt/cuno/lib/cuno.so` before executing a command and Object Mount will be enabled for that command.

NOTE

Installation Directory

The examples on this page use `/opt/cuno`, the directory of a system-wide installation. Replace it with your installation directory:

- `/opt/cuno` for a system-wide installation
- `$HOME/.local/opt/cuno` for a user-local installation (generally)

Inside a `cuno` shell, the `CUNO_BASEDIR` environment variable holds the installation directory:

```
echo $CUNO_BASEDIR
```

Otherwise, locate your deployment by running:

```
ls /opt/cuno
ls $HOME/.local/opt/cuno
```

One of those directories should display multiple cuno-related files.

See Installation Locations for more information on installation locations.

To start a `bash` shell wrapped by `cuno.so`, run:

```
LD_PRELOAD=/opt/cuno/lib/cuno.so bash
```

The `bash` shell, and any command executed from within this shell instance, will run with Object Mount enabled. Other processes and applications will not be affected.

Note: Loading the `cuno.so` library in this manner will **not** display the `(cuno)` prefix to the prompt.

Example: Browse satellite images on Google Cloud Storage:

```
ls gs://gcp-public-data-landsat/
```

Additional Environment Variables

You may consider setting the following environment variables for convenience:

Environment Variable	Description	Value
`LD_PRELOAD`	Load the Object Mount library	/opt/cuno/lib/cuno.so${LD_PRELOAD+:$LD_PRELOAD}
`PATH`	Path-free access to Object Mount executables	/opt/cuno/bin:${PATH}
`MANPATH`	Easy access to Object Mount manual pages	/opt/cuno/share/man:${MANPATH}

Via Shell Profiles

To *always* load Object Mount when starting a new interactive shell, append the following line to `~/.bashrc` or `~/.zshrc`:

```
export LD_PRELOAD=/opt/cuno/lib/cuno.so
```

`/etc/profile.d` contains application-specific startup scripts, which are executed at user login by their respective login shells.

A sample script is provided in `/opt/cuno/etc/profile.d/cunorc.sh`.

Copy this to `/etc/profile.d/` to enable Object Mount for all users.

Via Docker

There are multiple methods to use Object Mount from within Docker containers.

Automatic Docker Interception

To enable automatic Docker interception, set the environment variable `CUNO_INTERCEPT_DOCKER=1` and load Object Mount. Launching a Docker container via `docker run …` will make Object Mount available inside the container.

> **WARNING**
>
> **Automatic Interception within Docker**
>
> Support for automatic Docker interception is currently **experimental**.
>
> For ways of running Docker with Object Mount interception manually, see Manual Interception below.

For instance, the following code will run a command inside an Ubuntu container with Object Mount enabled with the hosts Object Mount credentials:

```
export CUNO_INTERCEPT_DOCKER=1
cuno run docker run --rm ubuntu:latest ls -l s3://bucket
```

Object Mount options will be forwarded into the container.

To override Object Mount options just for Docker containers, use the `CUNO_DOCKER_OPTIONS` environment variable. If set, its value overrides that of `CUNO_OPTIONS` inside Docker containers.

Manual Docker Interception

Besides automatic interception, the following manual methods can be used to enable Object Mount in Docker containers:

- Install Object Mount as part of the creation of the Docker image.
- Inject Object Mount at launch into an existing Docker image.

To inject Object Mount into a Docker container when it is launched:

1. Set `CUNO_CREDENTIALS` outside the container
   - See Credential File Location for details.
2. If desired, the `CUNO_OPTIONS` environment variable can be set using the Docker `--env` option when using `docker run`
   - See Set environment variables (--env) in the Docker documentation.
3. Use the following options when using `docker run`:

```
docker run                                                \
  --tmpfs /cunodb                                        \
  -v /opt/cuno:/opt/cuno:ro                              \
  -v /opt/cuno/etc/ld.so.preload:/etc/ld.so.preload:ro \
  -v $CUNO_CREDENTIALS:/opt/cuno-config/creds:ro       \
  <image> [container-commands]
```

> **NOTE**
>
> This uses the volume mount option (`-v`) to make Object Mount and other directories available to the container.
>
> This command requires Object Mount already be installed and activated, with a valid license key, on the host system.

The Object Mount credentials directory is only readable by the current user for security reasons.

However, credentials may need to be accessed by processes run by other users (e.g. NGINX worker processes) within the Docker container.

To enable this, bind mount the nested `bindpoint` directory instead of the credentials directory:

```
docker run                                                              \
    --tmpfs /cunodb                                                     \
    -v /opt/cuno:/opt/cuno:ro                                           \
    -v /opt/cuno/etc/ld.so.preload:/etc/ld.so.preload:ro                \
    -v $CUNO_CREDENTIALS/bindpoint:/opt/cuno-config/creds/bindpoint:ro \
    <image> [container-commands]
```

Within Kubernetes Clusters

You can use Object Mount on Kubernetes clusters with the Object Mount Kubernetes CSI Driver.

Refer to the Advanced Guide article: Kubernetes CSI Driver.

With Sylabs Singularity

There are multiple methods for enabling 🌐 Sylabs Singularity images with Object Mount, allowing transparent access to cloud hosted files.

You can:

- Install Object Mount as part of the creation of the Singularity image
- Inject the Object Mount library at launch into an existing Singularity image

The following Singularity definition file will install Object Mount into an image:

```
Bootstrap: docker
From: rockylinux:8

%files
    /home/admin/downloads/cuno-{FULL-VERSION}.x86_64.rpm /opt/src/

%post
    yum update -y
    yum install -y /opt/src/cuno-{FULL-VERSION}.x86_64.rpm
    echo "YOUR LICENSE KEY HERE" | cuno creds activate
    chmod og+r /opt/cuno/etc/license

%environment
    export LD_PRELOAD=/usr/lib/cuno.so
    export CUNO_CREDENTIALS=/opt/cuno-config/creds

%labels
    Name object mount
    URL storj.io
    Email supportdcs@storj.io
```

The above will require that the Object Mount software package is available at the path's starting `/home/admin/downloads` as listed in the `%files` section.

Rather than modifying an existing Singularity image, another option is to pass-through the Object Mount library and binaries making them available inside the image.

Optionally, the `CUNO_OPTIONS` environment variable can be set for a particular mode of operation.

The following commands demonstrate injecting Object Mount into a Singularity image:

```
$ singularity exec   \
    --bind /opt/cuno:/opt/cuno \
    --bind /opt/cuno/etc/ld.so.preload:/etc/ld.so.preload \
    --bind "$CUNO_CREDENTIALS":/opt/cuno-config/creds     \
    ./singularity_image.sif ls s3://commoncrawl

cc-index   crawl-002       hive_analysis  meanpath           projects      wikipedia
contrib    crawl-analysis  index2012      parse-output       robots.txt
crawl-001  crawl-data      mapred-temp    parse-output-test  stats-output
```

> **NOTE**
>
> **Activated License Required**
>
> This method requires Object Mount be installed and activated, with a valid license key, on the system that is used to execute the Singularity images.

Alternatively, rather than using command options, the `--bind` parameters can be specified as environment variables before the `singularity` binary is called, making it easier to modify existing pipelines without changing the command line call, like so:

```
export SINGULARITY_BIND="/opt/cuno:/opt/cuno,$CUNO_CREDENTIALS:/opt/cuno-config/creds"
```

As another alternative, Object Mount can be used via a FUSE mount inside Singularity with the `--fusemount` parameter:

```
$ singularity exec \
    --bind /opt/cuno:/opt/cuno \
    --bind "$CUNO_CREDENTIALS":/opt/cuno-config/creds \
    --fusemount 'container:/opt/cuno/bin/cuno mount --root /cuno /cuno' \
    ./singularity_image.sif /cuno/s3/commoncrawl
```

This will allow software inside the container to access cloud buckets via the `/cuno` path prefix.

Using the Lmod Environment Module System

The Lmod Environment Module System is a tool to help users manage shell environments. It allows groups of related environment variable settings to be made or removed dynamically.

Many HPC environments incorporate `module` as one of the means by which software can be loaded by their users.

A sample script has been provided within your Object Mount installation directory here: `/opt/cuno/share/modulefiles/cuno/{FULL-VERSION}.lua`.

It is built specifically for the **Lmod** implementation in **Lua**, and uses values set to example paths (see the `base` variable in particular) and default options that you might want to append to, modify, or remove to suit your environment and your installation choices.

After adding the parent directory of the file to the `MODULEPATH` with `module use /opt/cuno/share/modulefiles/cuno`, a user should be able to run `module load cuno` to add Object Mount to their environment.

With Spectrum LSF

Depending on your exact requirements and setup, there are multiple ways of using IBM's Spectrum LSF HPC Workload Management suite with Object Mount.

First, assuming that Object Mount is installed on all compute nodes, ensure that the `cuno` executable is available in the `PATH`, and that Object Mount credentials are configured for all nodes.

Jobs can be submitted by prefixing the submitted command with `cuno run` (without quotes).

For example:

```
bsub -Is cuno run ls -l s3://bucket
```

If further configuration of Object Mount is required, an LSF job starter can be created.

The script should be created at the location `$LSF_ENVDIR/cuno-starter.sh` or some other location that is common to all LSF nodes. (**Note:** The queue template that ships with Object Mount assumes this location).

The following script can be used as an outline for the job starter:

```
#!/bin/sh

cuno_basedir=<cuno_install_dir>
# e.g.:
# cuno_basedir=/opt/cuno

# Further configuration as necessary, e.g.:
# export CUNO_CREDENTIALS=/usr/share/cuno-creds
# export CUNO_OPTIONS=+cloudroot=acme

"$cuno_basedir"/bin/cuno run "$@"
```

Make the job starter executable:

```
chmod +x "$LSF_ENVDIR"/cuno-starter.sh
```

To configure a new LSF queue for Object Mount enabled jobs, fill in and append the LSF configuration template for Object Mount to `lsb.queues`.

Use the following command:

```
cluster_name=$(lsid | grep 'cluster name' | awk '{print $NF}')
cuno run sh -c "
    sed 's;@LSF_ENVDIR@;$LSF_ENVDIR;' \"\$CUNO_BASEDIR\"/share/lsf/lsb.queues \
    >>\"$LSF_ENVDIR\"/lsbatch/$cluster_name/configdir/lsb.queues
"
```

Afterwards, LSF needs to be reconfigured:

```
badmin reconfig
```

To check that the queue was set up correctly, run:

```
bqueues -l cuno
```

Regardless of whether an LSF queue was set up, the job starter can be used with the `lsrun` command (but **not** `bsub`).

For example:

```
export LSF_JOB_STARTER=$LSF_ENVDIR/cuno-starter.sh
lsrun ls -l s3://bucket
```

To use the dedicated queue, specify it when invoking `bsub`.

For example:

```
bsub -q cuno -Is ls -l s3://demo
```

Advanced Configuration Options

Setting the environment variable `CUNO_OPTIONS` can be used to further configure Object Mount.

The `-o` option added to the `cuno` command can also be used to specify these options.

Options Available for CUNO_OPTIONS

> **NOTE**
>
> **Space Separation**
>
> Individual options must be separated by spaces.
>
> Don't forget to enclose multiple options in quotes (`"` or `'`), or to escape spaces.

The static Option

To use Object Mount with static binaries, the *parent process* needs to be intercepted in static mode.

Interception of static binaries is enabled by default. To disable it requires setting both `CUNO_OPTIONS="-static"` and starting a new Object Mount wrapped shell.

The uricompat Option

Object Mount can override the handling of URI-style paths (`xx://`) in applications that attempt to provide internal handling of such URIs. Users may provide the `+uricompat` option to add similar patching for further applications.

To specify any applications to be patched, specify them as a colon-separated list:

```
+uricompat=<executable 1>:<executable 2>:< ... >
```

To limit the cases in which this patching occurs, you can filter interceptions to only occur when certain command-line arguments match specified keywords. This may be useful for certain code interpreters/executors (see the example below).

To do this, in place of an executable name, use a slash-separated list of the names, a 1-based index of the argument to be matched (or any using `*`), and a value to match against.

For example:

```
+uricompat=<name of binary>/<index of argument to be matched>/<argument value to match>
```

> **NOTE**
>
> `uricompat`
>
> The `+uricompat` option only works on URIs that have been paired.
>
> If you are working with a public access bucket, you will need to run the `cuno creds pair` command (see Pairing Containers and Credentials) against it for the `+uricompat` to take effect.

Applications With Patched URI Handling:

By default, Object Mount will attempt to override the internal URI handling of popular tools, such as:

- `rsync`

- `ffmpeg`
- `tar`

And some genomics tools:

- `samtools`
- `igv`
- `fastQC`

This means that when a URI-style path corresponding to a paired bucket or container is passed to any of the applications above, Object Mount will prevent the application from handling the path in a special way.

For example, `ffmpeg` has special handling for some 🌐 protocols which they specify in a similar URI format to Object Mount URI cloud paths (e.g. `ftp://example.foo`).

As a result, Object Mount needs to prevent `ffmpeg` from failing (when given a path like `s3://bucket/file`) because `s3`, `az` and `gs` are not protocols it supports.

Example:

To override a Java application's handling of URI-style paths, you can use:

```
export CUNO_OPTIONS="+uricompat=java/2/appname.jar:java/*/org.name.main.Class"
```

This will intercept any invocation of `java` with `appname.jar` as its second argument (e.g.: `java -jar appname.jar s3://bucketname/file.bam`).

It will also intercept invocations of Java where any argument is equal to `org.name.main.Class` (e.g.: `java org.name.main.Class s3://bucketname/file.bam`).

The cloudroot Option

You may define a custom directory path prefix to replace the default `/cuno` path. Once defined, the path specified will be made available at the root of the file system.

For example, if you configure the following:

```
export CUNO_OPTIONS='+cloudroot=my-object-storage'
```

You will then be able to list your configured S3 buckets with:

```
ls /my-object-storage/s3/<bucket>
```

> **WARNING**
>
> **Using `cloudroot`**
>
> We strongly recommend that the cloudroot path *does not exist locally* because it can confuse Object Mount's ability to distinguish whether a path refers to the local file system or to the cloud.
>
> If you specify a cloudroot which conflicts with an existing directory, then it must not contain directories or files named `s3`, `az` or `gs`.

> **NOTE**
>
> `cd to cloud`
>
> In order for the “cd to cloud” functionality to see this option, you will need to spawn a new shell after `CUNO_OPTIONS`.

FlexMount Usage:

Setting a cloudroot during the mounting process can affect the options required to be set when using it as a FlexMount.

Recommended option: If cloudroot was not set when creating the mount (e.g. `cuno mount ~/my-object-storage`), then cloudroot needs to be set as the path to the mount point when launching the Object Mount CLI in order to use that mount as a FlexMount (e.g. `cuno -o cloudrootover -o cloudroot="<full-path-to-mountpoint>"`).

In some circumstances, you may need to set a cloudroot when creating the mount (for example, if a `/cuno` directory already exists on your system). In these cases, we strongly recommend migrating or removing any such physical directory so that you can continue to use the default settings. If this is not possible, there are a number of options available.

Recommended option if cloudroot *must* be changed: If a cloudroot needs to be set when creating the mount, then you should set the `--root` option to the same cloudroot to get expected behavior when using Object Mount FlexMount (e.g. `cuno -o cloudroot=/altcloudroot mount $HOME/my-object-storage --root /altcloudroot`). In this case, continue to use the same options when launching cuno (e.g. `cuno -o cloudrootover -o cloudroot="<full-path-to-mountpoint>"`) to access the mount as a FlexMount.

In very limited circumstances, it is not possible to choose a cloudroot that can also be used as the `--root` of the mount point (for example, if the cloudroot is chosen to be somewhere that is not mountable). In this case, we would advise you to choose a different cloudroot, one that can be mounted (such as a position within your user home directory e.g. `cuno -o cloudroot=$HOME/mycloudroot mount $HOME/my-object-storage --root $HOME/mycloudroot`). If this is not possible, then there are two options:

HIGHLY discouraged: If a cloudroot needs to be set when creating the mount, and the `--root` option is set to a path within the cloudroot (e.g. `cuno -o cloudroot=/altcloudroot mount $HOME/my-object-storage --root /altcloudroot/s3/mybucket`), then, when using the FlexMount later, one must set the 'cloudrootover' path as so: `cuno -o cloudroot=/altcloudroot -o cloudrootover=$HOME/my-object-storage`. **Important:** always check that the mount point exposes the cloud at the same point when used directly as a Object Mount on FUSE (so outside of a Object Mount CLI instance) and when accessed as a FlexMount.

HIGHLY discouraged: If a cloudroot needs to be set when creating the mount, and it is *not possible* for `--root` to be set to a path within the cloudroot (i.e. `cuno -o cloudroot=/altroot mount $HOME/my-object-storage --root /`), then, when using the FlexMount later, one must set the cloudroot as follows: `cuno -o cloudrootover -o cloudroot=$HOME/my-object-storage/my/cloud/root`.

> **WARNING**
>
> **Critical Option Settings to Avoid Loss of Data**
>
> Object Mount behavior will be **broken and dangerous** if you do not set the correct options when launching Object Mount CLI with there types of FlexMounts.
>
> For example, in this case if the default parameters for launching a FlexMount are used, namely `cuno -o cloudrootover -o cloudroot=$HOME/my-object-storage`, then applications that are intercepted directly will see `$HOME/my-object-storage` as the cloudroot (so the immediate children are `az gs s3`) whereas any applications that fall through to the Object Mount on FUSE (static binaries, cloud binaries, SUID binaries, Snap apps, AppImage apps, and Flatpak apps) will see `$HOME/my-object-storage` as `/` - the root of your local filesystem.
>
> This could cause a **loss of data**, accidentally expose data in the wrong place or other misbehavior when running scripts.

Ownership and Permissions

Core File Access

In Core File Access Mode, and for files in object storage uploaded using tools *other* than Object Mount, we assign some dynamic defaults for ownership and permissions.

In these circumstances, the owner of cloud objects is always reported to be the current user, the directory mode is reported as `0777`, and the file mode is reported as `0666`.

The defaults can be overridden by using the `uid`, `gid`, `filemode` and `dirmode` options within the `CUNO_OPTIONS` environment variable.

By doing so, an administrator can set the default UID/GID and access mode permissions that apply to all files and directories of cloud storage accessed by Object Mount. This might be considered in circumstances where an application starts as one user, but mid-process switches context to run as another — services like web-servers often function in this way.

To demonstrate this in action, you could run the following:

```
$ cuno -o 'uid=0 gid=0 filemode=0444 dirmode=0447' \
  mount --root s3://examplebucket/directory/ /mnt/s3-bucket

$ ls -l /mnt/s3-bucket
dr--r--rwx  1 root root 0 Jan  1  1970 directory
-r--r--r--  1 root root 0 Nov 10 11:16 file
```

While the default UID/GID and access mode permissions can be set ahead of time, they cannot be modified later.

POSIX File Access

To **persist and modify** file system metadata, POSIX File Access must be enabled.

This allows the virtual files presented by Object Mount to have their system metadata modified by tools such as `chown`, `chmod`, and `touch`.

> **NOTE**
>
> **POSIX Metadata File**
>
> The system metadata is stored as objects within your object storage system within hidden subdirectories where the data resides. The name of that subdirectory is a period followed by a unicode U+FBF80 character.
>
> From within Object Mount you will not see that directory or objects inside of it, as Object Mount filters them from view. However, if accessing your object storage using other tools, they will be visible.
>
> Non-Object Mount access which renames, moves or copies objects with Object Mount file attributes, will result in those objects **losing their system metadata**. Instead, you should always use Object Mount to manage those files and preserve their attributes.

Enabling POSIX for Direct Interception:

There are two ways to enable POSIX File Access when using Direct Interception Mode via the Object Mount CLI:

1. **Setting a Bucket Tag**

   The `cuno creds setposix` command can be used to toggle POSIX mode `on` for the bucket and takes precedence over the `CUNO_POSIX` environment variable. This command writes a tag to the bucket's metadata on the bucket store.

   The setting set by this command applies to all users using the bucket through Object Mount.

   ```
   cuno creds setposix s3://examplebucket true
   ```

   > **NOTE**
   > `cuno creds setposix`
   >
   > The following permissions are required to use the `cuno creds setposix` command:
   >
   > - S3-compatible: `s3:PutBucketTagging` and `s3:GetBucketTagging`
   > - Google Cloud Storage: `storage.buckets.update` and `storage.buckets.get` IAM permissions
   > - Azure Blob Storage: Only the container owner may set labels

   Valid options are:

   - `true` - enable posix in enforcing mode, preventing unauthorized access
   - `false` - disable posix
   - `permissive` - enable posix in permissive mode, where object attributes like uid, gid, mode, timestamps are stored but user access privilege is not checked
   - `metadata` - enable posix in enforcing mode and store posix attributes as object metadata (only on S3)
   - `permissive-with-metadata` - enable posix in permissive mode and store posix attributes as object metadata
   - `unset` - remove posix configuration from the bucket

> **WARNING**
>
> **Object Storage Compatibility with Tags**
>
> Some S3-compatible stores may not support setting custom metadata flags required for this feature to function.
>
> If this is the case, you can still use the `--posix` flag to enable POSIX enforcement when starting **Object Mount on FUSE**.
>
> You can also still use the `CUNO_POSIX` environment variable to enable it for a **Direct Interception** session. If these methods are not satisfactory or preferable, please get in contact with our 🌐 Storj Support Team for additional assistance.

To unset the bucket-level setting and allow controlling POSIX mode using either the `CUNO_POSIX` environment variable or the `--posix` command line flag, use `cuno creds setposix` with a setting of `unset`:

```
cuno creds setposix s3://examplebucket unset
```

2. Using the CUNO_POSIX Environment Variable

   If POSIX mode is not set at the bucket level, it can be enabled by setting the environment variable `CUNO_POSIX` to `1`. If POSIX is enabled for the bucket via a tag, it **cannot** be overridden with this environment variable.

   For example, to enable POSIX mode for a Direct Interception session via Object Mount CLI:

   ```
   $ export CUNO_POSIX=1
   $ cuno
   ```

   This mode is suitable for when you need deeper POSIX compatibility in order to correctly run applications or workflows. For example, a tool such as `rsync` may be checking `mtime` in order to check for updates to a file thus requiring you to preserve such attributes on the object storage side.

   > **NOTE**
   >
   > **Non-Object Mount Access & Security**
   >
   > This is not “secure” in the sense that Direct Interception mode requires the user to have access to object storage credentials. Consequently, they can work around any access limitations by using the credentials directly with other tools (such as `awscli`).

Enabling POSIX for a Object Mount on FUSE:

If a bucket doesn’t have POSIX mode already enabled, you can use the option `--posix` when mounting a Object Mount on FUSE to enable fine-grained control over ownership and permissions and have them enforced.

If a bucket’s POSIX mode tag is already set but the `--posix` flag is not, the bucket’s setting will apply but the permissions will **not be enforced** by the mount.

If you are mounting for yourself and others, mount the bucket in a location that is accessible to the users who need access to the files. This can be a shared location or a location that is only accessible to the user who needs access.

`--posix` sets the FUSE option `-o allow_other`, so other users, including root, can access the mount as themselves. Do not add `-o allow_root`: FUSE refuses to mount with both `allow_root` and `allow_other`. If you mount as a non-root user, `allow_other` requires `user_allow_other` in `/etc/fuse.conf`.

```
$ cuno mount --posix --root s3://mybucket /mnt/cloud/bucket
```

Users will now be able to see the files in the bucket at `/mnt/cloud/bucket`. You should now treat this as any other POSIX file system, and set permissions as you need to control access.

Usage Examples

Assuming a mount location of `/mnt/s3-bucket`:

- Change permissions on an object/file:

```
$ chmod 0770 /mnt/s3-bucket/file
$ ls -l /mnt/s3-bucket/file
-rwxrwx---  1 root  root 0 Nov 10 11:16 file
```

- Change the owner of an object/file:

```
$ chown alice /mnt/s3-bucket/file
$ ls -l /mnt/s3-bucket/file
-rwxrwx---  1 alice root 0 Nov 10 11:16 file
```

- Alter the modify time:

```
$ echo "Hello" > /mnt/s3-bucket/file
$ ls -l /mnt/s3-bucket/file
-rwxrwx---+ 1 alice root 6 Nov 10 11:32 file
```

- Alter the change time:

```
$ chmod +x /mnt/s3-bucket/file
$ stat /mnt/s3-bucket/file
File: /mnt/s3-bucket/file
Size: 6               Blocks: 1          IO Block: 65536  regular file
Device: 49h/73d Inode: 4           Links: 1
Access: (0770/-rwxrwx---)  Uid: ( 1009/   alice)   Gid: ( 0/    root)
Access: 2021-11-10 11:32:59.185000000 +0000
Modify: 2021-11-10 11:32:59.185000000 +0000
Change: 2021-11-10 11:47:21.846000000 +0000
```

Other Advanced Configuration Options

CUNO_CREDENTIALS Variable

The environment variable `CUNO_CREDENTIALS` allows you to customize the location of the Object Mount credentials store. Full details can be found in the Advanced Guide article: .

Proxy Server Tunneling

Object Mount supports HTTP and HTTPS proxying access through to your cloud object storage.

Provide the environment variable `http_proxy` or `https_proxy` with your gateway and Object Mount will tunnel all transactions through your designated proxy.

Verbose Debug Output

There are rare occasions where Object Mount outputs to the command line, and this will generally indicate a fault.

All messages will be prefixed with `cuno:` and these will always be sent a to the standard error stream (`stderr`).

The options available for `CUNO_LOG` are cumulative and build on the previous level:

```
.. cssclass:: listtable43mm

  .. list-table::
    :header-rows: 0

    * - :code:`error`
      - At this level, only critical failures are output. (Default)
    * - :code:`warning`
      - At this level, errors and warnings are output.
    * - :code:`debug`
      - At this level, information relevant to debugging will be output.
    * - :code:`trace`
      - At this level, a verbose logging of all steps will be output.
```

For example, a directory listing with the trace level set, can produce output similar to the below:

```
$ CUNO_LOG=trace ls s3://example-bucket
cuno: [TRACE][01/02/21 08:39:07.39][Thread: 3817873920] Trying bucket store at: /home/
user/.config/cuno/creds/...
cuno: [TRACE][01/02/21 08:39:07.40][Thread: 3817873920] Loading S3Client
cuno: [DEBUG][01/02/21 08:39:07.40][Thread: 3817873920] Not On EC2
cuno: [TRACE][01/02/21 08:39:07.40][Thread: 3817873920] Attempting to find CaCerts.
cuno: [DEBUG][01/02/21 08:39:07.40][Thread: 3817873920] Using Certs: /etc/ssl/certs/ca-
certificates.crt
cuno: [TRACE][01/02/21 08:39:07.40][Thread: 3817873920] Attempting to load PGMAN
cuno: [DEBUG][01/02/21 08:39:07.40][Thread: 3817873920] Using custom Endpoint: http://
127.0.0.1:8080
cuno: [DEBUG][01/02/21 08:39:07.40][Thread: 3817873920] Using custom pathstyle
```

There is a separate option: `access` that enables detailed logging of the file access performed by Object Mount, which includes read, write, open, close, seek and similar.

`access` can be combined with the other log levels to log both file access and the desired logs.

For example:

```
$ CUNO_LOG=trace,access ls s3://example-bucket
cuno: [DEBUG][07/17/23 13:59:03.287][P/T: 296181:2897806400] Not On EC2
cuno: [DEBUG][07/17/23 13:59:03.287][P/T: 296181:2897806400] Using Certs: /etc/ssl/certs/ca-
certificates.crt
cuno: [ACCESS][07/17/23 13:59:03.355][P/T: 296181:2897806400] s3://example-bucket/file write() 12
@ 0
```

Server-Side Encryption

Object Mount fully supports server-side encryption for both streaming to and from remote locations.

That includes server-side encryption with customer-provided encryption keys (SSE-C) which allows users to set their own encryption keys.

Users are responsible for managing their encryption keys as well as providing them as part of every remote request.

As a result, users have to provide custom keys to Object Mount.

Server-Side Encryption in AWS S3

Object Mount fully supports the default behavior in AWS to always encrypt the user data on the server side, before it is written to disk.

Object Mount also supports customer-supplied encryption using AES-256 keys.

For more information see the AWS article: 🌐 Using server-side encryption with Amazon S3 managed keys (SSE-S3).

In AWS S3, users specify their encryption options by setting the `AWS_S3OPTIONS` environment variable.

The options are similar to those of `awscli`, but the names differ: for example, Object Mount uses `--sse-write-c` and `--sse-read-c` where `awscli` uses `--sse-c` and `--sse-c-copy-source`. Use the names in the table below.

For example:

```
export AWS_S3OPTIONS="--sse-write-c AES256 --sse-write-c-key xxxxxxxxxxxxxxxxx"
```

This enables AES256 server-side encryption with a customer-provided key.

Available options include:

Option	Description
`--sse [string]`	Specifies server-side encryption of the object in S3. Valid values are `AES256` and `aws:kms`. The default keys are used in **encryption/decryption**.
`--sse-read-c [string]`	This parameter should only be specified when **reading** an S3 object that was encrypted server-side with a customer-provided key. It specifies the algorithm to use when **decrypting** the source object. `AES256` is the only valid value.
`--sse-read-c-key [string]`	This parameter should be specified when **reading** an S3 object that was encrypted server-side with a customer-provided key. Specifies the customer-provided encryption key for Amazon S3 to use to **decrypt** the source object. The encryption key provided must be one that was used when the source object was created. `--sse-read-c` must be specified as well. The key provided should not be base64 encoded.
`--sse-write-c [string]`	Specifies server-side encryption for **writing** an S3 object using customer provided keys of the object in S3. `AES256` is the only valid value. If you provide this value, `--sse-write-c-key` must be specified as well.
`--sse-write-c-key [string]`	The customer-provided encryption key to use to server-side **encrypt** the object in S3. If you provide this value, `--sse-write-c` must be specified as well. The key provided should not be base64 encoded.

Option	Description
`--sse-kms-key-id [string]`	The AWS KMS key ID that should be used to server-side **encrypt** the object in S3. Note that you should only provide this parameter if the KMS key ID is different from the default S3 master KMS key.

> **NOTE**
>
> **AWS Key Management System**
>
> AWS KMS keys are required **only** for writing encrypted objects. Reading of encrypted objects does not require the corresponding KMS key to be specified.
>
> Furthermore, only one of `--sse-write-c-key` and `--sse-kms-key-id` options can be enabled at a time.

Server-Side Encryption in Google Cloud Storage

Object Mount fully supports the default behavior in Google Cloud Storage to always encrypt the user data on the server side, before it is written to disk.

In addition to this option, Object Mount also supports customer-supplied encryption using AES-256 keys.

For more information see the GCS article: 🌐 Customer-supplied encryption keys

To enable the SSE-C option the user must set the `GS_ENCRYPTION` environment variable.

For example:

```
export GS_ENCRYPTION="--sse-write-c-key xxxxxxxxxxxxxxxxx"
```

This enables AES256 server-side encryption with the specified customer-provided key.

Available options include:

Parameter	Description
`--sse-read-c-key [string]`	This parameter should be specified when **reading** a remote object on Google Cloud Storage that was encrypted server-side with a customer-provided key. Specifies the customer-provided encryption key for Google Storage to use to decrypt the source object. The encryption key provided must be one that was used when the source object was created.
`--sse-write-c-key [string]`	The customer-provided encryption key to use to server-side encrypt the object in Google Cloud Storage when **writing**. The key provided should not be base64 encoded. Object Mount is also fully compatible with Google's cloud-hosted key management service (KMS). Customer-managed keys used in encryption are provided `GS_ENCRYPTION` environment variable using the `--sse-kms-key-id` option described below.
`--sse-kms-key-id [string]`	The Cloud KMS key that should be used to server-side encrypt the object in Google Cloud Storage. Note that you should only provide this parameter if KMS key ID is different than the default KMS key associated with the remote bucket.Note that Cloud KMS keys are required only for **writing** encrypted objects. Reading of encrypted object does not require the corresponding Cloud KMS key to be specified. Furthermore, only one of `--sse-write-c-key` and `--sse-kms-key-id` options can be enabled at a time.

Server-Side Encryption in Microsoft Azure

Server-side encryption is always enabled at rest in Microsoft Azure containers

See the Microsoft Azure article: 🌐 Azure Storage encryption for data at rest.

It is the default, and cannot be disabled.

Kubernetes CSI Driver

The Object Mount Kubernetes CSI Driver facilitates seamless integration of your S3 cloud storage services within a Kubernetes cluster.

The driver is available through the Helm package manger under:

- `oci://registry-1.docker.io/cunofs/cunofs-csi-chart`.

More information can be found on Docker hub.

Installing Helm & CSI Driver

1. Ensure that Helm is installed.
   - If not, follow the Helm installation guide
2. Deploy the Object Mount CSI Driver.

```
helm install cunofs-csi-chart oci://registry-1.docker.io/cunofs/cunofs-csi-chart \
  --set cunofsLicense.license="<license-text>"                                   \
  --set credsToImport="{<credentials-1>,<credential-2>, ... ,<credentials-N>}"
```

   - `--set cunofsLicense.license`: (required) Object Mount license key (more details)
   - `--set credsToImport`: (optional) S3 cloud credentials (more details)
3. Display the status of the Object Mount CSI Driver resources.

```
kubectl get all -l app.kubernetes.io/name=Object Mount-csi-driver
```

> **NOTE**
>
> **Base64 Encoding of Credentials**
>
> For security reasons, `helm` does not allow access to files via paths.
>
> Therefore, you need to provide the credential file contents in `credsToImport`, and not via paths.
>
> To ensure that the cloud credentials are passed correctly, they should be `base64` encoded.
>
> For example:
>
> ```
> --set credsToImport="{$(cat creds-1.txt | base64), $(cat creds-2.json | base64)}"
> ```

Updating Helm

To upgrade `helm` to the latest version, use:

```
helm upgrade --reuse-values cunofs-csi-chart 'oci://registry-1.docker.io/cunofs/cunofs-csi-chart'
```

You can append the `--version <version>` to upgrade to a specific version.

Uninstalling Helm

To uninstall `helm`, use:

```
helm uninstall cunofs-csi-chart
```

Storage Allocation

The Object Mount CSI Driver supports the following Storage Allocation strategies:

- Static Storage Provisioning
- Dynamic Storage Provisioning

Static Storage Provisioning

To allocate storage **statically**, define one or more `PV` (🌐 Persistent Volumes) providing the bucket details and options:

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: cunofs-pv
spec:
  capacity:
    storage: 16Ei # ignored but required
  accessModes:
    - ReadWriteOncePod # Currently only support "ReadWriteOncePod"
  csi:
    driver: cunofs.csi.com # required
    volumeHandle: cunofs-csi-driver-volume
    volumeAttributes:
      root: "/cuno/s3/bucket/subdirectory/other_subdirectory" # optional
      posix: "true" # optional
      allow_root: "false" # optional
      allow_other: "true" # optional
      auto_restart: "true" # optional
      readonly: "true" # optional
```

Then, define a `PVC` (🌐 Persistent Volume Claim):

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: cunofs-pvc
spec:
  accessModes:
    - ReadWriteOncePod
  storageClassName: "" # ensures that no dynamic provisioning occurs
  resources:
    requests:
      storage: 16Ei # ignored but required
  volumeName: cunofs-pv # PV metadata.name
```

Finally, cluster users can mount the `PVC`:

```
apiVersion: v1
kind: Pod
metadata:
  name: consumer-pod
spec:
  containers:
    - name: cunofs-app
      image: centos
      command: ["/bin/sh"]
      args: ["-c", "echo 'Hello from the container!' > /data/s3/cuno-csi-testing/K8s_$(date -
        u).txt; tail -f /dev/null"]
      volumeMounts:
        - name: persistent-storage
          mountPath: /data
  volumes:
    - name: persistent-storage
      persistentVolumeClaim:
        claimName: cunofs-pvc # PVC metadata.name
```

Dynamic Storage Provisioning

To allocate storage **dynamically**, define a 🌐 Storage Class providing the bucket details and options:

```
apiVersion: storage.K8s.io/v1
kind: StorageClass
metadata:
  name: cunofs-storageclass
provisioner: cunofs.csi.com
  reclaimPolicy: Retain # default is Delete
  parameters:
    cloud-type: s3 # requires either of s3/az/gs
    bucket: cuno-csi-testing # requires bucket that already exists
    bucket-subdir: test_kubernetes # optional
    # Options passed down to the PV:
    posix: "true" # optional
    allow_root: "false" # optional
    allow_other: "true" # optional
    auto_restart: "true" # optional
    readonly: "true" # optional
```

Then, define a `PVC` which has a reference to the `StorageClass`:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: cunofs-pvc
spec:
  accessModes:
    - ReadWriteOncePod
  storageClassName: "cunofs-storageclass" # StorageClass metadata.name
  resources:
    requests:
      storage: 16Ei # ignored but required
```

Cluster users can mount the `PVC` similarly to the static allocation case:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: consumer-pod
spec:
  containers:
    - name: cunofs-app
      image: centos
      command: ["/bin/sh"]
      args: ["-c", "echo 'Hello from the container!' > /data/s3/cuno-csi-testing/K8s_$(date -
        u).txt; tail -f /dev/null"]
      volumeMounts:
        - name: persistent-storage
          mountPath: /data
  volumes:
    - name: persistent-storage
      persistentVolumeClaim:
        claimName: cunofs-pvc # PVC metadata.name
```

Alternatively, cluster users can create a generic inline volume which doesn't require a `PVC`:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: consumer-pod-dyn-inline
spec:
  containers:
    - name: cunofs-app-inline
      image: centos
      command: ["/bin/sh"]
      args: ["-c", "echo 'Hello from the container, inline volume!' >> /data/generic-inline-k8s_$
        (date -u).txt; tail -f /dev/null"]
      volumeMounts:
        - name: inline-cuno-storage
          mountPath: /data
  volumes:
    - name: inline-cuno-storage
      ephemeral:
        volumeClaimTemplate:
          metadata:
            labels:
              type: my-inline-volume
          spec:
            accessModes: [ "ReadWriteOncePod" ]
            storageClassName: cunofs-storageclass # StorageClass metadata.name
            resources:
              requests:
                storage: 16Ei # ignored but required
```

> **NOTE**
>
> **CSI Ephemeral Inline Volumes**
>
> Currently, Object Mount's CSI Driver does not support 🌐 CSI Ephemeral Inline Volumes.

Configuring Helm & CSI Driver

This section offers additional details about the configuration options for the Object Mount CSI driver.

Helm Chart

Helm chart can be also installed, configured and deployed manually.

Download the chart manually:

```
helm pull --untar oci://registry-1.docker.io/cunofs/cunofs-csi-chart
```

Set the `cunofsLicense.license` variable and import the cloud credentials:

```
# values.yaml file
cunofsLicense:
  license: "<your license key>"
credsToImport:
  - “<credential-1>"
  - "<credential-2>"
  - "<..>"
  - "<credential-N>"
```

Finally, install the local chart by pointing to its directory:

```
helm install cunofs-csi-chart <path-to-chart>
```

Available Options:

Yaml Value	Description	Default Value
`driverName`	Optionally change the name of the deployed driver. Only useful if you want to deploy several instances of the driver.	`cunofs.csi.com`
`cunofsCSIimage.pullPolicy`	Specifies how the docker image is deployed onto the Node and Controller. Only useful to change if self-hosting the docker image.	`Always`
`cunofsCSIimage.name`	Specifies the Object Mount CSI docker image. Only useful to change if self-hosting the docker image under a different name (Note: do not include the version here).	`cunofs/cunofs_csi`
`cunofsCSIimage.version`	Specifies the docker image’s version. No need to change it unless you have a good reason to.	`<equal to chart version>`
`cunofsLicense.license`	The license used for activating Object Mount on the Driver. It needs to be a valid Enterprise license.	`<empty>`
`credsToImport`	Yaml array that you can populate with your s3/az/gs credential files.	`<empty>`
`rbac.useRBAC`	Enables out of the box support for RBAC clusters (deploys the required ClusterRole/ClusterRoleBinding).	`true`
`eks.iam_arn`	On Amazon EKS, associates IAM role to `ServiceAccount`.	`<empty>`

`PersistentVolume` Options:

WARNING

Boolean Parameters

Due to K8s parameter passing design decisions, the boolean parameters require strings and not yaml booleans.

For this reason, use `"true"` and `"false"` instead of `true` and `false`.

Yaml Value	Description
`metadata.name`	Can be any legal, unique name
`spec.capacity.storage`	This value is ignored, but is required to be set by the CSI specification
`spec.csi.driver`	Set this to the name of the CSI driver you deployed which is `cunofs.csi.com` by default
`spec.csi.volumeHandle`	Name of the volume, needs to be unique
`spec.accessModes`	We support `ReadWriteOncePod`, `ReadWriteOnce`, `ReadOnlyMany`, `ReadWriteMany`. `ReadWriteMany` requires Object Mount Fusion for write consistency, but works in any mode
`spec.csi.volumeAttributes.root`	This is the cloud URI that will be mounted to the target mountpath. If not specified, you can access s3, az and gz through the target + `/az` or `/gs` or `/s3` directories
`spec.csi.volumeAttributes.posix`	Set it to `"true"` to enforce strict posix mode for Object Mount
`spec.csi.volumeAttributes.allow_root`	Set it to `"true"` to allow *only* the root user to access the mount. Overrides `allow_other`
`spec.csi.volumeAttributes.allow_other`	Set it to `"true"` to allow all users to use the mount (recommended)
`spec.csi.volumeAttributes.auto_restart`	Set it to `"true"` to automatically restart the Object Mount on FUSE if an error occurs
`spec.csi.volumeAttributes.readonly`	Set it to `"true"` to mount the volume as read only
`spec.csi.volumeAttributes.CUNO_OPTIONS`	Sets the `CUNO_OPTIONS` of the `cuno mount`
`spec.csi.volumeAttributes.CUNO_LOG`	Sets the `CUNO_LOG` of the `cuno mount`
`spec.csi.volumeAttributes.fusion_pvc`	Enables Object Mount Fusion on the `PV` with the input as the backing `PVC`

`StorageClass` Options:

Yaml Value	Description
`metadata.name`	Can be any name as long as it's unique
`provisioner`	The name of the driver, by default: `cunofs.csi.com`
`reclaimPolicy`	`Retain` will not delete the generated `PVs` and their storage when the `PVCs` go out of scope, `Delete` will
`parameters.cloud-type`	Can be `s3`, `az` or `gs`
`parameters.bucket`	The bucket used to create volumes

Yaml Value	Description
`parameters.bucket-subdir`	Optional. The subdirectory of the bucket where the `PVCs` will get generated. Can be nested subdirectories like "dir/other_dir/yet_another_dir"
`parameters.{posix, allow_root, allow_other, auto_restart, readonly, CUNO_OPTIONS, CUNO_LOG}`	These options will be passed down to the generated `PV` and behave the same way as described in the `PV` options
`parameters.fusionStorageClass`	Tells Object Mount Fusion to use the given `StorageClass` to allocate backing `PVs`

Object Mount Fusion Support

The Object Mount CSI Driver, as of version `v1.0.2`, supports all Object Mount Deployment Modes

However, using a single `PV` on multiple pods at once with the `ReadWriteMany` access mode (on the same node or not) does not guarantee write consistency to the same file.

We offer a way around this limitation by using Object Mount Fusion.

Object Mount Fusion enables users to get the best out of object storage and traditional shared filesystems at the same time. In the case of the Object Mount CSI Driver, it gives you the ability of writing to a `PV` with `ReadWriteMany` without potential issues with multiple writers.

Object Mount Fusion uses object storage with a supporting backing filesystem and will intelligently use the filesystem that has the best performance for large or small files, while ensuring that reads and writes remain ordered.

Static Allocation

Using Object Mount Fusion under static storage provisioning, you need to deploy two `PVs`: one for Object Mount, and one for the backing mount.

Choose a backing mount that offers write consistency (Amazon EFS, Amazon EBS, a local NFS server, etc.), and deploy it with a `PV` and `PVC`, as if it was used by a `Pod`.

Then, refer to the `PVC`'s name in the `spec.csi.volumeAttributes.fusion_pvc` parameter of the Object Mount `PV`.

The Object Mount CSI Driver will mount the `PV` to itself and bind the two filesystems.

```yaml
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: "backing-pv"
# <...>
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: backing-pvc
spec:
  volumeName: "backing-pv"
  # <...>
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: cunofs-pv
spec:
  capacity:
    storage: 16Ei # ignored, required
  accessModes:
    - ReadWriteMany
  csi:
    driver: cunofs.csi.com
    volumeHandle: cunofs-csi-driver-volume
    volumeAttributes:
      # <...>
      fusion_pvc: "backing-pvc" # gets the name of the pvc to try and mount to it
```

WARNING

Matching Access Modes

Please ensure that the `PV` / `PVC` pair you create has the same access mode as the Object Mount `PV` / `PVC` pair and compatible parameters (readonly, etc.).

If you have any issues deploying the Object Mount Fusion `PV` / `PVC` pair, first ensure that the backing pair is correctly set up.

Dynamic Allocation

The Object Mount CSI Driver supports dynamic provisioning of Object Mount Fusion `PV` pairs.

Simply deploy a backing `StorageClass` and refer to it in the Object Mount `StorageClass` `parameters.fusionStorageClass` parameter.

The Object Mount CSI Driver will use it to generate and delete backing `PVs` alongside Object Mount `PVs` and bind them as needed.

```
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: "backing-sc"
<...>
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: cunofs-storageclass
provisioner: cunofs.csi.com
parameters:
  cloud-type: s3
  bucket: cuno-csi-testing
  bucket-subdir: test_kubernetes
  fusionStorageClass: "backing-sc" # Refer to the backing StorageClass
```

WARNING

Please ensure that the deployed `StorageClass` can create `PVs` that bind to `PVCs` with `ReadWriteMany` of size `8Gi`. In a future release, you will be able to parameterize it.

RBAC Support

By default, the Object Mount CSI Driver deploys a `ServiceAccount`, `ClusterRole` and `ClusterRoleBinding` to support Role-Based Access Control (RBAC) out of the box.

They are respectively deployed under the following names:

- `<release name>-serviceaccount`
- `<release name>-clusterrole`
- `<release name>-clusterrolebinding`.

This also means that the Object Mount CSI Driver supports Amazon EKS clusters out-of-the-box.

You can choose *not* to deploy the `ClusterRole` and `ClusterRoleBinding` by setting the `rbac.useRBAC` property to `false` in the `values.yaml` file:

```
helm install cunofs-csi-chart oci://registry-1.docker.io/cunofs/cunofs-csi-chart \
  --set cunofsLicense.license="<license-text>"                                   \
  --set credsToImport="{<credentials-1>,<credential-2>, ... ,<credentials-N>}"   \
  --set rbac.useRBAC=false
```

Technical Details

The Object Mount CSI Driver abides by the 🌐 Kubernetes Container Storage Interface standard.

It implements the `Node`, `Controller` and `Identity` plugins and uses sidecar containers for simplifying its deployment and maintenance.

The CSI Driver is shipped into one binary, which can act as the `Node` or the `Controller` depending on the context (how it is deployed and which sidecar containers are connected).

The Helm chart deploys Docker containers that have this binary preinstalled.

The `Node` plugin refers to the ability to mount and organize existing `PersistentVolumes` on a Kubernetes node.

The `Controller` plugin implements the ability to create `PersistentVolumes` dynamically through a `StorageClass`.

The `Node` and the `Controller` need to handle logic at different levels:

Node:

- The `Node` plugin needs to be deployed on every K8s Node, since it handles mounting logic that's specific to each machine on which the application containers run. Therefore, it is deployed via a K8s 🌐 DaemonSet. Additionally, these sidecar containers are shipped with the `Node`:

  🌐 Liveness Probe

  This sidecar container ensures that the driver remains responsive, and replaces the driver container on crash.

  🌐 Node Driver Registrar

  This sidecar container registers the driver to the kubelet to simplify its discovery and communication.

Controller:

- The `Controller` plugin needs to be unique across a Kubernetes cluster, since it handles the lifecycle of `PersistentVolumes`, which are K8s global objects. It is therefore managed through a K8s 🌐 StatefulSet:

  🌐 Liveness Probe

  This sidecar container, like with the Node plugin, ensures that the driver remains responsive, and replaces the driver container on crash.

  🌐 External Provisioner

  This sidecar container helps the driver interacting with the K8s API by listening to volume provisioning-related calls.

During the deployment, the Object Mount CSI Driver deploys the Object Mount **license** and cloud **credentials** as 🌐 Secrets.

The license `Secret` is imported by the `Node` and the `Controller` through an environment variable.

The credentials `Secret` is mounted to the `Node` and the `Controller` through a 🌐 Projected Volume and sequentially imported by Object Mount.

Limitations

Not every existing `K8s` optional feature is currently implemented in this driver.

Please contact our Support Team for specific feature inquiries.

- Due to the internals of K8s, the Object Mount CSI Driver makes use of `Object Mount on FUSE` as a backend instead of regular `Object Mount`. This means that performance will be high, but not always as high as a regular `Object Mount` installation.
- Not every `Object Mount` option is currently available for use in the driver. Please refer to the configuration section for the available options.
- The `ReadWriteMany` access mode doesn't guarantee write consistency without Object Mount Fusion.
- The `Object Mount` CSI Driver currently doesn't support CSI Ephemeral Volumes, raw block volumes, volume snapshotting, volume expansion, volume cloning and volume topology options.

Machine Learning Accelerator

The Object Mount **Machine Learning Accelerator** (Object Mount MLA) is a software library that accelerates existing machine learning models by leveraging Object Mount's enhanced data retrieval and storage technology.

> **NOTE**
>
> **Enterprise Feature**
>
> The Object Mount MLA library is only available to Object Mount for Linux users with an **Enterprise License Key**.

Requirements

Python 3.7 or higher is required to use the Object Mount Machine Learning Accelerator.

If you require previous versions of Python 3, reach out to our Support Team.

Supported Functions

The Object Mount MLA has been tested with the following data loading functions:

- `open`
- `numpy.loadfile`
- `numpy.load`
- `numpy.loadtxt`
- `PIL.Image.open`
- `torchaudio.load`

It may also work with additional data loading functions.

Contact our Support Team for assistance with using additional functions.

Installation and Usage

1. **Installation:**

   To use Object Mount MLA you myst install the dependencies in your environment.

   Use `pip` to do so:

   ```
   pip install -r "<path to Object Mount MLA>/requirements.txt"
   ```

2. **Set Environment Variables:**

   Then set the following environment variables:

   - `CUNO_SPEEDUP_PYTHON=1`
   - `PYTHONPATH="<path to the Object Mount MLA>${PYTHONPATH+:$PYTHONPATH}"`

   The `CUNO_SPEEDUP_PYTHON` variable signals Object Mount MLA to enable specific Machine Learning optimizations.

   The `PYTHONPATH` alteration causes Python to load the Object Mount MLA's `sitecustomize.py` file first.

When finished, you can begin using Object Mount. All standard Object Mount deployment modes are support, including Direct Interception and Object Mount on FUSE.

Use the examples below for details on using Object Mount within your specific deployment mode.

Object Mount MLA with Direct Interception Mode

Object Mount's Direct Interception Mode is easily activate by running the `cuno` command to invoke a new Object Mount-wrapped shell.

```
# terminal
cuno
(cuno) $
```

If using the Object Mount CLI (`cuno`) is *not* an option, you can use `cuno run` to enable Object Mount for a *single command*.

You can also set the `LD_PRELOAD` environment variable directly: e.g.: `LD_PRELOAD=<path to your Object Mount installation>/lib/cuno.so` .

See the "User-Mode library: LD_PRELOAD" section in the Advanced Guide article: Advanced Loading Options for additional details on using `LD_PRELOAD` .

Example Usage:

Activating a shell with Object Mount as an example:

```
$ export CUNO_SPEEDUP_PYTHON=1
$ export PYTHONPATH="<path to the Object Mount MLA>${PYTHONPATH+:$PYTHONPATH}"
$ cuno
(cuno) $ python3 demo/verify.py
```

Object Mount MLA with Object Mount on FUSE

To use Object Mount MLA with Object Mount on FUSE you will need to configure your Python scripts to look for data *inside* the mount.

Example Usage:

Mount an S3 bucket named `dataset_bucket` at the location `$HOME/cloudmount` :

```
export CUNO_SPEEDUP_PYTHON=1
export PYTHONPATH="<path to the Object Mount MLA>/dataset_preload2${PYTHONPATH+:$PYTHONPATH}"
cuno mount --root s3://dataset_bucket "$HOME/cloudmount"
python3 dataset_preload2/demo/verify.py --location "$HOME/cloudmount/training_set_1"
```

Technical Limitations

> **NOTE**
>
> **Local File Access Unaffected**
>
> Object Mount imposes no restrictions on local file accesses.
>
> The limitations listed in this section only apply to data stored on object storage.

> **NOTE**
>
> **Technical vs. License-based Limitations**
>
> This document contains technical limitations applying to all Object Mount for Linux users.
>
> There are additional limitations dictated by your license tier - set up a discovery call for more information on licensing and pricing.

Direct Interception

Direct Interception Mode (using either Object Mount CLI or `LD_PRELOAD`) does not currently support SUID binaries, or certain packaged applications like 🌐 Snap, 🌐 AppImage, or 🌐 Flatpak.

Future updates are planned to improve this.

If you need to use such applications you should deploy either Object Mount on FUSE or Object Mount FlexMount.

Maximum Object Size

Depending on your Object Storage provider, Object Mount has a limitation on the maximum file size it can store on a remote location.

The following table indicates the maximum file sizes per provider:

Cloud Provider	Maximum File Size
AWS S3	5 TB
Google Cloud Storage	5 TB
Azure Storage	4.77 TB

Ownership and Permissions

If left at the default, file ownership, permissions and timestamps are set as follows when using POSIX Core File Access Mode:

- The owner of the remote objects will be reported as the **current user**, and remote file permissions will be set to `777`.
- The creation time of directories is displayed as the Unix Epoch (00:00:00 UTC on 1 January 1970).

These can be overridden using `CUNO_OPTIONS`. See the Ownership and Permissions section in the Advance Guide article: Advanced Configuration Options for details.

Directories in Azure

Creating a directory in Azure Storage (using `mkdir`) will result in a remote blob called `\<no name>` to be displayed inside the created directory when the user is using the GUI/File Explorer that the Azure Portal provides.

However, `ls` and all CLI commands will behave as expected, showing the correct directory name.

Tab Auto-Completion

Tab-based auto-completion and wildcard characters are fully supported within Object Mount-wrapped shells.

Wrapped-shell can be launched using either the `cuno` command or using `LD_PRELOAD` (e.g. `LD_PRELOAD=/usr/lib/cuno.so bash`).

In the latter, paths containing colons (such as `s3://bucket`) on cloud paths will only succeed if `:` is removed from the separator list variable `COMP_WORDBREAKS`.

For example:

```
"$LD_PRELOAD" =~ cuno ]] && export COMP_WORDBREAKS=${COMP_WORDBREAKS/:/}
```

Uninstalling Object Mount

To remove Object Mount for Linux from a workstation, follow one of these two uninstall procedures depending on how Object Mount was originally installed.

For **Scripted Installations** (`glibc` and `musl`):

1. Determine the installation root folder
2. Remove files for either one user or all users
3. Remove any shell customizations

For **Package Manager Installations** (Debian, Red Hat, Alpine):

1. Run the appropriate package manager uninstallation command (see below).

Uninstall for: Scripted Installations

If Object Mount was installed using a Scripted Installer, follow the steps below to uninstall Object Mount.

Step 1: Determine the Installation Root

The scripted installer installs Object Mount into:

- `/opt/cuno` for a system-wide installation
- `$HOME/.local/opt/cuno` for a user-local installation

Locate your installation by running:

```
ls /opt/cuno
ls $HOME/.local/opt/cuno
```

One of those directories should display multiple cuno-related files.

See: Installation Locations for more information on installation locations.

Step 2: Remove the Installation Directory

Delete the installation directory and all files and subfolders within it. This also removes the license (`etc/license`).

In addition, remove the symlinks the installer created:

For Single User/Local Uninstallation:

Remove the following additional files:

```
~/.local/lib/x86_64-linux-gnu/cuno.so
~/.local/bin/cuno
```

Also remove the links in `~/.local/share/bash-completion/completions/` that point into `~/.local/opt/cuno`.

For System-wide/Global Uninstallation:

Remove the following additional files:

```
/usr/lib/cuno.so
/usr/bin/cuno
```

Also remove the links in the system bash-completion directory (usually `/usr/share/bash-completion/completions/`) and in `/usr/share/zsh/vendor-completions/` that point into `/opt/cuno`.

Step 3: Shell Customization Removal

Remove any customizations made to shell initialization scripts such as

- `~/.bashrc`
- `~/.profile`
- `.zshrc`
- `/etc/profile.d`
- etc.

Uninstall for: Package Managers

If Object Mount was installed using a Package Manager, use the commands in the following sections to uninstall Object Mount for specific Linux distributions.

The package manager does not remove the license file (`/opt/cuno/etc/license`), which is created after installation. To remove it, delete `/opt/cuno` afterwards with `sudo rm -rf /opt/cuno`.

Debian derivatives (e.g. Ubuntu)

```
sudo apt-get remove cuno\*
```

RedHat derivatives (e.g. CentOS)

```
sudo yum remove cuno\*
```

Alpine Linux derivatives

```
sudo apk del cuno
```

Object Mount in Media Workflows

Object Mount is designed to make working with cloud storage as seamless as possible — especially in media production environments where performance, compatibility, and simplicity matter.

Watch on YouTube

The articles in this **Media Workflow Guides** section will provide an overview of how Object Mount fits into post-production workflows, as well as configuration tips for tools like **Adobe Premiere Pro**, **Avid Media Composer** and **DaVinci Resolve**.

Adobe Premiere Pro

Premiere Pro users benefit from Object Mount:

- Projects and source footage can be stored remotely and edited directly.
- Media can be linked and relinked to Object Mount volumes just like local storage.
- Assign proxies and cache files to local disks for the best performance.
- Object Mount volumes appear like regular drives in Finder or File Explorer, making them easy to integrate with Premiere's media browser and import panels.

See Using Object Mount with Premiere Pro for complete configuration tips.

Avid Media Composer

Object Mount integrates cleanly into Avid Media Composer workflows:

- Preview and scrub footage before importing.
- Store Scratch Disks and Proxies on local, fast volumes.
- Adjust Proxy file formats to improve performance.

See Using Object Mount with Avid Media Composer for full setup and setting guidelines.

DaVinci Resolve

Object Mount interacts seamlessly with DaVinci Resolve:

- Media can be accessed directly from your Object Mount volume.
- Stills, cache, and preview files are fastest when stored on local SSD/NVMe volumes.
- Direct I/O should be disabled for the Object Mount path.
- Optional settings like proxy mode and smart caching can improve playback on cloud-mounted media.

See Using Object Mount with DaVinci Resolve for more details.

When to Use Object Mount

Object Mount is ideal for:

- Remote editing workflows
- Team environments where storage is centralized
- Archival footage access directly from the cloud

- High-speed read access (e.g.: proxy editing or frame review)

Object Mount is particularly well-suited for **hybrid workflows** where some assets live on fast local disks, while other content comes in from the cloud — with no need for manual sync tools or complex workflows.

Final Notes

Object Mount is not a replacement for all local workflows — cache files, renders, and scratch disks are still best kept on local volumes.

But for accessing and working with cloud-hosted media, Object Mount bridges the gap between cloud storage and simplified usability beautifully.

Using Object Mount with Adobe Premiere Pro

Object Mount makes it easy to browse, import, and edit footage directly from S3-compatible storage in Adobe Premiere Pro.

Whether you're working with local proxies, editing remotely, or centralizing project media, Object Mount integrates smoothly into your editing pipeline.

Recommended Setup

Once you've mounted your object storage with Object Mount, your volume will appear like a standard drive in **File Explorer (Windows)** or **Finder (macOS)**.

From there, you can work as usual in Premiere Pro with just a few things to keep in mind.

Importing Footage

- Select **File > Import** (or use the **Media Browser**).
- Navigate to your **Object Mount** drive and select your files.
- Footage can be previewed and scrubbed before importing.

> **NOTE**
>
> **Large File Transfer Performance**
>
> If you notice slower performance with large files or over remote networks, consider using proxy workflows (see below).

Recommended Proxy and Cache Settings

While Object Mount enables direct-from-cloud editing, Premiere Pro still benefits from local cache for smooth playback and faster timeline scrubbing.

Suggested Settings:

1. **File > Project Settings > Scratch Disks**:
   - Set **Media Cache** and **Previews** to a fast, local SSD.
2. **Preferences > Media Cache**:
   - Enable **Auto delete cache files older than X days** to keep things tidy.
3. **Create Proxies** via **Ingest Settings** or manually using the **Project panel**:
   - Store proxies on a local drive.
   - Set **Proxy File Format** to "ProRes Proxy" or "H.264 Low Res" for faster performance.
4. Enable **Toggle Proxies**:
   - Navigate to the Program Monitor.
   - Set **Toggle Proxies** to switch views on the fly.

Relinking Media

If a project was opened on a different machine or the mount point has changed:

- Go to **File > Link Media**.
- Browse to your Object Mount volume.

- Premiere will relink to all matching file paths automatically.

> **NOTE**
>
> **Ease of File Management**
>
> Object Mount maintains consistent folder structures across platforms, which makes relinking straightforward.

Performance Tips

- Stick to **List View** in Media Browser when browsing large folders.
- Use proxies for multicam or high-resolution editing (e.g. 4K, RAW).
- Avoid saving the Premiere project file directly to the Object Mount drive.
  - Store locally, and then backup to the cloud separately, if needed.

Storage Visibility

Your Object Mount volume will show up:

- On **macOS**: Under **Locations** in Finder.
- On **Windows**: As a mapped drive under **This PC**.

Once mounted, there's no need to sync or copy files — you're working live from the source in the cloud.

Reminder on Write Operations

Premiere Pro reads from cloud storage efficiently, but for best results:

- Keep render outputs, scratch files, and auto-saves on a local drive.
- Use Object Mount for **read access to source footage**.
- Export final renders locally, then upload back to your storage if needed.

Using Object Mount with Avid Media Composer

This guide walks you through how to use **Object Mount** with **Avid Media Composer**, including key setup considerations and workflow tips to ensure smooth media management when working with cloud-mounted storage.

Recommended Setup

Avid Media Composer expects all media to reside on high-performance, locally-mounted storage paths.

Object Mount allows you to treat object storage as a native volume, but there are a few important things to keep in mind:

1. **Mount to a simple, short file path**
   - Avid prefers media directories without long paths or special characters.
   - Example: `D:\Avid_Mount` or `/Volumes/AvidMedia`.
2. **Ensure visibility using the `alldrive` Console command**
   - From within Avid Media Composer, open the Console from the Tools menu (CTRL+6 / CMD+6).
   - Type `AllDrives` and press Enter. This makes all mounted volumes visible to Avid.
3. **Save Avid project and bin files locally**
   - Save your `.avb`, `.avp`, and `.avf` files to a local volume.
   - Use Object Mount for read access to media assets (e.g., linked clips, raw footage).

Media and Performance Tips

To ensure a smooth editing experience:

- Enable the **Data cache** in the Object Mount **Settings** tab.
- Store **data and metadata cache** on a fast, local volume.
- If you're working with large, high-resolution files (4K, RAW), enable the following:
  - Increase the **S3 connections limit** in Advanced settings (e.g., 150-200).
  - Use Avid's **Background Bin Saving** to reduce interruptions.
- **Avoid background virus scanning** or indexing on the mounted path (especially on Windows)

Proxy & Optimized Workflows

Avid's DNxHR LB and DNxHD LB proxy workflows are recommended when working with high resolution media on cloud storage.

You may want to consider:

- Using the **Create Proxies** command to generate proxy media and toggle between proxy and high-resolution clips via the **Play button drop menu**.
- Or, transcode media to proxy formats such as **DNxHD LB** or **DNxHR LB** and later use **Relink** to conform back to high resolution for finishing.

In either case, **generating proxies to local storage** is recommended for optimal playback performance, especially when working over slower or high-latency connections.

Monitoring Performance

Transfers via Object Mount may not show progress in Avid's interface, so we suggest:

- On **macOS**: Open **Activity Monitor > Network**.
- On **Windows**: Use **Task Manager > Performance > Network**.
- For detailed diagnostics, set the **Object Mount log level** to `debug` or `trace` in the **Settings** tab.

> **NOTE**
>
> **Use Patience with Slow Transfers**
>
> If Object Mount appears slow when linking large files, it can be due to first-time access from cloud storage.
>
> Performance improves after initial reads, thanks to caching and file prefetch.

Known Considerations

- Avid doesn't natively understand cloud latency so playback may stutter if accessing un-transcoded high-resolution files over slower connections.
- Ensure the mount path is stable and doesn't unmount or disconnect mid-session.
- POSIX mode can help with permission-sensitive environments, but is not required for typical read-only workflows.

Summary

Avid Media Composer works effectively with Object Mount when media is properly organized and cache settings are optimized.

Local project files, proxy workflows, and sensible mount paths all contribute to a faster, more reliable experience.

For additional help tuning your Avid workflow with Object Mount, contact our support team — we're happy to assist.

Using Object Mount with DaVinci Resolve

This guide walks you through the best setup for using Object Mount with **DaVinci Resolve**, along with a few optional tweaks and performance tips that may help smooth things out depending on your system and workflow.

Recommended Settings (Per Project)

After opening or importing your Resolve project, please check the following:

1. From the top menu bar, go to: **Preferences > System > Media Storage**
2. Add your **Object Mount** volume as a **Media Storage Location**:
    - The Object Mount volume **should not** be first in the list.
    - Resolve uses the *top location* for gallery stills and cache files, so keep this separate.
3. Disable **Direct I/O** for the Object Mount path:
    - This helps ensure better compatibility and smoother read/write behavior.

Optional Performance Tweaks

These settings aren't required, but can improve responsiveness or playback performance — especially on larger projects or when working over cloud storage.

Playback and Caching:

- **Playback > Render Cache** → Set to **Smart**
- **Playback > Timeline Proxy Resolution** → Set to **Half** or **Quarter**
- **Playback > Proxy Handling** → Set to **Prefer Proxies**
- **Playback > Use Optimized Media if Available** → Enable

Preview Panels and Navigation:

- In the **Edit** tab:
    - Use the **Timeline View Options** (top-left of the timeline)
        - Set **Thumbnail View** to **Thumbnail** or **None**
        - Disable **Display Audio Waveform**
- In the **Media** tab:
    - Switch **Timelines panel** to **List View** (top right)
    - Open the overflow menu:
        - Disable **Film Strip**
        - Disable **Audio Waveforms**
- In **Media Storage**, switching from **Thumbnail View** to **List View** can also help when browsing through folders on Object Mount volumes.

Monitoring Performance

Because of how Object Mount optimizes transfers, the built-in file copy progress indicators (in Finder or File Explorer) may not accurately reflect what's happening behind the scenes.

For a better view of performance:

- On **macOS**: Open **Activity Monitor > Network tab**
- On **Windows**: Go to **Task Manager > Performance > Network**

If Object Mount is saturating the network connection — that's a good sign! It's doing what it's designed to do.

Transfers may still appear idle in the UI, but they are active "under the hood".

> **NOTE**
>
> **Use Patience with Slow Transfers**
>
> Don't cancel a transfer just because the Finder or Explorer bar seems stuck — always check your system's network usage first.

Appendix

Core Concepts

Characteristics of Object Storage

Object storage excels at high-throughput use-cases and is often considered best for “write-once, read-many” models.

Direct file editing with object storage is traditionally seen as too slow and often requires gateways, proprietary translations, and each file often needs to be completely rewritten — even for the slightest modification.

Additionally, object storage technology differs significantly from the conventional block-and-file I/O calls used by local drives and network-attached solutions — especially for applications that expect strict POSIX compliance.

Storj’s Object Mount solves these shortcomings, redirecting and translating file access calls, simplifying cloud storage implementations, and delivering the highest possible performance from your object storage. POSIX compliance and unique optimizations ensure compatibility and improve performance enabling fast editing and file access directly from the cloud.

Direct Use of Object Storage

Object Mount enables the use of object storage as “hot storage” or as “first-class” storage.

Prior to Object Mount, content workflows required you to “pre-stage” your files by copying content from the cloud down to local, faster drives. You would then run your tasks locally. When finished, you re-uploaded the resulting content back up to object storage to share with your team.

Object Mount’s elegant simplicity and improved performance means you can now run your tasks *directly* on your files in object storage — just mount and go.

Object Mount overcomes the traditional inefficiencies of direct access to object storage by using prediction engines, partial writes, client-side write-caching and other optimizations to achieve much higher performance. Use your cloud storage like it was local storage.

Read/Write Interception

Once installed, Object Mount transparently inserts itself between your applications and your host operating system. Object Mount dynamically detects and intercepts relevant file and system calls made by your application and redirects those requests to your object storage provider. This is called **Direct Interception**.

Direct Interception also facilitates the translation of your application’s standard file system calls (reads and writes) into the appropriate object storage API calls.

This enables *any* application to instantly access cloud objects as if they were local files — no code changes or workflow alterations required. Object Mount intercepts file commands from *any* application, binary or EXE, including those you’ve written yourself. It even works inside unprivileged containerized environments like Docker and virtual environments such as Hyper-V and Virtualbox.

What Object Mount Does

- Provides applications high-speed access to object storage content as if it were a local drive

- Allows standard file system operations to list, copy, move, add, remove, and modify files directly on object storage
- Supports rich POSIX metadata including: users, groups, permissions, symbolic & hard links
- Preserves NFS-equivalent POSIX consistency guarantees

What Object Mount Does NOT Do

- Object Mount does *not* interfere with how applications interact with local drives and any other non-S3 object data
- Object Mount does *not* alter or change the representation of your files on object storage — file data is not modified. This means your content can be accessed directly from your object storage using standard S3 tools and dashboards, bypassing Object Mount at any time.

  Note: Content accessed *directly* on your object storage (rather than through Object Mount) bypasses Object Mount's consistency guarantees.

POSIX Explained

What is POSIX?

The Portable Operating System Interface (🌐 POSIX) is a set of IEEE specifications intended to encourage compatibility between disparate Linux/UNIX flavors through standardized definitions of APIs, shells, interfaces, commands, etc.

Writing an application (or binary) which meets POSIX standards means that the program is "portable," and is more likely to run correctly no matter what UNIX-based OS it is installed on.

Some applications may (or may not) require *strict* POSIX-compliance to work properly.

Object Mount's POSIX Mode

To support those programs and applications that require POSIX-compliant interfaces, access-control, and metadata, Object Mount includes **POSIX Mode**.

When activated, POSIX Mode enables the necessary additional features to support applications that rely on the rigid, traditional UNIX-style file permissions and metadata.

Object storage, by default, lacks concepts like file ownership, permissions, and modification timestamps that are normal features of most local drives and file system calls. Object Mount's POSIX Mode recreates these features and behaviors in the object storage world.

When enabled, POSIX Mode generates, stores, and maintains the metadata that is typically expected in a local file system, including:

- File ownership (uid, gid)
- Read/Write/Execute permissions
- Timestamps (mtime, ctime, etc.)
- Symbolic links and directory flags (where supported)

To retain this metadata, Object Mount writes it as objects in a hidden folder inside each directory of your object storage bucket. Object Mount hides these folders from view.

> **NOTE**
>
> **POSIX Mode Requirements**
>
> To support POSIX Mode, ensure your S3 credentials allow, and your bucket supports, write access. Your credentials must allow `s3:PutObject` and `s3:DeleteObject`.

> **WARNING**
>
> **Accessing Content Outside of Object Mount**
>
> Using any non-Object Mount tool to rename, move, or copy your files will result in those objects *losing* their POSIX metadata. You should only use Object Mount to manage POSIX-enabled files in order to preserve their metadata and attributes.

When to Enable Object Mount's POSIX Mode

You should enable POSIX mode in Object Mount if:

- You are working with software that checks for or enforces POSIX-style permissions
- You require symbolic link emulation or fine-grained permission mapping
- You are mounting object storage as a shared filesystem in team environments

Enabling POSIX mode is recommended for workflows such as:

- Media Production
- Backup Solutions
- Archive Systems
- File Synchronization
- File Manager/Tools

When NOT to Enable Object Mount's POSIX Mode

You should *not* enable POSIX Mode in Object Mount for:

- **Read-only Bucket Credentials**

  POSIX Mode requires *write* access to the object storage bucket in order to write the hidden metadata objects. If your credentials are read-only, POSIX mode will not function and may prevent the mount from working correctly.

- **Lightweight Access**

  For read-only workflows or general content browsing, POSIX mode is typically unnecessary and can be left disabled.

Enabling POSIX Mode: macOS and Windows

POSIX mode is enabled on a mount-by-mount basis: You can choose which mounts require it and which don't.

You enable POSIX mode when **creating a new mount** within Object Mount for macOS or Windows.

> **WARNING**
>
> **Important:** You cannot change the POSIX Mode setting after a mount has been created in Object Mount for macOS or Windows.

Enabling POSIX mode in **macOS** and **Windows** environments is done through Object Mount's graphical user interface (GUI).

See the associated page for your operating system:

- Enabling POSIX Mode when creating a mount in macOS
- Enabling POSIX Mode creating a mount in Windows

NOTE

POSIX Feature Support on Mac & Windows

User and Group identity features are not supported on Mac and Windows platforms:

- The owner of cloud objects is always reported to be the current user.
- The directory mode is reported as `0777` and the file mode is reported as `0666`.

Also, POSIX access controls are not enforced by Object Mount on Mac and Windows.

For a deeper understanding of the full suite of POSIX Mode Features see the Linux Getting Started Guide article: and the Linux User Guide article: .

Enabling POSIX Mode: Linux

POSIX support in Object Mount for Linux includes more options, more modes, and offers more controls for the end user to configure their environment according to their unique needs and use cases.

Configuration of POSIX options in Object Mount for Linux is achieved:

- Via the command line interface (the Object Mount CLI)
- Through the setting of environment variables
- Applying “tags” directly to cloud buckets using your object storage provider’s dashboard

Use the links below to **learn** about the many POSIX Modes, options, and settings — and then **enable** the correct POSIX Mode for you use case:

Linux Next Steps	Description
Learn about POSIX Options	Understanding POSIX Modes in Object Mount for Linux
Enable POSIX File Access	Enabling POSIX File Access Mode in Object Mount for Linux
Enable Enforced POSIX File Access	Enabling *Enforced* POSIX File Access Mode in Object Mount for Linux

Moving POSIX attributes to the .om folder

> **WARNING**
>
> **Experimental.** The `.om` format, the `omposix` bucket tag and `cuno creds convertposix` may still change. Try them on a copy of a bucket first.

In POSIX mode, Object Mount keeps each directory's owner, group, mode, times, symlinks and hardlinks as empty objects in a hidden folder of that directory. Releases before `.om` name that folder `.` followed by the private-use character U+FBF80. The names inside it use characters such as space and `" * < > ? |`. Windows and some bucket copy tools reject them.

The `.om` format keeps the attributes in a folder named `.om`, and its names use only lowercase letters and digits. A bucket switches to it with the `omposix` tag. This page covers starting a new bucket on `.om`, migrating an existing one, backing up and restoring the attributes, and rolling back.

The values of omposix

Value	Clients write	Clients read	Use
`sidefiles-enforced`	`.om`	`.om`	A new or fully migrated bucket
`sidefiles-permissive`	`.om`	`.om`	The same, in permissive POSIX mode
`sidefiles-migration-enforced`	`.om`	`.om` and legacy folder	During a migration
`sidefiles-migration-permissive`	`.om`	`.om` and legacy folder	The same, in permissive POSIX mode
`sidefiles-legacy-enforced`	legacy folder	`.om` and legacy folder	Rolling back from `.om`
`sidefiles-legacy-permissive`	legacy folder	`.om` and legacy folder	The same, in permissive POSIX mode

Every value hides both folders. When a client reads both and both hold a record for the same file, it uses the newer one. Finish on `sidefiles-enforced` or `sidefiles-permissive`. They list one folder per directory instead of two.

Before you start

- **Every client must read .om.** Releases before `.om` ignore the `omposix` tag. They don't see attributes in `.om`, and they write the legacy folder again. Upgrade every machine that mounts the bucket first. Also run `cuno creds setposix <URI> false`, which stops a client you missed from using POSIX mode on the bucket. Clients that read `.om` follow `omposix` and ignore that older tag.
- **Credentials.** The steps need credentials that can list, read, write and delete objects in the bucket and edit its tags.
- **Migrating is offline.** Nothing may mount or use the bucket while `cuno creds convertposix` runs or while you change the tag. A new bucket needs no conversion.
- **S3 only.** `convertposix` supports S3 and S3-compatible stores so far. Azure and Google Cloud Storage buckets can use `.om` from the start but can't migrate yet.

A new bucket

Tag the bucket before anything writes to it:

```
cuno creds setposix --experiment s3://examplebucket sidefiles-enforced
cuno creds setposix s3://examplebucket false
```

The first command refuses if the bucket already has objects under a `.om` folder, because these values hide `.om` and treat it as their own. It also refuses if the bucket has a legacy sidefile folder, which `sidefiles-enforced` doesn't read. Pass `--force` only if nothing needs those objects.

To check the value, give only the URI:

```
cuno creds setposix --experiment s3://examplebucket
```

Making a backup

The migration changes three things in a bucket. It sets the tags `cunoposix`, `cunoposixmetadata` and `omposix`, it writes the `.om` folders, and `--delete-legacy` deletes the legacy folders. It never touches file data. A copy of the tags and the legacy folders is enough to put the bucket back.

The commands use the AWS CLI. For an S3-compatible store, add `--endpoint-url <endpoint>` to each one. The credentials must read the bucket and its tags and write to a second bucket, here `examplebucket-posix-backup`.

1. **Stop every client**, so the attributes don't change during the copy.
2. **Save the bucket tags:**

   ```
   aws s3api get-bucket-tagging --bucket examplebucket > examplebucket-tags.json
   ```

   A bucket without tags makes the command fail with `NoSuchTagSet`. Write that down instead.
3. **Copy the legacy folders** to the backup bucket. Each one is named `.` followed by U+FBF80. The `printf` below writes that character's UTF-8 bytes in any locale. Bash's `$'\U000FBF80'` produces a different name in the C locale.

   ```
   legacy=$(printf '.\363\273\276\200')
   aws s3 sync s3://examplebucket s3://examplebucket-posix-backup \
       --exclude '*' --include "*$legacy/*"
   ```

   S3 copies the objects from bucket to bucket, so the names never need to be valid local file names.
4. **Check the copy.** The backup should hold as many objects as the dry run reports as legacy objects:

   ```
   aws s3 ls --recursive s3://examplebucket-posix-backup | wc -l
   ```

An existing bucket

1. **Stop every client** that mounts or uses the bucket.
2. **See what the conversion would do.** A dry run writes nothing and doesn't tag the bucket:

   ```
   cuno creds convertposix --dry-run --tag sidefiles-migration-enforced s3://examplebucket
   ```

   It prints how many attributes, ACLs and links it would write into `.om`, and how many legacy objects `--delete-legacy` would delete.
3. **Back up the tags and legacy folders** as described in Making a backup.
4. **Convert and keep the legacy folder.** The command tags the bucket with the migration value, writes each directory's `.om` folder and checks that it reads the same as the legacy folder:

```
cuno creds convertposix --tag sidefiles-migration-enforced s3://examplebucket
cuno creds setposix s3://examplebucket false
```

It asks before writing, and `--yes` skips the question. It reports any directory it can't convert and leaves that directory alone. If the run stops, run the same command again to finish.

5. **Check the result.** Mount the bucket with an upgraded client and run `ls -l` on a few files. Owners, modes, times and symlinks should match what they were. Clients read both folders under the migration value, so the bucket can stay in use until you finish. Stop every client again before the next step.
6. **Delete the legacy folder:**

   ```
   cuno creds convertposix --delete-legacy s3://examplebucket
   ```

   The command converts any directory that changed since the last run again. It deletes a directory's legacy folder once the `.om` folder alone reads the same. After the last directory, it prints the command for the next step.
7. **Set the final value**, so clients read only `.om`:

   ```
   cuno creds setposix --experiment s3://examplebucket sidefiles-enforced
   ```

   It refuses while any legacy sidefile is left and names some of them.
8. **Start the clients again.** Keep the backup until you're sure you won't restore it.

For a bucket in permissive POSIX mode, use `sidefiles-migration-permissive` and then `sidefiles-permissive`.

By default `convertposix` writes path components of at most 250 characters and keys of at most 1015. If the bucket's credentials set a lower `max-path-element` or `max-path`, pass `--split-at` with the element limit minus 3, and `--max-path` with the key limit.

Rolling back

If `.om` causes problems, stop every client and set a rollback value. Clients then write the legacy folder again and still read both:

```
cuno creds setposix --experiment s3://examplebucket sidefiles-legacy-enforced
```

Use `sidefiles-legacy-permissive` for a bucket in permissive mode. Clients still read the attributes already in `.om`, and write new ones to the legacy folder.

Only clients that read `.om` know the rollback values. A release from before `.om` doesn't see the attributes in `.om`, so going back to one loses them. To go back that far, restore the backup instead.

To move forward again, stop every client and migrate what clients wrote since the rollback, as for an existing bucket:

```
cuno creds setposix --experiment s3://examplebucket sidefiles-migration-enforced
cuno creds convertposix --delete-legacy s3://examplebucket
cuno creds setposix --experiment s3://examplebucket sidefiles-enforced
```

`convertposix` refuses a bucket with a rollback value.

Restoring the backup

A restore puts the bucket back on the legacy folder, as it was before the migration. Clients from before `.om` can use it again. The restore loses any attribute changes clients wrote into `.om` after the backup. File data written since then stays.

1. **Stop every client.**
2. **Delete the .om folders:**

   ```
   aws s3 rm s3://examplebucket --recursive \
       --exclude '*' --include '.om/*' --include '*/.om/*'
   ```

   The bucket had no `.om` folders before the migration, because `setposix` and `convertposix` refuse to tag a bucket that has them.
3. **Copy the legacy folders back:**

   ```
   aws s3 sync s3://examplebucket-posix-backup s3://examplebucket
   ```

4. **Restore the tags:**

   ```
   aws s3api put-bucket-tagging --bucket examplebucket \
       --tagging file://examplebucket-tags.json
   ```

   If the bucket had no tags, remove them with `aws s3api delete-bucket-tagging --bucket examplebucket`.
5. **Start the clients.**

Limits

- A migration needs the bucket offline. There is no conversion yet that runs while clients write.
- Under a migration or rollback value, an ACL that no file uses can stay in the folder that clients only read, until something removes or converts that directory. It costs one empty object.
- On macOS and Windows, Object Mount doesn't show owners and groups. See Technical Limitations.

Libuplink mode (Storj only)

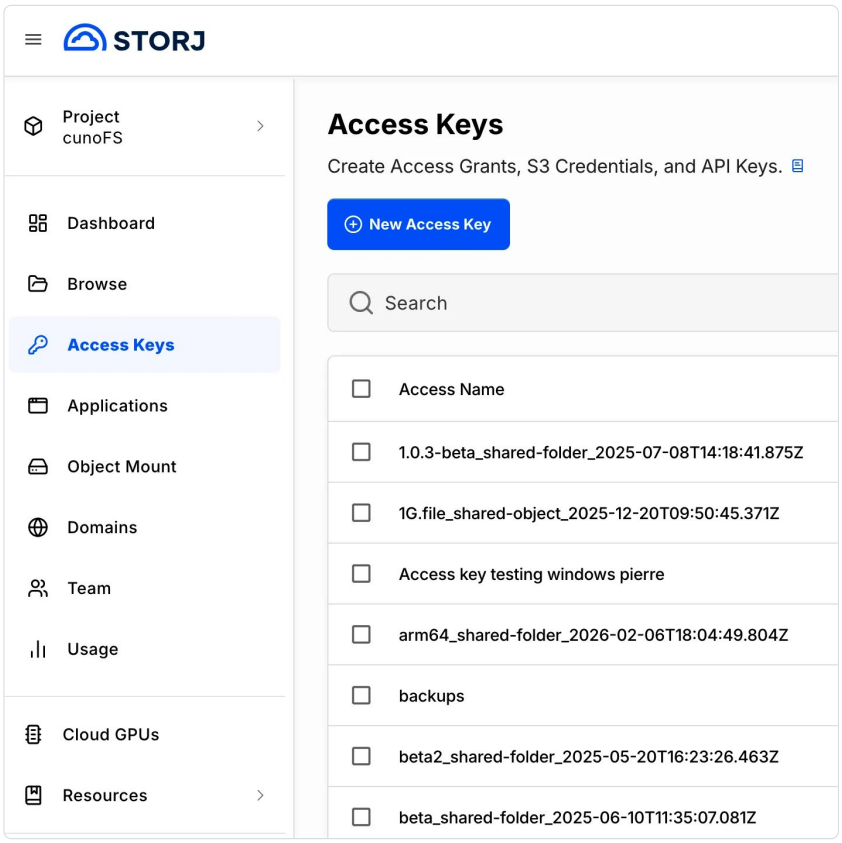
Object Mount can connect to Storj's Distributed Cloud Storage offering as well using Storj libuplink. This connection method is read only and designed for the quick data migration from Storj Object Storage.

Obtaining a latest version of Object Mount

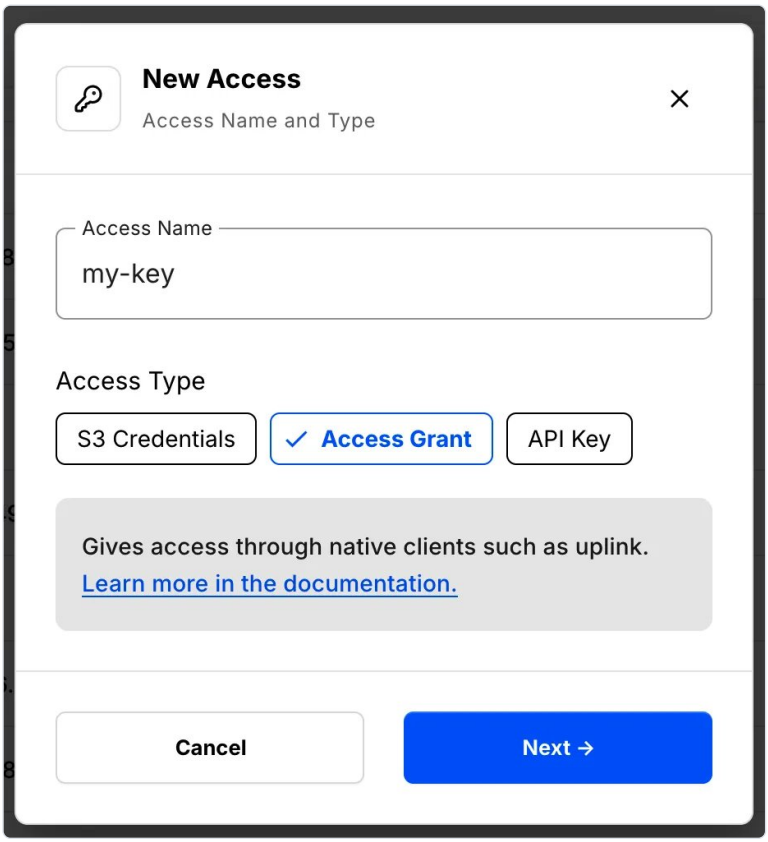
1. Download latest Object Mount release for your OS
2. Install and reboot if asked

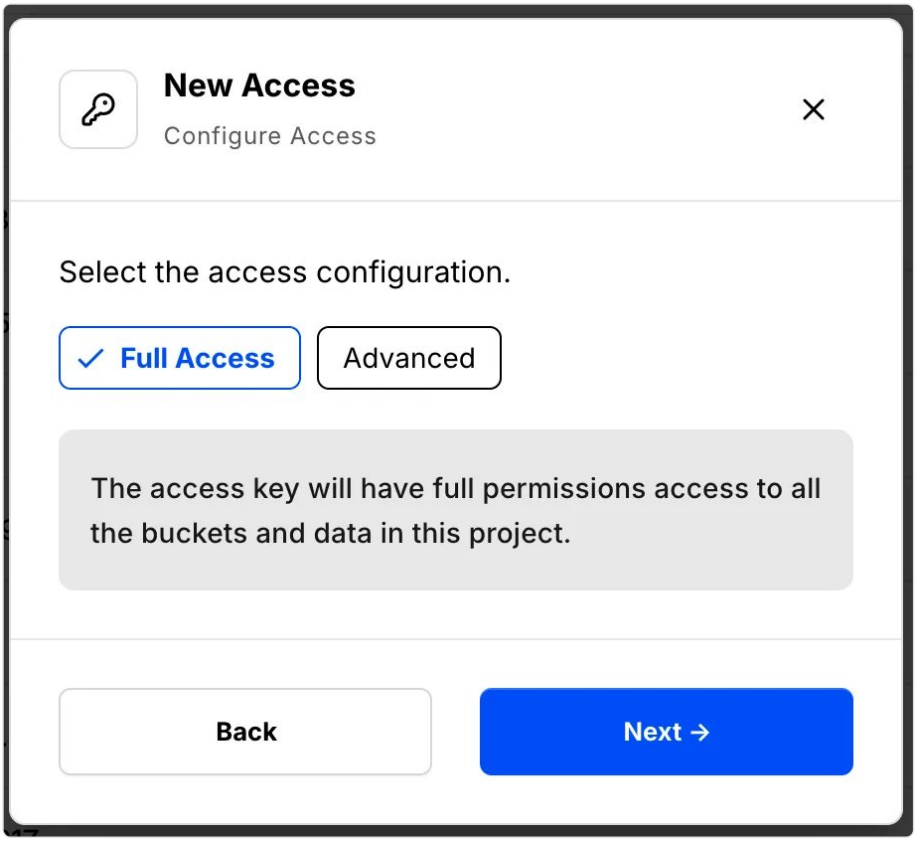
Obtaining an access grant for your Storj project

Open your project in Storj web portal, go to **Access Keys**



and create a new access grant:





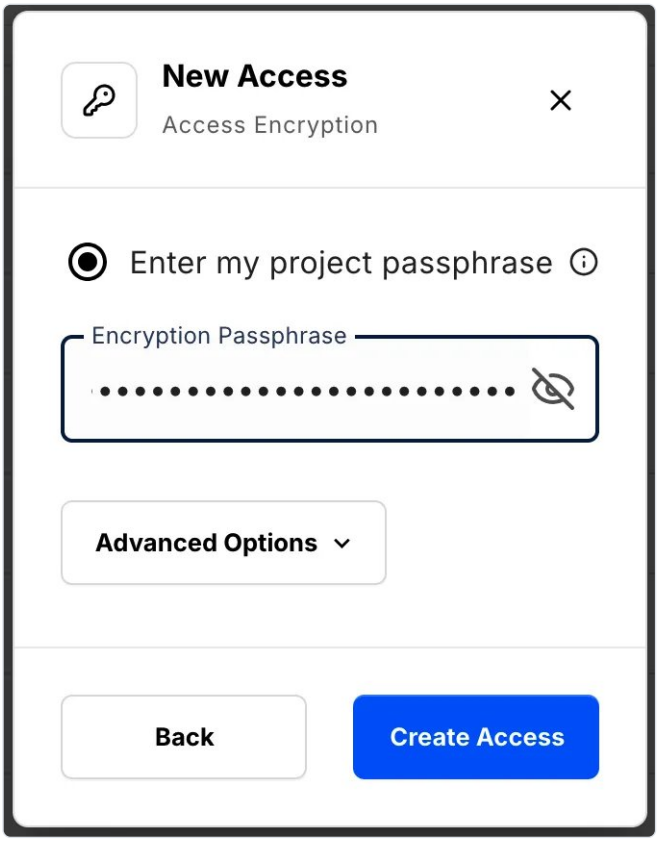
If you have a project with a Manual encryption phrases management, then you need to provide your encryption phrase, used to upload previous objects.

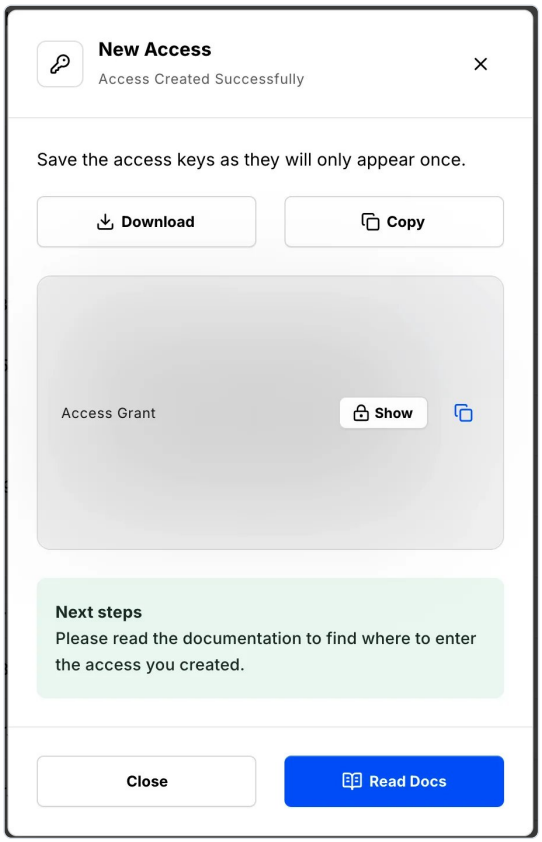
WARNING

This passphrase is important! Encryption keys derived from it are used to encrypt your data at rest, and your data will have to be re-uploaded if you want it to change!

Importantly, if you want two access grants to have access to the same data, **they must use the same passphrase**. You won't be able to access your data if the passphrase in your access grant is different than the passphrase you uploaded the data with.

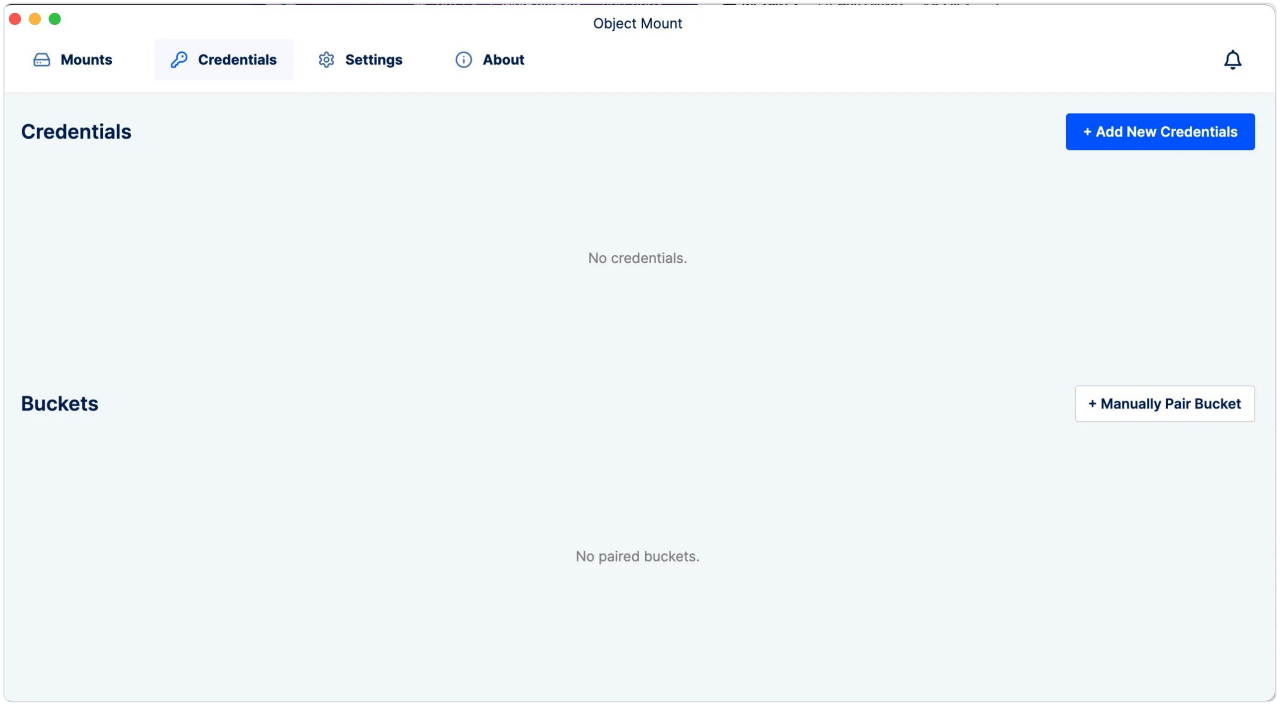
Please note that **Storj does not know or store your encryption passphrase**, so if you lose it, you will not be able to recover your files.



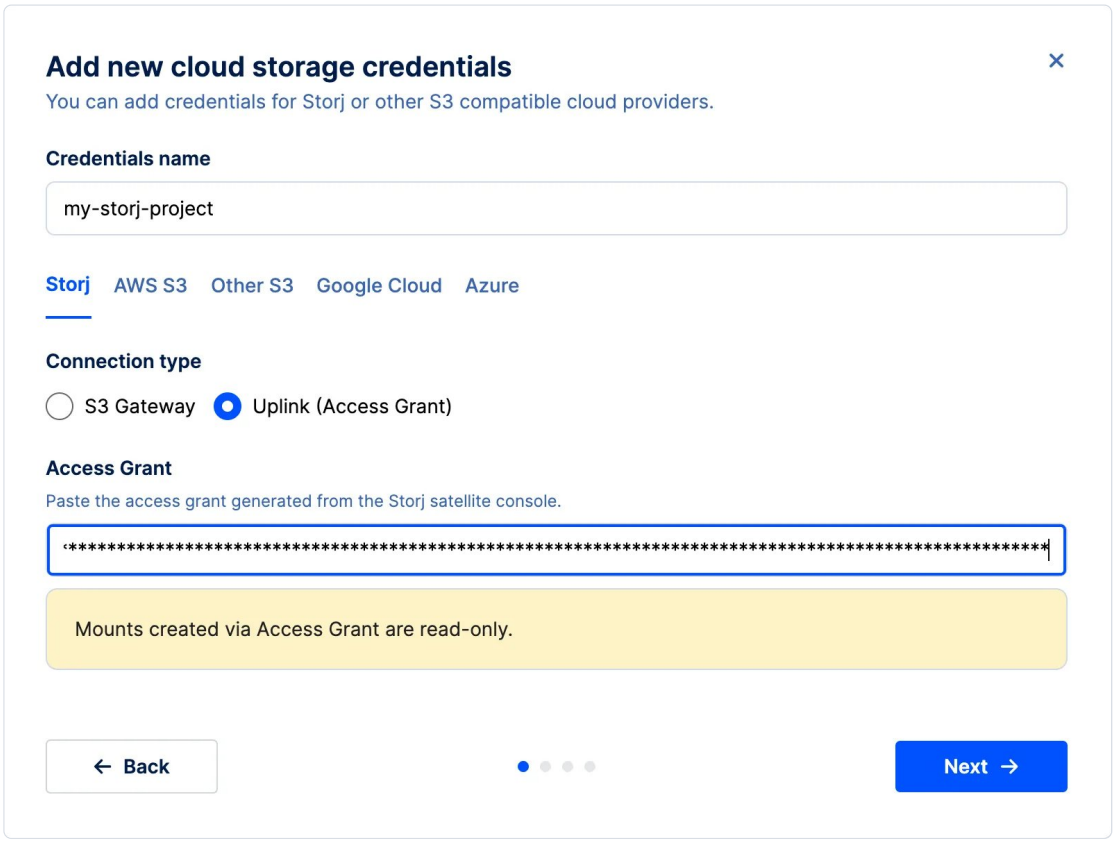


Configure a new Credentials record

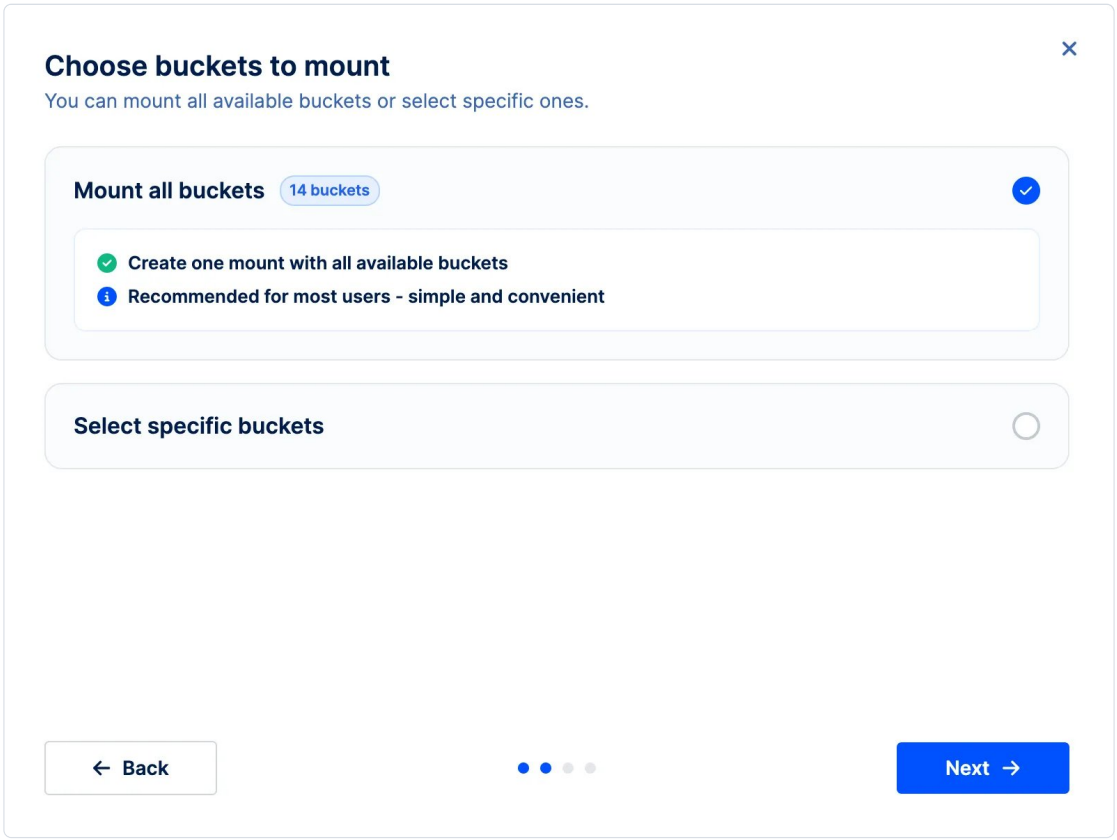
1. Open Object Mount, navigate to **Credentials** > *Add New Credentials*



1. Select **Storj**, switch to **Uplink (Access Grant)** and paste the access grant created previously



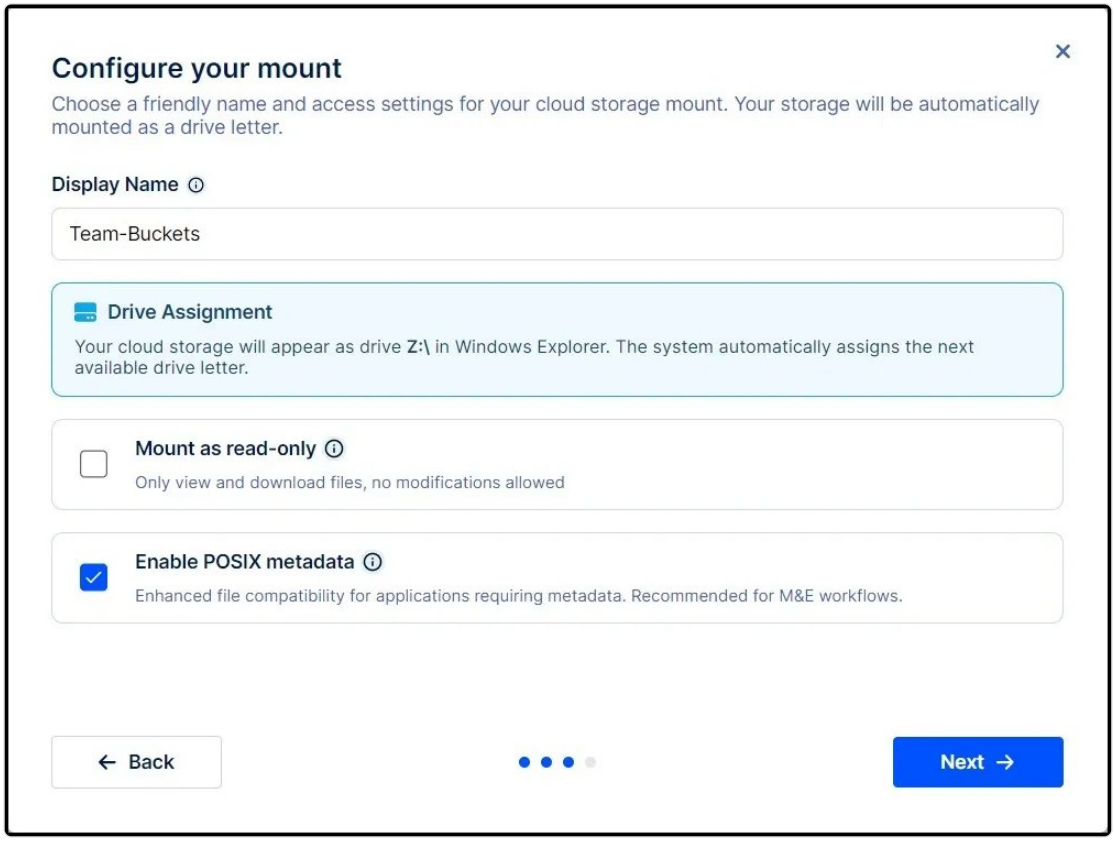
1. Complete the procedure by selecting all buckets or specific buckets you want to add to the mount



Add a new Mount

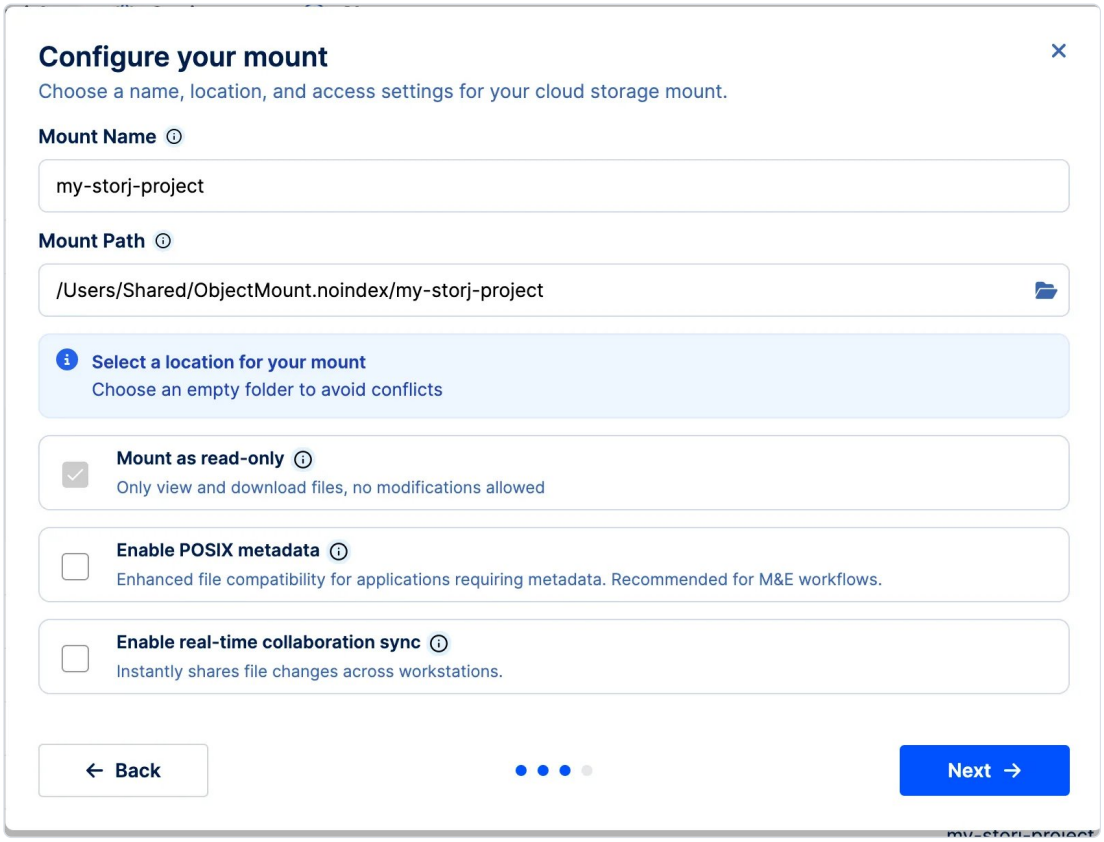
Navigate to **Mounts** and create additional mounts if desired using ***Create New Mount*** and specifying buckets and mount points.

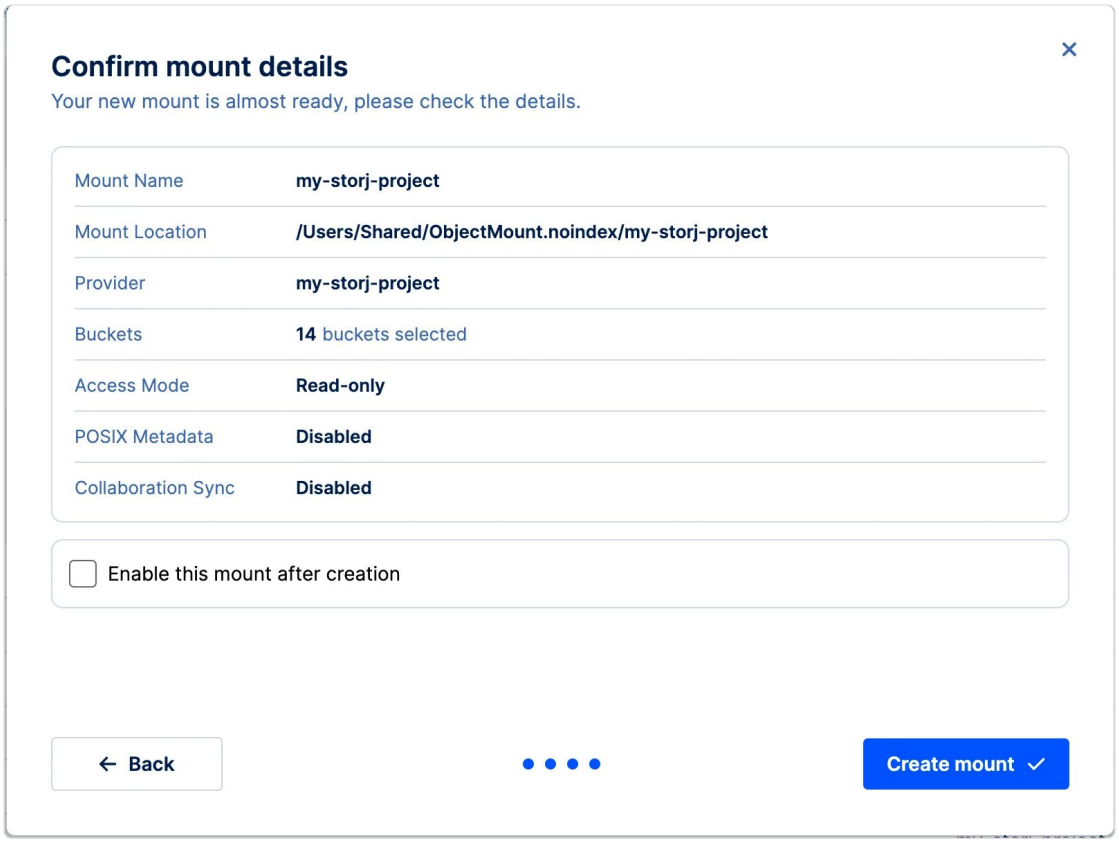
Windows



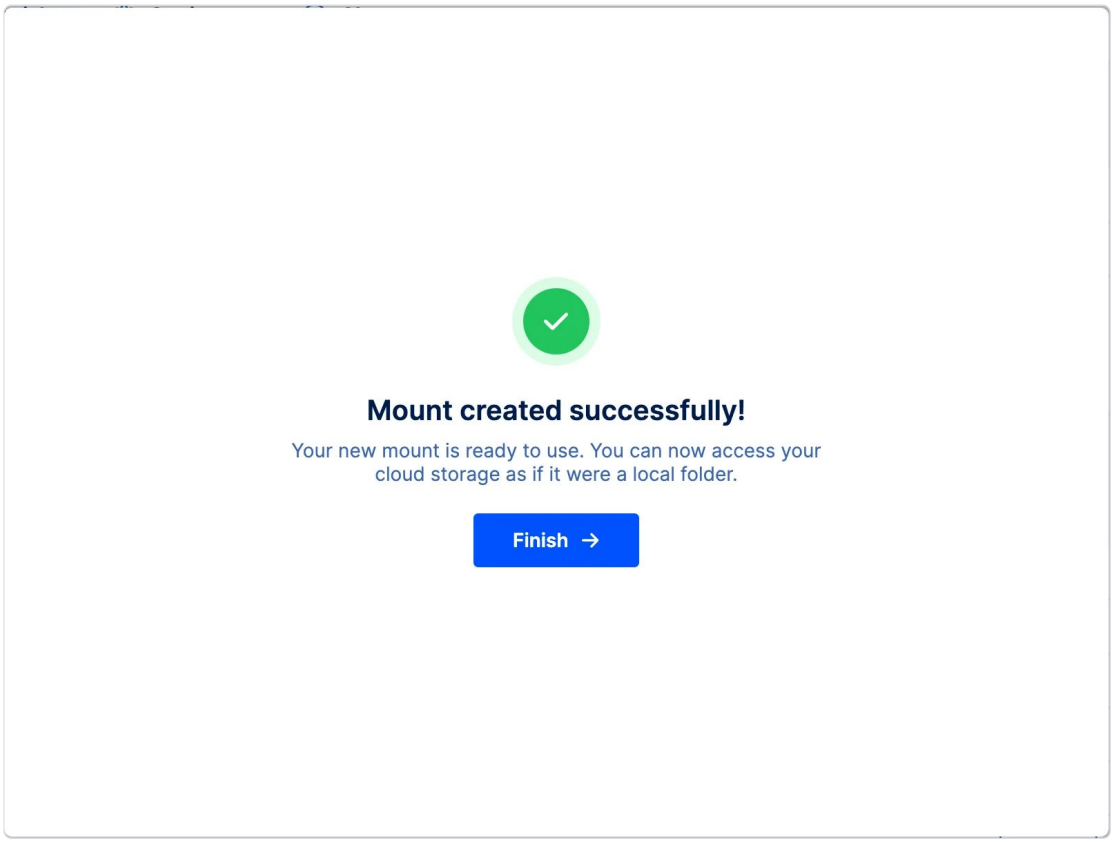
See for Windows.

macOS



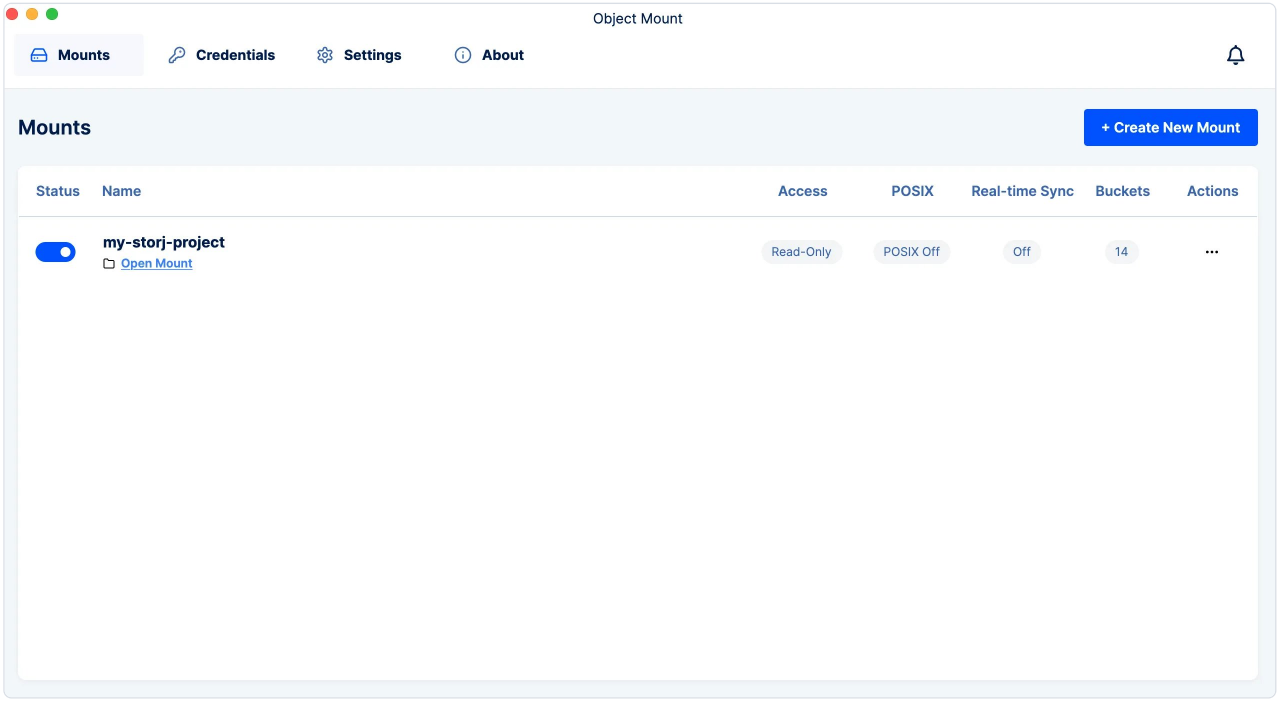


See for macOS.



Mount a bucket with a new Mount

Enable one of the added mounts and use it as a read-only file system to migrate the data.



See the Advanced Guide article for additional credential options.

S3 Credential Management

Object Mount can connect to Storj's Distributed Cloud Storage offering as well as other major S3-compatible cloud storage providers.

You will need to create and/or obtain credentials from your S3 provider, and then configure Object Mount to use those credentials.

Note: API access may also need to be enabled for some 3rd-party S3 providers. Consult your storage solution's documentation to learn how to enable API access.

Obtaining Your S3 Credentials

You will need to obtain the following three items from your storage provider:

Variable	Description
Access Key ID	Example: j4SQYkDxxxxxxxxxxsaSRYAahZgh
Access Key Secret	Example: vMxJBmUxxxxxxxxxxxxxxxxxxxxxxxxxxxK4kKvwJnw8PwKpj4PmNvL
Endpoint	An accessible IP address or domain/subdomain/host to use as an endpoint. Example: https://gateway.storjshare.io

> **WARNING**
>
> **Credential Requirements**
>
> The credentials you obtain need to have sufficient permissions in order for Object Mount to discover, read, write, and manage your data, including the permission to list buckets.
>
> If this is not possible (or desired), you can use `cuno creds pair` options. See the **Pairing Containers and Credentials** section in the Linux Advanced Configuration Guide article: Advanced Credential Options.

For instructions on how to create and/or obtain your S3 storage credentials, select the tab below that corresponds to your S3 provider:

AWS S3

Amazon Web Services

AWS EC2 with IAM:

If you are accessing S3 buckets through an Amazon AWS EC2 instance configured with an IAM role, no further configuration is needed and Object Mount will automatically authenticate using the AWS-managed configuration.

Using Existing Credentials:

You will need the Access Key ID and Access Key Secret for an AWS **IAM user** with permission to access your S3 buckets. These credentials would need to have been saved when the IAM user was *first created*.

Creating New Credentials:

Alternatively, create a new **IAM user** with "programmatic access" (access using keys), by following the AWS User Guide: 🌐 Creating an IAM user in your AWS account.

The steps should be similar to the following:

1. Log into the AWS S3 Console: 🌐 https://s3.console.aws.amazon.com
2. Navigate to: **Services > IAM**
3. Open **Users** and click on the **Add user** button
4. Set a username
5. Select the **Programmatic access** option
6. Click on **Next**
7. On the **Set Permissions** page:
8. Select **Attach existing policies directly**
9. Select an existing policy, e.g.: **AmazonS3ReadOnlyAccess** or **AmazonS3FullAccess**
10. or Create a new policy that has the **s3:ListAllMyBuckets** permission and at least the **s3:ListBucket** and **s3:GetObject** permissions for each bucket available to the user
11. Click **Next**
12. Review and confirm to create the new user
13. Download the CSV file by pressing the corresponding button

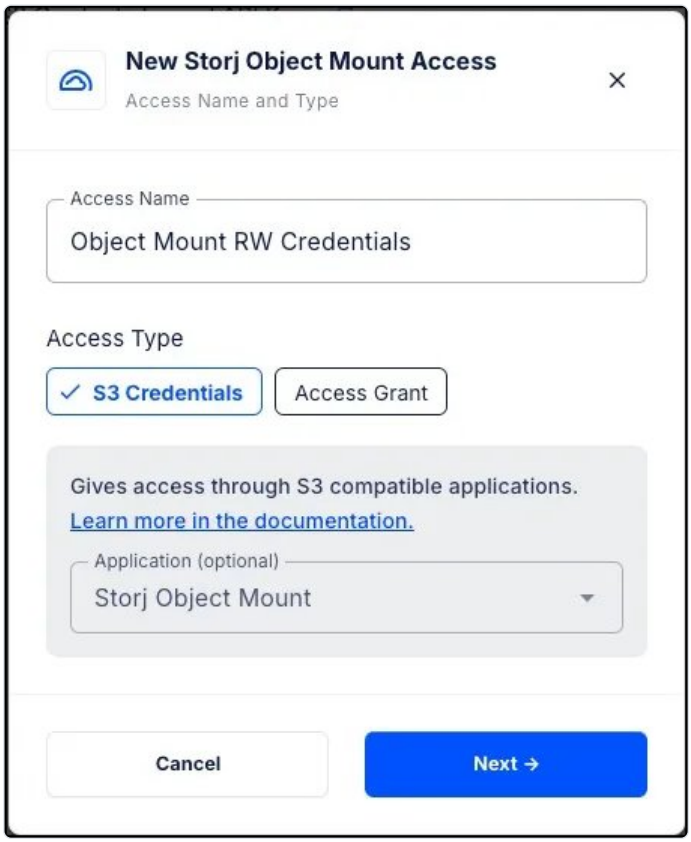
Storj

Storj Object Storage

Creating New Storj Credentials:

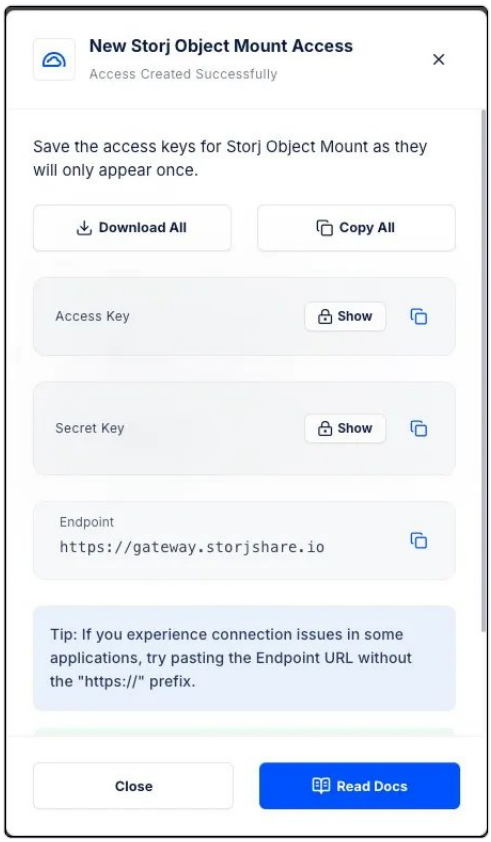
To create new Storj credentials to be used by Object Mount, follow these steps:

1. Log into the **Storj Dashboard** for your Satellite.
   For example: 🌐 https://us1.storj.io/login
2. Open your **Project** and click on **Access Keys** in the left-hand menu.
3. Click the **New Access Key** button.
4. Enter a **Name**, select **S3 Credentials** and choose **Storj Object Mount** from the Application drop-down menu. Click Next.



5. Choose **Full Access** and click Next.
6. Click **Create Access**.
7. Click the **Copy All** button (or selectively click **Show** / **Copy** for each value).

   Important: These values will *only appear once*. Copy them now!



8. (Optional) Click **Download All** to save the credentials to a text file.

   Note: This saved text file is *not* in the correct format for importing into Object Mount for Linux. See below for details on creating a credential file with the correct syntax.

Microsoft Azure

Microsoft Azure

You will need the account name and the associated access key. No region or endpoint is required.

To set up a new account, see 🌐 Create an Azure storage account.

If you already have an account and need the access key, see 🌐 View account access keys.

The steps should be similar to:

1. Log into the Azure Portal: 🌐 https://portal.azure.com/
2. From **services**, select **Storage Accounts**
3. Click on the Storage Account you want the key for
4. In the **Security + networking** section, click **Access keys**
5. Click on the **Show keys** button
6. Save the **Storage account name** and **Key**

Alternatively, Object Mount for Linux supports using Shared Access Signatures (SAS) to access Azure containers. See the **Microsoft Azure Storage with Shared Access Signatures** section in the Linux Advanced Configuration Guide article: Advanced Credential Options.

Google Cloud

Google Cloud Platform

You will need key-based access to be associated with your 🌐 Google Cloud service account.

If you need to set this up, you can have Google generate a new key by following Google's guide at 🌐 Create and delete service account keys.

The steps should be similar to:

1. Log into the GCS portal: 🌐 https://cloud.google.com/
2. Navigate to the **Service Accounts** page
3. Select the **project** you want to provide access to
4. Click the **email address** of the service account that you want to create a key for
5. Click the **Keys** tab
6. Click **Add key** from the drop-down menu, then select **Create new key**
7. Select **JSON** as the **Key type** and click **Create**
8. Save the downloaded JSON file.

Alternatively, you can generate your own key pair and upload it on the service account's key management page, by following this guide 🌐 Upload service account keys.

Other S3 Compatible

S3 Compatible Providers

Consult your object storage solution's documentation to find out how to retrieve credentials for authentication using S3 APIs.

You will need an **Access Key ID**, an **Access Key Secret**, and an accessible IP address/URL or domain/subdomain/host to use as an **endpoint**.

Using & Saving Your S3 Credentials

Depending on your S3 storage platform, you may only be able to see, copy, and save your access credentials *at the time you create them*. (For increased security, some platforms prevent the display of previously generated credentials.)

Therefore, be sure to copy and record your **Access Key ID** and **Access Key Secret** at the time you *create them*.

Entering Credentials: Object Mount for macOS and Windows

When activating Object Mount in **macOS** and **Windows**, you will need to copy/paste these keys with Object Mounts's graphical interface.

Entering Credentials: Object Mount for Linux

When activating Object Mount for Linux, you can copy/paste your your credentials, or you can import them from a previously saved text file.

See below for instructions (and the syntax required) to save you credentials to a text file to be used for import.

Saving Credentials as a File

A text file can be used to import your S3 credentials when activating a mount in Object Mount for Linux.

Credential files are `plain-text` and must comply with the proper format defined for your specific cloud provider.

Select your S3 storage provider below for details on syntax and variable naming:

AWS S3
Amazon Web Services

There are two equivalent formats for AWS S3 credential files. Use either one:

```
aws_access_key_id = <access_key_id>
aws_secret_access_key = <secret_access_key>
region = <region>
```

```
AWSAccessKeyId = <access_key_id>
AWSSecretKey = <secret_access_key>
region = <region>
```

`region` is optional.

Object Mount can also use an **AWS S3 Access Point** instead of a container.

For more details, see the **AWS S3 Access Point Support** section in the User Guide article: .

Storj
Storj Object Storage

Storj's object storage is S3-compatible.

Therefore, the syntax used for saving Storj credentials mimics the format used for AWS S3 storage:

```
aws_access_key_id = <access_key_id>
aws_secret_access_key = <access_key_secret>
endpoint = <endpoint>
```

`endpoint` is required.

The primary endpoint for Storj is: `endpoint = https://gateway.storjshare.io`

Microsoft Azure
Microsoft Azure

For Microsoft's Azure cloud storage, the credential file should conform to the following format:

```
AZURE_STORAGE_ACCOUNT= xxxxxxxxxxxxxxxxx
AZURE_STORAGE_ACCESS_KEY= xxxxxxxxxxxxxxxxx
```

No region or endpoint is needed.

Google Cloud
Google Cloud Platform

If you have an existing JSON/P12 (PCKS#12) file — that file is formatted correctly and will import properly.

If you provided your own private key when creating the service account, you will need to create a JSON file which conforms to the following format:

```
{
   "type": "service_account",
   "project_id": "PROJECT_ID",
   "private_key_id": "KEY_ID",
   "private_key": "-----BEGIN PRIVATE KEY-----\nPRIVATE_KEY\n-----END PRIVATE KEY-----\n",
   "client_email": "SERVICE_ACCOUNT_EMAIL",
   "client_id": "CLIENT_ID",
   "auth_uri": "https://accounts.google.com/o/oauth2/auth",
   "token_uri": "https://accounts.google.com/o/oauth2/token",
   "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
   "client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x509/
SERVICE_ACCOUNT_EMAIL"
}
```

Other S3 Compatible

S3 Compatible Providers

The file needs to contain at least:

```
aws_access_key_id = <access_key_id>
aws_secret_access_key = <access_key_secret>
region = <endpoint>
```

See the Advanced Guide article for additional credential options and endpoint formatting details.

Securing Your S3 Credentials

You can use any text editor to create the credential file — just remember to change the permissions on the file when you're done to prevent other users from accessing this sensitive file:

```
#terminal
chmod 0600 "<path to your credentials file>"
```

NOTE

Time Saving Tip

You can combine the **creation** of the file and securing **permissions** on the file in a single shell command (insert your credential details):

```
#terminal
touch credentials.txt ; chmod 0600 credentials.txt ; printf "aws_access_key_id =
xxxxxxxxxxxxxxxxxx \naws_secret_access_key = xxxxxxxxxxxxxxxxx \nendpoint=A.B.C.D:PORT" >
credentials.txt
```

SSee the Advanced Guide article for additional credential options.

Supported S3 Providers

Object Mount can be used with any S3-compatible object storage provider, including these **Primary S3 Providers**:

- 🌐 Storj's Object Storage
- 🌐 Amazon Web Services S3
- 🌐 Microsoft Azure Blob Storage
- 🌐 Google Cloud Platform
  - **Note:** For best GCP performance, we recommend using S3 with their 🌐 S3 Access Point, *not* with their S3 Gateway.

Tested Storage Providers

In theory, Storj's Object Mount works with *any* S3-compatible object storage provider.

In practice, each provider's "S3-compatible API" implementation can have slightly different nuances and behaviors — some additional configuration may be necessary.

Object Mount has been successfully tested with the **Compatible S3 Providers** below:

- 🌐 Oracle Cloud Infrastructure (OCI) Object Storage
- 🌐 Wasabi Cloud Object Storage
- 🌐 Min.IO AIStor
- 🌐 NetApp StorageGRID
- 🌐 Dell ECS Object Storage

Additional S3 Providers

The following S3 storage providers have not yet been validated by Storj — however, users have reported success with the **Additional S3 Providers** below:

- 🌐 IBM Cloud Object Storage
- 🌐 Backblaze B2 Cloud Storage
- 🌐 DigitalOcean Spaces
- 🌐 Cloudflare R2
- 🌐 Scality
- 🌐 DataDirect Networks (DDN) Storage

Performance Tuning

There are many variables to consider when optimizing **Object Mount** for the best performance, including:

- Operating System behavior
- Connection Speed
- Storage Provider Characteristics

The sections below will provide tips on optimization for each of these three areas, tailored for your specific environment.

Operating System Tips

Linux

- Use a recent kernel (5.10+ preferred) for improved FUSE performance.
- Ensure `fuse3` and required kernel modules are installed.
- Run Object Mount as a `systemd` user service if automatically mounting on login.
- Place cache directories (`data cache`, `metadata cache`) on a fast, local SSD or NVMe volume.
- Adjust `ulimit -n` if handling many open files (e.g. `ulimit -n 65535`).

macOS

- Intel Processors: Use macOS Monterey (12.0) or later for best compatibility.
- Apple Silicon (M-Chips): Use macOS Ventura (13.0) or later for best compatibility.
- Make sure **macFUSE 4.8.2 or newer** is installed and authorized in **System Settings > Privacy & Security**.
- Disable Spotlight indexing on mount paths to reduce background load.
- Cache directories should be located on fast internal storage (not external or network drives).
- POSIX mode can add overhead. Enable only if required by your workflow.

Windows

- Use Windows 10 (21H2+) or Windows 11 for best driver support.
- Place cache folders on fast local drives (e.g. `C:\Temp\Object MountCache`).
- Disable antivirus scanning on mount points if performance is impacted.
- Run the app with administrator rights if mounting to system-level paths.

Connection Speed Tips

Object Mount adapts well to different network conditions, but optimal performance depends on the speed and reliability of your connection to cloud storage.

Fast

10 Gbps or faster

- Increase **S3 connections limit** (e.g. 200+).
- If Client and Storage are on the same **Local Area Network (LAN)**, disable **Data cache**.
- Consider setting **metadata cache timeout** to a lower value for frequent sync.

Medium

1 Gbps to 10 Gbps

- Set **S3 connections limit** between 100-150.
- POSIX mode can add overhead. Enable only if required by your workflow.
- Use smart caching strategies if accessing large numbers of small files.

Slow

100 Mbps to 1 Gbps

- Keep metadata **cache timeout** higher (e.g. 300-600 seconds).
- Lower **S3 connections limit** (e.g. 50 or fewer) to prevent throttling.
- Limit the number of simultaneous mounts or background operations.
- Use proxy editing in creative apps only if working with very large assets.

Storage Provider Tips

Each storage backend behaves slightly differently depending on API responsiveness, region latency, and compatibility mode.

AWS S3

- Set an appropriate **region** to avoid cross-region latency.
- Use multipart uploads for large file writes (default behavior in Object Mount).
- Enable caching and proxy workflows for high-resolution media.
- Monitor AWS rate limits if using a high connection pool size.

Storj

- Use Storj-managed encryption projects.
- Use **lexicographically ordered buckets** for faster directory listings.
- Ensure your satellite and uplink are in the same region as your client when possible.
- POSIX mode is supported but not required for most workflows.

Microsoft Azure

- Use a compatibility layer or gateway if not directly S3-compatible.
- Access tiers (e.g. Hot, Cool) may affect read latency.
- Monitor API throttling if using many simultaneous requests.
- Proxy editing recommended over direct edit for best responsiveness.

Google Cloud

- Use the XML S3 API compatibility layer (via interoperability settings).
- Latency-sensitive: caching strongly recommended.
- Consider using a regional GCS bucket near your compute node.
- POSIX emulation may be limited by object store behavior.

Other S3

- Use the **S3-compatible** provider tab when importing credentials.
- Always test the endpoint and region for compatibility.
- Providers like Wasabi may require explicitly setting the region.

- Avoid POSIX mode if write operations to root are not supported.

More Questions?

Need help tuning for a specific workflow, app, or storage provider?

Reach out via our Object Mount 🌐 Support Team for additional assistance.

Troubleshooting Guide

This guide covers common issues encountered when using **Object Mount**, along with clear steps to resolve them.

If you're stuck, this is a good first place to check!

Still stuck?

- Reach out to others in the Storj Community Forum
- Or open a ticket with our Storj Support Team

Common Issues

Here are a few common issues to be aware of:

- POSIX mode requires write access and may silently fail if credentials are read-only.
- Mac Finder & Windows Explorer progress bars may not reflect actual transfer status. — Monitor network activity instead.
- Avid Media Composer has limited tolerance for long mount paths & non-local scratch files.
- Some S3-compatible providers require manual configuration (e.g. region or endpoint overrides).
- Mounts cannot currently be shared across users on the same machine (per-user context only).

Mount Doesn't Appear or Won't Activate

Possible Cause(s):

- Incorrect credentials
- No accessible buckets
- Dependency not installed (e.g.: macFUSE)
- POSIX mode enabled without write access

Things to Check:

- Verify your license is active
  - Windows and macOS: Click the **About** tab
  - Linux: run `cuno creds license`
- Re-import your credentials and confirm bucket visibility
- Ensure dependencies are installed:
  - **macOS**: macFUSE
- For read-only mounts, disable POSIX mode

Credentials Work Elsewhere, but Fail in Object Mount

Possible Cause(s):

Some S3-compatible providers require custom regions, endpoints, or flags.

Things to Check:

- Review the instructions on the **Other S3 Compatible** tab in the S3 Credential Management article
- Be sure to select your provider from the dropdown

- Set an explicit region (e.g. `us-east-1`)
- Ensure the endpoint is reachable from your network

Note: For Storj Object Storage, make sure the buckets are **lexicographically ordered** for best results.

Mount Is Very Slow or Freezes

Possible Cause(s):

- High latency to cloud provider
- File system operations triggering too many API calls
- No caching or prefetch enabled

Things to Check:

- Enable **data cache** and **metadata cache** in Preferences
- Set `CUNO_OPTIONS = -filePrefetch` to disable file pre-fetching
- Increase S3 connection pool (e.g. 150-200 for 1Gbps+)
- Avoid using POSIX mode unless required
- Place cache directories on an internal SSD

“Leave Files in Place” Doesn’t Work in Creative App

Possible Cause(s):

Some NLE Applications may try to verify write access or expect native macOS behavior.

Things to Check:

- Are you trying to import from a mounted Object Mount volume?
- Is POSIX mode conflicting with permission checks?
- Try relinking or using proxy workflows instead

Logs Don’t Show Much Info

Possible Cause(s):

By default, logging is set to `error` only.

Things to Check:

Enable more detailed logging:

1. Go to **Preferences > Advanced Settings**
2. Set **Log Level** to `debug` or `trace`
3. Reproduce the issue

Check the logs at:

- **macOS**: `~/Library/Application Support/Object Mount/cunoFS.log`
- **Windows**: `C:\Users\%username%\AppData\Local\Object Mount\cunofs.log`

Files Are Stalling on Paste

Things to Check:

If files are stuck during paste:

- Try using the **Fast Copy and Fast Move** on Mac or **Fast Paste** on Windows.

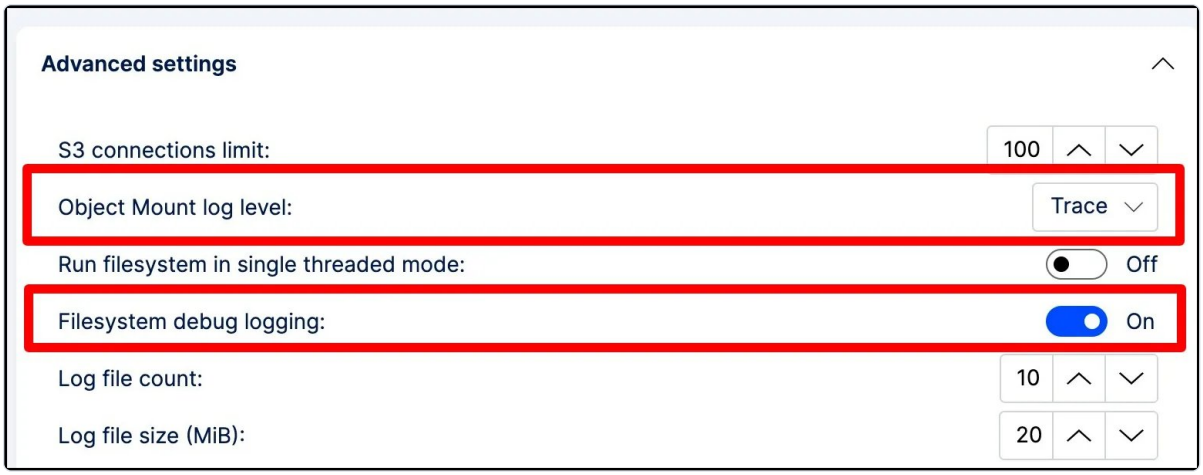
These features bypasses the Mac and Windows native copy tools and uses Object Mount's optimized transfer logic.

Still Not Working?

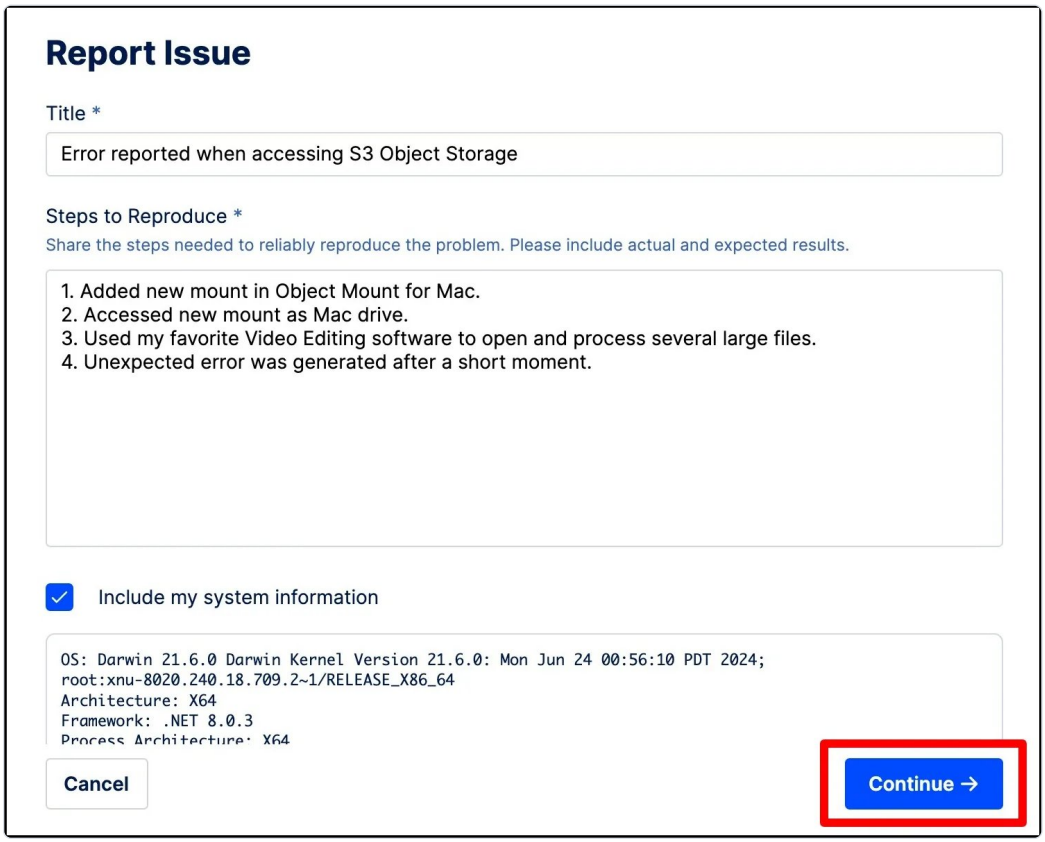
If you still need assistance in resolving your issue, follow the steps below to enable detailed logging and send the log files to the Storj Support team.

For macOS and Windows environments:

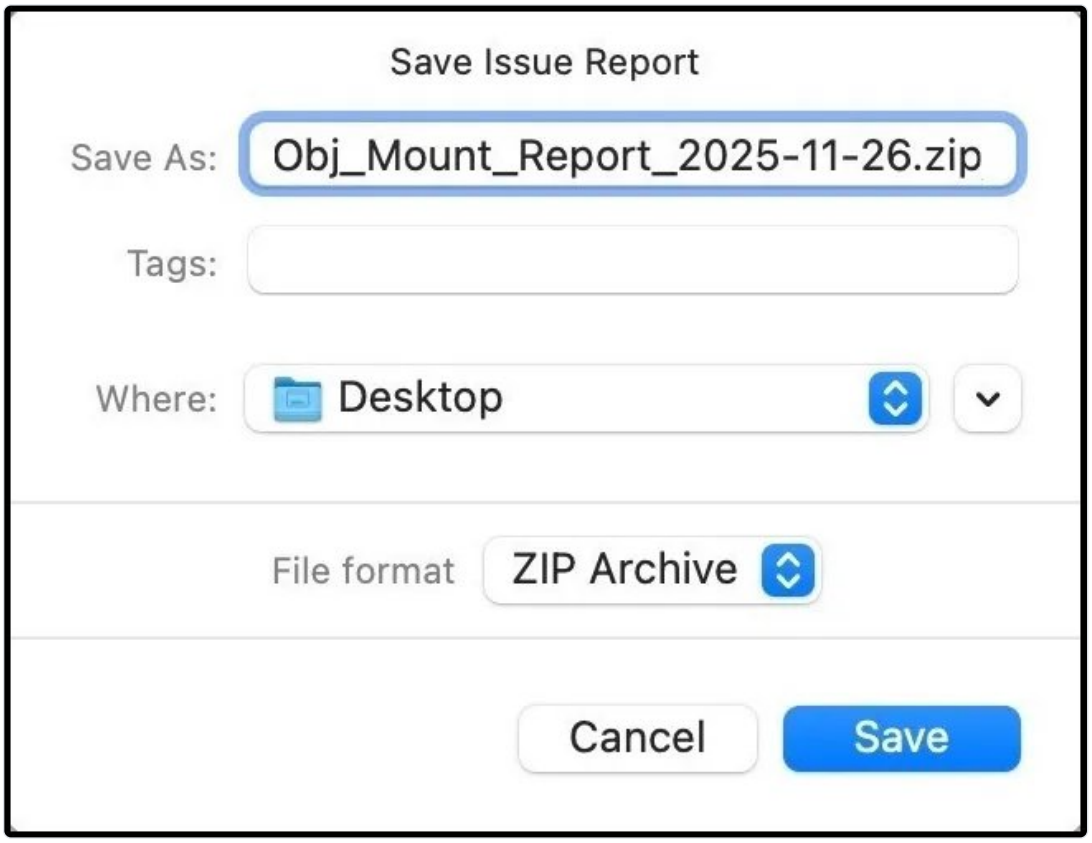
1. Run Object Mount, choose the **Settings** tab, and scroll down to **Advanced Settings**:
   - Change the "Object Mount log level" to `Trace`.
   - Set "Filesystem debug logging" to `On`.
   - Click the **Apply** button.



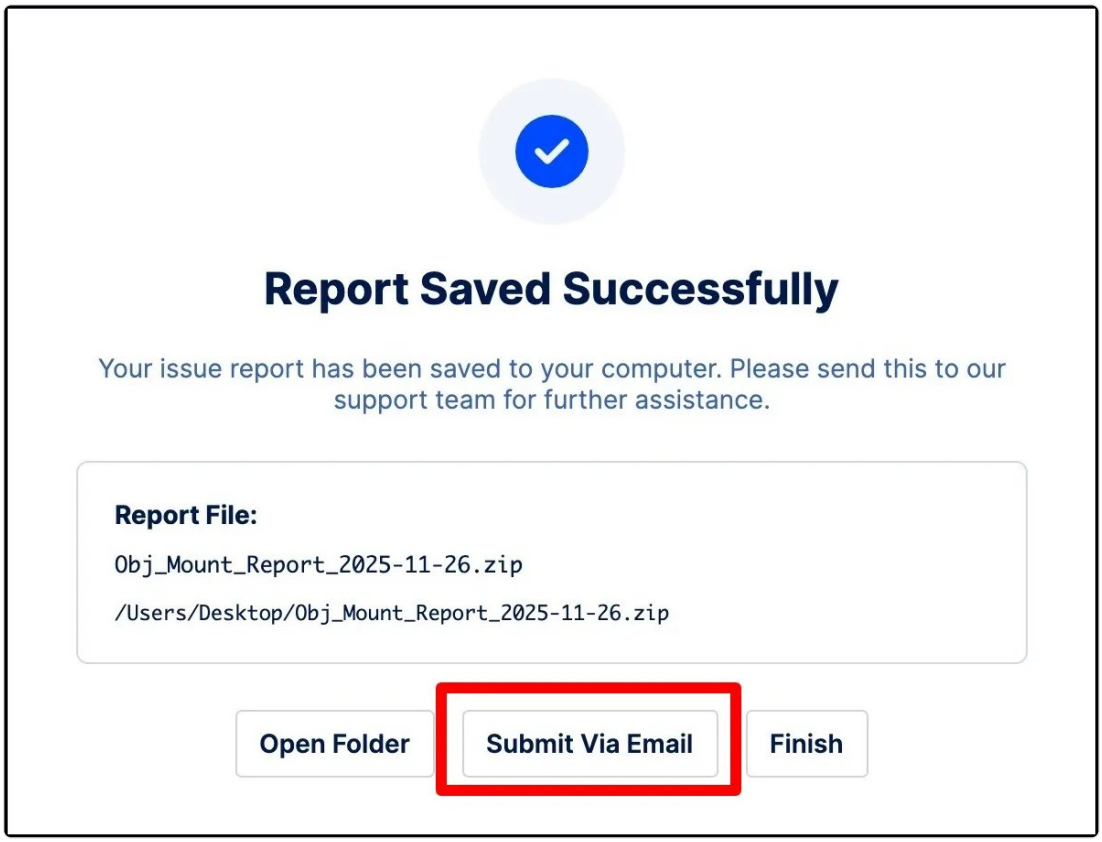
2. Choose the **About** Tab:
   - Click **Report Issue**.
   - Enter a **Title**.
   - Enter details and/or **Steps to Reproduce** the issue.
   - Ensure that **Include my system information** is checked.
   - Click the **Continue** button.



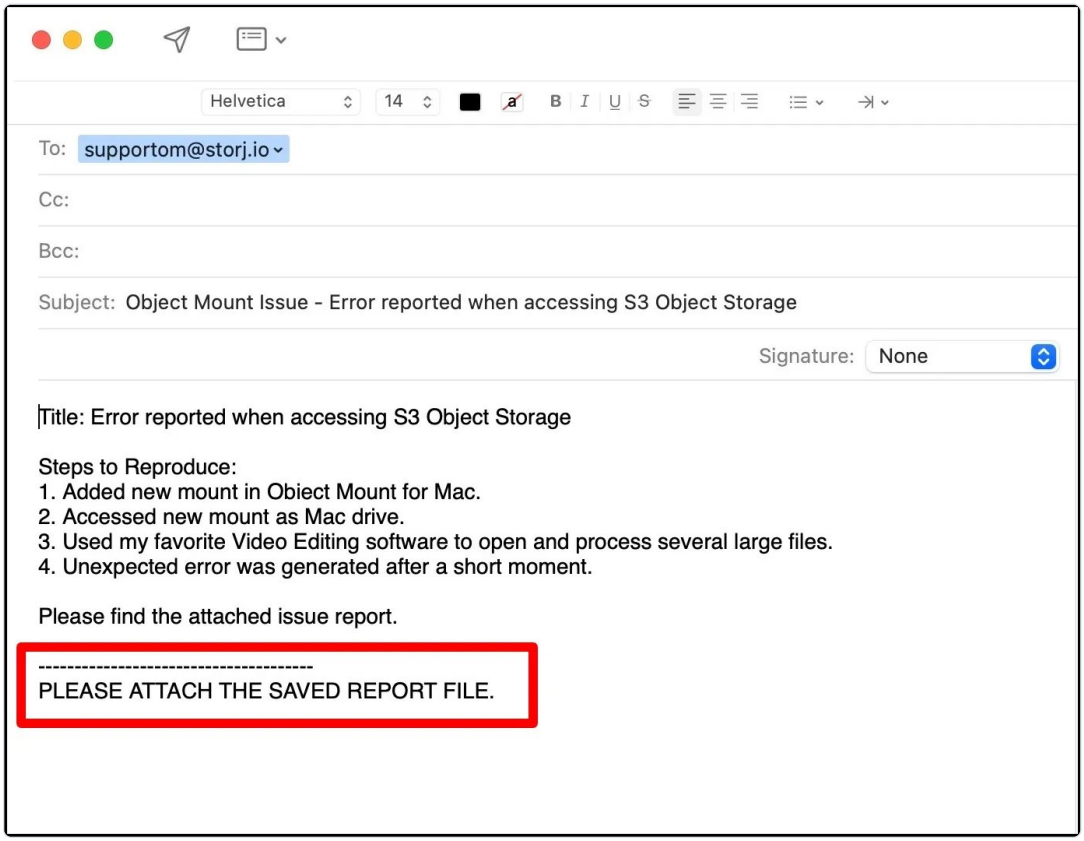
3. Save the Issue Report:
   - Name your `.zip` file.
   - Click **Save** to store a `.zip` file on your local drive.



4. Attach the `.zip` file and send the report:
   - Click the **Submit Via Email** button:



   - Your default Email application will load with content preloaded.
   - **IMPORTANT**: You must *manually* attach the `.zip` file to the email!



    - *Once attached*, click to **Send** the message to the Storj Object Mount support team.

Alternatively, you can open a support ticket and include:

- OS and Object Mount version
- Cloud provider and bucket name
- Screenshots or log excerpts
- Steps to reproduce the issue

FAQ

What S3 permissions are recommended to mount a bucket with Object Mount?

Short answer, for a generic cloud: read, write, and list permissions for the bucket intended to be mounted.

Are full permissions required to achieve maximum performance of Object Mount?

The maximum performance comes from Object Mount's ability to prefetch data based on how it handle certain file types. It is not dependent on being able to write the POSIX-metadata file. With read-only permissions to a bucket, Object Mount will still achieve maximum performance — but it will not be able to update any POSIX-related metadata associated with the original files.

Why don't read-only credentials work when POSIX is enabled in Object Mount?

The POSIX mode needs to write POSIX permissions, ownership and times as objects in a hidden folder inside each directory. This is why the read-only use case works *so long as* you mount the bucket without POSIX enabled.

How can I mount all buckets with Object Mount?

In order for Object Mount to "see" the buckets to be mounted when adding new credentials into Object Mount, the AWS equivalent of `ListAllMyBuckets` is required. In the absence of `ListAllMyBuckets`, you can specify what bucket to mount by manually pairing the bucket.

Where can I find the Object Mount logs?

For Windows, logs are available at `C:\Users\%username%\AppData\Local\Object Mount\cunofs.log`.

For macOS the logs are located at `/Users/$USER/Library/Application Support/Object Mount/cunoFS.log`.

The default logging level is "error", so changing this to "access" or "trace" will provide more verbose logs to better understand what is going on.

Where are the configs and license files located?

For Windows:

- License key information is located along side the logs: `C:\Users\%username%\AppData\Local\Object Mount\license`.
- The `config.json` file is in the same directory where the mount/bucket information is kept

For macOS:

- License key information is located along side the logs: `/Users/$USER/Library/Application Support/Object Mount/license`.
- The `config.json` file is in the same directory where the mount/bucket information is kept

I have multiple Mounts added but I can only activate one at a time?

- Object Mount for Windows and Mac can have several mounts active at the same time.
- Two mounts conflict, and cannot be active together, when their mount paths overlap or their cloud roots overlap (for example `s3://bucket` and `s3://bucket/prefix`). The toggle of a mount that conflicts with an active mount is grayed out.

How can I add a new bucket to the Mount created with the option "All buckets"?

You must add a new pair for the new bucket in the **Credentials** tab of the Object Mount window (Windows, macOS) or see Pairing Containers and Credentials for Linux. After adding the new pair, you must delete the current Mount and create a new one in *All buckets* mode.

Release Notes

Release notes for Object Mount can be found on our GitHub page.