S3-like object Storage

2006, RESTful, versioning / bucket / multipart upload, 2011: encryption, obj. expiration, 2015: cross region replication
2013: 2 trillion, 2021: 100 trillion objects.

BLOCK : HDD, SSD. iSCSI for VM, high performance App.
FILE : SMB, NFS. Hierarchical. File System.
OBJECT : "cold", slow. Metadata, ID, Payload. No mutable. RESTful access. Binary, unstructured.

EDS :

NFR: durability 6 nines 99.9999 %
availability 4 nines 99.99 %
100 PB

BEE : 20% small 0.5MB 60% medium 32 MB 20% large 200 MB

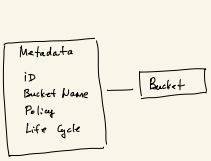
IOPS SATA 7200 rpm (100-150 IOPS)

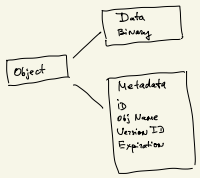
$$\frac{10'' \times 0.4}{(0.2 \cdot 0.5 + 0.6 \cdot 32 + 0.2 \cdot 200)} = 0.68 \text{ billion objects}$$

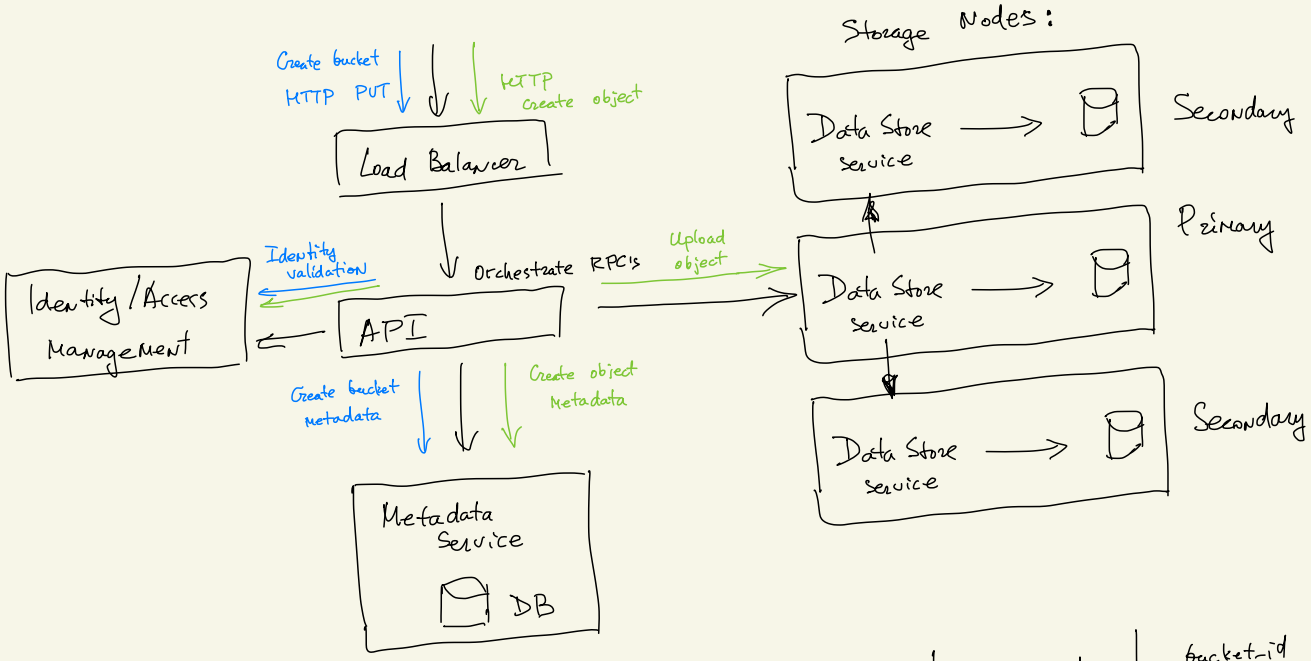
(0.4: 40% usage ratio)

+ 0.68 TB (by 1 KB metadata)

HLD : Objects do not support updates. Replace only. Immutability.
Key: URI, Value: Object.
95% read operations.
Unix-like : file → local system, fileName → iNode (pointers)



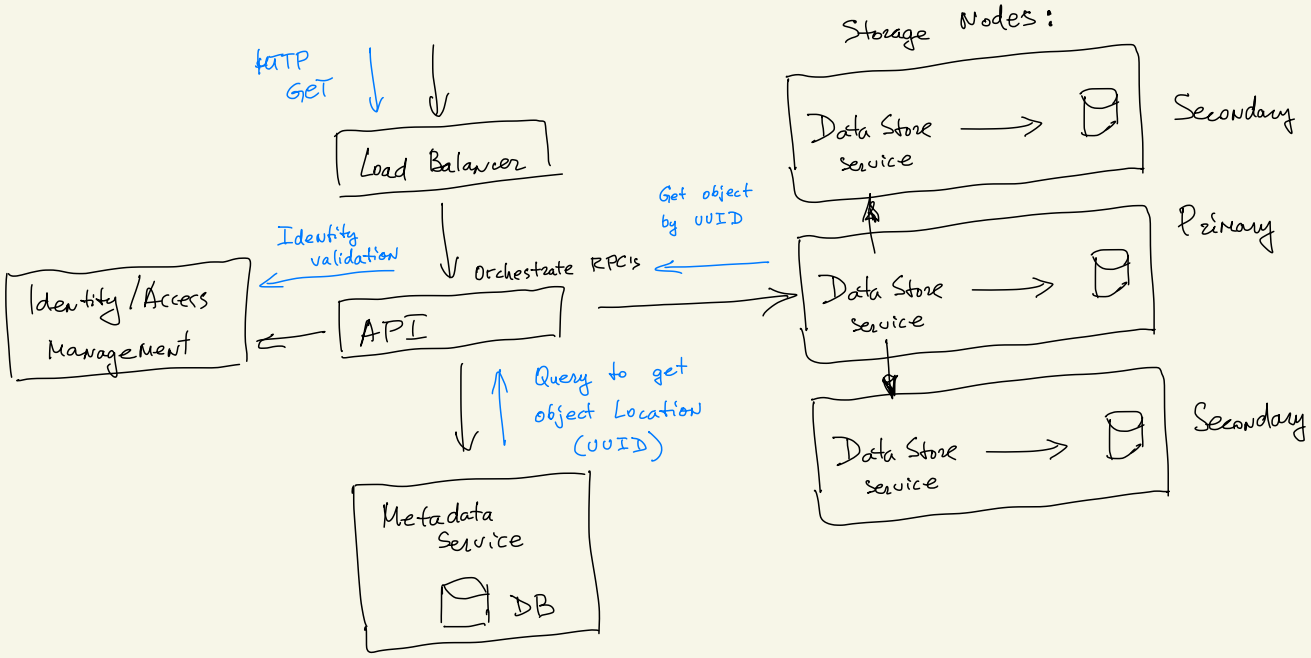




Upload

PUT /bucket_name/file.txt HTTP 1.1
Host :
Date :
Authorization :
Content-Type : text/plain
Content-Length :
Author :

object_name	object_id	bucket_id



GET /bucket_name/file.txt HTTP/1.1

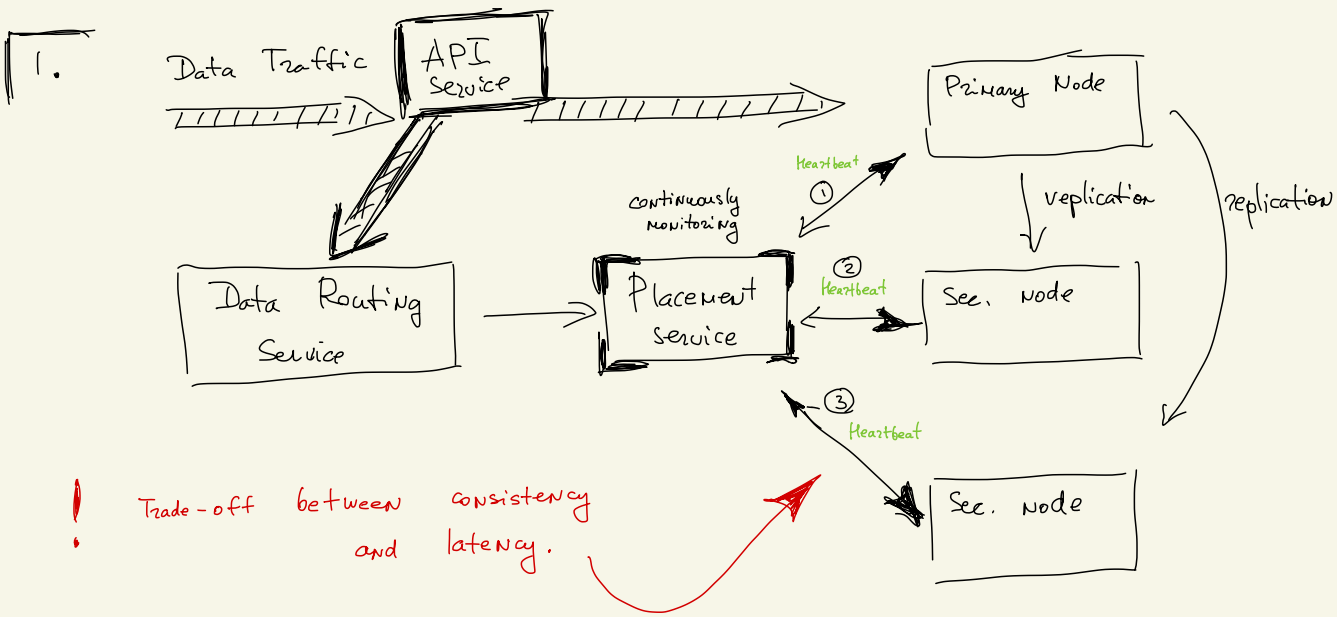
Host :

Date :

Authorization :

DDD :

1. Data Store
2. Metadata Data Model
3. Listing objects in a bucket
4. Object versioning
5. Optimizing uploads of large files
6. Garbage collection

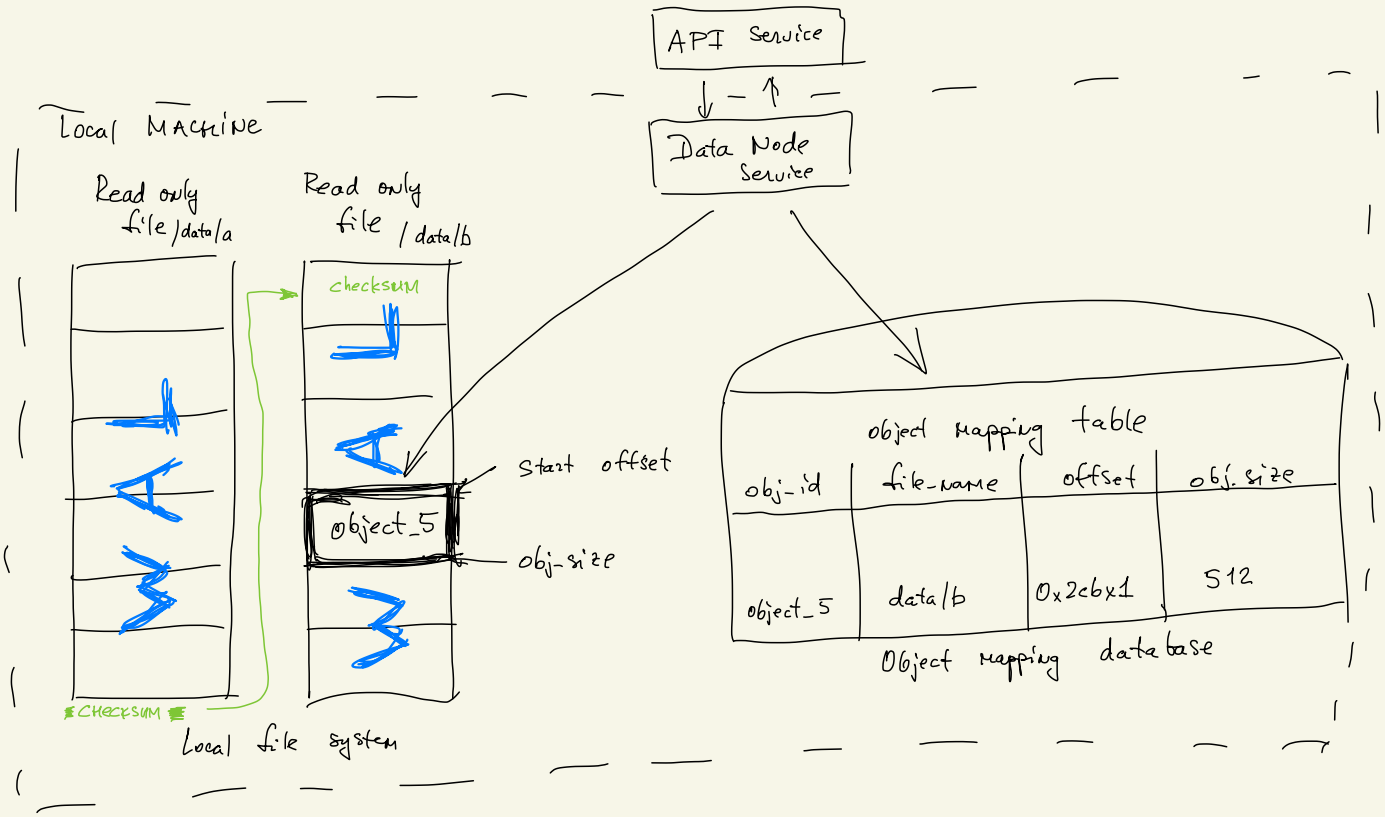


WAL (Write Ahead Log) Merge many small object into a larger file.

Object lookup:
1. Data file that contains the object
2. Starting offset
3. Object size

Data access pattern : "write once many reads."

Rock DB : file-based KV DB, but write heavy designed (SS Table)
Relational (B+ Tree) → fast for reads. SQLite → file-based DB.



Increase durability → data replication

Average spinning HD ~ 0.80% failures.

$1 - 0.0080^3 = 0.999999$ (6 nines) durability

Replicate data to different AZ's.

- Erasure coding. Calculate parities. Data reconstruction
- Correctness verification during data corruption. Verifying checksum.
  Compare "checksum of original data" with "checksum of received" data.
  MD5.

2. Metadata data model

Bucket, Object tables.

Scale Bucket table. 1 million × 10 buckets × 1 KB ≈ 10 GB.

Dataset won't fit to single DB → sharding.

~~bucket_id~~ ~~shard~~ object_id shard

[< bucket_name, object_name >] Sharding

3. Listing objects in a bucket.

s3://bucket_name/a/b/c/text.txt

```
# aws s3 list_buckets
aws s3 ls s3://bucket/abc/
aws s3 ls s3://bucket/abc/ --recursive
```

One Table
- SELECT * FROM bucket WHERE owner_id = {id}
- SELECT * FROM object WHERE bucket_id = "123" AND object_name LIKE 'abc/%'

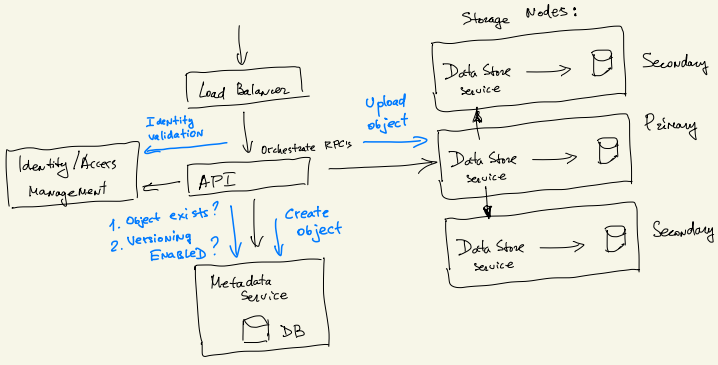
DISTRIBUTED DATA BASE [Shard 1, Shard 2, ..., Shard N

→ query1, query2, query N

query aggregation

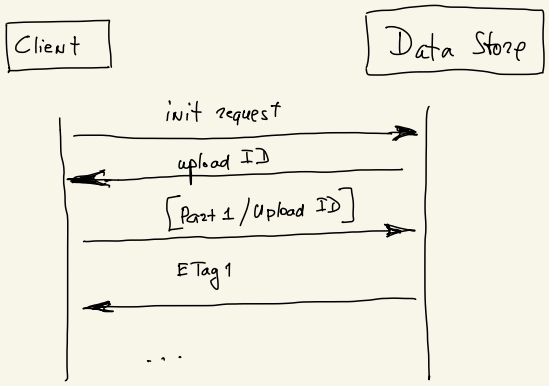
! pagination in distributed DB will be difficult

4. Object versioning



5. Optimizing Large files upload

Takes a long time if file > a dozens of GB's. Solution: divide to parts and upload them independently. After all the parts are uploaded reassembles the object from the parts. Multipart upload.



Multipart upload completion step bring all ETag's as ones.

6. Garbage collection

old parts after previous step are no longer useful.

- Lazy object deletion
- Orphan data
- Corrupted data (failed MD5 checksum verification)

Do not removing objects right away. Will use "compaction" mechanism.

↓

like "fragmentation"

Must delete from replicas.