Erasure Coding & QoS features

Mihai Patrascoiu
on behalf of the EOS team

EOS Storage Properties

- **Layout** – specifies how a file will be physically stored

  plain, replica, raiddp, raid6 (n, k), archive (n, 3)

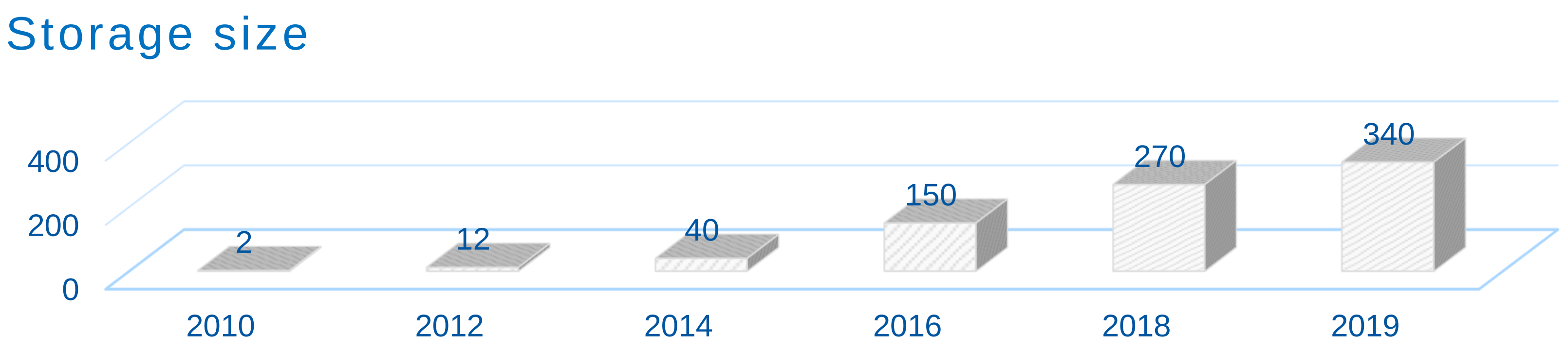
- **Number of replicas**
- **Checksum** *md5sum, adler32, crc32, crc32c, sha1*
- **Block Size + Block Checksum**
- **Placement policy** – physical file placement on disk servers

  scattered, gathered:<geotag>, hybrid:<geotag>

Placement documentation: File Geoscheduling

EOS Storage @ CERN

- 17 instances (5 LHC, 9 CERNBox, Media, Public, Backup)
- 340 PB / 5.67B files / 1600 storage nodes
- Production setup uses 2-replica layout



EOS Workshop 2020: CERN Disk Storage Services

Erasure Coding

- Way of encoding data into blocks + redundancy info, to be stored in different locations or storage media
- Files are split into data and parity blocks: EC(k+p, k)
- EC provides resilience and high availability at the cost of more IOPS and bandwidth
- Available in EOS via the *raid6* layout (using jerasure lib)

EOSALICEDAQ

- EOSALICEDAQ conversion experiment
- Conversion from 2-replica to RS(12, 10) [12 blocks, 2 parity, md5sum]
- Duration: 84 hours (600 TB / day encoded)
- Encoding of 2.4PB data → 2PB space saved

Andreas Peters - CHEP 2019: Erasure Encoding

EOSALICEDAQ – extrapolation

- 69 machines, 10 Gbit Ethernet, saturated reading 7GB/s + writing 7GB/s

⇩

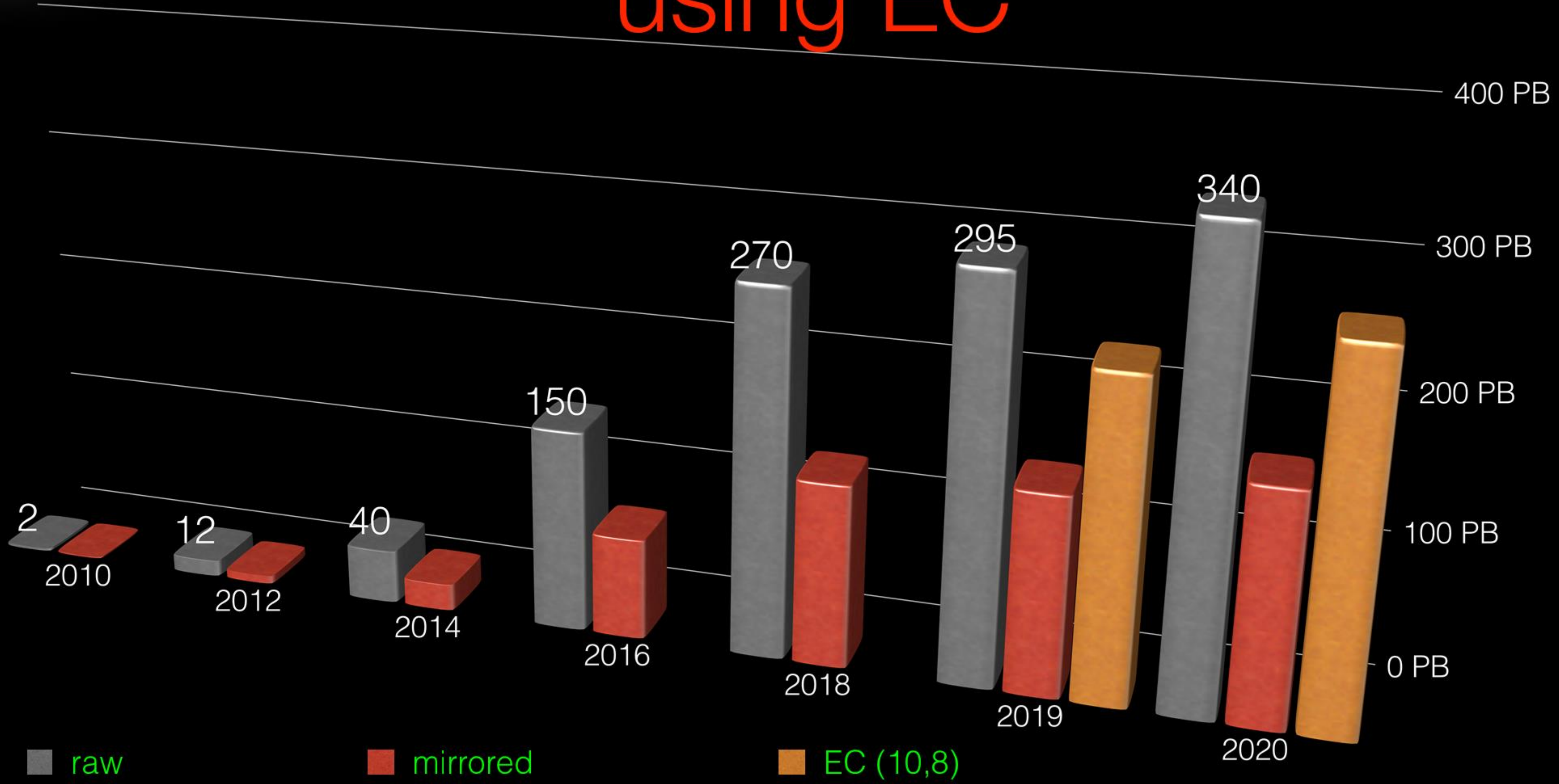
- 1000 machines ~ RS(10,8) encode 8 PB / day @ 100GB/s reading

- 10% only as background activity → 800 TB / day of free space

  → 24PB / month

Andreas Peters - CHEP 2019: Erasure Encoding

Storage Volume increase using EC



Files Stored	5 Bil
Storage Nodes	1500
Hard Disks	60k
Raw Space	340 PB
Single Instance	60 PB
IO Streams	>100k

IT-ST

Erasure Encoding – conclusions

- Erasure Coding – ideal for large files
- EC implies increased network traffic (read+write amplification)
- EC brings increased read latency for non-specialized clients
- Remains to test various EC parity choices

Andreas Peters - CHEP 2019: Erasure Encoding

QoS data management

> *Accommodate different use-cases with storage policies that can achieve the cheapest solution*

- Storage policy according to system rules or user-defined
- Leverage storage properties and QoS mechanisms

Storage policies – examples

- Store files in replica or erasure encoding format
  - Store only files unused for 6 months in EC
    - Store only files unused for 6 months and larger than 5GB in EC
- Transition to tape if inactive for # months

QoS development and future directions

QoS classes in EOS

- Abstraction entity over existing storage properties
- QoS API must allow classes to be:
  - Discoverable
  - Configurable
  - User applicable on a per file/directory basis
- Transitioning is supported between QoS classes

How do QoS classes work?

- A QoS class configures the following properties
  - Layout
  - # Stripes
  - Checksum type
  - Placement type
- A QoS class provides guarantees

  E.g.: redundancy level, geolocation
- QoS transitions from one class to another must be explicitly allowed

  E.g: disk $\rightarrow$ tape,

  tape $\rightarrow$ disk+tape, tape $\nrightarrow$ disk

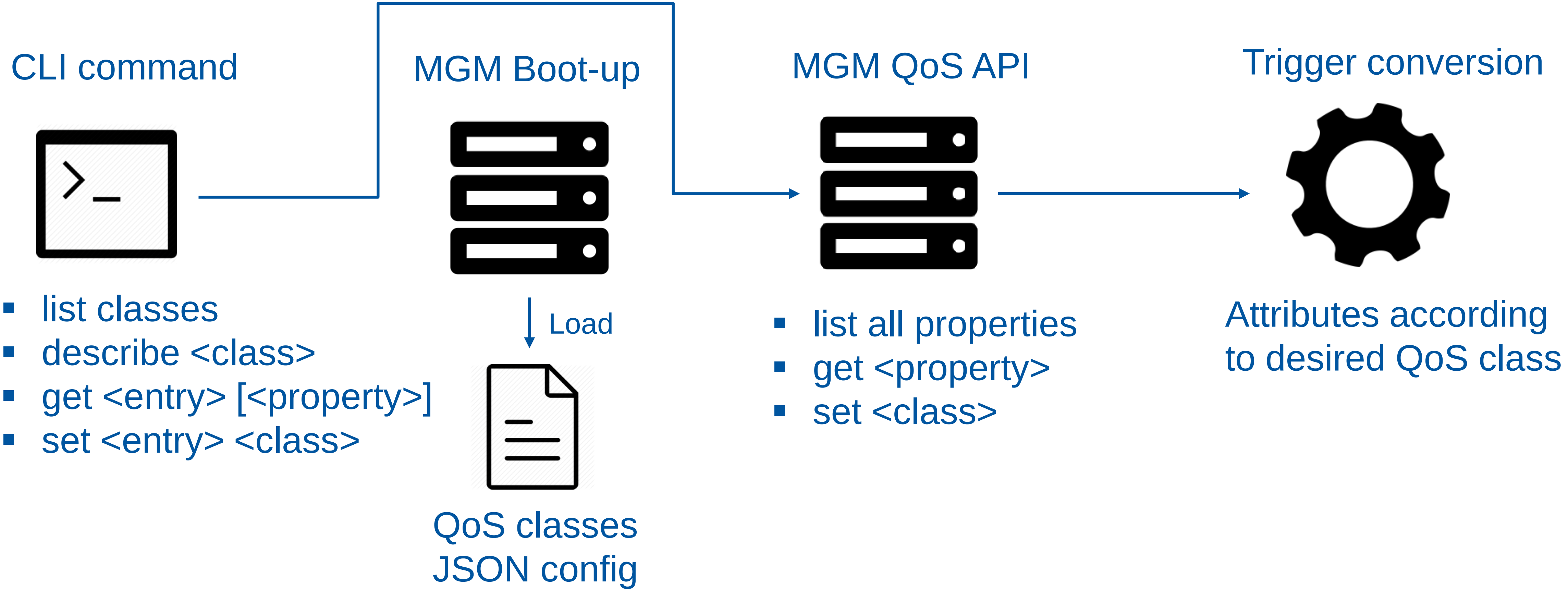
How do QoS classes work? (cont'd)

File QoS class deduced at runtime

Extended attribute for mid-transition:
user.eos.qos.target

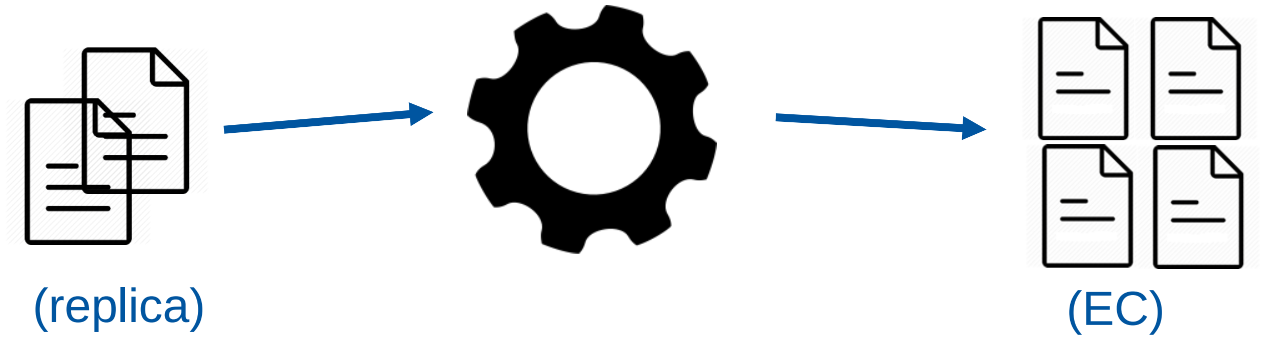
- QoS class applied to directory → propagates to files assigned in that directory
- Opaque info with desired QOS on RW-Open

QoS overview



Converter Engine

- Rewrite of the converter daemon
- One single converter instead of one per space
- Converts files from one layout/QoS class to another using ThirdPartyCopy



Converter Engine (cont'd)

- Persistent conversion jobs storage by using QuarkDB
- Jobs are fetched in batches
- Runtime scalable threadpool
- Interact via new `eos convert` command

```
$ eos convert status
Threadpool: thread_pool=converter_engine min=16 max=400
size=16 queue_size=82
Running jobs: 100
Pending jobs: 176
Failed jobs: 0
Failed jobs (QDB): 2
```

```
$   eos -j convert file /eos/xdc/test/file replica:4 adler32
<=> eos -j qos set /eos/xdc/test/file      qos_disk_replica_4

{
   "conversion_id" : "00000000000009dc:default#00650312",
   "path" : "/eos/xdc/test/convert"
}
```

Converter Engine – improvements

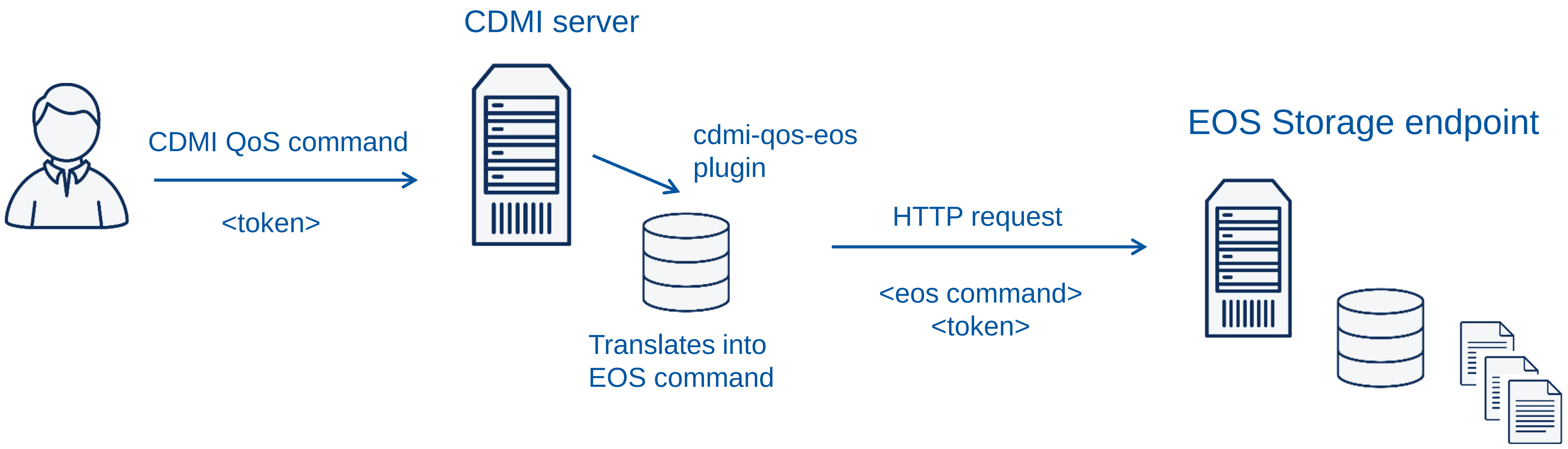
- Allow directory/bulk conversions
- Support periodic conversion rules on directories
- Testing at scale

CDMI – EOS interaction

- Achieved through CDMI QoS plugin for EOS backend
- Calls EOS HTTP interface for commands
- Requires trusted connection between CDMI server & EOS
  - Support for bearer tokens in sight (follow TPC AuthN discussion)

CDMI QoS plugin for EOS backend (repository)

CDMI – EOS interaction



CDMI QoS plugin for EOS backend (repository)

Conclusions

- EOS flexible enough to accommodate storage QoS
- Best storage policies have to be identified
- EC promising solution for storage space savings
- EC would require agreement and coordination with experiments
- CDMI gateway ready, in order to provide common QoS interface

Thank you for your time!

Icons: Computer Network Icons collection - openclipart.org

CERN

[Backup] Structure of a QoS class

- Name
- Transitions : [qos_class, qos_class, …]
- Metadata: { expected_redundancy,
  expected_latency,
  expected_geolocation: [geotag, …] }
- Attributes: { layout_type,
  nstripes,
  checksum_type,
  placement_type }

Mandatory fields

QoS property map

EOS specific

Structure compatible with INDIGO CDMI QoS specification

[Backup] QoS class example

```
{
  "name": "disk_plain",

  "transition": [ "disk_replica“ ],

  "metadata": {
    "cdmi_data_redundancy_provided": 0,
    "cdmi_geographic_placement_provided":
    [ "CH“ ],
    "cdmi_latency_provided": 75
  },

  "attributes": {
    "layout": "plain",
    "replica": 1,
    "checksum": "adler32",
    "placement": "scattered"
  }
}
```

[Backup] EOS CLI to HTTP

- CLI commands are serialized as Open call to */proc/user/* or */proc/admin/*
- Command arguments are passed as CGI

```
eos qos get /path/to/file  <=>

open /proc/user/?mgm.cmd=qos&mgm.subcmd=get&mgm.path=“/path/to/file”
```

- Command response is serialized in 3 main groups

```
mgm.proc.stdout=<string>&mgm.proc.stderr=<string>&

mgm.proc.retc=<integer>
```