Object Matrix

MatrixStore Administration and Programming Guidelines

Version 1.9, February 2021

ObjectMatrix

Contents

1 About this guide

Every software solution has been designed with certain best usage practices in mind and MatrixStore is no exception to that rule. This guide is intended to provide advanced information on how to get the most out of a MatrixStore solution both from an administration point of view and from a programmer (using the MatrixStore API) point of view. Furthermore, the guide points to the future direction of MatrixStore and how to program / use MatrixStore in a way that will best take advantage of those future versions.

2 MatrixStore Concepts

2.1 Cluster

A MatrixStore cluster consists of 3 or more servers, called Nodes. Each Node contains disk storage and runs the MatrixStore server software, which transparently maintains the integrity of your data.

2.2 Spaces

Version 4.2 introduces the concept of Spaces. Spaces allow for multiple tenancy of a single cluster by partitioning users/groups and vaults.

Every cluster has a default space, which does not have a name, but any new spaces require their own unique name. This space name is required during authentication for any user/group that is not part of the default space.

It is not possible for users and vaults across different spaces to interact with each other.

2.3 Vaults

A vault is a managed and scalable object storage partition within the cluster. In practice this may be used like mounted drive or network share. Users and groups will not be able to access vaults unless they have been given permission to do so by a vault administrator.

2.4 Users

Users in MatrixStore are created via the Web Admin GUI. They can be created normally, existing only within the cluster, or they can be imported from an external LDAP directory.

- MatrixStore supports any number of users

- Each User gets its own log in credentials, and can be set with its own read, write, delete, search and attribute update permissions per vault

Users normally authenticate using a username and password. Users can also be authenticated via access keys, normally to allow software to perform operations on their behalf without the sharing of passwords.

Users authenticating by username and password need to also supply their space's name. Each application may have a separate mechanism for providing the space name but if one is not provided them it is assumed they are using the default space (empty string). Authentication by access key has no such requirement.

2.4.1 Service/API Users

A service user, also referred to as an API user, is a user in MatrixStore who only has permissions and access to the vaults they require and can only authenticate via access keys. This type of user should be used for applications where there is no user interaction such as a service. They are effectively the same as a standard user, however their ability to use a username and password is disabled. This can be done by simply not providing them at creation, or by disabling sign-in capability.

It is recommended that you keep to one service user per service rather than many services using one user.

2.5 Access Keys

Access Keys provide an alternative authentication method, which is more secure than username and password. They allow access to a single user account without the need for password sharing. Made up of an **ID** and **secret**, the ID obfuscates the original user and the long secret makes them virtually impossible to brute force.

Access Key example

```
id: 66bdfb8a-87b6-42b3-baf8-ddcc09a5abf7
secret: sU7kbyxFPQGFTXxDjVnSeBQ0iTBpa4YKmGDkMhmem7FfyAvM
```

Access Keys are designed for use by services and applications where human interaction is not required. A user can have many access keys and they can be revoked at any time.

2.6 Vault file credentials deprecation

Prior to MatrixStore version 4.0 authentication required user credentials that were stored in a *.vault* file.

A .vault file example

```
cluster=a018e24b-7a9a-1565-e2df-9bd7d04e4dcd
user=acbea855-40cd-11e9-bad9-ef5d28a1bef7
password=sir bean channel nose heavy
```

```
vault=ab64e4b0-40cd-11e9-bad9-ef5d58a9bef7
addresses=10.0.20.1, 10.0.20.2, 10.0.20.3
vaultName=Productions 2018
protocol=arpc
```

Starting from version 4.0 that type of authentication method has been deprecated and superseded by access key credentials. Going forward your software should either use interactive input to obtain username and password from a user, or allow configuration of an access key if it is a service.

At present, the older authentication method will still work with existing credentials and MatrixStore version 4 remains backwards compatible with older versions of the API. However, you are no longer be able create new *.vault* file credentials. Legacy vault file credentials are migrated during the upgrade from version 3.x to 4 and converted into access keys. This means you can update your MatrixStore cluster and continuing using your software without having to make any code changes.

Any software using the older authentication method should be updated as soon as possible to ensure future compatibility with the API and to capitalise on newer features, such as access to multiple vaults under a single access key.

2.7 Communication Protocols

When building a MatrixStore Connection as a client there are 4 RPC protocols to use:

- rpc – synchronous non-encrypted connections
- arpc – asynchronous non-encrypted connections
- srpc – encrypted transmission of packets
- sarpc – encrypted asynchronous connections

3 MatrixStore API Programming

3.1 DropSpot and MXFS Metadata Compatibility

Objects dropped into MatrixStore are not required to contain any metadata whatsoever, but in order for the objects to be viewed within DropSpot and/or MXFS certain metadata items must be added to the objects. The following table describes the attributes that are shared between MatrixStore applications and can be shared with third-party applications.

***M**/Mandatory: *=Mandatory, RO=Read Only*
***T**/Type: S=String, I=Integer, B=Boolean, L=Long*
***S**/Searchable: *=Yes.*

Name	M	T	S	Notes
MXFS_CREATION_TIME		L	*	Epoch, number of milliseconds since UTC Jan 01 1970, indicating when file has been created. DropSpot gets that attribute from the source file during copying it to the cluster. MXFS on Mac OSX and Linux sets that value to a current time during archive operation. MXFS on Windows has that value explicitly set by OS to original timestamp of the source file. Application should not change that value later on. If application modifies this attribute it should also modify MXFS_CREATIONYEAR, MXFS_CREATIONMONTH and MXFS_CREATIONDAY to appropriate values
MXFS_CREATIONYEAR		I	*	Year extracted from MXFS_CREATION_TIME e.g. 2013. It should be updated whenever application modifies MXFS_CREATION_TIME
MXFS_CREATIONMONTH		I	*	Month extracted from MXFS_CREATION_TIME. Values from 1 (Jan) to 12 (Dec). It should be updated whenever application modifies MXFS_CREATION_TIME
MXFS_CREATIONDAY		I	*	Day extracted from MXFS_CREATION_TIME. Values from 1 to 31. It should be updated whenever application modifies MXFS_CREATION_TIME

Name	M	T	S	Notes
MXFS_ARCHIVE_TIME	RO	L	*	Generated by the cluster (v3.0+) and should not be changed. Epoch, number of milliseconds since UTC Jan 01 1970, indicating when file has been archived to the cluster. Application should not change that value later on.
MXFS_ARCHYEAR	RO	I	*	Generated by the cluster (v3.0+) and should not be changed. Year extracted from MXFS_ARCHIVE_TIME. Essential for compatibility with searching by date in DropSpot. E.g., 2011
MXFS_ARCHMONTH	RO	I	*	Generated by the cluster (v3.0+) and should not be changed. Month extracted from MXFS_ARCHIVE_TIME. Essential for compatibility with searching by date in DropSpot. From 1 (=Jan) to 12 (=Dec).
MXFS_ARCHDAY	RO	I	*	Generated by the cluster and should not be changed. Day extracted from MXFS_ARCHIVE_TIME. Essential for compatibility with searching by date in DropSpot. From 1 to 31.
MXFS_MODIFICATION_TIME		L	*	Epoch, number of milliseconds since UTC Jan 01 1970, indicating when was the last modification of the content (not metadata) of an object.
MXFS_ACCESS_TIME		L	*	Epoch, number of milliseconds since UTC Jan 01 1970, indicating when was the last time content (not metadata) of the object was read or written
DPSP_WEEK_OF_YEAR		I	*	Not used.
MXFS_TIMESTAMP				Deprecated. Should not be used. Please use MXFS_CREATION_TIME, MXFS_MODIFICATION_TIME, MXFS_ACCESS_TIME instead.
DPSP_TIMESTAMP		L	*	Deprecated. See MXFS_TIMESTAMP

Name	M	T	S	Notes
MXFS_PARENTOID	*	S	*	The MatrixStore object ID for the parent of this object. If this is a “top level” object then the parent OID should be set to be an empty string.
MXFS_FILENAME	*	S	*	Filename of the object, e.g., “movie.mpg”
MXFS_FILEEXT		S	*	Filename extension if known. E.g., “Doc”, “XLS” etc. Aids searching and sorting in DropSpot.
MXFS_FILENAME_UPPER	*	S	*	Uppercase version of MXFS_FILENAME
MXFS_INTRASH	*	B	*	This value must be set. Unless the file is to be put in the trashcan the value should be false.
MXFS_CATEGORY	*	I	*	Helps DropSpot identify the kind of contents the object contains. For correct behaviour, it should always be set to be a Folder when known. Categories are: *Unknown* = 0; *Folder* = 1; *Movies* = 2; Video, AVI, MPEG *Music* = 3; Audio, MP3 *Documents* = 4; MS Office, PDF, Text *Images* = 5; Images, JPG, GIF… *Vaults* = 6; MatrixStore Vaults
MXFS_DESCRIPTION		S	*	A string containing the published attributes pertaining to the object being stored. Generally, this value does not need to be set.
MXFS_GENERATOR		S	*	Set a code representing the writing application. E.g.“MXFS”
MXFS_LINK_SRC		S	*	Where an object is a link file to another object, this should be set to the (real) object’s MatrixStore ID.

Name	M	T	S	Notes
MXFS_MIMETYPE		S	*	String mimetype – helps DropSpot “double click open” files correctly. Examples are: "application/octet-stream" (Default) "text/directory" (Folders) "application/msword", "application/pdf", "audio/mp3", "application/x-wav", "audio", "video", "image", "text"
MXFS_OWNERGID		I	*	Non essential. If known set to the ID of the group the file belongs to.
MXFS_PATH	*	S	*	The absolute path of the original object. This is mandatory to be compatible with S3Connect and is the equivalent of your Amazon S3 object key.
MXFS_SIZE	RO	L	*	Used to retrieve the size of the object.
DPSP_TIMEBASEDUID		S	*	Important: A GUID that is entered into the metadata when an object is being stored. This way, if communication is lost with the server at a point in between the object write stream being closed, and receiving back the object ID for the object being stored, then on reconnection with the server the application storing the object can search for the object based upon the timebased uid to see if in fact the object did finish writing. This can avoid duplicate writing of objects.
DPSP_DPSPUSEROID		S	*	The application user login name.
MXFS_USERNAME		S	*	In DropSpot this is the operating system login username. Where possible the original file user name should be preserved.
MXFS_COMPATIBLE		I	*	Integer denoting version number. Optional field currently set to “1” in DropSpot.
MXFS_ENCRYPTED		B		Not used.

3.2 Additional information regarding timestamp attributes

Since MatrixStore v3, on every object creation cluster generates automatically the following attributes:

- MXFS_CREATION_TIME, MXFS_MODIFICATION_TIME, MXFS_ACCESS_TIME, MXFS_ARCHIVE_TIME,
- MXFS_CREATIONYEAR, MXFS_CREATIONMONTH, MXFS_CREATIONDAY
- MXFS_ARCHYEAR, MXFS_ARCHMONTH, MXFS_ARCHDAY,

Attribute modification

- MXFS_CREATION_TIME, MXFS_MODIFICATION_TIME, MXFS_ACCESS_TIME can be changed by application and should be taken from timestamps of a source file if possible (Dropspot). MXFS is platform specific and most of the time only mtime of the source file (MXFS_MODIFICATION_TIME) will be preserved
- When application or API changes MXFS_CREATION_TIME it should also provide additional 3 MXFS_CREATIONxxx attributes for consistency
- MXFS_MODIFICATION_TIME, MXFS_ACCESS_TIME can be modified freely but only if application knows the details about source file, otherwise changing is not necessary
- MXFS_ARCHIVE_TIME, MXFS_ARCHYEAR, MXFS_ARCHMONTH, MXFS_ARCHDAY are read only. Applications should not try to update, remove them etc. In that scenario UNSUPPORTEDOPERATION error will be thrown. Also creating object with any of those attributes is blocked. API and applications (Dropspot etc) should comply. Those attributes will not be changed by the cluster.

3.3 Cross-platform file naming compatibility

MatrixStore does not prevent applications from specifying any character as part of a file or directory name since it is an object store. It also does not replace any characters since this may break the compatibility with that application or others.

However, as MXFS is available for many operating systems, use of special characters should be avoided in file and directory names, to ensure best cross-platform compatibility.

Special characters include file path separators, punctuation and parentheses, which are reserved characters in file and directory names on some operating systems.

For best cross-platform compatibility, the use of the following characters should be avoided:

- **File path separators** e.g. ":" (colon), "/" (forward-slash) and "\" (back-slash), which are used as path separators on some platforms. Consider substituting these with "_" (underscore) or "-" (hyphen).
- **Punctuation, parentheses, quotation, brackets and operators** e.g. . , [ ] { } ( ) ! ; " ' * ? < > | as these are often reserved or used for special functions when scripting / programming.
- **White space characters** e.g. spaces, tabs, new lines and embedded returns. Although supported on some platforms, scripts and applications may not handle them as expected. Consider substituting these with "_" (underscore) or "-" (hyphen).

In addition, below are some platform specific requirements.

3.3.1 Windows

Windows has its own specific naming requirements detailed on the below link

https://msdn.microsoft.com/en-gb/library/windows/desktop/aa365247(v=vs.85).aspx

If you wish for files and directories to behave as expected in MXFS on Windows, the following should be considered:

- The filename cannot end with a space.
- The filename cannot end with a period.
- Names are case-insensitive.
- Avoid reserved characters including < > : / \ | ? * "
- Avoid reserved names (detailed in above link).

In addition, MXFS requires that filenames should be less than 128 characters in length.

3.3.2 MacOS / OS X

MacOS / OS X has the following requirements

- "/" (forward-slash) is not allowed in file and directory names (Finder will let you but will write a ":" in it's place).
- ":" (colon) although allowed, should be avoided as it's display will vary depending on the application used, e.g. terminal will show ":", but Finder will display "/".

3.3.3 Unix

Unix platforms have the following requirements:

- "/" (forward-slash) and the NULL ("\0") character are not allowed in file and directory names.

3.4 Searchable keywords in DropSpot

From MatrixStore v2.4, all stand-alone keywords that are required to be searchable from DropSpot need to be entered with the keyword preceded by the string “_fs_”, therefore: key="_fs_" + word you want to search. Keys should always be in lower case.
value=""

As an example: Superman movie.

File:
superman.mov

Synopsis:
A strong man

If you want DropSpot and MXFS to see the file, add the mandatory attributes in the table and any others that are known.

- If you want DropSpot to find the file searching for simple string "superman", add he attribute:
key="_fs_superman"; value=""

- If you want DropSpot to find the file searching for simple string "mov", add he attribute:
key="_fs_mov"; value=""

- If you want DropSpot to find the file searching for simple string "movie", add he attribute:
key="_fs_movie"; value=""

- If you want DropSpot to find the file searching for the string "category=action", add he attribute:
key="category"; value="action"

With the above attributes, if you search for "movie superman" in DropSpot, it will construct an AND term searching for 2 attributes:
_fs_superman="" and _fs_movie="", finding the file as well.

As a minimum, the following attributes could be added to the file from the table:
MXFS_PARENTOID=string("")
MXFS_INTRASH=boolean(false)
MXFS_FILENAME=string("superman.mov")
MXFS_FILENAME_UPPER=string("SUPERMAN.MOV")
MXFS_CATEGORY=int(0)
MXFS_PATH=string("/superman.mov")

However, then as many other attributes as possible should be added.

4 System Metadata Attributes

MatrixStore provides access to system attributes in READ ONLY mode providing information to applications. The list is described below:

Name	Notes
__mxs__locked	Returns if the object is locked
__mxs__inCompliantStore*	Returns if the vault has regulation compliance switched on
__mxs__storesCurrentRetentionPeriod*	Returns the period of time an object will be locked from deletion / updating for.
__mxs__clustersCurrentTime*	Returns the current time in the cluster
__mxs__creationTime	Returns the creation time of the object. Returned as an 8 byte long.
__mxs__modifiedTime	Returns the last modified time of the object. Returned as an 8 byte long.
__mxs__length	Returns the length of the object. Returned as an 8 byte long.
__mxs__calc_adler32	Calculates the adler32 and returns the result. Returned as an 8 byte long. If the return value is -1 or -2 then the value is being calculated, check back again for the calculated value a few seconds later.
__mxs__calc_md5	Calculates the MD5 of the object and returns the result. Returned as a byte array. If the return value is an empty array then the value is being calculated, check back again for the calculated value a few seconds later.
__mxs__vaultquotabytes*	Returns the maximum capacity allowed of the vault. Returned as an 8 byte long.
__mxs__vaultspaceusedbytes*	Returns the current space used by the vault. Returned as an 8 byte long.
__mxs__userHasTag=*	Returns if the current user has been tagged with the given key. As of server version 2.4: “dropspot admin” is supported as a tag.
__mxs__userCapabilities*	Returns the current user’s access capabilities. Returns a string composed of “R” “W” “D” “S” and “U” as appropriate.
__mxs__online	Since v3. It returns if the object is online, e.g. it has not been stubbed and the data is accessible.

* When searching on an attribute to return one of these values, use object ID “00000000-0000-0000-0000-000000000001-1”

To retrieve one of the listed values perform a get metadata attribute API command on the appropriate object with the listed key.