Spectrum Scale User Group

Object

24.02.2016 Oxford

Content provided by: Dean Hildebrand, Bill Owen, Brian Nelson, Sandeep Ramesh, Smita Raut, Deepak Ghuge, Simon Lorenz, Nilesh Bhosale, Joe Dain, and many more...

IBM

Object Topics

- **4.2.0.x and Road map**
- **Storage Policies**
  - Architecture
  - Container content -
    - compressed (4.2)
    - encrypted (4.2.1)
    - secure deleted (future)
    - expired (future)

      - via swift storage policies
- **Unified File and Object Access**
- **Multi-Region**
- **Swift3**
- **Object Meta Data search**
- **Implementation Guide with Spectrum Scale Object and Spectrum Archive (LTFS)**
- **Spectrum Scale facts**

4.2.0.x and Road Map

4.2.0.x and Road Map

Disclaimer:

IBM's statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM's sole discretion. Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision. The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract. The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon many factors, including considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results similar to those stated here.

4.2.0.x and Road Map

4.2.0.x Items:

- Enhanced swift storage policies to provide compression, multi-region and swiftonfile functionality
- Additional capabilities (SoF, Storage policy extensions, multi-region, …) and the needed services (including monitoring) can be enabled easily in config
- Unified File (POSIX, SMB, NFS) and Object (Swift, S3) access
- SoF Handle new empty containers via middleware dir_create
- Multi-region active-active object store with manual coordination between regions
- AFM-DR Support
- Running Keystone in Apache httpd server
- Redpaper Active Archive Implementation Guide with Spectrum Scale Object and Spectrum Archive (LTFS)

4.2.0.x and Road Map

Future:

- Spectrum Scale Encryption as a swift storage policy (4.2.1)
- Object meta data search (Paper describing setup in 4.2.1, integration into Spectrum Scale planned for 4.2.2)
- REST-API (GPFS Management) (Initial version with fileset management planned for 4.2.2)
- Spectrum Scale secure delete of containers linked to an encrypted swift storage policy (4.2.2)
- Billing and charge back data collection and interface (4.2.2+)
- Multiple Authentication (Keystone) domains (4.2.2)
- S3 improvements (Lifecycle policies, versioning) (4.2.2, depending on community progress)
- GPFS QoSIO integration for bkgrnd Swift tasks and between tenants (4.2.2)

Your feedback, what's important for you?

IBM

Storage Policies

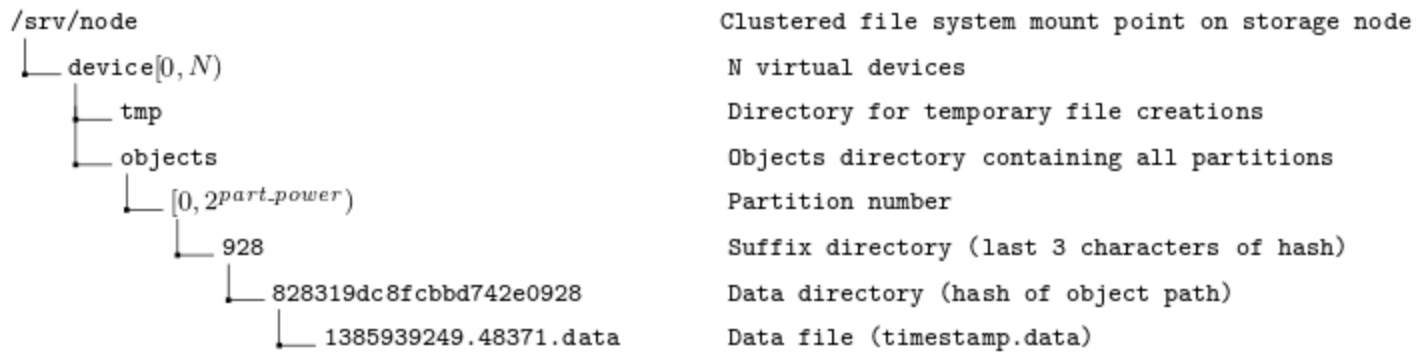
Architecture

Objects:

- compressed (4.2)
- encrypted (4.2.1)
- secure deleted (future)
- expired (future)

Storage Policies – Architecture
Object Setup on Spectrum Scale

Object Directory Structure:



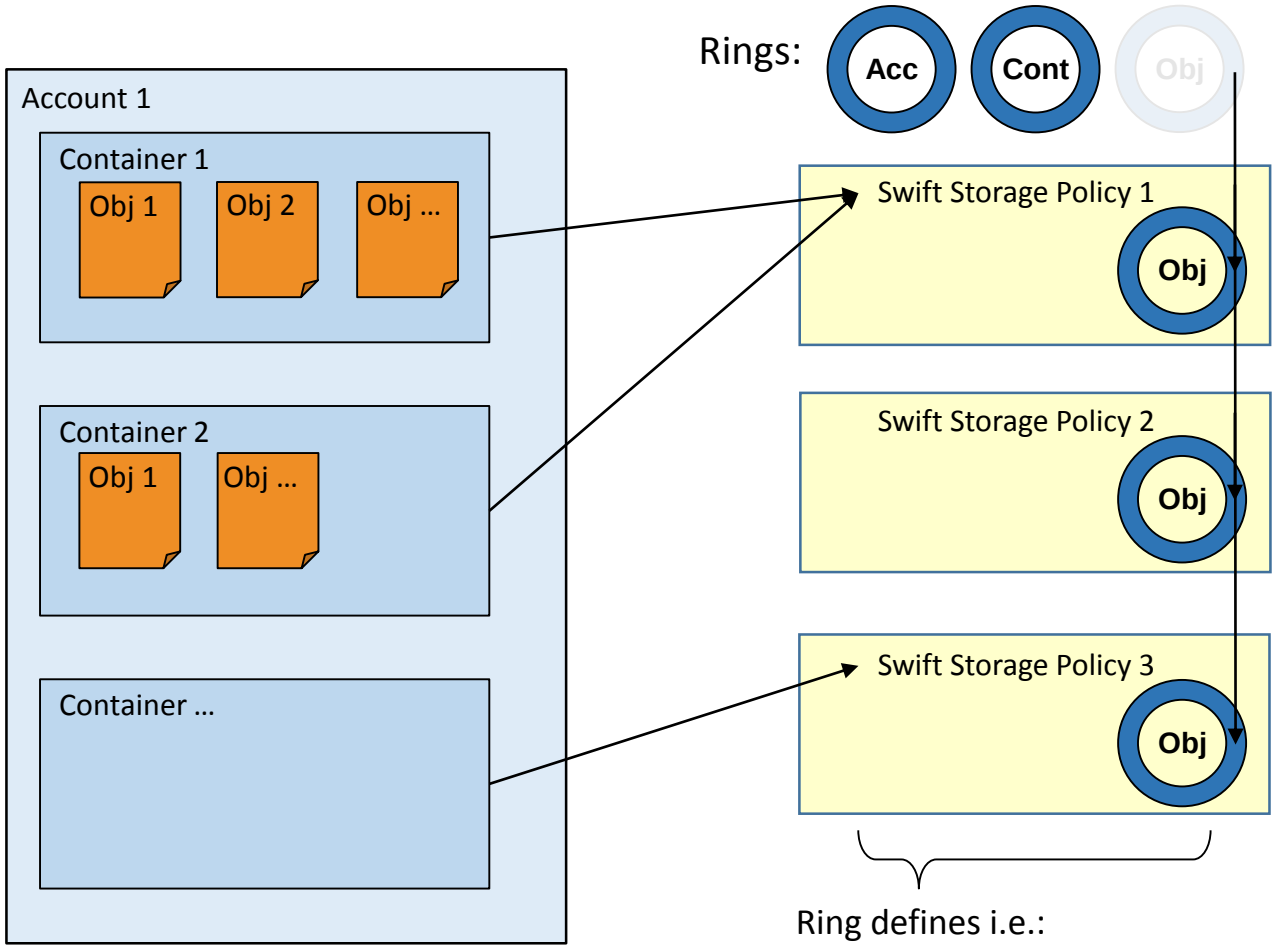
Base path („/srv/node“) is defined in object-server.conf and is valid for the **object server instance.**
Means: **What ever is stored in the object store, it will be placed under this path.**

We do use an independent fileset in the base path.

A fileset assigns a unique identifier to the entire object store, **allowing Information Lifecycle Management operations**, such as **snapshots, tiering, backup, and user policies**, to operate on the entire object store.

Devices for the rings are drives in community swift, but **in our case we create directories for it.**

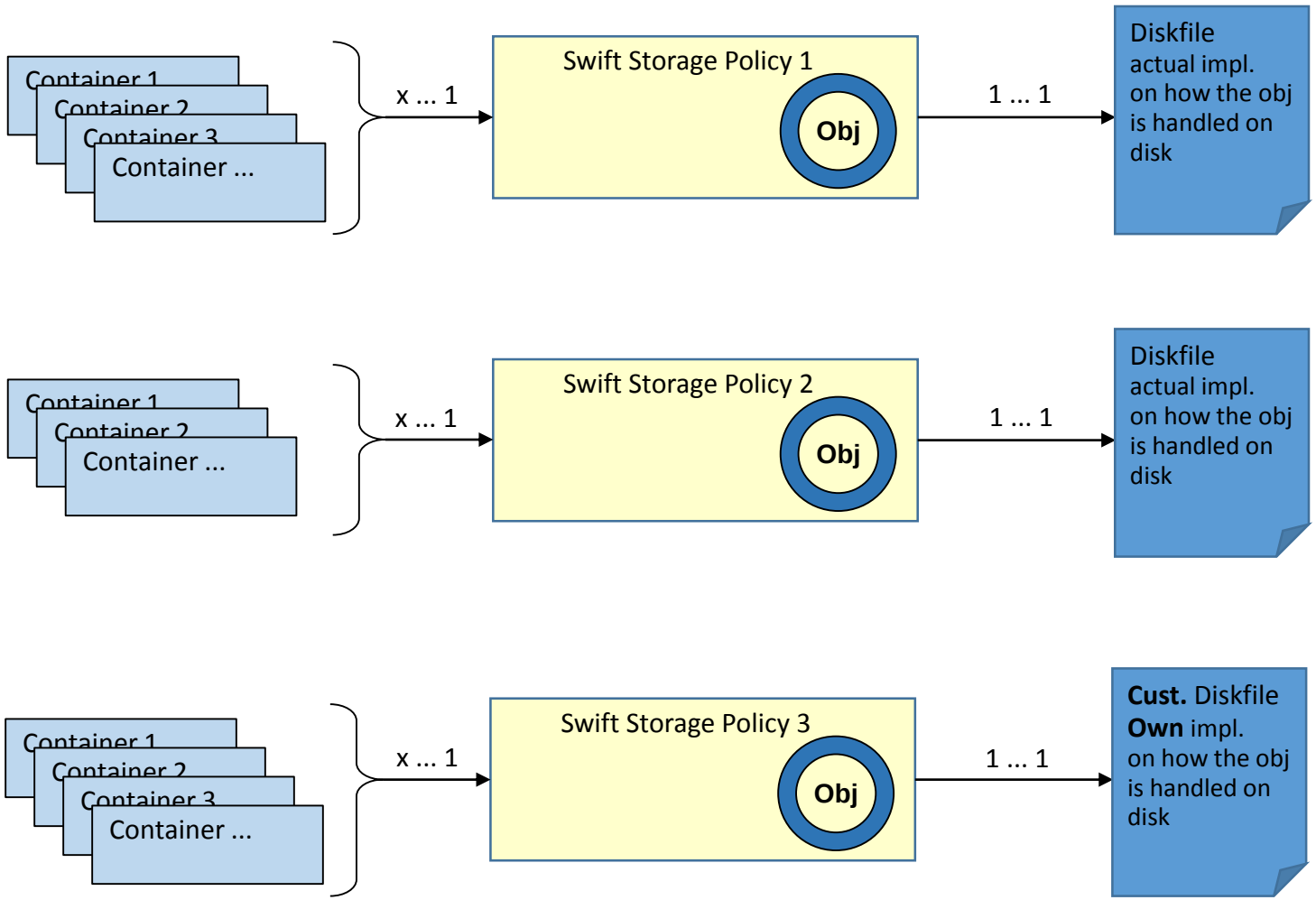
Storage Policies – Architecture
Swift Ring Files



Ring defines i.e.:

- How many replicas
- Where to store
- How to store

Storage Policies – Architecture
Swift Ring Files



Storage Policies – Architecture
Spectrum Scale Extensions

Every new storage policy will be linked to it‘s own independent Fileset.

This will allow us to run ILM features / storage policy

<base path>	<storage policy fileset>	<device folders>
<fs><fset>	<storage policy 1 fset>	<device 1> <device 2> <device …>
<fs><fset>	<storage policy 2 fset>	<device 1> <device 2> <device …>

How can we accomplish this? … as there is no Swift functionality like this

Storage Policies – Architecture
Spectrum Scale Extensions

Every new storage policy will be linked to it‘s own independent Fileset.

<base path> <storage policy fileset> <device folders>

<fs><fset> <storage policy 1 fset><device 1>
<device 2>
<device …>

<fs><fset> <storage policy 2 fset><device 1>
<device 2>
<device …>

Example:

/mnt/gpfs0/object_fileset /storagepolicy1 /device1/
/device2/
/device…/

/mnt/gpfs0/object_fileset /storagepolicy2 /device1/
/device2/
/device…/

How can we accomplish this? … as there is no Swift functionality like this

Storage Policies – Architecture
Spectrum Scale Extensions

Make the device folder a soft link to the fileset that is used for the storage policy

/mnt/gpfs0/object_fileset/sp2device1 links to:

/mnt/gpfs0/object_fileset/sp2/sp2device1

Example:

```
# pwd
/mnt/gpfs0/object_fileset
# ll
total 0
drwxr-xr-x 15 swift swift 4096 Jul  2 06:15 sp1
drwxr-xr-x 12 root  root  4096 Jul  2 07:28 sp2
lrwxrwxrwx  1 root  root    44 Jul  2 07:26 sp2device1 -> mnt/gpfs0/object_fileset/sp2/sp2device1
lrwxrwxrwx  1 root  root    44 Jul  2 07:26 sp2device2 -> /mnt/gpfs0/object_fileset/sp2/sp2device2
lrwxrwxrwx  1 root  root    44 Jul  2 07:26 sp2device3 -> /mnt/gpfs0/object_fileset/sp2/sp2device3
…
```

Storage Policies – Objects compressed 4.2

- Spectrum Scale Compression functionality is used

Example:

Create a compression storage policy as follows

```
# mmobj policy create CompressionTest --enable-compression --compression-schedule
"600:*:*:*"
```

The system displays output similar to the following:

```
[I] Getting latest configuration from ccr
[I] Creating fileset /dev/gpfs0:obj_CompressionTest
[I] Creating new unique index and building the object rings
[I] Updating the configuration
[I] Uploading the changed configuration
```

Every object stored using a storage policy that has compression enabled is compressed according to the specified schedule.
There is no need to uncompress an object in advance of a get request or any other object request. IBM Spectrum Scale™ automatically returns the uncompressed object.

Note: The download performance of objects in a compressed container is reduced compared to the download performance of objects in a non-compressed container.

Use mmlsattr –L <file> to display if it’s compressed.
A spectrum scale policy list run checking for attribute K can be used too.

Storage Policies – Objects encrypted 4.2.1

- Spectrum Scale Encryption functionality is used

Example:

Create an encrypted storage policy as follows

```
# mmobj policy create EncryptionTest --enable-encryption --encryption-keyfile
/tmp/key.file
```

The system displays output similar to the following:

```
[I] Getting latest configuration from ccr
[I] Creating fileset /dev/gpfs0:obj_EncryptionTest
[I] Creating new unique index and building the object rings
[I] Updating the configuration
[I] Uploading the changed configuration
```

Note: GPFS encryption is only available with IBM Spectrum Scale™ Advanced Edition.
The file system must be at the latest version for GPFS 4.1.
Secure storage uses encryption to make data unreadable to anyone who does not possess the necessary encryption keys. The data is encrypted while "at rest" (on disk) and is decrypted on the way to the reader. Only data, not metadata, is encrypted.
A Key Manager Server must be installed and configured before encryption functionality can be used.
The server that is supported is IBM® Security Key Lifecycle Manager (ISKLM) v2.5.0.1 or later.

Storage Policies – Objects secure deleted (future)

- Spectrum Scale Encryption functionality is used
- Secure deletion refers to both erasing files from the file system and erasing the MEKs (master encryption key) that wrapped the FEKs (file encryption key) that were used to encrypt the files.

  Basically the origin keys are exchanged with new keys to ensure, any file recovery will have no key to decrypt the file.
- We are looking at securely deleting the encrypted fileset via commands that ease the process.
  Means secure delete of multiple containers linked to an encrypted swift storage policy.

Storage Policies – Objects expired (future)

- Swift has expiration settings which can be set via X-Delete-At or X-Delete-After on object basis.
- We plan on doing expiration via storage policy, setting the policy to expire at and/or after.
- Taking care of expiration via Spectrum Scale policies

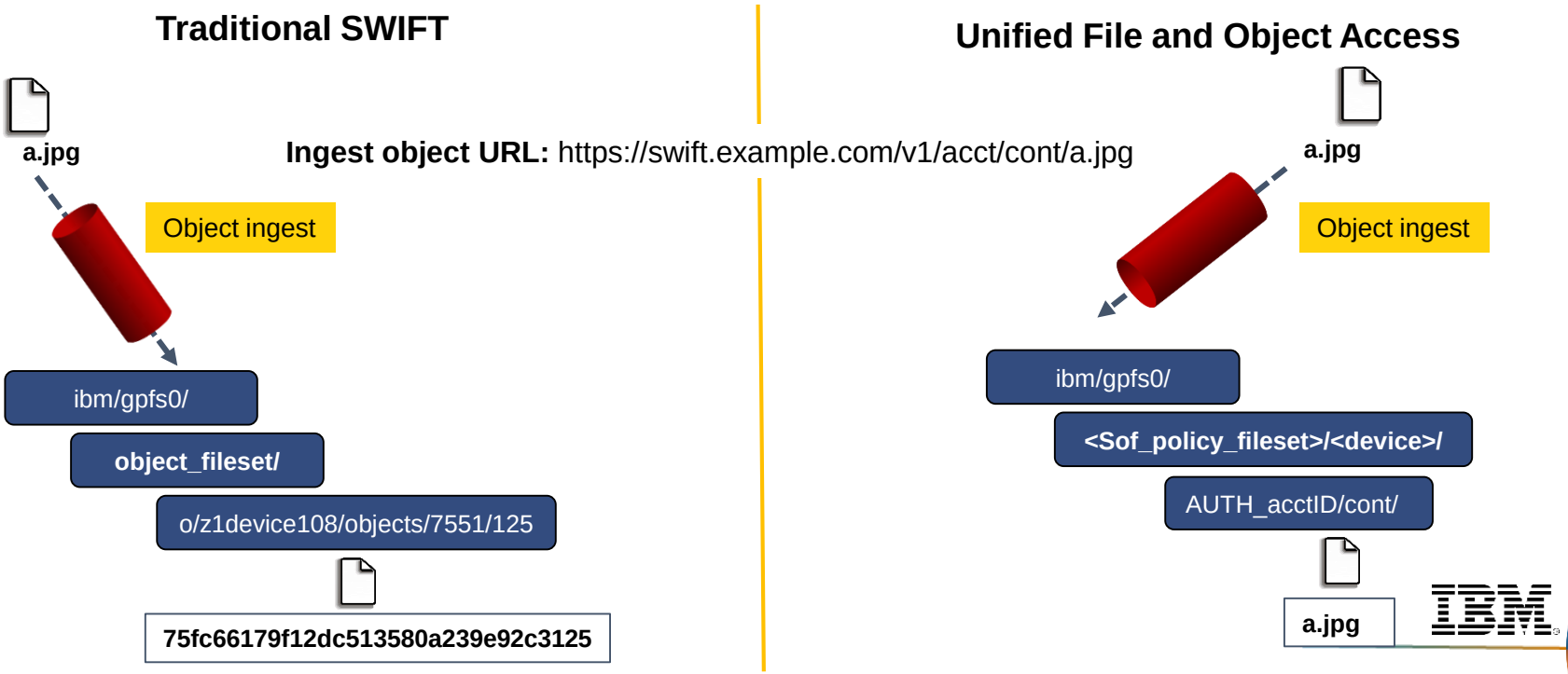
Unified File and Object Access

Unified File and Object Access

- Accessing object using file interfaces (SMB/NFS/POSIX) and accessing file using object interfaces (REST) helps legacy applications designed for file to seamlessly start integrating into the object world.
- It allows cloud data which is in form of objects to be accessed using files using application designed to process files.
- Multi protocol access for file and object in the same namespace allows supporting and hosting data oceans of different types with multiple access options.
- There is a rich set of placement policies for files (using mmapplypolicy) available with IBM Spectrum Scale™. With unified file and object access, those placement policies can be leveraged for object data.
- To analyse large amounts of data, advanced analytics systems are used. However, porting the data from an object store to a distributed file system that the analytics system requires is complex and time intensive. For these scenarios, there is a need to access the object data using file interface so that analytics systems can use it. Unified File and Object Access value adds in this scenario.
- Availability of spectrum Scale Hadoop Connectors over Unified File and Object access

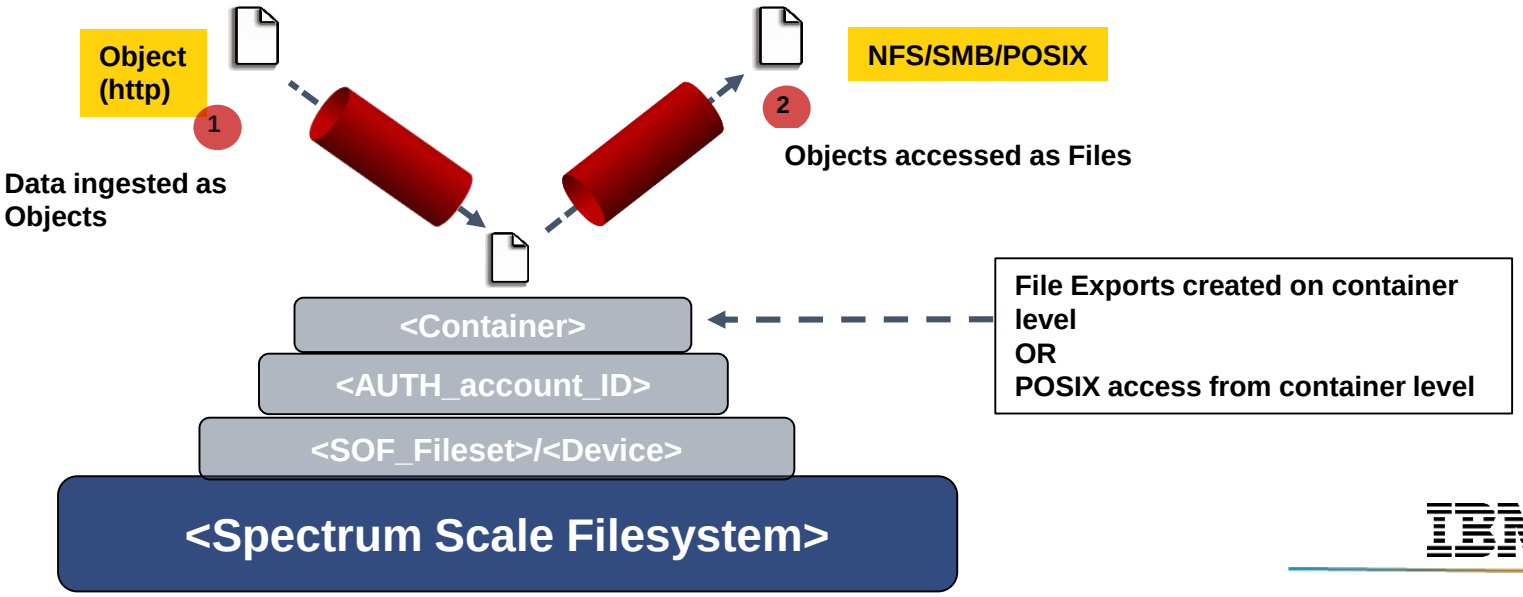
Filesystem Layout (Traditional Vs Unified File and Object Access)

- One of the key advantages of unified file and object access is the placement and naming of objects when stored on the file system. In unified file and object access stores objects following the same path hierarchy as the object's URL.
- In contrast, the default object implementation stores the object following the mapping given by the ring, and its final file path cannot be determined by the user easily.



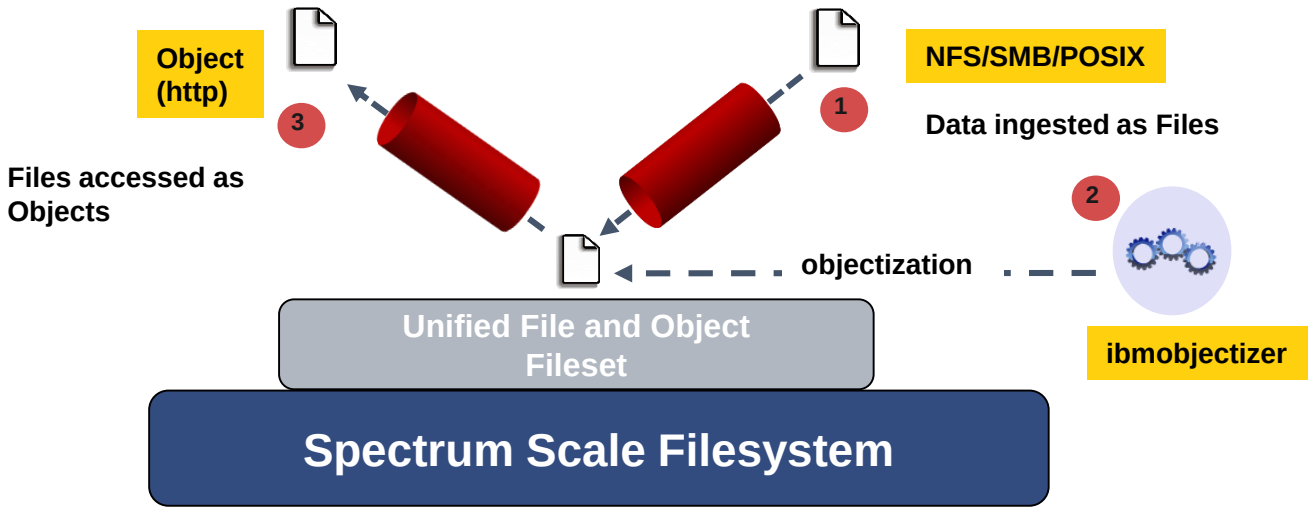
Easy Access Of Objects as Files via supported File Interfaces (NFS/SMB/POSIX)

- Objects ingested are available immediately for File access via the 3 supported file protocols.
- ID management modes (explained later) gives flexibility of assigning/retaining of owners, generally required by file protocols.
- Object authorization semantics are used during object access and file authorization semantics are used during object access of the same data – thus ensuring compatibility of object and file applications



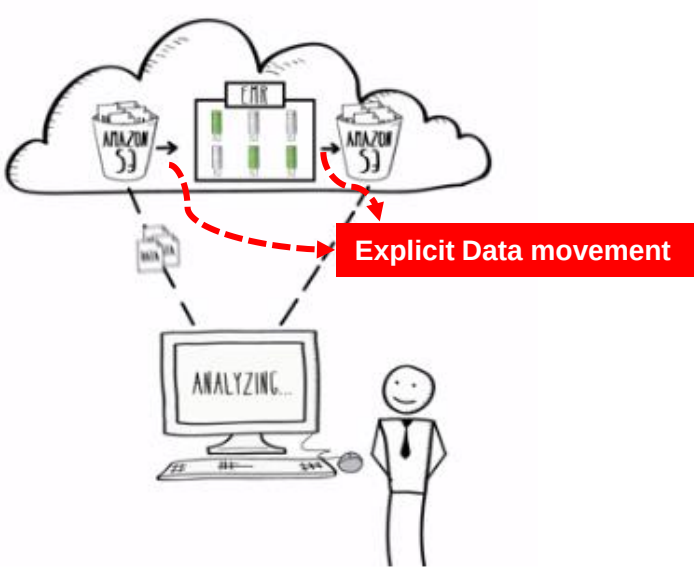
Objectization – Making Files as Objects (Accessing File via Object interface)

- Spectrum Scale 4.2 features with a system service called ibmobjectizer responsible for objectization.
- Objectization is a process that converts files ingested from the file interface on unified file and access enabled container path to be available from the object interface.
- When new files are added from the file interface, they need to be visible to the Swift database to show correct container listing and container or account statistics.



Use case – Enabling "In-Place" analytics for Object data repository

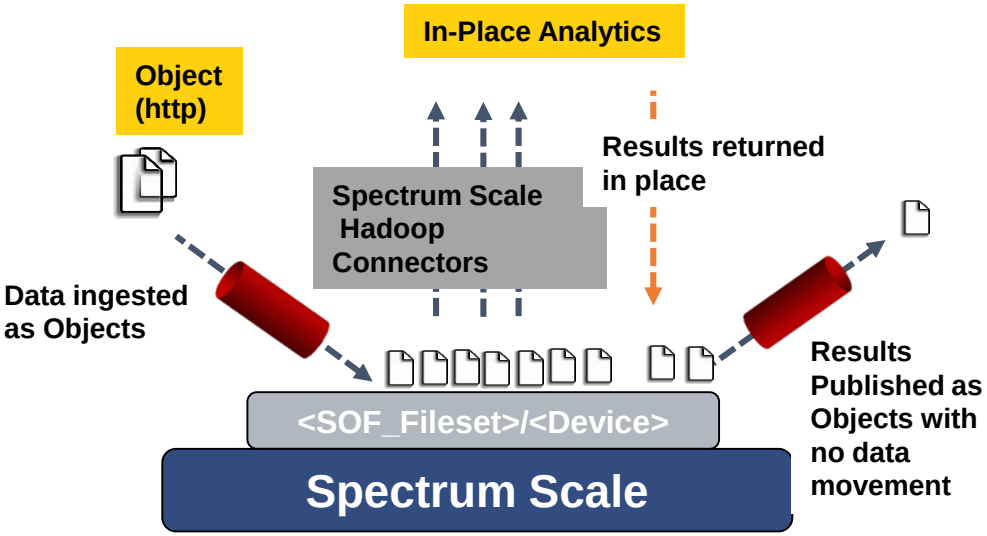
Analytics on Traditional Object Store



Traditional object store – **Data to be copied from object store to dedicated cluster** , do the analysis and **copy the result back to object store for publishing**

Source:https://aws.amazon.com/elasticmapreduce/

Analytics on Spectrum Scale Object Store With Unified File and Object Access



Spectrum Scale object store with Unified File and Object Access – Object Data available as File on the same fileset . Spectrum Scale Hadoop connectors allow the data to be directly leveraged for analytics.

No data movement / In-Place immediate data analytics.

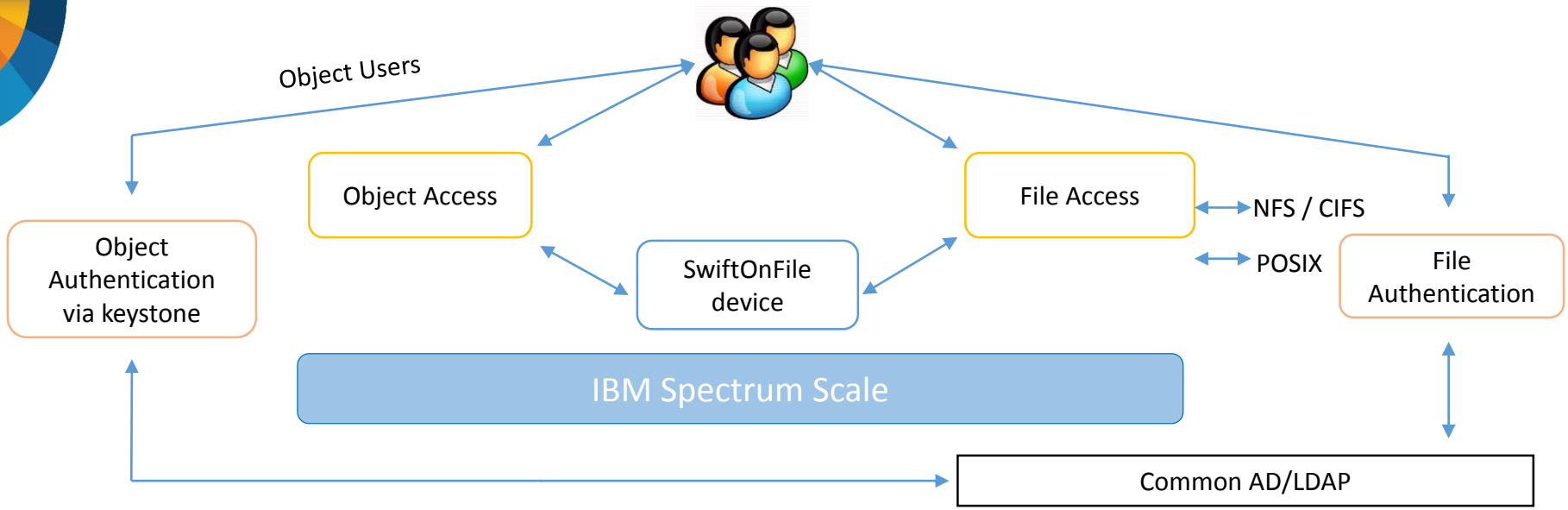
Policy Integration for Flexibility

This feature is specifically made available as an “object storage policy” as it gives the following advantages:

- Flexibility for administrator to manage unified file and object access separately
- Allows to coexists with traditional object and other policies
- Create multiple unified file and object access policies which can vary based on underlying storage
- Since policies are applicable per container , it gives end user the flexibility to create certain containers with Unified File and Object Access policy and certain without it.

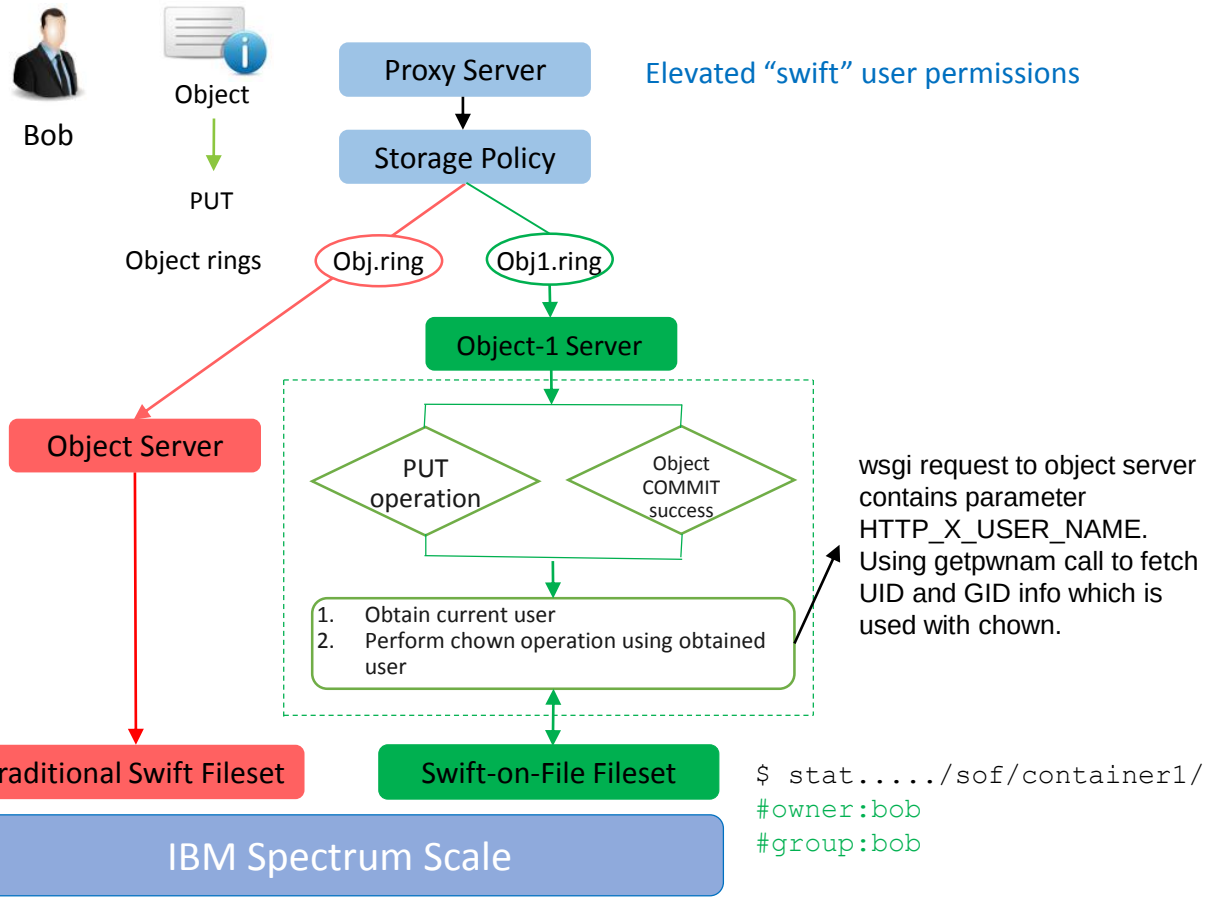
- Example: `mmobj policy create SwiftOnFileFS --enable-file-access`

Unified Identity Between Object and File



- Common set of Object and File users using same directory service (AD+RFC 2307 or LDAP)
- Objects created using Swift API owned by the user performing the Object operation (PUT)
  Note that if object already exists, existing ownership of object will be retained
- Retaining file ACL on PUT/POST
  If an object update is performed then existing “file ACL” will be retained
- For initial PUT operation of an object over a nested directory
  Object does not set ACLs on nested directories

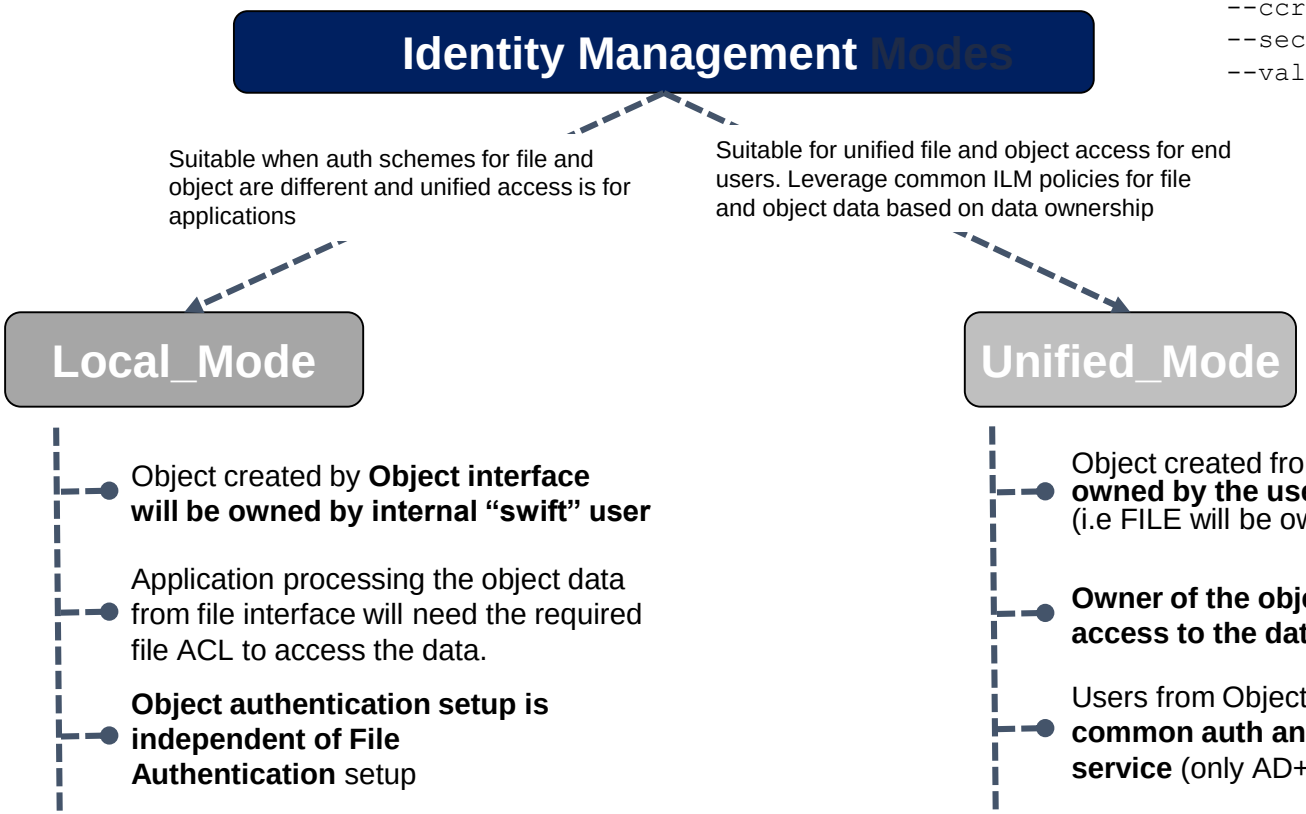
Accessing Objects via File WITH Ownership Architecture



Flexible Identity Management Modes

- Support's Two Identity Management Modes
- Administrators can choose based on their need and use-case using CLI

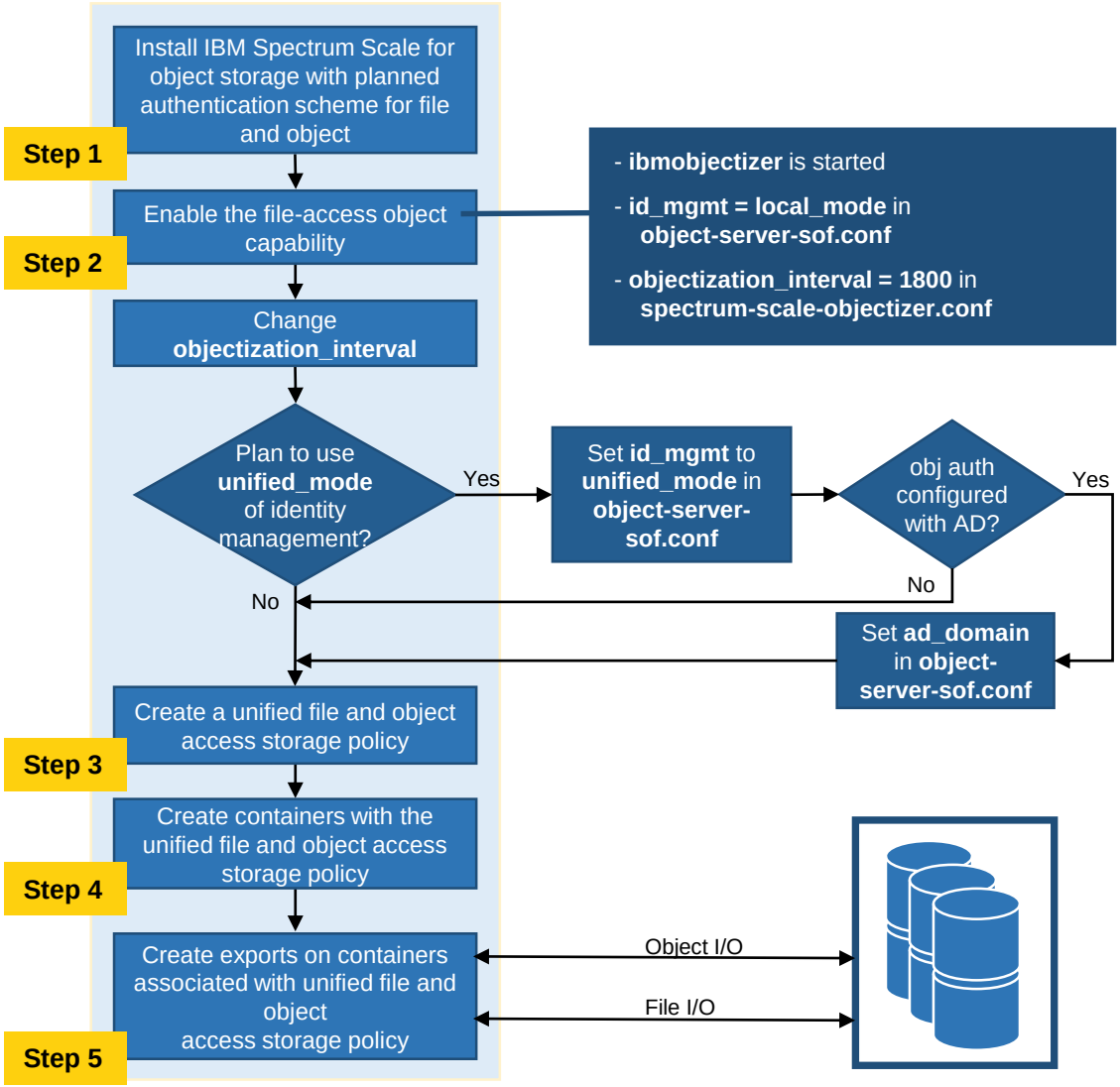
```
#mmobj config change
--ccrfile object-server-sof.conf
--section DEFAULT --property id_mgmt
--value unified_mode | local_mode
```



Key Components

Component & CLI	Associated Configuration File	Remark
Unified File and Object Access Object server and Diskfile	object-server-sof.conf	Object server for unified file and object access is a separate process (/usr/bin/swift-object-server-sof) which runs on all the protocol nodes.
Objectizer Service	spectrum-scale-objectizer.conf & spectrum-scale-object.conf	“ibmobjectizer“ runs as a singleton service, on the singleton node. To identify the node on which the ibmobjectizer service is running, use the mmces service list --verbose command.
“dir_create” proxy middleware	proxy-server.conf	A proxy-server middleware used by unified file and object feature to create empty directories when empty container is created.
“sof_constraints” proxy middleware	proxy-server.conf	A proxy-server middleware used by unified file and object feature to detect potential filesystem directory/file creation failures at the proxy server level and fail the request with a "400 Bad Request" response.
“mmobj file-access” CLI	spectrum-scale-objectizer.conf & spectrum-scale-object.conf	Regular objectization CLI which allows objectization of a file to object more or less immediately. Uses common code from objectizer.
mmobj policy create <policy name> --enable-file-access	spectrum-scale-object.conf, swift.conf	Allows to create unified file and object policies. To read and understand on SWIFT object policy refer to : http://docs.openstack.org/developer/swift/overview_policies.html

Simple 5 Steps for Configuration And Usage



Unified File and Object Access Execution Example

1. Enabling the file-access object capability as follows.

```
# mmobj config change --ccrfile spectrum-scale-object.conf  --section capabilities --
property file-access-enabled --value true
```

2. [Optional – Based on Usecase and Workload] Set up the objectizer service interval as follows.

```
# mmobj config change --ccrfile spectrum-scale-objectizer.conf --section DEFAULT --
property objectization_interval --value 600
```

3. [Optional – Based on Usecase] Change the identity management mode to unified_mode as follows.

```
# mmobj config change --ccrfile object-server-sof.conf --section DEFAULT --property
id_mgmt --value unified_mode
```

4. Create a unified file and object access storage policy as follows.

```
# mmobj policy create SwiftOnFileFS --enable-file-access
```

The system displays output similar to the following:

```
[I] Getting latest configuration from ccr
[I] Creating fileset /dev/cesSharedRoot:obj_SwiftOnFileFS
[I] Creating new unique index and building the object rings
[I] Updating the configuration
[I] Uploading the changed configuration
```

This command also creates a unified file and object access enabled fileset which is shown in the command output. Make a note of that fileset (marked in blue above).

Unified File and Object Access Execution Example (cont. 1)

5. Create a base container with a unified file and object access storage policy as follows (assuming you have valid tokens)

```
# swift post unified_access -H "X-Storage-Policy: SwiftOnFileFS"
```

Note: This will create a container called “unified_access” resulting into an directory “unified_access” on the filesystem under the appropriate fileset associated with “SwiftOnFileFS” storage policy.

6. Store the path created for the container by finding it in the newly created fileset as follows.

```
# export FILE_EXPORT_PATH=`find /ibm/cesSharedRoot/obj_SwiftOnFileFS/ -name
"unified_access"`

# echo $FILE_EXPORT_PATH

/ibm/cesSharedRoot/obj_SwiftOnFileFS/s18401510110z1device1/AUTH_c653056149d34f46bdfe
5b74f9fa2c07/unified_access
```

Note: It is highly recommended to create the File exports on the container level and not above it, as shown above.

7. Create an SMB export on the path as follows.

```
# mmsmb export add unified_access $FILE_EXPORT_PATH

The system displays output similar to the following:

mmsmb export add: The SMB export was created successfully
```

8. Create an NFS export on the path.

```
# mmnfs export add $FILE_EXPORT_PATH --client
"*(Access_Type=RW,Squash=no_root_squash,SecType=sys)"
```

Unified File and Object Access Execution Example (cont. 2)

9. Check the NFS and SMB exports.

```
# mmnfs export list
Path                                    Delegations Clients
-----------------------------------------------------------------------------------------------
-----------------------------------------
/ibm/cesSharedRoot/obj_SwiftOnFileFS/s18401510110z1device1/AUTH_c653056149d34f46bdfe5b74f9fa
2c07/unified_access none *

# mmsmb export list
export     path                                         guest ok smb encrypt
unified_access
/ibm/cesSharedRoot/obj_SwiftOnFileFS/s18401510110z1device1/AUTH_c653056149d34f46bdfe5b74f9fa
2c07/unified_access no        auto
Information:
The following options are not displayed because they do not contain a value:
"browseable"
```

10. Create a File from NFS client where you have mounted the export as follows.

```
# touch /mnt/mounted_export_from_spectrum_scale/samplefile.txt
```

11. Objectize that file immediately by using the following command or wait for the objectization cycle to

Complete so that it can be accessed from Object Interface.

```
# mmobj file-access --object-path \
/ibm/cesSharedRoot/obj_SwiftOnFileFS/s18401510110z1device1/AUTH_c653056149d34f46
bdfe5b74f9fa2c07/unified_access samplefile.txt
```

Unified File and Object Access Execution Example (cont. 3)

12. Download that object using the Swift client which is configured with required env variables.

```
# swift download container1/samplefile.txt
```

13. List the contents of the container using the Swift client which is configured with all variables as follows.

```
# swift list container1
```

Unified File and Object Access – Authentication support matrix 4.2.0.1

Authentication method	ID mapping method	SMB	SMB with Kerberos	NFSV3	NFSV3 with Kerberos	NFSV4	NFSV4 with Kerberos	Object
User-defined	User-defined	NA	NA	NA	NA	NA	NA	NA
LDAP with TLS	LDAP	✓	NA	✓	NA	✓	NA	✓
LDAP with Kerberos	LDAP	✓	✓	✓	✓	✓	✓	NA
LDAP with Kerberos and TLS	LDAP	✓	✓	✓	✓	✓	✓	NA
LDAP without TLS and without Kerberos	LDAP	✓	NA	✓	NA	✓	NA	✓
AD	Automatic	✓	✓	X	X	X	X	✓
AD	RFC2307	✓	✓	✓	✓	✓	✓	✓
AD	LDAP	✓	✓	✓	X	X	X	✓
NIS	NIS	NA	NA	✓	NA	✓	NA	NA
Local	None	NA	NA	NA	NA	NA	NA	✓

Multi-Region

Multi-Region
Active-Active Multi-Site Storage Cloud

Global Distribution

Ingest and Access from Any Data Center

Multi-Site Availability

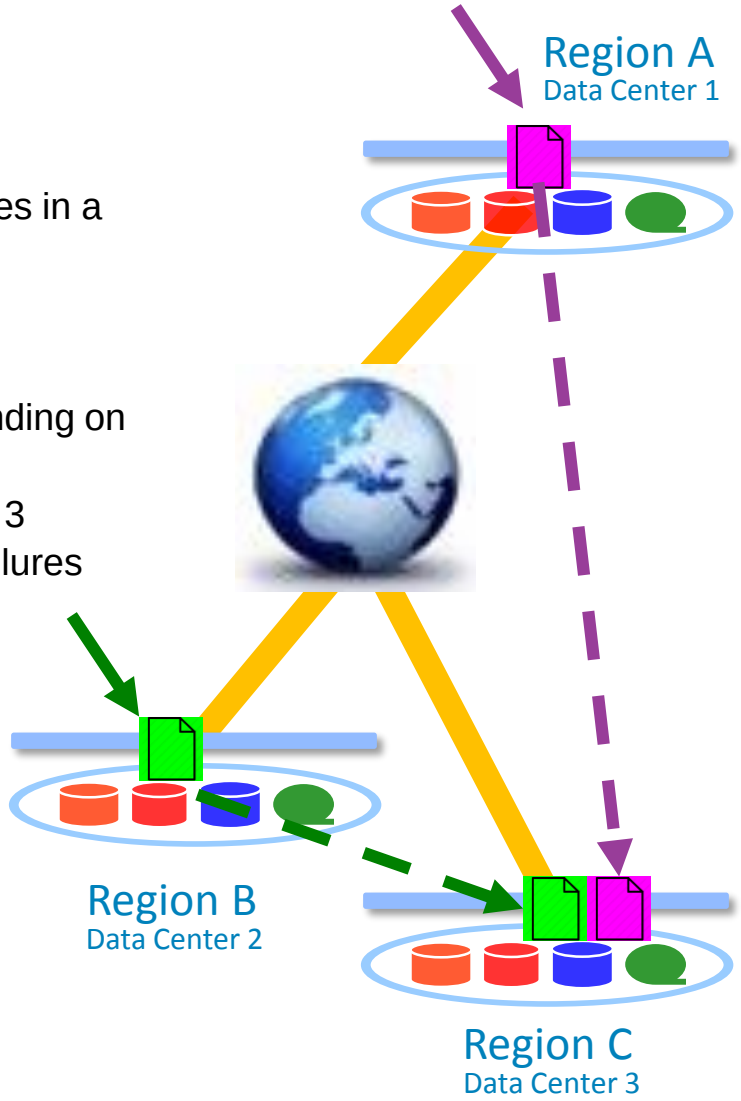
Objects Replicated Across 2 or more Sites

Flexible

Async or Sync Replication

Multi-Region Architecture Details

- Provides Disaster Recovery of data center failures in a Active-Active storage cloud
- Binds separate Spectrum Scale clusters into a practically limitless capacity storage cloud
- Objects are stored in one or more regions depending on
  - Required performance
    - Number of data copies can be 1, 2, or 3
  - Required number of supported data center failures
    - Currently tested limit is up to 3 sites
- Objects accessible from ANY site
  - If object not local, system retrieves it from remote region
- Supports asynchronous or synchronous replication
- Always returns latest copy across all sites
- Working on supporting WAN-acceleration technologies for replication such as Aspera or TransferSoft



Note: that this feature leverages Swift replication and is currently only supported for Object data-access

Multi-Region Execution Example

- To set up an initial multi-region environment, issue the following command on the 1st cluster after it has been installed:

```
# mmobj multiregion enable
```

- Use the following steps to add a region in a multi-region object deployment environment.
  In the command examples in the following steps, europe is the 1st region and asia is the 2nd region.

1. Export the 1st region's information to a file using the mmobj multiregion export command.

```
[europe]# mmobj multiregion export --region-file /tmp/multiregion_europe.dat
```

2. Copy that file manually to the 2nd region.

```
[europe]# scp /tmp/multiregion_europe.dat asia:/tmp
```

3. From the 2nd region, join the multi-region environment as follows:
Use the file generated in the 1st region while deploying object on the 2nd region using the `mmobj swift base` command.

```
[asia]# mmobj swift base ... --join-region-file /tmp/multiregion_europe.dat \
                --region-number 2 --configure-remote-keystone
```

This step installs the object protocol in the 2nd region and joins the 1st region. Additional devices are added to the primary ring files for this region.

4. Export the 2nd region's ring file data.

```
[asia]# mmobj multiregion export --region-file /tmp/multiregion_asia.dat
```


Multi-Region Execution Example (cont. 1)

5. Copy that file manually to the 1st region.

```
[europe]# mmobj multiregion export --region-file /tmp/multiregion_europe.dat
```

6. In the 1st region, update local ring files with 2nd region's configuration.

```
[europe]# mmobj multiregion import --region-file /tmp/multiregion_asia.dat
```

This step reads in the ring files which are updated with 2nd region's information. This update ensures that the 2nd region's data contains a new region and therefore replaces the associated ring files in the 1st region with the ones from the 2nd region.

Note:

Now the two clusters have been synced together and can be used as a multi-region cluster. Objects can be uploaded and downloaded from either region. If the installation of the 2nd region specified the --configure-remote-keystone flag, a region-specific endpoint for the object-store service for the 2nd region is created in Keystone.

The regions need to be synced in the future any time region-related information changes. This includes changes in the set of CES IP addresses (added or removed) or if storage policies were created or deleted within a region. Changes that affect the swift.conf file or ring files need to be synced to all regions. For example, adding additional CES addresses to a region causes the ring files to be rebuilt.

7. In the 2nd region, add CES addresses and update other clusters.

```
[asia]# mmces address add --ces-ip asia9
```

This steps adds an address to the CES IP pool. This also triggers a ring rebuild which changes the IP-to-device mapping in the ring files.

Multi-Region
Execution Example (cont. 2)

8. Export the ring data so the other clusters in the region can be updated with the new IPs from the 2nd region.

```
[asia]# mmobj multiregion export --region-file /tmp/multiregion_asia.dat
```

9. Copy that file manually to the 1st region.

```
[asia]# scp /tmp/multiregion_asia.dat europe:/tmp
```

10. In the 1st region, update with changes for new 2nd region address in the ring.

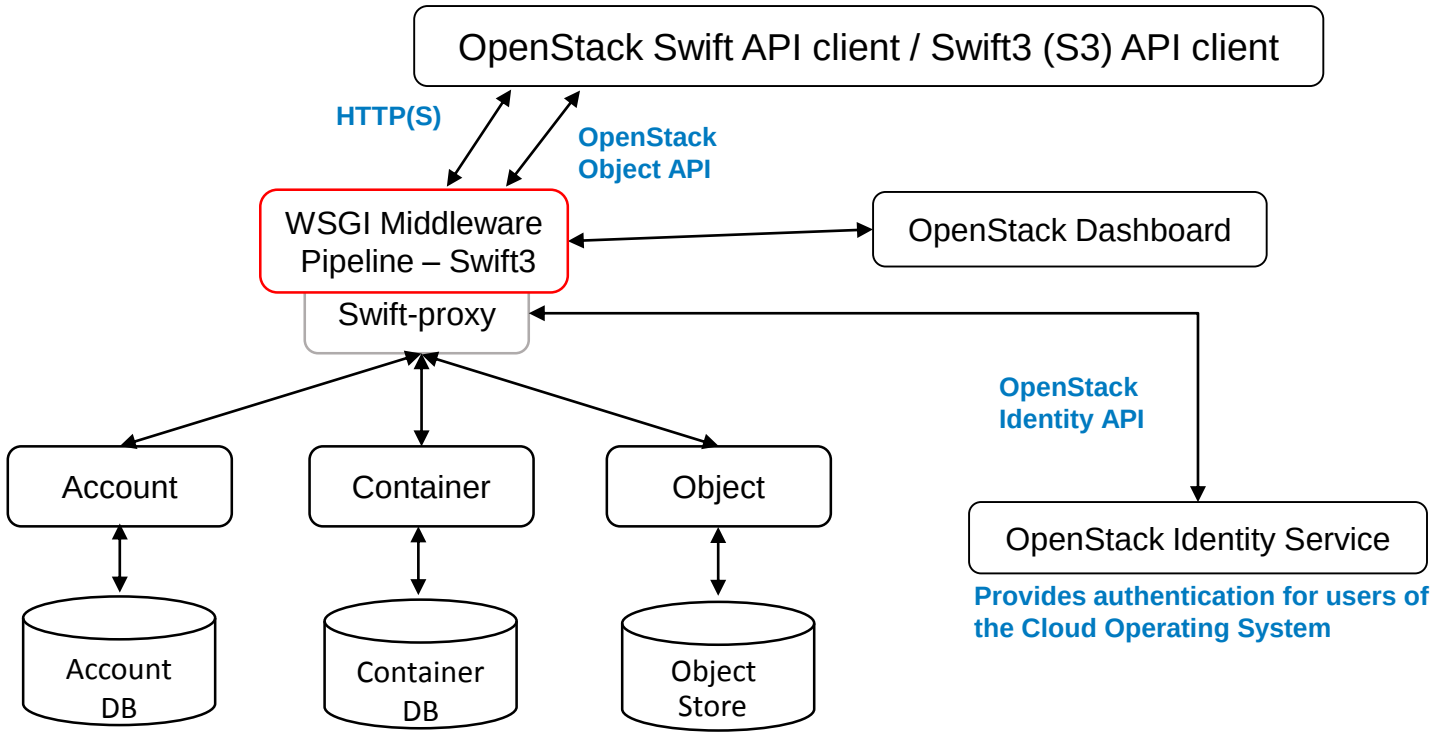
```
[europe]# mmobj multiregion import --region-file /tmp/multiregion_asia.dat
```

This step imports the changes from the 2nd region. When this is complete, a checksum is displayed which can be used to determine when regions are synced together. By comparing it to the one printed when the region data was exported, you can determine that the regions are synced when they match. In some cases, the checksums do not match after import. This is typically due to some local configuration changes on this cluster which are not yet synced to the other regions. If the checksums do not match, then this region's configuration needs to be exported and imported into the other region to sync them.

Swift3

Swift3

- S3 emulation via Swift3 Proxy middleware



Swift3
Validating the API

- Running capability tests

ceph-s3 tests: Open source compatibility tests for S3 clones

- Approximately 350 tests
- Swift3 v1.9 passes approx 75% of tests

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
regression	353	272	81	01:03:17	
s3	353	272	81	01:03:17	
s3_acl	59	48	11	00:08:20	
s3_auth	23	17	6	00:02:35	
s3_bucket	140	124	16	00:18:05	
s3_cors	2	0	2	00:00:15	
s3_metadata	16	8	8	00:01:59	
s3_multipart	11	11	0	00:20:06	
s3_multiregion	4	4	0	00:00:19	
s3_object	188	124	64	00:45:39	
s3_policy	3	0	3	00:00:22	
s3_versioning	15	0	15	00:01:48	

- Working with Community on Livecycle policies, versioning

Swift3 Experience? What's missing from your view?

Object Meta Data search

Object Meta Data search Why Valuable?

- Find needles in unstructured haystacks
- Help users and administrators perform Data Analytics
- Metadata can be on highest tier (SSD) while data resides on lower tier (Disk/Tape)

General Use Cases

- Data Mining
- Data Warehousing
- Selective data retrieval, data backup, data archival, data migration
- Management/Reporting

Object Meta Data search Example

Consider a Photo sharing application named 'MyPhotoSpace', storing the data in an object store in the backend, where an user has uploaded photos and added tags such as :

name: Times Square
city: New York
country: USA
region: America
time: night

name: China Wall
city: Beijing
country: China
region: Asia
time: day

name: London Bridge
city: London
country: UK
region: Europe
time: night

name: Las Vegas
city: Las Vegas
country: USA
region: America
time: night

name: Taj Mahal
city: Agra
country: India
region: Asia
time: day

name: Venice
city: Venice
country: Italy
region: Europe
time: night

name: Statue of Liberty
city: New York
country: USA
region: Americ
time: day

name: Tokyo Tower
city: Tokyo
country: Japan
region: Asia
time: night

Now, with metadata search, the user can search his album for various purposes:

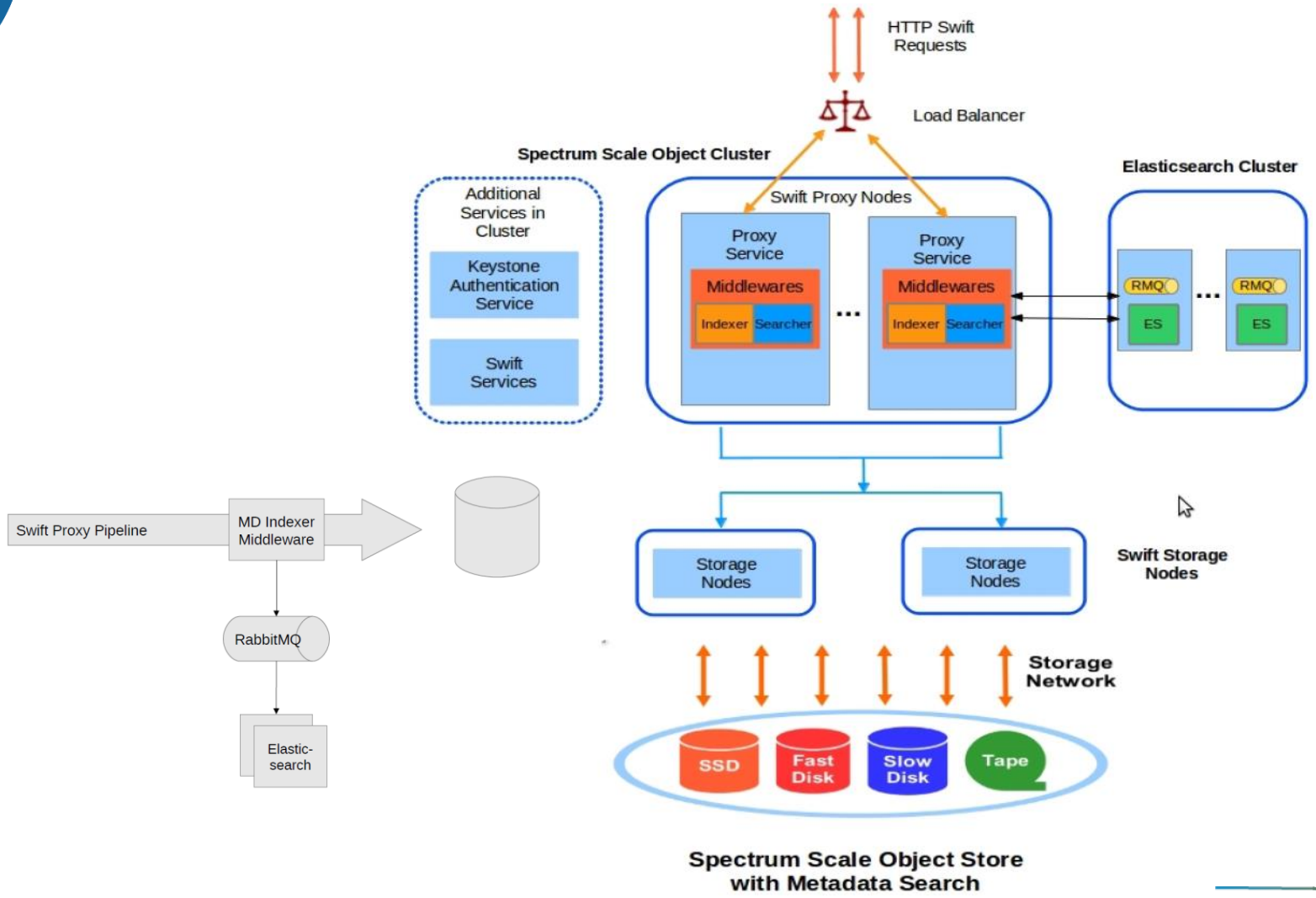
Case 1: GET /MyPhotoSpace?query=time=day

Case 2: GET /MyPhotoSpace?query=country='USA' AND time='night'

Object Meta Data search
How it is implemented ?

- OpenStack **Swift Middleware**
  - **Indexer middleware** – intercepts the object create/update/delete requests, updates metadata index
  - **Search middleware** – intercepts the object retrieval (GET) requests, returns objects matching the search criteria
- Uses open source **RabbitMQ** for async processing of indexing requests
- Uses open source **Elastic Search** engine (NoSQL DB) for indexing
  - Can support other NoSQL systems as well
- **Complex searches** with multiple criteria possible
- Support for metadata **data type mappings**

Object Meta Data search
How it is implemented ?



Object Meta Data search
Metadata Search API Syntax Details (future)

- HTTP Method: GET
- URI: /v1/<account>[/<container>[/<object>]] ?

```
[&query=<query expr1>[%20AND%20<query expr2>][%20AND%20…]]
[&format=json|xml|plain]
[&type=container|object]
[&sort=<query attr> asc|desc [,<query attr> asc|desc]* ]
[&start=<int>]
[&limit=<int>]
[&recursive= True | False]
[&sys.container=<container_name>]
[&sys.object=<object_name>]
[&sys.name=<object_name|container_name>]
[&sys.content_type=<content_type>]
[&sys.last_modified[=|>|<|>=|<=]<last_modified_date>]
```

- Headers:
  - X-Context : search
  - X-Auth-Token: <valid-authentication-token>

IBM

Implementation Guide with Spectrum Scale Object and Spectrum Archive (LTFS)

Implementation Guide with Spectrum Scale Object and Spectrum Archive (LTFS)

Storing objects in the Spectrum Archive tape tier

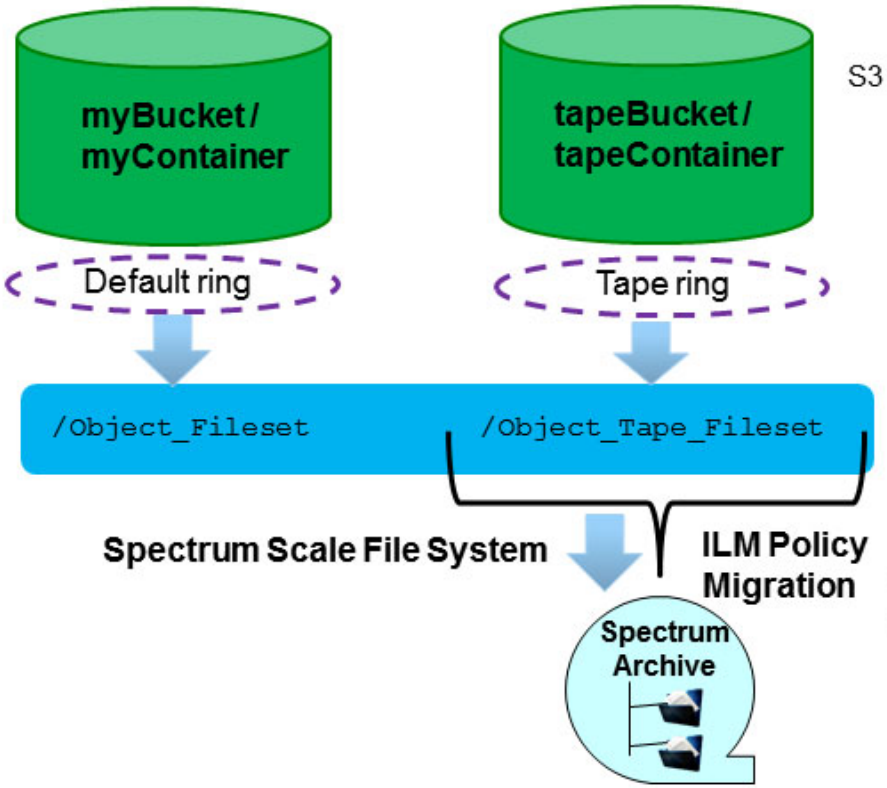
2 main methods to leverage the Spectrum Archive tape tier in the Spectrum Scale object store:

- 1. Application or end user explicitly copy data to specific S3 buckets and containers where the S3 buckets and Swift containers leverage Spectrum Scale object storage policies that immediately migrate the entire content of the buckets and containers to tape.
  This may be useful in creating a global archive for an entire data center where the Spectrum Scale and Spectrum Archive object store is primarily targeted towards cold data that will be immediately put on tape. The data put into the S3 buckets and Swift containers is not actively accessed and is not treated as an active resource.

  The main advantage of this method is that fine grain control of the migration of data on S3 bucket and Swift container boundaries is provided.
  The main trade-off is that data must be explicitly copied into the S3 buckets and Swift containers that immediately move the content to tape, which requires end user or application awareness of the tape tier.

Storing objects in the Spectrum Archive tape tier

Archive S3 bucket/Swift container – Any object placed into the S3 tapeBucket/Swift container is migrated to Spectrum Archive physical tape



S3 buckets/Swift containers map to Swift storage policy

Storage policy creates dedicated ring file

Ring maps to Spectrum Scale fileset

ILM migration policy scans Spectrum Scale fileset and migrates to Spectrum Archive tape

Storing objects in the Spectrum Archive tape tier

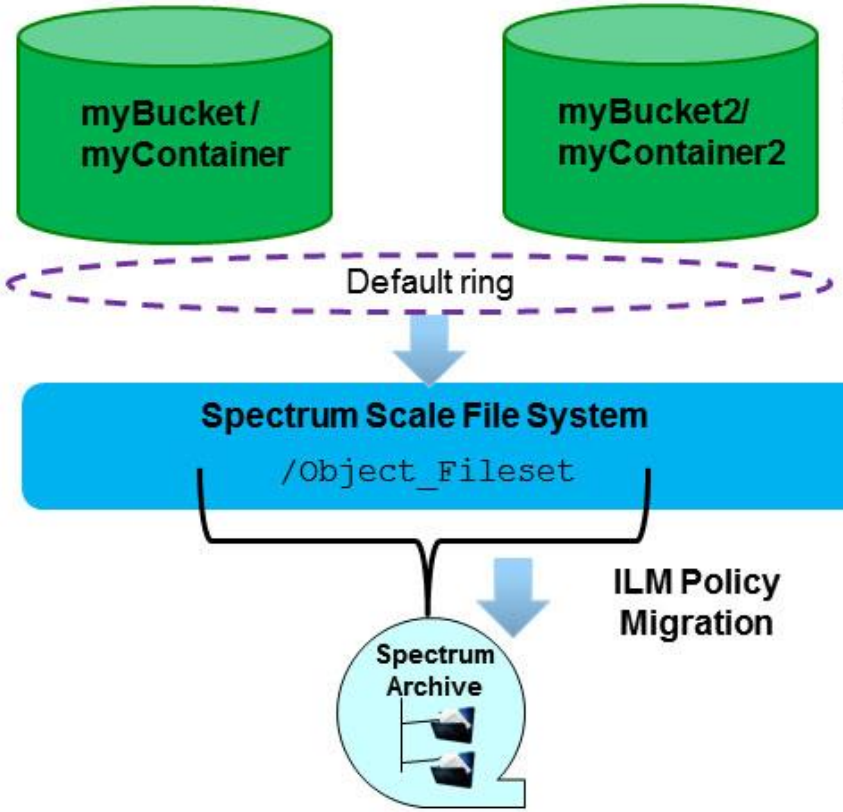
- 2. Create an online active archive that provides a single namespace to contain warm and cold data.
  Data can be migrated to tape in a selective, automated, and sweeping manner across all buckets and containers without application awareness. In this scenario, some of the data S3 buckets and Swift containers may reside on disk while some of the older data in the S3 buckets and Swift containers may reside on tape.

  The main advantage to this method is that no data copy or movement by the end user or application is needed.
  The main trade-off is that there is no S3 bucket or Swift container boundary to control the movement of data to the tape tier, making it more difficult to determine which objects reside on tape and must first be recalled in order to avoid application time-outs.

Storing objects in the Spectrum Archive tape tier

Spectrum Scale Swift Archive
Sweeping Migration (Application Un-Aware)

Any object in an S3 bucket/Swift container is eligible for migration to tape based on ILM policy criteria



S3 buckets/Swift containers use default object storage policy

Default Swift ring used for multiple containers

Default object fileset used for multiple containers

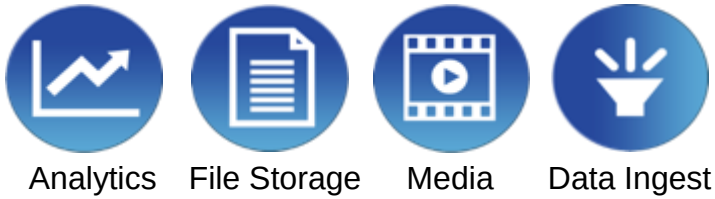
ILM migration policy scans object fileset on Spectrum Scale file system and migrates to Spectrum Archive tape

Spectrum Scale facts

Spectrum Scale facts

- 1000's of production systems
- Systems in production with 30+ PB capacity
- Several production clusters with >10k+ nodes
- Customers with more than >10 Billion files in a single system
- >400 GB/sec throughput to single system
  Currently building 1TB/s production system with 120PB capacity

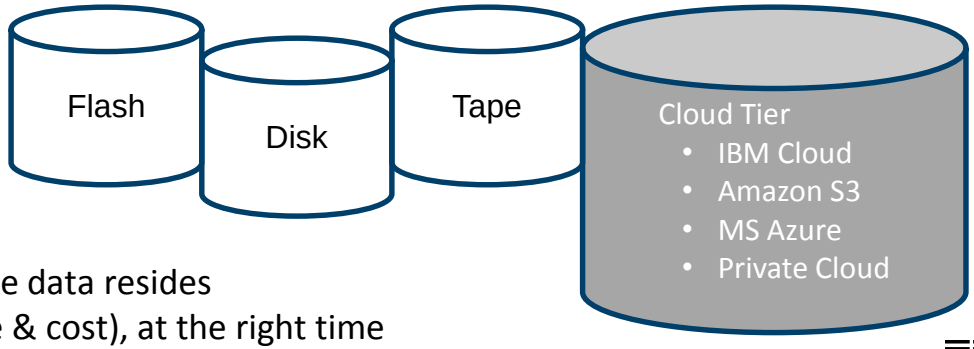
The Vision:
One solution for all of your data needs



- POSIX
- NFS
- Hadoop/Spark
- Object
- SMB
- iSCSI
- pNFS

Spectrum Scale

World-Wide
Data Distribution



- Single name space no matter where data resides
- Data in best location (performance & cost), at the right time
- Multi-tenancy
- All in software

IBM

*Future Capability

Key Software Value Adds

- Eliminate data migration through native File and Object integration
  - POSIX/NFS/SMB/S3/Swift
- High performance and scalability
- Authentication integration (LDAP/AD)
- Data protection
  - Snapshots, Backup, Disaster Recovery
- Encryption
- Compression
- Integrated or software-only solutions
- External storage integration
  - Integration of Tape or Deduplication appliance such as TSM, LTFS, ProtecTier
  - Optical storage integration on roadmap