ROSS (Research Object Storage System)

Overview

ROSS (short for Research Object Storage System) is a multi-protocol, web-based storage system, currently based on the Dell EMC Elastic Cloud Storage (ECS) system. The primary 4.2 PB storage unit for ROSS resides on the UAMS campus. In the coming months the UAMS unit will be connected with a 2 PB sister unit sited at the University of Arkansas in Fayetteville, to allow for a limited amount of off-site, replicated storage. ROSS currently has 23 storage nodes, with 15 at UAMS and 8 at UA Fayetteville. Future expansions could increase the number of storage nodes and capacity.

As the name implies, ROSS is an object store. An object store really does not have a directory tree structure like conventional POSIX file systems do. (Grace's storage systems are POSIX file systems.) Instead, objects have names and live in buckets, essentially a one-level directory hierarchy. People often name objects using what looks like a POSIX-style file path. For example "directory1/directory2/directory3/directory4/filename" might be an object name. The RESTful APIs used to access ROSS can do a prefix search, listing just the objects that start with a particular prefix. For example "directory1/directory2/" would pull back all the objects that start with that prefix. So in a way one can mimic a POSIX file system employing an object naming convention that uses the equivalent of POSIX path names. Unfortunately, prefix search is quite inefficient, making the simulated directory lookup slower than a native POSIX file system. Nevertheless, some tools exist that can mimic a POSIX-style file system using an object store behind the scenes. With a proper naming convention, it is fairly easy to backup and restore POSIX directory trees to and from an object store fairly efficiently. This is what makes ROSS an excellent backing storage system for Grace. (A backing storage system is the larger, but slower storage system that back ends a smaller, but faster storage cache in a hierarchical storage model. Grace's storage system should be considered as cache (fast temp storage).)

Remember, the HPC Admins do not back up Grace's shared storage system, since it is only intended to be a staging or scratch area (aka a cache) used to run HPC jobs. In other words, the primary copy of data should be elsewhere, not on Grace. ROSS is one option for keeping the primary copy of data. *In fact, the main reason UAMS purchased ROSS is as the primary location for storing research data.*

Data in ROSS starts as triple replicated on 3 different storage nodes, but then transforms to erasure coding giving resilience without major costs. ROSS currently uses 12/16 error coding, meaning that for every 12 blocks of data stored it actually writes 16 blocks. Up to four blocks could be damaged before ROSS is no longer able to recover data if a fifth block were damaged. In contrast, most RAID systems only have a 1 or 2 drive redundancy. ROSS scatters the 16 blocks across the storage nodes in a data center, improving the performance of data retrievals and further improving the resilience of the system (less chance that a node failure blocks access). Data can be accessed from any of the storage nodes. Currently the two systems (UAMS and UARK) are isolated from each other, not allowing replication. But the plans to join the two are in progress. Eventually all of the content in ROSS, regardless of which campus it physically lives on, would be accessible from either campus. Of course, if data living on one campus is accessed from the other campus, the access will be somewhat slower due to networking delays and bandwidth limitations going between campuses.

As mentioned, soon ROSS will have an option for replicating data in Fayetteville. However, even in this case I would not consider the copy in Fayetteville as a true backup copy. Replication is good for maintaining data that needs high availability and equivalent performance regardless of which campus the data is accessed from. Replication also doubles the storage cost since it consumes available storage capacity at twice the rate that non-replicated storage does. We still recommended that researchers keep backup or archive copies of data somewhere else, even if replication is turned on.

For data that meets OURRstore requirements (relatively static, STEM related, not clinical nor regulated), archiving to OURRstore is a very cost effective option to get resilient, long-term, offsite copies for little extra money (just media and shipping costs). Other options for backup include the UAMS campus Research NAS or cloud storage such as One Drive, Box, Azure, Google, Amazon, or IBM. USB or bare drives are also an option for backup, but not recommended, as they are quite error prone if not stored and managed properly. Some departments might have departmental storage systems that could hold backup copies of data, though the caveat of proper management of those devices still applies.

If you are unfamiliar with the object storage terms that ROSS documents use, we recommend reading the Terminology Used By the Research Object Store System (ROSS) *article on this wiki for an overview.*

Costs for using ROSS

The College of Medicine (COM) Associate Dean of Research, in consultation with an advisory group at UAMS, recommended charging an up-front payment of $70 per TB for the use of ROSS, good for the 5 year life of the storage. This is less than half the current replacement plus maintenance cost for the underlying hardware ($159 per TB). If one wants replicated storage (i.e. one copy in Little Rock, one copy in Fayetteville), the cost would be double, or $140 per TB for the pair of copies.

In comparison, Amazon would charge $276 for Standard S3 (the closest comparable storage class in AWS) over the same 5 year period, plus data access and networking charges. The access and networking fees can be higher than the storage fees, depending on access patterns.

The oversight of ROSS is moving from the UAMS COM to the Arkansas Research Computing Cooperative (ARCC), formed by a memorandum of understanding between UAMS and UA Fayetteville. ARCC may revisit the charge for using ROSS. One option being considered is a free tier, with yearly charges for any use over the free tier by individuals or groups (labs, departments, research projects, etc.). Also being considered is whether payment would be 'up front' for the expected life of the storage, or would be simply billed periodically (e.g. annually) as the storage is used.

> *Until pricing changes, users should be prepared for the up front $70 per TB per copy pricing set by the COM. Although we are not collecting fees at the moment, retroactive fee collection could begin after the ARCC steering committee settles on the fee schedule. Any storage in use at the time that ARCC sets the fee schedule would be charged the lower of the $70 fee imposed by COM, or the new fees set by ARCC, for 5 years of storage. Additional storage would be charged at the fee schedule set by ARCC.*

Restrictions on data that can be stored on ROSS

There are few restrictions on the types of data that can be stored on ROSS. Almost anything is allowed, as long as the data storage complies with both governmental, IRB, and UAMS rules and regulations.

Keep in mind that any campuses that are participants in ARCC, and by extension, the Arkansas Research Platform (ARP), have access to ROSS. Unlike the UAMS Research NAS, which is locked down behind UAMS firewalls hence only accessible inside the UAMS Campus, ROSS is located in the ARCC Science DMZ, a private network only accessible by a limited number of campuses, both within Arkansas and potentially beyond. As such, it is inappropriate to store unecrypted in ROSS fully identified patient (PHI) or personal (PII) data, as it could be a violation of UAMS HIPAA or FERPA policies. ROSS does have the ability to restrict access to buckets and to do server-side, data-at-rest encryption, but these capabilities have not been evaluated as to whether or not they are sufficient for HIPAA or FERPA compliance. For now, ROSS should not be used for data that is regulated by HIPAA, FERPA, or any other governmental regulation. De-identified human subject data is allowed.

In addition, by using ROSS, you agree that you are complying with any rules or restrictions place by your IRB protocol on data you store in ROSS.

> *Do not use ROSS for regulated data (e.g. data that must comply with HIPAA, FERPA, or other rules). If your data is sensitive, please turn on server side encryption. You are responsible for complying with any applicable IRB protocol.*

The server-side encryption can either be turned on at the namespace level, where all buckets in a namespace are required to be encrypted, or on the bucket level for namespaces that do not have 'encryption required' turned on. If you want namespace level encryption, please inform the HPC admins when requesting a namespace. The encryption choice must be made when the namespace or bucket created and cannot be changed afterwards. (One can copy data from an unencrypted bucket to an encrypted one, then destroy the unencrypted bucket, or visa versa, should a change be needed after bucket creation.)

Preparing to Use ROSS

> *Before requesting access to ROSS, in addition to reading this article, please also read the* Terminology Used By the Research Object Store System (ROSS) *article on this wiki, since it describes key concepts needed to understand how storage is laid out and accessed.*

If you are are good with the data restrictions and prepared to cover costs that might be incurred for using ROSS, there are steps that must be completed before you can actually move data between ROSS and Grace. First, you must request or be assigned to a namespace. Second, decide which APIs you might use. Third, your namespace administrator using the ECS Portal may wish to pre-create the buckets that you might use. Creating buckets in the ECS Portal can be a more convenient than creating them using APIs or tools. Finally, get access credentials for the APIs that you might use.

Requesting a namespace assignment on ROSS

All users of ARP facilities, including Grace, may ask for credentials in the "arp" namespace. When the administrators create your credentials, they will also create a bucket in that namespace for your use. The bucket will have the same name as your home directory on Grace or Pinnacle, which should be the same as your username. After creating your account, the administrators will place your S3 secret key into the file named ".ross/s3key" in your home directory. Your S3 ID is your ROSS object user name in the arp namespace, which is "<username>@arp", where <username> is your login name for Grace or Pinnacle.

If your lab, department, project, or group would like to purchase their own namespace for their exclusive use, please contact us at HPCAdmin@uams.edu. All namespaces have namespace administrators who manage buckets and object users within that namespace. A particular username@domain can only be a namespace administrator for one namespace. If a particular person needs to be the namespace administrator in more than one namespace, for example namespaces for two different groups, they must use different login names plus domains for each namespace they wish to administer. This is why we suggest always qualifying namespace users (object or administrative) with the "@<namespace>" suffix, where <namespace> is the name of the namespace that the user is assigned to.

Don't forget that a namespace administrator is not the same as an object user. See Terminology Used By the Research Object Store System (ROSS) for the difference between a namespace administrator and an object user. An object user has a different API-specific set of credentials.

The steps to gain access to a namespace differ, depending on whether the namespace already exists or not.

New namespaces

To initiate access to ROSS for a *new* personal or group (e.g. project, lab, department) namespace, please send a request via e-mail to HPCAdmin@uams.edu. In your request,

- Please indicate whether this is for a personal namespace (e.g. primary storage for processing data on Grace), or for a group (shared storage).
    - For a personal namespace, please indicate
        - your name
        - your e-mail
        - your departmental affiliation
        - what your login name (not your password) and domain you will use to access ROSS's administrative interface, e.g. johndoe@hpc.uams.edu (for personal namespaces we prefer that you use your HPC username)
        - why you do not wish to be part of the "arp" namespace (where most personal accounts go)
    - For a group namespace, please give
        - a name and brief description for the group
        - the primary e-mail contact for the group
        - the departmental affiliation of the group
        - who will be the namespace administrators - we need
            - their names
            - their e-mail address
            - their login name (not their password) and domain e.g. janedoe@ad.uams.edu (for group namespaces we generally prefer campus Active Directory usernames (e.g. the name the namespace administrator might use to login to Outlook Mail))
            - you may ask for more than one namespace administrator
            - if all the members of a particular campus AD group should be namespace administrators, you could also just give us the name and domain of the group instead of their individual names
- Please estimate approximately how much storage you or your group intend to use in ROSS for the requested namepace, divided into local and replicated amounts. Remember that replicated data costs twice as much as non-replicated data. The HPC Admins will use this information in setting the initial quotas and for capacity planning. You will be allowed to request increases in quota if needed and space is available.
- We would also appreciate a brief description of what you will be using the storage for. The "what it is used for" assists us in drumming up support (and possibly dollars) for expanding the system.

Existing namespace

If you wish to access an existing group namespace as an object user, please contact the namespace administrator of that namespace and ask to be added as an object user for that namespace. For the "arp" namespace, please contact "HPCAdmin@uams.edu". If you need assistance determining what namespaces are available and who are the namespace administrators feel free to contact the HPC Admins via e-mail (hpcadmin@uams.edu).

If you supervised an employee or student who is no longer with you, for example due to a change of assignment or leaving the university, then you are responsible for the final disposition of the storage used by that person. In this case you may request access to their personal namespace by sending that request to the HPC Admins via e-mail (hpcadmin@uams.edu). If the person is still with UAMS, the HPC Admins will coordinate with the two of you, in case the person wishes keep their namespace, including portions of the existing data.

APIs supported by ROSS

ROSS supports multiple object protocol APIs. The most common is S3, followed by Swift (from the Open Stack family). Although ROSS can support the NFS protocol, our experiments show that the native NFS support is slower for many use cases than other options that simply emulate a POSIX file system using object protocols. But NFS is available for those who would like to use it. ROSS can also support HDFS (used in Hadoop clusters) and a couple of Dell/EMC proprietary protocols (ATMOS and CAS); however, we have not tested any of these other protocols, hence the HPC Admins could only offer limited support for these other options.

The primary protocols that the HPC Admins support is S3 and Swift. We have more experience with S3, hence most of this article assumes S3. Note that S3 and Swift can be used interchangeably and simultaneously on the same bucket.

The file-based protocols NFS and HDFS must be enabled on a bucket at creation time, if you intend to use them, since the underlying system adds additional metadata when file access is enabled. When enabled, NFS and HDFS, too, can be used interchangeably with the S3 and Swift object protocols on the same bucket. However, a bucket with file access enabled loses some of the object features, such as life cycle management. So consider carefully before enabling file access. (You can still access a bucket using file system semantics with some external tools even when the bucket is not enabled for file access.)

Creating a bucket

There are three ways to create buckets (aka containers in the Swift world):

1. Many object tools have options to create and manipulate buckets, which is quite convenient, generally portable (i.e. would work on any object storage system), but also limited. In general, tools cannot create buckets with ECS options, such as encryption, replication and file access. You have to use one of the other bucket creation options. Object tools only need the object user's access credentials. Please see the tools' documentation for details.
2. The RESTful APIs, including S3, Swift, and an ECS-specific management API, can be used to create and manipulate buckets. With the appropriate headers and parameters, ECS options can be enabled. More information can be had in the ECS Data Access Guide and the ECS API Reference (hint - use the search function).
    a. The protocol-specific S3 or Swift REST APIs need an object user's credentials, and the namespace administrator must have given appropriate permissions to the object user.
    b. The ECS-specific management APIs (which we do not support users in using) require namespace administrator credentials.
3. Use the ECS Portal. This is the simplest option that gives full control over bucket characteristics. The ECS Portal can only be accessed by the namespace administrator. (That would be you, for your personal namespace.) The namespace administrator logs into the ECS Portal (https://ros

s.hpc.uams.edu) with the credentials tied to that namespace. Details on how to use the ECS Portal to manage buckets can be found in the Buckets chapter of the ECS Administration Guide.

Getting access credentials

In order to use either S3 or Swift, you need credentials.

The credentials are tied to a particular object user, who belongs in a particular namespace. For personal namespaces, the HPC Admins already added the owner as an object user as well as the namespace administrator. For group namespaces, the namespace administrators may add object users to the namespace using the ECS Portal following the instructions in the "Add an Object User" section of the ECS Administration Guide. Note that the object user names are not necessarily tied to any domain, though sometimes people add a domain-like suffix to differentiate object user names. Although an object user name is only tied to one namespace, they need to be unique throughout all namespaces within ROSS. A domain-like suffix linked to the namespace (e.g. "@<namespace-name>") can help insure uniqueness.

The namespace administrator can retrieve an object user's credentials by logging into the ECS Portal at https://ross.hcp.uams.edu using the credentials that are tied to the namepace for the namespace administrator. Remember, the owner of a personal namespace is their own namespace administrator, and typically logs in with their HPC credentials in the form of <user>@hpc.uams.edu. Once in the portal, click on "Manage" in the left side menu, then click on "Users". Make certain that the "Object Users" tab is highlighted. The type into the "Search" box the object user name for whom you wish to retrieve credentials and click the search icon (a magnifying glass). Find the line with the correct object user's name, and the correct namespace name. In the "Actions" column click the "Edit" button associated with the name. This should bring you to the object user's profile.

For S3 credentials, click the "Show Secret Key" checkbox. This should allow you to see, as well as copy to the clipboard, the S3 Secret Access Key. The object user's name, visible at the top of the profile page, serves as the S3 Access Key ID.

For Swift credentials, Note the "Swift Group" and "Swift Password" fields. If this is a new object users, you may need to fill in the group and password for Swift. Note that we do not currently run a keystone server, hence only Swift v1 or v2 credentials work.

Using ROSS

After getting access credentials for your chosen APIs, you are ready to use ROSS.

In experiments we have notices that write times to ROSS are considerably slower than read times, and slower than many POSIX file systems. However, read times are significantly faster than write times on ROSS. In other words, it takes longer to store new data into ROSS than to pull existing data out of ROSS. We also notice (as is typically of most file systems) that transfers of large objects can go significantly faster than tiny objects. Please keep these facts in mind when planning your use of ROSS - ROSS favors reads over writes and big things over little things.

In theory, a bucket can hold millions of files. We have noticed, though, that with path prefix searching the more objects that have a particular prefix, the longer the search takes. While this is not dissimilar to POSIX filesystems, where the more files there are in a directory. the longer it takes to do a directory lookup, on an object store, being a flat namespace (no directory hierarchy), such searches can be much slower than the typical POSIX file system. The total number of objects in a bucket also has a mild, though not dramatic, impact on the look up speed particular objects in a bucket. Again, keep this in mind when planning your use of ROSS. One way to get around this speed penalty is to use an external database (e.g. sqlite) to track in which buckets and objects your data resides in.

Tools recommendations for accessing ROSS

The bottom line - any tool that uses any of the supported protocols theoretically could work with ROSS. There are a number of free and commercial tools that work nicely with object stores like ROSS. Like most things in life, there are pros and cons to different tools. Here is some guidance as to what we look for in tools.

Being inherently parallel accessible (remember the 23+ storage nodes), the best performance is gained when operations proceed in parallel. Keep that in mind if building your own scripts (e.g. using ECS's variant of s3curl) or when comparing tools. More parallelism (i.e. multiple transfers happening simultaneously across multiple storage nodes) generally yields better performance, up to a limit. Eventually the parallel transfers become limited by other factors such as memory or network bandwidth constraints. Generally there is a 'sweet spot' in the number of parallel transfer threads that can run before stepping on each other's toes.

The S3 protocol includes support for multipart upload of objects, where a large object upload can be broken into multiple pieces and transferred in parallel. Tools that support multipart upload likely will have better performance than tools that don't. In addition, tools that support moving entire directory trees and that work in parallel will have better performance than tools that move files one object at a time.

After evaluating several tools, the HPC admins settled on 2 tools, ecs-sync and rclone, as 'best of breed' for moving data between ROSS and Grace's cluster storage system, where the /home, /scratch, and /storage directory live. The ecs-sync program is the more efficient and the speedier of the two for bulk data moves. It consumes fewer compute resources and less memory than rclone. When properly tweaked for number of threads (i.e. when the sweet spot is found) it moves data significantly faster than rclone. The rclone program has more features than ecs-sync, including ways to browse data in ROSS, to mount a ROSS bucket as if it were a POSIX file system, and to synchronize content using a familiar rsync-like command syntax. While ecs-sync is great for fast, bulk moves, rclone works very well for nuanced access to ROSS and small transfers. The rclone program also works quite nicely for moving data between a workstation or laptop and ROSS.

The following sections describe using ecs-sync and rclone for moving data between ROSS and Grace's cluster storage.

ecs-sync

The ecs-sync program is specifically designed for the parallel bulk moving of data from one storage technology to another. It comes from the Dell/EMC support labs, and is what EMC support engineers use for migrating data. It certainly is possible to install ecs-sync outside of Grace, for example on a lab workstation to rapidly move data to or from ROSS. However in this article we only discuss the use case of moving data between ROSS and Grace using the ecs-sync installed by the HPC Admins.

Due to security issues, we do not offer the job-oriented service mode described in the ecs-sync documentation. (If the ecs-sync maintainers ever fixed the security holes, we could reconsider.) Instead we only support running ecs-sync in what its documentation calls "Alternate (legacy) CLI execution", where a user runs ecs-sync as a command, instead of queuing up a job. This allows the command to be run in the context of the user's login, honoring permissions. One side effect is that the copied data could end up being owned by the user running the command, regardless of who owned the source data, which might not be desirable for all cases.

A command alias exists on Grace's login and compute nodes for running ecs-sync. The alias actually runs the command on a data transfer node, so does not bog down the node on which the command is issued. In other words, feel free to use ecs-sync on the login node, or from the command prompt in Grace's Open OnDemand portal. With this alias, there is no need to run ecs-sync from a compute job, as it brings no advantage.. The copying is being done from a Data Transfer Node, with little to no impact on the local node.

The syntax for calling ecs-sync interactively on Grace is:

Interactive ecs-sync command

```
ecs-sync --xml-config <config-file>.xml
```

The <config-file>.xml are the set of instructions of what is to move where in how many threads using which storage nodes. Of course, replace <config-file> with a name of your choosing. We will describe the XML config contents content later.

In running interactively, you see all the messages coming back from ecs-sync as it does its job, giving instant feedback. But if the shell dies, the command may stop. To avoid this, you can use nohup to run ecs-sync as a background job that continues when you log out and redirect standard out/err into a file, for example by using the following syntax:

Running ecs-sync in the background

```
nohup ecs-sync --xml-config <config-file>.xml > <log-file>.log &
```

Again, replace <config-file> and <log-file> with any name you wish.

The most complicated task in using ecs-sync is setting up the xml config file. The config file consists of three parts:

- the global settings define various parameters of how the sync should be run, such as buffer sizes, number of retries, number of parallel threads, etc. that are not specific to either the source or the target
- the source settings define where the data to be copied is coming from, including the type of storage, the location of the data, and access information
- the target settings defing where the data being copied should be stored, including the type of storage, the destination bucket or directory, and other access related information

The basic layout for the xml config file is as follows:

Overview of config file

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<syncConfig xmlns="http://www.emc.com/ecs/sync/model">
    <options>
        <!-- ... the global settings ... -->
    </options>

    <source>
        <filesystemConfig>
            <!-- ... file copy settings ... -->
        </filesystemConfig>
    </source>

    <target>
        <ecsS3Config>
            <!-- ... ecs s3 copy settings ... -->
        </ecsS3Config>
    </target>
</syncConfig>
```

Note that the contents of <source> and <target> are interchangeable, and the order of the blocks doesn't matter. So the direction of the copy can be switched by simply swapping the <source> and <target> keywords.

Although ecs-sync knows abaoaut other types of storage, since this guide is for moving data between Grace's storage and ROSS, we will only describe the filesystem (for Grace's storage) and ecsS3 (for ROSS's storage) types, since those are the 2 needed. If you are interested in other storage types, please consult the ecs-sync documentation.

The easiest way to create an xml config file is to copy an exiting template, and just change the pertinent values. To help guide, here are examples of the 3 sections with descriptions of the parameters

Global Options

This example shows available global parameter options. These are typical values, and may be changed. The important options are deleteSource, fourceSync, recursive, retryAttempts, threadcount, verify and verifyOnly,

Global Options

```
<options>
    <bufferSize>524288</bufferSize>
    <!-- <dbConnectString>dbConnectString</dbConnectString>
    <dbEncPassword>ChangeMe1234</dbEncPassword> -->
    <dbFile>/home/YourHomeDirectory/ecs-dbFile</dbFile>
    <dbTable>ecs_sync_log</dbTable>
    <deleteSource>false</deleteSource>
    <estimationEnabled>true</estimationEnabled>
    <forceSync>false</forceSync>
    <ignoreInvalidAcls>false</ignoreInvalidAcls>
    <logLevel>quiet</logLevel>
    <monitorPerformance>true</monitorPerformance>
    <recursive>true</recursive>
    <rememberFailed>false</rememberFailed>
    <retryAttempts>2</retryAttempts>
    <!-- <sourceListFile>sourceListFile</sourceListFile> -->
    <syncAcl>false</syncAcl>
    <syncData>true</syncData>
    <syncMetadata>true</syncMetadata>
    <syncRetentionExpiration>false</syncRetentionExpiration>
    <threadCount>24</threadCount>
    <timingWindow>1000</timingWindow>
    <timingsEnabled>false</timingsEnabled>
    <verify>false</verify>
    <verifyOnly>false</verifyOnly>
</options>
```

The following table describes all the global options.

Option	Description
bufferSize	Sets the buffer size (in bytes) to use when streaming data from the source to the target (supported plugins only). Defaults to 128K
dbConnectString	*We do not use this parameter.* Enables the MySQL database engine and specifies the JDBC connect string to connect to the database (i.e. "jdbc:mysql://localhost:3306/ecs_sync?user=foo&password=bar"). A database will make repeat runs and incrementals more efficient. With this database type, you can use the mysql client to interrogate the details of all objects in the sync.
dbEncPassword	*This parameter is only used with the MySQL option, which we do not use.* Specifies the encrypted password for the MySQL database.
dbEnhancedDetailsEnabled	*May not work.* Specifies whether the DB should included enhanced details, like source/target MD5 checksum, retention durations, etc. Note this will cause the DB to consume more storage and may add some latency to each copy operation.
dbFile	*We do recommend this option.* Enables the Sqlite database engine and specifies the file to hold the status database. A database will make repeat runs and incrementals more efficient. With this database type, you can use the sqlite3 client to interrogate the details of all objects in the sync. Be certain to replace the name with an absolute path to your home directory so that ecs-sync can consistently find it. One potential problem - if a transfers has errors, the system might not retry them. A recourse is to delete the db file, but of course you will lose your history.
dbTable	Specifies the DB table name to use. When using MySQL be sure to provide a unique table name or risk corrupting a previously used table. Default table is "objects".

deleteSource	Supported source plugins will delete each source object once it is successfully synced (does not include directories). Use this option with care! Be sure log levels are appropriate to capture transferred (source deleted) objects.
estimationEnabled	By default, the source plugin will query the source storage to crawl and estimate the total amount of data to be transferred. Use this option to disable estimation (i.e. for performance improvement).
forceSync	Force the write of each object, regardless of its state in the target storage.
ignoreInvalidAcls	If syncing ACL information when syncing objects, ignore any invalid entries (i.e. permissions or identities that don't exist in the target system).
logLevel	Sets the verbosity of logging (silent|quiet|verbose|debug). Default is quiet.
monitorPerformance	Enables performance monitoring for reads and writes on any plugin that supports it.
perfReportSeconds	*Not tested.* Report upload and download rates for the source and target plugins every <x> seconds to INFO logging. Default is off (0).
recursive	Hierarchical storage will sync recursively.
rememberFailed	Tracks all failed objects and displays a summary of failures when finished
retryAttempts	Specifies how many times each object should be retried after an error. Default is 2 retries (total of 3 attempts).
sourceListFile	Path to a file that supplies the list of source objects to sync. This file must be in CSV format, with one object per line and the absolute identifier (full path or key) is the first value in each line. This entire line is available to each plugin as a raw string.
sourceListFileRawValues	Whether to treat the lines in the sourceListFile as raw values (do not do any parsing to remove comments, escapes, or trim white space). Default is false.
syncAcl	Sync ACL information when syncing objects (in supported plugins).
syncData	Sync object data.
syncMetadata	Sync metadata.
syncRetentionExpiration	Sync retention/expiration information when syncing objects (in supported plugins). The target plugin will *attempt* to replicate retention /expiration for each object. Works only on plugins that support retention/expiration. If the target is an Atmos cloud, the target policy must enable retention/expiration immediately for this to work.
threadCount	Specifies the number of objects to sync simultaneously. Default is 16.
timingWindow	Sets the window for timing statistics. Every {timingWindow} objects that are synced, timing statistics are logged and reset. Default is 10,000 objects.
timingsEnabled	Enables operation timings on all plug-ins that support it.
verify	After a successful object transfer, the object will be read back from the target system and its MD5 checksum will be compared with that of the source object (generated during transfer). This only compares object data (metadata is not compared) and does not include directories.
verifyOnly	Similar to verify except that the object transfer is skipped and only read operations are performed (no data is written).

File System Options

This example shows the file system as a source. It could also be a target. The filesystem storage type plugin reads and writes to or from a directory. The example shows typical values, which generally need to change to fit your situation.

File System as Soure

```
<source>
    <filesystemConfig>
        <path>/home/user/the/subdirectory/to/copy</path>
        <followLinks>true</followLinks>
        <storeMetadata>false</storeMetadata>
    </filesystemConfig>
</source>
```

The following table describes the file system options.

Option	Description
deleteCheckScript	When the deleteSource global option is true, add this option to execute an external script to check whether a file should be deleted. If the process exits with return code zero, the file is safe to delete.
deleteOlderThan	When the deleteSource global option is true, add this option to only delete files that have been modified more than <delete-age> milliseconds ago.
excluded-paths	A list of regular expressions to search against the full file path. If the path matches, the file will be skipped. Since this is a regular expression, take care to escape special characters. For example, to exclude all .snapshot directories, the pattern would be .*/\.snapshot. Specify multiple entries by repeating the option or using multiple lines.
followLinks	Instead of preserving symbolic links, follow them and sync the actual files.
includeBaseDir	By default, the base directory is not included as part of the sync (only its children are). enable this to sync the base directory.
modifiedSince	Only look at files that have been modified since the specifiec date/time. Date/time should be provided in ISO-8601 UTC format (i.e. 2015-01-01T04:30:00Z, which is <yyyy-MM-ddThh:mm:ssZ>).
path	The directory tree or file that you would like to copy to/from. If the source path points to a single file or object instead of a directory, the destination path must also point to a single file or object. If the source path points to a directory, the directory name will not be included in the destination unless "includeBaseDir" is enabled.
relativeLinkTargets	By default, any symbolic link targets that point to an absolute path within the primary source directory will be changed to a (more portable) relative path. Set this option false to keep the target path as-is.
storeMetadata	When used as a target, stores source metadata in a json file, since filesystems have no concept of user metadata. When used as a source, uses the json file to restore the metadata in the target.
useAbsolutePath	When true, uses the absolute path to the file when storing it instead of the relative path from the source dir.

Ross (ECS S3) Options

ROSS uses the ECS S3 storage system type. This example shows ROSS as a destination. It could also be a target. The example shows typical values, some of which must be changed for the sync to work, such as accessKey, bucketName, keyPrefix, and secretKey.

ROSS ecs S3 as Target

```
    <target>
        <ecsS3Config>
            <accessKey>accessKey</accessKey>
            <secretKey>secretKey</secretKey>
            <port>9020</port>
            <protocol>http</protocol>
            <bucketName>bucketName</bucketName>
            <createBucket>false</createBucket>
            <enableVHosts>false</enableVHosts>
            <keyPrefix>keyPrefix</keyPrefix>
            <apacheClientEnabled>false</apacheClientEnabled>
            <decodeKeys>false</decodeKeys>
            <geoPinningEnabled>false</geoPinningEnabled>
            <!-- <host>host</host> -->
            <includeVersions>false</includeVersions>
            <mpuEnabled>true</mpuEnabled>
            <mpuPartSizeMb>128</mpuPartSizeMb>
            <mpuThreadCount>4</mpuThreadCount>
            <mpuThresholdMb>512</mpuThresholdMb>
            <preserveDirectories>true</preserveDirectories>
            <remoteCopy>false</remoteCopy>
            <resetInvalidContentType>true</resetInvalidContentType>
            <smartClientEnabled>true</smartClientEnabled>
            <socketConnectTimeoutMs>15000</socketConnectTimeoutMs>
            <socketReadTimeoutMs>60000</socketReadTimeoutMs>
            <urlEncodeKeys>false</urlEncodeKeys>
            <vdcs>uams(ross01.hpc.uams.edu,ross02.hpc.uams.edu,ross03.hpc.uams.edu,ross04.hpc.uams.edu,ross05.
hpc.uams.edu,ross06.hpc.uams.edu,ross07.hpc.uams.edu,ross08.hpc.uams.edu,ross09.hpc.uams.edu,ross10.hpc.uams.
edu,ross11.hpc.uams.edu,ross12.hpc.uams.edu,ross13.hpc.uams.edu,ross14.hpc.uams.edu,ross15.hpc.uams.edu)<
/vdcs>
        </ecsS3Config>
     </target>
```

The following table describes the options for the ECS S3 (ecs-s3:) plugin.

Option	Description
accessKey	The S3 access ID for ROSS's S3 API, which is the object user's username in the ECS world.
secretKey	The S3 access secret for ROSS associated with the object user, which can be copied from the ECS portal by the namespace administrator.
port	The port to use to access ROSS. Should 9020 if protocol is set to http, or 9021 if the protocol is set to https.
protocol	The protocol to use to access ROSS, either http or https.
bucket	The bucket to write or read objects from.
create-bucket	By default, the target bucket must exist. If true, this option will create the bucket if it does not exists.
enableVHosts	Specifies whether virtual hosted buckets will be used (i.e. bucket name in the hostname instead of in the path) (default is path-style buckets),
keyPrefix	Specifies a string to be prepended to the name generated for the object. For example, if keyPrefix is set to "prefix/", the source is a file system, and ecs-sync is copying a file with a path "subdir/subdir/filename", then the object name will be "prefix/subdir/subdir/filename".
apacheClientEnabled	Disabling this will use the native Java HTTP protocol handler, which can be faster in some situations, but is buggy
geoPinningEnabled	Enables geo-pinning. This will use a standard algorithm to select a consistent VDC for each object key or bucket name, taking into account where the request is made, and where the VDCs that hold the data are.
includeVersions	Enable to transfer all versions of every object. NOTE: this will overwrite all versions of each source key in the target system if any exist!

mpuEnabled	Enables multi-part upload (MPU). Large files will be split into multiple streams and (if possible) sent in parallel.
mpuPartSizeMb	Sets the part size to use when multipart upload is required (objects over 5GB). Default is 128MB, minimum is 4MB,
mpuThreadCount	The number of threads to use for multipart upload (only applicable for file sources).
mpuThresholdMb	Sets the size threshold (in MB) when an upload shall become a multipart upload.
preserveDirectories	If enabled, directories are stored in S3 as empty objects to preserve empty dirs and metadata from the source.
remoteCopy	If enabled, a remote-copy command is issued instead of streaming the data. Remote-copy can be much faster than the streaming alternative. Remote-copy can only be used when the source and target is the same system (e.g. both are on ROSS).
resetInvalidContentType	When set to true (the default), any invalid content-type is reset to the default (application/octet-stream). Turn this off to fail these objects (ECS does not allow invalid content-types).
smartClientEnabled	The smart client is enabled by default. Use this option to turn it off when using a load balancer (which presumably would perform a function similar to smart client) or a fixed set of nodes.
socketConnectTimeoutMs	Sets the connection timeout in milliseconds (default is 15000ms).
socketReadTimeoutMs	Sets the read timeout in milliseconds (default is 0ms).
urlEncodeKeys	Enables URL-encoding of object keys in bucket listings. Use this if a source bucket has illegal XML characters in key names.
vdcs	Sets which Virtual Data Center and which nodes in that virtual data center that ecs-sync sync will communicate with in carrying out the copying. The format is "vdc-name(host,..)[,vdc-name(host,..)][,..]", The smart client capability will load balance across the active nodes, skipping the inactive ones. We use this in favor of the hosts option. We use this option above, listing the current UAMS vdc and nodes.
host	This is an alternative to the vdcs command, where hosts can be comma separated list of host network names, or can be the name of a load balancer. *We do not provide an example of how to use the host option, as we prefer the vdcs option.*

rclone

Although not as fast nor efficient as ecs-sync, the rclone program (https://rclone.org/) is one of the most versatile tools for accessing object stores such as ROSS. It includes features for browsing various types of object and cloud storage systems, as well as local files using Posix file system conventions, using commands such as ls (list objects/files in a path), lsd (list directories/containers/buckets in a path), copy, move, delete, etc. For some object/cloud storage systems, rclone can mount buckets as if they were a network file systems, though with reduced functionality and speed, so that users can use familiar commands to access bucket contents instead of the rclone commands or object API. But the most popular feature of rclone is the ability to sync a directory tree to a bucket on an object store using a familiar rsync-like syntax. The rclone site includes documentation on how to install, configure, and use rclone. The rclone program is licensed under the permissive, open-source MIT license, hence is free to use and distribute.