OK S3
Строим Систему Сами

Вадим Цесько

Saint HighLoad++ 2022

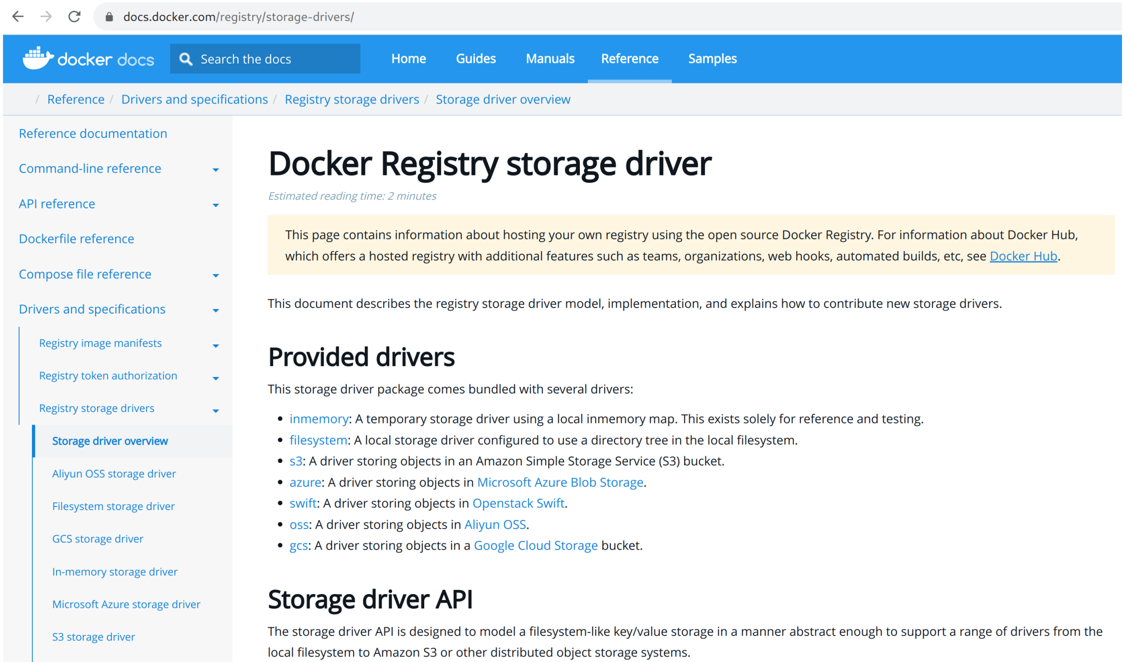
Odnoklassniki[1]

- $>$ **70M users** monthly
- $>$ **10K servers**
- **7 datacenters**
- $>$ **4 Tbps**
- $>$ **100K rps** per node (4 cores)
- **p99** $<$ **100 ms**
- **1 EB** and growing

[1] OK tech platform hosts VK Video, VK Calls, RuStore.

Why S3?

- Docker Registry
- Sonatype Nexus
- JetBrains Teamcity
- Apache Airflow
- ML/DS, RPM, CDN, static websites, autotest artifacts, dumps/traces, backups, . . .



docs.docker.com/registry/storage-drivers/

docker docs | Search the docs | Home | Guides | Manuals | Reference | Samples

/ Reference / Drivers and specifications / Registry storage drivers / Storage driver overview

Docker Registry storage driver

Estimated reading time: 2 minutes

This page contains information about hosting your own registry using the open source Docker Registry. For information about Docker Hub, which offers a hosted registry with additional features such as teams, organizations, web hooks, automated builds, etc, see Docker Hub.

This document describes the registry storage driver model, implementation, and explains how to contribute new storage drivers.

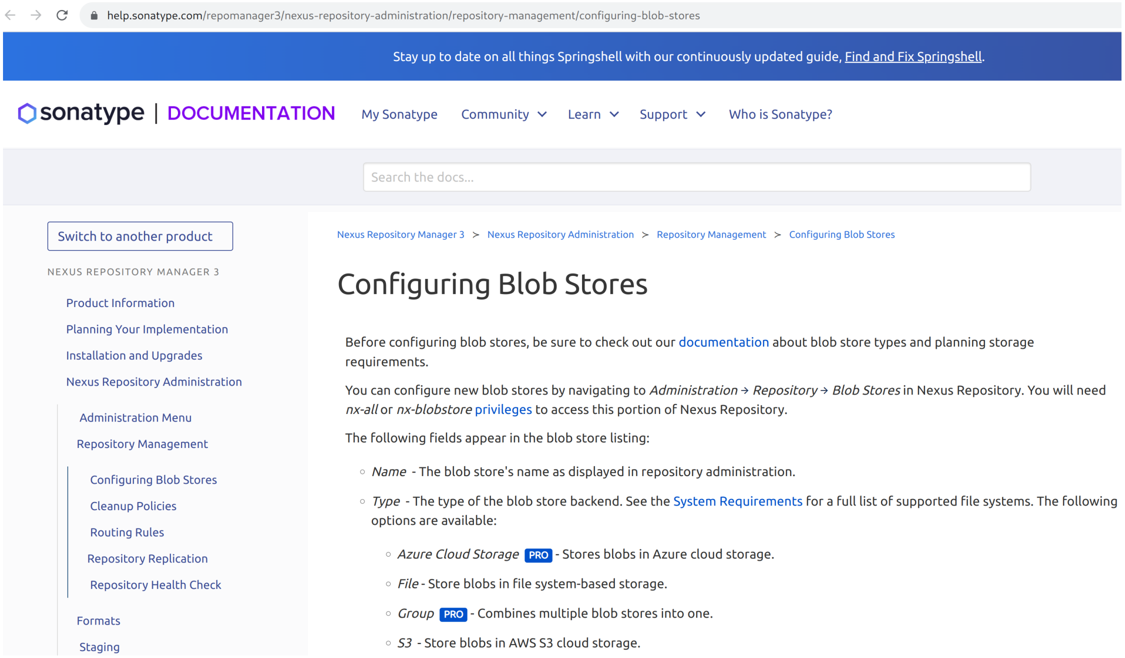
Provided drivers

This storage driver package comes bundled with several drivers:

- inmemory: A temporary storage driver using a local inmemory map. This exists solely for reference and testing.
- filesystem: A local storage driver configured to use a directory tree in the local filesystem.
- s3: A driver storing objects in an Amazon Simple Storage Service (S3) bucket.
- azure: A driver storing objects in Microsoft Azure Blob Storage.
- swift: A driver storing objects in Openstack Swift.
- oss: A driver storing objects in Aliyun OSS.
- gcs: A driver storing objects in a Google Cloud Storage bucket.

Storage driver API

The storage driver API is designed to model a filesystem-like key/value storage in a manner abstract enough to support a range of drivers from the local filesystem to Amazon S3 or other distributed object storage systems.


help.sonatype.com/repomanager3/nexus-repository-administration/repository-management/configuring-blob-stores

Stay up to date on all things Springshell with our continuously updated guide, Find and Fix Springshell.

Nexus Repository Manager 3 > Nexus Repository Administration > Repository Management > Configuring Blob Stores

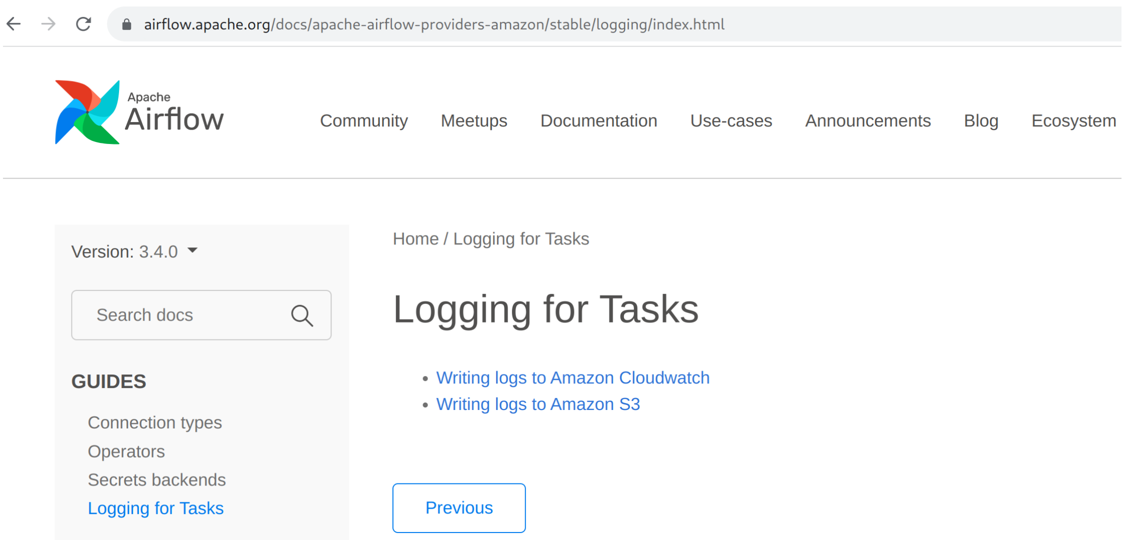
Configuring Blob Stores

Before configuring blob stores, be sure to check out our documentation about blob store types and planning storage requirements.

You can configure new blob stores by navigating to *Administration → Repository → Blob Stores* in Nexus Repository. You will need *nx-all* or *nx-blobstore* privileges to access this portion of Nexus Repository.

The following fields appear in the blob store listing:

- *Name* - The blob store's name as displayed in repository administration.
- *Type* - The type of the blob store backend. See the System Requirements for a full list of supported file systems. The following options are available:
  - *Azure Cloud Storage* PRO - Stores blobs in Azure cloud storage.
  - *File* - Store blobs in file system-based storage.
  - *Group* PRO - Combines multiple blob stores into one.
  - *S3* - Store blobs in AWS S3 cloud storage.


airflow.apache.org/docs/apache-airflow-providers-amazon/stable/logging/index.html

Apache Airflow

Community Meetups Documentation Use-cases Announcements Blog Ecosystem

Version: 3.4.0

Search docs

GUIDES

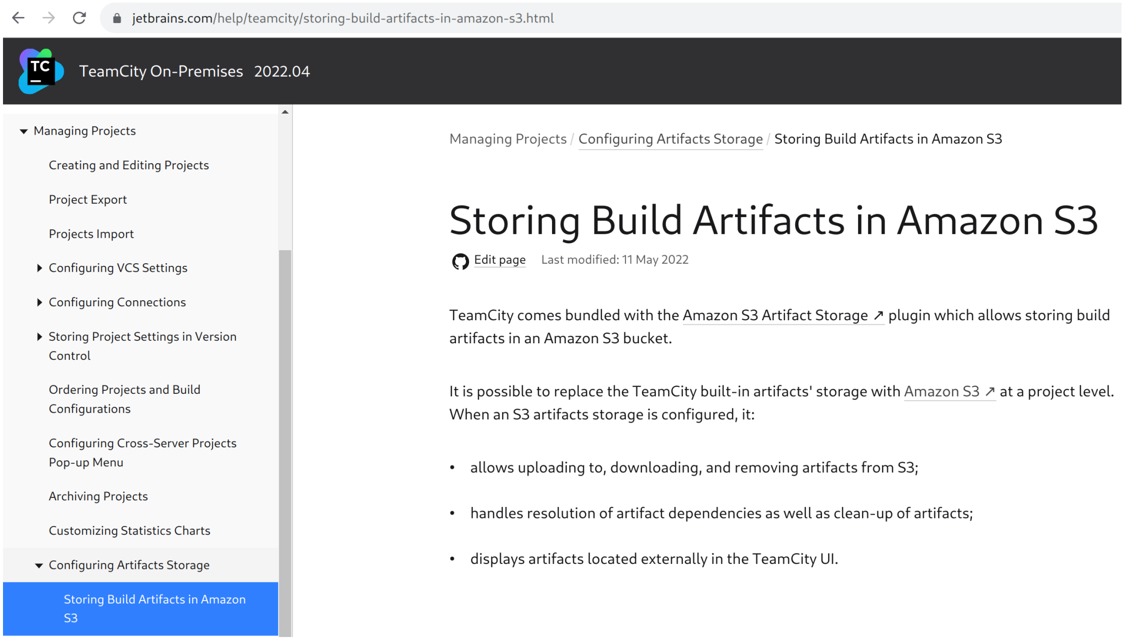
Connection types
Operators
Secrets backends
Logging for Tasks

Home / Logging for Tasks

Logging for Tasks

- Writing logs to Amazon Cloudwatch
- Writing logs to Amazon S3

Previous


jetbrains.com/help/teamcity/storing-build-artifacts-in-amazon-s3.html
TeamCity On-Premises 2022.04
Managing Projects
Creating and Editing Projects
Project Export
Projects Import
Configuring VCS Settings
Configuring Connections
Storing Project Settings in Version Control
Ordering Projects and Build Configurations
Configuring Cross-Server Projects Pop-up Menu
Archiving Projects
Customizing Statistics Charts
Configuring Artifacts Storage
Storing Build Artifacts in Amazon S3
Managing Projects / Configuring Artifacts Storage / Storing Build Artifacts in Amazon S3

Storing Build Artifacts in Amazon S3

Edit page Last modified: 11 May 2022

TeamCity comes bundled with the Amazon S3 Artifact Storage ↗ plugin which allows storing build artifacts in an Amazon S3 bucket.

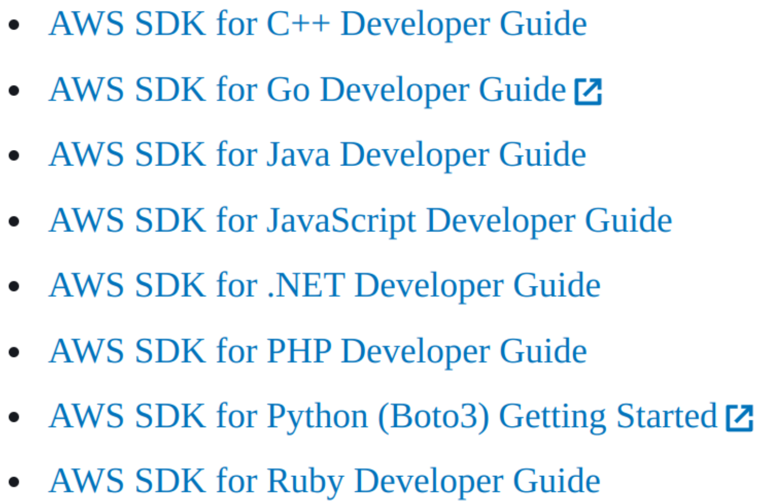
It is possible to replace the TeamCity built-in artifacts' storage with Amazon S3 ↗ at a project level. When an S3 artifacts storage is configured, it:

- allows uploading to, downloading, and removing artifacts from S3;
- handles resolution of artifact dependencies as well as clean-up of artifacts;
- displays artifacts located externally in the TeamCity UI.

Pluggable storage backends

- OpenStack Object Storage API
- Google Cloud Storage
- Azure Blob service REST API
- WebDAV
- **Amazon S3**
    - **Customizable service endpoint**

Rich ecosystem over SDKs



- AWS SDK for C++ Developer Guide
- AWS SDK for Go Developer Guide
- AWS SDK for Java Developer Guide
- AWS SDK for JavaScript Developer Guide
- AWS SDK for .NET Developer Guide
- AWS SDK for PHP Developer Guide
- AWS SDK for Python (Boto3) Getting Started
- AWS SDK for Ruby Developer Guide

Simple Storage Service[2]

- AWS Service and **API reference**
- Buckets
  - Unique
  - `https://`**`<bucket>`**`.s3.ok.ru/`
  - `https://s3.ok.ru/`**`<bucket>`**`/`
- Objects
  - **Immutable** and optionally **versioned**
  - `https://<bucket>.s3.ok.ru/`**`<object>`**
  - `eTag` — MD5
  - `Last-Modified` (seconds precision)

[2] https://docs.aws.amazon.com/AmazonS3/latest/dev/Introduction.html

Object limits

- **Unlimited** objects per bucket
- **Key** up to 1024 chars
- **Meta** up to 2 KB
- Up to 10 **tags**: **name** + **value**
- Up to **10K parts** per object
- Minimal **part size** 5 MB
- Maximum multipart **object size** 5 TB

Core S3 HTTP API

- ListBuckets
- {Get/Head}Object + If-* headers
- {Put/Delete}Object + DeleteObject**s**
- **Copy**Object
- {Get/Put/Delete}Object**Tagging**
- ListObjects + prefix/delimiter/marker
- ListObject**Versions** + prefix/delimiter/marker

Bucket Lifecycle Policies[3]

- Set of bucket lifecycle rules
- Filter objects by key **prefix** and/or **tags**
- `Expiration` after date or `n` days
- `NonCurrentVersionExpiration` after `n` days
- `ExpiredObjectDeleteMarker`
- `AbortIncompleteMultipartUpload`

[3] https://docs.aws.amazon.com/AmazonS3/latest/dev/intro-lifecycle-rules.html

Requirements

- **Scalability**
  - Capacity from PBs and beyond
  - Throughput from Gbps to Tbps
  - Locality (DC aware)
- **Fault-tolerance**
  - Lose any number of servers per DC
  - Disaster tolerant (offline DC)
  - No dedicated backups
- **Maintainability**
  - Automatic healing
  - Automated upscale/downscale

General components of S3

- **Metadata** storage (GBs/TBs)
  - File system (implicit)
  - In-memory
  - SQL/NoSQL/custom storage
- **Content** storage (TBs/PBs)
  - File system
  - Combined with metadata storage
  - Separate distributed blob storage

On-premises solutions

- **Ceph S3** (C++, Python)
    - USS Enterprise
    - Complicated maintainability/scalability
- **MinIO** (Go)
    - Filesystem + federation
    - No huge installations described
- **SeaweedFS** (Go)
    - Single developer
    - No huge installations described
- Zenko Cloudserver (JS), Riak Cloud Storage (Erlang), Joyent Triton Object Storage (JS), LeoFS (Erlang), . . .

On-premises solutions

- **Ceph S3** (C++, Python)
  - USS Enterprise
  - Complicated maintainability/scalability
- **MinIO** (Go)
  - Filesystem + federation
  - No huge installations described
- **SeaweedFS** (Go)
  - Single developer
  - No huge installations described
- Zenko Cloudserver (JS), Riak Cloud Storage (Erlang), Joyent Triton Object Storage (JS), LeoFS (Erlang), . . .

Data @ OK

- `one-blob/cold-storage` for **binary**
- NewSQL for **OLTP/meta**
- `one-cloud` as **runtime**

Key-value `one-blob/cold-storage`[4]

Hot `one-blob-storage` (OBS):

- 3x replication, tolerates loss of **2 disks**
- $>$ **100 PB**

`one-cold-storage`:

- Collects and **imports 40 GB segments** from OBS
- **2.1x** replication, tolerates loss of **5 disks**
- $\approx$ **EB** and growing

[4]Alexander Khristoforov. Petabytes of video and photo storage in Odnoklassniki: make it cheaper, simpler and more reliable @ Joker 2017 (RU)

NewSQL[5]

- $\approx$ **100** distributed clusters under heavy load
- Based on **Apache Cassandra**
- Dedicated **transaction coordinators**
- **Partitioned** transactions
- Cassandra **fat client** pattern
- **Speculative** execution
- p99 $<$ **3 ms**

[5] Oleg Anastasev. NewSQL = NoSQL + ACID (RU)

C* modeling recap

```
CREATE TABLE example (
    p text,
    t text,
    i int,
    d blob,
    PRIMARY KEY ((p), t, i));
```

Partition key

```
CREATE TABLE example (
    p text,
    t text,
    i int,
    d blob,
    PRIMARY KEY ((p), t, i));
```

Clustering key

```sql
CREATE TABLE example (
    p text,
    t text,
    i int,
    d blob,
    PRIMARY KEY ((p), t, i));
```

Example

p	t	i	d
p2	t1	1	0xDA7A1
p2	t1	2	0xDA7A2
p2	t2	3	0xDA7A3
p1	t3	4	0xDA7A4
p1	t4	5	0xDA7A5

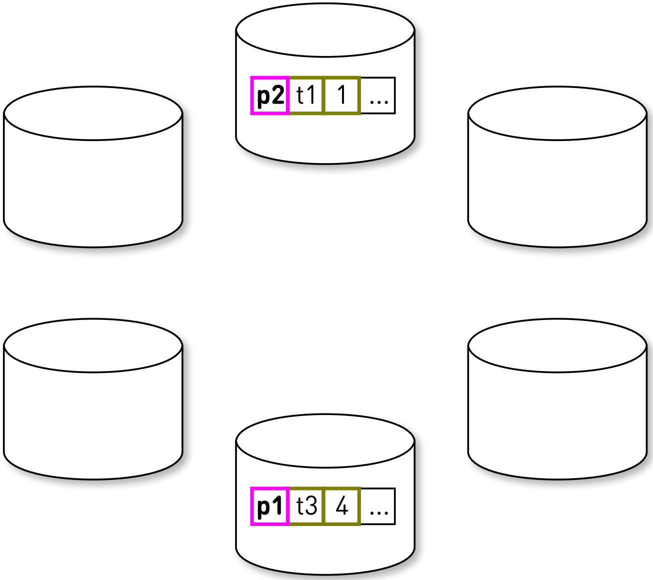
Partitions

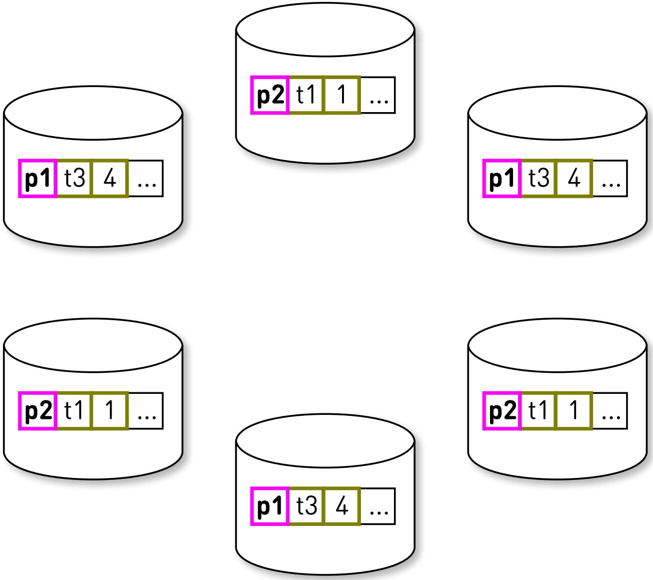
p	t	i	d
p2	t1	1	0xDA7A1
	t1	2	0xDA7A2
	t2	3	0xDA7A3
p1	t3	4	0xDA7A4
	t4	5	0xDA7A5

Physical data layout

p2	t1	1	0xDA7A1	t1	2	0xDA7A2	t2	3	0xDA7A3

p1	t3	4	0xDA7A4	t4	5	0xDA7A5


p2
t1
1
...
p1
t3
4
...


p2 t1 1 ...
p1 t3 4 ...
p1 t3 4 ...
p2 t1 1 ...
p2 t1 1 ...
p1 t3 4 ...

Data @ OK

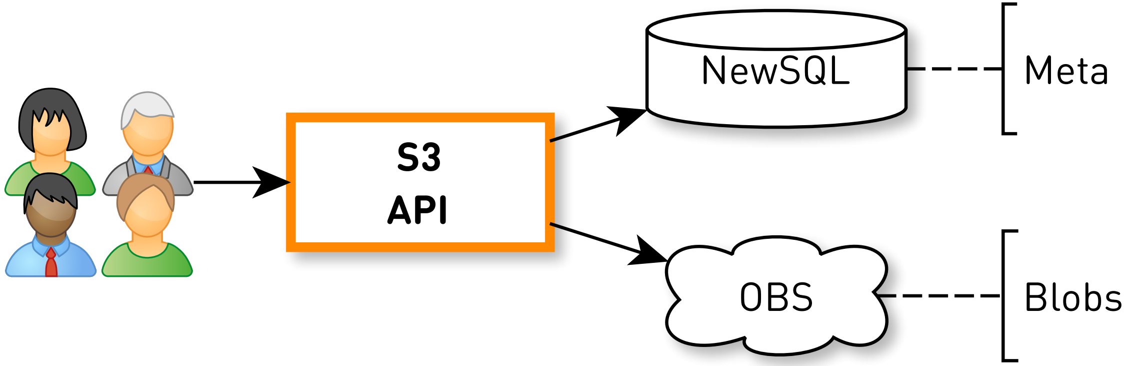
- `one-blob/cold-storage` for **binary**
- NewSQL for **OLTP/meta**
- `one-cloud` as **runtime**

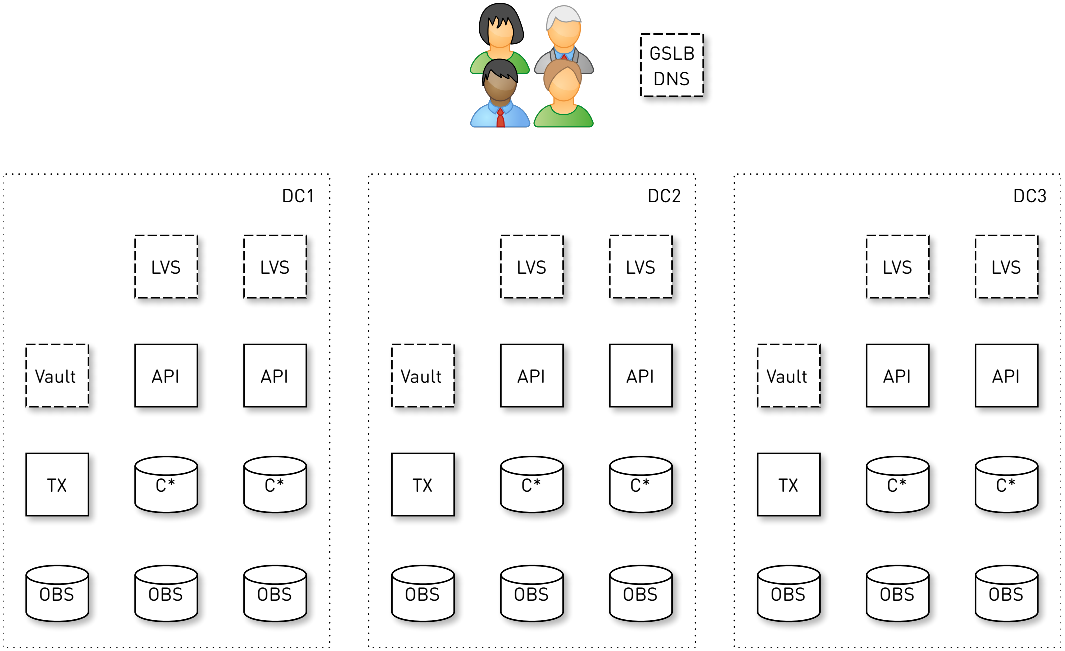
one-cloud[7]

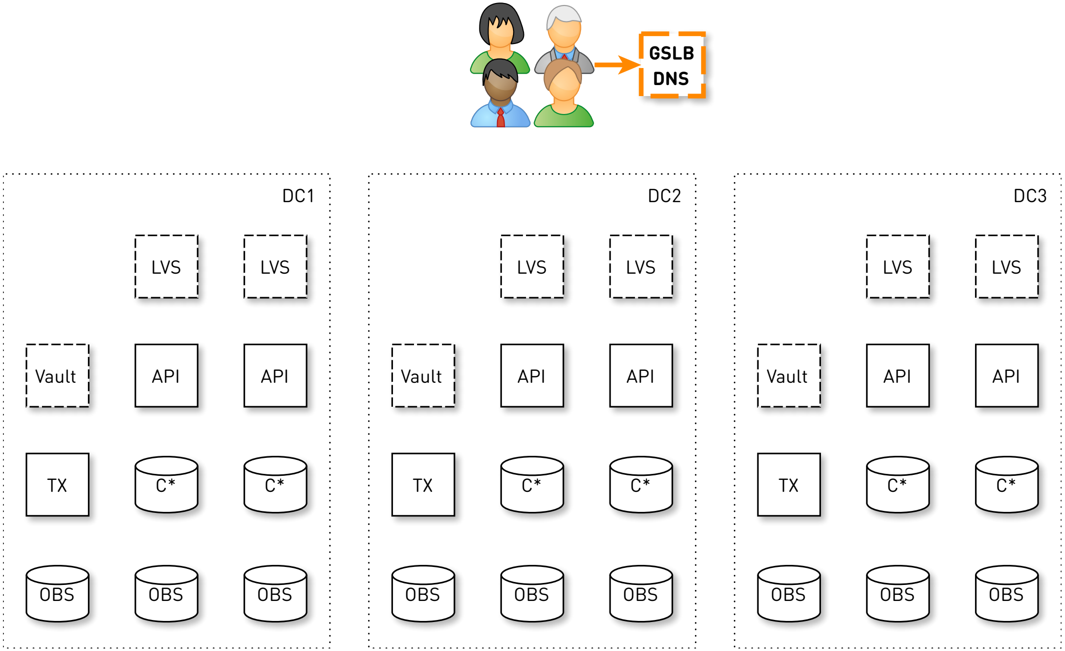
- Thread priorities (JIT, compaction, etc.)
- Storage **availability awareness**
  - Safe deployments
  - Rolling auto update/rebase
  - Auto migration
- **Automated cloud operations**[6]
  - Cluster upscale/downscale
  - Node replacement
  - Resource defragmentation

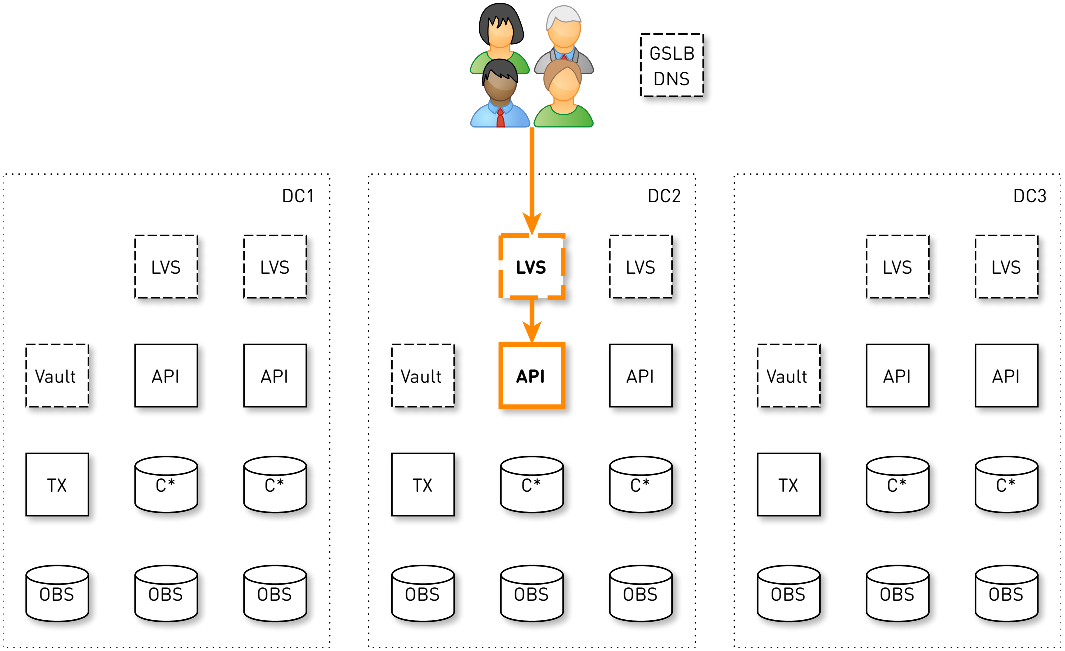
[6] Leonid Talalaev. “Rise of the Machines”is OK @ Highload++ 2019 (RU)
[7] Oleg Anastasev. Datacenter-level operating system by OK (RU)

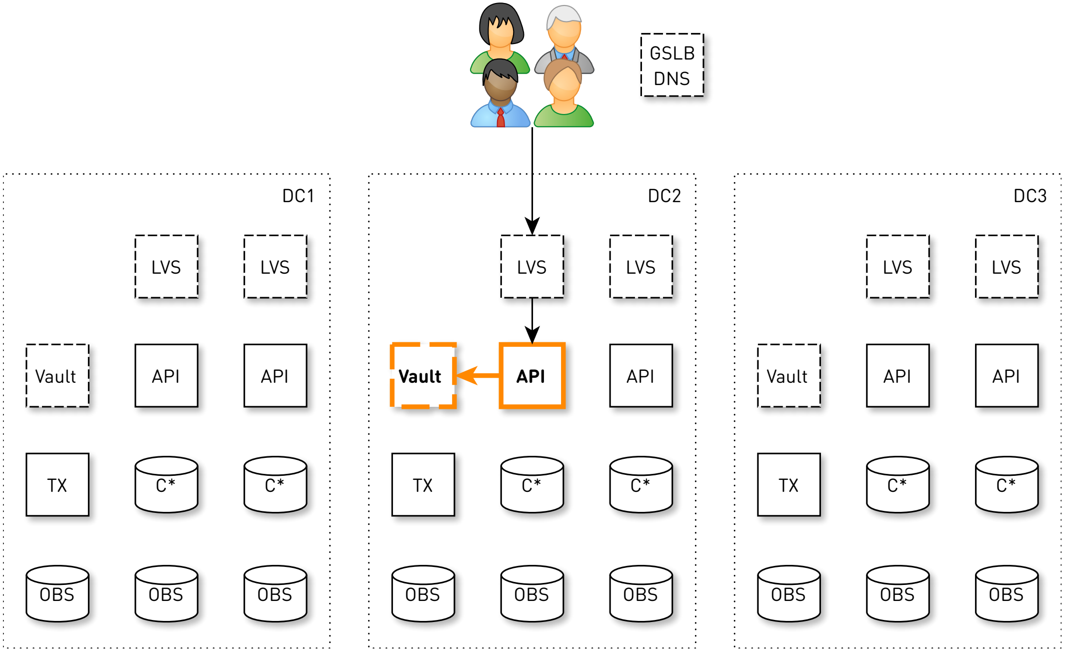
Architecture

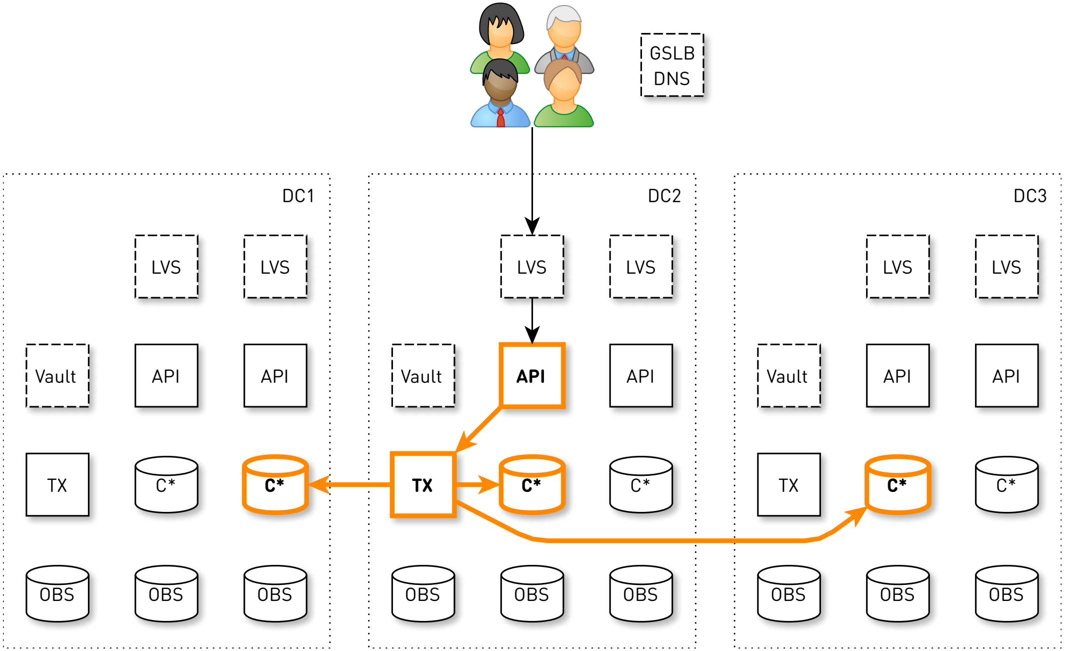


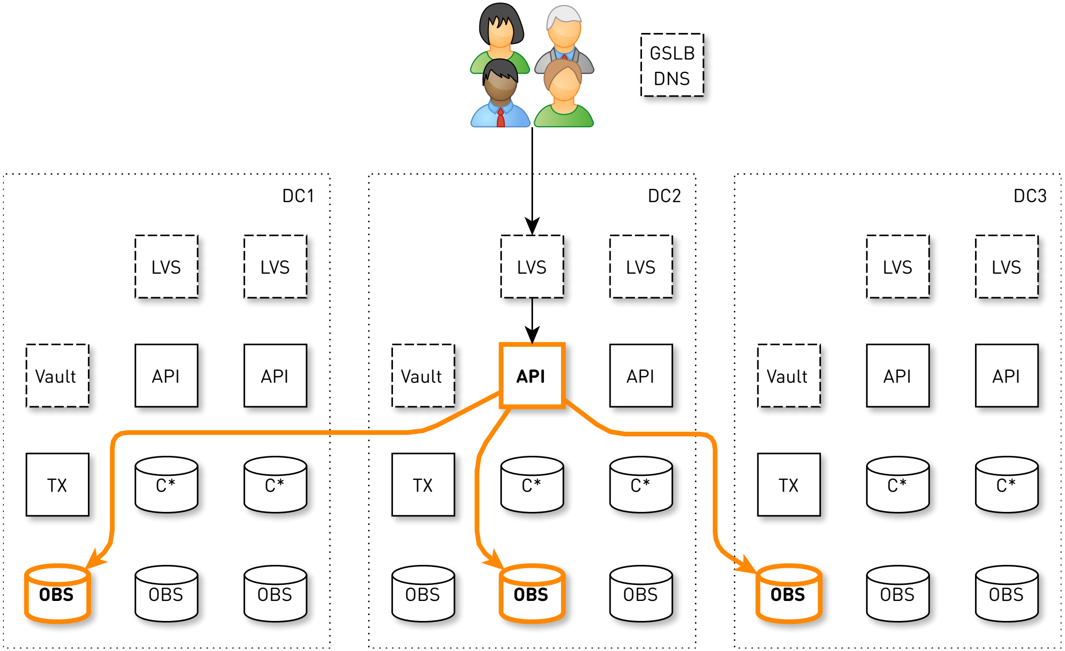

GSLB
DNS
DC1
LVS
LVS
Vault
API
API
TX
C*
C*
OBS
OBS
OBS
DC2
LVS
LVS
Vault
API
API
TX
C*
C*
OBS
OBS
OBS
DC3
LVS
LVS
Vault
API
API
TX
C*
C*
OBS
OBS
OBS

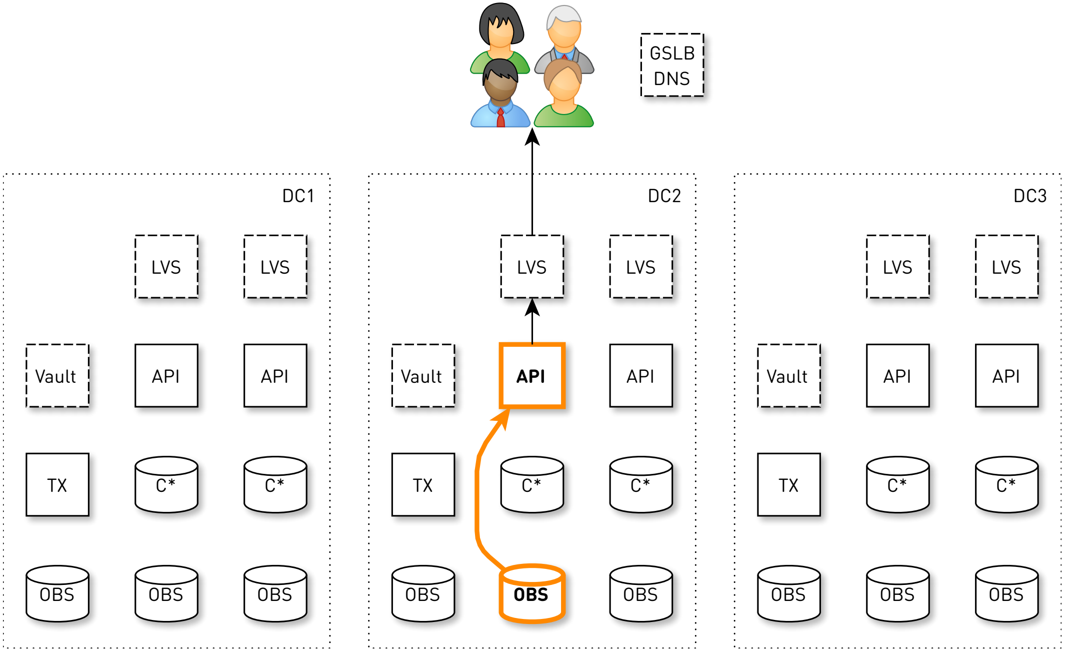

GSLB
DNS
DC1
DC2
DC3
LVS
LVS
Vault
API
API
TX
C*
C*
OBS
OBS
OBS

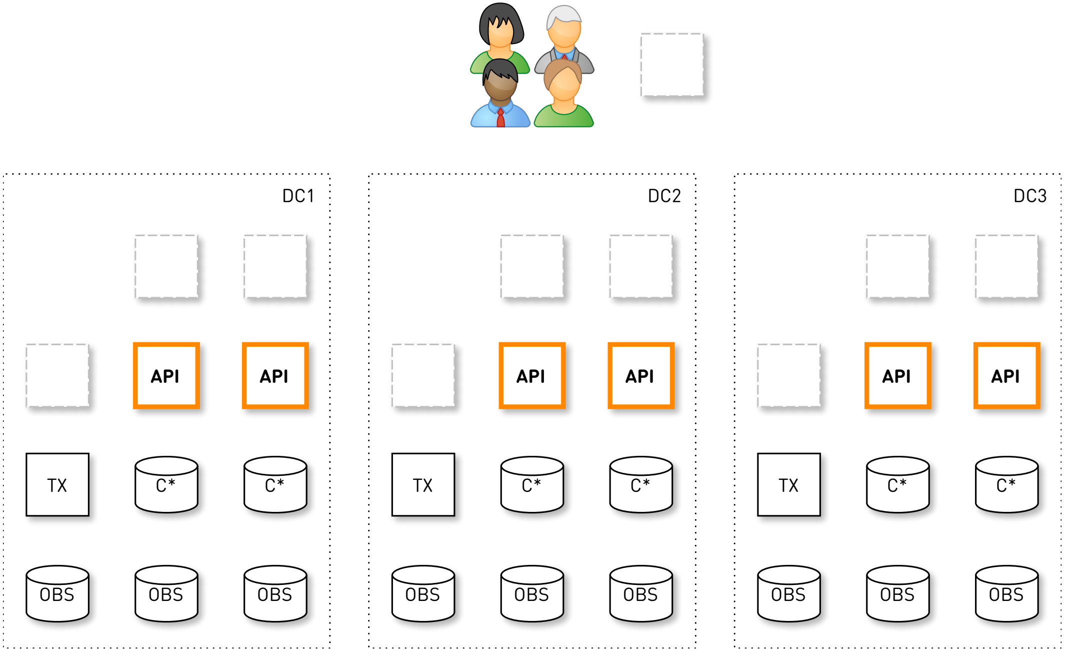

GSLB
DNS
DC1
LVS
LVS
Vault
API
API
TX
C*
C*
OBS
OBS
OBS
DC2
LVS
LVS
Vault
API
API
TX
C*
C*
OBS
OBS
OBS
DC3
LVS
LVS
Vault
API
API
TX
C*
C*
OBS
OBS
OBS


GSLB
DNS
DC1
LVS
LVS
Vault
API
API
TX
C*
C*
OBS
OBS
OBS
DC2
LVS
LVS
Vault
API
API
TX
C*
C*
OBS
OBS
OBS
DC3
LVS
LVS
Vault
API
API
TX
C*
C*
OBS
OBS
OBS


GSLB
DNS
DC1
DC2
DC3
LVS
LVS
LVS
LVS
LVS
LVS
Vault
API
API
Vault
API
API
Vault
API
API
TX
C*
C*
TX
C*
C*
TX
C*
C*
OBS
OBS
OBS
OBS
OBS
OBS
OBS
OBS
OBS


GSLB
DNS
DC1
DC2
DC3
LVS
LVS
LVS
LVS
LVS
LVS
Vault
API
API
Vault
API
API
Vault
API
API
TX
C*
C*
TX
C*
C*
TX
C*
C*
OBS
OBS
OBS
OBS
OBS
OBS
OBS
OBS
OBS


GSLB
DNS
DC1
DC2
DC3
LVS
Vault
API
TX
C*
OBS



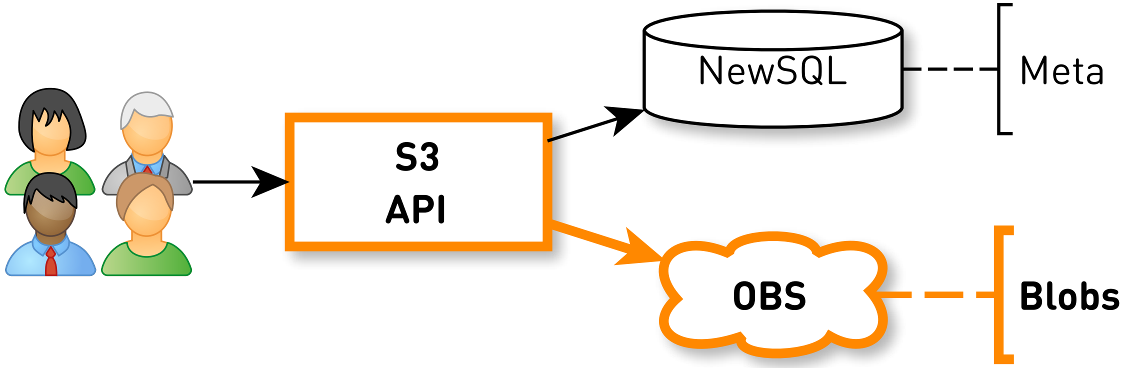
DC1
API
API
TX
C*
C*
OBS
OBS
OBS
DC2
API
API
TX
C*
C*
OBS
OBS
OBS
DC3
API
API
TX
C*
C*
OBS
OBS
OBS

Features

- **Deduplication**
  - Transparent server-side
  - Free object copy + space economy
  - Object/upload **part** = **block**
  - **Reference tracking** by object/upload parts
  - Blocks do not move
- **Versioning**[8]
  - Rollback/restore
  - No need for backups
  - `NoncurrentVersionExpiration` lifecycle rule

[8] https://docs.aws.amazon.com/AmazonS3/latest/userguide/versioning-workflows.html

Blocks



Block ID

Block size 8 bytes	SHA-256 32 bytes

Queries

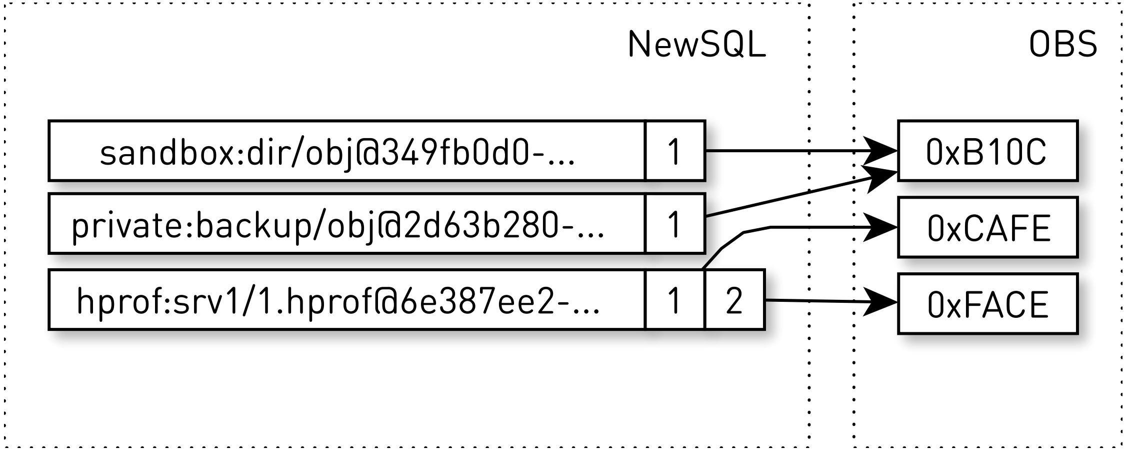
- **Reference** block
- **Unreference** block
- Check the block is **referenced**

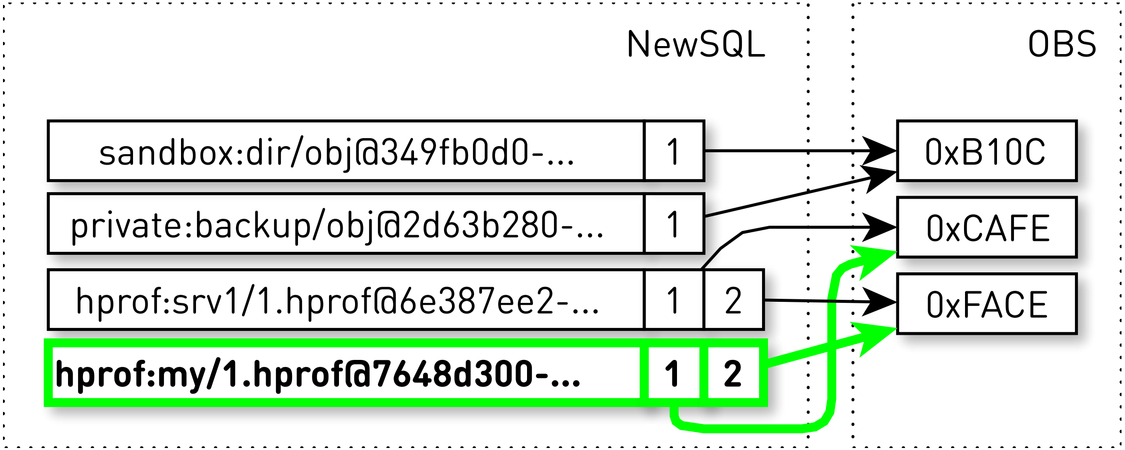
```sql
CREATE TABLE block_references (
    block blob,
    bucket text,
    name text, -- Object/upload key i.e. a/b/c
    version timeuuid, -- Object/upload version
    part int, -- Object/upload part
    PRIMARY KEY ((block), bucket, name, version, part));
```

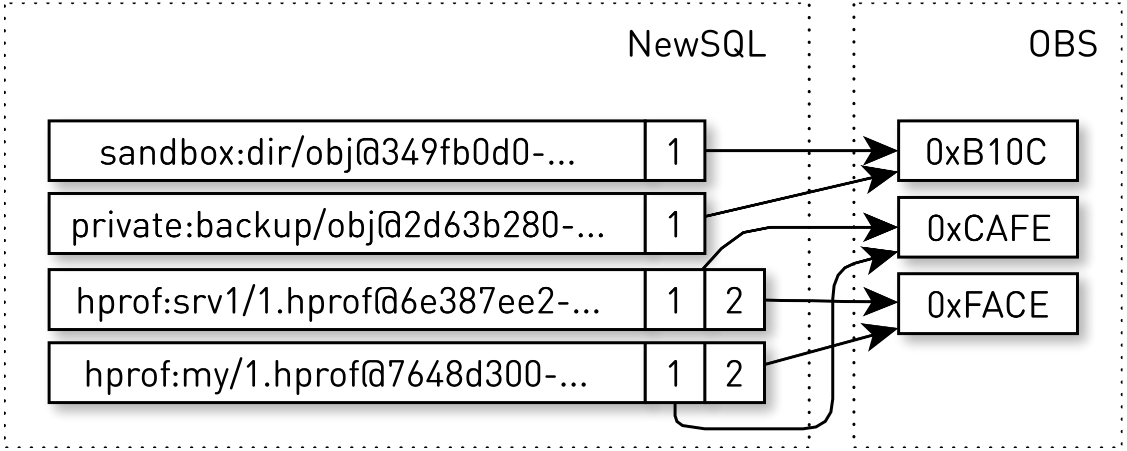
```
CREATE TABLE block_references (
    block blob,
    bucket text,
    name text, -- Object/upload key i.e. a/b/c
    version timeuuid, -- Object/upload version
    part int, -- Object/upload part
    PRIMARY KEY ((block), bucket, name, version, part));
```

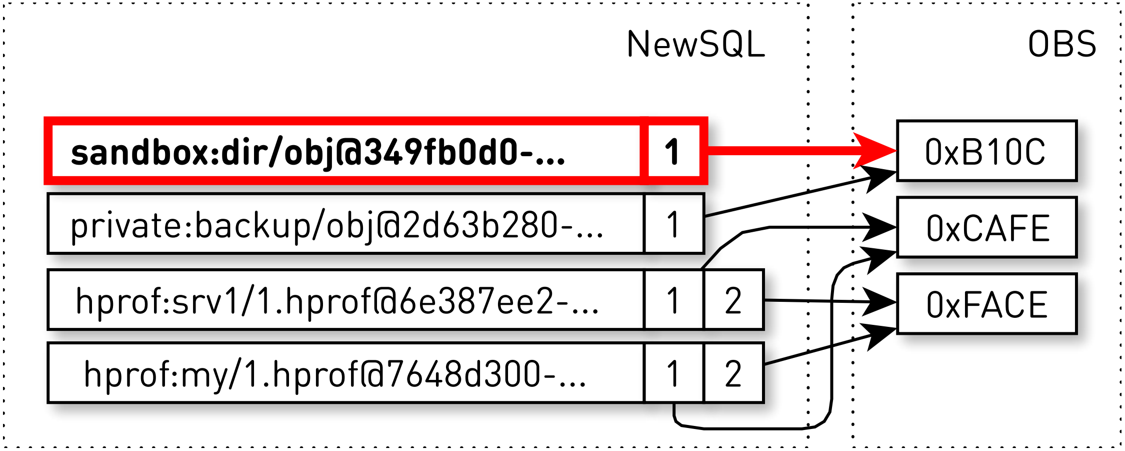
```
CREATE TABLE block_references (
    block blob,
    bucket text,
    name text, -- Object/upload key i.e. a/b/c
    version timeuuid, -- Object/upload version
    part int, -- Object/upload part
    PRIMARY KEY ((block), bucket, name, version, part));
```

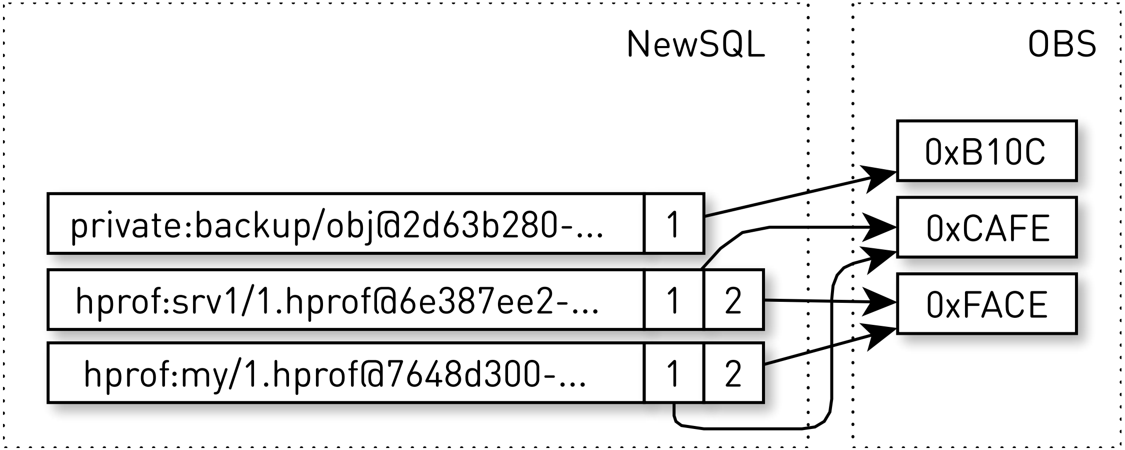
block	bucket	name	version	part
0xB10C	sandbox	dir/obj	349fb0d0-...	1
0xB10C	private	backup/obj	2d63b280-...	1
0xCAFE	hprof	srv1/1.hprof	6e387ee2-...	1
0xFACE	hprof	srv1/1.hprof	6e387ee2-...	2

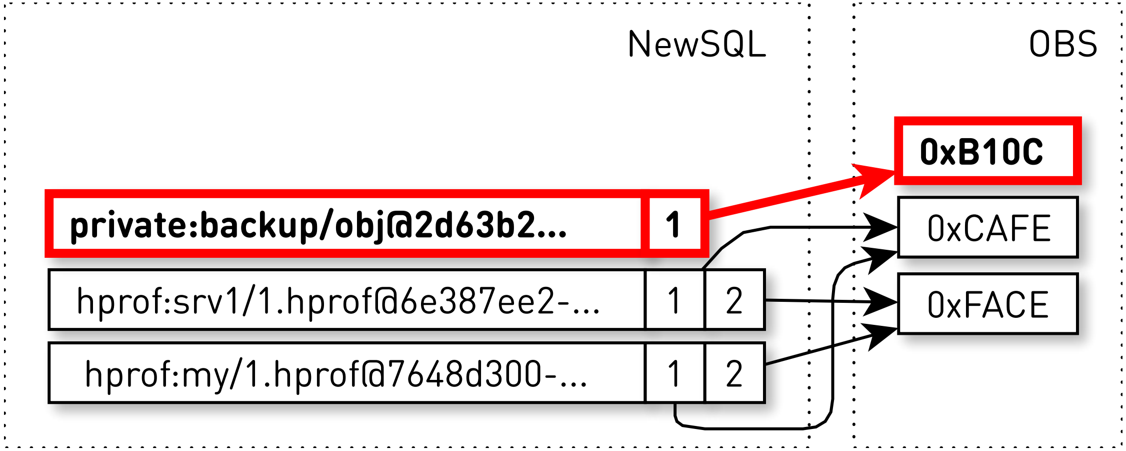

NewSQL
OBS
sandbox:dir/obj@349fb0d0-...
1
private:backup/obj@2d63b280-...
1
hprof:srv1/1.hprof@6e387ee2-...
1
2
0xB10C
0xCAFE
0xFACE

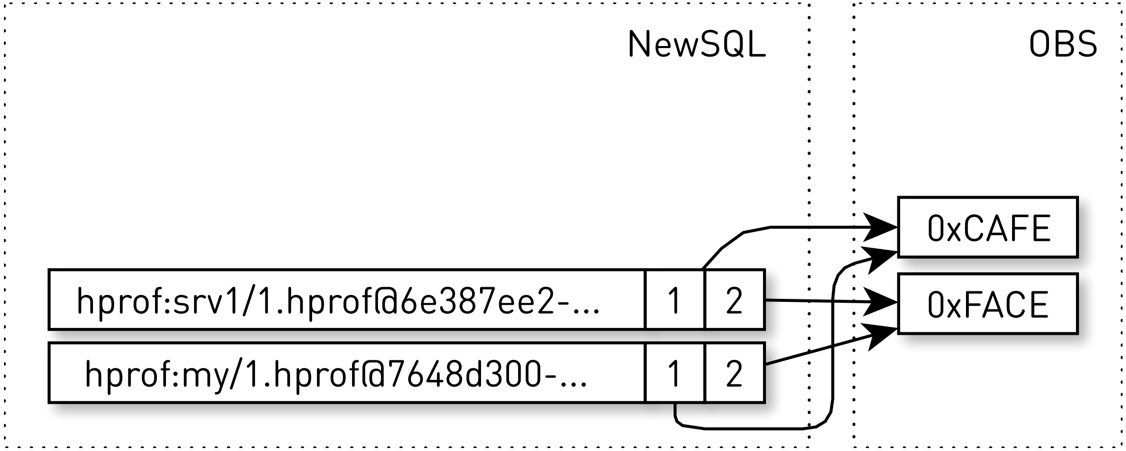

NewSQL
OBS
sandbox:dir/obj@349fb0d0-...
1
private:backup/obj@2d63b280-...
1
hprof:srv1/1.hprof@6e387ee2-...
1
2
hprof:my/1.hprof@7648d300-...
1
2
0xB10C
0xCAFE
0xFACE


NewSQL
OBS
sandbox:dir/obj@349fb0d0-...
1
private:backup/obj@2d63b280-...
1
hprof:srv1/1.hprof@6e387ee2-...
1
2
hprof:my/1.hprof@7648d300-...
1
2
0xB10C
0xCAFE
0xFACE

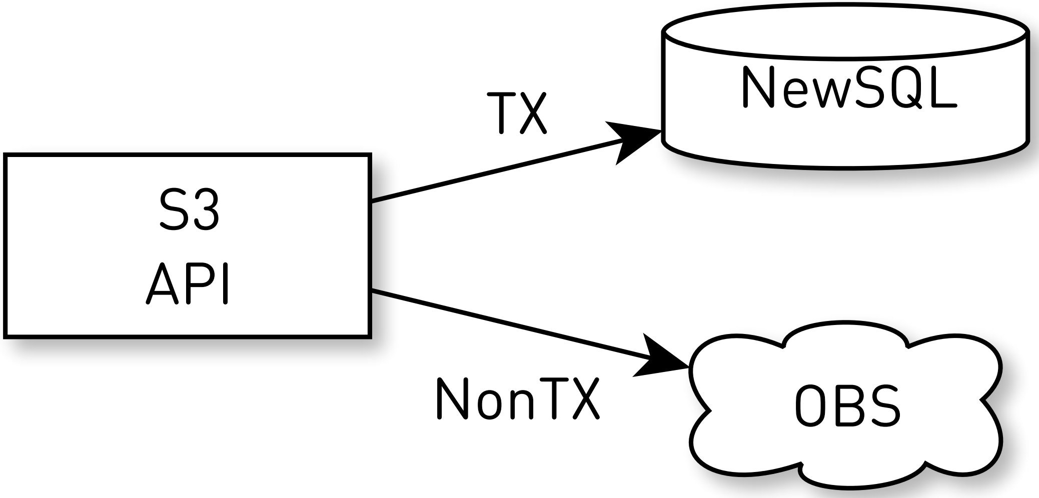

NewSQL
OBS
sandbox:dir/obj@349fb0d0-...
1
private:backup/obj@2d63b280-...
1
hprof:srv1/1.hprof@6e387ee2-...
1
2
hprof:my/1.hprof@7648d300-...
1
2
0xB10C
0xCAFE
0xFACE


NewSQL
OBS
private:backup/obj@2d63b280-...
1
hprof:srv1/1.hprof@6e387ee2-...
1
2
hprof:my/1.hprof@7648d300-...
1
2
0xB10C
0xCAFE
0xFACE

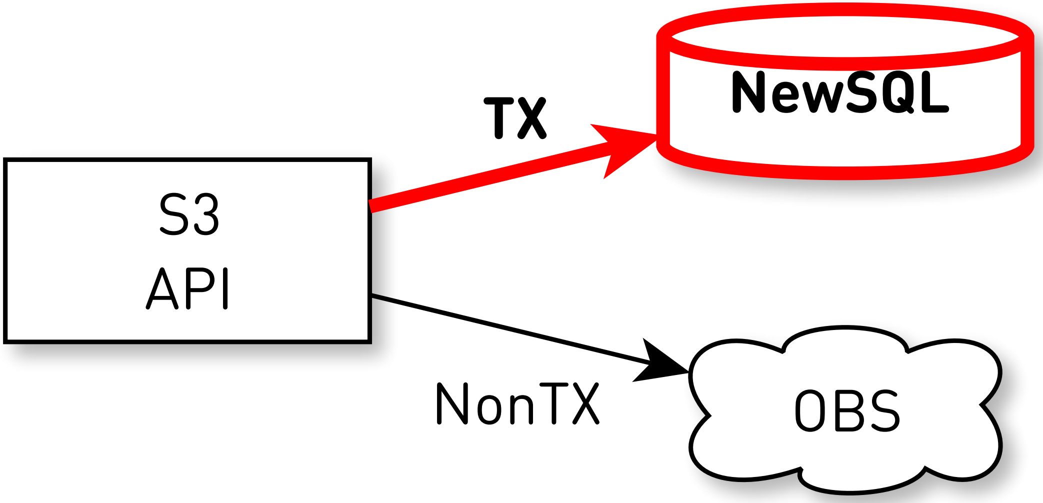

NewSQL
OBS
private:backup/obj@2d63b2...
1
hprof:srv1/1.hprof@6e387ee2-...
1
2
hprof:my/1.hprof@7648d300-...
1
2
0xB10C
0xCAFE
0xFACE


NewSQL
OBS
hprof:srv1/1.hprof@6e387ee2-...
1
2
hprof:my/1.hprof@7648d300-...
1
2
0xCAFE
0xFACE

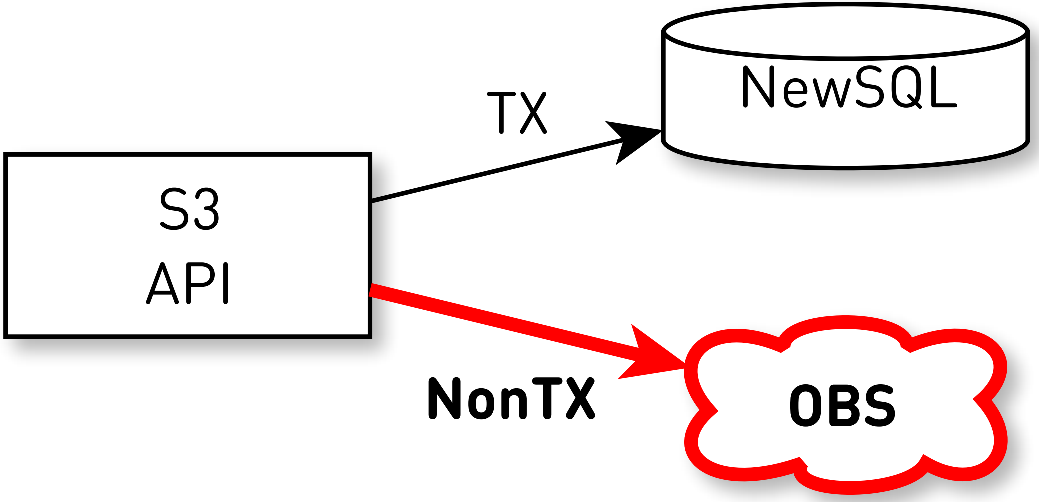
Problem

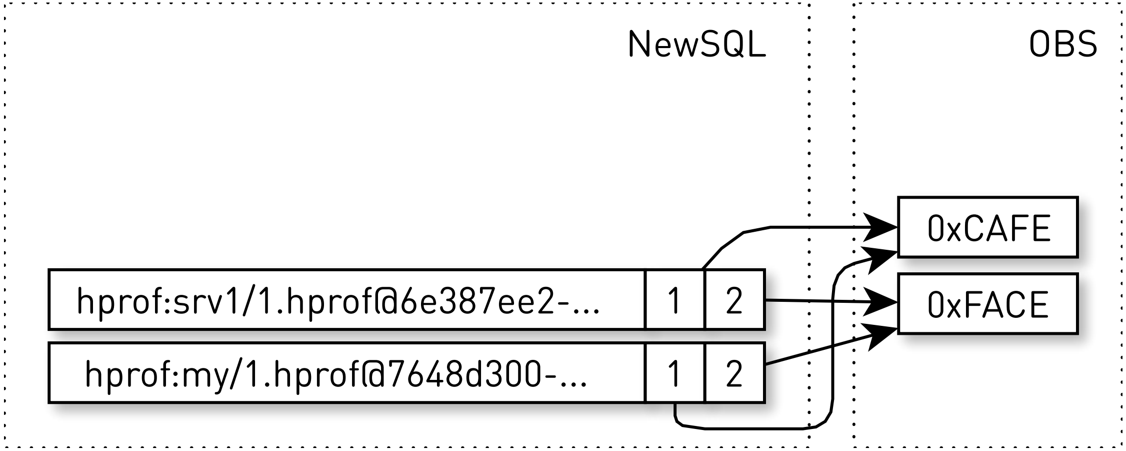


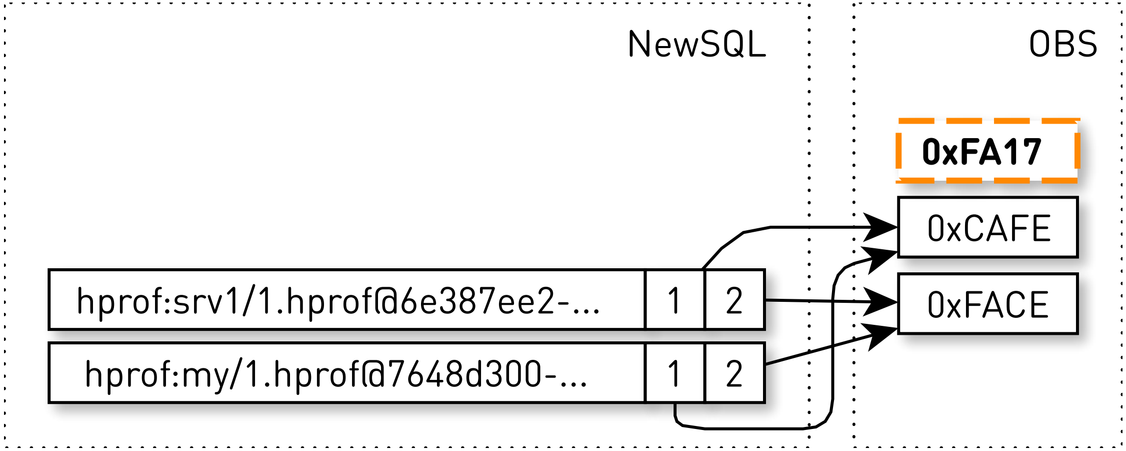
Problem

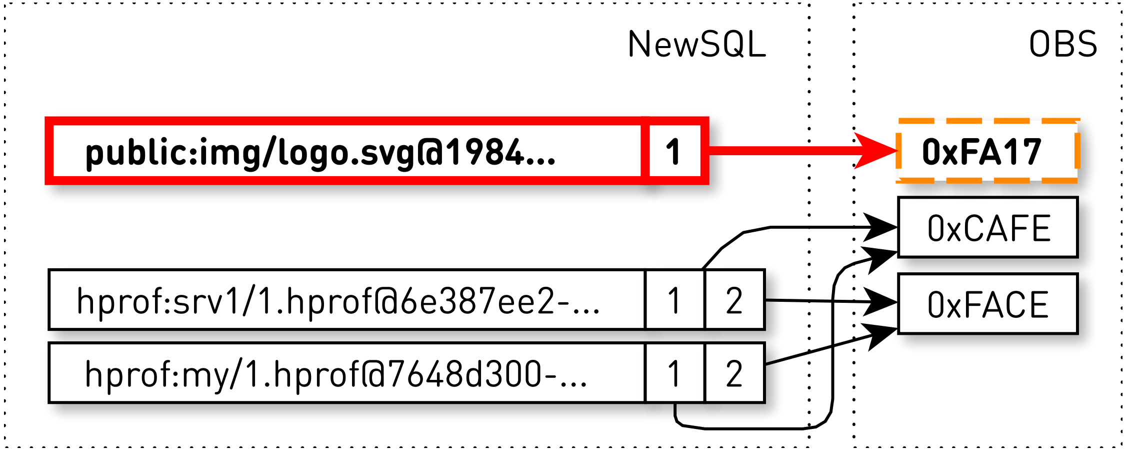


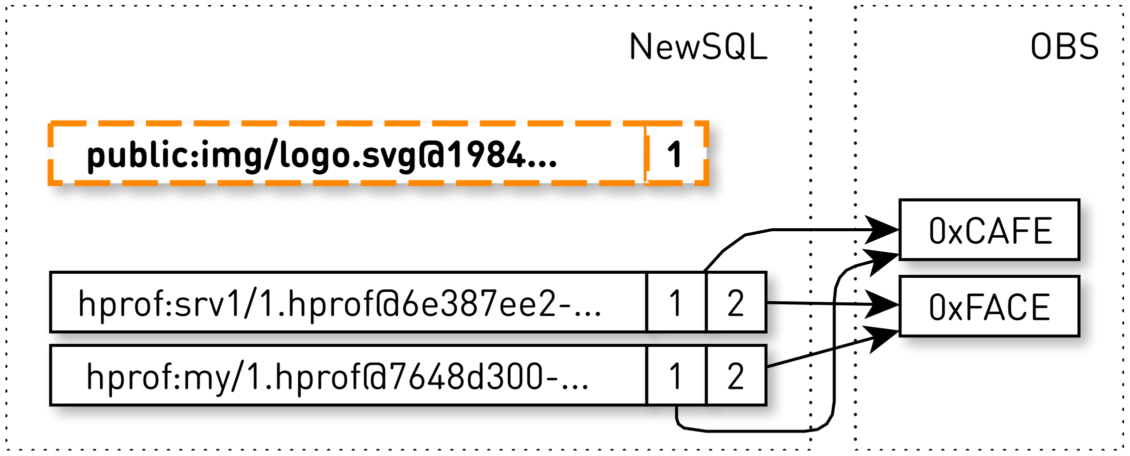
Problem

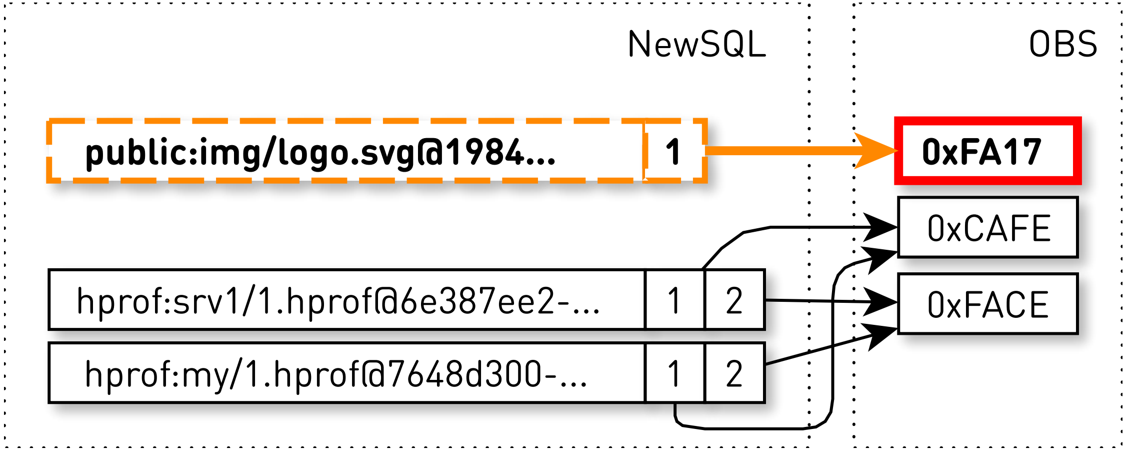



NewSQL
OBS
hprof:srv1/1.hprof@6e387ee2-...
1
2
hprof:my/1.hprof@7648d300-...
1
2
0xCAFE
0xFACE


NewSQL
OBS
0xFA17
0xCAFE
0xFACE
hprof:srv1/1.hprof@6e387ee2-...
1
2
hprof:my/1.hprof@7648d300-...
1
2


NewSQL
OBS
public:img/logo.svg@1984...
1
0xFA17
0xCAFE
hprof:srv1/1.hprof@6e387ee2-...
1
2
0xFACE
hprof:my/1.hprof@7648d300-...
1
2


NewSQL
OBS
public:img/logo.svg@1984...
1
hprof:srv1/1.hprof@6e387ee2-...
1
2
hprof:my/1.hprof@7648d300-...
1
2
0xCAFE
0xFACE

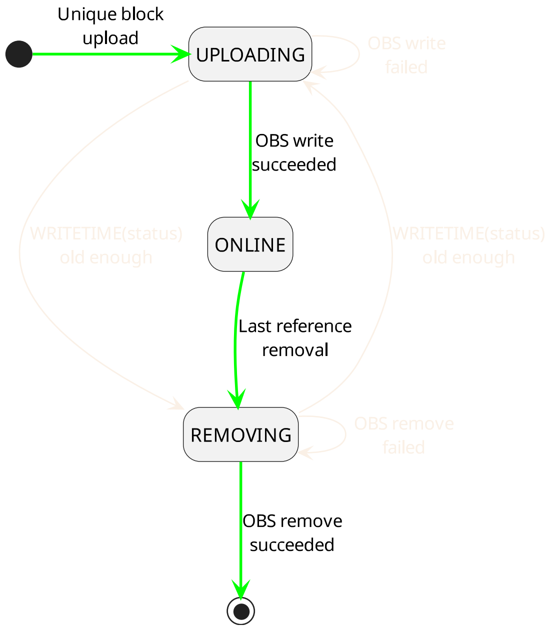

NewSQL
OBS
public:img/logo.svg@1984...
1
0xFA17
0xCAFE
hprof:srv1/1.hprof@6e387ee2-...
1
2
0xFACE
hprof:my/1.hprof@7648d300-...
1
2

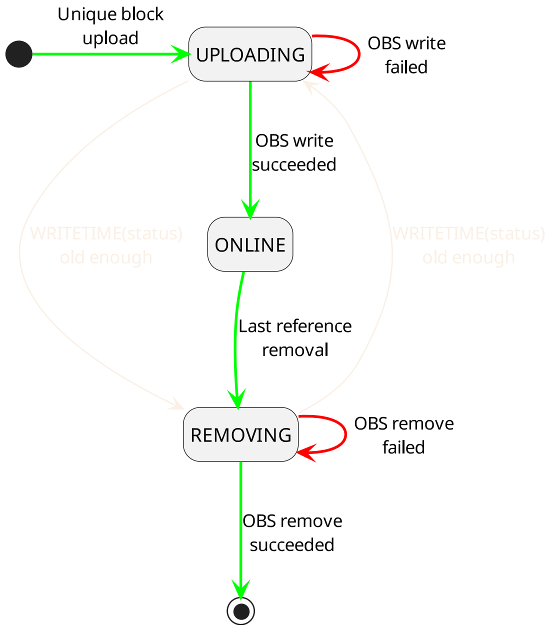
FSM to the rescue

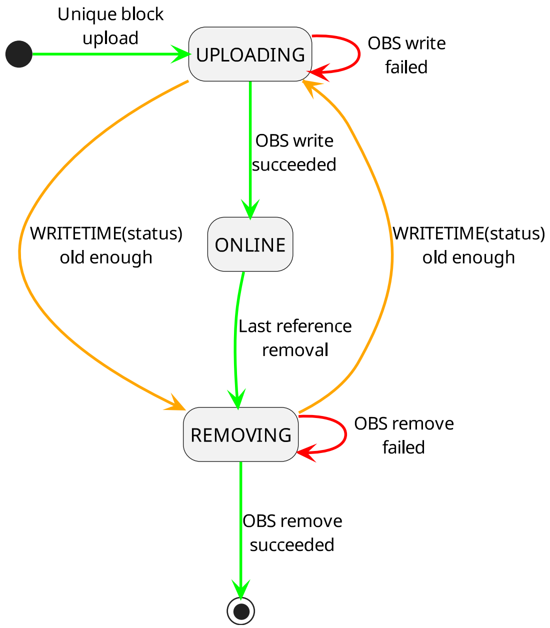
- Block **status tracking**
- Support for **retry/resume**
- Detect **race conditions**

Block Status

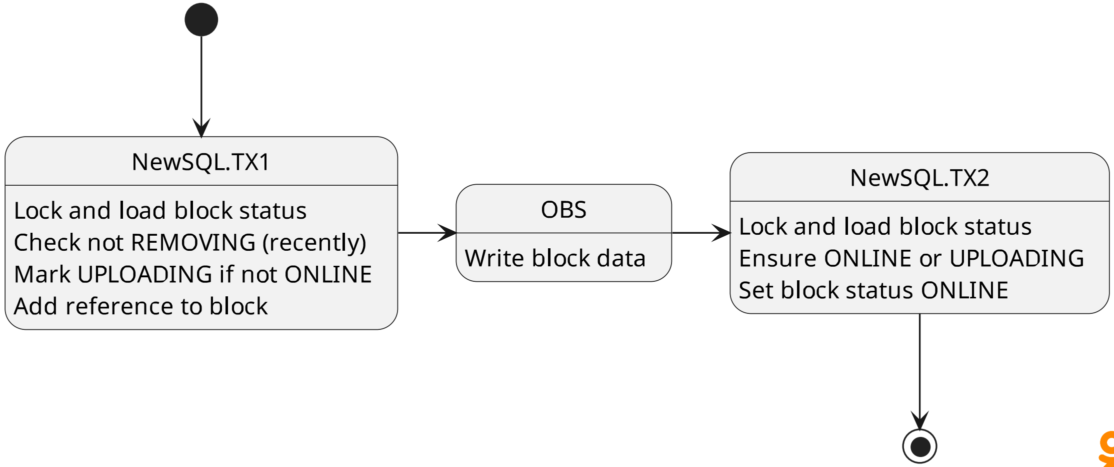
```sql
CREATE TABLE block_status (
    block blob, -- Block ID
    status int, -- UPLOADING/ONLINE/REMOVING
    PRIMARY KEY ((block));
```


Unique block upload
UPLOADING
OBS write failed
OBS write succeeded
ONLINE
WRITETIME(status) old enough
WRITETIME(status) old enough
Last reference removal
REMOVING
OBS remove failed
OBS remove succeeded

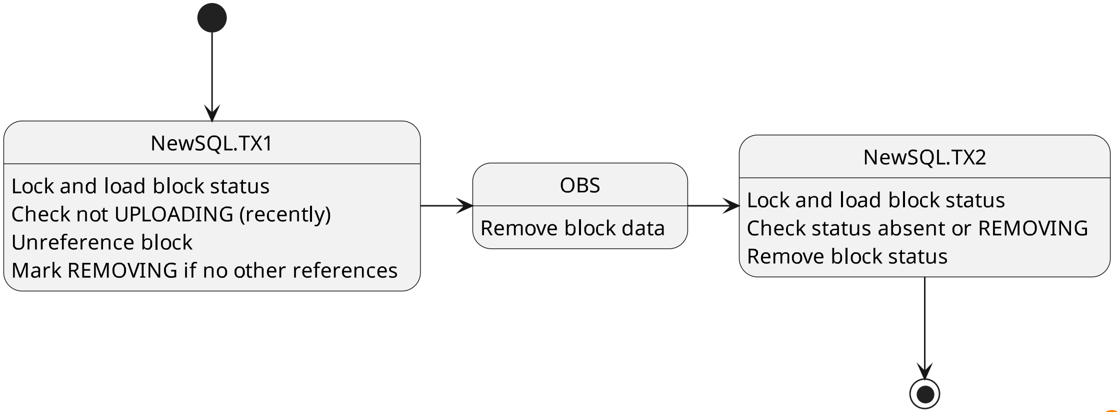

Unique block upload
UPLOADING
OBS write failed
OBS write succeeded
ONLINE
WRITETIME(status) old enough
WRITETIME(status) old enough
Last reference removal
REMOVING
OBS remove failed
OBS remove succeeded


Unique block upload
UPLOADING
OBS write failed
OBS write succeeded
ONLINE
WRITETIME(status) old enough
WRITETIME(status) old enough
Last reference removal
REMOVING
OBS remove failed
OBS remove succeeded

Unique block upload


NewSQL.TX1
Lock and load block status
Check not REMOVING (recently)
Mark UPLOADING if not ONLINE
Add reference to block
OBS
Write block data
NewSQL.TX2
Lock and load block status
Ensure ONLINE or UPLOADING
Set block status ONLINE

Last reference removal


NewSQL.TX1
Lock and load block status
Check not UPLOADING (recently)
Unreference block
Mark REMOVING if no other references
OBS
Remove block data
NewSQL.TX2
Lock and load block status
Check status absent or REMOVING
Remove block status

Once upon a time...

Timeouts on checking the block is still referenced:

```
Timeout executing [SELECT * FROM block_references WHERE block=? LIMIT 1
    ], bound: [...]:
org.apache.cassandra.exceptions.ReadTimeoutException:
Operation timed out - received only 1 responses of required 2.
at one.cassandra.client.impl.EmbeddedNewSQLClientImpl.execute(
    EmbeddedNewSQLClientImpl.java:496)
at one.cassandra.client.impl.EmbeddedCassandraClientImpl.paginate(
    EmbeddedCassandraClientImpl.java:288)
 ...
```

7 GB of references to **single block**

```
$ nodetool ... cfstats ... block_references
...
Compacted partition minimum bytes: 125
Compacted partition maximum bytes: 7'152'383'774

Compacted partition mean bytes: 721
```

Highly referenced block

0xB10C4	obj1	...	obj2	...	obj3	...	obj4	...

Tombstone accumulation

0xB10C4	obj1	. . .	obj2	∅	obj3	. . .	obj4	∅	. . .

Timeouts on scanning for live reference

0xB10C4	obj1	∅	obj2	∅	obj3	∅	obj4	∅	. . .

Wide C* partitions

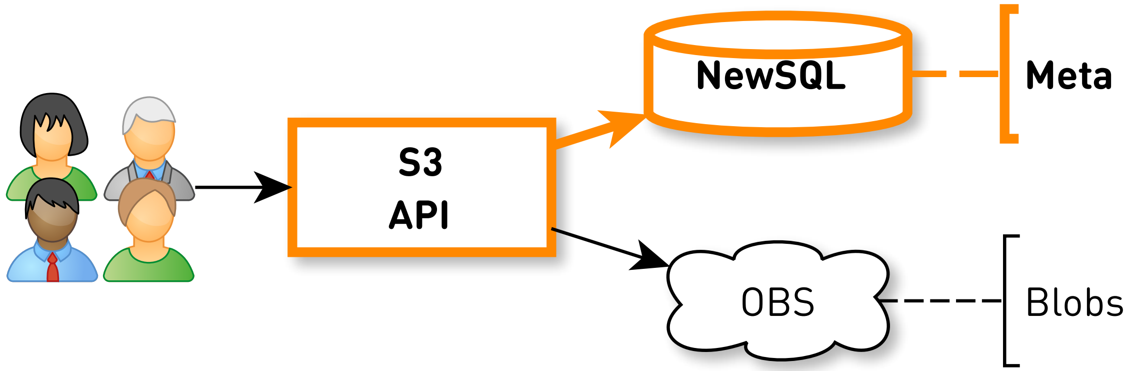
- Timeouts on `tombstone_failure_threshold`
    - **100K tombstones** by default
- Tombstones are purged after `gc_grace_period`
    - **10 days** by default
- Read repair operates **full partitions**
- `in_memory_compaction_limit`
    - **64 MB** by default

```java
/**
* Cache of small eternal blocks to solve numerous problems:
* <ul>
*     <li>Wide partitions due to highly referenced blocks</li>
*     <li>Huge number of tombstones in wide partitions</li>
*     <li>Serving tiny blocks from blob storage vs serving from
   memory</li>
* </ul>
*/
public final class Primordials {

    // The most popular empty block
    addPrimordial(cache, new byte[0]);

    // Hi, Apache Flink!
    addPrimordial(cache, "10".getBytes(StandardCharsets.US_ASCII));
```

Object metadata



Functional requirements

- **S3 documentation** for version/object/upload ops
    - Versioned and non-versioned
- **Transactional** concurrent modifications
- **Efficient** `List`-methods:
    - **Filtering** by `prefix`
    - **Grouping** by `delimiter`
    - Or **recursive**
    - **Sorted** by `name/version/upload`
    - **Paging** marker

Bucket = single partition + lock

bucket	**obj1**	. . .	**obj2**	. . .	**obj3**	. . .	**obj4**	. . .

- Might work
- Up to **100K objects per bucket**[9]

[9]See wide C* partition

Nonfunctional requirements

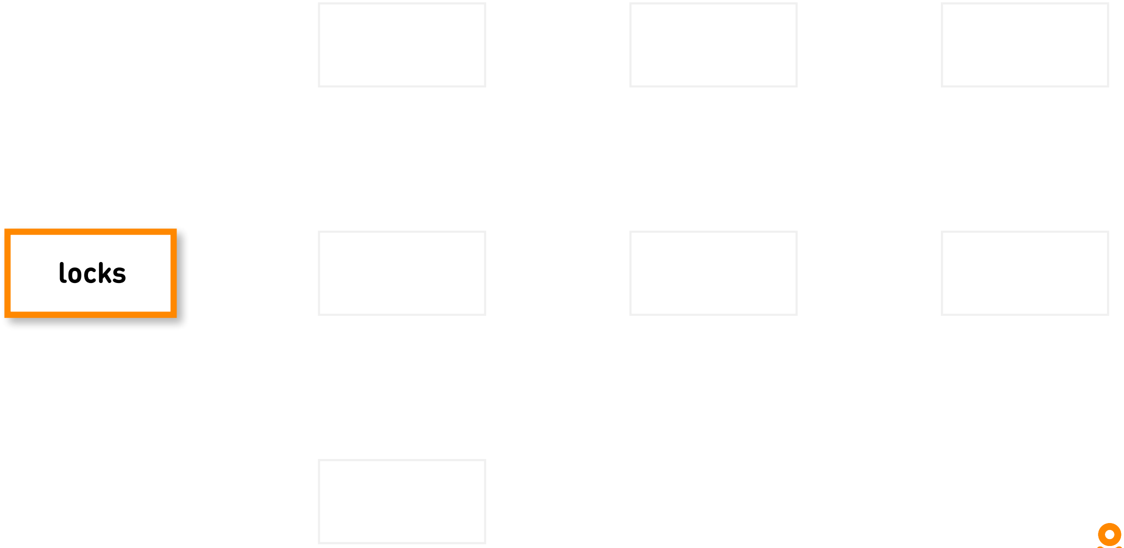
- **Data partitioning** for scalability
- **Transaction striping** for scalability
- **Efficient** upload publishing
  - No extra data copying
- **Fast** object/version serving
  - Handle extremely popular objects

Better way

- **Logical hierarchy** of “folders”by /
- **Partition** objects by `parent` folder
- **Cluster** inside `parent` by `name/version`
- Maintain **folder hierarchy** index
- Maintain **last object version** index
- **Atomicity** by `(parent, name)` key

Object partitioning

parent	name	version	meta
/	README.md	1	. . .
	TODO	2	. . .
data/	1.dat	4	. . .
	1.dat	3	. . .
	2.dat	5	. . .
data/log/	20220626.log	6	. . .


locks

Transient object/upload locks

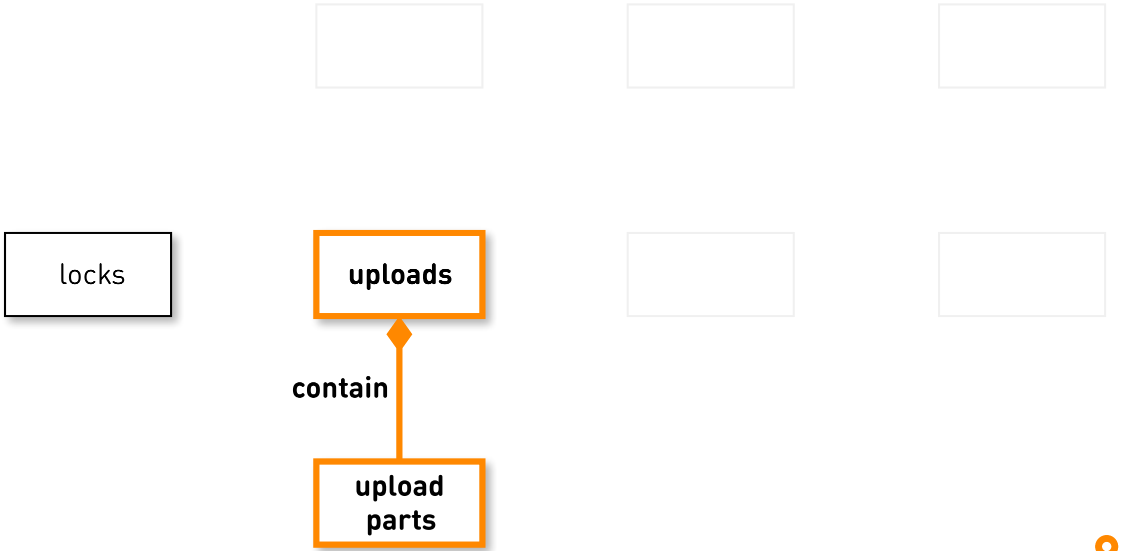
```sql
CREATE TABLE locks (
    parent text, -- Full parent name i.e. "a/b/"
    name text, -- Base child name i.e. "c"
    utoken int, -- xxhash32(parent + name)
    PRIMARY KEY ((parent), name, utoken),
    UPDATE TOKEN utoken);
```

Serializes (parent, name) ops

```
CREATE TABLE locks (
    parent text, -- Full parent name i.e. "a/b/"
    name text, -- Base child name i.e. "c"
    utoken int, -- xxhash32(parent + name)
    PRIMARY KEY ((parent), name, utoken),
    UPDATE TOKEN utoken);
```

Transaction striping

```
CREATE TABLE locks (
    parent text, -- Full parent name i.e. "a/b/"
    name text, -- Base child name i.e. "c"
    utoken int, -- xxhash32(parent + name)
    PRIMARY KEY ((parent), name, utoken),
    UPDATE TOKEN utoken);
```


locks
uploads
contain
upload parts

parent	name	upload	meta
a/	report.doc	123-...	type=doc
a/	report.doc	234-...	type=doc
b/	track.flac	345-...	type=flac

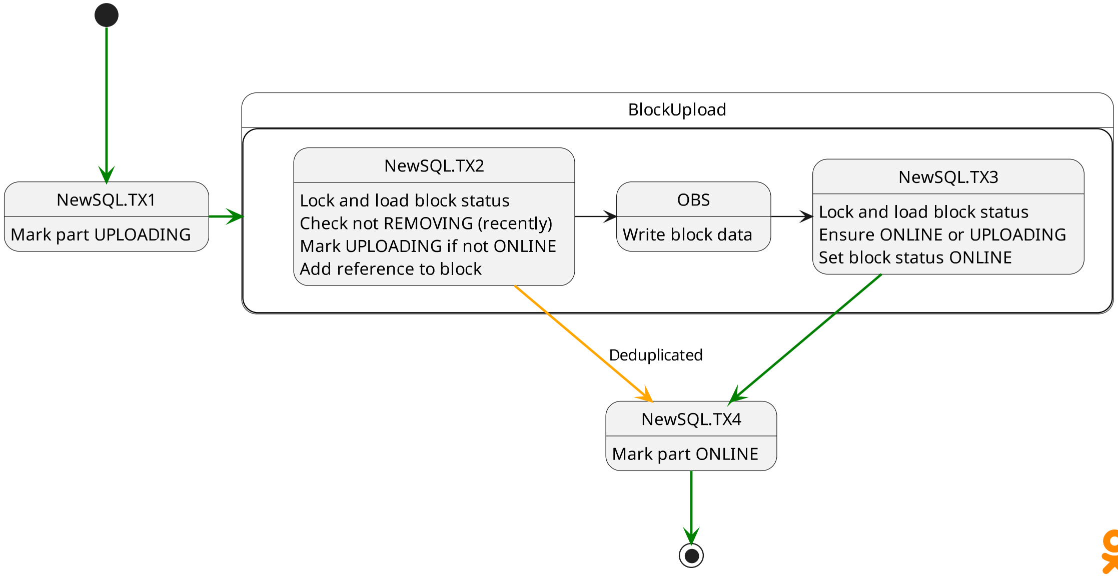
parent	name	upload	part	status	block
a/	report.doc	234-...	1	UPLOADING	0xBADDCAFE
a/	report.doc	234-...	2	ONLINE	0xCAFEFEED
b/	track.flac	345-...	1	ONLINE	0xA5A5A5A5
b/	track.flac	345-...	2	UPLOADING	0xABABABAB
b/	track.flac	345-...	3	ONLINE	0xBEBEBEBE

parent	name	upload	meta
a/	report.doc	123-...	type=doc
a/	report.doc	234-...	type=doc
b/	track.flac	345-...	type=flac

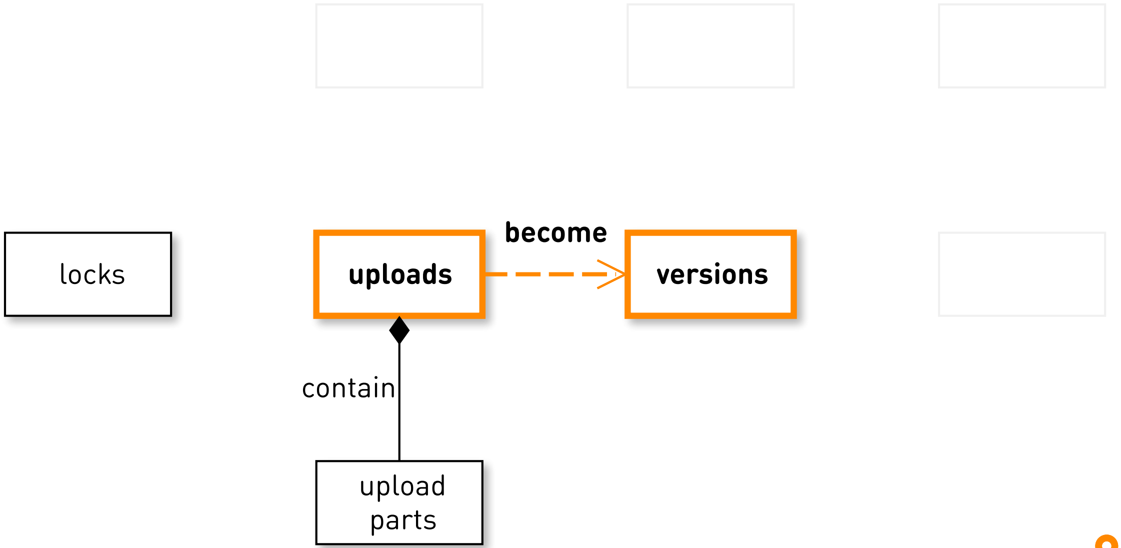
parent	name	upload	part	status	block
a/	report.doc	234-...	1	UPLOADING	0xBADDCAFE
a/	report.doc	234-...	2	ONLINE	0xCAFEFEED
b/	track.flac	345-...	1	ONLINE	0xA5A5A5A5
b/	track.flac	345-...	2	UPLOADING	0xABABABAB
b/	track.flac	345-...	3	ONLINE	0xBEBEBEBE

parent	name	upload	meta
a/	report.doc	123-...	type=doc
a/	report.doc	234-...	type=doc
b/	track.flac	345-...	type=flac

parent	name	upload	part	status	block
a/	report.doc	234-...	1	UPLOADING	0xBADDCAFE
a/	report.doc	234-...	2	ONLINE	0xCAFEFEED
b/	track.flac	345-...	1	ONLINE	0xA5A5A5A5
b/	track.flac	345-...	2	UPLOADING	0xABABABAB
b/	track.flac	345-...	3	ONLINE	0xBEBEBEBE



BlockUpload
NewSQL.TX1
Mark part UPLOADING
NewSQL.TX2
Lock and load block status
Check not REMOVING (recently)
Mark UPLOADING if not ONLINE
Add reference to block
OBS
Write block data
NewSQL.TX3
Lock and load block status
Ensure ONLINE or UPLOADING
Set block status ONLINE
Deduplicated
NewSQL.TX4
Mark part ONLINE


locks
uploads
become
versions
contain
upload parts

parent	**name**	**upload**	**meta**
a/	report.doc	123-...	type=doc
a/	report.doc	234-...	type=doc
b/	track.flac	345-...	type=flac

uploads

parent	**name**	**version**	**meta**	**blocks**

versions

parent	name	upload	meta
a/	report.doc	234-...	type=doc
b/	track.flac	345-...	type=flac

uploads

parent	name	version	meta	blocks
a/	report.doc	123-...	type=doc	[0xFA...]

versions

parent	name	upload	meta
b/	track.flac	345-...	type=flac

uploads

parent	name	version	meta	blocks
a/	report.doc	123-...	type=doc	[0xFA...]
a/	report.doc	234-...	type=doc	[0xBA..., 0xCA...]

versions

parent	name	upload	meta

uploads

parent	name	version	meta	blocks
a/	report.doc	123-...	type=doc	[0xFA...]
a/	report.doc	234-...	type=doc	[0xBA..., 0xCA...]
b/	track.flac	345-...	type=flac	[0xA5..., 0xAB..., 0xBE...]

versions

parent	name	upload	meta

uploads

parent	name	version	meta	blocks
a/	report.doc	123-...	type=doc	[0xFA...]
a/	report.doc	234-...	type=doc	[0xBA..., 0xCA...]
b/	track.flac	345-...	type=flac	[0xA5..., 0xAB..., 0xBE...]

versions

parent	name	upload	meta

uploads

parent	name	version	meta	blocks
a/	report.doc	123-...	type=doc	[0xFA...]
a/	report.doc	234-...	type=doc	[0xBA..., 0xCA...]
b/	track.flac	345-...	type=flac	[0xA5..., 0xAB..., 0xBE...]

versions

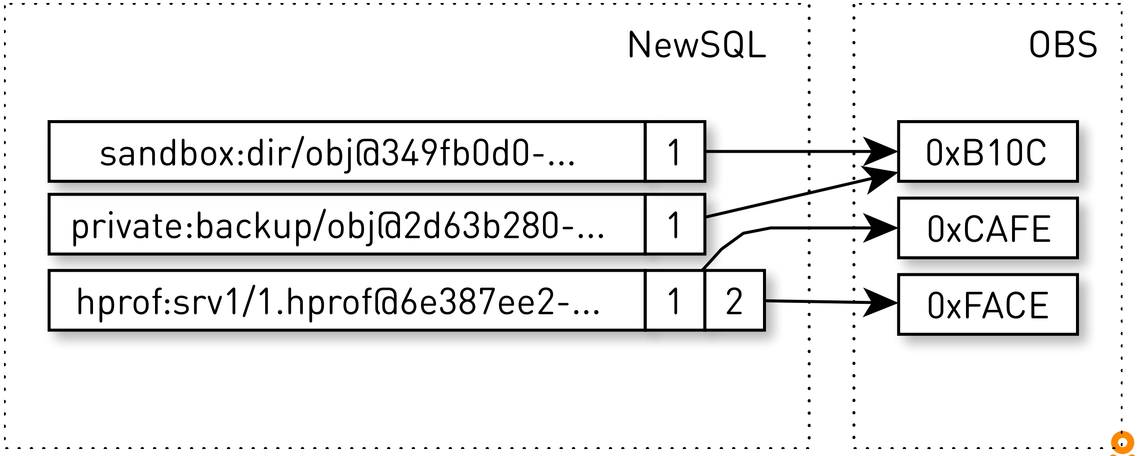
parent	name	upload	meta

uploads

parent	name	version	meta	blocks
a/	report.doc	123-...	type=doc	[0xFA...]
a/	report.doc	234-...	type=doc	[0xBA..., 0xCA...]
b/	track.flac	345-...	type=flac	[0xA5..., 0xAB..., 0xBE...]

versions

Reusing block references


NewSQL
OBS
sandbox:dir/obj@349fb0d0-...
1
private:backup/obj@2d63b280-...
1
hprof:srv1/1.hprof@6e387ee2-...
1
2
0xB10C
0xCAFE
0xFACE

parent	name	upload	meta
b/	track.flac	345-...	type=flac

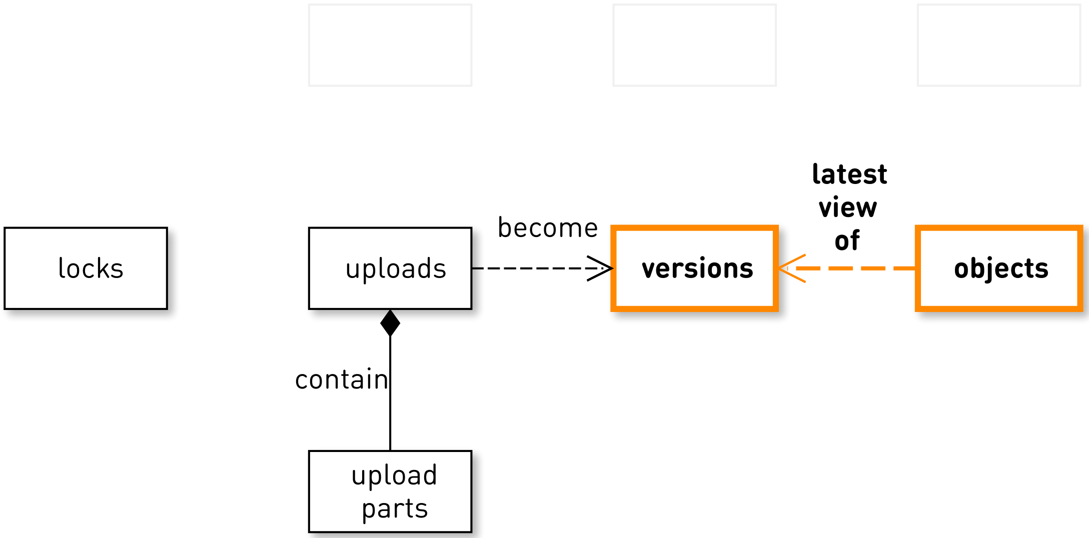
parent	name	upload	part	status	block
b/	track.flac	345-...	1	ONLINE	0xA5A5A5A5
b/	track.flac	345-...	2	ONLINE	0xABABABAB
b/	track.flac	345-...	3	ONLINE	0xBEBEBEBE

parent	name	version	meta	blocks

parent	name	upload	meta

parent	name	upload	part	status	block

parent	name	version	meta	blocks
b/	track.flac	345-...	type=flac	[0xA5..., 0xAB..., 0xBE...]

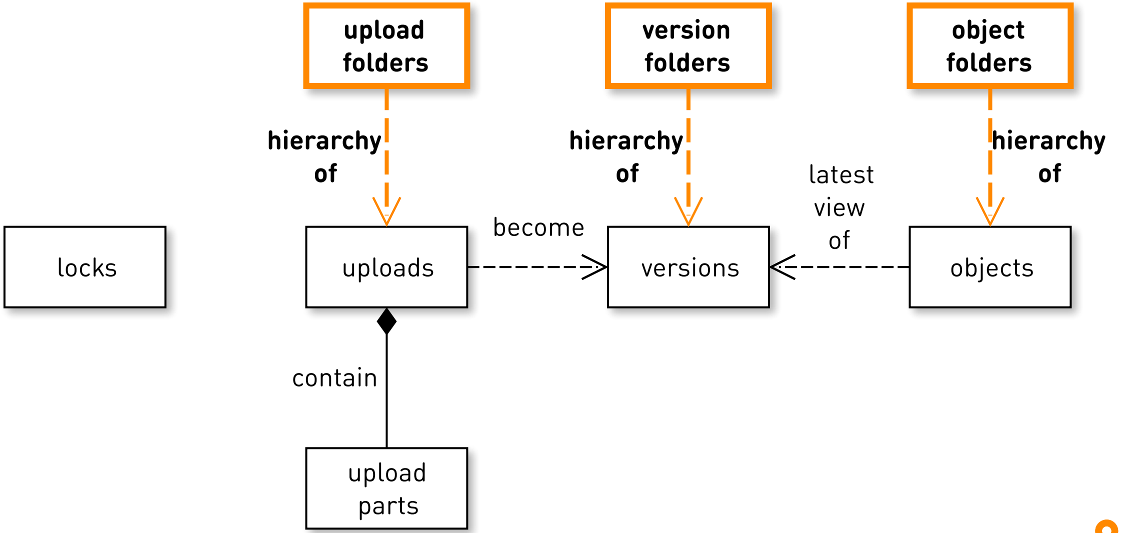

locks
uploads
become
versions
latest view of
objects
contain
upload parts

parent	name	version	eTag	blocks
docs/	0.doc	234-...	CDCDCDCD	[0xA1]
docs/	1.doc	456-...	0D15EA5E	[0xC3]
docs/	1.doc	345-...	D00D2BAD	[0xB2]
docs/	2.doc	678-...	NULL	NULL
docs/	2.doc	567-...	FEE1DEAD	[0xD4]

versions

parent	name	version	eTag	blocks
docs/	0.doc	234-...	CDCDCDCD	[0xA1]
docs/	1.doc	456-...	0D15EA5E	[0xC3]

objects


upload folders
version folders
object folders
hierarchy of
hierarchy of
hierarchy of
latest view of
locks
uploads
become
versions
objects
contain
upload parts

```sql
CREATE TABLE {object,version,upload}_folders (
    parent text, -- Full parent folder i.e. "a/b/"
    child text, -- Child folder name i.e. "c"
    ts timestamp, -- Last write timestamp
    PRIMARY KEY ((parent), child));
```

```sql
CREATE TABLE {object,version,upload}_folders (
    parent text, -- Full parent folder i.e. "a/b/"
    child text, -- Child folder name i.e. "c"
    ts timestamp, -- Last write timestamp
    PRIMARY KEY ((parent), child));
```

`{object,version,upload}_folders`

- `a/b/c/d` $\rightarrow$ `(/, a/)`, `(a/, b/)`, `(a/b/, c/)`
- **All** queries are **non-transactional**
- **Cleanup** with `WRITETIME(ts)` **slightly in the past**

List with delimiter a.k.a. ls

```
GET /?prefix=dir/
    &marker=dir/b
    &delimiter=/
    &max-keys=3
```

```sql
SELECT * FROM object_folders
    WHERE parent = 'dir/' AND child > 'b'
    LIMIT 4
```

```sql
SELECT * FROM objects
    WHERE parent = 'dir/' AND name > 'b'
    LIMIT 4
```

List with delimiter a.k.a. ls

```
GET /?prefix=dir/
        &marker=dir/b
        &delimiter=/
        &max-keys=3
```

```sql
SELECT * FROM object_folders
    WHERE parent = 'dir/' AND child > 'b'
    LIMIT 4
```

```sql
SELECT * FROM objects
    WHERE parent = 'dir/' AND name > 'b'
    LIMIT 4
```

List w/o delimiter a.k.a. `ls -R`

1. **Extract** `parent` from `prefix`
2. **Position** w.r.t. `marker` and suffix of `prefix`
3. In-order **tree traversal**
    1. Query sorted **subfolders**
    2. Query sorted **objects/versions/uploads**
    3. **Merge sorted**
    4. Dive **deeper**
4. Until `limit` objects/versions/uploads collected

List w/o delimiter a.k.a. `ls -R`

- Each **descending step** — **two** non-transactional queries (objects/versions/uploads + subfolders)
- Non-transactional **index cleanup** on empty folders
  - `WRITETIME` slightly in the past

Hard Case

Deep hierarchy of almost empty folders.

C* wide partitions

- Each folder content is stored in **one partition**
  - **Not including** child subfolders

Guideline

- < **100K** objects/versions per folder
- Introduce **folder striping**

Sonatype Nexus

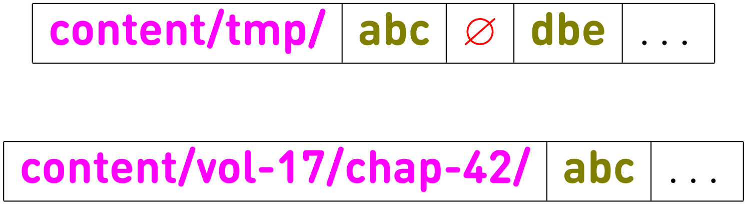
content/tmp/	**abc**	. . .

Move

content/tmp/	abc	∅

content/vol-17/chap-42/	abc	...

Put



Move

content/tmp/	abc	∅	dbe	∅

content/vol-17/chap-42/	abc	. . .
content/vol-23/chap-07/	dbe	. . .

Put

content/tmp/	0ab	. . .	abc	∅	dbe	∅

content/vol-17/chap-42/	abc	. . .
content/vol-23/chap-07/	dbe	. . .

Move

content/tmp/	0ab	∅	abc	∅	dbe	∅

content/vol-17/chap-42/	abc	. . .
content/vol-23/chap-07/	dbe	. . .
content/vol-31/chap-01/	0ab	. . .

Cemetery

content/tmp/	. . .	∅	. . .	∅	. . .	∅
	. . .	∅	. . .	∅	. . .	∅
	. . .	∅	. . .	∅	. . .	∅
	. . .	∅	. . .	∅	. . .	∅
	. . .	∅	. . .	∅	. . .	∅
	. . .	∅	. . .	∅	. . .	∅
	. . .	∅	. . .	∅	. . .	∅

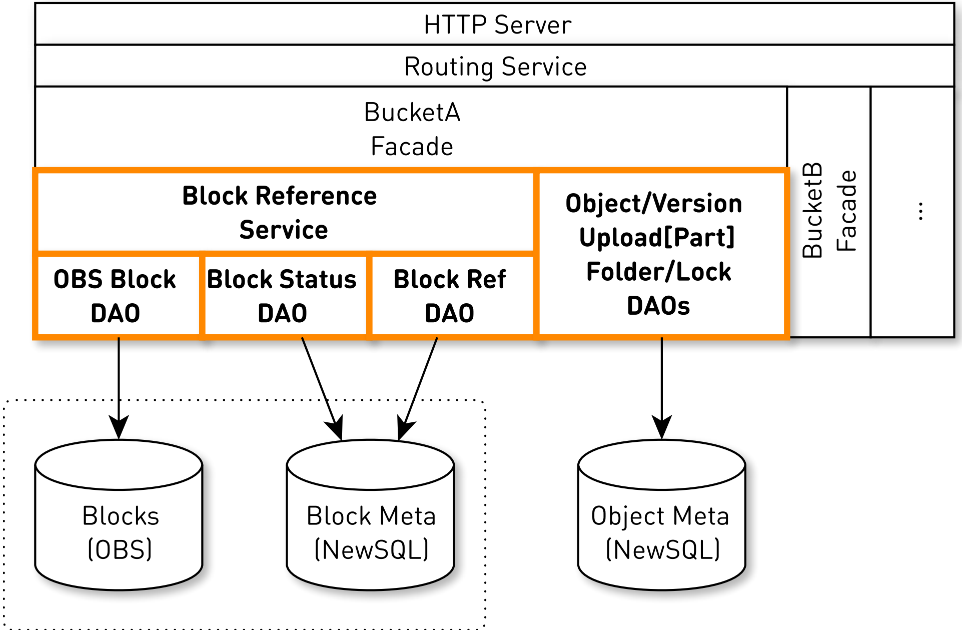
Patched `TemporaryLocationStrategy`

```java
public class TemporaryLocationStrategy
        extends VolumeChapterLocationStrategy {
    @Override
    public String location(final BlobId blobId) {
        // Delegate striping to parent
        // VolumeChapterLocationStrategy
        return String.format(
            "tmp/%s", super.location(blobId));
    }
}
```

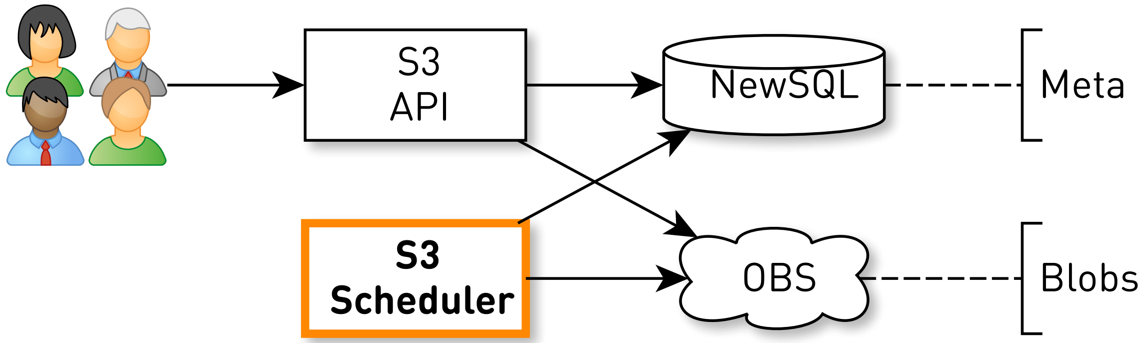
Profit

content/tmp/vol-17/chap-42/	abc	∅
content/tmp/vol-23/chap-07/	dbe	∅
content/tmp/vol-31/chap-01/	0ab	∅

content/vol-17/chap-42/	abc	...
content/vol-23/chap-07/	dbe	...
content/vol-31/chap-01/	0ab	...


HTTP Server
Routing Service
BucketA Facade
Block Reference Service
OBS Block DAO
Block Status DAO
Block Ref DAO
Object/Version Upload[Part] Folder/Lock DAOs
BucketB Facade
⋮
Blocks (OBS)
Block Meta (NewSQL)
Object Meta (NewSQL)

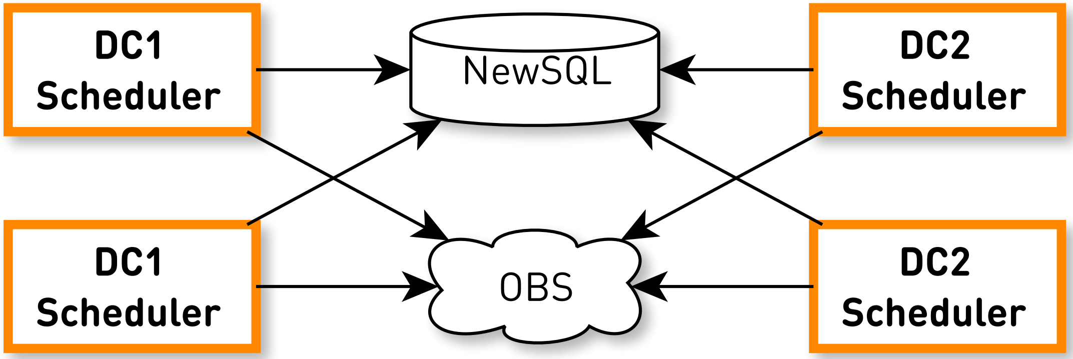
Bucket Lifecycle Scheduler



Bucket Lifecycle Scheduler

1. **Load** bucket lifecycle rules
2. Streaming **version scan**
3. Streaming **group by object**
4. **Filter** by rule
5. **Apply** the rule

Split work


DC1 Scheduler
NewSQL
DC2 Scheduler
DC1 Scheduler
OBS
DC2 Scheduler

Distributed parallel scanning

```sql
SELECT
    parent, name, version, etag, tags
FROM
    versions
WHERE
    TOKEN(parent) >= ? AND TOKEN(parent) <= ?
```

1M expiring objects in a folder...

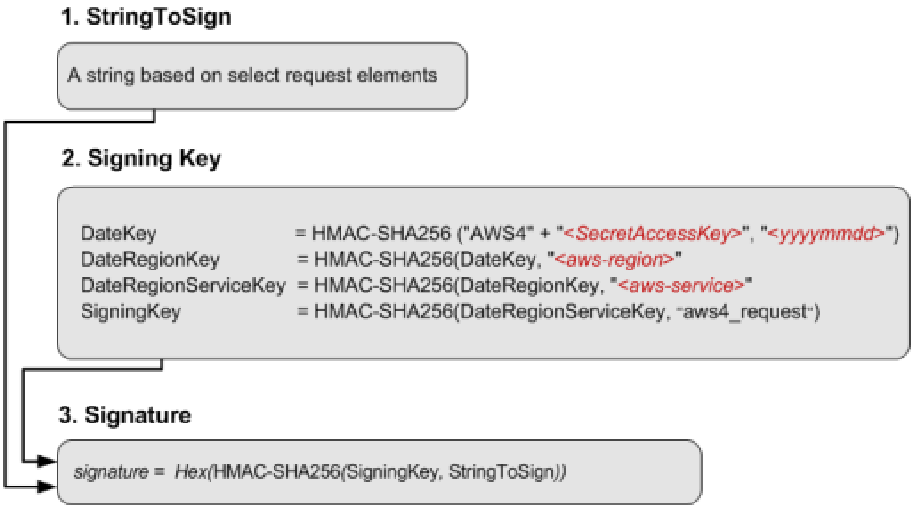
- A proven way to build a **cemetery of tombstones**
- With degraded performance and **timeouts**[10]

Solution

Per folder **tombstone throttling** when applying rules.

[10]See wide C* partition

Gotchas



1. StringToSign

A string based on select request elements

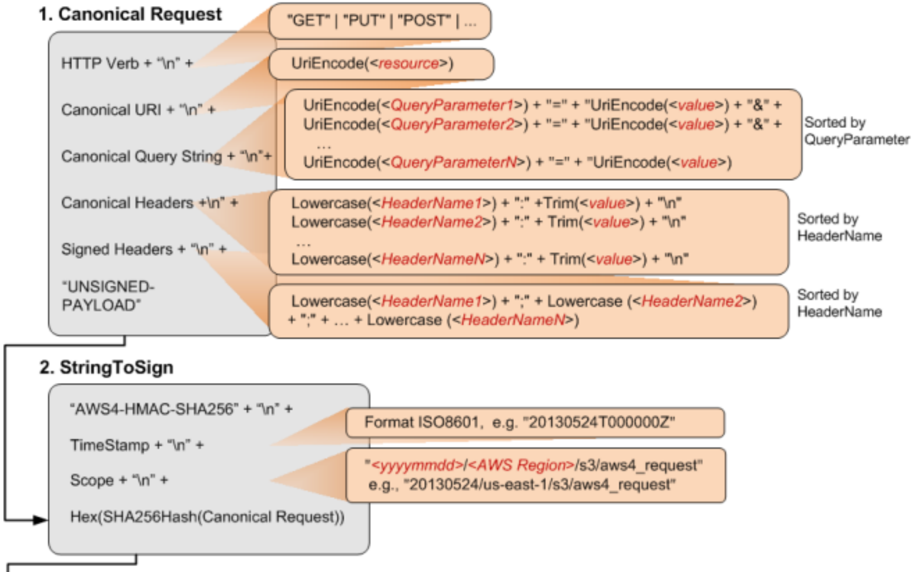
2. Signing Key

```
DateKey                  = HMAC-SHA256 ("AWS4" + "<SecretAccessKey>", "<yyyymmdd>")
DateRegionKey            = HMAC-SHA256(DateKey, "<aws-region>"
DateRegionServiceKey     = HMAC-SHA256(DateRegionKey, "<aws-service>"
SigningKey               = HMAC-SHA256(DateRegionServiceKey, "aws4_request")
```

3. Signature

signature = Hex(HMAC-SHA256(SigningKey, StringToSign))

AWS Signature Version 4 StringToSign



1. Canonical Request
HTTP Verb + "\n" +
Canonical URI + "\n" +
Canonical Query String + "\n"+
Canonical Headers +\n" +
Signed Headers + "\n" +
"UNSIGNED-PAYLOAD"
"GET" | "PUT" | "POST" | ...
UriEncode(<resource>)
UriEncode(<QueryParameter1>) + "=" + "UriEncode(<value>) + "&" +
UriEncode(<QueryParameter2>) + "=" + "UriEncode(<value>) + "&" +
...
UriEncode(<QueryParameterN>) + "=" + "UriEncode(<value>)
Sorted by QueryParameter
Lowercase(<HeaderName1>) + ":" +Trim(<value>) + "\n"
Lowercase(<HeaderName2>) + ":" + Trim(<value>) + "\n"
...
Lowercase(<HeaderNameN>) + ":" + Trim(<value>) + "\n"
Sorted by HeaderName
Lowercase(<HeaderName1>) + ";" + Lowercase (<HeaderName2>) + ";" + ... + Lowercase (<HeaderNameN>)
Sorted by HeaderName
2. StringToSign
"AWS4-HMAC-SHA256" + "\n" +
TimeStamp + "\n" +
Scope + "\n" +
Hex(SHA256Hash(Canonical Request))
Format ISO8601, e.g. "20130524T000000Z"
"<yyyymmdd>/<AWS Region>/s3/aws4_request"
e.g., "20130524/us-east-1/s3/aws4_request"

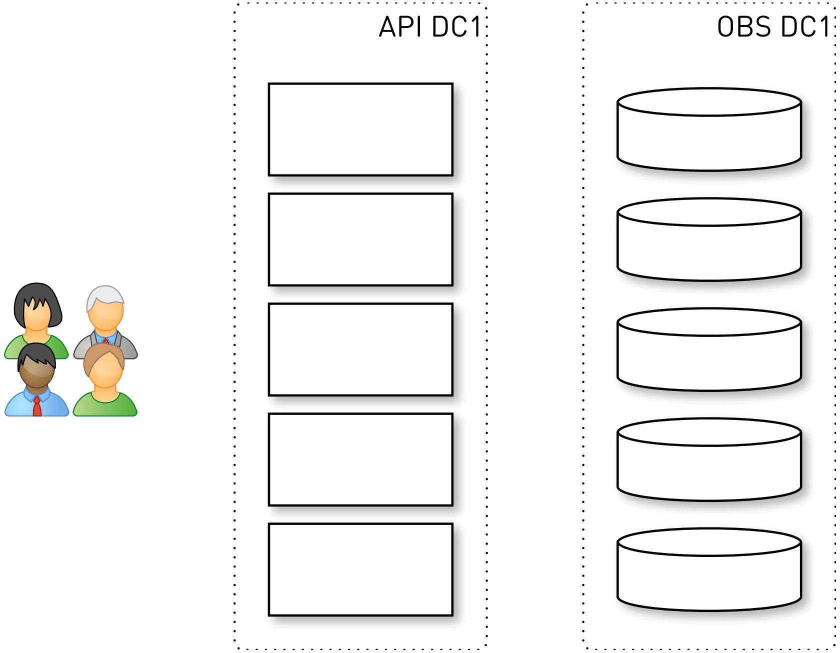
Gotchas

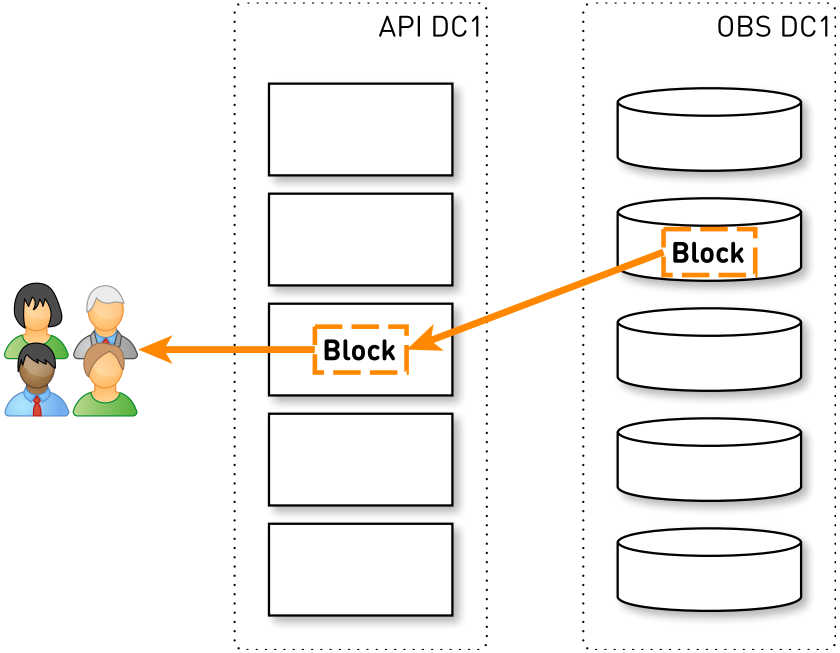
- Intricate canonicalization rules
- **Inconsistent** API[11] (i.e. Presigned URLs)
- Multipart object eTag
- **Custom** `UriEncode()` ☻
- Broken examples
- `Expect: 100-continue`
- `x-amz-content-sha256: STREAMING-AWS4-HMAC-SHA256-PAYLOAD`

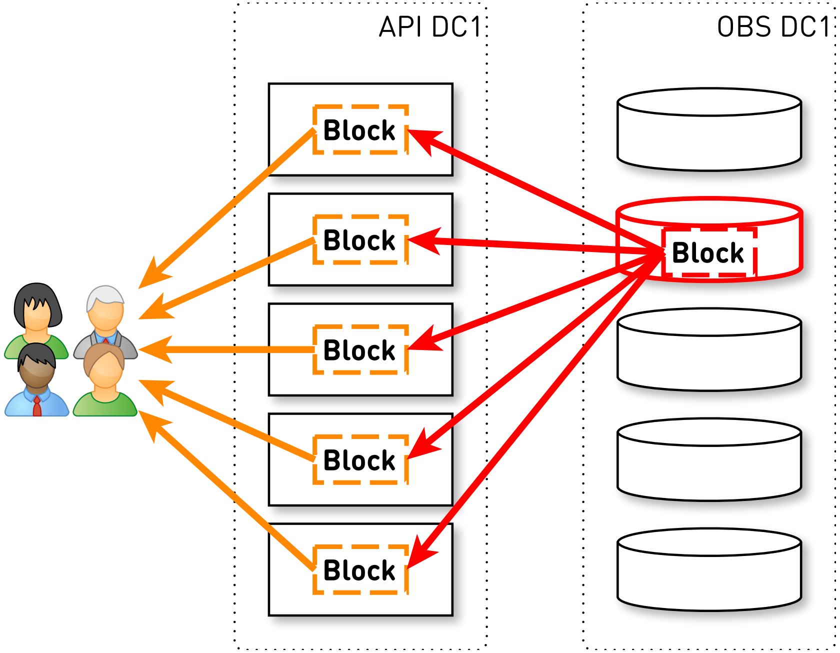
[11] Developer guide - AWS SDK for Java 2.x

Performance

- Reads
- Writes
- Latency
- Throughput
- Hotspots


API DC1
OBS DC1

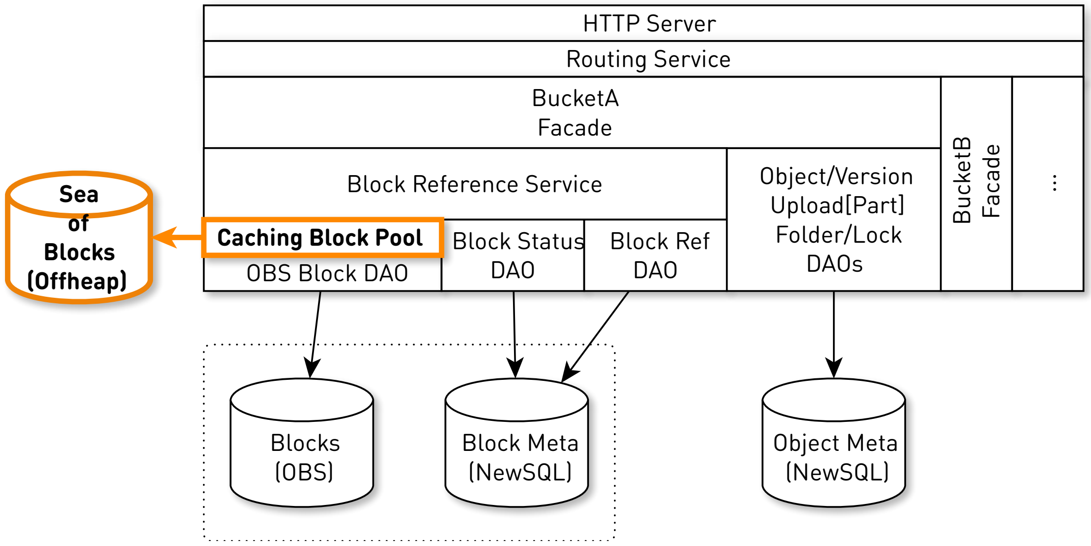

API DC1
OBS DC1
Block
Block

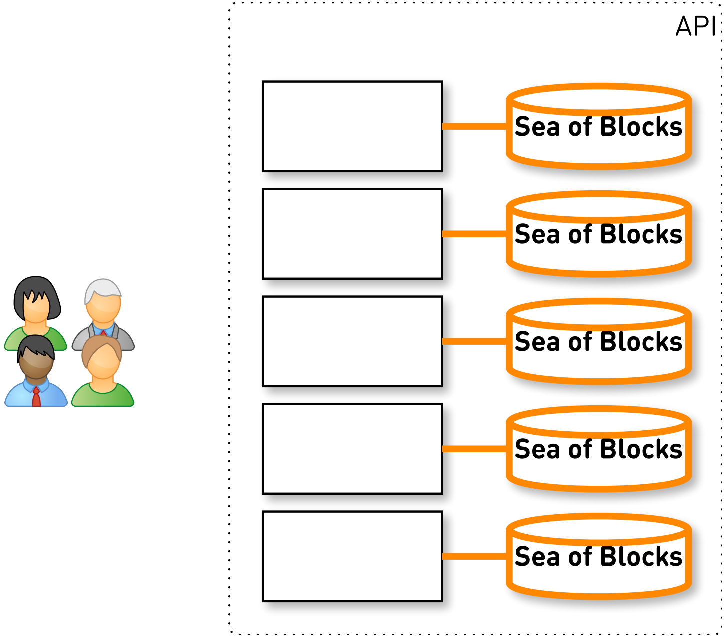

API DC1
OBS DC1
Block
Block
Block
Block
Block
Block

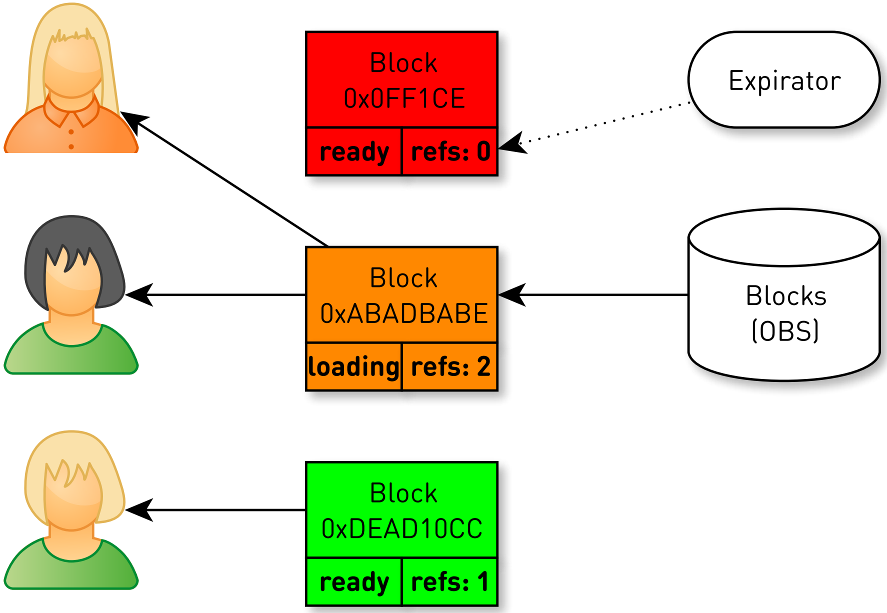
Naive block serving

- **Popular** blocks
  - Request/traffic **amplification**
  - **Identical** requests
  - Each block served by **3 blob replicas** (link saturation)
- **Extra data copying**[12]

[12] Vadim Tsesko. Reactive ok.ru/music streamer @ Joker 2018 (RU)

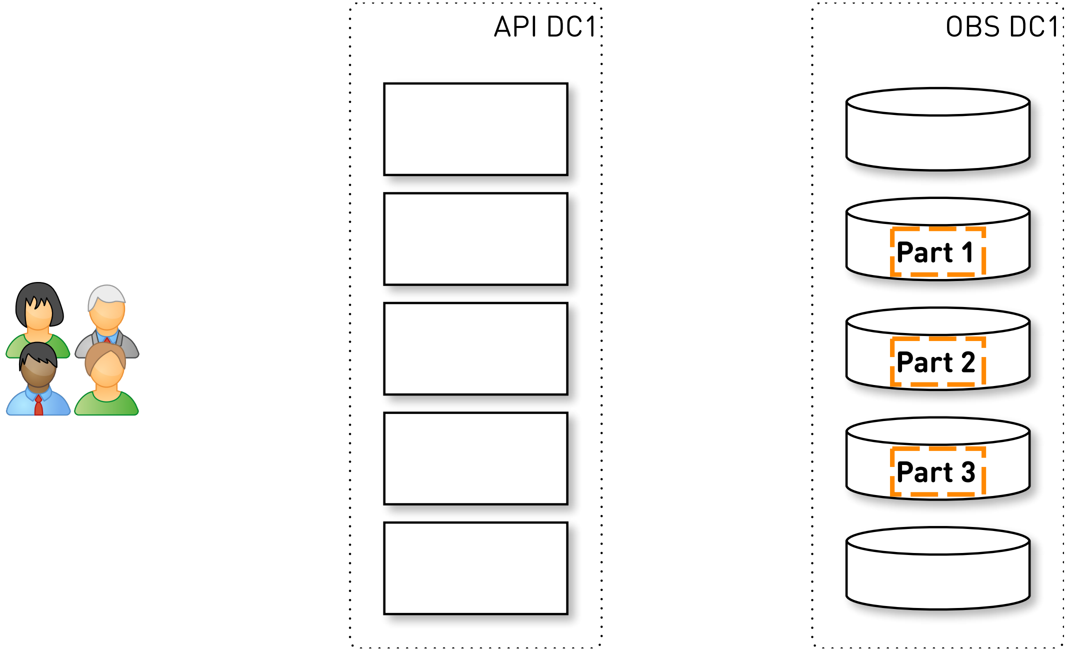

HTTP Server
Routing Service
BucketA Facade
BucketB Facade
⋮
Block Reference Service
Object/Version Upload[Part] Folder/Lock DAOs
Caching Block Pool
OBS Block DAO
Block Status DAO
Block Ref DAO
Sea of Blocks (Offheap)
Blocks (OBS)
Block Meta (NewSQL)
Object Meta (NewSQL)

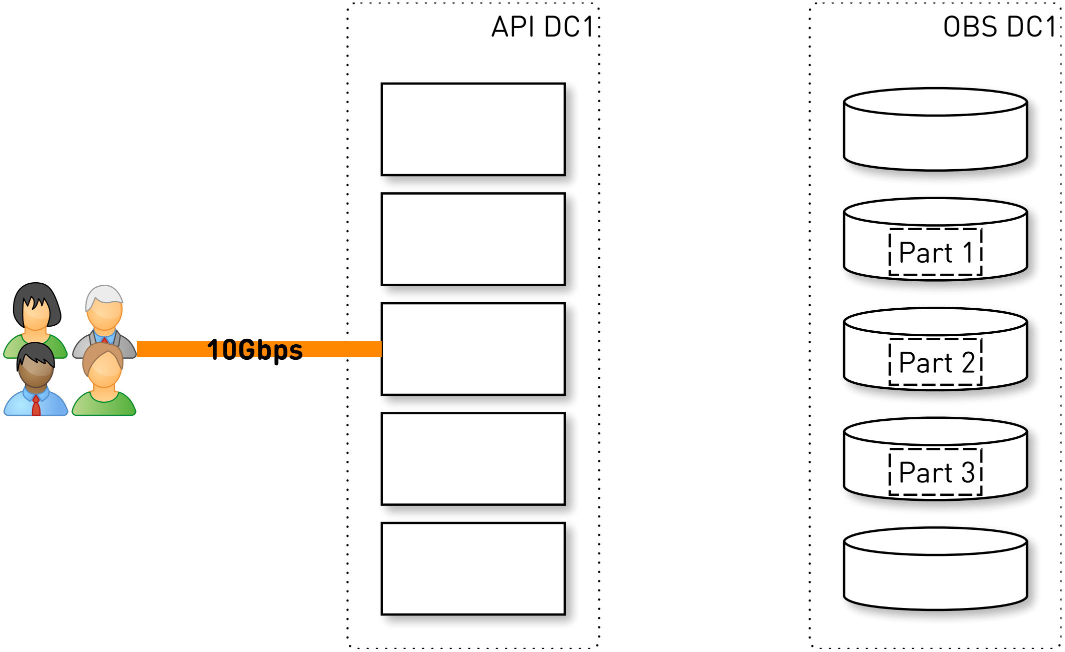

API
Sea of Blocks
Sea of Blocks
Sea of Blocks
Sea of Blocks
Sea of Blocks

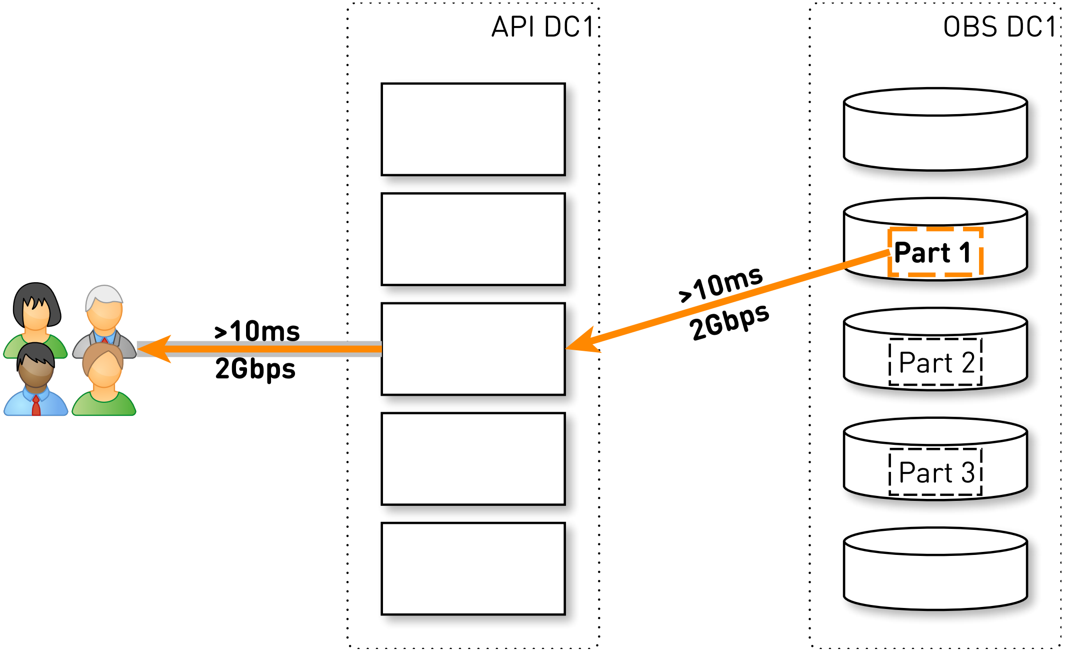

Block
0x0FF1CE
ready
refs: 0
Expirator
Block
0xABADBABE
loading
refs: 2
Blocks
(OBS)
Block
0xDEAD10CC
ready
refs: 1

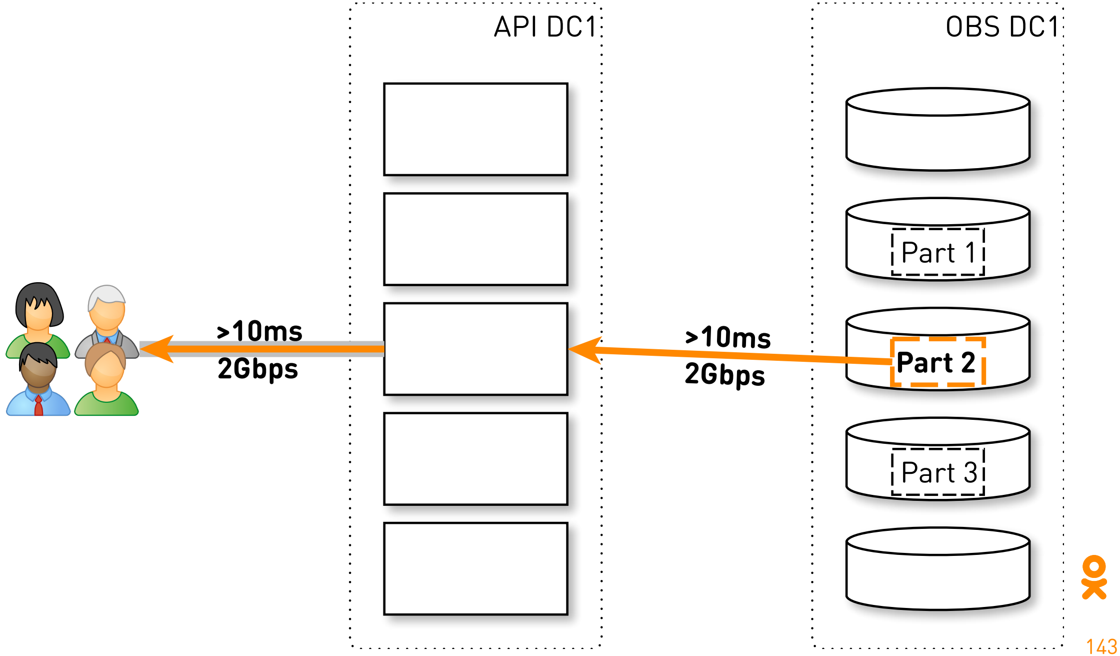
Sea of Blocks

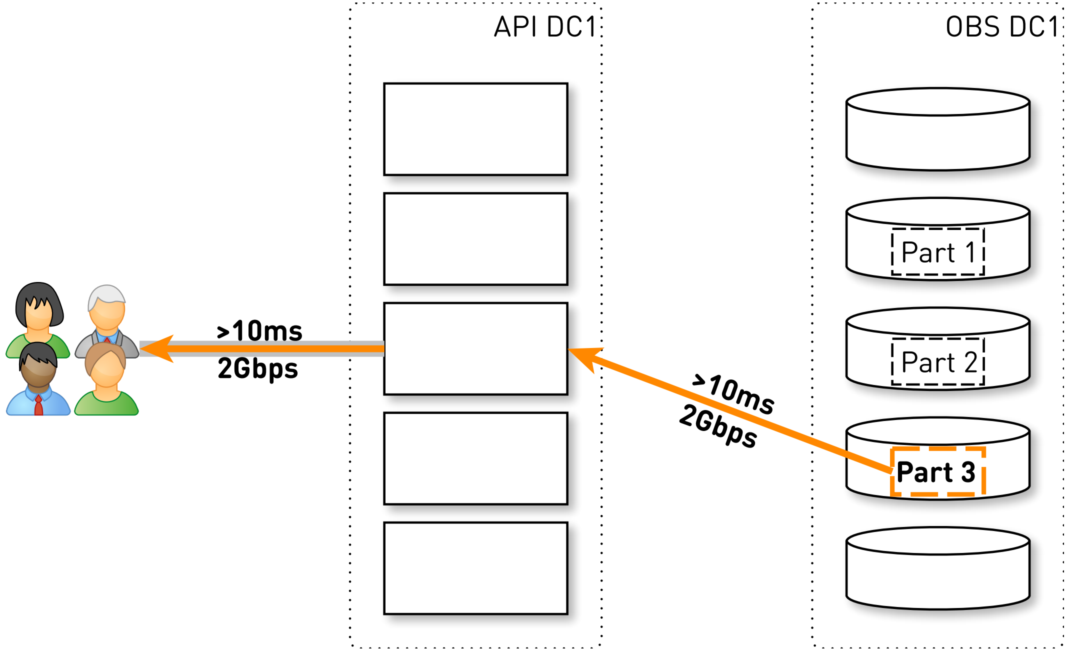
- **Cache** recent blocks
  - Lower OBS load
  - Lower latency
  - **LRU** eviction
- Each block loaded **only once**
  - `CompletableFuture<CloseableByteBuffer>`
- **Offheap**
  - No extra memory copying
  - Reference tracking


API DC1
OBS DC1
Part 1
Part 2
Part 3


10Gbps
API DC1
OBS DC1
Part 1
Part 2
Part 3


API DC1
OBS DC1
Part 1
Part 2
Part 3
>10ms
2Gbps
>10ms
2Gbps

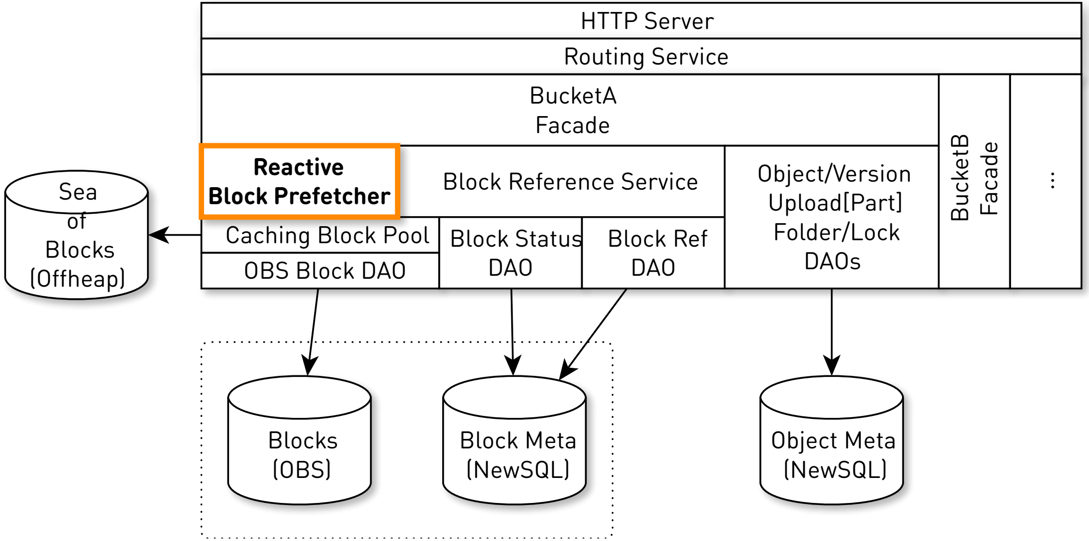

API DC1
OBS DC1
Part 1
Part 2
Part 3
>10ms
2Gbps
>10ms
2Gbps


API DC1
OBS DC1
Part 1
Part 2
Part 3
>10ms
2Gbps
>10ms
2Gbps

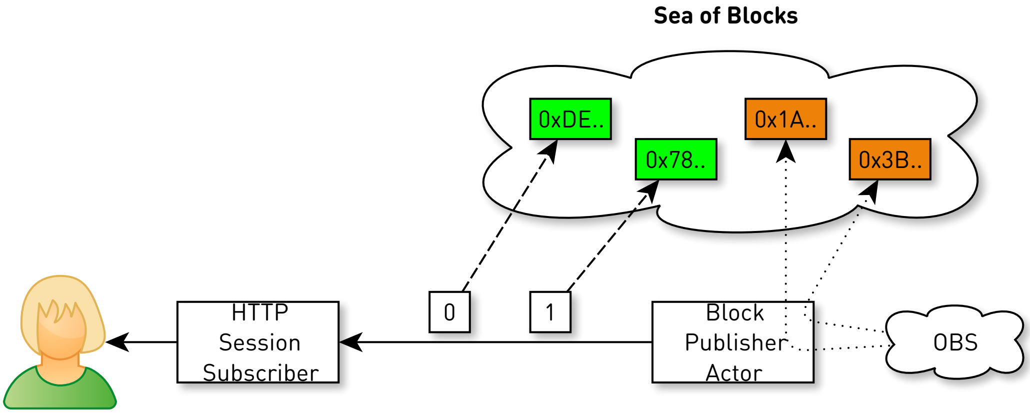
Huge Multipart Objects

If parts served **sequentially** and **synchronously**:

- At the speed of **single OBS node**
- **Delays** to load next block
- **Underutilized** client throughput


HTTP Server
Routing Service
BucketA Facade
BucketB Facade
⋮
Reactive Block Prefetcher
Block Reference Service
Object/Version Upload[Part] Folder/Lock DAOs
Caching Block Pool
OBS Block DAO
Block Status DAO
Block Ref DAO
Sea of Blocks (Offheap)
Blocks (OBS)
Block Meta (NewSQL)
Object Meta (NewSQL)

Reactive Block Prefetcher[13]



[13] Powered by `odnoklassniki/one-nio`.

Reads are fast

- **Single non-transactional query** to get object meta and list of blocks
- Prefetch from **many OBS nodes** in parallel
- Wait for **the first block only if uncached**
- Reactive streams
- **Back pressure**

Modifications are slower

- **Common case**[14]
  - **4 sequential** transactions (10-30 ms)
  - **Write/remove** from OBS (30-300 ms)
- **Best case**[15]
  - **3 sequential** transactions
- **Amortized** due to **parallel** multipart upload

[14] Upload unique block or remove last block reference

[15] Deduplicated block

Numbers

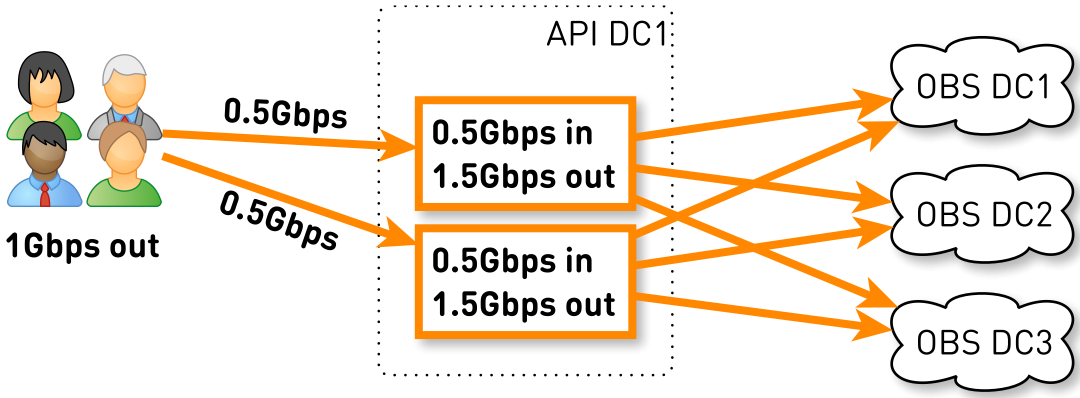
Client hardware: **4x10 Gbps** + 256 GB **/dev/shm**

```
$ cat ~/.aws/config
...
s3 =
    signature_version = s3v4
    max_concurrent_requests = 64
    multipart_threshold = 16MB
    multipart_chunksize = 16MB
```

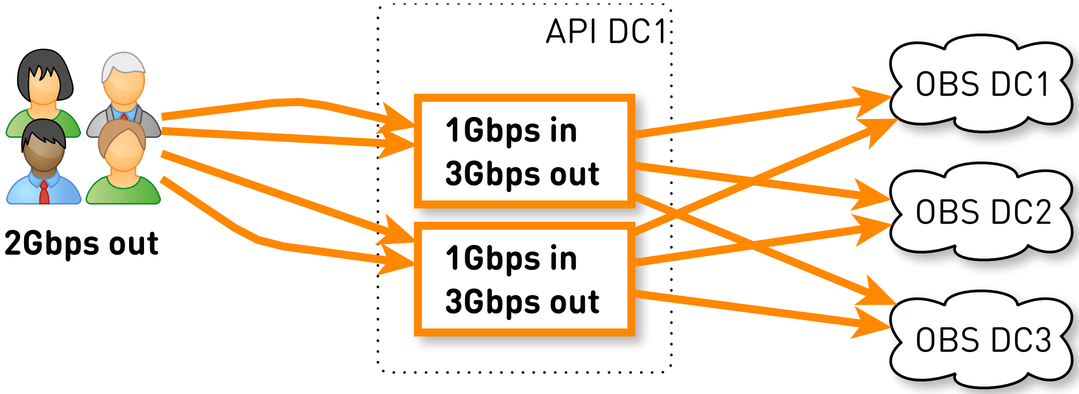
/dev/shm/s3/200GB.random

```
$ dd if=<(openssl enc -aes-256-ctr -pass pass:"$(dd
    if=/dev/urandom bs=128 count=1 2>/dev/null | base64)"
    -nosalt < /dev/zero) of=/dev/shm/s3/200GB.random
    bs=1G count=200 iflag=fullblock
```

```
$ aws s3 cp 200GB.random \
    s3://sandbox/tsesko/stress/
```



0.5Gbps
0.5Gbps
1Gbps out
API DC1
0.5Gbps in
1.5Gbps out
0.5Gbps in
1.5Gbps out
OBS DC1
OBS DC2
OBS DC3


2Gbps out
API DC1
1Gbps in
3Gbps out
1Gbps in
3Gbps out
OBS DC1
OBS DC2
OBS DC3

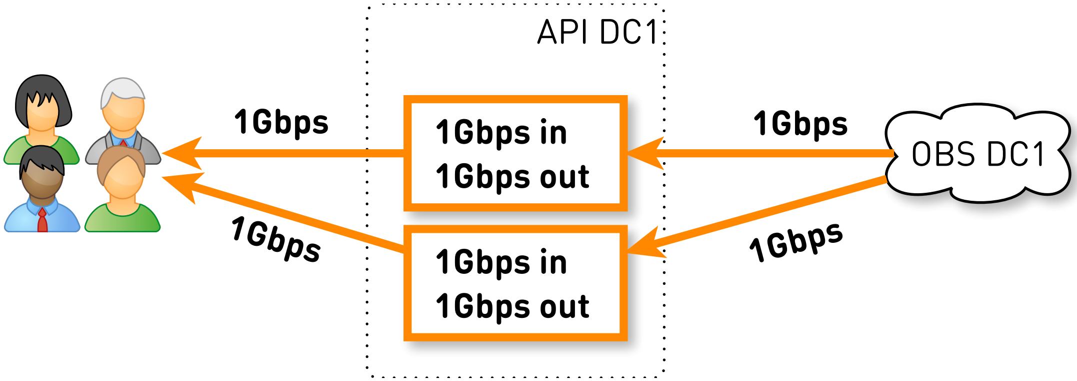
Upload from `/dev/shm`

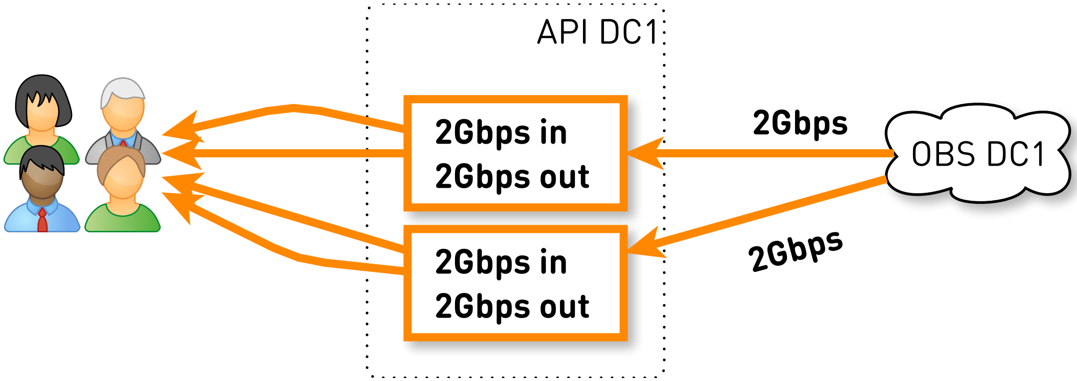
- **1 Gbps** AWS CLI upload limit
- 2 CLI instances hit **2 containers** in **1 DC**
- 4 cores, 1 Gbps IN, 3 Gbps OUT (each)
- CPU profile[16]
  - 47% SHA-256 (block ID)
  - 24% MD5 (checksum)
  - 15% `NativeSslSocket.read()` (HTTPS)
  - 6% `NativeSocket.write()` (OBS)

[16]https://github.com/jvm-profiling-tools/async-profiler

Download to /dev/shm

```
$ aws s3 cp \
    s3://sandbox/tsesko/stress/200GB.random \
    200GB.random
```


API DC1
1Gbps
1Gbps in
1Gbps out
1Gbps
OBS DC1
1Gbps
1Gbps in
1Gbps out
1Gbps


API DC1
2Gbps in
2Gbps out
2Gbps in
2Gbps out
2Gbps
OBS DC1
2Gbps

Download to `/dev/shm`

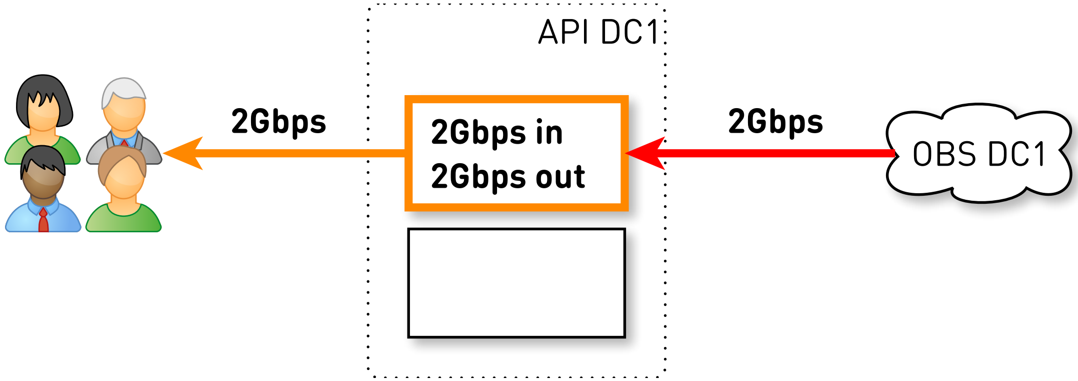
- **2 Gbps** AWS CLI download limit
- **2 processes** — **2x** throughput
- **Ranges** by AWS CLI
- 2 cores, 2 Gbps IN, 2 Gbps OUT (each)
- CPU profile[17]
    - 55% SHA-256 (checksum)
    - 17% `NativeSocket.readRaw()` (OBS)
    - 15% `NativeSslSocket.writeRaw()` (HTTPS)

[17]https://github.com/jvm-profiling-tools/async-profiler

Download presigned в /dev/null

```
$ aws s3 presign s3://sandbox/tsesko/stress/200GB.random
<presigned URL>
```

```
$ curl "<presigned URL>" >/dev/null
```

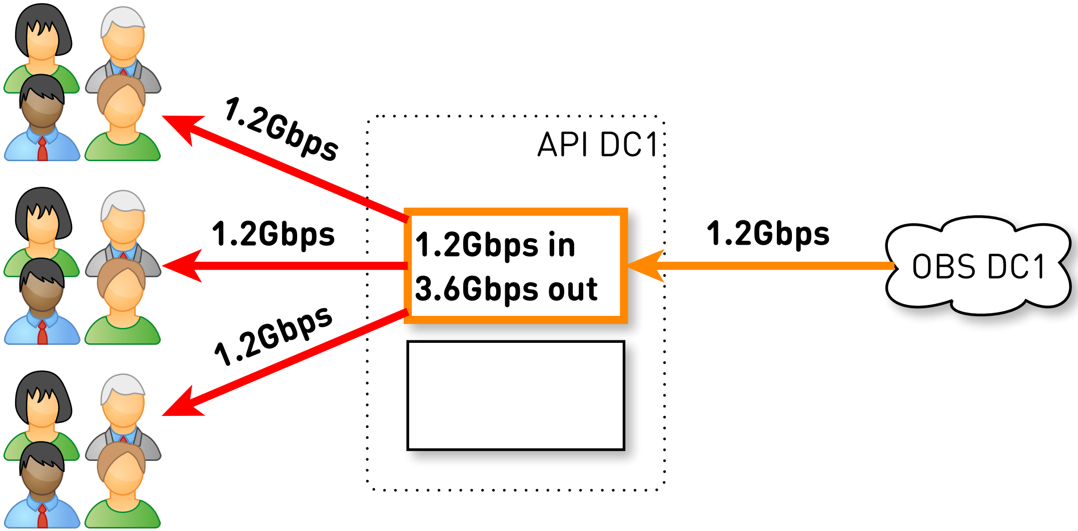

API DC1
2Gbps
2Gbps in
2Gbps out
2Gbps
OBS DC1

Download presigned в `/dev/null`

- **Single connection**
- **2 Gbps** download speed
- Limited by **inbound network** from OBS to API

Simulate popular object

```
$ curl "<presigned URL>" >/dev/null &
$ curl "<presigned URL>" >/dev/null &
$ curl "<presigned URL>" >/dev/null &
```


1.2Gbps
1.2Gbps
1.2Gbps
API DC1
1.2Gbps in
3.6Gbps out
1.2Gbps
OBS DC1

Single-node stress test

- **3x** presigned `curl 200GB.random`
- Serves **3.6 Gbps** (4 Gbps quota)
- Downloads **1.2 Gbps** from OBS
- Block pool **cache hit 60%**
- **2 cores**[18]
    - 46% `NativeSslSocket.writeRaw()` (HTTPS)
    - 35% SHA-256 (checksum)
    - 10% `NativeSocket.readRaw()` (OBS)
    - GC is idle (offheap)

[18] https://github.com/jvm-profiling-tools/async-profiler

Production

- Tens of buckets and growing
- Buckets up to **50 TB**
- Up to **100M objects** per bucket
- $>$ **6Krps**
- **99.9%** $<$ **15 ms** (meta)

Use Cases

- Docker registry
- Airflow Logs
- Nexus Artifacts
- Teamcity Artifacts
- Flink Checkpoints
- ...

Internal Users

- RPM repository
- Static resources
- Autotest artifacts
- Android profiling artifacts[19]
- Continuous profiles[20]
- RuStore beta

[19] Kirill Popov. Profiling in production @ Mobius 2021 Piter (RU)
[20] Andrei Pangin. Continuous eBPF-assisted cloud profiling @ JPoint 2022

Вадим Цесько

Одноклассники

incubos.org/contacts

oktech.ru

Оценить доклад

Saint HighLoad++ 2022