EGEEKCONSULTING

Making the improbable easy

How To Make HSM Viable in a Hybrid Multi-Cloud World

What is Hierarchical Storage Management (HSM) and why does it matter? The short version of HSM is that it is a descriptor for a series of technologies and techniques aimed at reducing the cost of storage by moving infrequently accessed data to less expensive storage solutions. The long version is quite a bit longer.

HSM has been around for at least as long as humans have been recording information. Regardless of the data to be stored, or the storage mediums available, there have always been more expensive and less expensive ways to store data.

Our ancestors preserved mythology and history for millennia by carving that data into stone, pouring metals into molds, or even building grand monuments. This data was important enough for them to use the expensive storage solutions to record it, and preserve it for as long as possible.

It is rather more rare, however, for modern archeologists to find someone's shopping list or stock inventory preserved in such a manner. Occasionally, a shopping list (or similar low-value data item) does turn up at a dig, but these are rarely stored using the same methods and technologies as grand tales and mythos. Even at the dawn of recorded history, different data had different values, and was stored using different storage solutions.

Today, organizations do a little less carving epics onto stone tablets and rather a lot more preserving every conceivable scrap of digital data they produce. Regulatory requirements dictate minimum data preservation times, and combine with a human nature that makes us collectively reluctant to clean our digital rooms.

In a perfect world, we'd keep all of our data in a single storage solution, with a single namespace, on the most performant storage available. In the real world, however, those all-flash arrays cost rather a lot; the shopping lists really do need to be stored somewhere else.

The above is the theory underpinning is hierarchical storage management. As one might imagine, there are numerous ways to approach it.

Nomenclature

HSM is as much a discussion about nomenclature as it is about techniques and technologies. When talking about block storage solutions that migrate infrequently accessed (cold)data blocks to slower (and thus less expensive) media, we call these hybrid storage solutions, and talk about block-level tiering. This sort of tiering occurs entirely within the storage solution itself, and is transparent to both workloads and end users.

Tiering can also refer to the process of moving unstructured data (files or objects) from one storage device to another, instead of simply between different classes of media within a single storage solution. Here, the tiering is usually not transparent: workloads and users would have to look for cold data in a different location.

When an unstructured data storage solution can provide access to the cold data in a transparent manner, this is traditionally referred to as engaging in HSM rather than tiering. An HSM solution can – and usually does – tier data, however, the primary function of the HSM is to present that data to the user as though all data was stored in one location. This is significantly different from more primitive archival approaches to tiering.

This being a discussion about nomenclature, one should take the above with a grain of salt. There is usually debate about the meaning of various IT terms, but for the purposes of this piece, we'll stick to the definition we have.

HSM solutions have been around for decades, and were once heavily used to migrate cold files to and from tape storage solutions, leaving the far more expensive hard disks available for more frequently accessed files. Today, HSM solutions are seeing a resurgence of interest as organizations embrace multi-cloud storage, and look to not only tier their data, but seek to unify data stored upon multiple infrastructures under a single namespace.

There exist three primary approaches to making data stored on multiple devices appear as though it all exists in one place: shortcuts, symlinks and stubs.

Shortcuts

The simplest form of HSM is the shortcut. A shortcut is a small file that contains a reference to another file or storage location. Shortcuts have a number of drawbacks, and are not a good technology to use for large scale HSM.

Shortcuts are specific to the Operating System Environment (OSE) that created them. For example, a computer running Windows will not be able to understand shortcuts created by a computer running MacOS.

Applications may not support shortcuts. Consider an attempt to save a file. When using the "save file" dialogue box in an applications, users typically browse to where they would like to save their file. It is not uncommon for an application to attempt to interpret a double click on a shortcut as an indication that the application should save a file as that file name, instead of interpreting it as an indication that the application should traverse the shortcut and attempt to save the file in the location indicated by the shortcut, or as the file the shortcut might link to.

Shortcuts are primitive constructs. They don't contain much in the way of metadata, and they don't preserve any authentication structures. A shortcut is just a pointer: keeping track of what the data it points to is, and who is allowed to use it is up to the client OSE.

This can lead to a number of security, auditing, and regulatory compliance problems. Complying with the European Union's General Data Protection Regulation (GDPR) provides a good thought exercise.

The GDPR enshrines the principle of least privilege in law: no individual or computer system should access (or have access to) data they don't absolutely need. Proving that this has been implemented requires tracking every access to a given piece of data - read, write, modify and delete - over the entire lifetime of that file.

Client systems can't be relied upon to keep records of these sorts of accesses: they are almost certain to be wiped or replaced long before other regulatory regimes allow deletion of that data. This means that data access needs to be recorded at the server; however, shortcuts don't funnel all data access through a single server.

Shortcuts achieve HSM by pointing to where data is located, and nothing more. If that data is moved, the shortcut must be updated with the new location, however, as discussed above, the client computer does all the interaction with the destination server directly.

Building and preserving an audit record for a file would thus require a record of every location that file has been moved to. In addition, each server to which it has been moved to would have to be capable of – and configured to – retain access records. Stitching that together for a single file over the course of a 10 year retention period would be difficult. Doing so for millions of files would be a nightmare

Symlinks

Symbolic links – or symlinks – are the second primary approach to HSM. Similar to shortcuts, symlinks are file system objects that contain pointers to an arbitrary location within a file server's namespace. Like shortcuts, symlinks can point to a file, a directory, or nothing at all.

Unlike shortcuts, modern symlinks are not files. Instead, symlinks are special file system objects that a file system will engage with in a transparent manner. The user of an application traversing a symlink will not know they are traversing a symlink, a behavior very different from a shortcut.

A symlink to a file is effectively an alias of that file. This is to say that it is another address at which that file can be located. On some systems (such as MacOS), changing the location of the target file will cause some or all classes of symlink to that file to be updated. On other systems moving the target file invalidates the alias pointer.

The difference between shortcuts and symlinks means that one could theoretically back up all of one's shortcuts simply by copying the shortcut files. In order to back up symlinks one would need a storage solution that supported exporting all symlinks to a database or flat file.

This can make migrating symlinks between platforms exceedingly difficult: the source storage solution has to export its list of symlinks to a format that the destination storage solution understands. Cross-vendor support for symlink-based HSM transfers is limited.

All modern server and NAS operating systems support local symlinks. A local symlink could make /disk1/location1 link to /disk2/location1. If an administrator shared out /disk1 as /share then a client accessing /share/location1 would in fact access /disk2/location1, even though /disk1/ is what was shared. Wide symlinks are used to point to network locations, but they are not universally supported.

Because symlinks integrate with the storage solution's file system, local symlinked files can be presented to clients with a subset of the same metadata as the target. This metadata is generally limited to POSIX-compatible access control and ownership information.

Stubs

Stubs are the third approach to HSM, and can be thought of as "shortcuts on steroids". Like a shortcut, a stub is a file. Unlike a shortcut, stubs generally contain rich metadata about the target, and act in a transparent manner similar to symlinks.

Stubs are reliant upon a bridge driver or agent to be installed onto the server or NAS that is serving files. This stub agent allows the file server to interpret the stub files, and perform the appropriate redirect.

Because stubs rely on stub agents to operate, the implementation of stubs varies per vendor. Each vendor approaches stubs in their own way: some (typically older) vendors combine stub files with a database. Newer vendors include rich metadata in the stub file itself, eschewing the use of a database.

As a general rule, stub files preserve the complete metadata of the target file, regardless of where that file moves. This is extremely valuable for establishing and preserving a file's access control and chain of custody, especially as it is moved from storage solution to storage solution over time.

A stub server doesn't have to actually store any files. A stub server can just be a stub server, with all data stored on other storage solutions. A stub server can do this without compromising the depth of the metadata presented to clients, allowing, for example, complex Windows-compatible Access Control Lists (ACLs) to be used, instead of only offering POSIX-compatible support.

Terrible Analogy Time

A terrible, but serviceable, modern analogy is the Twitter-like @ symbol usage to identify someone with whom you wish to communicate. Let's say that we wish to communicate with nerd messiah Elon Musk. There are a few different ways we might go about this.

If we type @elonmusk into a Twitter client, the Twitter client recognizes that we are directing communication to the user account whose profile page is located at https://twitter.com/elonmusk. In this case, the text string "@elonmusk" is acting as a shortcut.

The existence of the @ symbol tells Twitter clients that everything that follows the @ is the username of the target, at least until a special character (such as a dash or a space) is encountered. Type that same string into an email client, however, and it will behave entirely differently.

An email client doesn't know what to do with @[username]. It expects to see [user]@[domain].[tld]. Just as one operating system can't interpret the shortcuts of another operating system, the email client can't interpret a Twitter handle.

In this context, a symlink would be a Twitter handle or email address that allowed messages to be delivered to the same user. Twitter does not currently allow this, but most email systems will allow administrators to point as many different email addresses at a single account as one might wish.

A stub file would be the equivalent of a complete vcard contact file. On its own, this is useless to both a Twitter client and an email client. With the right plug-in, however, it becomes quite powerful.

Imagine that as you typed in @elonmusk into any communications medium you chose a popup appeared that would offer you every contact method that Musk used. You could ignore the popup, continue typing, and send your tweet, and things would work just fine.

Alternately, you could pause, see that you have Musk's phone number, click it and start a phone call. Or you might be able to pull up the complete history of your interaction with Musk, or see everyone who is authorized to use different types of communications methods to contact him.

As one might imagine, different stub implementations offer varying levels of functionality. Similarly, some vendors are more successful than others at making stubs easy to use and administer. In general, however, stubs are far more capable than either shortcuts or symlinks.

Deep Dive

When stubs are used on a file server they can prevent users from needing direct access to secondary storage. The driver or agent on the file server translates the request for access, fetches the data, and then presents it along the original request path. Symlinks and shortcuts both redirect the client directly to the secondary storage.

The high level analogy for shortcuts and most symlink implementations works like this: the client computer tells the HSM server that it would like to access file A. The HSM server responds to the client that file A lives on server B. The client then goes to server B to ask for access to file A.

The HSM server in the above analogy is blind to the interaction between the client and server B. Because of this, it can't log most of what goes on, or exert much control over this interaction whatsoever.

With stubs, and some types of symlink setups, the HSM server controls the entire interaction. Here, the client would ask the HSM server for access to file A, the HSM server would fetch file A on the client's behalf from wherever it happened to live, and present that file back to the client. In this interaction, the HSM server would retain the ability to leverage advanced features, such as multi-protocol support, quotas, traffic priority, etc.

Symlink implementations that preserve advanced file server features can only do so where the symlinks are only actually visible to the file server itself, and not via the network file protocol. This means that the symlink server would act just like a stub server (no client-visible redirects), but without the rich metadata support that stubs offer.

Because symlink-based HSM servers pass through the POSIX-compatible security and user information to the client, they may run in to ACL inheritance issues. These occur when the file is supposed to have one set of ACLs (as per the HSM server), but has a different set of HCLs on the actual storage server. (This is usually because someone made changes to the storage server where the file actually resides.)

Stubs-based HSM servers don't usually have this problem. The metadata for the file is stored in the stub file. So long as the stub server can access the file – wherever it happens to live – then it can apply the ACL information stored in the stub metadata. This allows files to be stored on destination storage solutions that offer reduced metadata functionality, such as cloud storage or other object stores.

Maintenance

Shortcuts, symlinks and stubs all require regular pruning (or "grooming"). This process ensures that files which have been deleted have their links removed from the HSM server.

With stubs (and some symlink implementations) all client access to files must traverse the HSM server. It is thus rare that files are moved or deleted without the HSM server being aware of it. Because shortcuts and symlinks can operate as redirects, however, HSM implementations based on them may also be vulnerable to files being moved without informing the HSM server.

Vendor implementation matters when considering the long term maintenance of an HSM server, especially when discussing backups. Any piece of IT infrastructure can fail, including one's HSM server. The HSM server's critical role as the index of all of an organization's unstructured data makes data protection of the HSM server a critical consideration, even if the volume of the data to be backed up isn't large.

As discussed above, shortcuts are files and can simply be backed up as files. Symlinks must be exported to some form of database. Depending on implementation, stubs may be as easy to back up as shortcuts, or they may require a combination of a file backup and a database backup. The latter option is more common with older stub implementations.

Newer stub solutions, which store all information in the stub files instead of using a database, can be made extremely powerful by combining them with a versioning system. Because the stub files contain rich metadata about the target file, a versioning system becomes a quick and effective way to not only back up the stub files, but to create a complete history of that file, and ever movement it has ever undergone through the organization's storage infrastructure.

As organizations seek to store their data using multiple infrastructures, HSM is seeing a resurgence of interest – and relevant – to organizations of all sizes. Take the time to research the details of the HSM storage solutions available, and avoid the temptation to simply choose that which is most convenient. HSM solutions cannot be easily replaced or interchanged, making them a strategic commitment that could dictate an organization's IT for years to come.