MatrixStore Architecture

ObjectMatrix

Contents

1 Overview

This document describes the thinking and rational behind the MatrixStore products. The document is written on a technical level for those who care about the nuts and bolts of how data is kept and accessed securely and forever within MatrixStore storage.

It is the aim of this document:

- To provide a deep understanding of the software architectures involved and to describe why design decisions were taken / compare those decisions to alternatives.
- To explain key “how-to” mantras for using the solution.
- To encourage discussion via peer-led review / to encourage design improvements and new functionality.

2 MatrixStore Software Overview

2.1 MatrixStore Server Software

MatrixStore Server Software runs on computer servers commonly called “nodes”.

Figure 1 - Nodes with / without faceplate

Each node contains storage capacity, CPU processing power and network connectivity (see MatrixStore Hardware Overview for further information).

MatrixStore Server Software provides the logic to join nodes together into a cluster. Commonly the cluster of nodes is simply referred to as “MatrixStore”.

Other critical MatrixStore Server Software features include enabling the cluster to:

- Act as an “Object Storage Solution” and single namespace.
- Provide automated high resilience against hardware failure.
- Provide metadata: handling; extraction; storage; search.
- Enact “Digital Content Governance” including allowing the user to set data storage policies, replication, audits and other tasks.
- Automate load balancing, which in turn allows a mix of hardware to be used (e.g. if expandingthe cluster in the future).
- Ensure future proof access to data.

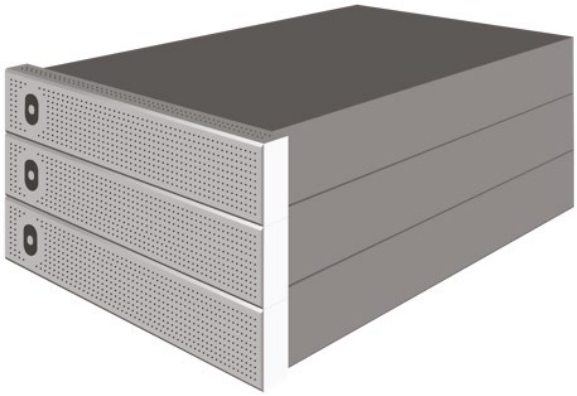


Figure 2 - MatrixStore cluster

These features and many more besides are described in greater detail throughout the document. As mentioned, the key concept of the MatrixStore Server is that it provides Object Storage.

2.2

Object Storage vs File Systems

Object storage is a way of describing a storage system that has a base logical building block of "objects". Storage Objects effectively have a non-hierarchical relationship making the number of objects stored easy to scale and the overhead of storing them minimal. This is often compared to filesystem storage, which is hierarchical by nature and has a filesystem database (metadata server) overhead.

Objects consist of three parts: data, metadata and data storage policy information.

- **Data:** can be any piece of information, e.g., a media file or a data record.
- **Metadata:** tags or key-value pairs; typed or un-typed – typically information about the data that is in the object and sometimes used to find the object.
- **Policy control information:** key about how the object should live. Often this information is used by the system – e.g., how many instances of the object to keep, how long to keep the object, who can access the object and whether the objects should be geographically dispersed, etc.

The objects are kept within MatrixStore. An often asked question is, "what are the advantages of object storage over a file system?" Object Matrix has in depth papers on this subject, but in short:

- File systems primarily provide a tree of directories within which files can be dropped. This is a very top-down restricted view of the world and this restriction has several issues:
  - Over time the meaning of the rigid file structures becomes lost and fragmented.
  - Trawling through directories to drop files into the correct one, or, to obtain the right file is time consuming.
  - When dealing with millions (or billions) of files file systems can be become less efficient.
- Fragmented file systems over multiple geographic locations work badly.
- File systems treat metadata like a bolt-on, if at all. It is rarely ever searchable, and is rarely automatically preserved. With object storage it is automatically kept with the object: transported, replicated, and generally preserved at the same levels as the essence itself.
- Object storage systems don't have the overhead of the filesystem metadata controllers.
- File systems are normally built with data blocks on block storage devices and typically work badly if block storage returns results with different latencies (i.e., file systems typically require hardware to be of the same or a similar generation).
- This adds up to object systems being inherently scalable to billions of objects, e.g., Amazon, Google and Facebook use object storage for their solutions. The architecture is flexible (e.g., for replication) and data is searchable.

In the media industry, with MatrixStore it could be added:

- Built-in (to the core of MatrixStore) media meta data analysis and extraction along metadata indexing and search functionality.
- Built to handle large media "objects".
- Built-in full audit trail handling.
- API integrations bring tight coupling between MatrixStore and its technical partners.

Amongst other features.

MatrixStore server software is written in Java, C, and Python programming languages.

2.3

MatrixStore Node Operating System

A stripped down and optimised version of Ubuntu Server is used at the operating system level. Ubuntu has been selected for its high reliability and ability to support a wide range of hardware configurations. However, the Ubuntu build has been hardened for usage with MatrixStore:

- All services not required are either removed or turned off. Since very few externally communicating services are left switched on this helps to protect MatrixStore from operating system security vulnerabilities and improves the reliability (uptime) of what is already considered to be one of the most stable of operating systems.
- The build is optimised for performance for data storage.
- The XFS (file system) is used on each node within Ubuntu. Again, this was selected for its extremely strong reliability track record, as well as its built-in resistance to fragmentation. MatrixStore Server Software is largely independent of the operating system and has previously been run on various operating systems including our own distribution of a highly secure Linux kernel and Mac OSX.

Other notes:

- Fragmentation is auto monitored and de-frag tools are auto run if necessary.
- The operating system is completely hidden from the user – admin and maintenance tools are the sys admins access to the nodes. This has the major benefit that mistakes can't be made by an admin at the operating system level endangering data.
- All API access to the cluster via MatrixStore APIs (both management and data APIs) is via certificated access.
- Data transmissions can be optionally encrypted. Certificates and checksums are always encrypted and/or sent via encrypted channels.

2.4

Node Software Behaviours

MatrixStore Server auto-boots on each node running on top of the operating system. Primarily written in Java, it occasionally calls on a few C libraries and on operating system scripts. Underlying fundamentals in the server layer are:

- **Self-sufficient:** each node is able to run without a high dependency on services from other nodes.
- **Lightweight inter-node communication:** by keeping communication between nodes to a minimum the cluster is highly-scalable (by avoiding inter-node traffic noise).
- **Highly-reliable:** nodes should be able to run for many months at a time without requirement for reboot. This has been witnessed in real-world usage.

Architecturally the server is split into three layers, a "system layer", a "services" layer and a "process-in-place" layer. System layer is a minimal layer that is intended to be fairly static, provide security protocols, and to gather hardware information. Services layer handles all high-level cluster functionality, client connections, etc. The process in place (PiP) layer contains applications to examine media data and to, e.g., extract metadata from that media data.

2.5

NTP Connection

MatrixStore servers can be connected to an NTP server to set system time to the UTC world clock.

2.6

MatrixStore Clients

Client connectivity to the cluster is always through one of two APIs:

- MatrixStore API: C or Java. This API allows connectivity for data storage, retrieval, search, etc operations. This API has been programmed to by numerous 3rd companies.
  - MatrixStore APIs guarantees security, cluster location discovery, data transmission retries and much more besides.
- MatrixStore Management API: REST interface. This API allows higher level operations such as user creation and statistic information gathering.

Clients can be on MacOS, Windows, various Linux versions.

Clients communicate with the server using TCP/IP. UDP is used by some maintenance tools.

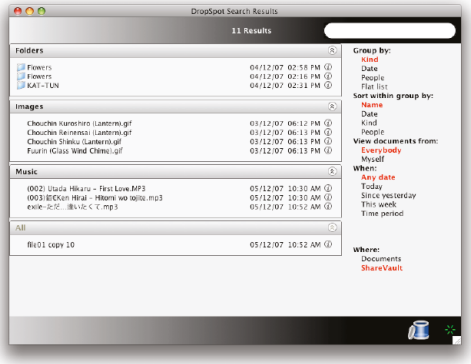


Figure 3 - DropSpot main interface

Figure 4 - DropSpot metadata form

2.7

Client Applications

The following applications are provided by Object Matrix. (It is worth noting that all data written via the MatrixStore API can be read by any Object Matrix application):

2.7.1 DropSpot

A client GUI application for performing data archive / retrieval / search. DropSpot is multithreaded and provides strong job control and verification – it is typically used for data ingest tasks, e.g., camera data on to the storage. Metadata can be added when data is ingested via a highly customisable metadata entry form. Advanced search can be performed using that metadata.

Runs on: Mac, Windows, Linux.

2.7.2 MatrixStore Vision

MatrixStore Vision is a webserver that enables connecting Web browsers to search, preview, select, add metadata, and upload and download data. 3rd party transcode engines such as Elemental, Ortana Cubix, Glookast, Root6 ContentAgent, Telestream Vantage, Drastic or FFMPEG can be used to generate proxies that can be viewed in Vision.

Application server runs on: Linux.

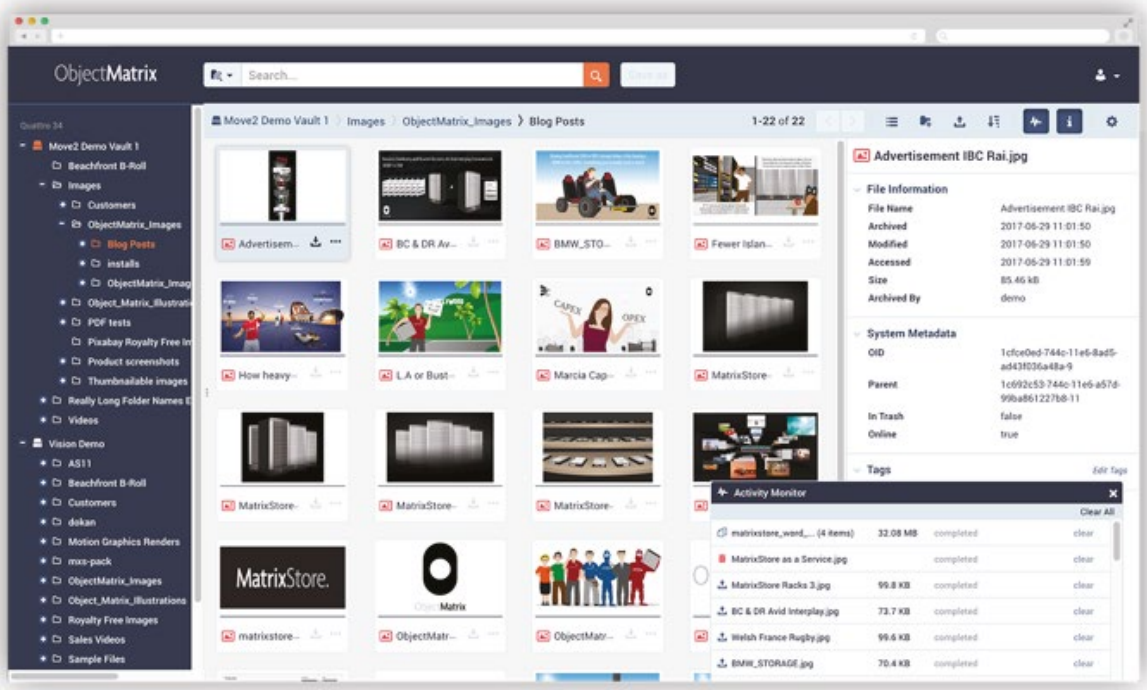


Figure 5 - MatrixStore Vision interface

2.7.3 MatrixStore Sense

Sense provides a analytics and monitoring dashboard for MatrixStore[1]. Analytics currently available and planned in the 2020 Sense roadmap[2] include:

- Cluster and Node status and history
- Vault usage and history stats
- Application usage stats
- User/machine usage stats
- Full system alerts and monitoring capabilities
- Plugins for other storage devices that the customer may be using, including cloud plugins for stats from cloud services

Application server runs on: Linux.

[1] Pre MatrixStore v4.1 monitoring was carried out via a Zabbix based application, v4.1 onwards begins to migrate the monitoring towards the single MatrixStore Sense interface.

[2] Roadmap is not fixed and from time to time is subject change.

Figure 6 - MatrixStore Sense interface

Future expansions of this module will allow for deeper data analysis, usage and content analysis. This will answer questions such as “has a project been used in the last 60 days”, “how much audio data do I have” and “is my data being kept on the correct storage tier”.

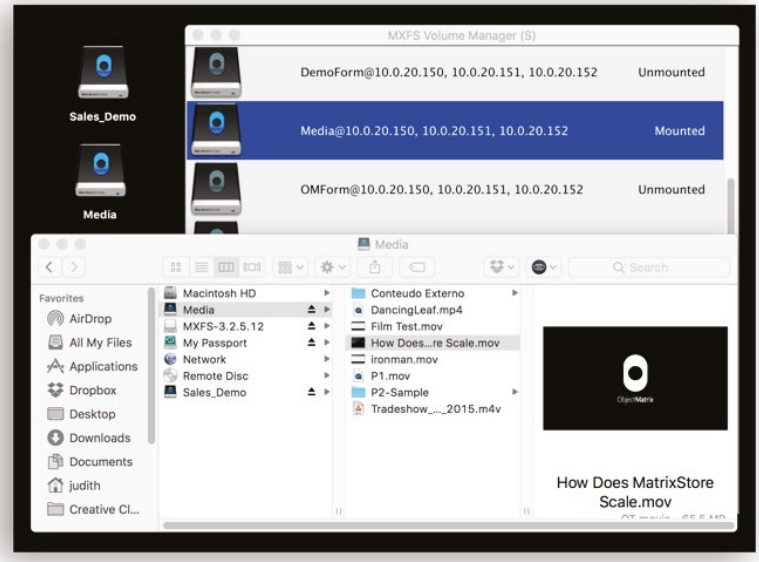


Figure 7 - Graphic design vault mounted with MXFS

2.7.4 MXFS (MatrixStore File System)

This filesystem can be launched on a client machine to provide file system type access to the cluster from that machine. The client machine will see the cluster as a drive letter / Volume. In the background MXFS sends and receives data to/from the cluster using MatrixStore API calls. Some of the advantages of MXFS is that it doesn’t need to cache data at the client machine, and rather can map byte read and write requests directly on to the objects themselves. This avoids all types of issues such as client machine crashes, and means that when a file is written the user knows that it is safely on MatrixStore. When reading a file data can be random accessed or streamed / contiguously read. This differs from some object storage platforms that read files in randomly positioned chunks (that method, whilst it can be efficient doesn’t work in many media workflows that require a consistent and fast time to first byte and time to continue streaming).

MXFS is available with MatrixStore without a requirement for licenses (ie at no additional cost). MXFS is currently one of the most efficient ways to read/write data to MatrixStore, second only to MatrixStore API access for performance.

Runs on: Mac, Windows, Linux.

2.7.5 MatrixStore Move2 (SGL / Sony ODA / S3 / Xendata / SpectraLogic BlackPearl)

Move2 is used to control hierarchical storage management between a vault (see MatrixStore Concepts) on the MatrixStore cluster and 3rd party systems such as LTO or optical archive systems. The applications can move data on demand from a user or according to business rules (age etc). When data is archived to a 3rd party system the metadata can also optionally be archived. The metadata also remains on the MatrixStore system to enable search and retrieval of the objects. DropSpot and Vision can be used to restore data.

Move2SGL refers to SGL Flashnet from Masstech.

Move2S3 is a key application for copying assets to AWS S3. This is used in hybrid workflows, i.e., to move or copy assets from MatrixStore to S3. This can be for the purpose of backup, content distribution, remote teams, transferring assets from one region to another region, etc. Where assets arrive into AWS and need to be ingested into MatrixStore, please discuss with Object Matrix since there are available methods.

GUI Runs on: Windows, engine runs on Windows or and Linux.

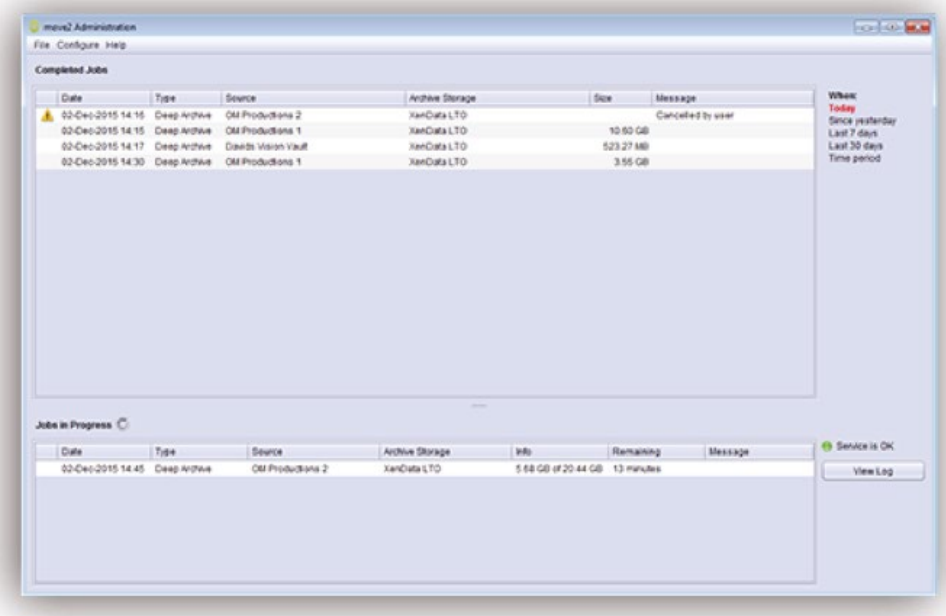


Figure 8 - Move2 admin panel

2.7.6 MatrixStore InterConnect

Integrates with Avid Interplay through Interplay Web Services to allow the archiving and retrieving of Interplay data. To perform archiving/retrieval of media assets the user simply uses the Interplay interface. InterConnect is feature rich, e.g., the user can select to only archive certain video resolutions, can archive sequences, will detect when sequences have been changed or media assets updated, can chose to delete after archive from Interplay (in which case the system works out what can and can't be removed) and many other features besides.

Runs on: Windows.

2.7.7 MatrixStore FTPConnect

This is an FTP server that runs on Windows or MacOSX. It translates client FTP requests into MatrixStore API calls to store/retrieve data in the cluster.

Runs on: Mac, Windows, Linux.

2.7.8 MatrixStore S3Connect

This is an S3 gateway, it translates client S3 requests into MatrixStore API calls to store/retrieve data in the cluster.

Runs on: Linux.

2.7.9 MatrixStore Maintenance Tool

This application is typically used by Object Matrix engineers to create clusters, inject upgrades into nodes and to perform other low-level administration functions. One key feature of the maintenance tool is that it can simultaneously upgrade the software in an entire cluster rather than requiring nodes to be upgraded one by one.

Runs on: Mac, Windows.

2.7.10 MatrixStore WebAdmin Tool

This application is typically used by the administrator at the customer site to monitor a cluster and to perform basic administration upon the cluster. Since v4 the admin tool has been fully browser based.

Runs on: Chrome, Safari.

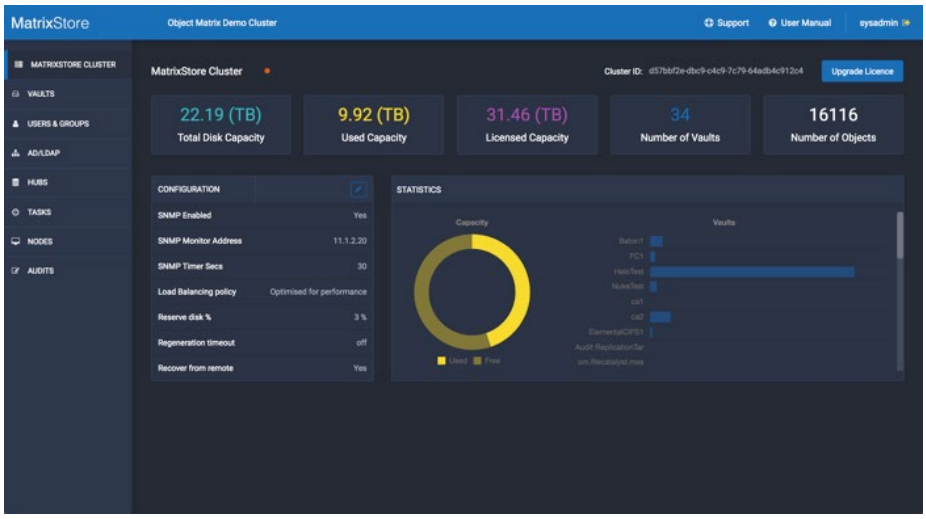


Figure 9 - Admin info panel

2.7.11 MatrixStore Shell

This is only available to Object Matrix support and allows shell access to the MatrixStore cluster with a command layer. Shell access allows the support engineer to perform low-level engineering tasks.

Runs on: Mac, Linux.

2.7.12 Samba / NFS

MatrixStore has a SMB(CIFS) interface that runs as a gateway. It's currently based on SAMBA v4.2, with supports SMB v3.0, v2 and v1 access.

- Provides file system access without needing to install special software on client machines
- Optionally launch multiple Samba servers against different MatrixStore vaults when wider bandwidth is required
- Runs on MatrixStore Hub hardware (See configurations)
- Centralised control of multiple Samba servers and user access rights via the MatrixStore Administrator application
- Supports SMB and CIFS protocols
- Can be installed with failover configurations

Additionally a MatrixStore NFS interface provides access via the NFS protocol.

Runs on: Linux.

2.7.13 3rd Party Applications

There are numerous Object Matrix partners who have integrated their products into MatrixStore as a storage solution.

3 MatrixStore Software vs Other Object Storage

No two object storage solutions are alike, but in general the following differences can be said to be true:

3.1

MatrixStore is built as “plug and play”

Some on-prem object storage systems are more like projects where the end user gets caught up into a high maintenance cycle of understanding, installing and maintaining the product. MatrixStore is typically delivered on well defined, well tested (due to the 100’s of similar node configs installed) hardware, and ready to use. Many object storage systems are designed to be configured for B2C use cases that may require various levels of object caching whereas MatrixStore is built for media customers and the internal and B2B use cases that customers have there.

3.2

MatrixStore is built with future expansion in mind

This is a broad claim, but we frequently see solutions where although you can expand the solution with new nodes or metadata servers, when those nodes or metadata servers have different characteristics or performance (e.g., new generations of hardware) the solution requires all nodes to be updated or replaced.

The MatrixStore architecture makes each node self-sufficient for the data and metadata that it holds and is loosely coupled to other nodes in the cluster, and therefore the architecture doesn’t care if another node is slower or faster than itself.

This makes future expansion a key benefit of MatrixStore over other solutions.

3.3

MatrixStore is built for Media Data Assets

Object Storage is often built with one or both of the following in mind:

- **B2C** – therefore very focused on flexibility / data caching / billions of objects
- **Erasure coding** – therefore trying to keep hardware costs down at the expense of speed of access and often software features

In the case of B2C the trade-off is normally that the storage is very simple – often without digital content governance, process in place and fully searchable distributed DB’s that MatrixStore provides. Erasure coding has huge side effects around rebuild times, contiguous file reads, and general system load in stressed circumstances.

MatrixStore is focused on keeping assets in contiguous files to allow the fastest possible random access, as well as a time to first byte that is uniform and reliably fast. Some object storage will return a file to a client through multiple blocks that may not arrive in a first through to last byte fashion. There are many workflows in the media world that rely on time to first byte and fast random access, which means that simply will not work on erasure coded data when that is the case.

3.4

MatrixStore is built for Media Data Handling

The Digital Content Governance feature set in MatrixStore, with its ability to easily manage vaults of data, audit, replicate, lock down and other such functionality is unique and built around requirements from the media industry.

Process in place, extracting metadata from assets in MatrixStore allows for further metadata augmentation than media companies might already have and reduces the media company's reliance on a proprietary asset manager database.

3.5

MatrixStore Metadata Handling

It cannot be stressed enough that most object storage solutions allow the storage of metadata but expect you to maintain an external DB of metadata for search purposes. This search is built into MatrixStore for ease of use, maintenance and expansion and again harks back to our key philosophies of ease of use and ease of future expansion.

3.6

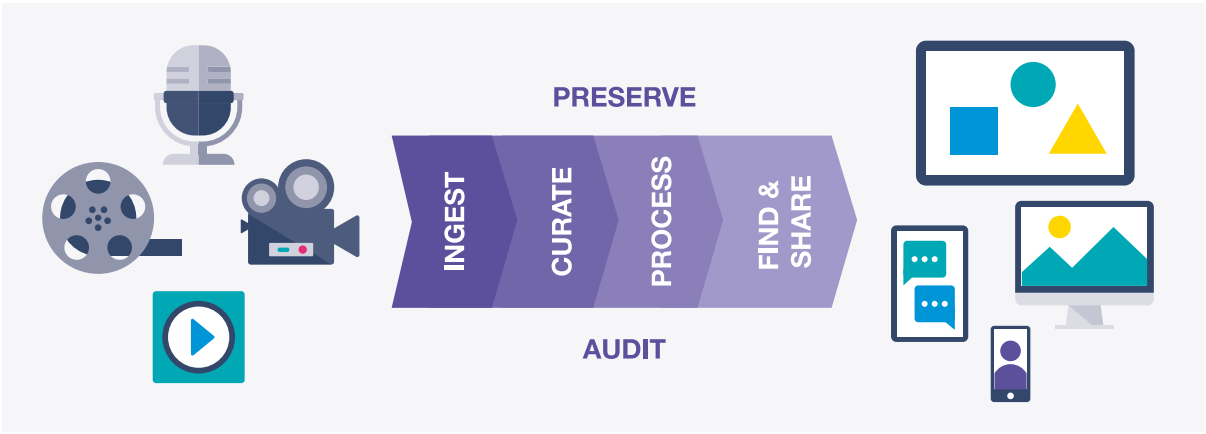
Open and Non-Proprietary

Vendor lock-in comes in many subtle and nasty ways including:

- Proprietary databases that make it difficult to unload / understand metadata stored. MatrixStore makes unloading metadata in text key-value formats and against objects easy.
- Storing assets through one interface doesn't make them available except back through that same interface. MatrixStore allows (e.g.,) assets stored through the s3 interface available through the file system interface. This is possible because of the centralized, used by all, "database" that is in the object storage cluster.
- Global namespace lock-in – many global namespaces are just an excuse to keep all the ids in a proprietary database that is accessed through a proprietary application. It is extremely risk ridden if that application was to be removed and if then all assets and metadata cannot be easily accessed and used.
- Cloud lock-in – where the cost of offloading assets from a service is prohibitively expensive.

MatrixStore is object storage designed and built for media workflows and for B2B or internal workflows. Other object storage provide complex geo caching algorithms that are designed for B2C, sell on erasure code claims of using less hardware (whatever the reality of hardware usage, future proofing or access speeds) or are very simple DBs of IDs rather than fully fledged redundancy and scalability.

4 Digital Content Governance



Digital Content Governance is an extension of Information Governance tailored for Media Assets.

Digital Content Governance (DCG) answers the problems of "storage" by creating an environment where assets are controlled with an emphasis on:

- Security
- Portability
- Defined lifetime
- Auto-protection

DCG is a software layer that manages the assets, their locations, and their safe-keeping along with other features such as search and audit.

To achieve this DCG utilises data storage policies. Data storage policies are generally (but not necessarily) set when an asset is stored. E.g., "This is an archive asset and it should be kept both in London and New York", or, "this is an asset for local editing". It is then the responsibility DCG to look after the asset using the data policies attached to it throughout the lifetime of that asset regardless of hardware failures, network down times or hardware changes. If DCG is implemented using an object based storage system then these policies live with the object itself. In this way and by the asset knowing how it wants to live, the policy becomes transportable, much as metadata stored inside an MXF is kept with the essence.

governance /ˈgʌv(ə)nəns/ *noun*

Conducting the policy, actions, and organization of a state, organization, or … media assets …

Secondly, DCG can harvest the metadata from within the assets making them searchable, as well as augmenting that metadata from external sources such as asset managers or user provided information. This metadata should be searchable but should also be portable – from one system to the next and, like an MXF wrapper, not tied to an individual application that can quickly go out of date.

Lastly, DCG can add additional services such as auditing when an asset is accessed or updated.

MatrixStore is therefore a Digital Content Governance platform. It achieves this through the usage of object storage and business logic.

5 MatrixStore Hardware Overview

Hardware from any vendor that supports Linux can be used with MatrixStore, although only hardware that Object Matrix has qualified will be supported. A cluster consists of nodes and switches:

Cluster	Notes
Nodes	Minimum 3 (see below).
1GB or 40GB switches	Two for internal traffic. One for IPMI traffic (optional). One to Two for external traffic (all external connectivity is supplied by the customer).

Hardware is grouped into nodes:

Node	Notes
CPU Storage	Dual core 2GHz and higher, 4 core minimum. Up to 2 volumes, tested up to 960TB per node. On Enterprise nodes this is with RAID6. V5 nodes use Dual CPU 4110 Skylake processors.
1 Gigabit to 40 Gigabit NICs (typically 10GbE)	Two for internal traffic and two for external. External can be 1G copper RJ45 BaseT, 10G copper RJ45 BaseT, 10G SFP+ copper or up to 40Gbe.
IPMI Operating system	One port for IPMI node control (optional). MatrixOS (based on Ubuntu Linux).

RAID Control	Hardware based for performance and reliability. Capacitor and flash backed RAM.
Memory	64GB per node (minimum of 4GB).
Size	Unrestricted but typically 24TB to 960TB.

Typically Object Matrix supplies only enterprise quality components, such as 5 year warranty HDDs.

Why a minimum of three nodes?

- The cluster has hardware failover at every level (switches, networking, nodes).
- When a data object is stored into the cluster it is always initially stored to 2 separate nodes in the cluster. Therefore, a cluster must have at least 2 nodes in operation to be able to store data.
- If the cluster only had 2 nodes then writes couldn't continue in the eventuality of a node being down.

Other notes about the hardware configuration:

- If a node goes down, data is regenerated from the good node (that is still up) to a new node location. This ensures that within an allowable time period data is always kept with two good instances.
- For failover there should be 2 external traffic and 2 internal traffic NICS.
- Storage volumes are typically set to be RAID6. With the data kept in two locations (both RAID6), a minimum of 6 disks would need to irrecoverably fail for data loss to occur, furthermore, that failure would have to occur before data has a chance to regenerate.
- Nodes are typically configured with mirrored system drives. System drives can be used to hold the operating system, system logs and some data that is shared between multiple nodes.
- 300 million metadata entries per node
- CPU speed: Recommended to use at least quad core 2GHz, preferably higher. Modern architectures should consider dual Skylake 4110 or Cascade Lake 4120 CPU.
- Cluster is designed for up to 120 nodes (115PB), however, there are no hard and fast reason not to go above that number. Cluster is tested up to 40 nodes.
- Time for the cluster to self-heal is a calculation based upon the size of the node that has gone down ÷ the number of nodes in the cluster ÷ internal NIC speeds. Or: the smaller the size of each node and/or the more nodes there are, the faster regeneration will be.
- The more nodes there are the greater the bandwidth of Ethernet connections to external connections to the cluster. A "fast" cluster should therefore have low amounts of storage per node, a "high density" cluster should have high amounts of storage.

5.1

Networking

Client machines access the cluster via IP addresses. Nodes within the cluster contact each other via the internal network.

Figure 10 - Nodes front and back

External Traffic	Examples
API, ports 1907, 1908	Write and read operations.
Management API, port 667	Vault creation, user modifications.
Secure Socket Layer (SSL), port 8443	For Vision traffic only

Internal Traffic	Examples
Management API, ports 666, 667	Vault creation, user modifications.
API, ports 1907, 1908	Write and read operations.

External Maintenance Traffic	Examples
SSH, port as allocated	SSH should normally be switched off.

When configuring a MatrixStore it is essential to:

1. Assign an internal and external IP address to each node.
2. Ensure that firewalls within the network do not block traffic on any of the above external or internal ports*.
3. Ensure that each node can see each other node via Ethernet networking, both within the internal and the external networks.

When configuring MatrixStore it is normal/strongly recommended to:

1. Configure one (or two for failover) network interface ports to be dedicated to internal traffic.
2. Configure one (or two for failover) network interface ports to be dedicated to external traffic.
3. Dedicate one (or two for failover) switches to internal traffic, and thus isolate internal traffic from the rest of your organisation's network.

Switches will typically be unmanaged Ethernet. As standard, Object Matrix implements failover on the switching using switches that support IEEE 802.3ad.

Data transmitted over the internal network is not encrypted. Therefore, internal switches should be physically isolated and protected. Because the internal switches sit behind the nodes, compatibility with switches in the rest of the client's network is not required.

5.2

Virtual MatrixStore

MatrixStore can be configured on virtual machines for the purpose of testing.

6 MatrixStore Core Concepts

6.1

Overview

MatrixStore is an object based storage solution. Every object based storage solution has strengths and weaknesses that are a result of underlying architectural decisions and project aims. MatrixStore includes the following high-level ambitions:

- Scalable solution to 100's of Petabytes; not, scalable to Exabytes (e.g., Amazon).
- Hybrid solution usage: fast object storage that also delivers full speed filesystem access - the solution provides exceptionally fast random access to objects.
- Provides the highest levels of data security and data protection.
- Object storage with full search capabilities (many object stores have very limited search).
- Completely plug and play architecture for easy on-premises maintenance, expansion etc.
- For 100's of simultaneous users rather than "millions".
- Object storage via data + metadata + data storage policy control.
- Full suite of data services including search, metadata handling, replication options, self healing and full support for an eco-system of client applications.
- Capable of mixing different hardware nodes, e.g., as a result of scaling the cluster over a number of years.
- Process-in-Place: augmentation of metadata and artificial intelligence.

6.1.1 Why archive to disk?

Disk is a proven well-understood technology with good transfer speed and excellent random access. File systems are also well established. However, in terms of archive for large organisations the challenge comes when 1000s of disks need to be managed as storage locations, including the challenges of authenticating that data is maintained bitwise exact.

Normally on disk based solutions:

- Large scale disk solutions are difficult to manage, e.g., 500TB+ SAN solutions.
- Bad administration can very easily lead to malicious or accidental data loss.
- Solutions become outdated and data needs to be transferred to the "next generation" solution.
- Downtime is caused by individual component failures.
- Transferring data offsite, search, firewalling, etc., requires separate software modules, each of which might result in data loss and downtime when upgraded or broken. Over time, the systems become disjointed as individual components become out of date.

MatrixStore has software augmented solutions to overcome each of those challenges.

Disk based archiving is a proven way of keeping large amounts of data as shown by modern object stores such as Amazon S3 or Microsoft Azure.

MatrixStore has significantly more than 100 man years of development and has been proven by long running customers in the media sector.

6.1.2 MatrixStore "Services"

MatrixStore software is designed from the inside out to support the requirements of Nearline storage and medium to long-term archive. Normal file systems provide only basic software services (such as write and read of a file).

MatrixStore adds multiple software services:

- Secure / 'firewalled'.
- Near zero maintenance, e.g., to scale/to manage large volumes of data.
- Provides guarantees about the delivery and storage of data to and from the disk.
- Monitors the archive for hardware failure, taking automated actions where required.
- Fast.
- Searchable.
- Scalable.
- Process-in-place.
- Stubbed objects (for archive).
- Replication.
- Easy to roll-in/roll-out technologies into the pool of storage as time goes on to name but some.

6.2

Object Based Storage

It wasn't so long ago that just about all data was stored within a file system, but as the scale of data storage has increased dramatically, so the rise of object based storage has become an essential part of keeping digital media assets. Examples of object based storage include Amazon s3, Cloudian, Google, Facebook, Scality, Microsoft Azure, to name but a few systems. Object based storage has the following fundamentals:

- Like files, objects contain data, but unlike files, objects are not organized in a hierarchy. Every object exists at the same level in a flat address space (sometimes called a storage pool).
- Objects can also represent records rather than data files.
- Both files and objects have metadata associated with the data they contain, but objects are characterized by their extended metadata. Each object is assigned a unique identifier that allows the server or the end user to retrieve the object without needing to know the physical location of the data. This approach is useful for automating and streamlining data storage in scalable environments.
- Within MatrixStore, the user never needs to know the unique identifier, since the metadata can be used to look up the object.
- Objects are self-describing: unlike in a file system where should metadata servers blocks of data are individually meaningless, object based solutions are highly resilient to data loss through their very atomicity.

On top of massive pools of objects virtual views can be built to view the data. One view of a bunch of objects might be from an asset manager, another view might be a file system view. However, in order to support file system views the underlying object based storage solution must support expected file system behaviours, such as response times.

Under the hood, a MatrixStore cluster actually stores objects in numerous file systems. Benefits of this are:

- Proven technology.
- Low interdependency between objects.
- Scalable over main nodes.

Negatives:

- MatrixStore keeps indexed metadata with objects stored – there is a small cost (space and time to index) when an object is written.
- MatrixStore bit checks correct receipt of data when written – there is a small overhead for the check summing (that can be adjusted by the user to the level of check summing required).
- MatrixStore mirrors data when it is stored – there is a small overhead for flushing data to two locations over flushing to a single location.

MatrixStore is designed for the storage of millions of objects.

When data is stored into MatrixStore it is stored as an “Object”. An Object instance consists of the original data in an unchanged format and an associated metadata file containing user and system attributes for the object.

All data is stored into virtual constructs called “vaults”. A vault contains meta attributes pertaining to the handling of objects within that vault, such as whether those objects must comply to regulations (See “Enforcing the longevity of data”) or whether data should be replicated, etc. In effect, that set of policies can be considered to be attached to each object.

6.3 Object Metadata

Object metadata is used by MatrixStore and by users for different purposes. Users generally use metadata to identify objects, but may also use metadata to hold attributes about the object.

MatrixStore servers uses metadata to perform system tasks, such as to authenticate that a corruption hasn’t occurred within the data file.

Object metadata is primarily stored in two locations:

1. In “flat file” format along with the object itself.
2. In a searchable database.

The “flat files” are stable and are generally stored alongside the data files. The contents of the flat file is a standard Java HashMap, wherein the keys are ASCII-US text, and the data is the binary characters representing the value of the attributes. Since each object stores its own metadata in its own file, corruption in one will not affect another, however, the downsides are the filesystem overhead of storing extra files as well as storage speeds if many attribute updates are made to an object. It is not expected that rapidly changing attributes (e.g., a counter) will be stored within MatrixStore objects.

Every node stores a database of all of the metadata upon that node. This allows for distribution (as clusters grow) and low levels of interdependency between nodes. Should the database become corrupt, it can be rebuilt from the metadata files on the node.

Metadata can be attached with two types of identified to an object:

- System attributes are used internally and are not commonly returned to the user.
- User attributes are those added by the user and/or 3rd party applications (e.g., 3rd party MAM).

When stored, attributes can be tagged as searchable or non-searchable. Only searchable attributes are added to the server database.

There are two types of searchable databases in the cluster – in-memory for fast search (such as filesystem attributes) and content search for more complex search, e.g. “cam*” to find all matches beginning with “cam”.

MatrixStore (v3.1 or newer) also extracts metadata (and can extrapolate additional metadata) from the contents of objects. This is called “process-in-place” (PiP).

6.4

Vaults

Any object stored in MatrixStore is in fact stored into a logical entity called a vault. As many vaults as required can be created in a single cluster (Object Matrix tests for up to 2000 vaults per cluster, but can test for higher numbers of vaults should there be a requirement to do so). Vaults are intended to be used:

- One per project.
- One per department.
- One per data type or workflow.
- One per customer.

Vaults are not normally intended to be used per user, rather, per user group.

When a vault is created it can be given a set of properties dependent on the data to be stored within it. Those attributes are:

- Provisioned Capacity (see section below).
- Object Policies.
  - Replication settings.
  - Group access settings.
  - Compliance settings (data retention period, retention policies).
  - Audit settings.

Via vaults, MatrixStore supports “Multiple Tenancy” and can provide limited statistics for companies wishing to implement chargeback mechanisms for vault throughput / storage space usage.

Vaults are created / modified via the Management API. This is called from the MatrixStore WebAdmin.

Security between vaults is stringent (see Security section), therefore the system administrator is not allowed to write / read / etc data within a vault, but he does have permission to reset the vault administrator’s account.

6.4.1 Vault Provisioned Capacity

It is possible to set and update a boundary on the capacity that a vault can consume within the cluster.

Should the provisioned capacity be exceeded the users of the vault will not be able to perform any further write operations until such time as the provisioned capacity is extended or the user deletes existing data in their vault thus freeing up capacity.

It is also possible to set the provisioning to be boundless, as such a vault can grow and grow as long as there is cluster capacity available.

Capacity is not reserved for the vault, thus, many vaults could be given a “1000TB” provision, even if only “100TB” of storage space is available on the cluster.

The system will allow writes until that capacity has been exceeded. i.e., if the last object written is a very large object, it could be that the capacity is significantly exceeded, but then the next object write will fail.

6.4.2 Vault Audits

It is possible to select whether to audit reads, writes, and / or deletions at a vault by vault

level. Audits on objects allow admin to see the user, date and time, IP address and action on the objects. The cost of auditing is a space and indexing overhead, so they should be switched off on vaults where auditing is not required, e.g., if performance is the primary concern.

Audit	Notes
Reads	Track when an object is read; typically only switch this on where data is sensitive.
Writes	Track when an object is written; typically switch this off if performance is more important that being able to track objects.
Deletes	Track when an object has been deleted; this is typically set on.

All audits for a vault are removed when the vault is deleted.

6.4.3 Vault Integrity Level

Users can select the “integrity level” on a vault, i.e., the checksum used during data transmission to and from the vault.

Integrity Level	Notes
Fast / DCC	DCC is a very lightweight checksum that spot checks data has been correctly received in the cluster from the client. It should only be used where performance is paramount.
Medium / Adler32	Typical checksum used. The chance of an object having been received incorrectly and for this checksum failing to notice the incorrect receipt is extremely unlikely and is in the region of 1 in 65,000.
Strong / MD5	If performance is less important and there is a high chance of corruption during the transmission of data then a very high vault integrity level should be used.

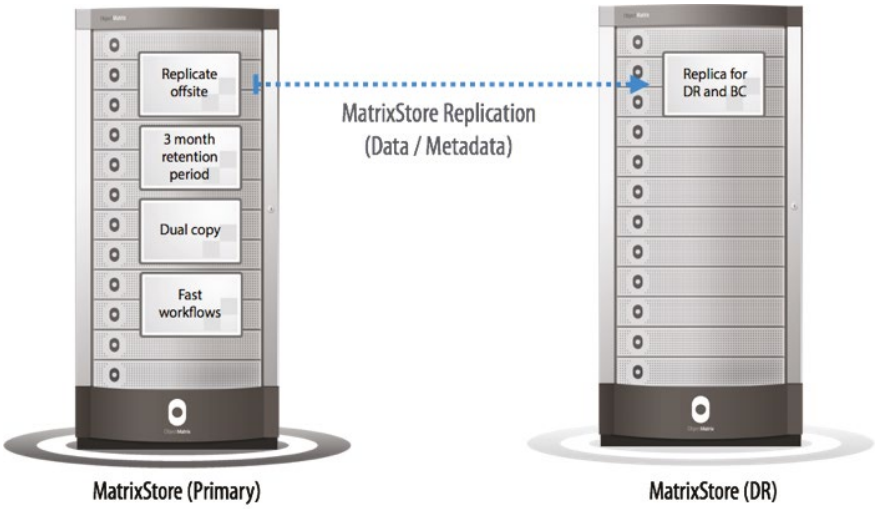


Figure 11 - Object longevity diagram

6.4.4 Object Longevity Guarantees

Users can select the "integrity level" on a vault, i.e., the checksum used during data transmission to and from the vault.

When designing MatrixStore to hold archive data, it was seen as essential to allow users to lock down data such that it cannot be deleted for a period of time or even forever.

Locked data is effectively read-only for a fixed (or administrator adjustable) amount of time.

This can be useful for a multitude of reasons:

- To protect against viruses/malicious users.
- To protect against human error (oops! Just deleted the data in the wrong vault type errors).
- To comply to a multitude of government and/or industry regulation compliance requirements.

When the user creates a vault the vault object is set to have a compliance attribute. The compliance attribute can be set to be on or off. If on, the amount of time that the data is stored for can be modifiable or extendable only.

When compliance is on, any object stored cannot be deleted for the period of time indicated in the vault.

Notes

Although the "modifiable" setting doesn't guarantee data longevity against user actions, it does however guarantee that any non-system administrator will not delete data, and is therefore popular in smaller companies.

If the setting is extendable only then the length of time that was selected can only be increased (not decreased). This means that the administrator cannot quickly change the setting, delete a file and then change the setting back again.

If the setting is modifiable, then an administrator can change the setting to then delete a file.

Data will not automatically be deleted when the longevity date / compliance threshold has expired.

MatrixStore allows metadata pertaining to objects to be modified at any time.

MatrixStore implements the compliance setting at a vault level. When a user wishes to perform an action such as a "delete" the settings of the vault are checked, by the MatrixStore Services layer, before allowing the delete to occur. This effectively stops users without physical access to hardware from being able to delete objects that have been stored protected.

6.5

MatrixStore Regulation Compliance

Government regulations often include requirements/ guarantees to be made about the way in which data is handled, accessed, security protected, protected against loss, protected against accidental and/or purposeful deletion, searched, stored, audited and authenticated.

MatrixStore helps to support, amongst others, the:

- Security and Exchange Commission Rule 17a.4 that aims to prevent overwriting, erasure or alteration of records.
- HIPAA privacy ruling for Data Protection, requiring compliant backup methodologies to ensure the security and confidentiality of patient records.
- The Sarbanes-Oxley Act of 2002 protecting investors by improving the accuracy and reliability of corporate disclosures. The Act amends mail and wire fraud infractions with harsher punishments and imposes fines and prison sentences of up to 20 years for anyone who knowingly alters or destroys a record or document with the intent to obstruct an investigation.

The set-up of a vault should be governed by the classification or type of data to be stored in that vault in accordance with any internal or legislative requirements. Once set-up, data stored into that vault will be enforced to comply with that set-up.

MatrixStore achieves compliance via functionality implemented in the server layer and the client:

Data Immutability

- MatrixStore can lock down data and can be set to disallow any updating of objects archived.

Data Longevity/Non-repudiation

- Vaults can be set such that data cannot be deleted for a period of time. The MatrixService layer enforces adherence to the policy. The period of time cannot be changed if the vault is set to be unchangeable.
- Different vaults can be set for different data types.

Data Security

- Communication with MatrixStore can be up to 256-bit asymmetrically encrypted.
- A public key encryption mechanism is used such that data sniffing, data modification, and replay attacks are protected against.
- Vaults can be created so that the system administrator does not have access to the data, only the vault users.

Audits

- All actions on vaults/data may be audited.
- Audits can be set to include/exclude read operations.
- Audits are maintained, secured, and protected to the same levels that data objects are.

Search

- Data may be searched back using metadata labels.

Trusted 3rd Party Verification

- Data may be replicated offsite, where appropriate, to a trusted 3rd party.

Data Protection and Authenticity

- Data and hardware is automatically monitored to be correct, thus ensuring the long-term authenticity of data down to a bit level.
- Replication of data to another site ensures protection against physical hardware attacks.

Delete

- MatrixStore does not currently shred (overwrite deleted data multiple times with zeroes before deletion), but could easily changed to do so if that is a requirement for someone.
- Vaults can be created with Trashcans that keep data for a set period of time after deletion where disk space is available.

MatrixStore has a very strong set of functionality to help meet compliance requirements.

6.6

Data Compression, Data De-duplication, CAS

Data Compression is only significantly relevant to companies that store uncompressed data. Object Matrix takes the view that most of its customers are storing video format files, and recompressing these files would be both time consuming and fruitless. Furthermore, compressed files cannot easily be edited or partially restored. Client software can easily compress data before storing; MatrixStore does not do this automatically.

Data de-duplication is not currently a feature of the cluster but is often carried out by the storing client application (e.g., backup software, asset management software). De-duplication comes with a strong drawback – that it creates data fragmentation. Object Matrix believes in storing data in an open, well-established formats.

Content addressed storage (CAS), if required, is easily implemented in MatrixStore, simply store the data's digest with the data object as an attribute and use that to search back the data when required. CAS has generally become an out-dated concept.

MatrixStore is setup to keep data in an uncomplicated, secure, and long-term fashion and has a healthy disrespect for risk.

6.7

External Services

To allow nodes to act coherently in a loosely coupled, self-supporting and scalable architecture server software runs on each node providing a set of services.

Internal services are covered in the Tasks section of this document. Some tasks that run in the cluster provide services that can be accessed externally, such as SNMP messages, and security services. Examples of such services are:

External Service	Notes
MAPI, User APIs	Management APIs, user APIs. E.g., to perform security actions, to check node status, to search data.
Sense	Provides statistics on cluster usage.
SNMP Service	Provides SNMP status.

6.8

Search

MatrixStore is extremely efficient at searching for metadata since it takes advantage of distributed search across the nodes and an optimised (for metadata) database.

MatrixStore is designed to allow up to 300 million entries into a DB, per node. A typical cluster can handle and respond to thousands of search requests per second with zero to low impact on other concurrently occurring operations.

Thus, since the database is distributed, it scales in capacity as the cluster scales upwards in nodes.

MatrixStore guarantees attributes about the database that are essential for long-term cluster integrity and low-maintenance:

1. That the database can be rebuilt from the objects stored. That is to say that the object's metadata is stored within each object as well as within the database.
2. That should a component part of the database become lost or corrupted, that the database as a whole can continue to function.
3. That the database does not become slower, or require "tuning" as the number of metadata items it holds grows.

Some organisations may require or intend to use high numbers of metadata items. The metrics around the number of metadata entries vs node storage capacity may have an effect on cluster configuration selections.

Under the hoods, MatrixStore actually contains two distributed databases – one that is used for fast metadata search such as metadata used for file system information and one that can be filled with unlimited search metadata, such as content metadata. The content metadata can be searched using normal absolute values or using fuzzy search (e.g., searching for "Welsh Rugby" would also find misspellings or alternatives such as "Walsh Rugby", "Welsh Rygbi" etc.) The content database is also filled with information discovered by Process In Place.

6.9

Object Encryption

MatrixStore does not automatically encrypt the data stored, but it can encrypt data during transmission. If encryption is required, then it should be carried out in the client software layer and should use a certificate authority whose lifespan will outlive the archive!

6.10

Object Writing, Load Balancing

All data written to MatrixStore is written through the MatrixStore API. The API ensures that the transmission is completed quickly, successful, and when selected, securely.

The steps involved in sending data are:

1. Client initiates the construction of a secure connection to the cluster. Connection can be to any node.
2. Request is sent to obtain a location to send the data to. Client is given a location and a security certificate to send the data.
3. Client sends data to that location using a direct IP link to the node.
4. As the cluster receives data at one location it simultaneously relays the data to a second storage location via the internal network.
5. When an end-of-object indicator is sent by the client to the cluster, client also sends the checksum for the object. MatrixStore confirms that both nodes have received the correct data with the same checksum, and then syncs the data to disk together with any metadata and policy information. A unique ID for the object just stored is returned to the client.
6. The client can (if required) safely remove the original copy of data, knowing that the data has been correctly received and flushed to disk at two separate locations.

As can be seen by the steps above, secure, reliable transmission of data is paramount; checksums, secure transmission and careful multiple location sync'ing of data are not good for performance but are key to archiving.

Data transmission options are:

Type	Notes
Secure	Data is up to 256-bit encrypted when sent over the connection. Note that MatrixStore builds are also available for territories with export restrictions that include lighter or no encryption options. The packets are encrypted with their own keys, with the key being changed during each handshake with the cluster. Data transmission is therefore strongly protected against sniffing and replay attacks.
Unsecured	Only the connection to the cluster is secured. Unsecured transmission is faster than secured transmission, although the difference in speed depends on the hardware used (50% is typical).
Asynchronous	Either of the above options can be used in an asynchronous mode. Using this option the client continues to stream packets to the server without waiting for ACKs, but rather, checking that the server has received the correct data at checkpoints.

Load balancing is essentially carried out on a round-robin basis, though the algorithm can also takes the following factors into account:

Factor	Notes
Disk space remaining	Nodes with the most disk space remaining are preferred to those that are almost full.
Recent activity	To a lesser extent, load is spread so that the entire cluster is used (not just the most empty nodes).

Data will always be stored in two separate nodes. Thus, should a node go down, the data will be available for reading from the second location. The user can however select that a vault keeps only a single instance of data. If that is the case then once the data received has been verified (ie, read back from disk) then one instance of the data will be removed.

Note that since data is kept in two locations if massive multiple user read access is required then a read cache should be used in front of the storage. This is typical in VoD (Video on Demand) type services where an individual film may be being watched by 10,000's of end users.

When data is sent through the API to be stored, it is sent together with metadata and policy information. This information is stored together with the data to form an object instance.

Data is stored in a non-proprietary format on the disks.

6.11

Performance

Unlike file systems which typically go through metadata controllers (which then become the bottleneck), via TCP/IP connections, one client writing data to one MatrixStore node will not affect the performance of another client writing data to another MatrixStore node.

Latest performance figures are available on request.

6.12

Object Reading

As with writing data to MatrixStore, all read operations pass through the MatrixStore API. The MatrixStore API enforces that the connection is made with the correct security credentials and then that the transmission of data is both checksummed and secure.

The following steps take place during reading data:

1. Client initiates the construction of a secure connection to the cluster.
2. Request is sent to obtain the locations from which to read the object from.
3. Client reads data from one of those locations using a direct IP link to the node.

The cluster load balances reads using a simple random selector to select from which node the client should read the data.

6.13

Object Identification

Every item of data stored is stored as an object. Every object is given an unique ID by MatrixStore. The ID is a 256 bit unique identifier (in an ASCII printable format).

When an object is stored, and the ID returned, the client may select to store the ID in a database to be able to retrieve the object at a later date. However, it is not necessary to do so if enough metadata has been attached to the object to allow it to be identified via that means. E.g., the storer may wish to attach its own ID to the object and use that as the means to retrieve the object. Whilst requiring an extra round trip between the client and the server to start reading the object (the first round-trip being to perform a search, the second to retrieve the object), the high speed of searching means that this is still an excellent way to retrieve data.

6.14

Quality of Service

SNIA define QoS as “A technique for managing computer system resources such as bandwidth” … “Policy rules are used to describe the operation of network elements to make these guarantees.” … “RSVP allows for the reservation of bandwidth in advance”.

MatrixStore does not provide QoS electing rather to serve data as quickly as possible rather than to guarantee a steady stream of traffic.

MatrixStore should not be seen as a playout server, rather, it can provide the playout server with all the data it requires. For example, a data centre topology may have:

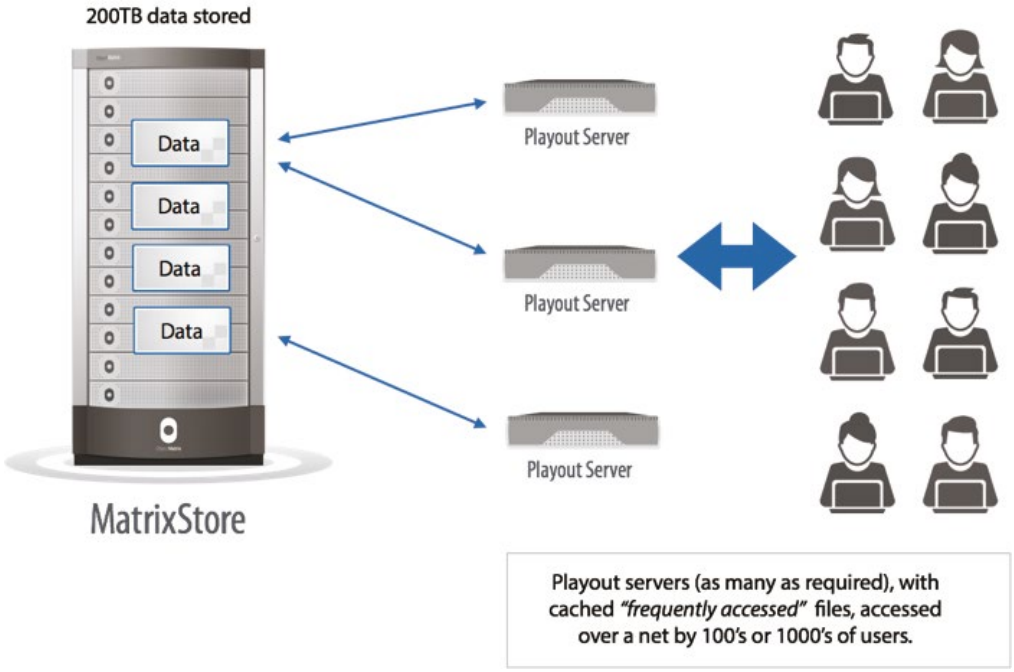


Figure 12 - Quality of service diagram

6.15

Security Overview

From the inside out MatrixStore was built with security in mind:

- Security is included “out of the box” without requirement for specialist training or knowledge.
- Security is on the logins, data transfer, and on the node firewalls.
- Security is simple: because the only access allowed is via the APIs any security hole would have to exploit those APIs which is highly controllable.

6.15.1 Security – Firewall

The ports required by MatrixStore are 666, 667, 1907, and 1908. They use protocols designed for MatrixStore only, and communications over these ports are fully secured. Port 8443 is used by MatrixStore Vision.

The Linux operating system build has been stripped down to remove all applications from the operating system that could communicate over other ports. Since there is only a minimum of background processes that are running, the node is stable as a reliable long-term storage device.

When installing MatrixStore, it is optional to leave an ssh port open on the cluster. If left on, the system is only as secure as the password(s) on that port. It is therefore recommended not to leave an ssh port open. It is recommended to allow access to ssh for Object Matrix maintenance operations.

Note that if the company installing MatrixStore has a firewall installed in the data centre, then it must be adjusted to allow for traffic on the ports used within MatrixStore.

6.15.2 Security – Data Communications[1]

Every client that talks to the server uses the MatrixStore's API in that connection (important: see footnote). The connection sets up a PPK type connection, the sequence of events being:

1. Request and agree up to a 256-bit secure channel to communicate with the server.
2. Pass a 256bit password over this channel for access to the vault.
3. Transmit any data. Data is up to 256-bit encrypted.
4. On the handshake of each packet, agree the encryption code for the next packet.
5. On receipt of a packet, de-encrypt (against previous handshake).
6. (Until end of communication).

Thus,

1. All communication is encrypted and authenticated.
2. Replay attacks will not work (due to evolving keys).
3. Packet decryption over long time packet sniffing is difficult due to the usage of evolving keys.
4. Data can be safely sent over the Internet, and indeed the whole cluster can safely be placed on the internet.

To encrypt data the Helix algorithm is used. The security solution has been built to allow other algorithm's to be plugged in as required.

Whilst it is desirable to encrypt the traffic of data to and from the server, the act of doing so generates CPU overhead at both client and server sides. In situations where data does not need to encrypted, encryption may be switched off.

6.15.3 Security – User Types

Data is stored and accessed within logical constructs called "Vaults". A client with access rights to a Vault has access to that and only that Vault (i.e., unlike groups in UNIX type systems). Thus, a client will need to maintain separate access credentials if he/she wishes to access different Vaults on a MatrixStore archive.

Vaults can be organised as required, e.g., one vault could be for pre-production data, another for post-production data, another for customer-x, and yet another for the accounts department.

[1] MatrixStore version for export to restricted territories does not include these options.

Access credentials for a Vault can provide read, write, delete and/or search capabilities for that vault. Thus, a customer could be given read only credentials for a vault relating to the customer.

There are also several special pre-defined user roles that have special roles in MatrixStore clusters:

User Type	Notes
Cluster Admin	This user can run certain administration tasks on the cluster, can shutdown the cluster, and (v4.2) can create / modify / delete Spaces (Spaces are groups of Vaults and users – e.g., for when a cluster is used for multi-tenancy).
Space Admin	V4.2 – can administer a Space including the creation of Vaults within a space.
Vault Admin	Can configure/modify a vault's settings.

Within a vault users can:

Vault User	Notes
RWDSU	Be assigned read, write, delete, search and update metadata permissions.
Be assigned fine grained Capabilities	V4.2 e.g., capability to be able to be able to create a Vault form, capability to be able to download hi-res video in Vision etc.

6.15.4 Security – Viruses and Human Error

Most disk based archive or mass storage solutions are extremely vulnerable to viruses and/or human error. A virus with access to a volume could quite easily delete the content of an entire archive. Likewise, a human error, e.g., typing "rm –r *" at the wrong location, could easily lead to data loss. MatixStore is built to avoid such losses.

All communications to the cluster go via secure protocols. A virus would need to have intimate knowledge of the protocol and even then would need to have the right security credentials.

Furthermore, all data can be stored as immutable for a determined amount of time thus stopping data from being accidentally and/or deliberately deleted.

6.15.5 Security - Adding Nodes

To avoid rogue nodes trying to attach themselves to the cluster as a malicious entity or simply before the administrator is ready to start using the node, new node attachment to an existing cluster is a two stage process:

1. Install MatrixStore software on the new node, and physically attach the new node to the other nodes in the cluster.
2. Use the Maintenance tool from a client machine to add the node to the cluster. This requires a system administration password.

6.16

Maintenance – Storage Space Configuration

Neither a node nor the cluster should ever be allowed to run out of storage space.

At a cluster level, ideally, there should always maintain enough space into which to recover data should there be a single node failure. If there is less space remaining that this, the cluster has an “amber” status.

A cluster will always be considered full if there aren’t at least two nodes left with disk space available. At that stage the cluster will no longer accept data writes.

At a node level, the server will fill a node to 97% capacity. This allows space for log files, auditing, attribute changes and database space. A node that is full will no longer be selected for writes. Note that an admin can adjust this amount of headroom in the administration tool.

A rough formula for calculating whether the cluster goes to amber status (in a cluster with equally loaded, equally sized nodes) is:

> *If (space_remaining - space_remaining_on_one_ node) < (total_space * 97%) / num_nodes -> “amber”*

Or, where disks are empty:

> *% space available = n-1/n where n = number of nodes.*

Or, if there are 4 nodes keep 25% free. If there are 10 nodes, keep 10% free, etc.

MatrixStore server software contains a registration key license that also limits the amount of space available for use. This allows Object Matrix to freely distribute the software for trials.

Concerning adding storage capacity, the system is only tested for adding extra nodes – Object Matrix does not currently support changing the capacity available on a single node.

6.17

Server Tasks

On each node a number of background tasks run to check and maintain the integrity of the data on the cluster. These tasks include:

- Self-healing.
- Data replication.
- Data verification.
- Regulation compliance.

Tasks run in the server layer and can be controlled, to some degree, via the administration screens. E.g., some tasks may be paused, or forced to run immediately.

MatrixStore version 3.3 and newer allows the replication task to be scheduled, e.g., to run overnight.

6.17.1 Data Integrity and the Verify Object Task

When an object is stored to the cluster it is mirrored to two nodes. Transfer checksums verify that the correct data was received at each location and disk flushing is completed before the object ID is returned to client application, however there is a chance that there are underlying disk block corruptions (it is a known that RAID cards are not always able to detect nor recover from such situations, Robin Harris on StorageMojo has several interesting articles on this subject from in-the-field surveys). Therefore, when an object is stored it is listed to be verified.

The verify object task will, approx. 20 minutes after the object has been written, verify the object to see that the checksum matches the contents. Should a corruption be noticed, then the object will be moved to a corrupt objects folder, and a replacement will be retrieved from the other mirror copy of the data. If both versions are corrupt (which would seem extremely unlikely) then both copies are left untouched.

An interesting question is how often should objects be rechecked for correct integrity? Too often and the risk is that hardware failure may be invoked by constantly running disks, too long and corruptions may go unnoticed before both copies have become corrupt (or the good node has been replaced).

6.17.2 Self-Healing Tasks (Data Regeneration)

If a node containing a duplicate of the data is un-contactable for a set period of time then self-healing / automated data regeneration will be invoked. The period of time here is a critical factor: the longer the period of time, the more the cluster is in risk of a second failure that causes data the to potentially be permanently lost, the shorter the period the more likely undesirable regeneration will start to occur, e.g., if a node was accidentally powered down for a period of time. Typically the timeout period is set to 14 days on a raided disk solution.

Thus, should a node be offline for more than 14 days, the other nodes in the cluster will begin to regenerate any data that they share with that node. All nodes do this in unison, thus in a ten node cluster, on average the other 9 nodes will hold 1/9th of the data held on the down node. If the node

is fairly large and filled to 80% full – e.g., 40TB in size, then each other node will need to regenerate 40TB * 80% * 1/9th = approx. 3TB. If data is regenerated to a new location at 400MB/s then the regeneration period will be approx. 10 hours. This is astonishingly quick compared to just about every other clustered storage solution out there.

It is possible to stop automated data regeneration (e.g., if you know a node is going to be offline for a period of time). To do this, simply go to the task pane on the Administration tool, select and pause the Regeneration tasks.

6.18

Node Re-attachment (Version 2.4 onwards)

If a node that was regenerated is reattached to the cluster then the data that was regenerated elsewhere in the cluster, and that is on that reattached node, is moved to a "to-delete area". The administrator can empty that to-delete area via the administration console tasks screen.

6.19

Node Decommissioning (Version 2.4 onwards)

A node may be decommissioned because it is seen as going faulty or because it is simply being phased out. Whilst the node could simple be switched off and unplugged, it is less susceptible to risk to decommission the node from the administration console. (e.g., if the node is simply unplugged, then data could be lost if another node were to subsequently fail before regeneration took place). Also, if "Single Instance Vaults" (see below) are being used then it is very important indeed to decommission a node rather than just switching it off.

6.20

Single Instance Vaults (Version 2.4 onwards)

When a file is copied from the client on to the cluster two copies of that file are made at two separate locations, metadata and policy information are added, resulting in two object instances.

From MatrixStore v2.4 onwards the administrator can elect to make a vault a single instance vault, in which case the following will occur:

- If the vault is a replicated vault, both instances will be stored and verified. The object will then be replicated to the remote cluster, where another two instances of the object will be stored. At that point the source cluster knows that one instance of the object can be replaced by a stub. Should the good instance of the data be lost, then the stub will attempt to fetch the object from the remote cluster to recover the data.
- If the vault is not replicated then once the object has been verified as not corrupt, one instance will be replaced by a stub. Should the good node instance be lost then the data will be permanently lost.

This powerful store and forward functionality of MatrixStore allows up to 50% data space to be saved at any one cluster location. A vault may manually be changed from being a dual instance vault to a single instance vault at any time.

Note that the level of data protection with a single instance vault is strongly compromised. Should a node in the cluster become unavailable (e.g., via a local filesystem corruption), then data will become irrecoverable. Therefore data should only ever be kept "Single Instance" if there is another copy of the data available elsewhere, or if the data can afford to be lost.

6.21

Removing MatrixStore Software

An archive solution should store data in an open, and well-tested format; not in a format that is subject to frequent change (as tape formats are) and not one that depends on the support of an individual company.

Since MatrixStore stores data in an open format it is possible to remove the software, reboot the node and then to access the data directly.

To do this:

1. Switch off all the nodes.
2. One by one, load the desired operating system onto the nodes (e.g., reload Linux from DVD).
3. Mount the existing file systems.
4. Create a script/application to walk the file system and to collect the metadata for the files into a DB, spreadsheet, or other format of your choice. Example applications for this are included with MatrixStore / available from Object Matrix, these are documented and are included in the MatrixStore delivery.

Unlike many archive solutions, MatrixStore's philosophy is that the data is yours, and that you should not be irrevocably tied in to the solution.

6.22

Data Mirroring

Mirroring on a standard device usually involves making two copies of every piece of data stored, one on each volume (or side) of the Raid unit.

This method has several physical advantages for high speed throughput, but also has several disadvantages, including but not limited to:

- The volumes typically have to be of the same size.
- If the device is going to be rolled out, then all of the data needs to be moved off of it first, and then all of the indexes pointing to the data need to be changed to point to the new location of the data.
- If the device has a problem and needs to be replaced, then data may be lost.

MatrixStore mirroring is based upon object mirroring across independent nodes. When an object is stored in one location, it is simultaneously stored on at a second location.

Since mirroring is carried out at an object level nodes/storage volumes can be of any size.

Mirroring across nodes is not optimal in performance compared to using dedicated fast IO throughput hardware, but it is reliable and flexible.

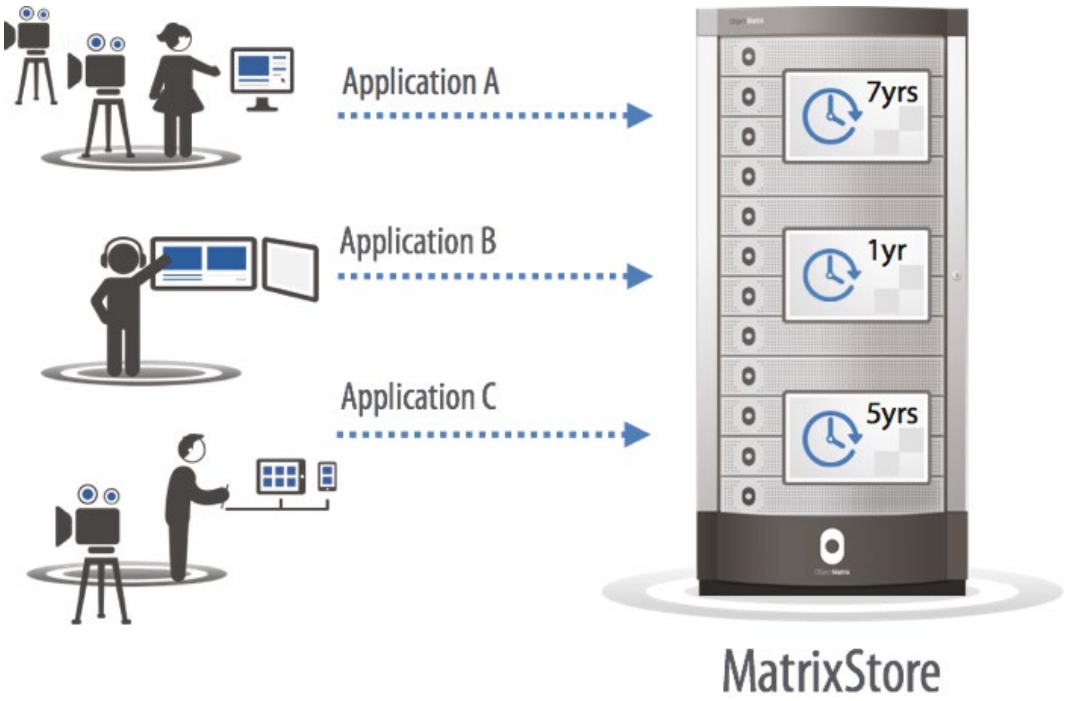


Figure 13 - Regular compliance diagram

6.23

Regulation Compliance

MatrixStore supports a large number of compliance requirements by ensuring that all data stored is secured from public access, protected against loss, audited, authenticated, available at all times and protected from unauthorised deletion.

MatrixStore helps to support, amongst others:

- Security and Exchange Commission Rule 17a.4 that aims to prevent overwriting, erasure or alteration of records.
- HIPAA privacy ruling for Data Protection, requiring compliant backup methodologies to ensure the security and confidentiality of patient records.

The Sarbanes-Oxley Act of 2002 protecting investors by improving the accuracy and reliability of corporate disclosures. The Act amends mail and wire fraud infractions with harsher punishments and imposes fines and prison sentences of up to 20 years for anyone who knowingly alters or destroys a record or document with the intent to obstruct an investigation.

Individual vaults of data can support different regulations.

Some of the ways that MatrixStore helps to support regulations are:

Regulation Compliance Requirement	Notes
Guaranteed retention of data for specific amount of time	Policy stored with data enforces that the data cannot be deleted before its time.
Audit logs and log files	Configurable audit log can track writes, reads, deletes. Log files track other system modifications.
WORM	Data is stored fixed cannot be modified.
Authentication	A 256bit digest calculated when the data is stored acts as a guarantee that the data is bitwise exactly the same when it is read back as when it was originally stored.
Security and Privacy	Full network security is employed to stop replay, sniffing, data modification and other attacks. User access rights can be set to only give access to certain groups of data.
Accessibility	Time to first byte is sub-second, even under heavy load.
Searchability	Built-in database can support many searches per second across 100's of millions of database entries.
Disaster recovery	Covered on two levels: by replication to a separate cluster and data is stored on a single cluster such that 4 disks would need to irrevocably fail before data is at risk.

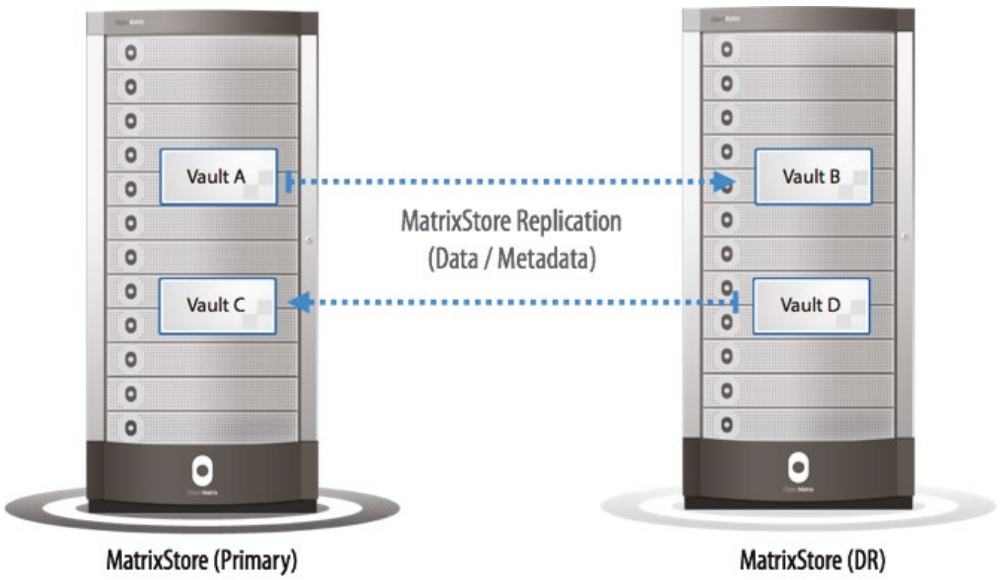


Figure 14 - Data replication diagram

6.24

Data Replication

It is sometimes desirable for purposes of data protection / disaster recovery (DR) / business continuance to ensure that data is stored in more than one geographic location.

Good replication solutions should provide the following attributes:

- Simple to set-up and use = less human or technical errors.
- Not tied to the underlying hardware = flexibility and serviceable in the future.
- Neither operating system specific nor data type specific.
- Works over standard transport protocols or VPNs.
- Secure data transmission options.
- Scalable performance.
- Easy to select data that needs replicating and data that doesn't.
- Ability to replicate deletes or not to.
- Facilitate high availability, e.g., by allowing Reads from either the source or target of the replication.
- Facilitate fast disaster recovery in the case that one site is lost, that work can immediately continue at the replicated site.
- Objects can be optionally stubbed after a certain amount of time has passed after replication.

MatrixStore supports and/or facilitates each of the above attributes. Its replication works on an asynchronous model of transmission to the second store. The replication is optionally carried out over a secure (encrypted) TCP/IP connection. Therefore, replication communication can be carried via a public internet provider or via a VPN / private dedicated connection.

Individual vaults within MatrixStore can be selected to be (or not to be) replicated. Thus, not all the data on a MatrixStore cluster needs to be replicated and two clusters do not need to be the same size as one another.

In this first example we show a set up where one vault is being replicated to another MatrixStore, which in turn is replicating another vault back to the first MatrixStore.

A cluster topology might simply contain two clusters, but also, a cluster may receive data from several other clusters so that, e.g., a central data repository could be created that receives data from many branch offices. E.g.,:

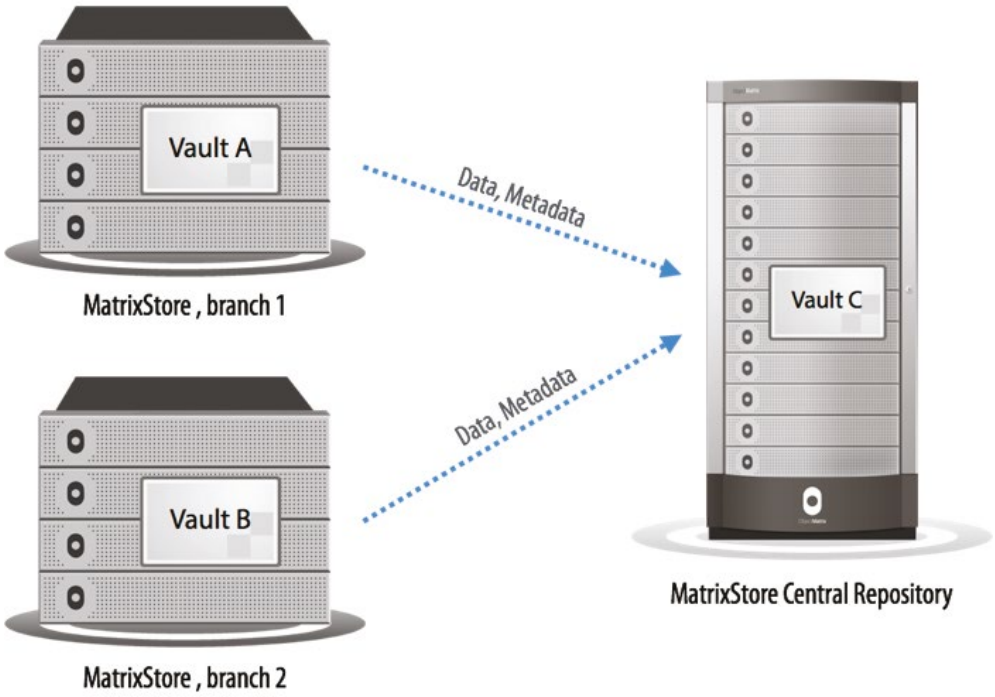


Figure 15 - Data replication diagram 2

The central repository vault can be attached to for reads.

Of course, replication could also be set to replicate to individual vaults at the central repository if separation of the data is required.

Other features are:

- Optionally deletes can be forwarded or not. Thus the Vault at the central repository can become a media library, whilst the original (e.g., branch) location can be kept small.

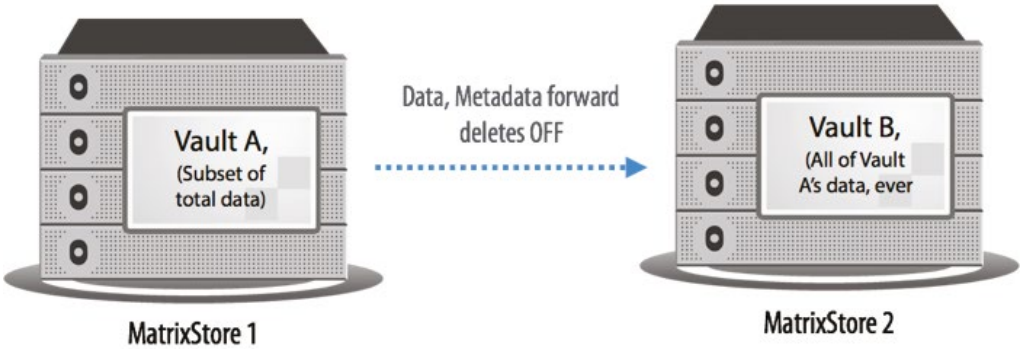


Figure 16 - Data replication diagram 3

- A vault can be relayed onwards, such that the data added to a vault can be kept on all MatrixStores.

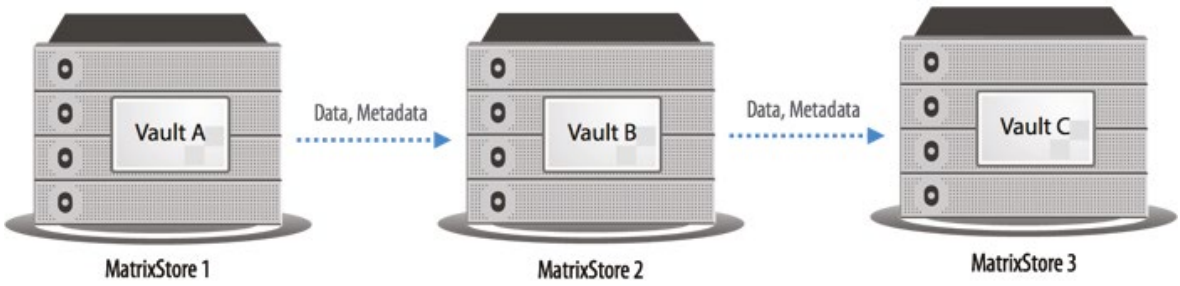


Figure 17 - Data replication diagram 4

- If the vault is single instance, two instances will be kept at the source cluster until one instance is effectively forwarded to the second cluster. Thus at no time will the user have a single instance of data. See also Single Instance Vaults, which ensure that only one copy of data per location is kept.

MatrixStore Replication Restrictions:

- Two way replication is not supported.
- For disaster recovery situations the users of Cluster 1 need to also be kept in Cluster 2.

6.25

Management API

The Management API:

- Enables commands that can update / effect the whole cluster.
- Ensures that commands reach all the nodes, and that settings are received by nodes that were temporarily offline when the command was issued.

The management API is REST based and includes commands such as adding vaults, changing users, getting audit logs, changing task settings, etc.

6.26

MatrixStore API

All communication with MatrixStore goes from the client to the server using MatrixStore's API. This enables us to maintain security, failover, cluster discovery, retries, transport protocol selection, and metadata control.

The MatrixStore API is available in C and Java.

Java is considered the superior API to use since:

- It has better performance.
- It supports updating an object's data.

Full documentation for programming the API is available upon request from Object Matrix.

6.27

Upgrading Software

Software is upgraded from the MatrixStore Maintenance tool. The tool will inject the software into all the online nodes of the cluster and will restart them to complete the upgrade.

6.28

Cluster Monitoring and SNMP

There are three ways to monitor the health status of the cluster:

- SNMP traps
- Via the MatrixStore WebAdmin
- Via MatrixStore Sense

A client must register the SNMP monitor via the MatrixStore MAPI in order to receive traps.

6.29

Updateable Objects (Version 2.4 onwards)

A Vault can be created or changed to be “updateable”. Unlike other objects in the cluster, objects in an updateable vault can be modified at any time.

This is critical for any application using random access (e.g., a filesystem interface) to write data to MatrixStore.

Primarily, updateable vaults enable filesystem front ends to be easily implemented.

If a vault is set to be both compliant and updateable, then updates are allowed only for a short amount of time (first 10 minutes).

7 Process-in-Place

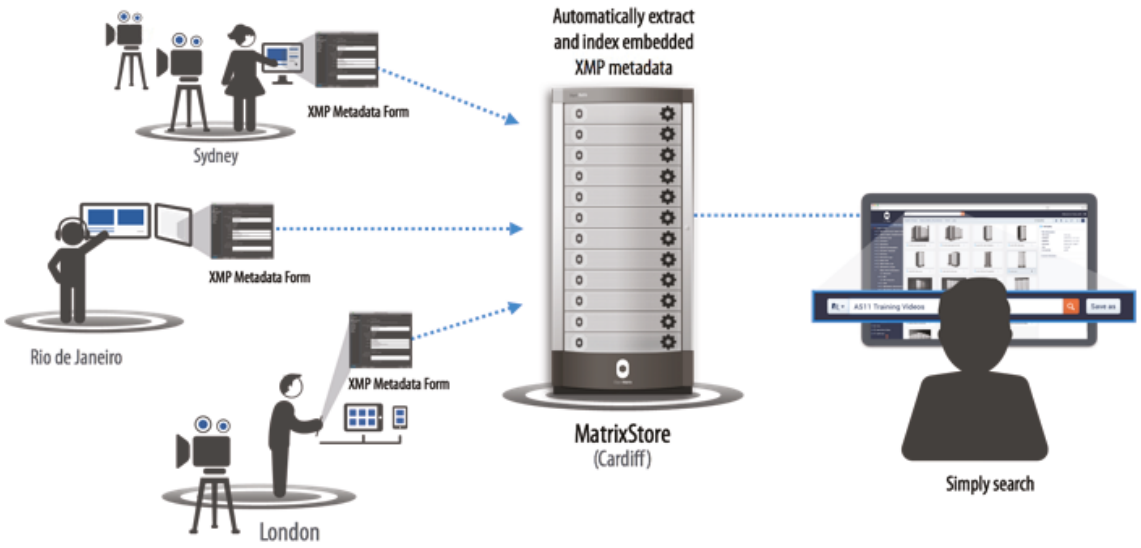


Figure 18 - Process in Place (PiP) for Adobe XMP diagrama

7.1

Overview

With every node having CPU power and fast direct access to its data it makes the perfect place to perform media related functions such as metadata extraction, transcode, applications and artificial intelligence. Version 3.2 of MatrixStore saw the first of this functionality with the implementation of metadata extraction from media assets. Future versions of MatrixStore will extend this core functionality into other areas mentioned.

7.2

Metadata Extraction

Many formats of content get delivered with important and valuable metadata built in. Having a storage solution that understands those formats and that can automatically index the metadata means people or further processes are not required to perform the task. MatrixStore currently supports AS11, EXIF and Adobe XMP metadata. Using XMP as an example: XMP files allow you to transfer your image metadata, ratings, tags, keywords, geolocation, and other attributes about the image outside of the actual JPG, CR2, or other file type. The XMP format was created by Adobe to standardize how metadata is stored and transferred and is a standard used by many news entities, photographers, photojournalists, image resellers, etc. The example shows:

- Media managers at remote locations add XMP metadata to the content they are creating.
- That content is sent to a central MatrixStore repository into a vault that is configured for XMP metadata extraction.
- MatrixStore PiP extracts the metadata automatically when the cluster is not performing digital preservation tasks. The metadata is stored as part of the object within MatrixStore and indexed to enable search operations.
- The data is not delayed for pre-processing nor moved for post-processing. It is processed in place, where it lives.

8 Index

8 Index