Evolution of XRootD Monitoring

Detailed Monitoring Collector for Logs and Traces

Guilherme Amadio

XRootD and FTS Workshop @ CNRS Rhône Auvergne Delegation Lyon, France
16 September 2026

Outline

01 XRootD Monitoring Overview

02 Optimizing Network Configuration

03 WLCG Monitoring Requirements

04 Collector Design and Architecture

05 Collector Features and Configuration

06 Collector Health Metrics

07 OpenTelemetry Semantic Schemas

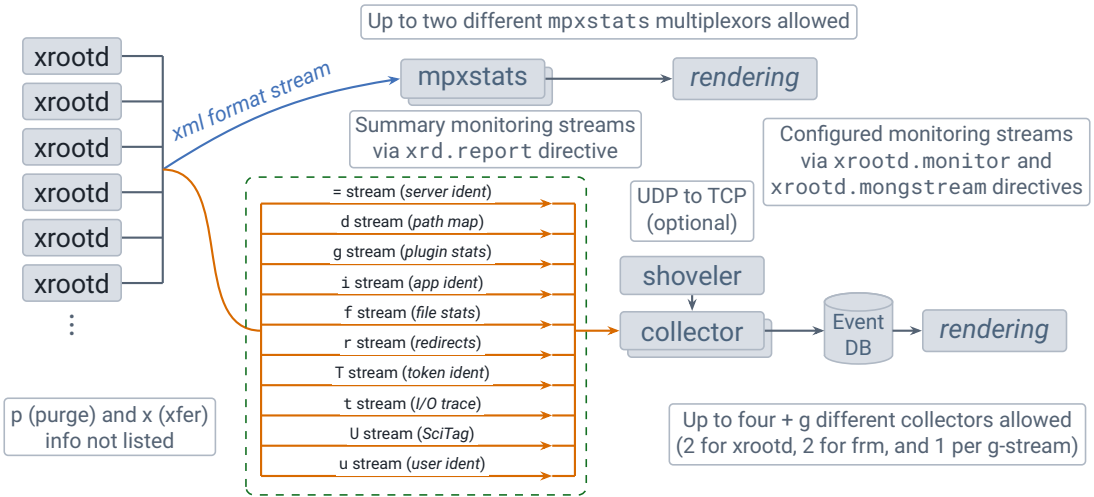
08 Grafana Dashboards

09 Summary

Section 1

XRootD Monitoring Overview

XRootD Monitoring Streams



Monitoring Record Types

Code	Stream	Option	One packet holds
Continuous streams: many events per packet			
`f`	file statistics	`fstat`	opens, closes, in-flight I/O, disconnects
`t`	I/O trace	`io`, `iov`	every read, readv and write
`r`	redirections	`redir`	where each request was sent
`g`	plugin events	`pfc`, `tpc`, `ccm`, …	one text line per event
Map messages: one event per packet, needed to decode the rest			
`=`	server identity	`ident`	site, host, port, instance, version
`u`	user login	`auth`, `user`	dictid → user, DN, VO, role
`d`	file path	`files`	dictid → user and path
`i`	app info	`info`	dictid → client-supplied string
`T`	token identity	`user`	dictid → token subject, VO, groups
`U`	SciTags	`user`	dictid → experiment and activity ids

Two more map records, `p` and `x`, carry FRM events—rarely used.

Monitoring Configuration Directives

XRootD provides summary monitoring and detail monitoring.

Channel	Configuration Directive	Information Type	Output Format
Summary	`xrd.report`	periodic statistics	Flat Text / XML
Detailed	`xrootd.monitor`	per-operation events	binary UDP streams
G-stream	`xrootd.mongstream`	plugin-specific events	binary or JSON/XML, UDP

Detailed monitoring is what the monitoring collector has to process.
It's controlled by the `xrootd.monitor` and `xrootd.mongstream` directives.

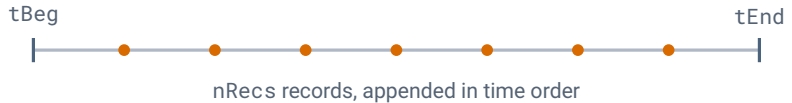
Configuring Detailed Monitoring

```
xrootd.monitor all auth flush 30s ident 5m window 5s fstat 60 lfn ops ssq
 ↪  xfr 5 dest files info user redir fstat collector.example.org:9930

xrootd.mongstream pfc tpc use send json dflthdr collector.example.org:9931
```

- `all auth` — monitor everyone, enable authentication detail in the user map
- `flush 30s` — bound buffering delay to 30 s
- `window 5s` — set the timing window granularity
- `ident 5m` — re-announces server identity every 5 minutes
- `fstat 60` — emit file statistics every minute
  - `lfn`, `ops`, `ssq` enrich `fstat` records (`io`/`iov` would add the per-I/O `t`-stream)
- `fbuff`/`gbuff`/`mbuff`/`rbuff` — set the size of the UDP datagrams per stream
- `dest` — route chosen streams to a collector (up to two destinations)
- `xrootd.mongstream` — pick CGI or JSON headers for the plugin g-streams

Monitoring Time Windows

Monitoring records on the wire do not carry individual timestamps. Instead, `isTime` records mark the boundaries of the time window: everything between two marks has the same timestamp.

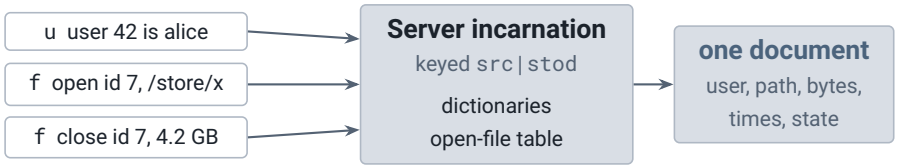


- `window 5s` in the configuration sets the time window mark interval
- Additionally, a wrapping 0–255 counter named `pseq` is in the packet header
- The g-stream packet headers also carries a `tBeg`/`tEnd` pair

Collectors must reconstruct timestamps using `pseq` and time window markers

Server Incarnation and Correlation

Events in the monitoring streams name user, path and app by dictid—the collector uses the map records to reconstruct the information.



- A server incarnation is built from its address plus the server start time
- The collector state is keyed on the server incarnation, map records are bound it
- Dictids are unique within a server incarnation

The collector produces a JSON/OTel document when a terminal record is received

Section 2

Optimizing Network Configuration

UDP Packet Loss: Sources, Reasons, and Mitigations

Unlike TCP, UDP has no retransmission. If a UDP datagram is dropped anywhere along the network path to the destination, it's gone permanently.

Where datagrams get dropped	Reason	Mitigation
sender's stream buffer	record does not fit the buffer	size buffers to the records
the network	congested switch queues	keep the path short
NIC ring, kernel backlog	bursts outrun the kernel	raise ring and backlog
socket receive buffer	collector busy, buffer full	big `rcvbuf`—main lever
collector's own queue	the decoder falls behind	the collector's design

In our experience, most packet loss happens on the busy network

Optimizing Kernel Configuration

```
# /etc/sysctl.d/99-xrootd-udp.conf     -- on the collector host
net.core.rmem_max = 134217728           # ceiling: 128 MiB
net.core.netdev_max_backlog = 30000     # default 1000

# /usr/lib/systemd/system/xrdmoncollect.socket
[Socket]
ReceiveBuffer = 16M
```

- A large kernel receive buffer helps to absorb network bursts and collector restarts
- Raising `rmem_max` doesn't cost memory until packets actually queue
  - It is a ceiling, not a fixed allocation
  - It also needs privileges to ask, unprivileged requests are silently capped
- The collector's systemd unit is used to create the socket as root
  - Uses `ReceiveBuffer=` while privileged, so it's not capped
  - Holds the socket across collector restarts to prevent loss of data

Optimizing XRootD Configuration

One report is one UDP datagram, which can be up to 64 KiB of data and may be fragmented based on the MTU of the network. A wire frame carrying 1472 bytes would split a 64 KiB datagram into ~44 fragments:

Reassembly is all-or-nothing: if the **?** never arrives, the kernel discards the other 43.

- At 0.1% frame loss that is over 4% of fstat datagrams lost—40× amplification
  - It's very important to set small buffer sizes to avoid fragmentation
- XRootD's default for `fbuff` is 64K, and it does not follow `mbuff`, so you must set all of them to a reasonable size if sending UDP to a collector not on localhost
- What is a reasonable size? With MTU=1500, IPv6 packets can fit at most 1452 bytes after headers and other overheads are subtracted, so using 1472 bytes as commonly used for IPv4 would fragment packets on a dual-stack path. For safety, use ≤1400 bytes

Section 3

WLCG Monitoring Requirements

List of Requirements

For every completed operation—whether it succeeds or fails—XRootD must generate one self-contained operation report carrying the fields needed to reconstruct what happened:

Group	Required fields
who	`user` (DN or token subject), `vo`, `auth_method`, `activity`
where	`server_*` (name, IP, site), `client_*` (IP, hostname, site), `ip_version`
what	`file_name`, `operation_type`, `bytes`, `start_time`, `end_time`, `is_local`
outcome	`operation_state`, `error_message`, `error_category`

Modelled on the FTS transfer events and delivered as JSON/OTel documents.

CMS Data Popularity

CMS is the main consumer behind these requirements.

- Which datasets are read, how much, by whom, from where
- It can drive important decisions: replica placement, XCache sizing, and the deletion campaigns to reclaim space from cold data
- CMS has built it on XRootD detailed monitoring since the AAA federation days: UDP streams, external collectors, then aggregation at CERN
- Map file names to a dataset, user, and site, say who and where, so that the data popularity dashboard may be built on top of it

Data popularity is a per-operation question that cannot be answered by summary monitoring alone, the detailed monitoring and collector are needed for this

What XRootD Already Provides

Most of the list was already on the wire.

Requirement	Where it can be found
file name, user, client address	the map streams u, d
operation type and bytes	`f`-stream, at file close
start and end time	`f`-stream reporting windows
server name, site, address	the = identity stream
auth method, VO, IP version	the u-stream CGI tail
activity tag	app info and SciTags (`i`, U)
operation state, error message, error category	nowhere

Error monitoring is a real gap in the server that needed to be addressed

Error Monitoring with Backward Compatibility

The detailed streams feed MonALISA and the OSG shoveler/collector. We do not control them, so we must maintain backward compatibility.

A new f-stream record type

- Unmodified collectors pass it on
- Decoders skip unknown record types
- Nobody has to upgrade

A new e-stream for errors

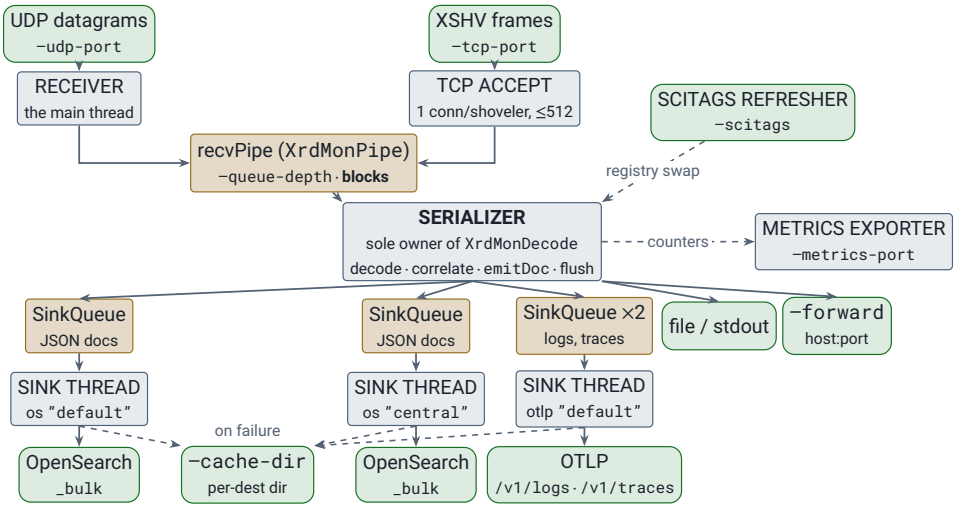
- Rejected by their packet dispatch
- Every consumer breaks until it is fixed
- So we did not do this

Verified by looking at OSG collector's Go source code

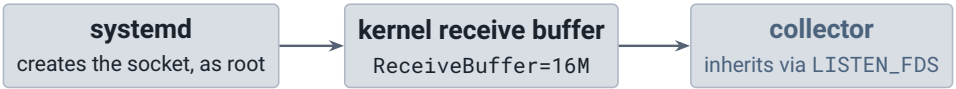
Section 4

Collector Design and Architecture

The Collector Architecture



Integration with systemd



- The daemon runs unprivileged, with an empty capability set. The socket exists before it starts—and its buffer is the uncapped one from the previous section.
- Restart, upgrade, or crash: systemd holds the socket, datagrams queue in the kernel buffer, and the new process drains them. Zero-loss restarts, the same trick HAProxy uses.
- Not available in containers—there the host's `rmem_max` is the real ceiling.

The end-to-end test plays PID 1 itself: a file operation sent while nothing is running still comes out as one document.

Memory Management and LRU

Correlation state is the one unbounded input: every open waits for its close, and there are user/path records that can go stale. Bounding it took many iterations:

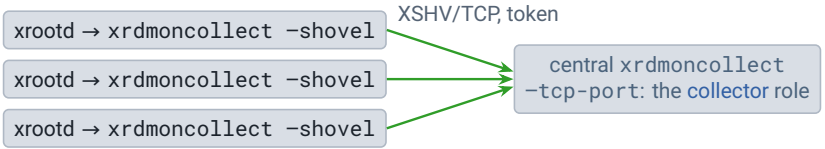
1. `-max-entries`, one million per map—abandoned: hash-order eviction throws away live entries while stale ones survive.
2. A byte budget with LRU eviction—right idea, wrong number: it bounded an estimate that ran 4–8× low. Sites that set 256M passed a gigabyte and were OOM-killed.
3. Redefine the cap on measured RSS: evict oldest-first until back under. The first version could not use roughly half its allowance—now it steers into a band just below the cap.
4. TTLs for what LRU cannot see: `-user-ttl` and `-user-idle-ttl` bound how long a session outlives its disconnect.

Flood test, 2.5 M unclosed opens: 7869 MB uncapped; 156 MB at the default cap.

Deployment Architectures

A — site collector: plain UDP on the LAN

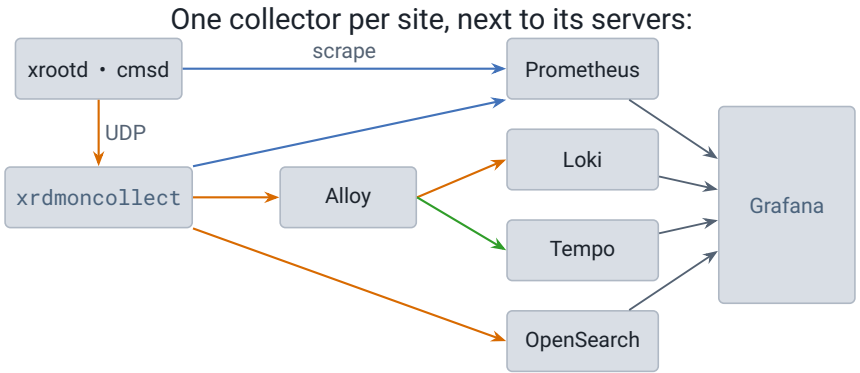
B — WLCG central: a shoveler on every node



Shovelers spool to disk while the centre is unreachable; each frame keeps the datagram's original source, so per-server accounting still works.

One binary, playing either part—shoveler, or collector.

Monitoring Stack Deployment Architecture



The collector POSTs to OpenSearch directly, or goes through Alloy for logs and traces—both paths are tested. Every Grafana panel in this section reads one of these four sources.

Section 5

Collector Features and Configuration

Ingest Configuration

One INI file: `/etc/xrootd/xrdmoncollect.cfg`. Every key mirrors its long command-line option, and the command line wins.

```
[xrdmoncollect]
port         = 9930   # UDP; kept open by the systemd socket unit
rcvbuf       = 16M    # the SO_RCVBUF request
queue-depth  = 64     # packet batches between receiver and serializer
flush-count  = 500    # packets per batch...
flush-secs   = 5      # ...or the max age of a partial one
metrics-port = 9931   # the collector's own /metrics endpoint
```

- These are the knobs from the architecture section: the receive buffer, the bounded pipe, and the batch that becomes one POST.
- The defaults are chosen to be left alone—`rcvbuf` is the one to revisit on a busy collector.

State Management

```
state-file = /var/lib/xrootd/xrdmoncollect-state.json
state-ttl  = 15m     # discard an older snapshot
user-ttl   = 300     # reap a user this long after its disconnect
file-ttl   = 7200    # reap an open whose close was lost
server-ttl = 86400   # reap an idle server incarnation
```

- Dictionaries are re-sent only every `ident 5m`: a restarted collector would decode blind until then. The state file closes that window—saved on shutdown, reloaded on start.
- It is written only once the serializer has been joined and the state is quiescent; the shutdown sequence exists for this.
- The TTLs cover what packet loss leaves behind: an open whose close never came, a session whose disconnect never came.

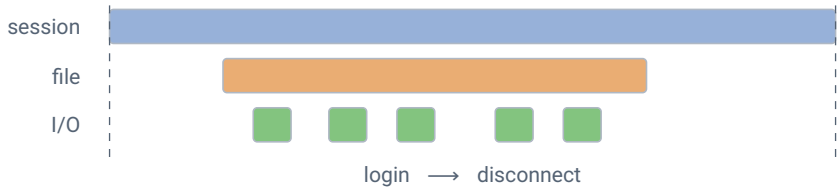
Shoveler Mode: UDP to TCP

```
[xrdmoncollect]                  ; on every server node
shovel       = collector.example.org:9931
shovel-token = @/etc/xrootd/shovel.token
spool-max    = 1G
[xrdmoncollect]                  ; on the central collector
tcp-port     = 9931
tcp-token    = @/etc/xrootd/shovel.token
```

- Each TCP frame carries the datagram's original source address, so per-server state and loss estimation still work centrally—decoding happens once.
- While the centre is unreachable, framed buffers spool to disk, capped by `spool-max`, oldest evicted first.
- In shovel mode the decode and sink keys are ignored, with a warning—so both roles can share one config file.
- `@<file>` tokens are read from disk and stay out of `ps` and `argv`.

Sessions, Spans, and Traces

```
sessions = true   # one rollup document per client disconnect
spans    = true   # OTel spans: session and file operation
traces   = true   # t-stream: every read and write (high volume)
```



Trace and span ids are synthesized from the wire (`src` | `stod` | `user` | `fileID`)—deterministic, with no id allocator, idempotent across collector restarts.

OpenSearch and OpenTelemetry

```
os-url        = https://opensearch.example.org:9200
os-index      = xrootd-file-ops
os-token      = @/etc/xrootd/opensearch.token
os-datastream = false

otlp-url   = http://alloy.example.org:4318   # /v1/logs, /v1/traces
otlp-token = @/etc/xrootd/otlp.token

cache-dir  = /var/lib/xrootd/moncollect      # on-failure disk cache
```

- The same documents in two dialects: OpenSearch `_bulk` (rolling index or data stream), and OTLP/JSON for Alloy or any OpenTelemetry collector.
- One serialization feeds every sink; each destination caches separately under `cache-dir`, replayed oldest first.

Multiple Destination Support

The typical use: keep the site's own monitoring *and* feed a central collector.

```
[otlp "wlcg"]
url   = https://otel.wlcg.example:4318
token = @/etc/xrootd/wlcg.token

[opensearch "central"]
url        = https://es.wlcg.example:9200
index      = wlcg-file-ops
datastream = true
```

- One UDP ingest, one correlation state, one memory budget, one serialization—shared by every destination.
- A named destination inherits nothing: an inherited credential would be POSTed to an endpoint it was never meant for.
- Each destination runs its own thread, queue, retry state and cache directory, and labels its metrics `destination="..."`.

Enrichment: Site, Dataset, SciTags

```
site    = CERN-PROD
dataset = ^/store/[^/]+/[^/]+/([^/]+)/   # first capture group
scitags = https://www.scitags.org/api.json
scitags-refresh = 3600
```

- The site is deliberately not on the wire: `all.sitename` names the storage cluster (eosalice, eoscms), and one site holds several. One collector per site, so it comes from here—as a label on every series and every document.
- `dataset`: the first capture group, matched against each file path, becomes `xrootd.dataset`—what the data-popularity dashboards group by.
- The U stream carries numeric SciTags ids; the registry maps them to experiment and activity names, refreshed hourly.

Filtering and Labelling

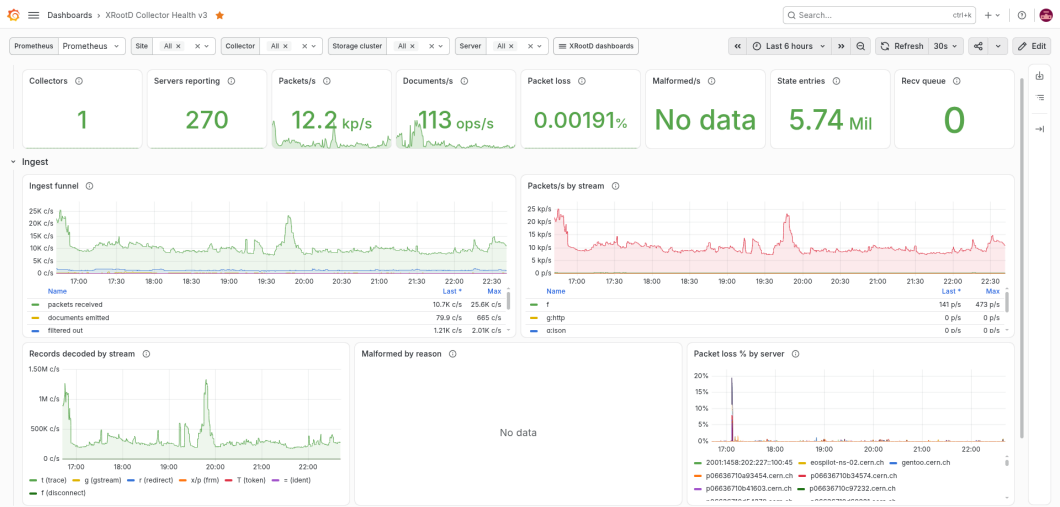
```
[filter "eos-internal"]
user     = daemon, root, ~^[0-9]+$
authprot = sss
action   = tag        ; or: drop | keep
label    = internal

[filter "keep-vo"]
vo     = ~.+
action = keep         ; keep beats drop
```

- Conditions AND together
- A value is an OR-list of exact strings, globs, or ~-regexes, negated with `!`.
- Filtering runs at the very end: correlation and the Prometheus aggregates update first, so a dropped document is still fully measured and reported in the collector metrics
- An unrecognised key is fatal at start-up
- A rule that silently matches nothing may be indistinguishable from one that works

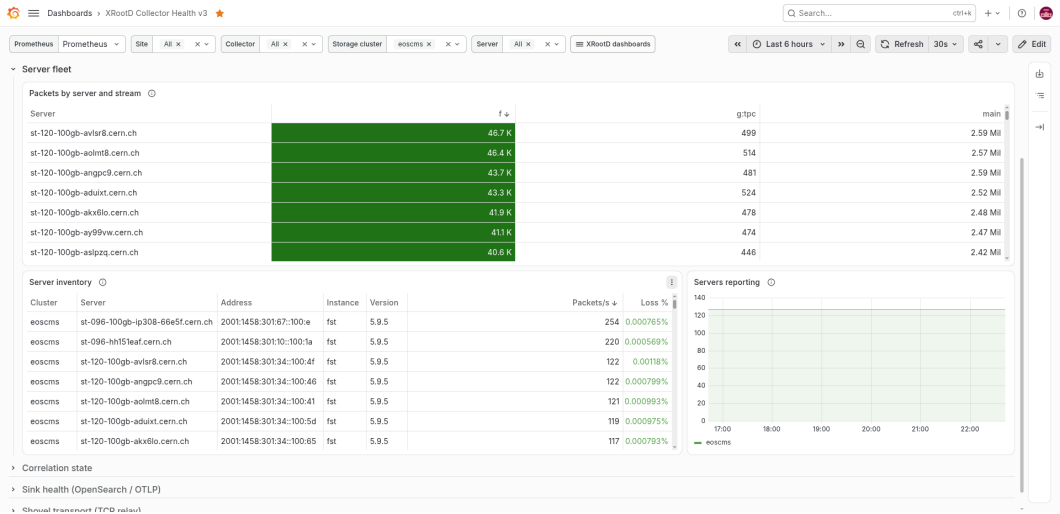
Section 6

Collector Health Metrics

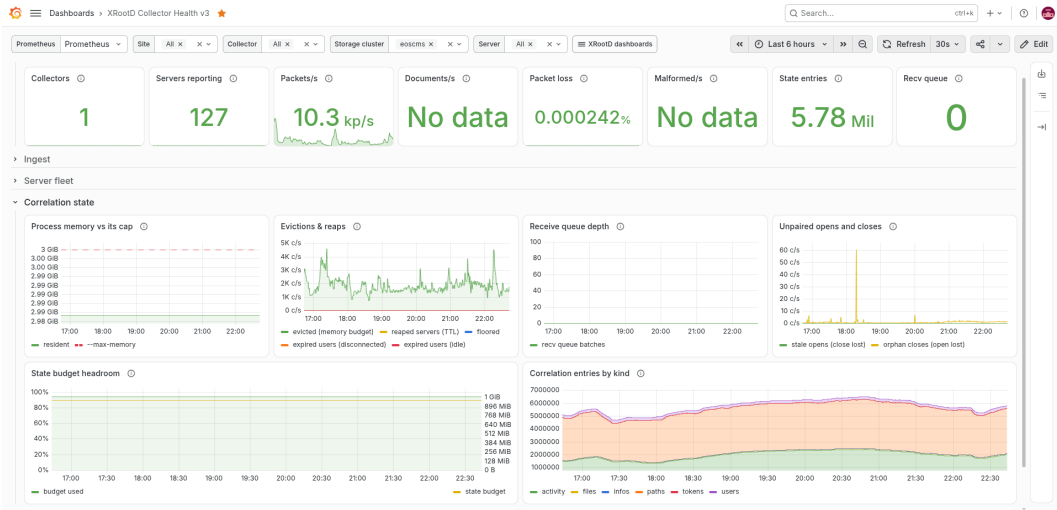
Ingest Metrics



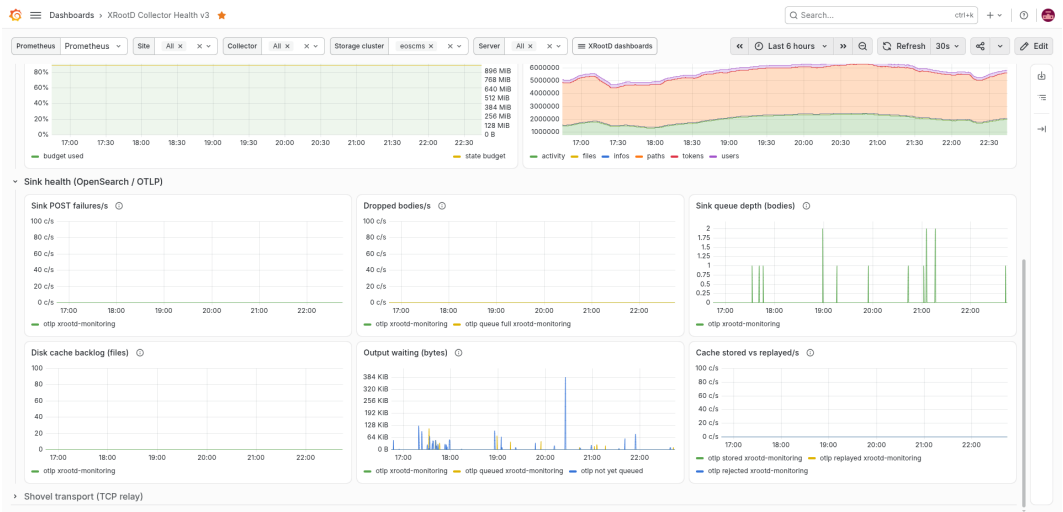
The Server Fleet



Correlation State



Sink Metrics



Section 7

OpenTelemetry Semantic Schemas

Resource and Attributes

```
{"resource":   {"service.namespace": "eosatlas",
                "service.instance.id": "srv1.example.org:1094",
                "server.address": "srv1.example.org",
                "xrootd.site": "CERN-PROD",
                "xrootd.server.incarnation": 1700000000},
 "eventName":  "xrootd.read",
 "timeUnixNano": "1751450432000000000",
 "traceId": "9f1c8b0d...", "spanId": "3ab4c1d2...",
 "attributes": {"file.path": "/store/data/Run2026A/file.root",
                "xrootd.dataset": "Run2026A",
                "client.address": "wn.example.org",
                "user.name": "alice", "xrootd.vo": "atlas",
                "scitags.activity": "analysis",
                "xrootd.operation.state": "Successful",
                "xrootd.operation.duration": 300.25,
                "xrootd.read_bytes": 805306368}}
```

`resource` describes the sender, `attributes` the event. Vendor keys live under `xrootd.*` and `scitags.*`; everything else is OTel semconv. Abridged—the full README example carries some sixty fields.

One Document, Two Dialects

Aspect	OpenSearch `_bulk`	OTLP/JSON
nested objects	dotted keys, queryable	`KeyValue` arrays
64-bit integers	JSON numbers	strings (proto3-JSON)
scalar arrays	JSON array	typed `arrayValue`
envelope fields	top-level document keys	native LogRecord fields
grouping	one document per event	grouped by resource
spans	ordinary documents	split to `/v1/traces`

Equivalent content, different shape—a schema test asserts no field reaches one sink but not the other.

WLCG Field Mapping

The wishlist from earlier, closed out field by field:

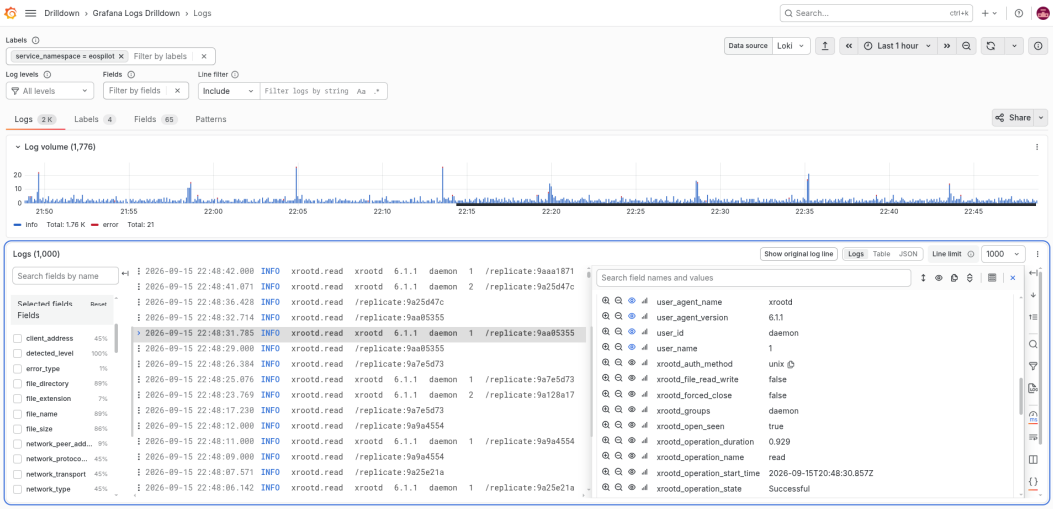
WLCG field	In the document	Needs
file_name	`file.path` (+ name, dir, ext)	`fstat lfn`
operation_type	`xrootd.operation.name`	`fstat xfr`
operation_state	`xrootd.operation.state`	the terminal report
error_message	`error.type`	the terminal report
user, vo	`user.name`, `xrootd.vo`	u map, T token
auth_method	`xrootd.auth.method`	`monitor auth`
activity	`scitags.*`	U map + registry
client_site	`xrootd.client.site`	login CGI (client 6.3+)
start / end time	`...start_time`, `@timestamp`	windows, interpolated
bytes	`xrootd.*_bytes`	`fstat xfr`
is_local	`xrootd.is_local`	derived from domains

The full mapping, with the server directive each field requires, is a table in the README.

Section 8

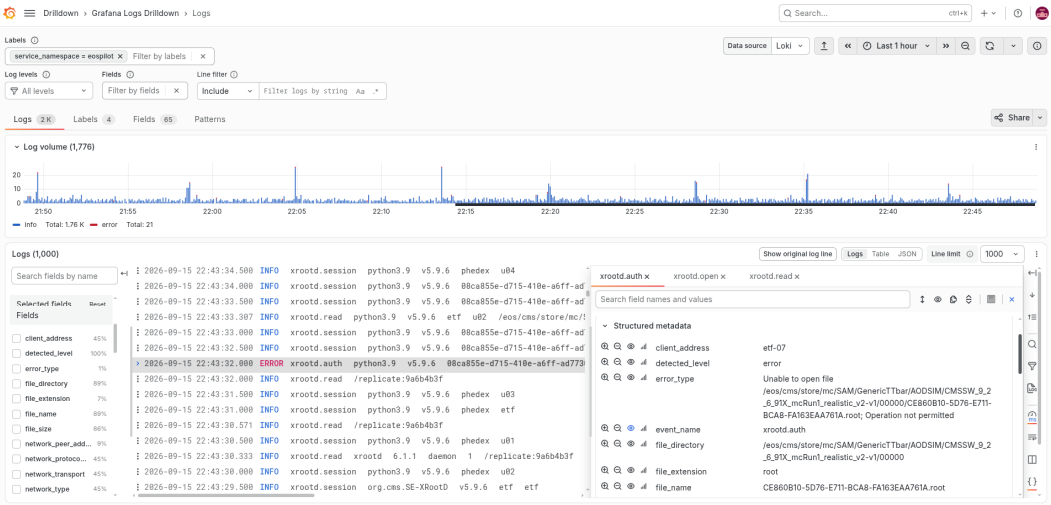
Grafana Dashboards

Grafana Integration: Logs

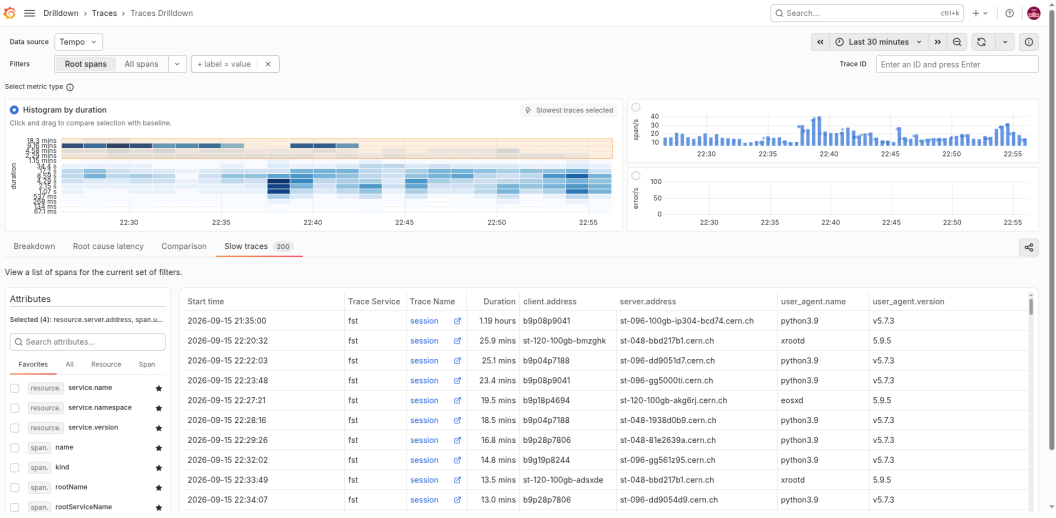


Logs Drilldown: Fields

A Failed Operation, in Production

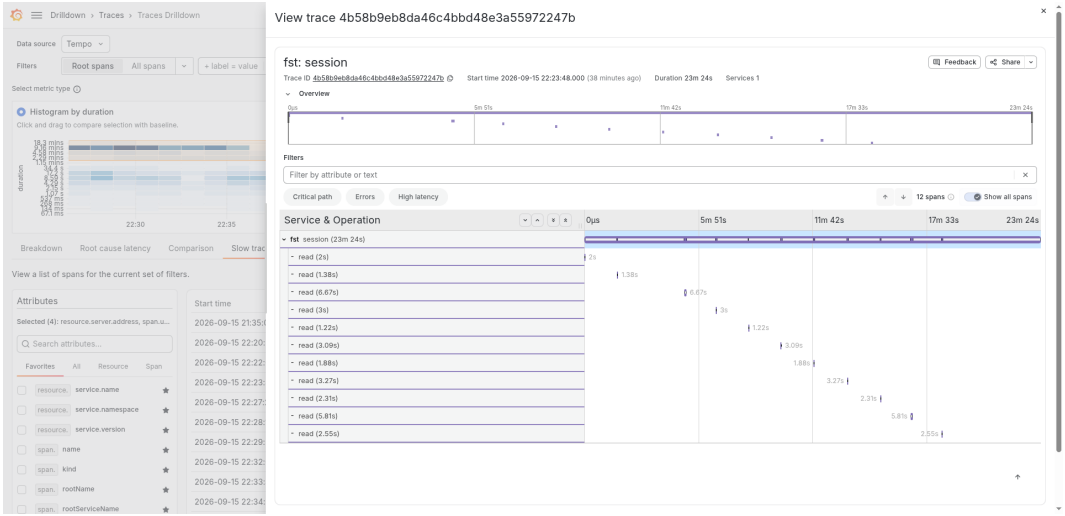


Grafana Integration: Traces

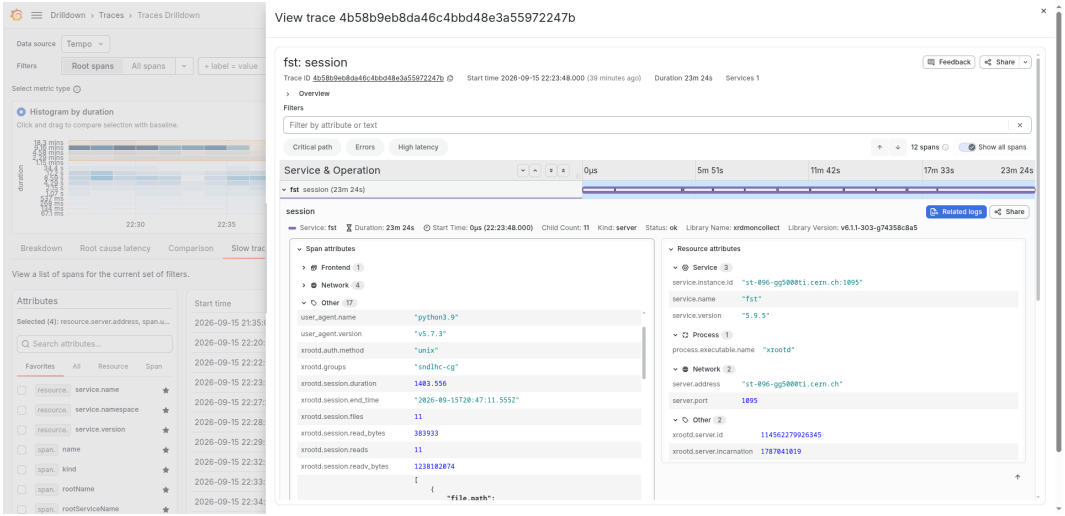


Start time	Trace Service	Trace Name	Duration	client.address	server.address	user_agent.name	user_agent.version
2026-09-15 21:35:00	fst	session	1.19 hours	b9p08p9041	st-096-100gb-ip304-bcd74.cern.ch	python3.9	v5.7.3
2026-09-15 22:20:32	fst	session	25.9 mins	st-120-100gb-bmzghk	st-048-bbd217b1.cern.ch	xrootd	5.9.5
2026-09-15 22:22:03	fst	session	25.1 mins	b9p04p7188	st-096-dd9051d7.cern.ch	python3.9	v5.7.3
2026-09-15 22:23:48	fst	session	23.4 mins	b9p08p9041	st-096-gg5000ti.cern.ch	python3.9	v5.7.3
2026-09-15 22:27:21	fst	session	19.5 mins	b9p18p4694	st-120-100gb-akg6rj.cern.ch	eosxd	5.9.5
2026-09-15 22:28:16	fst	session	18.5 mins	b9p04p7188	st-048-1938d0b9.cern.ch	python3.9	v5.7.3
2026-09-15 22:29:26	fst	session	16.8 mins	b9p28p7806	st-048-81e2639a.cern.ch	python3.9	v5.7.3
2026-09-15 22:32:02	fst	session	14.8 mins	b9g19p8244	st-096-gg561z95.cern.ch	python3.9	v5.7.3
2026-09-15 22:33:49	fst	session	13.5 mins	st-120-100gb-adsxde	st-048-bbd217b1.cern.ch	xrootd	5.9.5
2026-09-15 22:34:07	fst	session	13.0 mins	b9p28p7806	st-096-dd9054d9.cern.ch	python3.9	v5.7.3

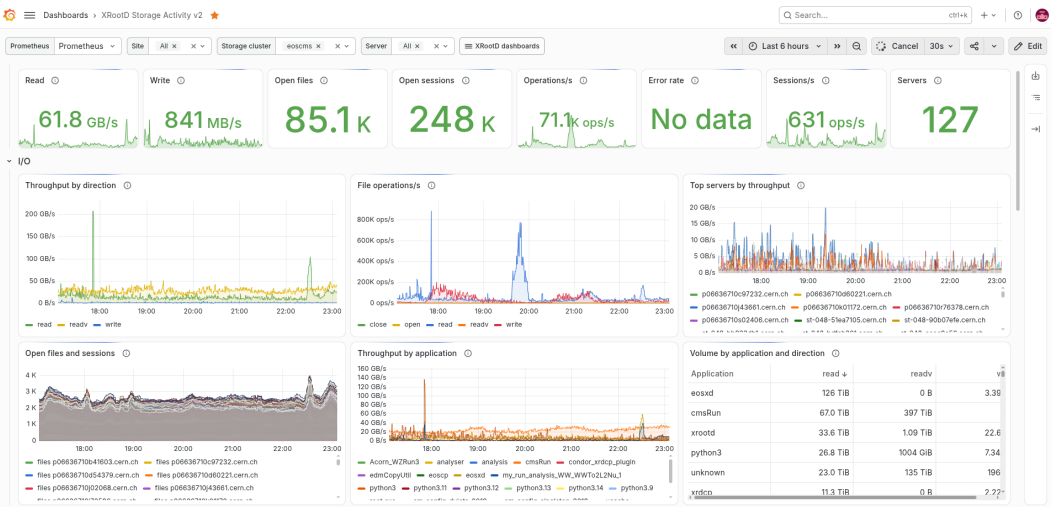
A Session, as a Trace



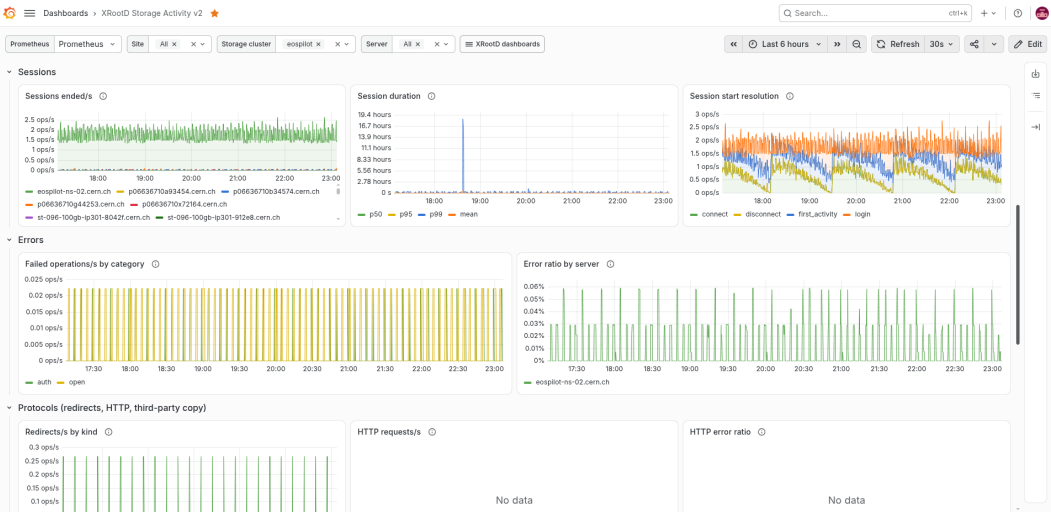
One Span, Expanded



Storage Activity



Storage Activity: Sessions and Errors



Section 9

Summary

Work in Progress

What's presented today is the result of a few very intense months of work with AI. It's a prototype, but it works, and it will definitely change between now and the release in December.

- Field names, config keys and dashboards may still shift as the OpenTelemetry semantic conventions and early feedback settle.
- Known edges: `–forward` and the file sink are single-destination; an I/O record arriving before its file's open lands in the wrong trace; part of the serialization cost remains.
- The wire-format additions are the stable part: the terminal-error records are in, and old consumers keep working, byte for byte.
- Try it, and complain now: this is the window in which feedback is cheap to act on.

Release in XRootD 6.3

- ✔ `XrdMetrics` registry and `XrdHttpMetricsExporter`: pull, Pushgateway, OTLP push
- ✔ `xrdmoncollect`: documents, traces, metrics—shoveler and collector in one binary
- ✔ Terminal error reporting on the f-stream
- ✔ A deployment guide (`DEPLOY.md`), network tuning guide, and man page
- ✔ All dashboards ship with XRootD, ready to import into Grafana
- ✖ Client-side metrics—after 6.3

Please try the release candidate when it comes out, and send your feedback!

Questions?