MINIO

The Architect's Guide to Software Defined Object Storage

WHITE PAPER

Introduction

Data is the currency that powers the modern enterprise. The ability to leverage that data on behalf of the organization's diverse stakeholders is a function of modern, cloud-native, performant and cost effective systems. Underscoring these modernization efforts is a single persistent theme - enabling the enterprise to be better at serving their customers.

That goal may be singular but has many paths. It can be developer agility, capex savings, opex savings, resiliency or performance. All of these paths will intersect with the data strategy.

This document outlines the considerations an enterprise should make when evaluating software defined storage. While some organizations will continue to opt for appliance based solutions, this is not the cloud-native way and will ultimately restrict the types of use cases, the deployment scenarios and the developer traction.

The guide starts with object storage, its myths, how it compares to file and block, its relationship to the cloud and Kubernetes and why software-defined matters.

It then turns its attention to the critical selection criteria for object storage with a focus on software defined object storage.

What is Object Storage

Object storage is a type of data storage that stores data immutably as distinct units, called objects. Objects are discrete units of data that are stored in a namespace that can be flat or can contain the concept of "folders" or "buckets" allowing them to be managed as groups. Each object is a simple, self-contained repository that includes the data, metadata (descriptive information associated with an object), and a unique identifying ID number. Metadata can be written atomically with the contents of the object or can be stored externally to the object and the object is provided with a unique identifier. This information enables an application to locate and access the object.

Object storage is known for its unlimited scale. Object storage is scaled through building blocks called storage pools. Pools can be combined and distributed across the network - effectively infinitely. This distributed model is particularly relevant from a resilience and disaster recovery perspective.

Unlike other storage approaches there is no requirement for data stored in object storage to fit into rows and columns. This type of data is referred to as unstructured data and represents 80% of the enterprise's overall data volume. This can include a broad range of data including email, videos, photos, web pages, audio files, sensor data, and other types of media and web content (textual or non-textual). This may also include streaming data and time series data.

Objects (data) in an object-storage system are accessed via Application Programming Interfaces (APIs), the defacto standard for which is Amazon Web Service's S3. The native API for object storage is an HTTP-based RESTful API (also known as a RESTful Web service). These APIs query an object's metadata to locate the desired object (data) via the Internet from anywhere, on any device. Because of this, object storage is inherently distributed and can be called through https.

Another way to think about modern object storage is as a key value store not dissimilar to modern cloud databases.

The Myths of Object Storage

Modern object storage is quite different from the traditional systems developed prior to the cloud-native movement. Many of the myths associated with object storage are no longer true or relevant. Here are a few:

1. Object Storage is slow.
Modern object storage is very performant. In general, it can be expected to perform at the top range of the hardware and very often to saturate that hardware. For HDDs that will be the drives. For NVMe, that will be the network. For well configured, 100 GBe, NVMe setups, object storage can exceed 365 GiB/s on READ and 145 GiB/s on WRITE.

2. Object storage is for archival workloads.
While object storage is superior for archival workloads, it is, given its scalability and performance characteristics, well suited for AI/ML, application, analytics and even database workloads. This is evident in the cloud where core services like Snowflake, Redshift and BigQuery are all run on object storage.

3. Object storage cannot handle small objects.
Correctly architected object storage systems are very effective at object sizes in the KB range. This requires an atomic metadata approach.

4. Object storage is poor fit for latency oriented use cases.
While object storage excels at throughput, again, properly architected object stores are also very effective at IOPS-powered workloads.

Object storage vs. file storage vs. block storage

Now that we understand object storage, how does it compare to file and block?

File storage stores data as a single piece of information in a folder to help organize it among other data. This is also called hierarchical storage, imitating the way that paper files are stored. When you need access to data, your computer system needs to know the path to find it. Filers do not scale particularly well over the Internet and thus they generally hit their ceiling at a few PBs of data.

Block storage takes a file apart into singular blocks of data and then stores these blocks as separate pieces of data. Each piece of data has a different address, so they don't need to be stored in a file structure. Because block storage doesn’t rely on a single path to data it can be retrieved quickly. It works well with enterprises performing big transactions and those that deploy huge databases. Block storage can be expensive and has limited capability to handle metadata, which means it needs to be dealt with at the application or database level.

	Object	Block	File
Price	$	$$$	$$
Data Type	Semi and Unstructured Data	Structured Data	Structured, Semi Structured and Unstructured
Scale	Massive Scale (starts small from TBs and grow to EBs seamlessly	Very few TBs	TBs to PBs
Protocols	S3 API	Block Protocols (iSCSI)	CIFS, NFS
Workloads	AI/ML, Big Data, Streaming, Database, Snapshots, Backup	Transactional DB	Archive, User generated files
Approach	Object = File + Metadata + Globally unique identifier	File is written on block on spinning media	File is stored and limited metadata is stored along as a separate entry

Enterprises opting for object storage can expect the following benefits:

Greater data analytics. Object storage is driven by metadata, and with this level of classification for every piece of data, the opportunity for analysis is far greater.

Infinite scalability. Keep adding data, forever. There's no limit.

Faster data retrieval. Due to the categorization structure of object storage, and the lack of folder hierarchy, you can retrieve your data much faster.

Reduction in cost. Due to the scale-out nature of object storage, it's less costly to store all your data.

Optimization of resources. Because object storage does not have a filing hierarchy, and the metadata is completely customizable, there are far fewer limitations than with file or block storage.

Object Storage and the Cloud

The term ‘cloud native’ is widely used in technical circles but doesn’t have a particularly clear definition. The confusion lies in the fact that being ‘cloud native’ has little to do with the environment your application is deployed to—the term is equally applicable to on-premise or the public cloud. Rather, the term refers to application and architecture characteristics. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach.

These techniques enable loosely coupled systems that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil.

This has particular implications for storage infrastructure.

Modern application architectures are based on the S3 API, developed by Amazon. The API only interacts with object storage, and because of its RESTful APIs, made object storage the dominant class of storage in the cloud. As a result, every major service in the cloud is built on object storage, from Redshift and BigQuery to Snowflake.

Given that object storage was dominant in the cloud, it evolved along the cloud-native roadmap, adhering to the same dynamic, API-driven way as everything else in the cloud native ecosystem - which naturally includes Kubernetes.

Object Storage and Kubernetes

What exactly does it mean for storage to be Kubernetes native? There are six main criteria.

1. Let Kubernetes orchestrate storage nodes

Kubernetes is a powerful orchestrator, and can be leveraged to handle both compute and storage orchestration. A truly cloud native storage option integrates with Kubernetes, allowing operators to manage storage with the Kubernetes interface and Kubernetes to handle everything from provisioning to volume placement.

2. Highly multi-tenant

Multi-tenancy allows multiple customers to use a single instance of an application, and when implemented correctly can reduce operational overhead, decrease costs and reduce complexity, especially at scale. However, it also requires strict resource isolation, so that multiple users can access either compute or storage resources without impacting other users. A truly cloud native storage solution will provide sufficient resource isolation to make multi-tenant architecture secure, highly available and performant.

What this means in the object storage world is that the Kubernetes infrastructure needs to isolate and manage storage tenants. If Kubernetes isn't managing the underlying infrastructure then it is not truly cloud native. This disqualifies those appliance vendors with CSI or Operator integration.

3. Lightweight

Multi-tenancy isn't possible unless the storage system is extremely lightweight and able to be packaged with the application stack. If the storage system takes too much resources or contains too many APIs, it won't be possible to pack many tenants on the same infrastructure.

4. Scalable

Scalability is one of the key attributes of cloud native storage systems. Kubernetes' advantage is that it has proved itself at scale. Kubernetes can also be used to manage storage scaling, but only if the underlying storage system integrates with Kubernetes and hands off provisioning and decommissioning capabilities to Kubernetes.

5. API-driven

One of the core tenets of Kubernetes and cloud native systems in general is to manage as much as possible through automation. For a storage system to be cloud native in the truest sense, it must integrate with Kubernetes through APIs and allow for dynamic, API-driven orchestration.

6. User Space

The final consideration is perhaps the most difficult. For an object storage solution to be cloud native it has to run entirely in the user space with no kernel dependencies. This is not how most object storage systems have been built, particularly the hardware appliances. Nonetheless, if you want to containerize your storage and deploy it on any Kubernetes cluster, you have to abide by these constraints. By definition, this means solutions that require kernel patching or have specialized hardware won't be cloud native.

While simple on some level, these two criteria to claim "cloud-native" status are actually quite difficult in practice. Public cloud object storage vendors do quite well against them - indeed one would expect them to given that Google is the source of Kubernetes and Amazon is the source of S3. Private cloud object storage vendors have a much more difficult time passing these tests. Traditional SAN/NAS architectures are not well suited to the requirements of the cloud-native world and as result, you don't find them deployed at scale in the public cloud. Part of the reason, beyond the technical arguments is that The cloud providers ensure this by pricing file at 13x and block at 4.6x the cost of object while delivering a superb object storage experience (fast, reliable, secure, RESTful).

The Importance of Software Defined Object Storage

While the benefits of object storage should be clear at this point, the totality of those benefits are only available to those enterprises that opt for software defined object storage. The reasons are straightforward:

1. Software defined is the way of the cloud. You cannot run in and across clouds as an appliance.

2. You cannot containerize an appliance. One of the core benefits of object storage is its cloud-nativeness. You relinquish that with an appliance model. If you cannot containerize the storage, Kubernetes is also off the table.

3. Software defined increases the universe of hardware. This matters given that heterogeneity is the natural state of things and your object store needs to run from the edge to the core.

4. Software defined enables greater flexibility when it comes to capacity and performance. Can you buy an appliance(s) for HDD and another for NVMe, yes - but software provides significantly more flexibility and alternative chip sets (Intel, ARM, Nvidia etc).

Core Evaluation Criteria for Software Defined Object Storage

Pure Play Solutions

Many solutions try to offer block, file and object under a single brand. That may seem appealing from a management perspective but it is not possible to be a first class filer, a first class block store and a first class object store. They require different focus, architecture and approaches.

Choose a vendor that just does object storage. It will ensure that it is best in class. If the enterprise has file and block requirements, they should seek out the pure play solutions in those markets as well.

AWS S3 Compatibility

S3 compatibility is a hard requirement for cloud-native applications. Select solutions that support both the V2 and V4 versions of the API. Similar to the pure play criteria above, seek out solutions that focus exclusively on S3.

The S3 API is the de facto standard in the cloud and, as a result, alternatives to AWS must speak the API fluently to function and interoperate across diverse environments - public cloud, private cloud, datacenter, multi-cloud, hybrid cloud and at the edge.

The ultimate question is whether or not the application in question will work against the object storage endpoint using standard S3 calls. This can be determined in advance of the selection.

For more general, multi-purpose use cases, consider the validity of the claims made by the vendor. Can they be tested? Is the software proprietary, thereby limiting the use cases, application and hardware environments experienced via the API?

S3 Select

To deliver high-performance access to big data, analytic and machine learning workflows require server-side filtering features - also referred to as "predicate pushdown".

Developers and architects should seek a SIMD accelerated version of the S3 Select API which effectively enables SQL query capabilities directly on the object store. Users can execute SELECT queries on their objects, and retrieve a relevant subset of the object, instead of having to download the whole object. With the S3 Select API, applications can now download a specific subset of an object - only the subset that satisfies the given SELECT query.

This directly translates into efficiency and performance by reducing bandwidth requirements, optimizing compute and memory resources meaning more jobs can be run in parallel - with the same compute resources. As jobs finish faster, there is better utilization of analysts and domain experts. This capability works for objects in CSV, JSON and Parquet formats and is effective on compressed objects as well.

Multi-Cloud

An object store should run on any cloud the enterprise operates on. Given that most enterprises are multi-cloud at this juncture, that means all of the major public clouds, all of the major private clouds and Kubernetes distributions (Tanzu, OpenShift, Ezmeral and SUSE), data centers running in the cloud operating model - even the edge.

Failure to achieve multi-cloud capabilities from your object store will significantly constrain the object storage footprint, application portability and business continuity goals. By definition, multi-cloud can only be achieved with a software-defined solution.

Erasure Coding

Any object store should protect its data with per-object, inline erasure coding. The erasure coding should be written in assembly code to deliver the highest performance possible. The state of the art in erasure code is Reed-Solomon, which stripes objects into data and parity blocks - although these can be configured to any desired redundancy level.
This means that in a 12 drive setup with 6 parity configuration, an object is striped across as 6 data and 6 parity blocks. Even if you lose as many as 5 ((n/2)–1) drives, be it parity or data, you can still reconstruct the data reliably from the remaining drives. Select implementations that ensure that objects can be read or new objects written even if multiple devices are lost or unavailable.

Erasure code protects data without the limitations of RAID configurations or data replicas. Replication results in 3 or more copies of the object on each of the sites. Erasure-code offers a significantly higher level of protection while only consuming a fraction of the storage space as compared to replication.

Erasure coding should be at the individual object level. This allows the healing at an object level granularity. For RAID-protected storage solutions, healing is done at the RAID block layer, which impacts the performance of every file stored on the volume until the healing is completed.

BitRot Protection

Select object stores that protect against BitRot.

Silent data corruption or bitrot is a serious problem for drives resulting in the corruption of data without the user's knowledge. As the drives get larger and larger and the data needs to persist for many years, this problem is more common than we imagine. The data bits decay when the magnetic orientation flips and loses polarity. Even solid state drives are prone to this decay when the electrons leak due to insulation defects. There are also other reasons such as wear and tear, voltage spikes, firmware bugs and even cosmic rays.

Start of the art BitRot protection solutions will have SIMD accelerated implementations of the HighwayHash algorithm. This will ensure that the object store never returns corrupted data - it will capture and heal corrupted objects on the fly.

Integrity is ensured from end to end by computing hash on WRITE and verifying it on every READ from the application, across the network and to the memory/drive. Implementations should be performance optimized and should be able to achieve hashing speeds in excess of 10 GB/sec.

Identity and Access Management

Select object stores that support both internal (all batteries included) IAM plus external IAM. Examples include OpenID connect and LDAP compatible IDP providers. By creating this criteria, developers and architects centralize access, ensuring that passwords are temporary and tokens are rotated, not stored in config files and databases.

Access policies should be fine grained and highly configurable at the API level granularity, which means that supporting multi-tenant and multi-instance deployments become simple.

Again, this requirement will lead the enterprise to software-defined, cloud-native solutions where there is a rich network of integrations.

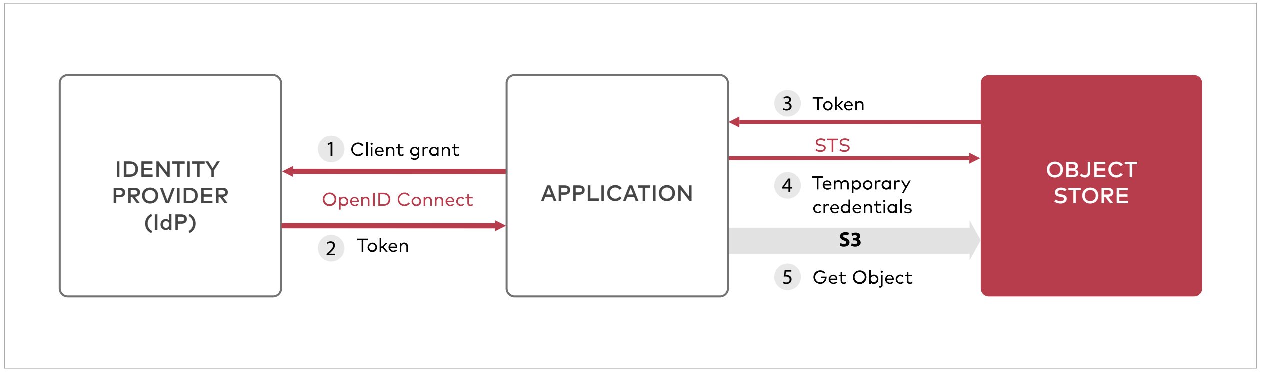

Figure 6: Identity protection and single sign on (SSO) are critical enterprise features.

Security

When selecting an object store, select for solutions that support both the AWS standard as well as other sophisticated schemes including those optimized for the object store itself. Those encryption schemes should support granular, object-level encryption using modern, industry-standard encryption algorithms, such as AES-256-GCM, ChaCha20-Poly1305, and AES-CBC.

Encryption should be able to be applied in-flight and at rest with minimal overhead from a performance perspective. Any selected solution should include support for non-AWS key management services such as Hashicorp Vault, Gemalto KeySecure, and Google Secrets Manager.

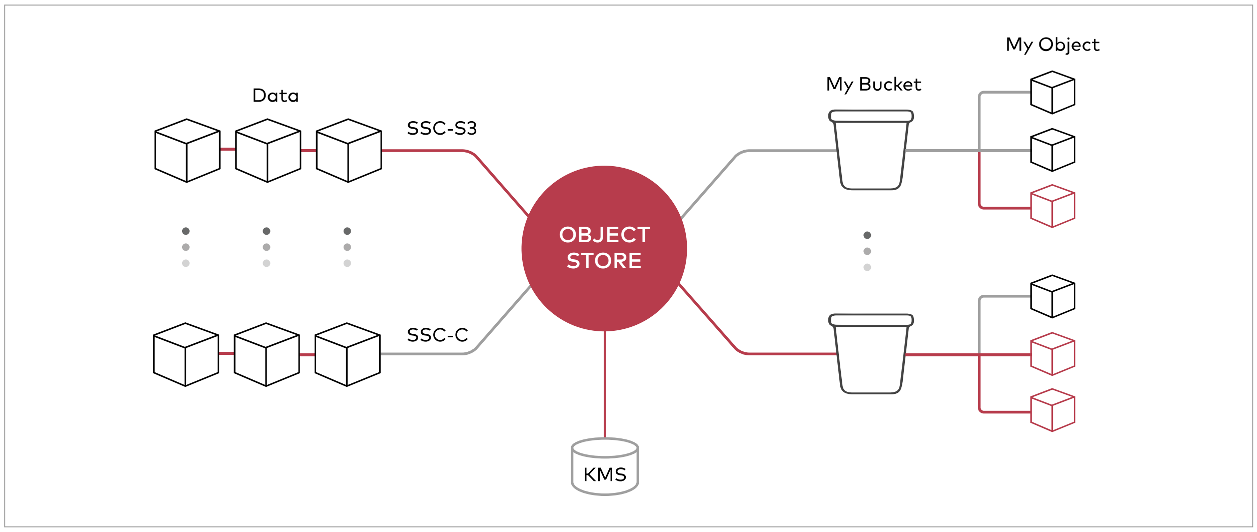

Figure 7: Encryption and WORM protect data in flights and at rest.

Active Active Replication

Object storage buyers should require Active Active Replication. This feature is important for mission-critical production environments and offers unparalleled business continuity in the cloud operating model. Many public cloud providers don't allow for such a configuration across clouds.

Any selected solution should support synchronous and near-synchronous replication depending on the architectural choices and rate of change with the data, available throughput and latency between sites.

Data Protection through Object Locking

Data protection on object storage starts with immutability. Objects cannot, by definition, be changed (mutated). However, this does not mean that they cannot be deleted, overwritten, versioned etc. Any object store selected by the enterprise should support a complete range of functionality including object locking, retention, legal holds, governance, and compliance.

Object locking is used in conjunction with versioning to ensure data immutability and eliminate the risk of data tampering or destruction. An enterprise grade object store should be FIPS 140 compliant and meet securities industry requirements for preserving electronic records in a non-rewritable and non-erasable form requirements as specified by:

Securities and Exchange Commission (SEC) in 17 CFR § 240.17a-4(f), Financial Industry Regulatory Authority (FINRA) Rule 4511(c) and the Commodity Futures Trading Commission (CFTC) in 17 CFR § 1.31(c)-(d).

Metadata Architecture

As noted above, there are two approaches to metadata in the object storage world. Metadata that is written atomically with the object and metadata that is stored in third party database - often Cassandra.

To achieve small object performance and scalability, the object store should write metadata atomically with the object with guarantees for strong-consistency. This approach isolates any failures to be contained within an object and prevents spillover to larger system failures.

Because each object is strongly protected with erasure code and bitrot hash, a failure, even in the middle of a busy workload, would not result in the loss of any data. Another advantage of this design is strict consistency which is important for distributed machine learning and big data workloads.

Lambda Function Support

For an object store to notify serverless applications of individual object actions such as access, creation, and deletion the object store must support Amazon compatible Lambda event notifications. The events should be delivered using industry standard messaging platforms like Kafka, NATS, AMQP, MQTT, Webhooks, or a database such as Elasticsearch, Redis, Postgres, and MySQL.

This will not be required in all cases and architects and developers should consider the need.

Integrations

When selecting a software-defined object store, the architect and/or developer should seek the broadest portfolio of integrations. Choosing a cloud-native object store is one way of achieving the goal.

Those integrations would include external identity providers, external key management, monitoring and alerting, notification targets, federation, orchestration, load balancing and backup targets. This includes mission critical integrations with Active Directory and LDAP (or any OpenID (OIDC)-compatible providers).

CLI and Graphical User Interfaces

Any object store, and in particular, software defined object stores should support both API and Command Line Interface options as well as robust Graphical User Interface options. Both options should have a premium on simplicity while ensuring a complete range of capabilities.

Scalability

Scalability is a multi-dimensional problem in the object storage world. While inherently more scalable than other approaches (block + file), there are factors to consider.

1. How the system scales. A system should scale in discrete building blocks, just like the hyper-scalers was, keeping the failure domain in mind. A system should also scale in such a way as to support heterogeneous hardware, which is a fact of life in both the hardware appliance approach as well as the software defined approach.

2. Performance at scale. Performance can be evaluated across multiple dimensions—raw, straightline performance and performance at scale. The difference is simple—running a benchmark for your object store and a few TBs of data may produce some nice numbers but the real test is to sustain that performance across multiple PBs for all kinds of access patterns and object sizes.

3. Secure. This is why security needs to scale too. That means that security cannot have performance overhead that keeps you from running it all the time. Scalable encryption should also protect data everywhere—in flight (TLS certificates) and at rest (KMS, encryption). Security also includes access management (authentication and authorization) and object locking. They all have to scale if you want to deliver comprehensive protection. Taken together these are monumental requirements that most object stores cannot deliver against.

4. Operational Scale. The ability to manage massive infrastructure with just a handful (or even just a couple to manage across time zones) people is operational scale. OPEX is orders of magnitude higher than CAPEX over time. The ability to scale is a function of the software selected. Simple, powerful software will end up being superior because operational scalability is a software problem, not a people problem.

Metrics and Logging

Metrics and logging are critical when tracking the health and performance of any system. An object store should provide complete visibility into clusters with detailed storage performance monitoring, metrics, and per-operation logging.

The object should ideally support a Prometheus-compatible metrics endpoint. Prometheus is a cloud-native monitoring platform consisting of a multi-dimensional data model with time series data identified by metric name and key/value pairs and outputs to Grafana.

On the logging front, the object store should generate a log for every operation on the cluster. Every operation should generate an audit log with a unique ID and detailed information on the client, object, bucket and all other operation-related metadata. The log data should be written to a configured HTTP/HTTPS webhook endpoint. Custom adapters should be supported to meet specific requirements for audit logging targets.

Data Lifecycle Management & Tiering

An object store should enable the tiering of data across media, across clouds and even within storage classes of a cloud. These capabilities allow the enterprise to optimize for performance and economics.

For example, data in the “hot” tier can be stored on NVMe or SSD to maximize performance, then moved to HDDs for larger scale workloads or long-term storage as it gets colder. Alternatively, the user could tier high performance workloads in private cloud storage while moving colder data to less expensive public cloud infrastructure. Finally, the object store should be able to determine whether data should be moved to block or object storage within a public cloud.

The underlying requirement here is for the object store to be able to run seamlessly in multiple clouds (public, private, edge). To achieve this capability the object store must be software defined.

About MinIO

MinIO is pioneering high performance, Kubernetes-native object storage for the multi-cloud. The software-defined, Amazon S3-compatible object storage system is used by more than half of the Fortune 500. With 781M+ Docker pulls, MinIO is the fastest-growing cloud object storage company and is consistently ranked by industry analysts as a leader in object storage. Founded in 2014, the company is backed by Intel Capital, Softbank Vision Fund 2, Dell Technologies Capital, Nexus Venture Partners, General Catalyst and key angel investors.

Additional Information

Email: hello@min.io
MinIO Inc.
530B University Avenue,
Palo Alto, CA 94301

Resources

https://min.io
https://docs.min.io/
https://blog.min.io/

© 2022 MinIO, Inc.