Implementing S3-fronted cold storage at CERN

Mario Vitale, Computing Engineer @ CERN | FOSDEM 2026

About me

And why am I here

- Computing Engineer @ CERN
- Tape Archival & Backup team
- FOSS enthusiast
- Proud homelab dad
- First task: review S3+Tape PoC
- Goal: design an S3 interface for our Tape infrastructure

What we will discuss

- Project goal
- Technical context
    - CTA
    - S3+Glacier API
- Proof-of-concept analysis
- Solution brainstorming
- Questions

CERN is the world's biggest laboratory for particle physics.

Our goal is to understand the most fundamental particles and laws of the universe.

Located near Geneva on either side of the Swiss French border

Project goal

CERN hits one exabyte of stored experimental data from the LHC

One million terabytes of experimental data from the LHC have now been sent to the CERN storage system

17 DECEMBER, 2025 | By Rory Harris

A computing server corridor in CERN's main data centre. (Image: CERN)

One exabyte of experimental data has now been gathered from the Large Hadron Collider (LHC), marking a major milestone for CERN's storage system.

- What does CERN store on tape?
  - Physics experiments data
  - User data archival (compliance, DR, etc)
- How does it store them?
  - CERN Tape Archive (CTA) provides access to our tape libraries
- Target solution: S3+Glacier API backed by CTA
  - Widespread client support
  - Avoid reinventing the wheel

Project goal

CERN hits one exabyte of stored experimental data from the LHC

One million terabytes of experimental data from the LHC have now been sent to the CERN storage system

17 DECEMBER, 2025 | By Rory Harris

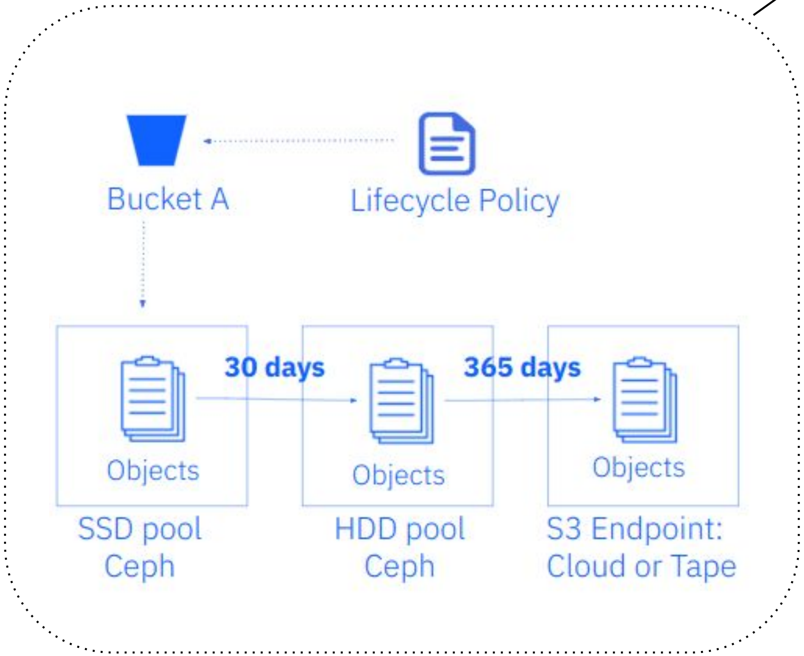
A computing server corridor in CERN's main data centre. (Image: CERN)

One exabyte of experimental data has now been gathered from the Large Hadron Collider (LHC), marking a major milestone for CERN's storage system.

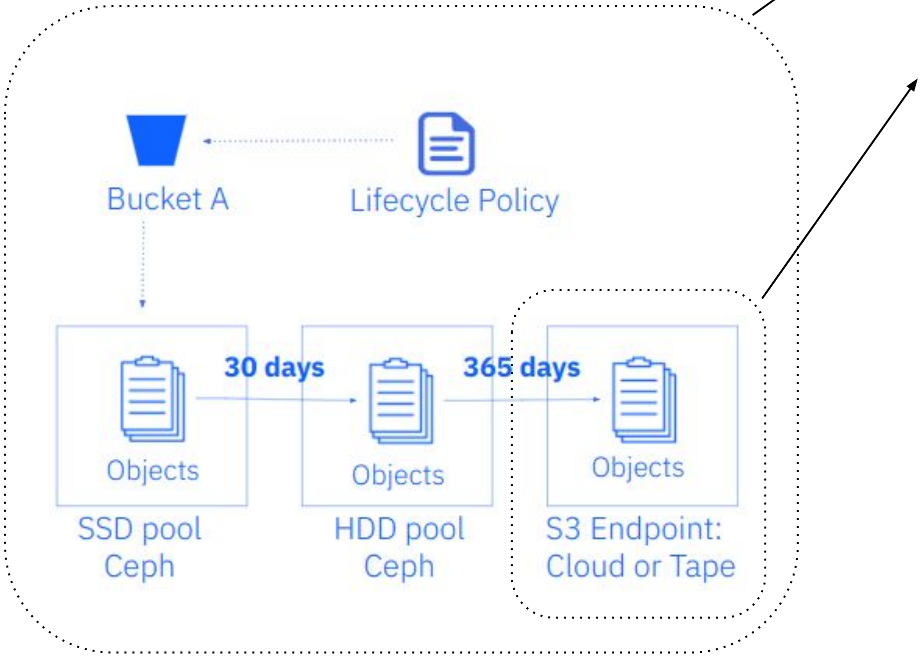
- What does CERN store on tape?
  - Physics experiments data
  - User data archival (compliance, DR, etc)
- How does it store them?
  - CERN Tape Archive (CTA) provides access to our tape libraries
- Target solution: S3+Glacier API backed by CTA
  - Widespread client support
  - Avoid reinventing the wheel

Our target

“FOSS tape-backed backup service”



Our target

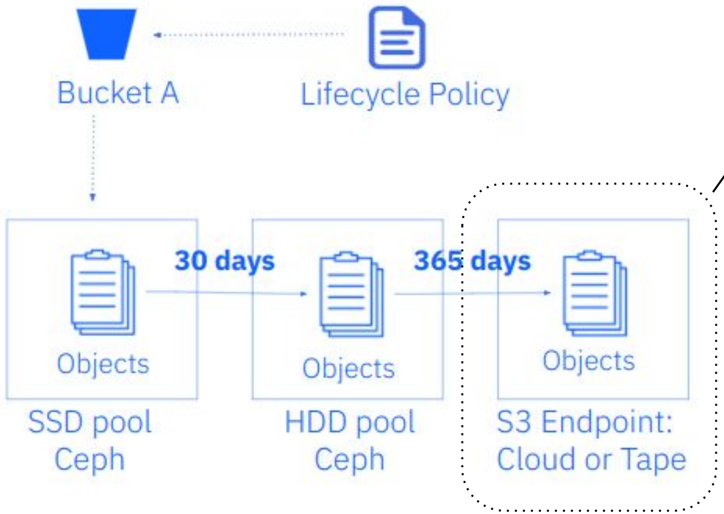


"FOSS tape-backed backup service"

"The Appliance"

- Independent S3 endpoint
- Physical tape storage
- CTA underneath
- FOSS

Our *challenge*: building “The Appliance”



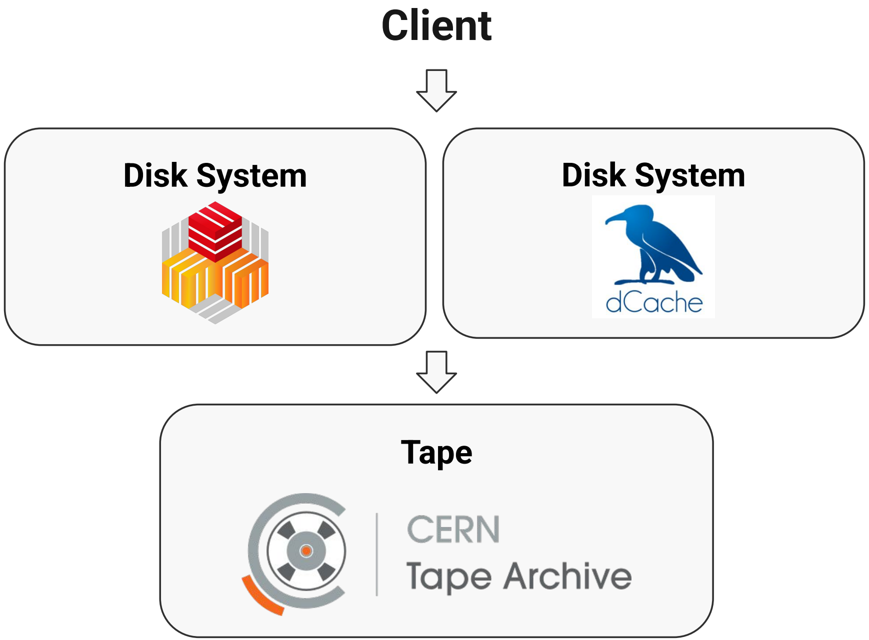
“The Appliance”

- Independent S3 endpoint
- Physical tape storage
- CTA underneath
- FOSS

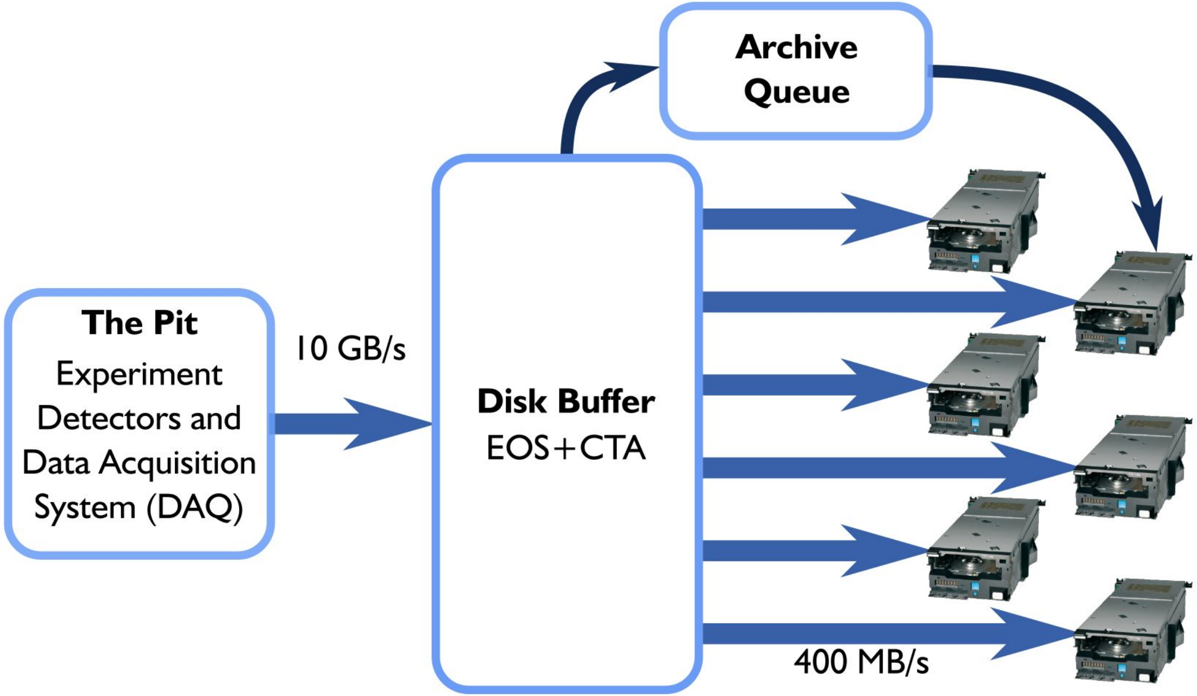
CERN Tape Archive
(aka CTA)

About CTA

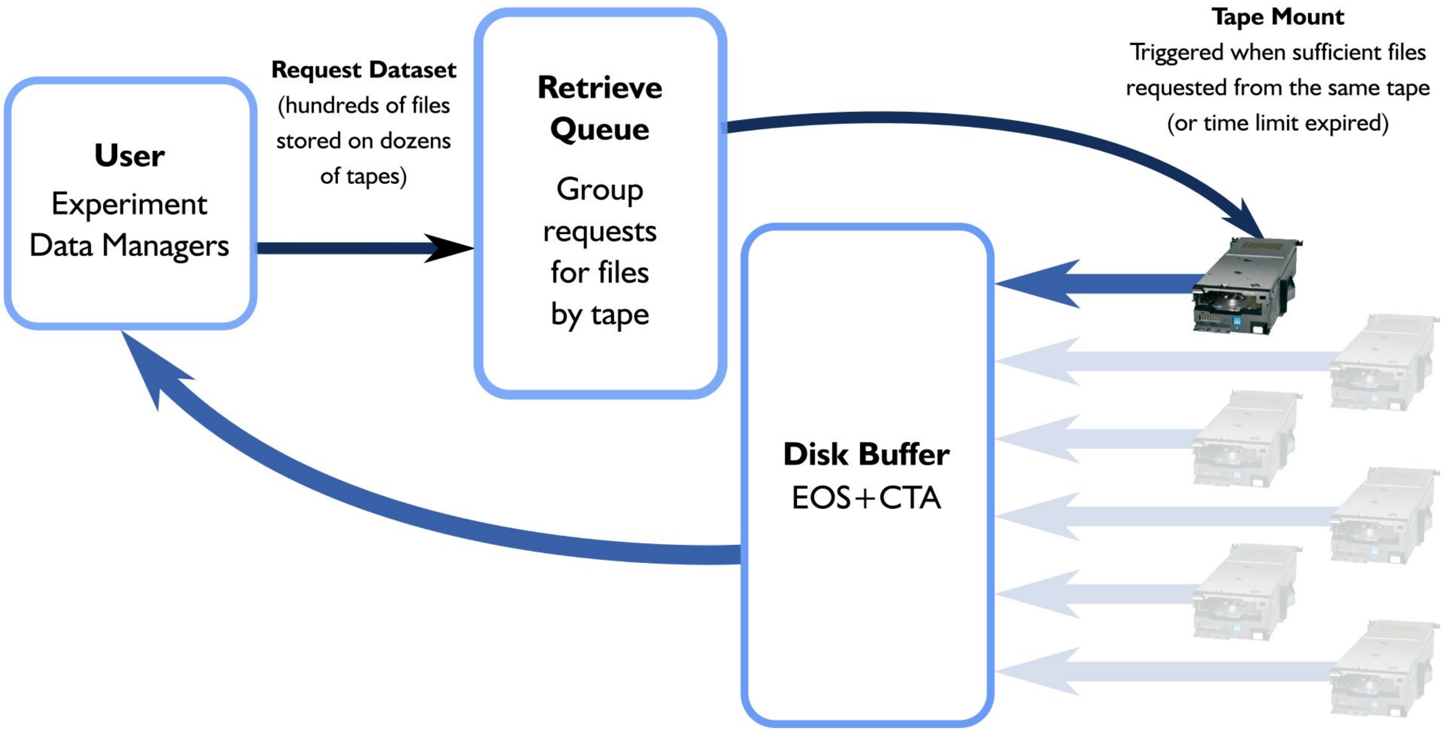
- Provides an interface to physical tape infrastructure
- Clients can only use CTA through a disk buffer
  - "CTA is a tape backend for the disk buffer"
  - Two disk buffers supported: EOS and dCache
- Supported flows:
  - Archival & Recall
  - Deletion (data not immediately overwritten)
  - Repack
    - Think "defrag a tape into a new tape"
    - Not disk-buffer initiated
- FOSS (GPLv3 licensed)



Archival flow

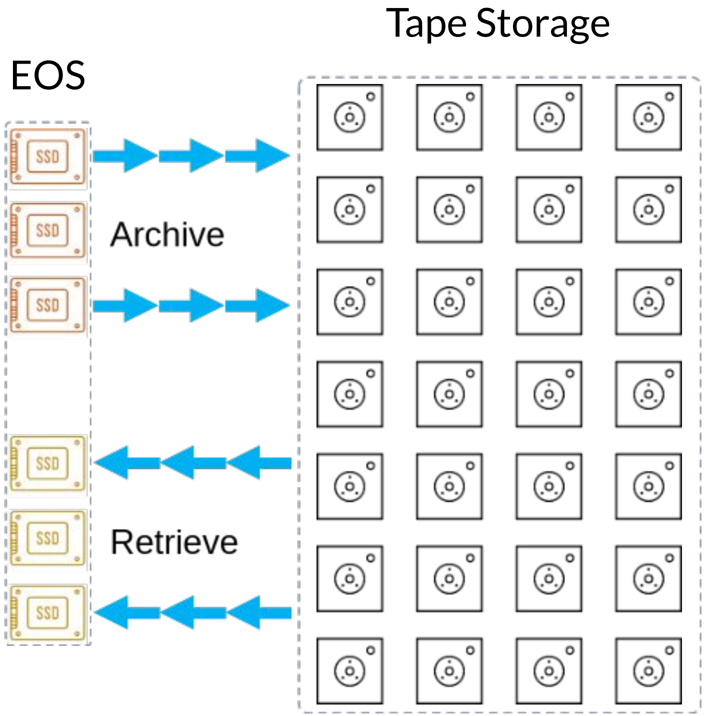


Recall flow


User
Experiment
Data Managers
Request Dataset
(hundreds of files
stored on dozens
of tapes)
Retrieve
Queue
Group
requests
for files
by tape
Tape Mount
Triggered when sufficient files
requested from the same tape
(or time limit expired)
Disk Buffer
EOS+CTA

EOS
aka “The Disk Buffer”

- File metadata is stored only on EOS
  - EOS persistency is critical!
- File content may be:
  - “Online”: file content is on-disk
  - “Offline”: file content is not-on-disk
    - But retrievable from tape
    - File content takes no space on-disk
    - Non-trivial semantic!
- Designed for large & stable throughput
  - Array of independent SSDs
  - R/W to SSDs in round-robin fashion
  - No data redundancy
- Network connection to CTA: 25 Gb/s



CTA's Nota Bene

Current features/constraints

- Tape has <u>minimum</u> speed requirements
    - To be provided by the disk buffer
    - Requires stable R/W of at least ~180MB/s (hardware dependent)
    - Current system performs at 400MB/s
    - Too slow → shoe-shining → Inefficient, bad for hardware
- File metadata lives on the disk buffer
    - Mapped back to the object through the Archive ID
    - 1 file on disk buffer = 1 record on tape
    - Disk buffer is critical!

CTA's Nota Bene

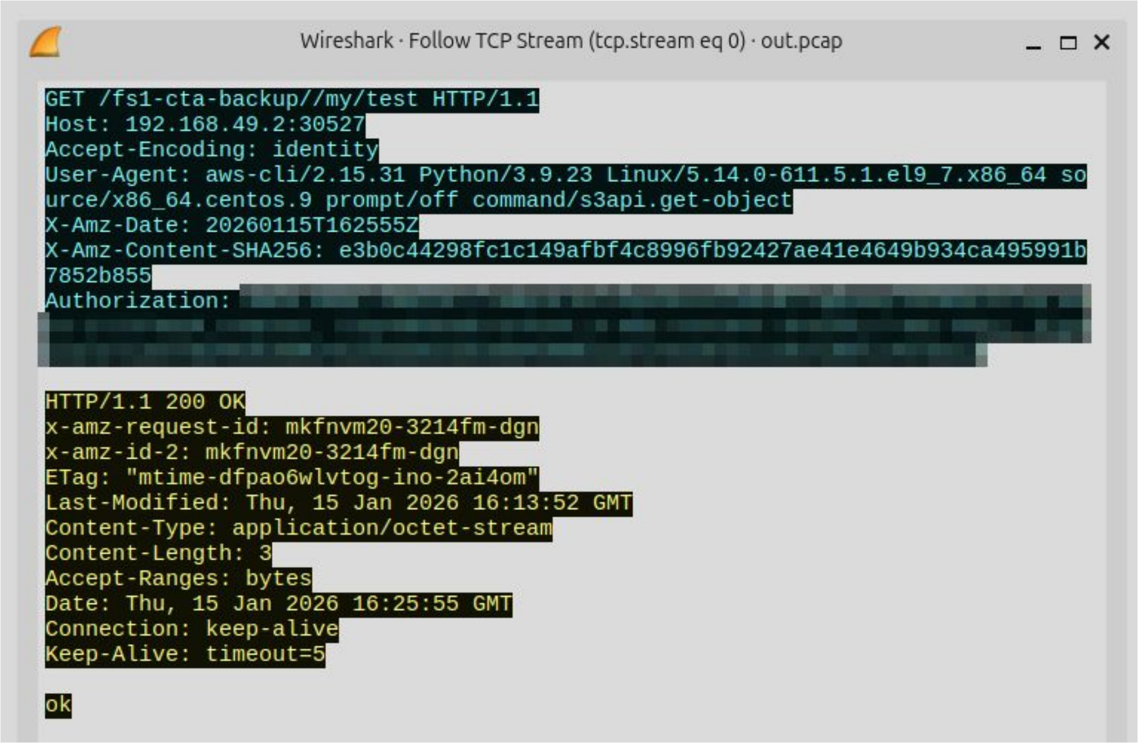
Current features/constraints

- No object affinity logic in CTA (as of today)
    - A tape daemon can't know which objects should be colocated
    - Result: any object could end up on any tape
    - Solution WIP
- File considered safe when it's fully on tape
    - The user polls the disk buffer for this confirmation
    - If anything fails before then: the user must send the file again
- Cannot modify files. Only delete
    - Tape storage is linear → To modify is to fragment

S3+Glacier API

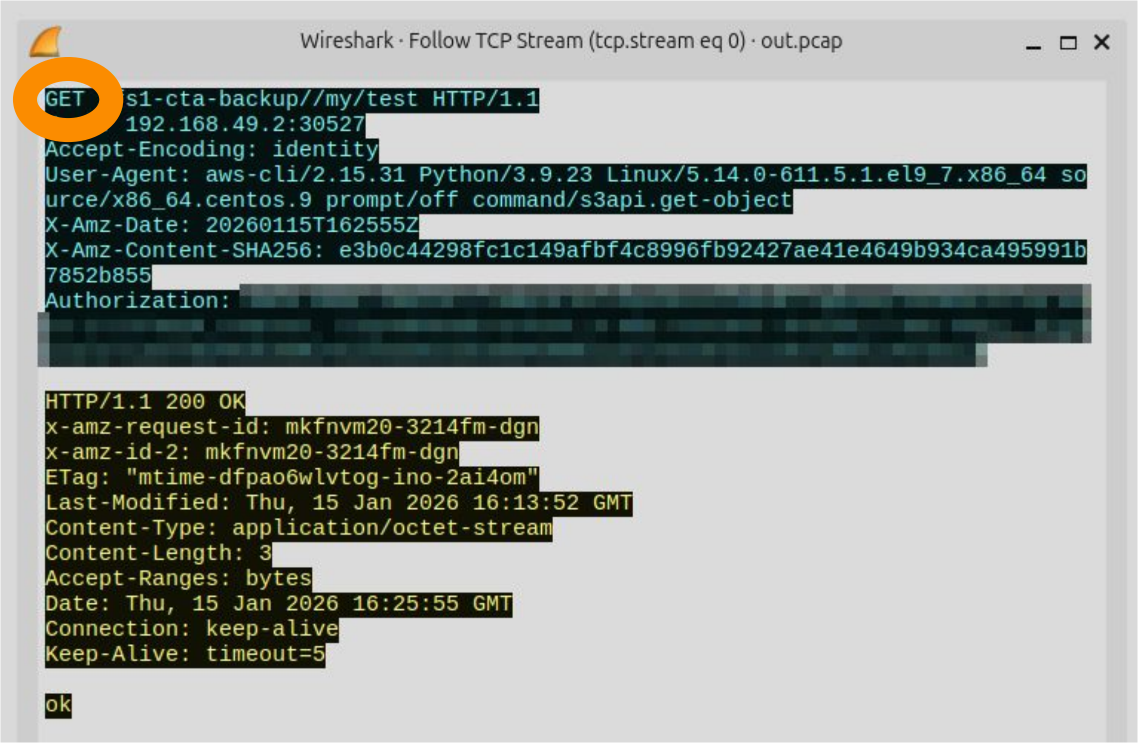
S3 API: AWS' Object Storage interface

```
AWS_DEFAULT_REGION=us-east-1
AWS_ENDPOINT_URL=http://192.168.49.2:6001
AWS_SECRET_KEY_ID=*******
AWS_SECRET_ACCESS_KEY=*******
aws s3api get-object \
  --bucket=fs1-cta-backup \
  --key=/my/test \
  /dev/stdout
```



S3 API: AWS' Object Storage interface

```
AWS_DEFAULT_REGION=us-east-1
AWS_ENDPOINT_URL=http://192.168.49.2:6001
AWS_SECRET_KEY_ID=*******
AWS_SECRET_ACCESS_KEY=*******
aws s3api get-object \
  --bucket=rs1-cta-backup \
  --key=/my/test \
  /dev/stdout
```

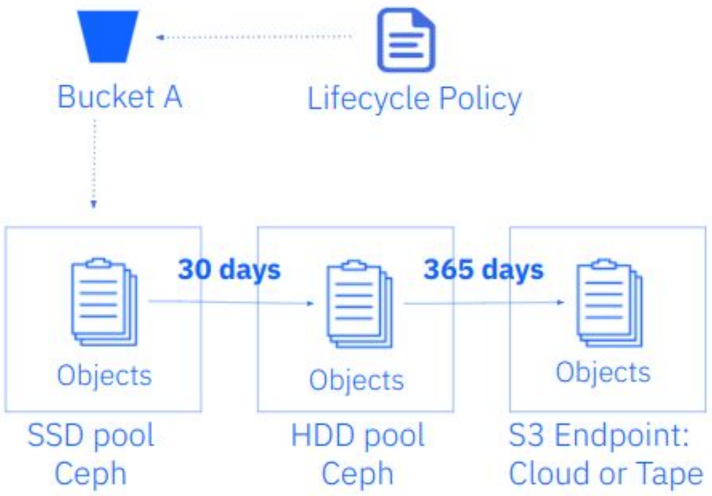


S3 API: common actions

- Object
  - {Get,Head,Put,Delete}Object
  - MultiPart upload, Ranged download ← Parallel! Cool!
    - So cool that RGW Cloud Tier uses MultiPart automatically
- Bucket
  - {List,Create,Delete}Bucket
  - Lifecycle configuration
- Metadata
  - {Put,Get,Delete}ObjectTagging
  - PutObject's `x-amz-meta-*` HTTP headers
- Policy, Locking, Notifications…and more

S3 Glacier API

- Archival → Lifecycle policy (LC)
  - Not user initiated, no imperative API
    - Emulation 1: tag-based rules
    - Emulation 2: direct write-to-GLACIER
  - Under which conditions is it transitioned?
  - Where is it transitioned? (Storage Class)
    - LC rules only make objects *colder*
    - "GLACIER" = de-facto standard
  - Returns `InvalidObjectState` until restored
- Recall → RestoreObject action
  - Choice of restore type:
    - Temporary: warm copy expires
    - Permanent: warm copy subject to LC rules
  - Client poll to track restore status (*`x-amz-restore`*)

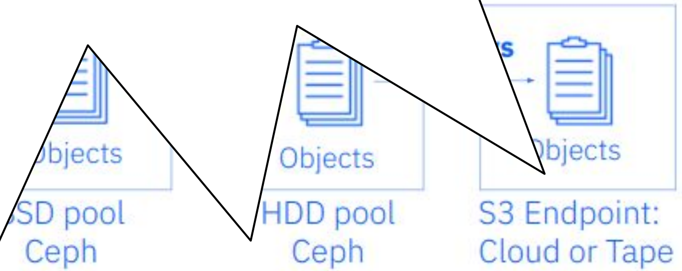


S3 Glacier API

- Archival → Lifecycle policy (LC)
  - Not user initiated, no i
    - Emulation 1: tag-base
    - Emulation 2: direct write-
  - Under which conditions is it tr
  - Where is it transitioned? (
    - LC rules only make objects *colder*
    - “GLACIER” = de-facto standard
  - Returns `InvalidObjectState` until resto
- Recall → RestoreObject action
  - Choice of restore type:
    - Temporary: warm copy expires
    - Permanent: warm copy subject to LC rules
  - Client poll to track restore status (`x-amz-restore`)

As a user-facing API,
S3 Glacier doesn’t define any interface to the tape infrastructure

Cold-storage data transfer mechanisms are implementation specific



S3 API support

Ceph

Feature	Status	Remarks
List Buckets	Supported	
Delete Bucket	Supported	
Create Bucket	Supported	Different set of canned ACLs
Bucket Lifecycle	Supported	
Bucket Replication	Partial	Permitted only across zones
Policy (Buckets, Objects)	Supported	ACLs & bucket policies are supported
Bucket Website	Supported	
Bucket ACLs (Get, Put)	Supported	Different set of canned ACLs
Bucket Location	Supported	
Bucket Notification	Supported	See S3 Bucket Notifications Compatibility
Bucket Object Versions	Supported	
Get Bucket Info (HEAD)	Supported	
Bucket Request Payment	Supported	
Put Object	Supported	
Delete Object	Supported	
Get Object	Supported	
Object ACLs (Get, Put)	Supported	
Get Object Info (HEAD)	Supported	
POST Object	Supported	
Copy Object	Supported	
Multipart Uploads	Supported	
Object Tagging	Supported	See Object Related Operations for Policy verbs
Bucket Tagging	Supported	
Storage Class	Supported	See Storage Classes
Bucket Logging	Supported	
Object Ownership	Supported	

NooBaa

Category	Feature	API Action	Backingstore	NSFS	NS AWS	NS Azure	Commen
Basic	Bucket	HeadBucket	✓	✓	✓	✓	
	Bucket	CreateBucket	✓	✓	✓	✓	
	Bucket	DeleteBucket	✓	✓	✓	✓	
	Bucket	ListBuckets	✓	✓	✓	✓	
	Bucket	GetBucketLocation	✓	✓	✓*	✓*	*Always returns empty string
	Object	HeadObject	✓	✓	✓	✓	
	Object	GetObject	✓	✓	✓	✓	
	Object	PutObject	✓	✓	✓	✓	
	Object	DeleteObject	✓	✓	✓	✓	
	Object	DeleteObjects	✓	✓	✓	✗	
	Object	ListObjects	✓	✓	✓	✓	
	Object	ListObjectsV2	✓	✓	✓	✓	
	Object	CopyObject	✓	✓	✓	✓	
	Object	GetObjectAttributes	✓*	✓*	✓	✓*	*Partially implen
	Multipart Upload	CreateMultipartUpload	✓	✓	✓	✓	
	Multipart Upload	CompleteMultipartUpload	✓	✓	✓	✓	
	Multipart Upload	AbortMultipartUpload	✓	✓	✓	✓*	*Azure does not aborting upload: operation is ignc Azure will clean u parts after 7 day
	Multipart	ListMultipartUploads	✓	✓	✓	✗	

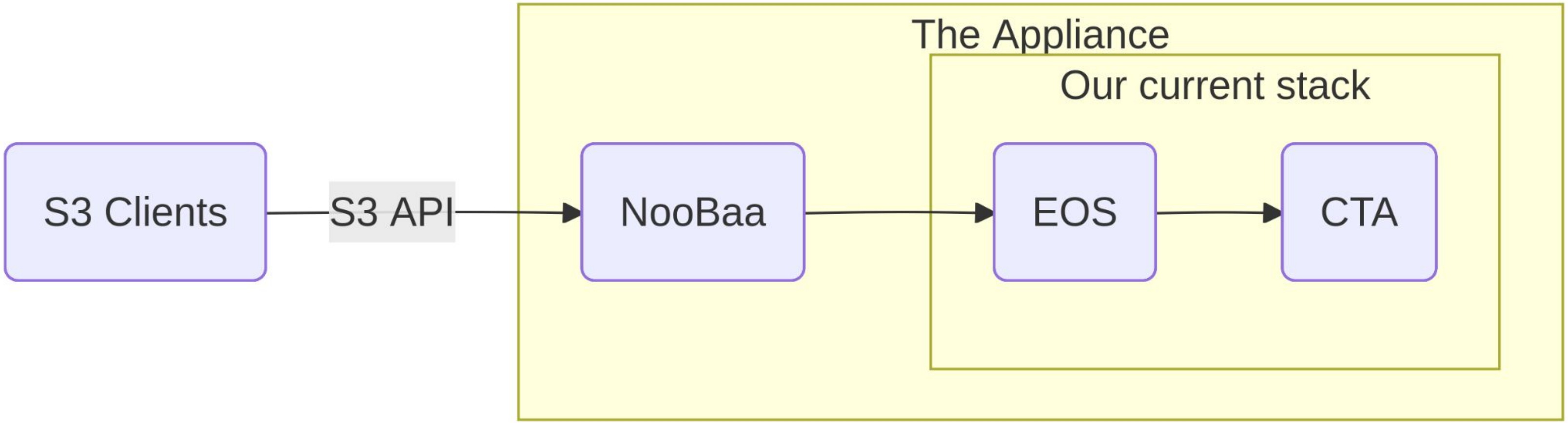
Garage

Core endoints

ENDPOINT	GARAGE
CreateBucket	✅ Implemented
DeleteBucket	✅ Implemented
GetBucketLocation	✅ Implemented
HeadBucket	✅ Implemented
ListBuckets	✅ Implemented
HeadObject	✅ Implemented
CopyObject	✅ Implemented
DeleteObject	✅ Implemented
DeleteObjects	✅ Implemented
GetObject	✅ Implemented

Proof-of-Concept Analysis

PoC architecture



Credits: Pablo Oliver Cortes (CERN), Sarthak Negi

Why noobaa?

- FOSS software
- Commercial production usage

NooBaa Deployment

- Kubernetes operator (comes with CRDs)
  - `noobaa install` → `kubectl -n noobaa get service s3`
- TAPECLOUD interface to external systems
  -
```
CONFIG_JS_NSFS_GLACIER_ENABLED="true"
CONFIG_JS_NSFS_GLACIER_LOGS_ENABLED="true"
CONFIG_JS_NSFS_GLACIER_BACKEND="TAPECLOUD"
CONFIG_JS_NSFS_GLACIER_TAPECLOUD_BIN_DIR="/opt/cta/glacier/"
CONFIG_JS_NSFS_GLACIER_LOGS_DIR="/var/log/noobaa/nsfs/"
```
- Online data on NSFS
  - aka “in a directory”, could be CephFS
  - Metadata as xattrs

NooBaa Deployment

- Kubernetes operator (comes with CRDs)
    - `noobaa install` → `kubectl -n noobaa get service s3`
- TAPECLOUD interface to external systems
    - ```
      CONFIG_JS_NSFS_GLACIER_ENABLED="true"
      CONFIG_JS_NSFS_GLACIER_LOGS_ENABLED="true"
      CONFIG_JS_NSFS_GLACIER_BACKEND="TAPECLOUD"
      CONFIG_JS_NSFS_GLACIER_TAPECLOUD_BIN_DIR="/opt/cta/glacier/"
      CONFIG_JS_NSFS_GLACIER_LOGS_DIR="/var/log/noobaa/nsfs/"
      ```
- Online data on NSFS
    - aka “in a directory”, could be CephFS
    - Metadata as xattrs

```yaml
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: nsfs-local
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: WaitForFirstConsumer
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nsfs-vol
spec:
  storageClassName: nsfs-local
  volumeMode: Filesystem
  persistentVolumeReclaimPolicy: Retain
  local:
    path: /nsfs/
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteMany
  nodeAffinity:
    required:
      nodeSelectorTerms:
        - matchExpressions:
          - key: kubernetes.io/os
            operator: Exists
```
```

TAPECLOUD: how does it work?

- File movement is asynchronous
  - File write to GLACIER StorageClass → append to migrate.log
  - RestoreObject API call → append to recall.log
- A cronjob (or operator) will call the log handling scripts…
  - `node manage_nsfs.js glacier {migrate,restore}`
- …which are mainly wrappers around your executable implementation
  - Exec interface: `/opt/cta/glacier/{migrate,recall,low_free_space}`
  - Your executable will read `{migrate,recall}.log` and migrate/recall each file*
  - Plus safety handling (locking, migration log rotation, etc)

PoC observations

About NooBaa:

- Exec interface is extremely flexible
  - NB: conditional execution (is there enough space on disk buffer?)
- Documentation could be improved
  - Migration logs format did not match code documentation
  - Boundary of failure handling responsibility was unclear
- Certain useful features are missing
  - StorageClass Lifecycle transitions; object deletion semantics
  - GLACIER files immediately inaccessible - even if still on disk
- Observed failure at around 10k migrations
  - To investigate. PoC obviously not production ready, bash implementation

PoC observations

About the architecture:

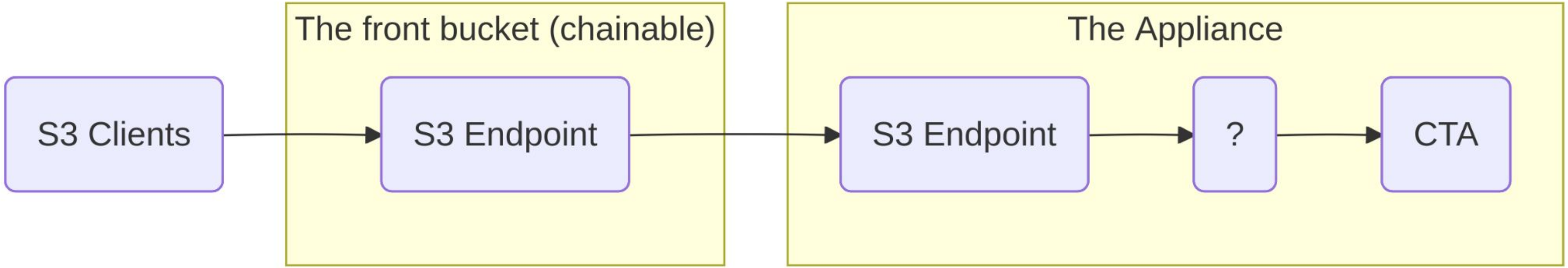
- Duplicate disk buffer
    - Duplicated metadata
    - Duplicated provisioned space for disk buffer
- "One-more-layer" approach → Troubleshooting was painful
- File content migration initiated by NooBaa
    - Needs either intermediary disk buffer so CTA can pull, or long-running implementation
    - e.g. tape likely not ready to receive when `node manage_nsfs.js glacier migrate` is invoked
    - The backing buffer needs some control to not be overwhelmed (backoff? Another buffer?)
- Expertise bias
    - Widespread production Ceph use in CERN

Solution Brainstorming

Premise (1/2): notable Ceph RGW features

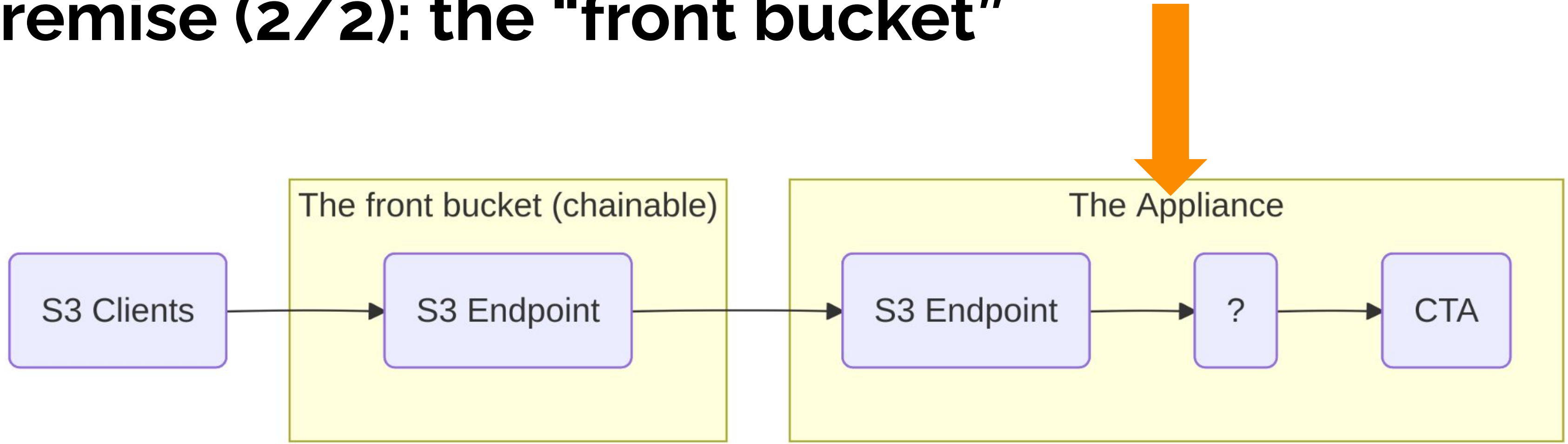
- SAL: Software Abstraction Layer (aka “zipper project”)
  - Split S3 API protocol handling from Storage layer
  - Lets you implement:
    - Filters (e.g. modify response status code, conditionals, etc)
    - Drivers (e.g. backed by POSIX filesystem, instead of RADOS)
- *Cloud Transition* & *Cloud Restore* features
  - Tl;dr: Cold Storage = another S3 endpoint (aka “Cloud Tier”)
  - `retain_head_object=true` to retain metadata in RGW
  - `allow_read_through=true` interesting, but HEAD is unsupported
  - Versioning is well supported
  - Implemented through SAL
- Lua scripting
  - Essentially a SAL filter

Premise (2/2): the “front bucket”



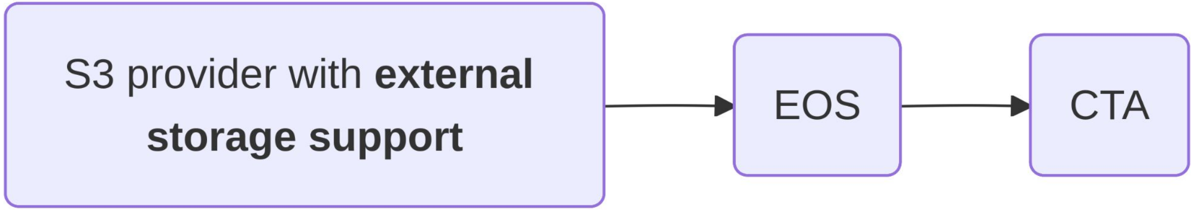
- Benefits of the split: isolation and control (bandwidth, file layout, policy, metadata…)
- Ceph’s Cloud Tier makes this trivial…
- …minus upstream-bound metadata propagation
    - Not an issue *if* the bucket chain guarantees uniform data durability
    - `allow_read_through=true` would help with HEAD support
        - Alternative: Lua filter on HEAD requests

Premise (2/2): the "front bucket"



- Benefits of the split: isolation and control (bandwidth, file layout, policy, metadata...)
- Ceph's Cloud Tier makes this trivial...
- ...minus upstream-bound metadata propagation
  - Not an issue *if* the bucket chain guarantees uniform data durability
  - `allow_read_through=true` would help with HEAD support
    - Alternative: Lua filter on HEAD requests

Solution family 1: “One more layer”



Examples:
- NooBaa with TAPECLOUD executable
- Ceph with EOS SAL driver

Pros:
- Only additions to an already-working system
    - E.g. performance requirements satisfied
- Low development effort (relatively)

Cons:
- Must still maintain our own storage driver
- Duplication (metadata, provisioned capacity)
- The more there is, the more can fail
- Configuration and debugging is painful

Solution family 1.5: “One more (thin) layer”

Examples:

- VersityGW with EOS storage module (see EOSS3 project)
- Ceph with EOS SAL driver (possibly)
- XRootD’s XrdS3 plug-in

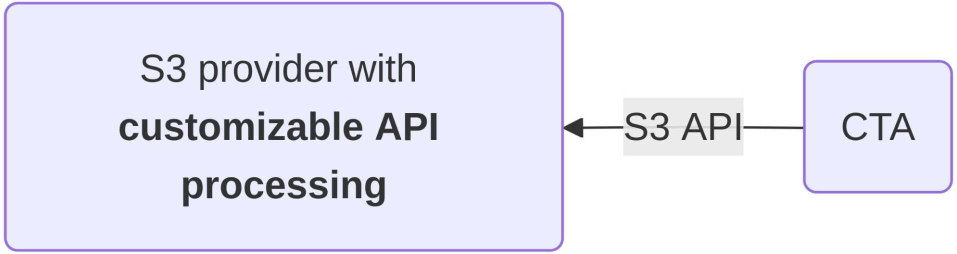
Pros:

- Only additions to an already-working system
  - E.g. performance requirements satisfied
- Low development effort (relatively)

Cons:

- Thorough emulation of S3 API is challenging
- ~~Duplication (metadata, provisioned capacity)~~
- The more there is, the more can fail (mitigated)
- Configuration and debugging is painful

Solution family 2: “Client-driven emulated Glacier”



Examples:

- Ceph with CTA-driven Lua filter (e.g. CTA writes object tag to mark file as offline)
- VersityGW customization. The company already provides a paid Tape offering

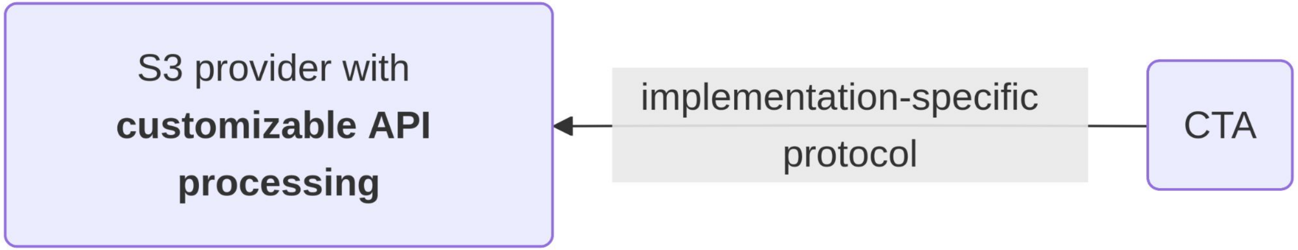
Pros:

- Simple
  - One way to access files: S3 client
  - Less moving parts, less failure points
- CTA controls file movement
- MultiPart uploads / Ranged downloads
  - Leverage scale out for performance

Cons:

- Emulation accuracy (e.g. temporary restores)
- Needs careful performance planning
  - But is doable
- How to know if RestoreObject was called?
  - Dependent on Notification API support

Solution family 3: “Out-of-band data control”



Examples:

- Ceph, using librados to truncate/rehydrate RGW’s underlying RADOS objects
- Ceph with non-S3-standard API or tooling (e.g. radosgw-admin)
  - Like Cloud Transition truncates/rehydrates file, but client receives/provides data instead

Pros:

- Implementation dependent…
- CTA controls file movement
- Low development effort (relatively)

Cons:

- Implementation dependent…
- May expose internal details
  - Could break on new provider releases
- Provider must be aware of offline files
- How to know if RestoreObject was called?
  - Dependent on Notification API support

Solution family 4: “CTA implements S3 API”

CTA with **S3-fronted, distributed disk buffer implementation**

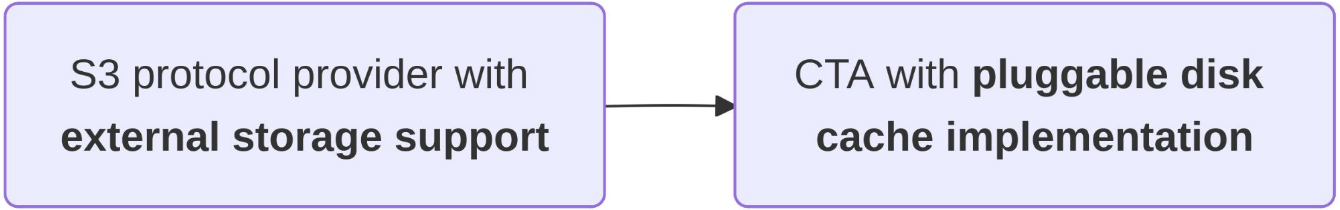
Pros:

- CTA is a standalone, full backup and archival software appliance
    - ~~No~~ Less dependent on external projects
- Object lifecycle is fully managed by us

Cons:

- **New functional scope for CTA, new problems**
    - For which other software solutions exist
- Huge development & maintenance effort
    - CTA is now the disk buffer
    - Keep up with S3 API changes
- Reinventing the wheel (good enough reason?)
    - S3 API: Ceph’s SAL exists
    - Distributed disk buffer: EOS exists

Solution family 5: “CTA implements disk buffer”



Pros:

- CTA now has a pluggable storage interface
- Object lifecycle is fully managed by us
- Development effort to implement new drivers moved out of CTA

Cons:

- **New functional scope for CTA, new problems**
    - For which other software solutions exist
- Large development & maintenance effort
    - CTA is now the disk buffer
- Reinventing the wheel (good enough reason?)

What's your approach?

Acknowledgements

- Michael Davis
- Vladimir Bahyl
- Niels Buegel
- https://ceph.io
- https://mermaid.js.org/

Questions?

S3-based Disk Buffer: implementation draft using Ceph

- CTA saves its metadata in object *tags*
  - Example: Archive ID, operation-in-progress, etc
  - Don't use metadata: it's semantically part of the object
  - Assumes Ceph Cloud Transition doesn't handle tags
- Lua filter: appliance's response based on tag content
  - Example: if CTA has marked an object as "offline", Lua will reply with `InvalidObjectState`
- CTA will read/write objects as S3 client
  - When archiving: read, mark with tag, rewrite object with size zero
  - When restoring: recall, write object with its content, mark with tag

Focus on major challenges

- Performance level & guarantees
  - 400MB/s per tape drive
- Tape colocation of objects in the same bucket
  - Scheduling *and* repacking
- API Notification support
  - How can CTA know that it needs to perform a rehydration or eviction?
- How to handle versioning?
  - Clean solution in Ceph
- How to handle file modifications?
  - CTA doesn't support it
- Propagate info upstream
  - Has an object been persisted to tape yet?

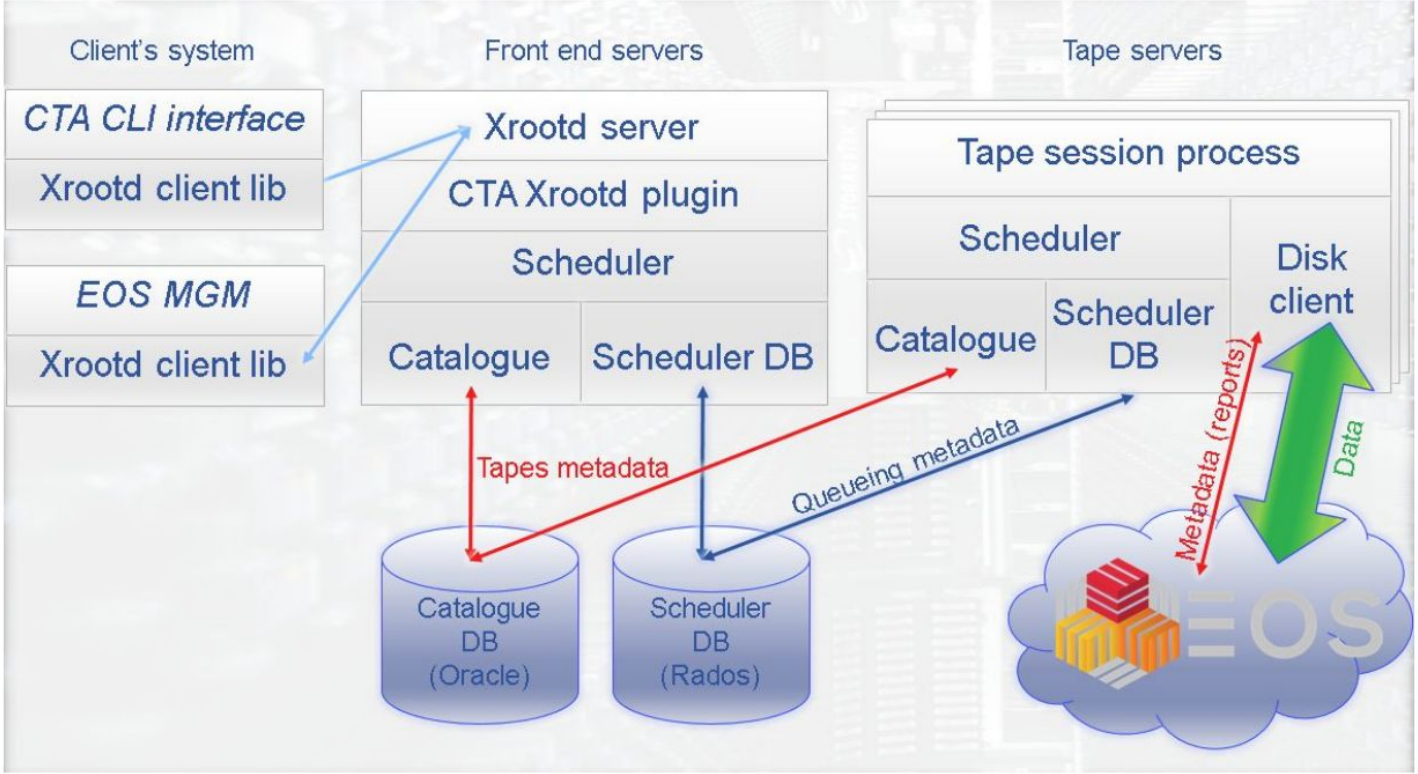
CTA's file flows (CLI)

- Archival: `xrdcp ./file.dat root://ctaeos/eos/ctaeos/cta/file.dat`
  - On tape yet? `eos root://ctaeos ls eos/ctaeos/cta/file.dat -y`
  - Archive ID saved back by CTA as xattr: `sys.archive.file_id`
  - File content only on tape → Becomes “offline”
  - Metadata still on disk (EOS' QuarkDB, critical!)
- Retrieval: `xrdfs root://ctaeos prepare -s /eos/ctaeos/cta/file.dat`
  - On disk yet? See above command
  - Clients initiate disk space reclaim (but GC is also performed)
  - File back on disk → Becomes “online” again
- Deletion: `eos root://ctaeos rm eos/ctaeos/cta/file.dat`
  - Not really deleted yet → “Shadow data” until relabeling

NooBaa TAPECLOUD: usage

1. End-user writes to cold storage: `aws s3 cp ~/file.dat s3://mybucket/file.dat --storage-class GLACIER`
   - End-user cannot access the file immediately after this operation
2. Cronjob migrates file: `node manage_nsfs.js glacier migrate`
3. End-user requests restore: `aws s3api restore-object --bucket mybucket --key testfile.img --restore-request Days=2`
   - End-user polls for restore completion: `aws s3api head-object --bucket mybucket --key file.dat`
4. Cronjob recalls files: `node manage_nsfs.js glacier recall`
5. End-user can access the file again: `aws s3 cp s3://mybucket/file.dat ~/file.dat`

CTA: detailed architecture



CERN **Data Flow**