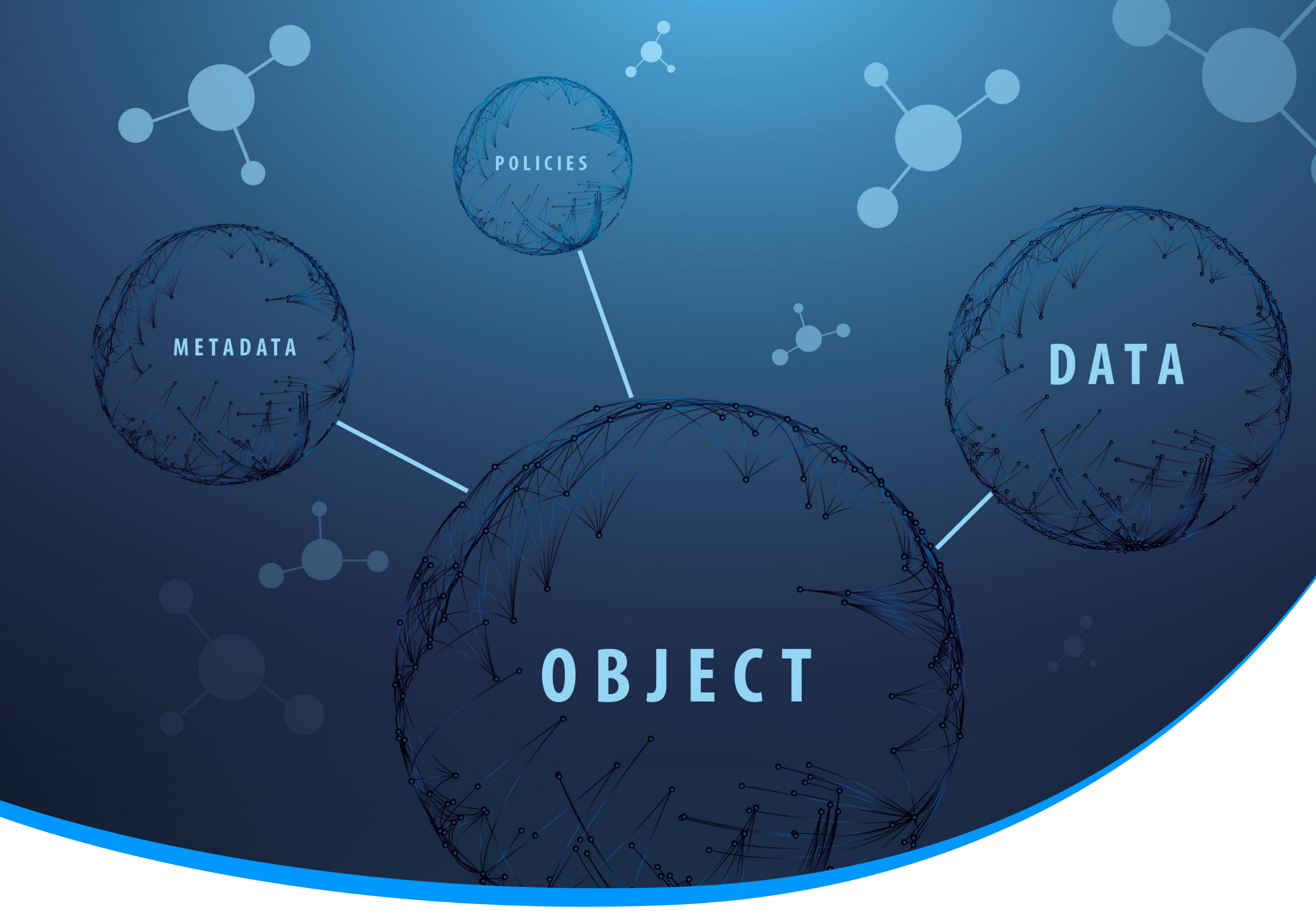

ObjectMatrix

What is Object Storage?

What is object storage? How does object storage vs file system compare? When should object storage be used? This short paper looks at the technical side of why object storage is often a better building block for storage platforms than file systems are.

Object Matrix Ltd
Experts in Digital Content Governance & Object Storage

www.object-matrix.com
info@object-matrix.com
+44(0)2920 382 308

The Rise of Object Storage

Centera the trail blazer…

What exactly Object Storage is made of will be discussed later; its benefits and its limitations included. But first of all a brief history of the rise of Object Storage:
Concepts around object storage can be dated back to the 1980's[1] , but it wasn't until around 2002 when EMC launched Centera to the world – a Content Addressable Storage product[2] - that there was an object storage product for the world in general[3].

However, whilst Centera sold well – some sources say over 600PB were sold – there were fundamental issues with the product. Companies railed against having to use a *"proprietary API"* for data access and a simple search on a search engine shows that Centera had plenty of complaints about its performance. It wasn't long until the industry was calling time on Centera and its *"content addressable storage"* (CAS) version of object storage: not only that, but it had single handedly given object storage a bad name. Articles such as *"Centera, end of an era or end of an error?"* abounded – it was fashionable in large companies to cling on to the past. But the pronounced end just didn't happen[4].

In, 2005 I had a meeting with a *"next generation guru"* of a top 3 storage company, and he boldly told me: *"There is no place for Object Storage. Everything you can do on object storage can be done in the filesystem. No one wants to use APIs."* Funny how the largest storage company in the world could now be argued to be Amazon rather than one of the traditional players…

Object Storage Post Centera…

Need is a great leveller and perceptions have now changed. Mass cloud storage with required demands on performance, distribution, metadata handling and notably scale went to a level never previously seen because of the Internet. Billions of people wanting to access the same resources such as Facebook or Google created unique problem sets. Those companies and many like them needed to be able to store more data than could possibly be kept within a filesystem and needed a truly scalable solution quickly. Amazon created an internal object storage system for their own purposes and notably, in 2006 they turned that into their S3 cloud storage solution. S3, Google and many other players were all turning to object storage and the world's population were now using object storage, even if they didn't have a clue what it was!

Years of blinkered belief in the *"filesystem fits all"* were over and fast track to today; Object Storage is well and truly accepted as a better way to store and access data, in many many use cases, over the once ubiquitous file system.
And yet, whilst use cases for file and block storage are well understood, object storage remains a concept that is confusing to many and often misunderstood. Is it only for large internet based solutions? Can you search in an object storage? Are all object storage systems alike?

Block and File Storage

Think of data storage and most people think of a filesystem. Within that is a hierarchy of files wherein you start from a top directory and drill down through the directories to the file that is required.

How the computer sees a filesystem is important to understand the benefits and problems. Under the hoods everything on a HDD (file system or otherwise) is in small blocks of bytes, e.g., 4096 bytes. The files we see are often made up of many blocks of bytes[5].

---

[1] *https://en.wikipedia.org/wiki/Object_storage*
[2] *https://en.wikipedia.org/wiki/Content-addressable_storage*
[3] *The author of this document, Jonathan Morgan, worked for FilePool before it became a part of EMC*
[4] *It was around this time, 2003, that Object Matrix bucked the trend, developing MatrixStore;*
[5] *One enterprise implementation of this is block storage.*

The *“filesystem”* is the software that can read, write and interpret those blocks of data to allow people to store and read files. The file system knows which blocks make up a file because it keeps a list of the order and locations of the blocks. Sometimes the filesystem keeps those lists of blocks in what is termed a *“metadata server”*. The metadata servers keep not only what blocks make up a file, but also the hierarchy of the files, the file names and other pieces of metadata about the files such as when the file was last accessed.

Where filesystems are great is:

- They are well understood; around for decades, most software understands filesystems
- Sharing files between a group of computers since the filesystem protocols exist on many clients
- Performance within a predictable network – they lock and unlock files efficiently and provide almost direct block level access

But filesystems just aren’t made with the building blocks that are required when it comes to the demands of large scale, long term or highly flexible data storage such as:

- Filesystems don’t general handle descriptive metadata[6] at all (or at least in a search context). Where they do, it tends to be very proprietary.
- Their view of the world is a top-down hierarchy – this is very inflexible (“did I store that file as *“weddings/july”* or *“july/weddings”* etc).
- Filesystems have trouble with scale beyond certain size: this is caused by the lock manager, the metadata database, highly-coupled nodes in scale-out solutions, etc.
- Filesystems often have expectations around speed of reply (timeouts) and therefore don’t scale well in high latency networks (internet etc).
- At worst, even in local systems this makes filesystems highly dependent on the underlying hardware being of very similar speeds, creating issues with future upgrades of hardware.
- Applications have to decide where in the hierarchy to put the files.
- File systems, being well understood, having little in the way of authentication, often only need a single *“rouge client”* to be extremely susceptible to malicious or accidental data lost.

One spin off issue from file systems being very bad at scaling (especially with different hardware over time) is that organisations often end up with multiple individual file systems / hardware solutions (*“data silos”*) rather than a single storage pool.

However, that said many solution stacks are built on top of filesystems, taking advantage of filesystems well-known behaviours and wide support.

Object Storage

Let’s define object storage.

Universal Object Storage Characteristics

Common amongst most object storage systems are that they store **objects**:

- Objects are unstructured – they do not inherently have relationship to one another, e.g., are not immediately arranged into directories or other hierarchies
- Objects are simply identified with a GUID (globally unique ID)
- Objects consist of metadata and data (typically a file, but it could also be any record of information)

The object storage system itself then:

- Provides a space and an API wherein and whereby objects can be stored and retrieved
- Will often apply storage policies, e.g., to distribute objects across multiple geographic locations

This *“building block”* provides the following benefits over the filesystem approach:

- Freedom from the constraints of metadata controllers
- Freedom from the constraints of a fixed data hierarchy
- The possibility to build highly scalable and flexible implementations
- A focus on using an *“API”* that includes the storage and usage of metadata
- A freedom for the object storage system to arrange objects on multiple servers, e.g., across multiple geographies

[6] *https://en.wikipedia.org/wiki/Metadata*

The main point is this: the filesystem is a structural constraint and an overhead and by storing data as simple objects via an API the solution is now free to build a vast array of more flexible and powerful structures for handling data. We'll look at key drivers for its adoption later.

Prevalent Object Storage Characteristics

Features often found in object storage, but not always, include:

- ✓ Automated maintenance of data storage policies – e.g., minimum data redundancy levels of objects
- ✓ A searchable distributed metadata database
- ✓ Compliance Regulation Policy features such as audit trails, security features, time based restrictions
- ✓ Scalability from Gigabytes to Exabytes
- ✓ Hardware independence
- ✓ Single namespace, even across multiple geographies
- ✓ Automated local caching of popular data
- ✓ Cluster self-healing (e.g., where a single location is down)
- ✓ APIs including UDP, TCP/IP, RCP, RESTful, SOAP types
- ✓ Replication
- ✓ Object level security
- ✓ Multi-tenancy
- ✓ Highly scalable aggregate bandwidths (as the object storage system grows, so does the aggregate bandwidth)
- ✓ Data analytic tools and management APIs
- ✓ Authenticated and check-summed storage and delivery of data

These are just a few of the features that often exist, but the reality is, at the end of the day every object storage solution has its own feature set, strengths and trade-offs.

2003, Object Matrix developed its product MatrixStore to address all of those features but with a focus upon speed of access / updating of objects thereby allowing natural filesystem type browsing and updating of objects (a business focus) over and above features that are focused on worldwide distribution of content (a B2C focus). Additionally Object Matrix added features for regulation compliance, deep data analysis and media industry features that understand media data and media plugins.

Is Object Storage All the Same?

Gartner, March 2016 published10:

> *Object storage is characterized by access through RESTful interfaces via a standard Internet Protocol (IP), such as HTTP, that have granular, object-level security and rich metadata that can be tagged to it. Object storage products are available in a variety of deployment models - virtual appliances, managed hosting, purpose-built hardware appliances or software that can be installed on standard server hardware. These products are capable of huge scale in capacity … They are better-suited to workloads that require high bandwidth than transactional workloads that demand high IO/s and low latency.*

The above is an interesting definition, born out of the *"norm"* of how analysts currently see the marketplace, but it most certainly a very limited description of object storage. There isn't a reason why transactional workloads should be slower on object storage than via a filesystem nor is there an intrinsic reason why object storage should have high latencies[7]. And, just because many object storage solutions have gone down the path of RESTful interfaces, wide area data distribution algorithms and rich metadata filing, doesn't mean that all object storage solutions have to go down those routes.

However, perhaps this demonstrates just how far object storage has come. There are different categories of object storage that are fit for different purposes. You wouldn't use Amazon S3 for transactional workflows and you wouldn't use Object Matrix MatrixStore for B2C workflows where the *"C"* could be a million people all wanting to access the same object at the same time.

[7] *An example is that MatrixStore from Object Matrix that is built for low latencies in many of its workflows*

Multiple Instances vs Splitting Data

One major difference between how Object Storage is implemented by different manufacturers is whether individual objects stored should be *"sliced"* across multiple nodes or kept in their entirety: multiple instances.

Data slicing algorithms can conceptually be thought about as something like a *"RAID6"* algorithm where the object is split into data and parity slices and each slice is kept on a separate node (this isn't a strict definition but it is helpful if you understand RAID). In fact, more modern algorithms based on Reed-Solomon algorithms allow variable numbers of *"parity"* blocks to be kept – one such vendor, CleverSafe, talks about an "*m+n"*, *"10+5"* algorithm where if even 5 locations were down then the data could still be read from the other 10 locations.

Other vendors, including Object Matrix, use a multiple instance algorithm that puts the individual instances at more than one location. With Object Matrix each location is typically RAID6 meaning that with 2 instances 6 disks would have to simultaneously fail before data couldn't be read.

While seeming like a fairly *"technical"* point the choice between algorithms has far reaching consequences[8].

Feature	Multiple Instances (MI)	M+N data distribution (M+N Erasure Codes)	Winner?
Minimum number of nodes	Typically 2 nodes only.	Normally from 8 to 15 nodes (m+n = 6+2 or 10+5). A m+n 2+1 is possible but means just 2 disks going down could result in data loss in some implementations.	MI: Can start with smaller systems. In the worst case M+N might require 15 separate nodes just to start with.
Single Location & Physical disk space required overhead	100% if 2 instances of the data are kept – could be 120% with RAID6 etc.	A 10+5 algorithm creates only a 50% overhead.	M+N or MI: Clearly M+N has a lower total disk space overhead at a single site to keeping multiple instances but only if data is kept at a single geographic location. (see *"Replication"*)
Replication & physical disk space overhead	If 1 instance at or MI each geography – then just the overhead of e.g., RAID6 at each geography.	Possibly 10+5 at each geography.	MI: Where replication is used typically MI will use less disk space.
Data Analytics	Each instance can be read and analysed. For Object Matrix this can be done within the node with out needing to read the data out to a "client" machine and every node can analyse its data simultaneously.	To analyse data, it must be rejoined, probably at a client machine.	MI: Clearly analytics are possible in both solutions but where it can be done within a node then it is hugely beneficial in terms of system load.
Data Writing / Reading. Data throughput	Data can be streamed direct to/from a single location, which can simultaneously stream to a second location.	Data must be split into multiple locations this can be done as a background task after data has written but then there is a period when data isn't protected at the suggested level. Alternatively it can be done in the client or as it arrives – both requiring CPU / creating latency.	MI: Although both systems have their advantages and disadvantages, MI typically has less overhead. Where M+N can sometimes win is where data is streamed from multiple locations to a single location during reads (thus taking advantage of more hardware).
Encryption and Data Security	Data can be stored encrypted, but typically all data is kept in a single location.	Data parts can be kept in geographically separate locations. Whilst this is impractical in terms of speed when joining up the parts to read the object, it does have the advantage of scattering the object across multiple locations.	M+N? Although at the cost of distributed reads otherwise equal.

[8] *This is not unlike Object Storage as a building block – the foundation changes are subtle but the impact becomes large as the systems become larger.*

Object Storage Weaknesses

One concern with Object Storage is that by using a proprietary API vendor Lock-in can start to occur.

However, to counter-act that, where object storage can be mounted using FTP or filesystem interfaces then it can be argued that the vendor lock-in is limited to just the metadata. Some manufacturers are pushing the S3 API as a type of defacto *"standard"*, though that API is not universally loved (it is truly complex and has very limited metadata manipulation) and cannot perform many of the features that are available on some other object storage solutions such as analytics or metadata searching.

Key Drivers for Object Storage Adoption

Key drivers for object storage adoption can include any of the follow:

- ✓ A desire for a deeper protection of data stored
  - Using: authenticated delivery, automated data redundancy policies etc
- ✓ Wishing to unify a large pool of storage, e.g., as a *"private cloud"*
  - Using: scalability; an ability to virtualise hardware so that new hardware can be added to the existing storage pool without throwing out the old hardware, etc
- ✓ Wishing to share and utilise metadata without having to create separate databases that are difficult to keep in sync with the data being stored, e.g., between applications
- ✓ A desire to analyse data, especially unstructured data
- ✓ Requirement to reduce management overhead of storing, maintaining and making available petabytes of data
- ✓ Security concerns about storing data in a filesystem
- ✓ Building a solution that can span multiple tiers of storage including public cloud
- ✓ Regulation compliance requirements

There are of course many more drivers that could be mentioned besides.

Conclusion

Object storage has come a long way from an ostracised/niche solution to storing vast amounts of the world's data. Its popularity is only growing and whilst the filesystem is here to stay, it is by no means the only kid on the block anymore (pun intended!).

Within itself *"Object Storage"* is actually a very simple building brick but because it is a powerful concept often complex, feature rich solutions have been built to preserve and distribute data.

As so often happens with a technology that grows in popularity there are likely to be schisms in the term "object storage" as time goes on: object storage for public cloud, object storage for fast data access and object storage for private cloud are all functionally quite different from each other and rarely compete with each other.

Lastly, object storage solutions can likely offer every major organisation that stores a quantity of unstructured data a better, safer, more future proof and more integrated with other applications way to keep their data in a truly accessible and reusable manner.

About the Author

Jonathan Morgan researched Grid Computing at Texas Christian University in the 1990's. In his work career he joined FilePool briefly before it was acquired by EMC for its object storage technology. That product went on to become *"Centera"*, one of the world's first commercially successful object storage solutions. At EMC Jonathan led the largest development team for Centera developing *"content parity protection"* (an erasure code algorithm). Founding Object Matrix in 2003, Jonathan has been the CEO since its inception seeing the company grow from a concept to storing data for some of the world's largest media organisations. Object Matrix is based in Cardiff, UK.

Object Matrix Ltd

To learn more about Object Storage, please visit
www.object-matrix.com or contact us to **info@object-matrix.com | +44(0)2920 382 308**