DATA IS POTENTIAL

CORTX-S3 Architecture

Basavaraj Kirunge

SEAGATE

Agenda

CORTX S3 Server

- Overview
- Software stack
- S3 Metadata
- Object ID (OID) generator
- Upload (PUT) Object – Sequence
- Upload (PUT) Object overwrite– Sequence
- Delete Object - Sequence
- KVS Async API
- S3-Motr Async call sequence
- Adding a new S3 API
- Supported API

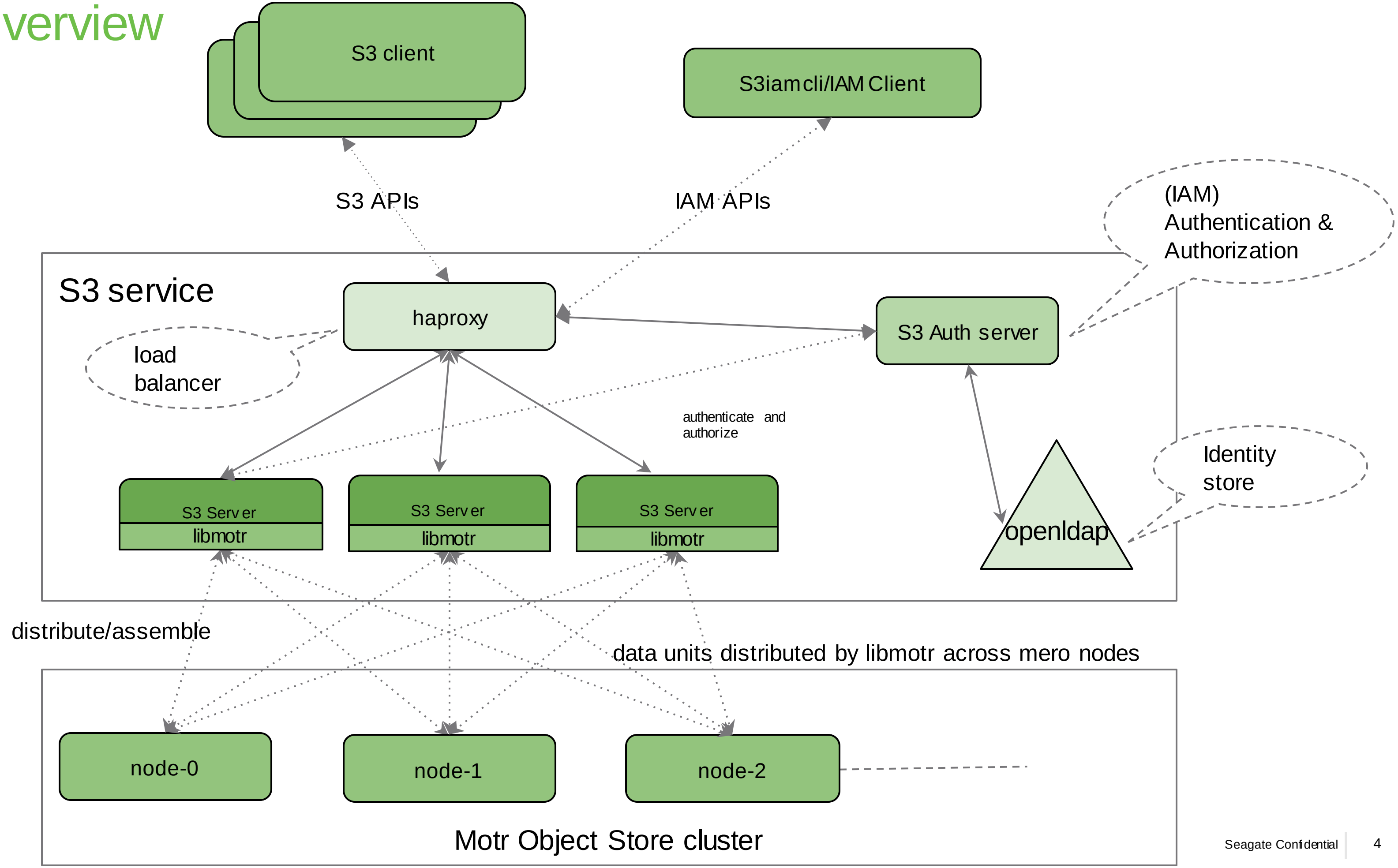
CORTX S3 Authserver

- Overview
- IAM metadata DIT
- Authentication and Authorization
- IAM API's

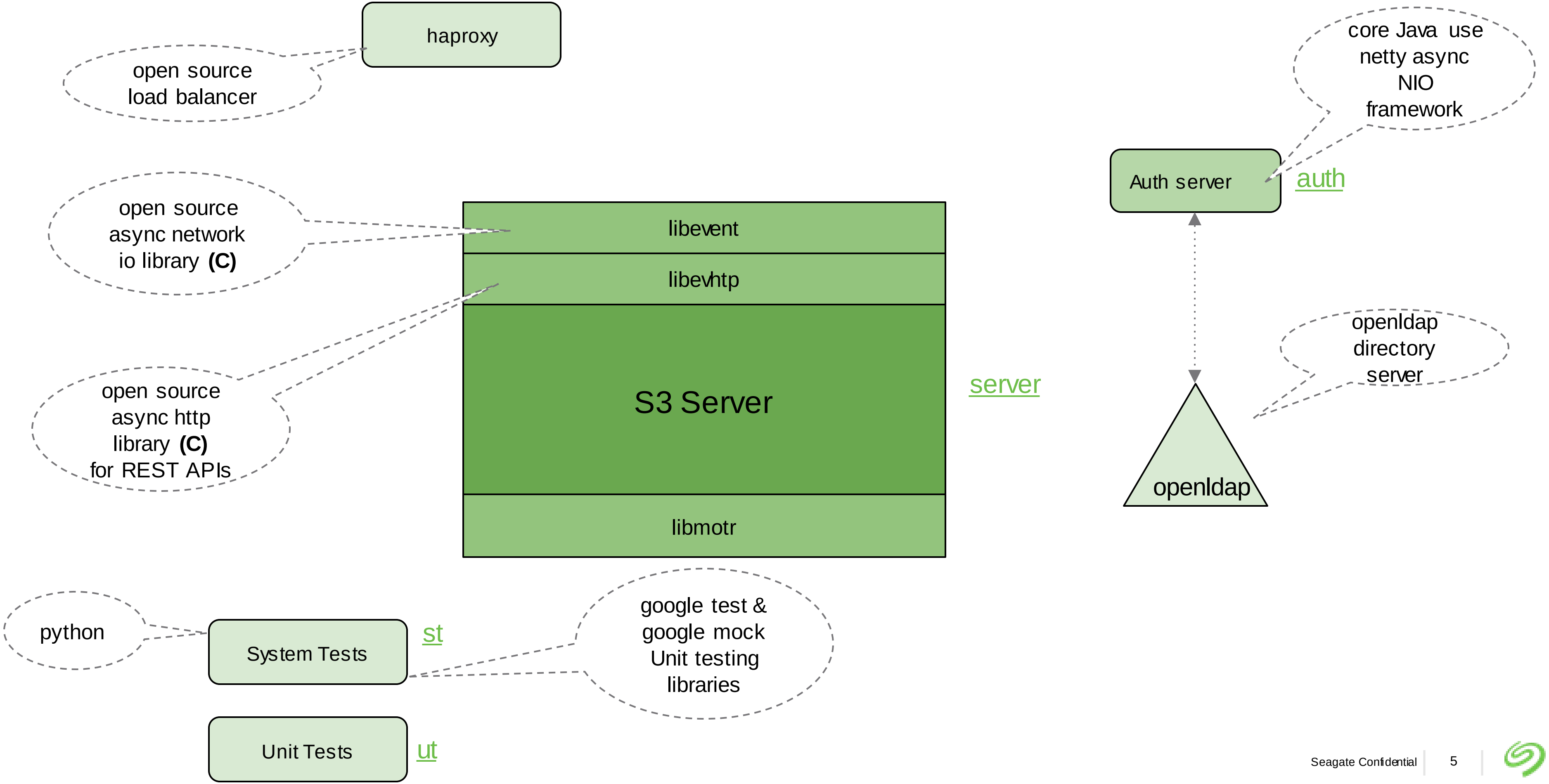
CORTX S3 Server overview

- S3 (Simple Object Storage) API interface to CORTX object storage.
- S3 server is developed in C++/C.
- S3 server based on libevent, async IO with event loop.
- S3 objects stored as CORTX Motr Objects.
- S3 object and bucket metadata is stored in Motr Key Value Store (KVS).
- Uses motr library C API (libmotr) to talk to Motr IO/KVS services.
- S3 Server can be installed on same Motr node or a separate node.

S3 Overview

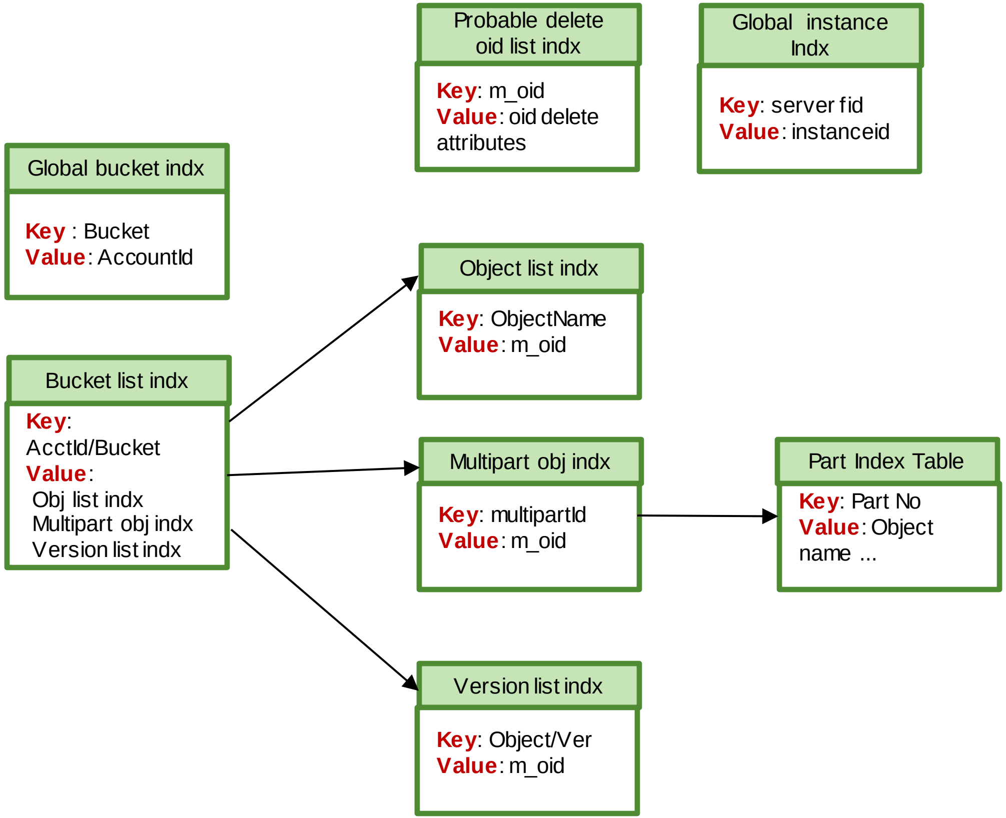

S3 client
S3iamcli/IAM Client
S3 APIs
IAM APIs
(IAM)
Authentication &
Authorization
S3 service
haproxy
S3 Auth server
load
balancer
authenticate and
authorize
S3 Server
libmotr
S3 Server
libmotr
S3 Server
libmotr
openldap
Identity
store
distribute/assemble
data units distributed by libmotr across mero nodes
node-0
node-1
node-2
Motr Object Store cluster

S3 software stack



haproxy
open source load balancer
libevent
open source async network io library (C)
libevhtp
open source async http library (C) for REST APIs
S3 Server
server
libmotr
Auth server
auth
core Java use netty async NIO framework
openldap
openldap directory server
python
System Tests
st
Unit Tests
ut
google test & google mock Unit testing libraries

S3 Metadata is Stored across Eight Motr Index Tables



Probable delete oid list indx

Key: m_oid
Value: oid delete attributes

Global instance Indx

Key: server fid
Value: instanceid

Global bucket indx

Key : Bucket
Value: AccountId

Bucket list indx

Key: AcctId/Bucket
Value:
Obj list indx
Multipart obj indx
Version list indx

Object list indx

Key: ObjectName
Value: m_oid

Multipart obj indx

Key: multipartId
Value: m_oid

Part Index Table

Key: Part No
Value: Object name ...

Version list indx

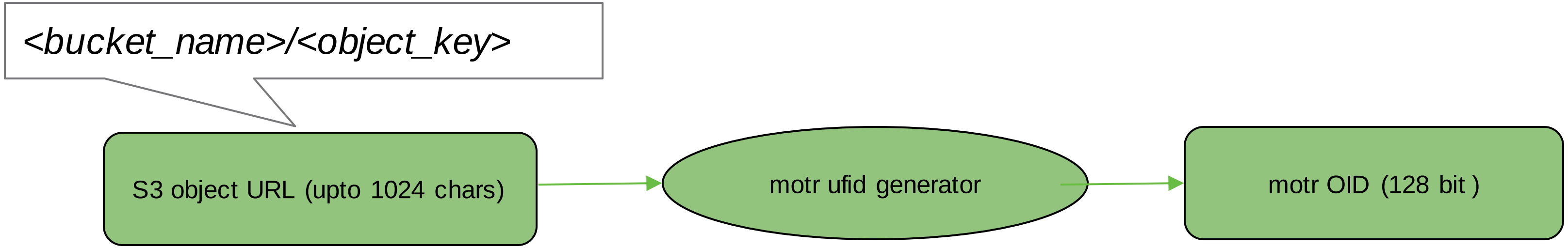
Key: Object/Ver
Value: m_oid

Key : String
Value : JSON

Object list index table (sample entry)

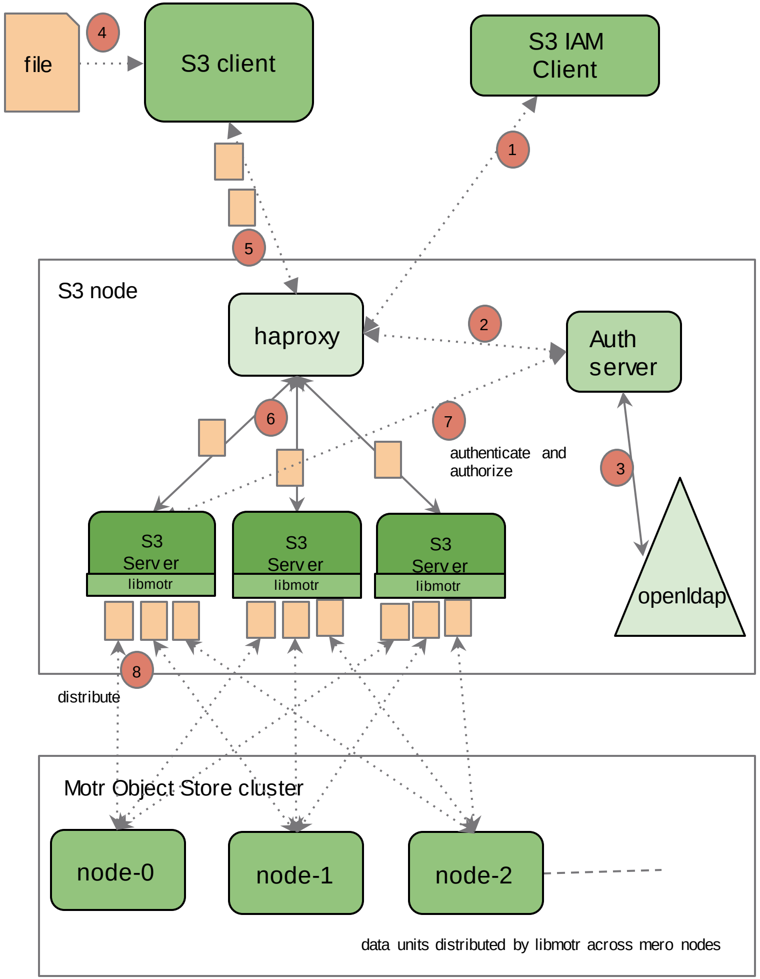
Object Key	Metadata (JSON)
file.txt	{"ACL":"<ACL>", "Bucket-Name":"cortx-bucket1", "Object-Name":"file.txt", "Object-URI":"cortx-bucket1\\file.txt", "System-Defined":{"Content-Length":"14866308", "Content-MD5":"50907fdfcdbe228e31216dcbe52f0fea-2", "Content-Type":"application/x-rpm", "Date":"2020-11-03T04:35:44.000Z", "Last-Modified":"2020-11-03T04:35:44.000Z", "Owner-Account":"cortx", "Owner-Account-id":"378252219198", "Owner-Canonical-id":"1a602b4df9e44ae9ac91733fa4e1f6770244b79245e3478ab3bfb49d284d4403", "Owner-User":"root", "Owner-User-id":"HzFu5EP1RXOy1GgATDhpbA" ,"x-amz-server-side-encryption":"None", "x-amz-server-side-encryption-aws-kms-key-id":"", "x-amz-server-side-encryption-customer-algorithm":"", "x-amz-server-side-encryption-customer-key":"", "x-amz-server-side-encryption-customer-key-MD5":"", "x-amz-storage-class":"STANDARD", "x-amz-version-id":"MTg0NDY3NDI0NjkzMzE0MDY3ODQ", "x-amz-website-redirect-location":"None"}, "create_timestamp":"2020-11-03T04:35:44.000Z" ,"layout_id":9, "**motr_oid":"rAp8AwAAAAA=-EgAAAAAA5pI="**}

S3 Object to motr object mapping



S3 URl - OID mapping stored in S3 Object metadata in KVS
S3UriToMotrOID

S3 and IAM workflow



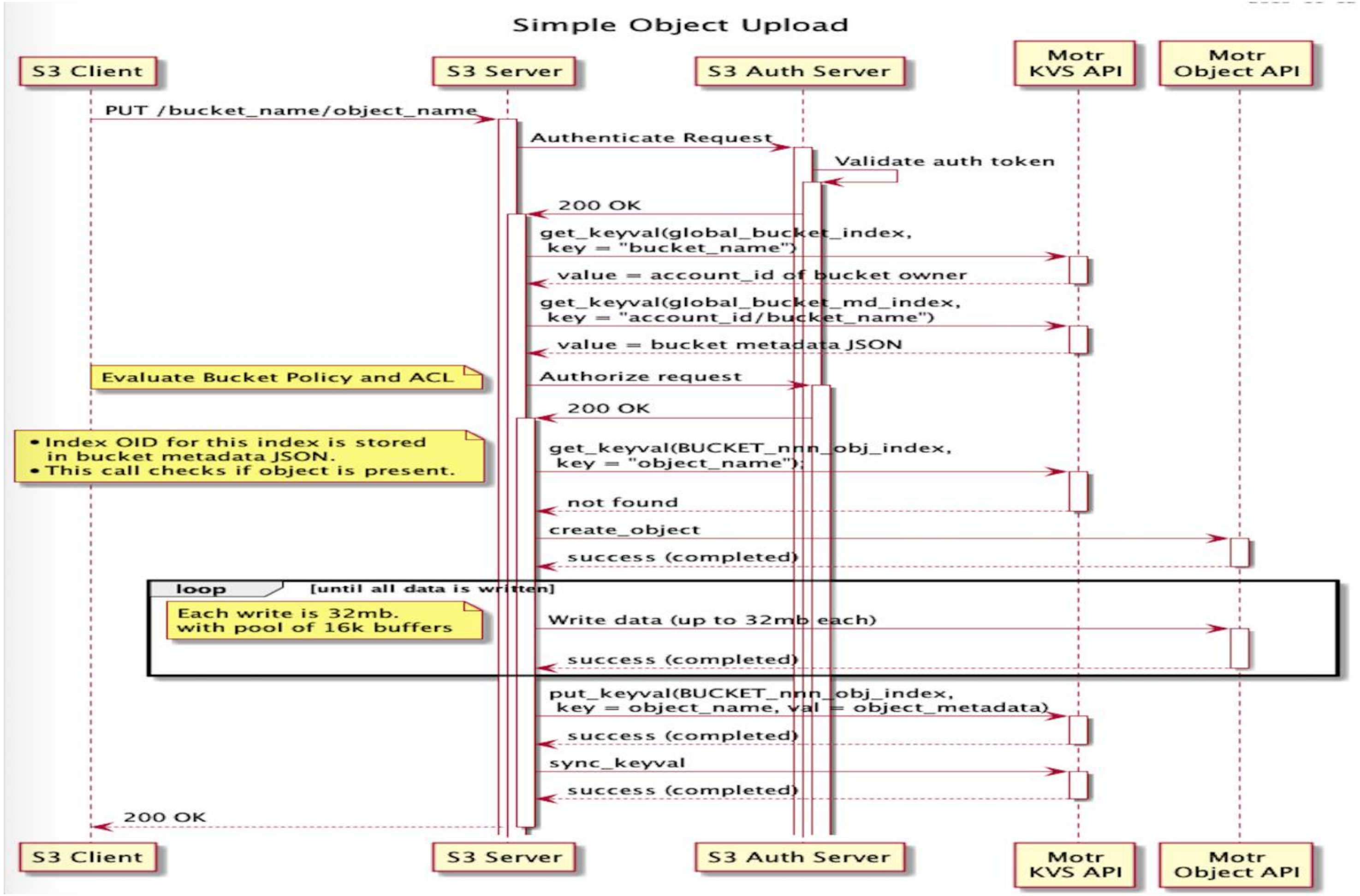
Identity access management (IAM)

1. Create Account/User/Access keys using credentials sent to haproxy.
2. haproxy forwards request to Auth server to create Account/User/Access keys.
3. Auth server authenticates request and creates Account/User/Access keys in openldap and response is sent back to s3iamcli via haproxy.

Object upload via S3 API

4. S3 client reads file to be uploaded as object.
5. S3 client uses PUT Object API to upload Object. For large object it divides file into parts and uploads using Multipart upload (POST Object, PUT Part and Complete upload) APIs.
6. haproxy receives these API requests and distributes to different S3 instances.
7. S3 instances request Auth server to verify the API signatures to authenticate and authorize the request.
8. S3 instance creates an object in motr and writes data using libmotr APIs. libmotr uses erasure coding/replication depending on configuration for data resiliency.

PUT object (Sequence diagram)

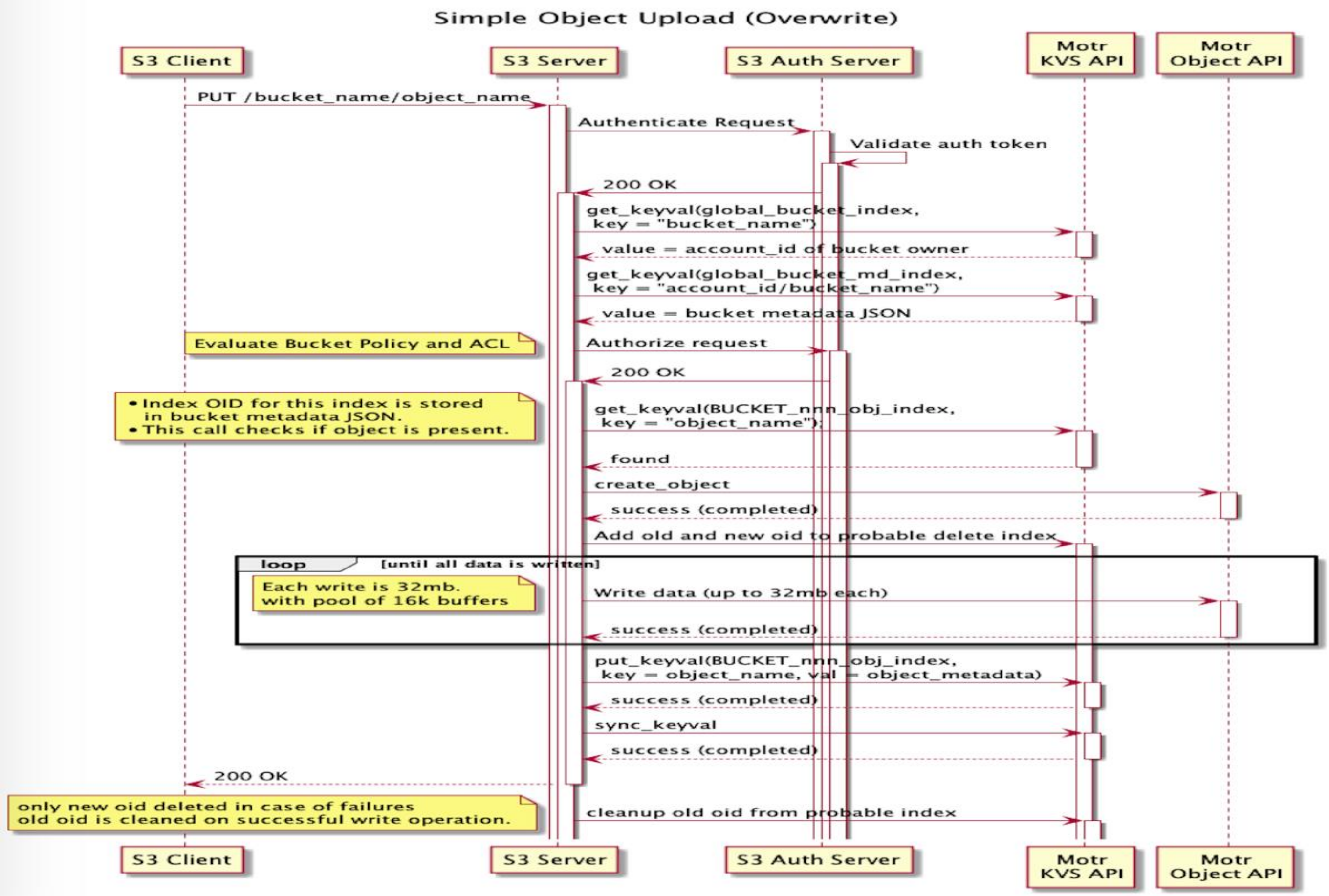

Simple Object Upload
S3 Client
S3 Server
S3 Auth Server
Motr KVS API
Motr Object API
PUT /bucket_name/object_name
Authenticate Request
Validate auth token
200 OK
get_keyval(global_bucket_index, key = "bucket_name")
value = account_id of bucket owner
get_keyval(global_bucket_md_index, key = "account_id/bucket_name")
value = bucket metadata JSON
Authorize request
200 OK
get_keyval(BUCKET_nnn_obj_index, key = "object_name");
not found
create_object
success (completed)
loop
[until all data is written]
Write data (up to 32mb each)
success (completed)
put_keyval(BUCKET_nnn_obj_index, key = object_name, val = object_metadata)
success (completed)
sync_keyval
success (completed)
200 OK
S3 Client
S3 Server
S3 Auth Server
Motr KVS API
Motr Object API

Evaluate Bucket Policy and ACL

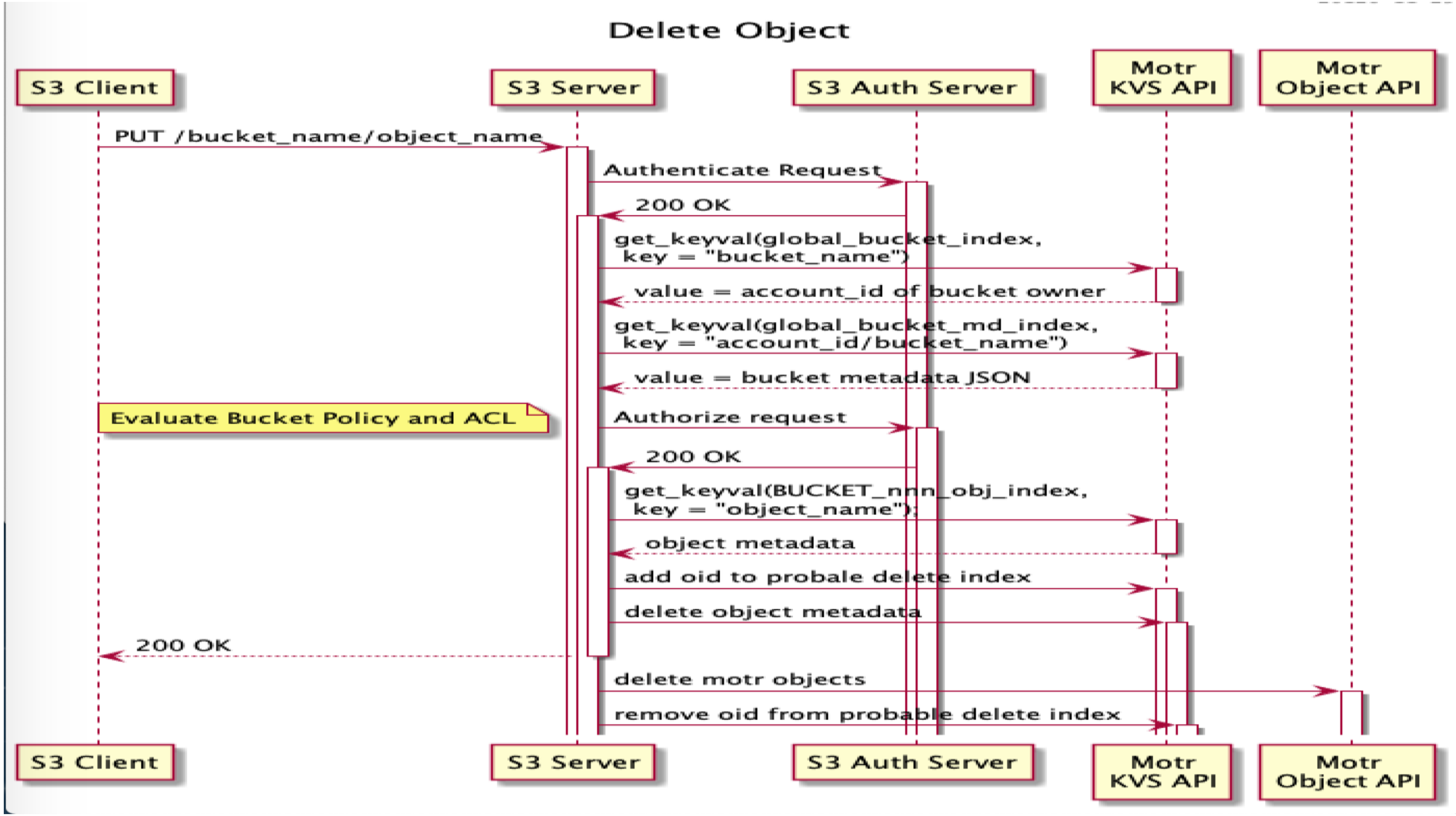
- Index OID for this index is stored in bucket metadata JSON.
- This call checks if object is present.

Each write is 32mb. with pool of 16k buffers

PUT object Overwrite (Sequence diagram)


Simple Object Upload (Overwrite)
S3 Client
S3 Server
S3 Auth Server
Motr KVS API
Motr Object API
PUT /bucket_name/object_name
Authenticate Request
Validate auth token
200 OK
get_keyval(global_bucket_index, key = "bucket_name")
value = account_id of bucket owner
get_keyval(global_bucket_md_index, key = "account_id/bucket_name")
value = bucket metadata JSON
Evaluate Bucket Policy and ACL
Authorize request
200 OK
• Index OID for this index is stored in bucket metadata JSON.
• This call checks if object is present.
get_keyval(BUCKET_nnn_obj_index, key = "object_name");
found
create_object
success (completed)
Add old and new oid to probable delete index
loop
[until all data is written]
Each write is 32mb. with pool of 16k buffers
Write data (up to 32mb each)
success (completed)
put_keyval(BUCKET_nnn_obj_index, key = object_name, val = object_metadata)
success (completed)
sync_keyval
success (completed)
200 OK
only new oid deleted in case of failures old oid is cleaned on successful write operation.
cleanup old oid from probable index
S3 Client
S3 Server
S3 Auth Server
Motr KVS API
Motr Object API

Delete Object (Sequence diagram)



Delete Object

S3 Client

S3 Server

S3 Auth Server

Motr KVS API

Motr Object API

PUT /bucket_name/object_name

Authenticate Request

200 OK

get_keyval(global_bucket_index,
key = "bucket_name")

value = account_id of bucket owner

get_keyval(global_bucket_md_index,
key = "account_id/bucket_name")

value = bucket metadata JSON

Evaluate Bucket Policy and ACL

Authorize request

200 OK

get_keyval(BUCKET_nnn_obj_index,
key = "object_name");

object metadata

add oid to probale delete index

delete object metadata

200 OK

delete motr objects

remove oid from probable delete index

S3 Client

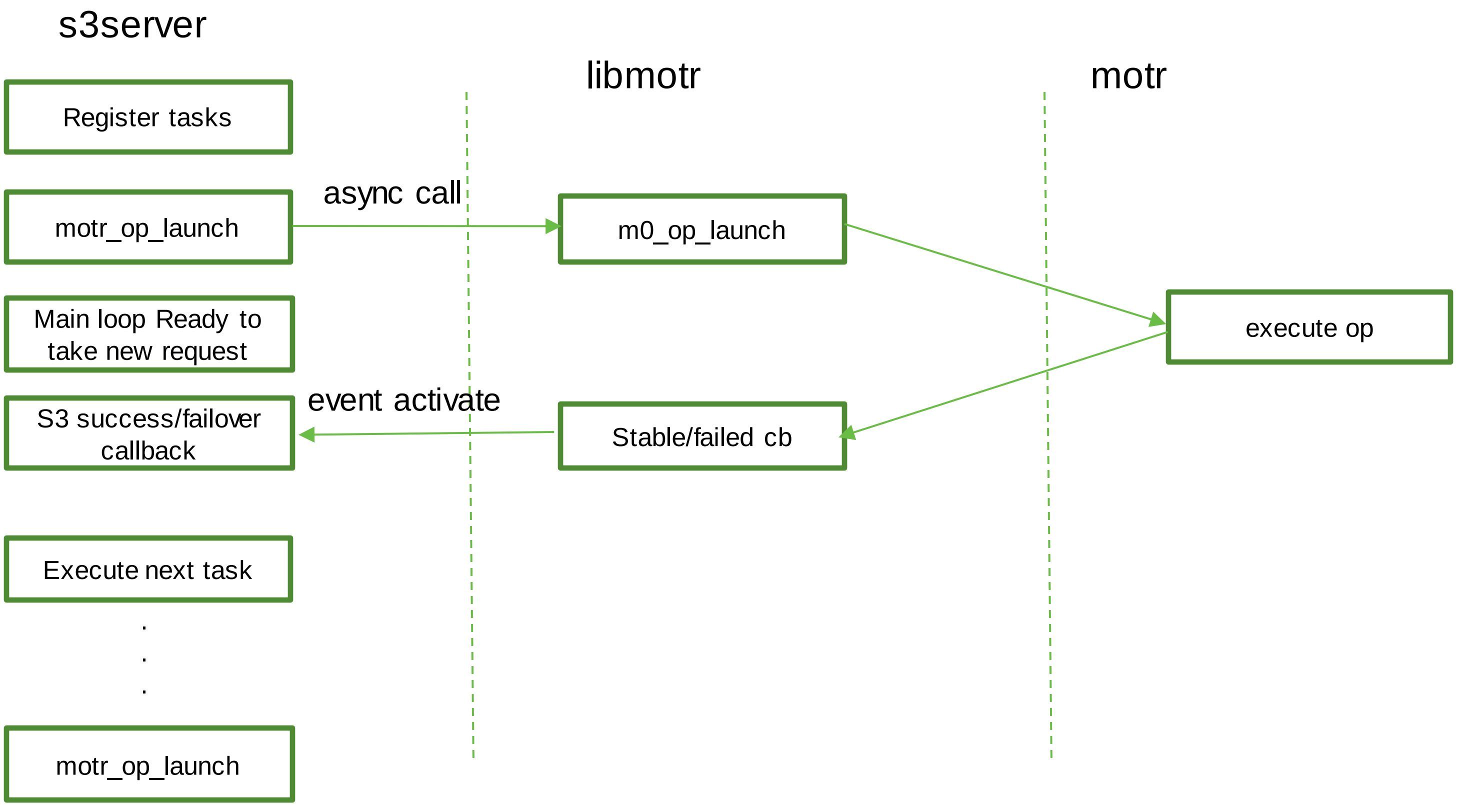
S3 Server

S3 Auth Server

Motr KVS API

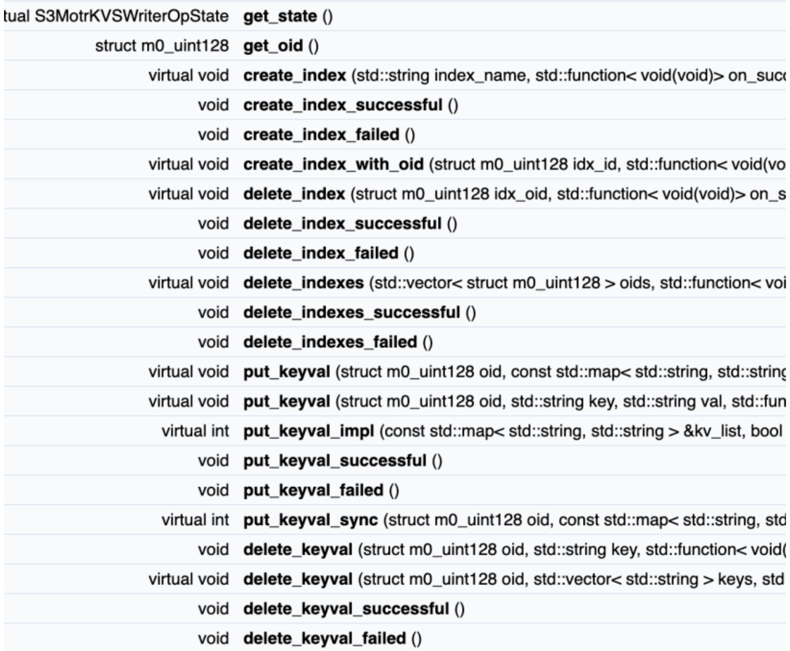
Motr Object API

S3 - Motr async call sequence



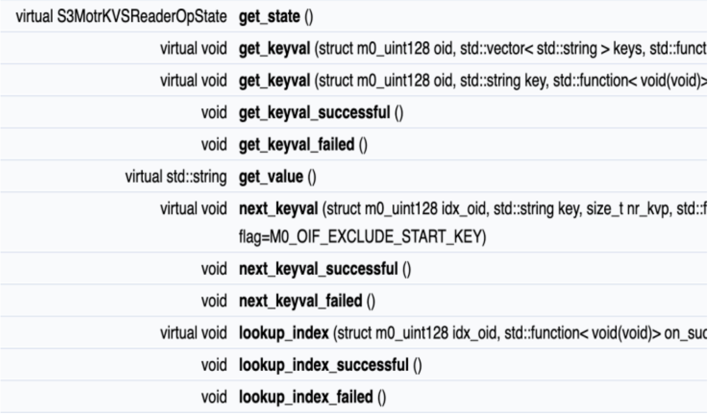
KVS async API

S3MotrKVSWriter



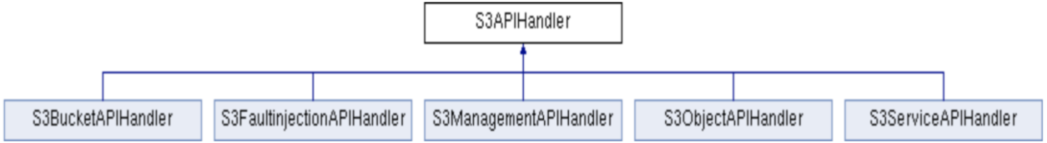
tual S3MotrKVSWriterOpState	**get_state** ()
struct m0_uint128	**get_oid** ()
virtual void	**create_index** (std::string index_name, std::function< void(void)> on_succ
void	**create_index_successful** ()
void	**create_index_failed** ()
virtual void	**create_index_with_oid** (struct m0_uint128 idx_id, std::function< void(vo
virtual void	**delete_index** (struct m0_uint128 idx_oid, std::function< void(void)> on_s
void	**delete_index_successful** ()
void	**delete_index_failed** ()
virtual void	**delete_indexes** (std::vector< struct m0_uint128 > oids, std::function< voi
void	**delete_indexes_successful** ()
void	**delete_indexes_failed** ()
virtual void	**put_keyval** (struct m0_uint128 oid, const std::map< std::string, std::string
virtual void	**put_keyval** (struct m0_uint128 oid, std::string key, std::string val, std::fun
virtual int	**put_keyval_impl** (const std::map< std::string, std::string > &kv_list, bool
void	**put_keyval_successful** ()
void	**put_keyval_failed** ()
virtual int	**put_keyval_sync** (struct m0_uint128 oid, const std::map< std::string, std
void	**delete_keyval** (struct m0_uint128 oid, std::string key, std::function< void(
virtual void	**delete_keyval** (struct m0_uint128 oid, std::vector< std::string > keys, std
void	**delete_keyval_successful** ()
void	**delete_keyval_failed** ()

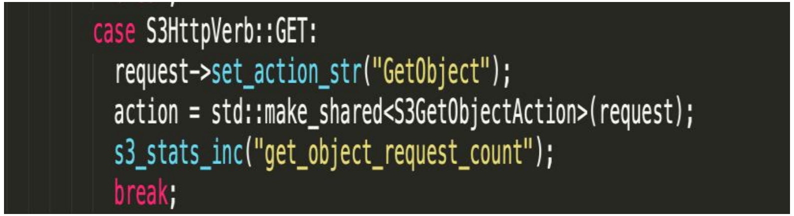
S3MotrKVSReader



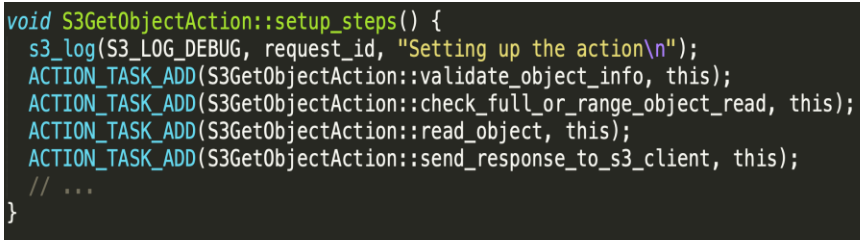
virtual S3MotrKVSReaderOpState	**get_state** ()
virtual void	**get_keyval** (struct m0_uint128 oid, std::vector< std::string > keys, std::funct
virtual void	**get_keyval** (struct m0_uint128 oid, std::string key, std::function< void(void)>
void	**get_keyval_successful** ()
void	**get_keyval_failed** ()
virtual std::string	**get_value** ()
virtual void	**next_keyval** (struct m0_uint128 idx_oid, std::string key, size_t nr_kvp, std::f flag=M0_OIF_EXCLUDE_START_KEY)
void	**next_keyval_successful** ()
void	**next_keyval_failed** ()
virtual void	**lookup_index** (struct m0_uint128 idx_oid, std::function< void(void)> on_suc
void	**lookup_index_successful** ()
void	**lookup_index_failed** ()

Adding new S3 API

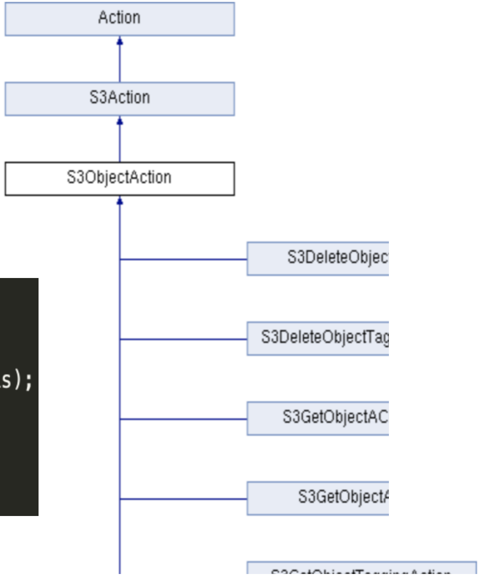


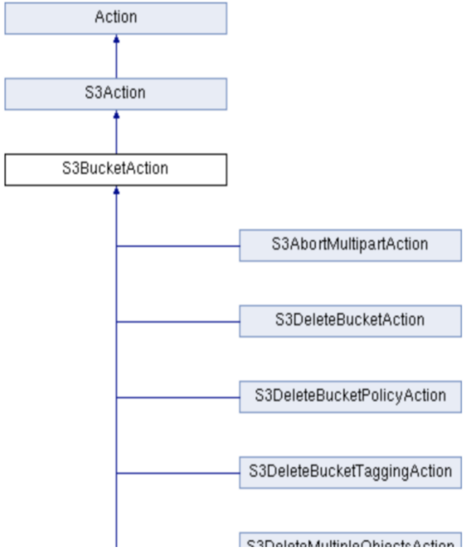


```
    case S3HttpVerb::GET:
      request->set_action_str("GetObject");
      action = std::make_shared<S3GetObjectAction>(request);
      s3_stats_inc("get_object_request_count");
      break;
```



```
void S3GetObjectAction::setup_steps() {
  s3_log(S3_LOG_DEBUG, request_id, "Setting up the action\n");
  ACTION_TASK_ADD(S3GetObjectAction::validate_object_info, this);
  ACTION_TASK_ADD(S3GetObjectAction::check_full_or_range_object_read, this);
  ACTION_TASK_ADD(S3GetObjectAction::read_object, this);
  ACTION_TASK_ADD(S3GetObjectAction::send_response_to_s3_client, this);
  // ...
}
```




Action
S3Action
S3BucketAction
S3AbortMultipartAction
S3DeleteBucketAction
S3DeleteBucketPolicyAction
S3DeleteBucketTaggingAction
S3DeleteMultipleObjectsAction

Supported S3 API

- S3 API (https://github.com/Seagate/cortx-s3server)

S3 Auth Server

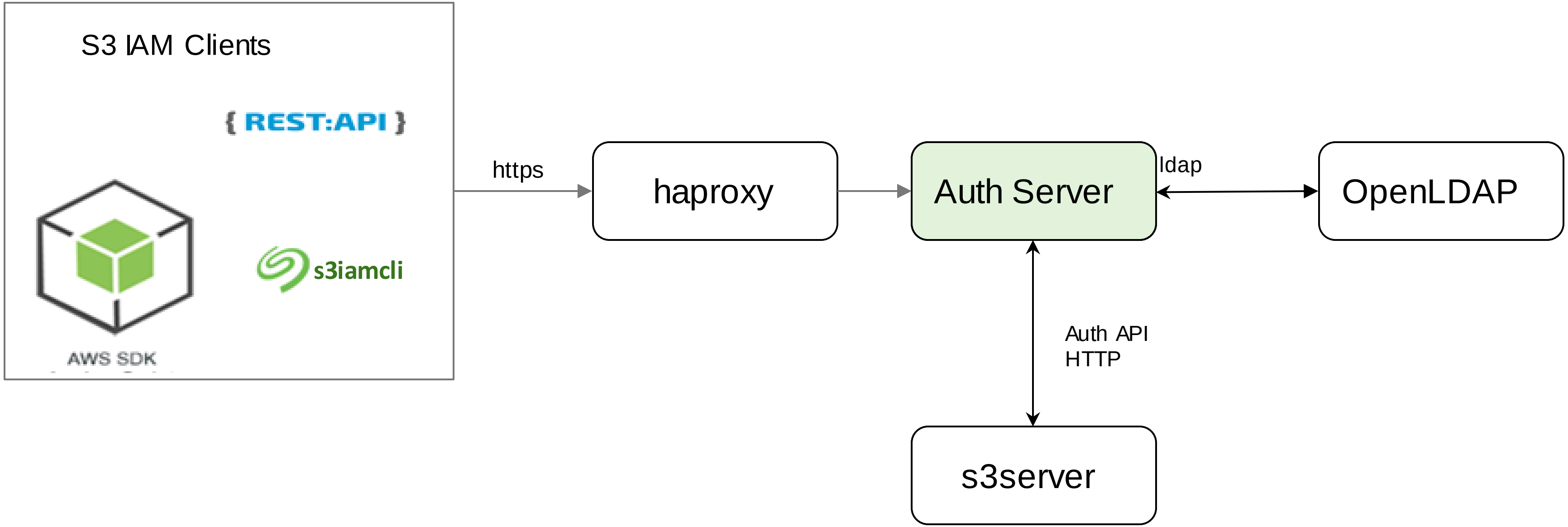
Authentication : “Verify Identity of a User”

Authorization : “Permission to perform operation”

S3 Auth Server Overview

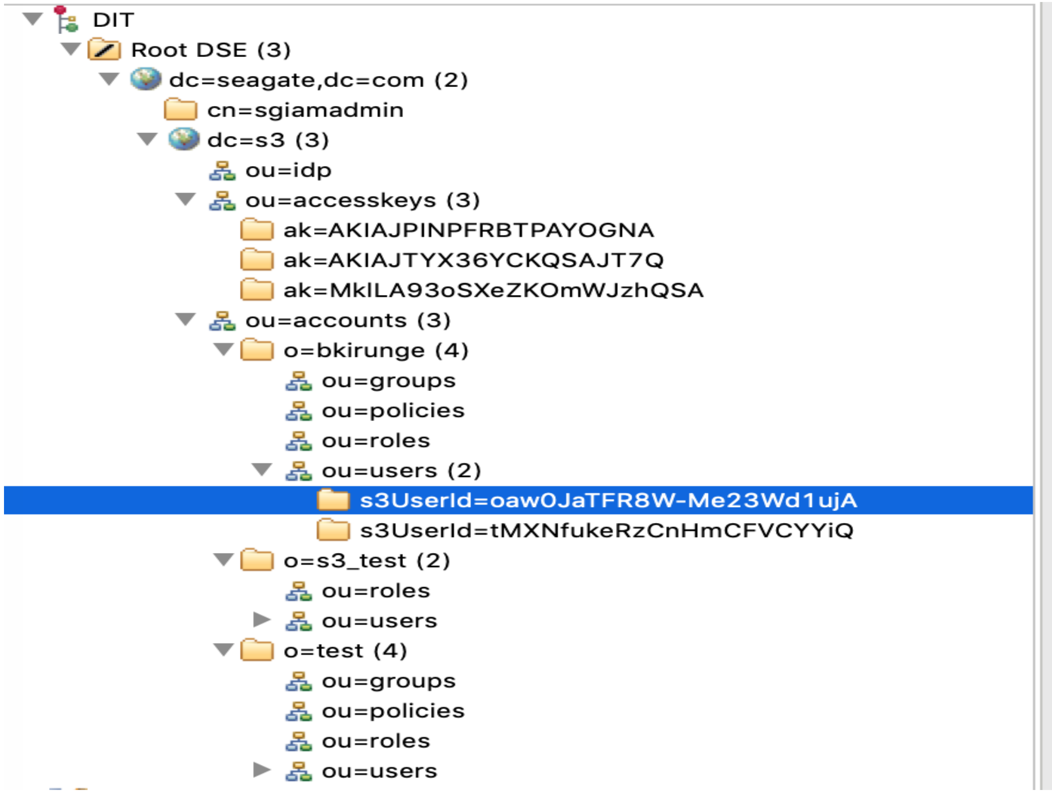
- Enable access control to resources (Bucket, Objects).
- Developed in Java using Java netty io framework.
- One instance of Auth server runs on each node.
- Uses OpenLDAP as backed to store IAM data.
- OpenLDAP is clustered, IAM data gets replicated across nodes.
- Provides REST API interface, AWS IAM API.
- Every S3 request generates authentication and authorization request.

S3 Auth Server



S3 IAM endpoint : *iam.seagate.com:9443*

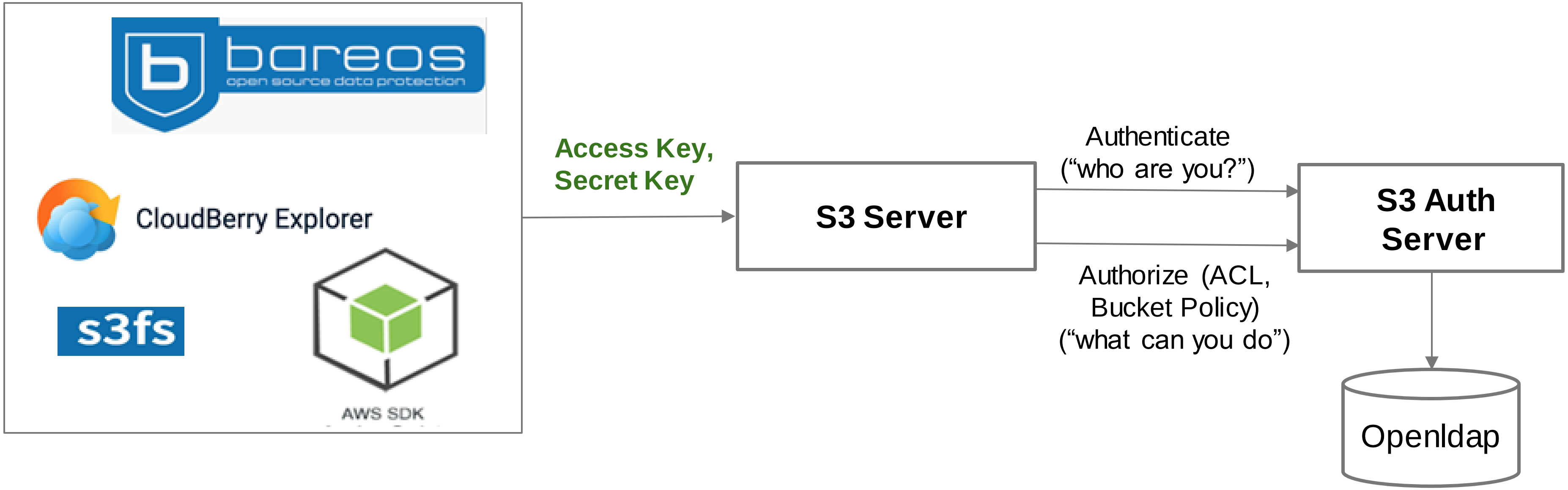
S3 User Schema (DIT in OpenLDAP)



```
▼ DIT
  ▼ Root DSE (3)
    ▼ dc=seagate,dc=com (2)
        cn=sgiamadmin
      ▼ dc=s3 (3)
          ou=idp
        ▼ ou=accesskeys (3)
            ak=AKIAJPINPFRBTPAYOGNA
            ak=AKIAJTYX36YCKQSAJT7Q
            ak=MklLA93oSXeZKOmWJzhQSA
        ▼ ou=accounts (3)
          ▼ o=bkirunge (4)
              ou=groups
              ou=policies
              ou=roles
            ▼ ou=users (2)
                s3UserId=oaw0JaTFR8W-Me23Wd1ujA
                s3UserId=tMXNfukeRzCnHmCFVCYYiQ
          ▼ o=s3_test (2)
              ou=roles
            ▶ ou=users
          ▼ o=test (4)
              ou=groups
              ou=policies
              ou=roles
            ▶ ou=users
```

S3 Authentication and Authorization

S3 Clients



Access Key, Secret Key

S3 Server

Authenticate ("who are you?")

Authorize (ACL, Bucket Policy) ("what can you do")

S3 Auth Server

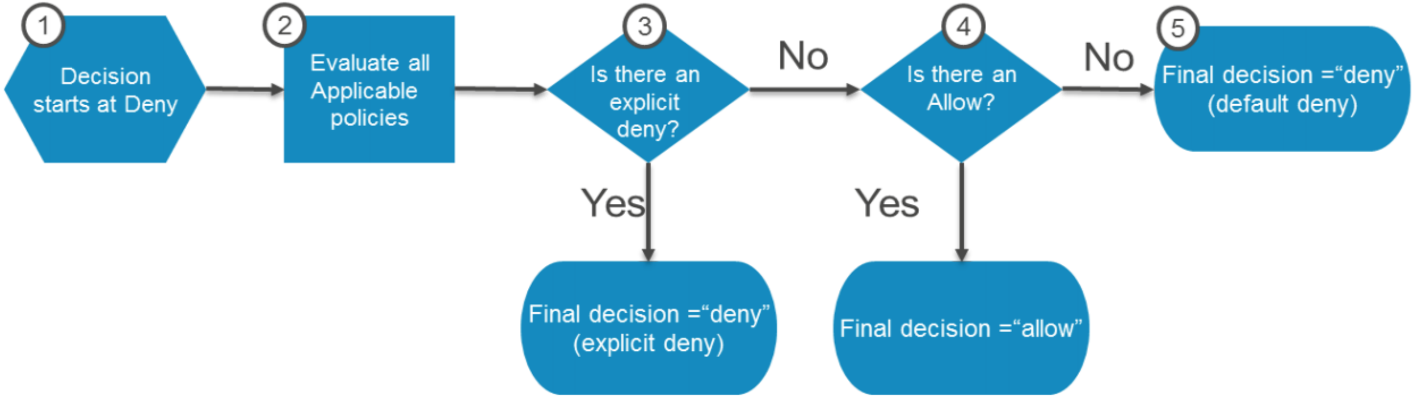
Openldap

Support AWS V2 and V4

Source/Footnote:

Authorization

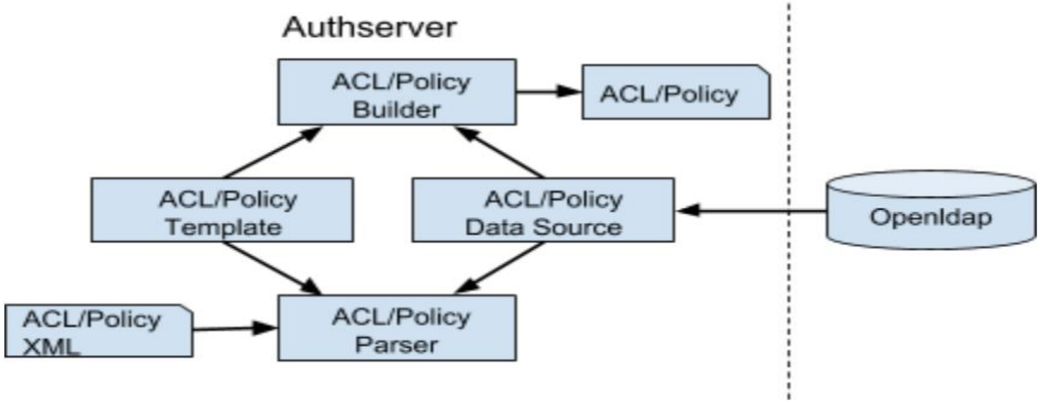
- Authorization depends on the union of all policies and S3 acls that apply.
- Decisions defaults to DENY and an explicit DENY always overrules an ALLOW.


1
Decision starts at Deny
2
Evaluate all Applicable policies
3
Is there an explicit deny?
No
4
Is there an Allow?
No
5
Final decision ="deny" (default deny)
Yes
Final decision ="deny" (explicit deny)
Yes
Final decision ="allow"

Authorization

S3 Account - S3 ACL

IAM User and Account - S3 Bucket Policy



S3 IAM API

Authetication and Authorization API's supported

AuthenticateUser : Autheticate API

AuthorizeUser : Authorize Request using ACL and Policy

ValidateACL : Validate given ACL

ValidatePolicy : Validate given Policy

IAM API’s supported

S3 Account : Create, Delete, List

S3 User : Create, Modify, Delete, List

S3 Access Key/Secret Key : Create, Delete, List

S3 Web login profile : Create, Update, Get Temp Access Key.

Questions ?

Slack : https://cortx.link/join-slack #cortx-s3

Post queries/Issues
https://github.com/Seagate/cortx-s3server

Thank You

SEAGATE