

SETTING UP AN S3 COMPLIANT OBJECT STORE IN IBM SPECTRUM SCALE WITH OBJECT COMPRESSION, ILM AND TIERING

Abstract

This document describes the steps to enable AWS S3 compatibility in IBM Spectrum Scale Object with virtual hosted style bucket addressing. In addition, advanced Spectrum Scale features like Object Compression, Object Information Lifecycle Management and Object Tiering are discussed in depth.

NISHAAN DOCRAT
IBM Systems Hardware

Version 2.00 – April 2022

Trademarks

IBM, IBM Spectrum Scale, Spectrum Scale are trademarks or registered trademarks of the International Business Machines Corporation in the United States and other countries.

OpenStack, OpenStack Swift are trademarks or registered trademarks of the OpenStack Foundation.

Amazon Web Services, AWS, Amazon EC2, EC2, Amazon S3 are trademarks or registered trademarks of Amazon.com, Inc. or its affiliates in the United States and other countries.

HAProxy, HAProxy Community Edition are trademarks or registered trademarks of HAProxy Technologies LLC and its affiliated entities.

Dnsmasq and s3fs-fuse are distributed under the GPL.

s3cmd is the copyright of s3tools.org

MINIO is a trademark of Minio, Inc.

Other products may be trademarks or registered trademarks of their respective companies.

Acknowledgments

A special thanks to Maarten Kreuger and Harald Seipp for reviewing this document and suggesting improvements. Maarten has a comprehensive blog post on Swift and S3 ACLs that can be found here:

<https://community.ibm.com/community/user/storage/blogs/maarten-kreuger1/2022/04/29/spectrum-scale-object-access-with-swift-and-s3>

Both Maarten and Harald are active members of the IBM File and Object Storage community which can be found here:

<https://community.ibm.com/community/user/storage/communities/community-home?CommunityKey=1142f81e-95e4-4381-95d0-7977f20d53fa>

Table of Contents

| | |
|---|-----------|
| Executive Summary | 3 |
| S3 Compatibility and Virtual Hosted Style Bucket Addressing | 3 |
| Test Environment Setup | 4 |
| Setting Up Wildcard DNS Using dnsmasq | 5 |
| IBM Spectrum Scale Configuration | 7 |
| Setting up HAProxy for HTTP and HTTPS access | 20 |
| Setting up the S3 API Emulation on IBM Spectrum Scale Object | 28 |
| Enabling Virtual-hosted Style Bucket Addressing | 34 |
| Practical Use Case for an S3 Compliant Object Store | 39 |
| Sharing Buckets with different Users | 46 |
| Swift Object Storage and IBM Spectrum Scale | 56 |
| Spectrum Scale Object Compression | 64 |
| Object Information Lifecycle Management (ILM) | 76 |
| Object Tiering | 90 |
| Spectrum Scale Object Performance Tuning | 96 |
| Spectrum Scale Object Backup and Restore | 97 |
| Using AFM with Spectrum Scale Object | 98 |
| Summary | 98 |
| Appendix A: testobj.sh | 99 |

Executive Summary

IBM Spectrum Scale Object storage combines the many benefits of IBM Spectrum Scale with the most widely used open source Object store, OpenStack Swift. Swift offers cloud storage so that you can easily store and retrieve lots of data via a simple API. Its built for scale and optimized for durability, availability and concurrency across the entire data set and is ideal for storing unstructured data that can grow without bound.

Amazon's object storage solution, referred to as Simple Storage Service (S3), is the de-facto standard for object storage due to its large-scale commercial adoption. The S3 API has become the most common way to consume data hosted in object mode and has become the interoperability standard. Whilst Swift and AWS S3 are similar, AWS S3 includes features that are outside the scope of Swift itself. AWS S3 is a proprietary protocol whereas Swift is open source.

This document covers the steps to enable AWS S3 compatibility in IBM Spectrum Scale Object including the support for virtual hosted style bucket addressing. In addition, this document covers the use of advanced Spectrum Scale features like Object Compression, Object Information Lifecycle Management (ILM) and Object Tiering to showcase the benefits of using IBM Spectrum Scale Object for your Object storage requirements. The ability to support both OpenStack Swift and S3 Object protocols along with the advanced Spectrum Scale features are some of the compelling reasons to make use of IBM Spectrum Scale Object over native OpenStack Swift or competing products.

AUDIENCE: This document is intended for anyone involved in evaluating, acquiring, managing, operating, or designing an Object storage installation using IBM Spectrum Scale.

S3 Compatibility and Virtual Hosted Style Bucket Addressing

Since IBM Spectrum Scale is based on OpenStack Swift, in order to support a broader range of applications, it is necessary to emulate the S3 protocol on top of Swift. Swift has the capability to emulate S3 by implementing middleware. This middleware for the most part supports the vast majority of S3 API calls (see https://docs.openstack.org/swift/train/s3_compat.html for the S3/Swift REST API Comparison Matrix). As of the time of writing, IBM Spectrum Scale V5.1.3 is bundled with Swift V2.23.1 (Train). The emulation of the S3 API is delivered by the s3api middleware. Apart from the difference in authentication, S3 supports two types of bucket addressing. These are referred to as Path-style and Virtual-hosted style addressing. For Virtual-hosted style bucket addressing, the bucket name is part of the DNS name in the URL. For example:

<https://bucket.s3.amazonaws.com/object>
<https://bucket.s3-aws-region.amazonaws.com/object>

For Path-style addressing, the bucket name is not part of the DNS name in the URL. For example:

<https://s3.amazonaws.com/bucket/object>
<https://s3-aws-region.amazonaws.com/bucket/object>

Swift on the other hand supports the Path-style addressing by default. This is in the form of:

<http://swift.example.com/v1/account/container/object>

Apart from enabling S3 API compatibility using the s3api middleware, it is important to also be able to support the Virtual-hosted bucket addressing scheme that S3 offers. To support Virtual-hosted style addressing requires the use of a Wildcard DNS. It also requires DNS compliant bucket names.

The main purpose of this document is to explain how to implement S3 compatibility in IBM Spectrum Scale Object and also how to implement S3 Virtual-hosted style addressing in Swift. This is especially important as Amazon intends to deprecate Path-style API requests.

“Amazon S3 currently supports two request URI styles in all regions: path-style (also known as V1) that includes bucket name in the path of the URI (example: //s3.amazonaws.com/<bucketname>/key), and virtual-hosted style (also known as V2) which uses the bucket name as part of the domain name (example: //<bucketname>.s3.amazonaws.com/key). In our effort to continuously improve customer experience, the path-style naming convention is being retired in favor of virtual-hosted style request format. Customers should update their applications to use the virtual-hosted style request format when making S3 API requests before September 30th, 2020 to avoid any service disruptions. Customers using the AWS SDK can upgrade to the most recent version of the SDK to ensure their applications are using the virtual-hosted style request format.

Virtual-hosted style requests are supported for all S3 endpoints in all AWS regions. S3 will stop accepting requests made using the path-style request format in all regions starting September 30th, 2020. Any requests using the path-style request format made after this time will fail.” Source: <https://aws.amazon.com/blogs/aws/amazon-s3-path-deprecation-plan-the-rest-of-the-story/> and <https://docs.aws.amazon.com/AmazonS3/latest/userguide/access-bucket-intro.html>.

The deprecation of Path-style requests was originally targeted for September 2020. As of the time of writing, this deadline has been extended.

“Updated on September 23, 2020

Over the last year, we’ve heard feedback from many customers who have asked us to extend the deprecation date of path-style URLs. Based on this feedback, we have decided to delay the deprecation of path-style URLs to ensure that customers have the time that they need to transition to Virtual-hosted-style URLs.” Source: <https://aws.amazon.com/blogs/aws/amazon-s3-path-deprecation-plan-the-rest-of-the-story/>

Without the ability to implement Virtual-hosted style bucket addressing in IBM Spectrum Scale Object, we risk losing the ability to support applications built to make use of this addressing scheme exclusively. This document also includes the steps required to enable HTTPS access to the Spectrum Scale Object Store using HAProxy.

Test Environment Setup

The test environment consists of two virtual machines. A single-node Spectrum Scale cluster is configured on one of them with Cluster Export Services (CES) enabled. It therefore acts as an NSD server and protocol node. It is also configured as a DNS server and used for SSL termination via HAProxy. A second virtual machine is purely used as an object client. It is not part of the Spectrum Scale cluster. Unlike releases prior to 5.1, Object protocol support is only available on RHEL 8.2, 8.4 and 8.5. The latest compatibility matrix can be found here:

<https://www.ibm.com/support/knowledgecenter/STXKQY/gpfsclustersfaq.html>.

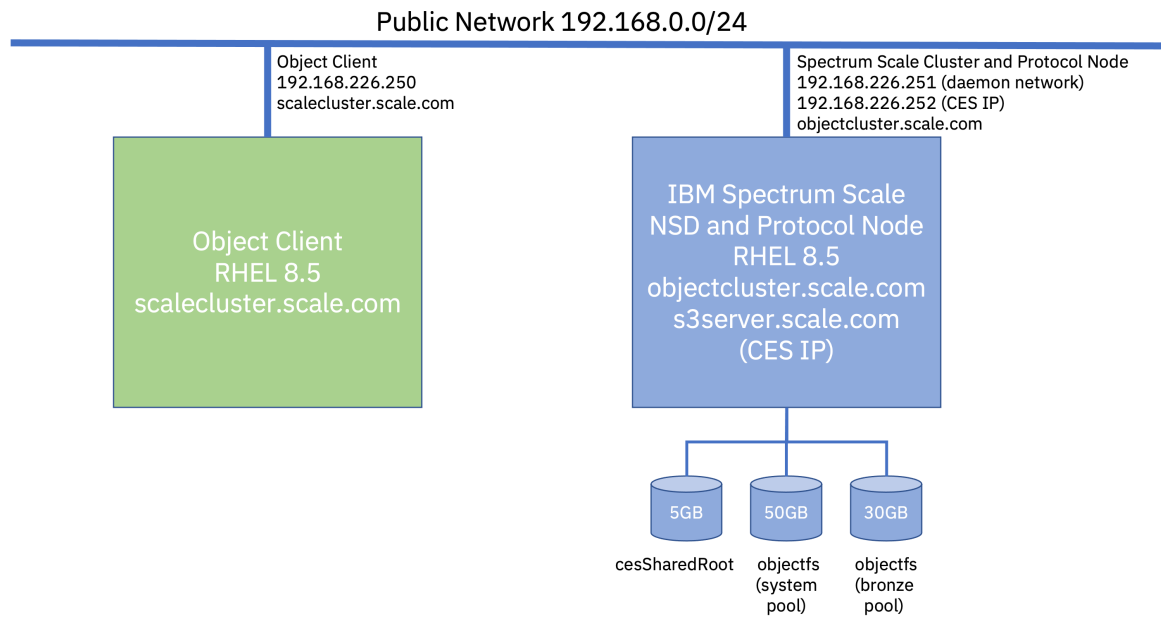


Figure 1: Test environment setup

It's important to note that the use of NTP or Chrony is critical for IBM Spectrum Scale Object to function. This should be setup across all nodes and clients in the environment. A single 5GB network shared disk was used to host the `cesSharedRoot` filesystem and a separate `objectfs` filesystem makes use of a 50GB network shared disk for the system pool and a 30GB network shared disk for the bronze pool. If you are using a Linux distribution that makes use of SELinux, make sure you set it to permissive mode.

Setting Up Wildcard DNS Using dnsmasq

Before installing IBM Spectrum Scale we need to have a functioning DNS server. Since we also want to enable an S3 compliant Object Storage using IBM Spectrum Scale, we need to be able to support both Path-style and Virtual-hosted style bucket addressing as described earlier. In order to achieve this, we need to setup wildcard DNS addressing for our domain (in this case, we are using `scale.com` as our domain). A wildcard DNS record is a record that answers DNS requests for subdomains we haven't defined yet (e.g. `bucket1.s3server.scale.com` should resolve to `s3server.scale.com`). In our test environment, `objectcluster.scale.com` will serve as our DNS server and HAProxy server.

To support wildcard DNS addressing in our test environment, we are going to use `dnsmasq` which is an open source, lightweight, easy to configure DNS forwarder and DHCP server. On RHEL 8.x, `dnsmasq` is installed by default. Below are the `dnsmasq` modifications that need to be made to the `/etc/dnsmasq.conf` file:

```
listen-address=::1,127.0.0.1,192.168.226.251 #change to match your public IPv4 address
server=8.8.8.8
server=8.8.4.4
address=/scale.com/192.168.226.252 #change to match your public IPv4 address
address=/scale.com/127.0.0.1
domain=scale.com
interface=ens160 #this matches my VM interface so make sure you adjust this to match yours
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

Note, the “`address=/scale.com/192.168.226.252`” directive forces any hostname that we don’t have an entry for to the CES IP address (acts as a wildcard). We also need to add the following lines to our hosts file as dnsmasq uses `/etc/hosts` as one of the sources to provide DNS services.

```
127.0.0.1    localhost localhost.scale.com localhost4 localhost4.scale.com4
::1         localhost localhost.scale.com localhost6 localhost6.scale.com6
192.168.226.250 scalecluster.scale.com scalecluster
192.168.226.251 objectcluster.scale.com objectcluster
192.168.226.252 s3server.scale.com s3server
```

Lastly, we need to ensure we are using local name resolution in the `/etc/resolv.conf` file. Also note that we have made the `/etc/resolv.conf` immutable using the `chattr` command so that it doesn’t get overwritten by NetworkManager on reboot. As an alternative, you could issue `nmcli con mod <interface> ipv4.dns "127.0.0.1"`.

```
[nishaandocrat@objectcluster ~]$ cat /etc/resolv.conf
# Generated by NetworkManager
search scale.com
nameserver 127.0.0.1
[root@objectcluster tmp]# chattr +i /etc/resolv.conf
[root@objectcluster tmp]# lsattr /etc/resolv.conf
---i----- /etc/resolv.conf
```

Once that is done, enable and restart the dnsmasq service and make sure it is running.

```
[root@objectcluster ~]# systemctl enable dnsmasq
[root@objectcluster ~]# systemctl restart dnsmasq
[root@objectcluster ~]# systemctl status dnsmasq
● dnsmasq.service - DNS caching server.
   Loaded: loaded (/usr/lib/systemd/system/dnsmasq.service; enabled; vendor pre>
   Active: active (running) since Fri 2022-04-01 23:03:45 SAST; 20s ago
   Main PID: 37033 (dnsmasq)
     Tasks: 1 (limit: 23527)
    Memory: 1.3M
   CGroup: /system.slice/dnsmasq.service
           └─37033 /usr/sbin/dnsmasq -k

Apr 01 23:03:45 objectcluster.scale.com systemd[1]: Started DNS caching server..
Apr 01 23:03:45 objectcluster.scale.com dnsmasq[37033]: started, version 2.79 c>
Apr 01 23:03:45 objectcluster.scale.com dnsmasq[37033]: compile time options: I>
Apr 01 23:03:45 objectcluster.scale.com dnsmasq[37033]: using nameserver 8.8.4.>
Apr 01 23:03:45 objectcluster.scale.com dnsmasq[37033]: using nameserver 8.8.8.>
Apr 01 23:03:45 objectcluster.scale.com dnsmasq[37033]: reading /etc/resolv.conf
Apr 01 23:03:45 objectcluster.scale.com dnsmasq[37033]: using nameserver 8.8.4.>
Apr 01 23:03:45 objectcluster.scale.com dnsmasq[37033]: using nameserver 8.8.8.>
Apr 01 23:03:45 objectcluster.scale.com dnsmasq[37033]: ignoring nameserver 127>
Apr 01 23:03:45 objectcluster.scale.com dnsmasq[37033]: read /etc/hosts - 6 add>
[root@objectcluster ~]#
```

Now we need to check name resolution is working.

```
root@objectcluster etc]# nslookup s3server
Server:      127.0.0.1
Address: 127.0.0.1#53

Name:      s3server.scale.com
Address: 192.168.226.252

[root@objectcluster etc]# nslookup objectcluster
Server:      127.0.0.1
Address: 127.0.0.1#53

Name:      objectcluster.scale.com
Address: 192.168.226.251

[root@objectcluster etc]#
```

And that wildcard name resolution also works.

```
[root@objectcluster etc]# nslookup bucket1.s3server.scale.com
Server:      127.0.0.1
Address: 127.0.0.1#53

Name:      bucket1.s3server.scale.com
Address: 192.168.226.252

[root@objectcluster etc]# nslookup bucket2.s3server.scale.com
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
Server:      127.0.0.1
Address: 127.0.0.1#53

Name:      bucket2.s3server.scale.com
Address: 192.168.226.252

[root@objectcluster etc]#
```

We now have wildcard DNS working in order to support Virtual-hosted style bucket addressing. This configuration is very simplistic and only meant to illustrate how a wildcard DNS works. In a real-world environment, most enterprises would need to involve their networking teams to facilitate this.

IBM Spectrum Scale Configuration

In the first version of this document, Spectrum Scale 5.0.5 was used and the cluster and protocol services were configured manually. In this revision, we will make use of the Spectrum Scale Installation Toolkit for cluster creation and protocol deployment. It is important to note that as of release 5.1, the OpenStack Swift RPMs are no longer bundled with Spectrum Scale. On Spectrum Scale 5.1.2.0 and other releases of 5.1.x the OpenStack 16 installation repositories were required on all protocol nodes to satisfy the required installation dependencies. This required a separate RHEL subscription license (OpenStack Platform subscription). As of 5.1.3.0, this is no longer required and the necessary dependencies for deploying the Object protocol service can be satisfied by the base RHEL OS repositories which is a welcome change.

Firstly, you need to make sure that all the required network ports are opened on your firewall prior to the installation. You can find a consolidated list of Spectrum Scale firewall ports here <https://www.ibm.com/docs/en/spectrum-scale/5.1.3?topic=firewall-recommendations-protocol-access>.

As an example of installing the required firewall rules, below is the output from our objectcluster host where we will be deploying Spectrum Scale.

```
[root@objectcluster ~]# firewall-cmd --list-all
public (active)
  target: default
  icmp-block-inversion: no
  interfaces: ens160
  sources:
  services: cockpit dhcpv6-client dns http https ssh
  ports: 1191/tcp 60000-61000/tcp 10080/tcp 4379/tcp 445/tcp 2049/tcp 111/tcp 32765/tcp 32767/tcp 32768/tcp
32769/tcp 8080/tcp 5000/tcp 5431/tcp 6200-6203/tcp 11211/tcp 47080/tcp 47443/tcp 80/tcp 443/tcp 4739/tcp 4739/udp
9980/tcp 9981/tcp 1191/udp 60000-61000/udp 2049/udp 111/udp 32765/udp 32767/udp 32768/udp 32769/udp 11211/udp
7452/tcp 7453/tcp 53/tcp 53/udp 35357/tcp 35357/udp
  protocols:
  forward: no
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:
```

Once the firewall rules are in place, you need to make sure you meet all the pre-requisites for using the Installation Toolkit. These can be found here https://www.ibm.com/docs/en/spectrum-scale/5.1.3?topic=toolkit-preparing-use-installation#task_j5r_dkv_1cb. After verifying you have met all the pre-requisites, you need to download the Spectrum Scale installation file and transfer it to your installation node. Once it is transferred, change the permissions to make it executable and run the install package.

```
[root@objectcluster tmp]# chmod 755 Spectrum_Scale_Data_Management-5.1.3.0-x86_64-Linux-install
[root@objectcluster tmp]# ./Spectrum_Scale_Data_Management-5.1.3.0-x86_64-Linux-install

Extracting License Acceptance Process Tool to /usr/lpp/mmfs/5.1.3.0 ...
```


Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
tail -n +660 ./Spectrum_Scale_Data_Management-5.1.3.0-x86_64-Linux-install | tar -C /usr/lpp/mmfs/5.1.3.0 -xvz --
exclude=installer --exclude=*_rpms --exclude=*_debs --exclude=*rpm --exclude=*tgz --exclude=*deb --exclude=*tools*
1> /dev/null

Installing JRE ...

If directory /usr/lpp/mmfs/5.1.3.0 has been created or was previously created during another extraction,
.rpm, .deb, and repository related files in it (if there were) will be removed to avoid conflicts with the ones
being extracted.

tail -n +660 ./Spectrum_Scale_Data_Management-5.1.3.0-x86_64-Linux-install | tar -C /usr/lpp/mmfs/5.1.3.0 --
wildcards -xvz ibm-java*tgz 1> /dev/null
tar -C /usr/lpp/mmfs/5.1.3.0/ -xzf /usr/lpp/mmfs/5.1.3.0/ibm-java*tgz
Defaulting to --text-only mode.

Invoking License Acceptance Process Tool ...
/usr/lpp/mmfs/5.1.3.0/ibm-java-x86_64-80/jre/bin/java -cp /usr/lpp/mmfs/5.1.3.0/LAP_HOME/LAPApp.jar
com.ibm.lex.lapapp.LAP -l /usr/lpp/mmfs/5.1.3.0/LA_HOME -m /usr/lpp/mmfs/5.1.3.0 -s /usr/lpp/mmfs/5.1.3.0 -
text_only

LICENSE INFORMATION

The Programs listed below are licensed under the following
License Information terms and conditions in addition to the
Program license terms previously agreed to by Client and
IBM. If Client does not have previously agreed to license
terms in effect for the Program, the International Program
License Agreement (i125-3301-15) applies.

Program Name (Program Number):
IBM Spectrum Scale Data Management Edition 5.1.3.0 (5737-
F34)
IBM Spectrum Scale Data Management Edition 5.1.3.0 (5641-
DM1)

Press Enter to continue viewing the license agreement, or
enter "1" to accept the agreement, "2" to decline it, "3"
to print it, "4" to read non-IBM terms, or "99" to go back
to the previous screen.
1

License Agreement Terms accepted.

Extracting Product RPMs to /usr/lpp/mmfs/5.1.3.0 ...
tail -n +660 ./Spectrum_Scale_Data_Management-5.1.3.0-x86_64-Linux-install | tar -C /usr/lpp/mmfs/5.1.3.0 --
wildcards -xvz Public Keys ansible-toolkit hdfs_rpms/rhel/hdfs_3.1.1.x hdfs_rpms/rhel/hdfs_3.3.x
ganesha_debs/ubuntu ganesha_rpms/rhel7 ganesha_rpms/rhel8 ganesha_rpms/sles15 gpfs_debs/ubuntu gpfs_rpms/rhel7
gpfs_rpms/rhel8 gpfs_rpms/sles15 object_rpms/rhel8 smb_debs/ubuntu smb_rpms/rhel7 smb_rpms/rhel8 smb_rpms/sles15
tools/repo zimon_debs/ubuntu zimon_rpms/rhel7 zimon_rpms/rhel8 zimon_rpms/sles15 gpfs_debs gpfs_rpms manifest 1>
/dev/null
- Public Keys
- ansible-toolkit
- hdfs_rpms/rhel/hdfs_3.1.1.x
- hdfs_rpms/rhel/hdfs_3.3.x
- ganesha_debs/ubuntu
- ganesha_rpms/rhel7
- ganesha_rpms/rhel8
- ganesha_rpms/sles15
- gpfs_debs/ubuntu
- gpfs_rpms/rhel7
- gpfs_rpms/rhel8
- gpfs_rpms/sles15
- object_rpms/rhel8
- smb_debs/ubuntu
- smb_rpms/rhel7
- smb_rpms/rhel8
- smb_rpms/sles15
- tools/repo
- zimon_debs/ubuntu
- zimon_rpms/rhel7
- zimon_rpms/rhel8
- zimon_rpms/sles15
- gpfs_debs
- gpfs_rpms
- manifest

Removing License Acceptance Process Tool from /usr/lpp/mmfs/5.1.3.0 ...
rm -rf /usr/lpp/mmfs/5.1.3.0/LAP_HOME /usr/lpp/mmfs/5.1.3.0/LA_HOME

Removing JRE from /usr/lpp/mmfs/5.1.3.0 ...
rm -rf /usr/lpp/mmfs/5.1.3.0/ibm-java*tgz

=====
Product packages successfully extracted to /usr/lpp/mmfs/5.1.3.0

Cluster installation and protocol deployment
To install a cluster or deploy protocols with the IBM Spectrum Scale Installation Toolkit:
/usr/lpp/mmfs/5.1.3.0/ansible-toolkit/spectrumscale -h

To install a cluster manually: Use the GPFS packages located within /usr/lpp/mmfs/5.1.3.0/gpfs_<rpms/debs>
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
To upgrade an existing cluster using the IBM Spectrum Scale Installation Toolkit:
1) Review and update the config: /usr/lpp/mmfs/5.1.3.0/ansible-toolkit/spectrumscale config update
2) Update the cluster configuration to reflect the current cluster config:
   /usr/lpp/mmfs/5.1.3.0/ansible-toolkit/spectrumscale config populate -N <node>
3) Use online or offline upgrade depending on your requirements:
   - Run the online rolling upgrade: /usr/lpp/mmfs/5.1.3.0/ansible-toolkit/spectrumscale upgrade -h
   - Run the offline upgrade: /usr/lpp/mmfs/5.1.3.0/ansible-toolkit/spectrumscale upgrade config offline -N;
     /usr/lpp/mmfs/5.1.3.0/ansible-toolkit/spectrumscale upgrade run
You can also run the parallel offline upgrade to upgrade all nodes parallelly after shutting down GPFS
and stopping protocol services on all nodes.
You can run the parallel offline upgrade on all nodes in the cluster, not on a subset of nodes.

To add nodes to an existing cluster using the IBM Spectrum Scale Installation Toolkit:
1) Add nodes to the cluster definition file: /usr/lpp/mmfs/5.1.3.0/ansible-toolkit/spectrumscale node add -h
2) Install IBM Spectrum Scale on the new nodes: /usr/lpp/mmfs/5.1.3.0/ansible-toolkit/spectrumscale install
-h
3) Deploy protocols on the new nodes: /usr/lpp/mmfs/5.1.3.0/ansible-toolkit/spectrumscale deploy -h

To add NSDs or file systems to an existing cluster using the IBM Spectrum Scale Installation Toolkit:
1) Add NSDs or file systems to the cluster definition: /usr/lpp/mmfs/5.1.3.0/ansible-toolkit/spectrumscale
nsd add -h
2) Install the NSDs or file systems: /usr/lpp/mmfs/5.1.3.0/ansible-toolkit/spectrumscale install -h

To update the cluster definition to reflect the current cluster config examples:
   /usr/lpp/mmfs/5.1.3.0/ansible-toolkit/spectrumscale config populate -N <node>
1) Manual updates outside of the installation toolkit
2) Sync the current cluster state to the installation toolkit prior to upgrade
3) Switching from a manually managed cluster to the installation toolkit

=====
To get up and running quickly, consult the IBM Spectrum Scale Protocols Quick Overview:
https://www.ibm.com/docs/en/STXKQY_5.1.3/pdf/scale_povr.pdf
=====
[root@objectcluster tmp]#
```

Now we need to navigate to the Installation Toolkit directory and create our cluster configuration. The installation directory should be `/usr/lpp/mmfs/5.1.3.0/ansible-toolkit/`. The first step is to setup the installer node which in our case is the host `objectcluster`. Make sure your installation repositories are configured and available using the `"yum repolist"` and `"yum repoinfo"` commands prior to running this command. The full list of `"spectrumscale"` command options can be found here <https://www.ibm.com/docs/en/spectrum-scale/5.1.3?topic=reference-spectrumscale-command#spectrumscale>.

```
[root@objectcluster ansible-toolkit]# ./spectrumscale setup -s 192.168.226.251 -st ss
[ INFO ] Installing prerequisites for install node
[ INFO ] Installing Ansible version 2.9.15.
[ INFO ] Install Toolkit setup type is set to Spectrum Scale (default). If an ESS is in the cluster, run this
command to set ESS mode: ./spectrumscale setup -s server_ip -st ess
[ INFO ] Your ansible controller node has been configured to use the IP 192.168.226.251 to communicate with other
nodes.
[ INFO ] Port 10080 will be used for package distribution.
[ INFO ] SUCCESS
[ INFO ] Tip : Designate protocol, nsd and admin nodes in your environment to use during install:./spectrumscale -
v node add <node> -p -a -n
[root@objectcluster ansible-toolkit]#
```

We can now populate our cluster information. We will start by adding our Spectrum Scale nodes. In our case, the single node `objectcluster` is going to serve multiple purposes.

```
[root@objectcluster ansible-toolkit]# ./spectrumscale node add -g -q -m -a -n -c -p objectcluster.scale.com
[ INFO ] Adding node objectcluster.scale.com as a GPFS node.
[ INFO ] Setting objectcluster.scale.com as a protocol node.
[ INFO ] Configuration updated.
[ INFO ] Tip : If all node designations are complete, configure the protocol environment as needed:
./spectrumscale config protocols -f cesSharedRoot -m /ibm/cesSharedRoot
[ INFO ] Adding node objectcluster.scale.com as a quorum node.
[ INFO ] Adding node objectcluster.scale.com as a manager node.
[ INFO ] Adding node objectcluster.scale.com as an NSD server.
[ INFO ] Configuration updated.
[ INFO ] Tip : If all node designations are complete, add NSDs to your cluster definition and define required
filesystems:./spectrumscale nsd add <device> -p <primary node> -s <secondary node> -fs <file system>
[ INFO ] Setting objectcluster.scale.com as an admin node.
[ INFO ] Configuration updated.
[ INFO ] Tip : Designate protocol or nsd nodes in your environment to use during install:./spectrumscale node add
<node> -p -n
[ INFO ] Setting objectcluster.scale.com as a callhome node.
[ INFO ] Configuration updated.
[ INFO ] Setting objectcluster.scale.com as a GUI server.
[root@objectcluster ansible-toolkit]#
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

Now we populate the cluster information.

```
[root@objectcluster ansible-toolkit]# ./spectrumscale config gpfs -c objectcluster -e 60000-61000
[ INFO ] Setting clustername to objectcluster
[ INFO ] Setting ephemeral_port_range to 60000-61000
[root@objectcluster ansible-toolkit]#
```

We want to also enable performance monitoring. It should be on by default (-l lists the current configuration).

```
root@objectcluster ansible-toolkit]# ./spectrumscale config perfmon -l
[ INFO ] Current settings are as follows:
[ INFO ] Performance Monitoring reconfiguration is on. Collectors may be moved to different nodes, sensors may be added to nodes, and sensors may be reset to defaults.
[root@objectcluster ansible-toolkit]#
```

Its time for our NSD and filesystem definitions. The available disk device names can be found using the *lsblk* command.

```
[root@objectcluster ~]# lsblk
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sr0          11:0    1 1024M  0 rom
nvme0n1      259:0    0   50G  0 disk
├─nvme0n1p1  259:1    0  300M  0 part /boot
├─nvme0n1p2  259:2    0    4G  0 part [SWAP]
└─nvme0n1p3  259:3    0 45.8G  0 part /
nvme0n2      259:4    0    5G  0 disk
nvme0n3      259:6    0   50G  0 disk
nvme0n4      259:8    0   30G  0 disk
[nishaandocrat@objectcluster ~]$
```

For now, we will just be creating a separate filesystem for the cesSharedRoot and objectfs.

```
[root@objectcluster ansible-toolkit]# ./spectrumscale nsd add -p objectcluster.scale.com -fs cesSharedRoot -u dataAndMetadata /dev/nvme0n2
[ INFO ] Connecting to objectcluster.scale.com to check devices and expand wildcards.
[ INFO ] Looking up details of /dev/nvme0n2.
[ INFO ] The installer will create the new file system cesSharedRoot if it does not exist.
[ INFO ] Adding NSD nsd1 on objectcluster.scale.com using device /dev/nvme0n2.
[ INFO ] Configuration updated
[ INFO ] Tip : If all node designations and any required protocol configurations are complete, proceed to check the installation configuration: ./spectrumscale install --precheck
[root@objectcluster ansible-toolkit]# ./spectrumscale nsd add -p objectcluster.scale.com -fs objectfs -u dataAndMetadata /dev/nvme0n3
[ INFO ] Connecting to objectcluster.scale.com to check devices and expand wildcards.
[ INFO ] Looking up details of /dev/nvme0n3.
[ INFO ] The installer will create the new file system objectfs if it does not exist.
[ INFO ] Adding NSD nsd2 on objectcluster.scale.com using device /dev/nvme0n3.
[ INFO ] Configuration updated
[ INFO ] Tip : If all node designations and any required protocol configurations are complete, proceed to check the installation configuration: ./spectrumscale install --precheck
[root@objectcluster ansible-toolkit]#
```

A quick verify of the NSD information.

```
root@objectcluster ansible-toolkit]# ./spectrumscale nsd list
[ INFO ] Name FS          Size(GB) Usage      FG Pool   Device      Servers
[ INFO ] nsd1 cesSharedRoot 5.0      dataAndMetadata 1 system /dev/nvme0n2 objectcluster.scale.com
[ INFO ] nsd2 objectfs     50.0     dataAndMetadata 1 Default /dev/nvme0n3 objectcluster.scale.com
[root@objectcluster ansible-toolkit]#
```

The filesystem information is created after you add NSD's. We can list them as follows. If you need to make any changes you can do so now. In our use case, all the defaults are fine.

```
[root@objectcluster ansible-toolkit]# ./spectrumscale filesystem list
[ INFO ] Name      BlockSize Mountpoint  NSDs Assigned  Default Data Replicas  Max Data Replicas
Default Metadata Replicas  Max Metadata Replicas
[ INFO ] cesSharedRoot 4M      /ibm/cesSharedRoot 1              1                      2
1
[ INFO ] objectfs     4M      /ibm/objectfs     1              1                      2
1
[ INFO ]
[root@objectcluster ansible-toolkit]#
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

If this was a production cluster, you would need to populate the callhome information. Since this is just a test environment, we will disable callhome.

```
[root@objectcluster ansible-toolkit]# ./spectrumscale callhome disable
[ INFO ] Disabling the callhome.
[ INFO ] Configuration updated.
[root@objectcluster ansible-toolkit]#
```

The next step is to configure protocols and enable the object protocol.

```
[root@objectcluster ansible-toolkit]# ./spectrumscale config protocols -f cesSharedRoot -m /ibm/cesSharedRoot -e 192.168.226.252
[ INFO ] Setting export_ip_pool to ['192.168.226.252']
[ INFO ] Setting filesystem to cesSharedRoot
[ INFO ] Setting mountpoint to /ibm/cesSharedRoot
[ INFO ] Tip :Enable NFS, Object, SMB or HDFS protocols as appropriate:./spectrumscale enable nfs|object|smb|hdfs
[root@objectcluster ansible-toolkit]# ./spectrumscale enable object
[ INFO ] Enabling OBJECT on all protocol nodes.
[ INFO ] Tip :If all node designations and any required protocol configurations are complete, proceed to check the installation configuration:./spectrumscale deploy --precheck
[root@objectcluster ansible-toolkit]#
```

We now need to configure the object protocol service. The command to do so and options are as follows.

```
root@objectcluster ansible-toolkit]# ./spectrumscale config object --help
usage: spectrumscale config object [-h] [-l] [-f FILESYSTEM] [-m MOUNTPOINT]
                                   [-e ENDPOINT] [-o OBJECTBASE]
                                   [-i INODEALLOCATION] [-au ADMINUSER]
                                   [-ap [ADMINPASSWORD]] [-su SWIFTUSER]
                                   [-sp [SWIFTPASSWORD]]
                                   [-dp [DATABASEPASSWORD]] [-s3 {on,off}]

optional arguments:
  -h, --help                show this help message and exit
  -l, --list                 List the current settings in the configuration
  -f FILESYSTEM, --filesystem FILESYSTEM
                           Specify the Object File System Name
  -m MOUNTPOINT, --mountpoint MOUNTPOINT
                           Specify the absolute path to your file system in which
                           the objects will reside
  -e ENDPOINT, --endpoint ENDPOINT
                           Specify the hostname which maps round-robin to all CES
                           IP addresses
  -o OBJECTBASE, --objectbase OBJECTBASE
                           Specify the GPFS fileset to be created/used as the
                           object base
  -i INODEALLOCATION, --inodeallocation INODEALLOCATION
                           Specify the GPFS fileset inode allocation to be used
                           by the object base
  -au ADMINUSER, --adminuser ADMINUSER
                           Specify the user name for the Admin User
  -ap [ADMINPASSWORD], --adminpassword [ADMINPASSWORD]
                           Specify the password for the Admin User (or leave
                           blank to prompt)
  -su SWIFTUSER, --swiftuser SWIFTUSER
                           Specify the user name for the Swift User
  -sp [SWIFTPASSWORD], --swiftpassword [SWIFTPASSWORD]
                           Specify the password for the Swift User (or leave
                           blank to prompt)
  -dp [DATABASEPASSWORD], --databasepassword [DATABASEPASSWORD]
                           Specify the password for object database (or leave
                           blank to prompt)
  -s3 {on,off}, --enable_s3 {on,off}
                           Specify if S3 emulation is enabled
[root@objectcluster ansible-toolkit]#
```

We will issue the following command to match our requirements. S3 compatibility is also enabled in this command via the “-s3 on” option. Note, you will be prompted for the encryption key during execution of this command.

```
[root@objectcluster ansible-toolkit]# ./spectrumscale config object -f objectfs -m /ibm/objectfs -e s3server.scale.com -o object_fileset -i 200000 -au ksAdmin -ap Passw0rd -su ksSwift -sp Passw0rd -dp Passw0rd -s3 on
[ INFO ] Setting Object File System Name to objectfs
[ INFO ] Setting Object File System Mountpoint to /ibm/objectfs
[ INFO ] Setting Endpoint Hostname to s3server.scale.com
[ INFO ] Setting GPFS Object Base to object_fileset
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
[ INFO ] Setting GPFS Fileset inode allocation to 200000
[ INFO ] Setting Admin User to ksAdmin
Enter the secret encryption key:
Repeat the secret encryption key:
[ INFO ] Setting Admin Password
[ INFO ] Setting Swift User to ksSwift
[ INFO ] Setting Swift Password
[ INFO ] Setting Database Password
[ INFO ] Setting Endpoint Hostname to s3server.scale.com
[ INFO ] Setting GPFS Object Base to object_fileset
[ INFO ] Setting Object S3 to True
[ INFO ] Setting Object File System Mountpoint to /ibm/objectfs
[ INFO ] Setting Object File System to objectfs
[ INFO ] Setting GPFS Fileset inode allocation to 200000
[ INFO ] Tip :If all node designations and any required protocol configurations are complete, proceed to check the
installation configuration:
[ INFO ] ./spectrumscale deploy --precheck
[root@objectcluster ansible-toolkit]#
```

Just to be safe, we will double check our object configuration to ensure we didn't make any mistakes.

```
[root@objectcluster ansible-toolkit]# ./spectrumscale config object --l
[ INFO ] Current settings are as follows:
[ INFO ] Object File System Name is objectfs.
[ INFO ] Object File System Mountpoint is /ibm/objectfs.
[ INFO ] Endpoint Hostname is s3server.scale.com.
[ INFO ] GPFS Object Base is object_fileset.
[ INFO ] GPFS Fileset inode allocation is 200000.
[ INFO ] Admin User is ksAdmin.
[ INFO ] Admin Password is stored in the clusterdefinition file
[ INFO ] Swift User is ksSwift.
[ INFO ] Swift Password is stored in the clusterdefinition file
[ INFO ] Database Password is stored in the clusterdefinition file
[ INFO ] Tip :If all node designations and any required protocol configurations are complete, proceed to check the
installation configuration:
[ INFO ] ./spectrumscale deploy --precheck
[root@objectcluster ansible-toolkit]#
```

And just double check the node configuration prior to installation and deployment.

```
[root@objectcluster ansible-toolkit]# ./spectrumscale node list
[ INFO ] List of nodes in current configuration:
[ INFO ] [Installer Node]
[ INFO ] 192.168.226.251
[ INFO ]
[ INFO ] [Cluster Details]
[ INFO ] Name: objectcluster
[ INFO ] Setup Type: Spectrum Scale
[ INFO ]
[ INFO ] [Protocols]
[ INFO ] Object : Enabled
[ INFO ] SMB : Disabled
[ INFO ] NFS : Disabled
[ INFO ] HDFS : Disabled
[ INFO ]
[ INFO ] [Extended Features]
[ INFO ] File Audit logging : Disabled
[ INFO ] Watch folder : Disabled
[ INFO ] Management GUI : Enabled
[ INFO ] Performance Monitoring : Enabled
[ INFO ] Callhome : Disabled
[ INFO ]
[ INFO ] GPFS Admin Quorum Manager NSD Protocol GUI Callhome OS Arch
[ INFO ] Node Node Node Server Node Server Server Server
[ INFO ] objectcluster.scale.com X X X X X X X rhel8 x86 64
[ INFO ]
[ INFO ] [Export IP address]
[ INFO ] 192.168.226.252 (pool)
[root@objectcluster ansible-toolkit]
```

We can now run the Installation Toolkit pre-installation check. The first run failed as we were missing a required package “*elfutils-devel*”. The beauty of the Installation Toolkit is that you can just fix whatever error is reported and rerun the command to pick up from where it left off.

```
[root@objectcluster ansible-toolkit]# ./spectrumscale install -pr
[ INFO ] Logging to file: /usr/lpp/mmfs/5.1.3.0/ansible-toolkit/logs/INSTALL-PRECHECK-20-03-2022_02:37:20.log
[ INFO ] Validating configuration
[ WARN ] Ensure that base OS repositories are configured and enabled so that package dependencies can be satisfied
during installation.
[ WARN ] objectcluster.scale.com is specified as both an NSD server and a protocol node. It is recommended that a
node not be both an NSD server and a protocol node
[ WARN ] Only one GUI server specified. The Graphical User Interface will not be highly available.
[ INFO ] Install toolkit will not configure file audit logging as it has been disabled.
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
[ INFO ] Performing GPFS checks.
[ INFO ] Running environment checks
[ INFO ] Skipping license validation as no existing GPFS cluster detected.
[ WARN ] With 1 node specified functionality will be limited
[ WARN ] Firewall running on objectcluster.scale.com node. Install toolkit may fail if firewall settings are
incorrect. Consult the documentation for more information on running Spectrum Scale behind a firewall:
https://www.ibm.com/docs/en/spectrum-scale/5.1.0?topic=topics-securing-spectrum-scale-system-using-firewall.
[ INFO ] Checking pre-requisites for portability layer.
[ INFO ] GPFS precheck OK
[ INFO ] Performing NSDs checks.
[ WARN ] The NSD nsd1 only has one server configured. This may affect the ability to run concurrent maintenance on
this cluster.
[ WARN ] The NSD nsd2 only has one server configured. This may affect the ability to run concurrent maintenance on
this cluster.
.
.
.
.
.
.
*****
*****
[ INFO ] ok: [objectcluster.scale.com]
[ INFO ] PLAY RECAP
*****
*****
[ INFO ] localhost : ok=4 changed=1 unreachable=0 failed=0 skipped=0 rescued=0
ignored=0
[ INFO ] objectcluster.scale.com : ok=14 changed=0 unreachable=0 failed=0 skipped=48 rescued=0
ignored=0
[ INFO ] Pre-check successful for install.
[ INFO ] Tip : ./spectrumscale install
[root@objectcluster ansible-toolkit]#
```

Assuming your pre-check came back fine, you can now run the install command to create the cluster and all of its associated resources.

```
[root@objectcluster ansible-toolkit]# ./spectrumscale install
[ INFO ] Logging to file: /usr/lpp/mmfs/5.1.3.0/ansible-toolkit/logs/INSTALL-20-03-2022_02:42:07.log
[ INFO ] Validating configuration
[ WARN ] Ensure that base OS repositories are configured and enabled so that package dependencies can be satisfied
during installation.
[ WARN ] objectcluster.scale.com is specified as both an NSD server and a protocol node. It is recommended that a
node not be both an NSD server and a protocol node
[ WARN ] Only one GUI server specified. The Graphical User Interface will not be highly available.
[ INFO ] Install toolkit will not configure file audit logging as it has been disabled.
[ INFO ] Running pre-install checks
[ INFO ] Running environment checks
[ INFO ] Skipping license validation as no existing GPFS cluster detected.
[ WARN ] With 1 node specified functionality will be limited
[ WARN ] Firewall running on objectcluster.scale.com node. Install toolkit may fail if firewall settings are
incorrect. Consult the documentation for more information on running Spectrum Scale behind a firewall:
https://www.ibm.com/docs/en/spectrum-scale/5.1.0?topic=topics-securing-spectrum-scale-system-using-firewall.
[ INFO ] Checking pre-requisites for portability layer.
[ INFO ] GPFS precheck OK
[ WARN ] The NSD nsd1 only has one server configured. This may affect the ability to run concurrent maintenance on
this cluster.
[ WARN ] The NSD nsd2 only has one server configured. This may affect the ability to run concurrent maintenance on
this cluster.
[ INFO ] Running environment checks for Performance Monitoring
[ INFO ] Running environment checks for file Audit logging
[ INFO ] Network check from admin node objectcluster.scale.com to all other nodes in the cluster passed
[ INFO ] Network check from protocol node objectcluster.scale.com to all other protocol nodes in the cluster
passed
[ INFO ] The install toolkit will not configure call home as it is disabled. To enable call home, use the
following CLI command: ./spectrumscale callhome enable
[ INFO ] Preparing nodes for install
[ INFO ] PLAY [localhost]
.
.
.
.
.
.
[ INFO ] PLAY RECAP
*****
*****
[ INFO ] localhost : ok=4 changed=1 unreachable=0 failed=0 skipped=0 rescued=0
ignored=0
[ INFO ] objectcluster.scale.com : ok=170 changed=41 unreachable=0 failed=0 skipped=304 rescued=0
ignored=0
[ INFO ] Destroying repository...
[ INFO ] Checking for a successful install
[ INFO ] Checking state of GPFS
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
[ INFO ] Checking state of GPFS on all nodes
[ INFO ] GPFS active on all nodes
[ INFO ] GPFS ACTIVE
[ INFO ] Checking state of NSDs
[ INFO ] NSDs ACTIVE
[ INFO ] Checking state of Performance Monitoring
[ INFO ] Running Performance Monitoring post-install checks
[ INFO ] pmcollector running on all nodes
[ INFO ] pmsensors running on all nodes
[ INFO ] Performance Monitoring ACTIVE
[ INFO ] Checking state of GUI
[ INFO ] Running Graphical User Interface post-install checks
[ INFO ] Graphical User Interface running on all GUI servers
[ INFO ] Enter one of the following addresses into a web browser to access the Graphical User Interface:
objectcluster.scale.com
[ INFO ] GUI ACTIVE
[ INFO ] SUCCESS
[ INFO ] All services running
[ INFO ] StanzaFile and NodeDesc file for NSD, filesystem, and cluster setup have been saved to /usr/lpp/mmfs
folder on node: objectcluster.scale.com
[ INFO ] Installation successful. 1 GPFS node active in cluster objectcluster. Completed in 10 minutes 50 seconds.
[ INFO ] Tip :If all node designations and any required protocol configurations are complete, proceed to check the
deploy configuration:./spectrumscale deploy --precheck
[root@objectcluster ansible-toolkit]#
```

Its always a good idea to run the post-install check as well.

```
[root@objectcluster ansible-toolkit]# ./spectrumscale install -po
[ INFO ] Logging to file: /usr/lpp/mmfs/5.1.3.0/ansible-toolkit/logs/INSTALL-POSTCHECK-20-03-2022_02:53:27.log
[ WARN ] objectcluster.scale.com is specified as both an NSD server and a protocol node. It is recommended that a
node not be both an NSD server and a protocol node
[ WARN ] Only one GUI server specified. The Graphical User Interface will not be highly available.
[ INFO ] Install toolkit will not configure file audit logging as it has been disabled.
[ INFO ] Checking state of GPFS
[ INFO ] Checking state of GPFS on all nodes
[ INFO ] GPFS active on all nodes
[ INFO ] GPFS ACTIVE
[ INFO ] Checking state of NSDs
[ INFO ] NSDs ACTIVE
[ INFO ] Checking state of Performance Monitoring
[ INFO ] Running Performance Monitoring post-install checks
[ INFO ] pmcollector running on all nodes
[ INFO ] pmsensors running on all nodes
[ INFO ] Performance Monitoring ACTIVE
[ INFO ] Checking state of GUI
[ INFO ] Running Graphical User Interface post-install checks
[ INFO ] Graphical User Interface running on all GUI servers
[ INFO ] Enter one of the following addresses into a web browser to access the Graphical User Interface:
objectcluster.scale.com
[ INFO ] GUI ACTIVE
[ INFO ] SUCCESS
[ INFO ] All services running
[ INFO ] StanzaFile and NodeDesc file for NSD, filesystem, and cluster setup have been saved to /usr/lpp/mmfs
folder on node: objectcluster.scale.com
[root@objectcluster ansible-toolkit]#
```

Now that our cluster is created, we can deploy protocols. We first need to do a deploy pre-check.

```
[root@objectcluster ansible-toolkit]# ./spectrumscale deploy -pr
[ INFO ] Logging to file: /usr/lpp/mmfs/5.1.3.0/ansible-toolkit/logs/DEPLOY-PRECHECK-20-03-2022_02:54:13.log
[ INFO ] Validating configuration
[ WARN ] Ensure that base OS repositories are configured and enabled so that package dependencies can be satisfied
during installation.
[ WARN ] objectcluster.scale.com is specified as both an NSD server and a protocol node. It is recommended that a
node is not both a protocol and NSD server.
Enter the secret encryption key:
Repeat the secret encryption key:
[ WARN ] Only one GUI server specified. The Graphical User Interface will not be highly available.
[ INFO ] Install toolkit will not configure file audit logging as it has been disabled.
[ INFO ] Performing Filesystem checks.
[ INFO ] NSDs are in a valid state
[ INFO ] Performing Cluster Export Services checks.
[ INFO ] Running environment checks for protocols
[ INFO ] Checking state of GPFS on all nodes
[ INFO ] GPFS active on all nodes
[ INFO ] Checking state of GPFS on all nodes
[ INFO ] GPFS active on all nodes
[ WARN ] Firewall running on objectcluster.scale.com node. Install toolkit may fail if firewall settings are
incorrect. Consult the documentation for more information on running Spectrum Scale behind a firewall:
https://www.ibm.com/docs/en/spectrum-scale/5.1.0?topic=topics-securing-spectrum-scale-system-using-firewall.
[ INFO ] protocol precheck OK
[ INFO ] Performing Object Store checks.
[ INFO ] Running environment checks for Object Storage
[ INFO ] Object Storage ready for install
[ INFO ] Performing Performance Monitoring checks.
[ INFO ] Running environment checks for Performance Monitoring
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
.
.
.
.
.
.
*****
*****
[ INFO ] ok: [objectcluster.scale.com]
[ INFO ] PLAY RECAP
*****
*****
[ INFO ] localhost : ok=4 changed=1 unreachable=0 failed=0 skipped=0 rescued=0
ignored=0
[ INFO ] objectcluster.scale.com : ok=14 changed=0 unreachable=0 failed=0 skipped=48 rescued=0
ignored=0
[ INFO ] Pre-check successful for deploy.
[ INFO ] Tip : ./spectrumscale deploy
[root@objectcluster ansible-toolkit]#
```

If the deploy pre-check came back fine, we can do the actual deployment as follows.

```
[root@objectcluster ansible-toolkit]# ./spectrumscale deploy
[ INFO ] Logging to file: /usr/lpp/mmfs/5.1.3.0/ansible-toolkit/logs/DEPLOY-20-03-2022_02:55:54.log
[ INFO ] Validating configuration
[ WARN ] Ensure that base OS repositories are configured and enabled so that package dependencies can be satisfied
during installation.
[ WARN ] objectcluster.scale.com is specified as both an NSD server and a protocol node. It is recommended that a
node is not both a protocol and NSD server.
Enter the secret encryption key:
Repeat the secret encryption key:
[ WARN ] Only one GUI server specified. The Graphical User Interface will not be highly available.
[ INFO ] Install toolkit will not configure file audit logging as it has been disabled.
[ INFO ] Running pre-install checks
[ INFO ] NSDs are in a valid state
[ INFO ] Running environment checks for protocols
[ INFO ] Checking state of GPFS on all nodes
[ INFO ] GPFS active on all nodes
[ INFO ] Checking state of GPFS on all nodes
[ INFO ] GPFS active on all nodes
[ WARN ] Firewall running on objectcluster.scale.com node. Install toolkit may fail if firewall settings are
incorrect. Consult the documentation for more information on running Spectrum Scale behind a firewall:
https://www.ibm.com/docs/en/spectrum-scale/5.1.0?topic=topics-securing-spectrum-scale-system-using-firewall.
[ INFO ] protocol precheck OK
[ INFO ] Running environment checks for Object Storage
[ INFO ] Object Storage ready for install
.
.
.
.
.
.
[ INFO ] PLAY RECAP
*****
*****
[ INFO ] localhost : ok=4 changed=1 unreachable=0 failed=0 skipped=0 rescued=0
ignored=0
[ INFO ] objectcluster.scale.com : ok=80 changed=20 unreachable=0 failed=0 skipped=121 rescued=0
ignored=0
[ INFO ] Destroying repository...
[ INFO ] Checking for a successful install
[ INFO ] Checking state of Filesystem
[ INFO ] File systems have been created successfully
[ INFO ] Filesystem ACTIVE
[ INFO ] Checking state of Cluster Export Services
[ INFO ] Checking state of CES on all nodes
[ INFO ] Checking state of CES on all nodes
[ INFO ] Cluster Export Services ACTIVE
[ INFO ] Checking state of Object Store
[ INFO ] Running Object post-install checks
[ INFO ] Checking state of OBJ on all nodes
[ INFO ] Checking state of OBJ on all nodes
[ INFO ] Object Store ACTIVE
[ INFO ] Checking state of Performance Monitoring
[ INFO ] Running Performance Monitoring post-install checks
[ INFO ] pmcollector running on all nodes
[ INFO ] pmsensors running on all nodes
[ INFO ] Performance Monitoring ACTIVE
[ INFO ] Checking state of GUI
[ INFO ] Running Graphical User Interface post-install checks
[ INFO ] Graphical User Interface running on all GUI servers
[ INFO ] Enter one of the following addresses into a web browser to access the Graphical User Interface:
objectcluster.scale.com
[ INFO ] GUI ACTIVE
[ INFO ] SUCCESS
```


Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
[ INFO ] All services running
[ INFO ] StanzaFile and NodeDesc file for NSD, filesystem, and cluster setup have been saved to /usr/lpp/mmfs
folder on node: objectcluster.scale.com
[ INFO ] Successfully installed and configured protocols. 1 protocol nodes were enabled. Components installed:
Filesystem, Cluster Export Services, Object Store, Performance Monitoring, GUI, FILE AUDIT LOGGING. It took 9
minutes 45 seconds.
[root@objectcluster ansible-toolkit]#
```

And there we have it. Our object protocol was successfully deployed. We need to just do the post-deployment check as well to make sure everything is okay.

```
[root@objectcluster ansible-toolkit]# ./spectrumscale deploy -po
[ INFO ] Logging to file: /usr/lpp/mmfs/5.1.3.0/ansible-toolkit/logs/DEPLOY-POSTCHECK-20-03-2022 03:22:18.log
[ WARN ] objectcluster.scale.com is specified as both an NSD server and a protocol node. It is recommended that a
node is not both a protocol and NSD server.
Enter the secret encryption key:
Repeat the secret encryption key:
[ WARN ] Only one GUI server specified. The Graphical User Interface will not be highly available.
[ INFO ] Install toolkit will not configure file audit logging as it has been disabled.
[ INFO ] Checking state of Filesystem
[ INFO ] File systems have been created successfully
[ INFO ] Filesystem ACTIVE
[ INFO ] Checking state of Cluster Export Services
[ INFO ] Checking state of CES on all nodes
[ INFO ] Checking state of CES on all nodes
[ INFO ] Cluster Export Services ACTIVE
[ INFO ] Checking state of Object Store
[ INFO ] Running Object post-install checks
[ INFO ] Checking state of OBJ on all nodes
[ INFO ] Checking state of OBJ on all nodes
[ INFO ] Object Store ACTIVE
[ INFO ] Checking state of Performance Monitoring
[ INFO ] Running Performance Monitoring post-install checks
[ INFO ] pmcollector running on all nodes
[ INFO ] pmsensors running on all nodes
[ INFO ] Performance Monitoring ACTIVE
[ INFO ] Checking state of GUI
[ INFO ] Running Graphical User Interface post-install checks
[ INFO ] Graphical User Interface running on all GUI servers
[ INFO ] Enter one of the following addresses into a web browser to access the Graphical User Interface:
objectcluster.scale.com
[ INFO ] GUI ACTIVE
[ INFO ] SUCCESS
[ INFO ] All services running
[ INFO ] StanzaFile and NodeDesc file for NSD, filesystem, and cluster setup have been saved to /usr/lpp/mmfs
folder on node: objectcluster.scale.com
[root@objectcluster ansible-toolkit]#
```

We can verify the cluster information.

```
[root@objectcluster cli]# mmlscluster

GPFS cluster information
=====
GPFS cluster name:      objectcluster.scale.com
GPFS cluster id:       1872806154467703008
GPFS UID domain:       objectcluster.scale.com
Remote shell command:  /usr/bin/ssh
Remote file copy command: /usr/bin/scp
Repository type:       CCR

Node  Daemon node name      IP address      Admin node name      Designation
-----
1    objectcluster.scale.com  192.168.226.251 objectcluster.scale.com quorum-manager-perfmon

[root@objectcluster cli]# mmlscluster --ces

GPFS cluster information
=====
GPFS cluster name:      objectcluster.scale.com
GPFS cluster id:       1872806154467703008

Cluster Export Services global parameters
-----
Shared root directory:  /ibm/cesSharedRoot
Enabled Services:       OBJ
Log level:              0
Address distribution policy: even-coverage

Node  Daemon node name      IP address      CES IP address list
-----
1    objectcluster.scale.com  192.168.226.251  192.168.226.252

[root@objectcluster cli]#
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

If you connect to the GUI for the first time, you will see a message about having to create a user who belongs to the *SecurityAdmin* group in order to login.

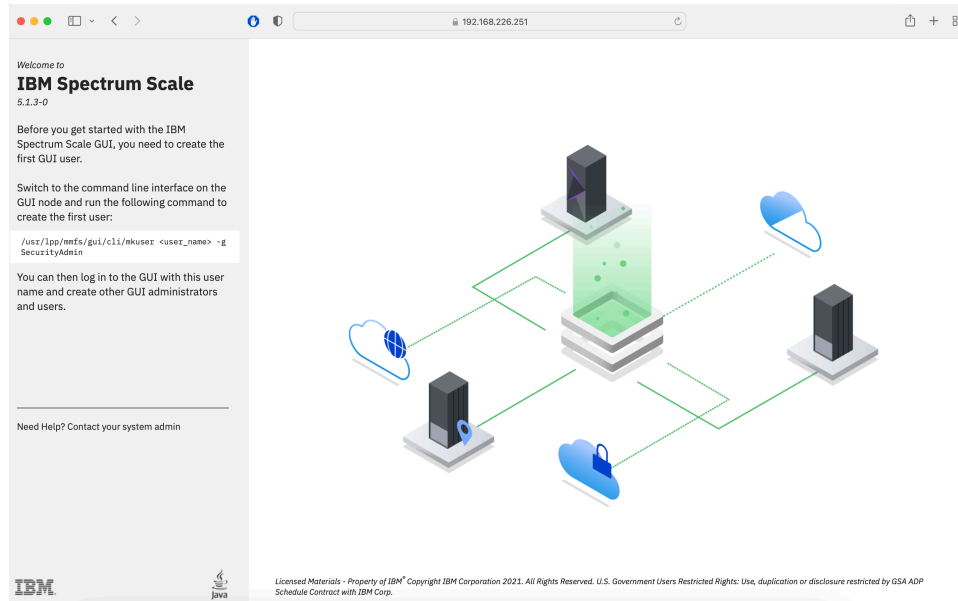


Figure 2: IBM Spectrum Scale GUI (first time access)

We can do that as follows. In our case, we are creating a GUI user called *ssadmin*.

```
[root@objectcluster etc]# /usr/lpp/mmfs/gui/cli/mkuser ssadmin -g SecurityAdmin
EFSSG1007A Enter password for User :
EFSSG0225I Repeat the password:
EFSSG0019I The user ssadmin has been successfully created.
EFSSG1000I The command completed successfully.
[root@objectcluster etc]#
```

You should then be able to login to the GUI as *ssadmin*.

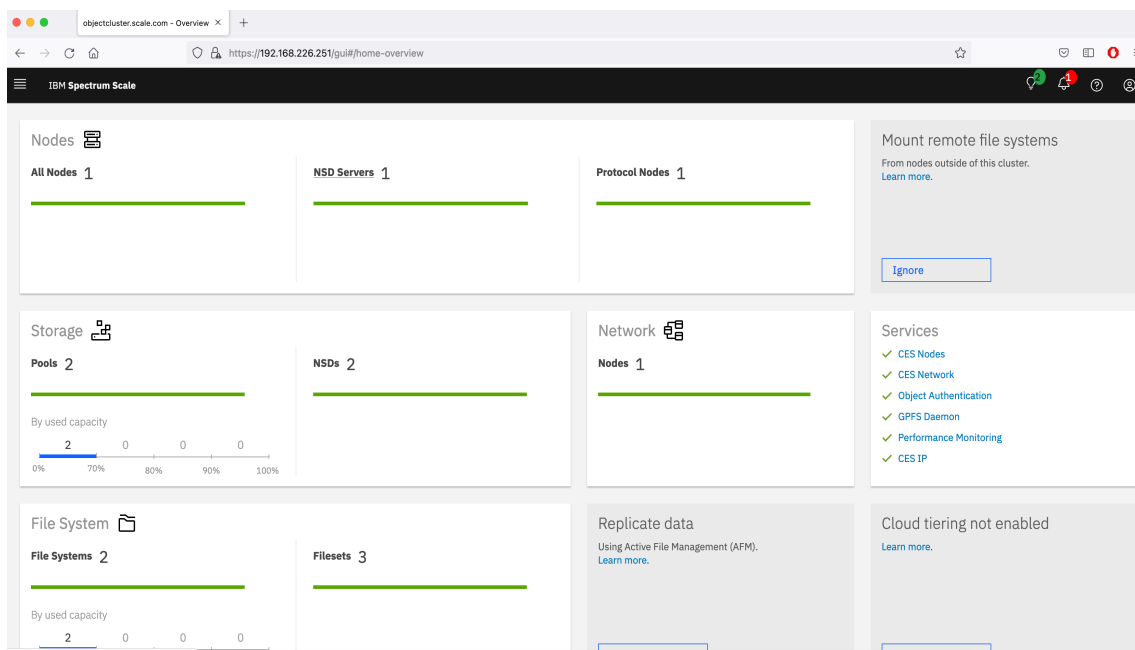


Figure 3: Spectrum Scale GUI login

Check to make sure all the required object services are running.

```
[root@objectcluster ~]# mmces service list -a -v
Enabled services: OBJ
objectcluster.scale.com: OBJ is running
objectcluster.scale.com: OBJ:openstack-swift-object-updater is running
objectcluster.scale.com: OBJ:openstack-swift-object-expirer is running
objectcluster.scale.com: OBJ:ibmobjectizer is not running
objectcluster.scale.com: OBJ:openstack-swift-object is running
objectcluster.scale.com: OBJ:openstack-swift-account is running
objectcluster.scale.com: OBJ:openstack-swift-container is running
objectcluster.scale.com: OBJ:memcached is running
objectcluster.scale.com: OBJ:openstack-swift-proxy is running
objectcluster.scale.com: OBJ:openstack-swift-object-replicator is running
objectcluster.scale.com: OBJ:openstack-swift-account-reaper is running
objectcluster.scale.com: OBJ:openstack-swift-account-auditor is running
objectcluster.scale.com: OBJ:openstack-swift-container-auditor is running
objectcluster.scale.com: OBJ:openstack-swift-container-updater is running
objectcluster.scale.com: OBJ:openstack-swift-account-replicator is running
objectcluster.scale.com: OBJ:openstack-swift-container-replicator is running
objectcluster.scale.com: OBJ:openstack-swift-object-sof is not running
objectcluster.scale.com: OBJ:postgresql-obj is running
objectcluster.scale.com: OBJ:httpd (keystone) is running
[root@objectcluster ~]#
```

And that S3 compatibility is enabled.

```
[root@objectcluster ~]# mmobj s3 list
[I] The S3 API is enabled.
[root@objectcluster ~]#
```

If you interested, you can query the current Swift proxy-server.conf settings using the command below.

```
[root@objectcluster ~]# mmobj config list --ccrfile proxy-server.conf
[DEFAULT]
bind_port = 8080
workers = auto
user = swift
log_level = ERROR
[pipeline:main]
pipeline = catch_errors healthcheck proxy-logging cache container_sync formpost tempurl authtoken s3api s3token
keystoneauth container-quotas account-quotas staticweb bulk slo dlo proxy-logging proxy-server
[app:proxy-server]
use = egg:swift#proxy
allow_account_management = true
account_autocreate = true
node_timeout = 120
conn_timeout = 120
read_affinity = r1=1
write_affinity = r1
[filter:healthcheck]
use = egg:swift#healthcheck
[filter:cache]
use = egg:swift#memcache
memcache_servers = 127.0.0.1:11211
memcache_max_connections = 8
[filter:ratelimit]
use = egg:swift#ratelimit
[filter:catch_errors]
use = egg:swift#catch_errors
[filter:tempurl]
use = egg:swift#tempurl
[filter:proxy-logging]
use = egg:swift#proxy_logging
[filter:bulk]
use = egg:swift#bulk
[filter:slo]
use = egg:swift#slo
[filter:dlo]
use = egg:swift#dlo
[filter:container-quotas]
use = egg:swift#container_quotas
[filter:account-quotas]
use = egg:swift#account_quotas
[filter:gatekeeper]
use = egg:swift#gatekeeper
[filter:container_sync]
use = egg:swift#container_sync
[filter:versioned_writes]
use = egg:swift#versioned_writes
[filter:copy]
use = egg:swift#copy
object_post_as_copy = false
[filter:authtoken]
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
paste.filter_factory = keystonemiddleware.auth_token:filter_factory
project_name = service
username = ksSwift
password = Passw0rd
auth_protocol = http
signing_dir = /tmp/keystone-signing-swift
auth_plugin = v3password
auth_url = http://127.0.0.1:35357/v3
project_domain_name = Default
user_domain_name = Default
delay_auth_decision = true
[filter:s3token]
auth_uri = http://127.0.0.1:35357/v3
use = egg:swift#s3token
[filter:staticweb]
use = egg:swift#staticweb
[filter:formpost]
use = egg:swift#formpost
[filter:keystoneauth]
reseller_admin_role = ResellerAdmin
use = egg:swift#keystoneauth
operator_roles = admin, SwiftOperator
is_admin = true
cache = swift.cache
[filter:s3api]
use = egg:swift#s3api
s3_acl = true
allow_no_owner = true
dns_compliant_bucket_names = false
[root@objectcluster ~]#
```

At this point, you might want to disable object versioning unless you explicitly want to use it. Since we don't need it (to save space), we are going to disable it as follows.

```
[root@objectcluster ~]# mmobj config change --ccrfile proxy-server.conf --section 'filter:versioned_writes' --
property allow_versioned_writes --value false
```

You need to stop and restart the object service for this change to take effect. You can do so by issuing *“mmces service stop OBJ”* and *“mmces service start OBJ”*. You can verify everything is up by issuing *“mmces service list -v”*.

We now have a working Swift environment which we need to validate. By default, Spectrum Scale deploys a single Swift domain with two projects. To check the default Swift configuration, source `openrc` from the root user's home directory and query it as follows:

```
[root@objectcluster ~]# source ./openrc
[root@objectcluster ~]# openstack domain list
+-----+-----+-----+-----+
| ID      | Name  | Enabled | Description |
+-----+-----+-----+-----+
| default | Default | True    | The default domain |
+-----+-----+-----+-----+
[root@objectcluster ~]# openstack project list
+-----+-----+-----+
| ID      | Name  |
+-----+-----+
| 434caa7f2cbe4e038b30917d956238c7 | admin |
| d9c16538de8e49269f73db9ba5f7943d | service |
+-----+-----+
[root@objectcluster ~]# openstack user list
+-----+-----+-----+
| ID      | Name  |
+-----+-----+
| 96db1ab24b464c7e8d0948bdaaa41722 | ksAdmin |
| 5195672d40344114856436a09cec5d30 | ksSwift |
+-----+-----+
[root@objectcluster ~]# openstack role list
+-----+-----+-----+
| ID      | Name  |
+-----+-----+
| 57d271a0ebb44ccd816db3827571cdbc | reader |
| ef507f8e0c1340d2b30f7e44ad5e9c53 | member |
| 25fea0bc18d04e64b79be9fecba2540e | admin |
+-----+-----+
[root@objectcluster ~]# openstack role assignment list --names
+-----+-----+-----+-----+-----+-----+-----+
| Role  | User          | Group          | Project          | Domain | System | Inherited |
+-----+-----+-----+-----+-----+-----+-----+
| admin | ksAdmin@Default |                | admin@Default    |        |        | False     |
| admin | ksSwift@Default |                | service@Default  |        |        | False     |
| admin | ksAdmin@Default |                |                  |        | all    | False     |
| reader | ksAdmin@Default |                |                  |        | all    | False     |
+-----+-----+-----+-----+-----+-----+-----+
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
+-----+-----+-----+-----+-----+-----+-----+-----+
[root@objectcluster ~]#
```

A Swift domain is a collection of projects, users and groups. Each project and group can only belong to a single domain. Projects are organizational units in the cloud to which you assign users. Project are also referred to as accounts. Users can be members of one or more projects. Roles define which actions users can perform. You typically assign roles to a user-project pair. As per the above, we have a single domain with two accounts (or projects). We have two users with each user assigned to their own project (ksAdmin/admin and ksSwift/service). Each user is assigned the role admin.

A simple test to validate the deployment is to make use of the SWIFT CLI on the protocol node. Once openrc is sourced as root, issue the following commands to create a container, upload a file to it, query the contents of the container and lastly to delete the object and container.

```
[root@objectcluster ~]# swift post swiftcontainer
[root@objectcluster ~]# swift list
swiftcontainer
[root@objectcluster ~]# swift upload swiftcontainer /etc/hosts
etc/hosts
[root@objectcluster ~]# swift list swiftcontainer
etc/hosts
[root@objectcluster ~]# swift download swiftcontainer etc/hosts
etc/hosts [auth 0.246s, headers 0.305s, total 0.305s, 0.006 MB/s]
[root@objectcluster ~]# swift delete swiftcontainer etc/hosts
etc/hosts
[root@objectcluster ~]# swift delete swiftcontainer
swiftcontainer
[root@objectcluster ~]# swift list
[root@objectcluster ~]#
```

When you source openrc, the following domain/project/user combination is used:

```
[root@objectcluster ~]# cat openrc
# Sun Mar 20 03:17:09 SAST 2022
export OS_AUTH_URL="http://127.0.0.1:35357/v3"
export OS_IDENTITY_API_VERSION=3
export OS_AUTH_VERSION=3
export OS_USERNAME='ksAdmin'
export OS_PASSWORD='Passw0rd'
export OS_USER_DOMAIN_NAME='Default'
export OS_PROJECT_NAME=admin
export OS_PROJECT_DOMAIN_NAME=Default
export OS_SYSTEM_SCOPE=all

#export OS_REGION_NAME="RegionOne"

[root@objectcluster ~]#
```

So, the above SWIFT CLI commands were using project admin in the default domain as user ksAdmin. Up till now, we have a working Swift implementation. Whilst we have the S3 API enabled, we still have additional tasks to perform before testing S3 access. Let us now configure HAProxy to enable HTTPS access to our OBJECT store (note, Swift uses HTTP by default and binds to port 8080).

Setting up HAProxy for HTTP and HTTPS access

As per our test environment, objectcluster will act as our DNS server and also as our HAProxy server. We require the use of HAProxy for Spectrum Scale Object to enable SSL termination and load balancing. Apart from enabling HTTPS access to our OBJECT store, it allows us to implement load balancing across the protocol nodes for OBJECT requests (you can also use DNS Round Robin for load balancing). Indirectly it also allows us to make use of port 80 for Object access as the default Swift installation included with IBM Spectrum Scale is bound to port 8080 and cannot be changed (even though you can change the bind port in the proxy-server.conf file to 80, the proxy-server will fail to start). Some applications are hard coded to make use of port 80 (i.e. when specifying the DNS name

even in a path statement) so this might be a useful feature to ensure compatibility across both S3 and Swift applications.

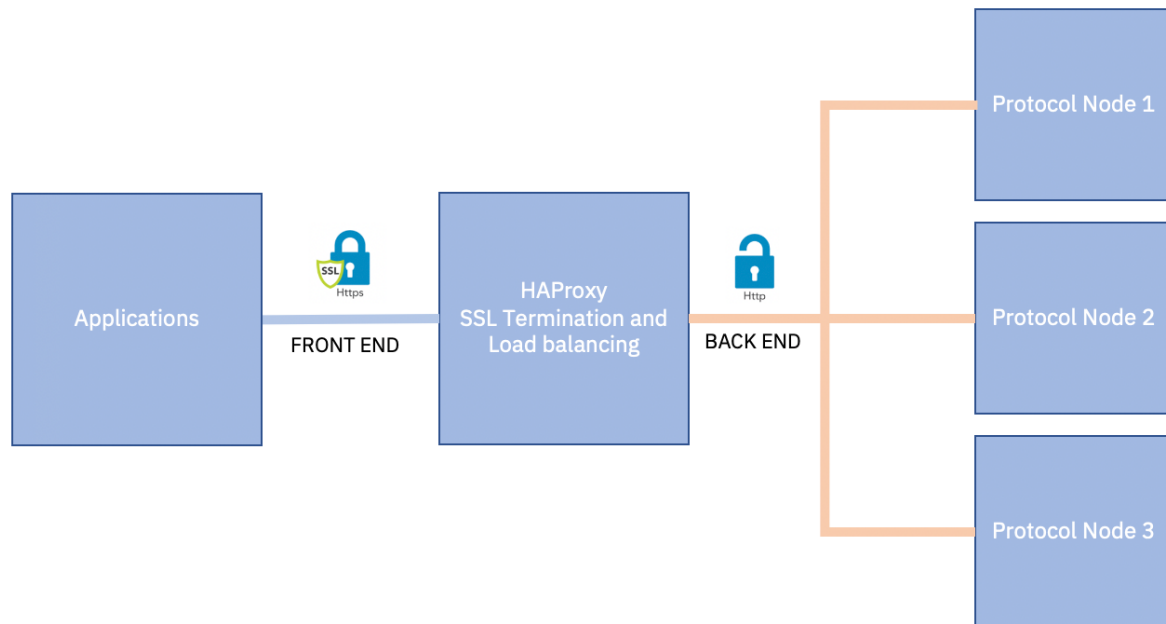


Figure 4: HAProxy typical deployment

More information on HAProxy can be found here <http://www.haproxy.org>. For the purposes of demonstrating its application with IBM Spectrum Scale Object, this document will only cover what is relevant to our use case.

Before we can configure HAProxy, we need to ensure port 80 and port 443 are free. In our test setup, since we deployed the Spectrum Scale GUI service on our protocol node, these ports are in use. Luckily Spectrum Scale 5.1.3 allows us to modify the Spectrum Scale GUI ports. If you need to change the ports, you can do so as follows. Note, we are going to use 7452 for http and 7453 for https. You can check if the ports you chose are in use with the netstat command. Don't forget to add firewall rules for the new GUI ports.

```
[root@objectcluster etc]# grep 7452 /etc/services
[root@objectcluster etc]# grep 7453 /etc/services
[root@objectcluster etc]# netstat -an | grep 7452
[root@objectcluster etc]# netstat -an | grep 7453
[root@objectcluster etc]# mkdir -p /etc/scale-gui-configuration
[root@objectcluster etc]# echo 7452 > /etc/scale-gui-configuration/scale_gui_http_port
[root@objectcluster etc]# echo 7453 > /etc/scale-gui-configuration/scale_gui_https_port
[root@objectcluster etc]# firewall-cmd --zone=public --add-port=7452/tcp --permanent
success
[root@objectcluster etc]# firewall-cmd --zone=public --add-port=7453/tcp --permanent
success
[root@objectcluster etc]# firewall-cmd --reload
success

[root@objectcluster scale-gui-configuration]# systemctl stop gpfscli
[root@objectcluster scale-gui-configuration]# systemctl start gpfscli
[root@objectcluster scale-gui-configuration]# systemctl status gpfscli
● gpfscli.service - IBM Spectrum Scale Administration GUI
   Loaded: loaded (/usr/lib/systemd/system/gpfscli.service; enabled; vendor preset: disabled)
   Active: active (running) since Sun 2022-03-20 09:21:35 SAST; 11s ago
     Process: 117231 ExecStopPost=/usr/lpp/mmfs/gui/bin-sudo/cleanupiptables (code=exited, status=0/SUCCESS)
     Process: 118271 ExecStartPre=/usr/lpp/mmfs/gui/bin-sudo/cleanupdumps (code=exited, status=0/SUCCESS)
     Process: 117849 ExecStartPre=/usr/lpp/mmfs/gui/bin-sudo/check4sudoers (code=exited, status=0/SUCCESS)
     Process: 117677 ExecStartPre=/usr/lpp/mmfs/gui/bin-sudo/check4iptables (code=exited, status=0/SUCCESS)
     Process: 117605 ExecStartPre=/usr/lpp/mmfs/gui/bin-sudo/check4pgsql (code=exited, status=0/SUCCESS)
     Process: 117599 ExecStartPre=/usr/lpp/mmfs/gui/bin-sudo/update-environment (code=exited, status=0/SUCCESS)
  Main PID: 118285 (java)
    Status: "GSS/GPFS GUI started"
      Tasks: 112 (limit: 23527)
    Memory: 307.0M (limit: 2.0G)
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
CGroup: /system.slice/gpfscli.service
└─118285 /usr/lpp/mmfs/java/jre/bin/java -XX:+HeapDumpOnOutOfMemoryError -
XX:OnOutOfMemoryError=/usr/lpp/mmfs/gui/bin/oom.sh -Dhttps.protocols=TLSv1.2>

Mar 20 09:21:35 objectcluster.scale.com systemd[1]: Started IBM Spectrum Scale Administration GUI.
Mar 20 09:21:35 objectcluster.scale.com sudo[120298]: scalemgmt : TTY=unknown ; PWD=/opt/ibm/wlp ; USER=root ;
COMMAND=/usr/lpp/mmfs/bin/mmccr fget gui.settings>
Mar 20 09:21:35 objectcluster.scale.com sudo[120313]: scalemgmt : TTY=unknown ; PWD=/opt/ibm/wlp ; USER=root ;
COMMAND=/usr/lpp/mmfs/gui/bin-sudo/chown_ccr fget >
Mar 20 09:21:35 objectcluster.scale.com sudo[120355]: scalemgmt : TTY=unknown ; PWD=/opt/ibm/wlp ; USER=root ;
COMMAND=/bin/test -e /var/lib/mmfs/gui/settings.js>
Mar 20 09:21:35 objectcluster.scale.com sudo[120371]: scalemgmt : TTY=unknown ; PWD=/opt/ibm/wlp ; USER=root ;
COMMAND=/usr/lpp/mmfs/bin/mmccr fput -c 16 _gui.se>
Mar 20 09:21:36 objectcluster.scale.com sudo[120372]: scalemgmt : TTY=unknown ; PWD=/opt/ibm/wlp ; USER=root ;
COMMAND=/usr/lpp/mmfs/bin/mmnode -N localhost
Mar 20 09:21:36 objectcluster.scale.com java[118285]: [AUDIT ] CWWKZ0001I: Application / started in 21.608
seconds.
Mar 20 09:21:40 objectcluster.scale.com sudo[120662]: scalemgmt : TTY=unknown ; PWD=/opt/ibm/wlp ; USER=root ;
COMMAND=/usr/lpp/mmfs/bin/mmccr vget gui.hide even>
Mar 20 09:21:44 objectcluster.scale.com java[118285]: [AUDIT ] CWWKF0012I: The server installed the following
features: [apiDiscovery-1.0, appSecurity-2.0, dis>
Mar 20 09:21:44 objectcluster.scale.com java[118285]: [AUDIT ] CWWKF0011I: The gpfscli server is ready to run a
smarter planet. The gpfscli server started in 3>
[root@objectcluster scale-gui-configuration]#

[root@objectcluster cli]# /usr/lpp/mmfs/gui/cli/lsnode
Hostname IP Description Role Product version Connection status
GPFS status Last updated
objectcluster.scale.com 192.168.226.251 Master GUI Node management,storage,ces 5.1.3.0 HEALTHY
HEALTHY 3/20/22 12:51 PM
EFSSG1000I The command completed successfully.
[root@objectcluster cli]#
```

Connect to the GUI using the new https port (7453) to verify its working.

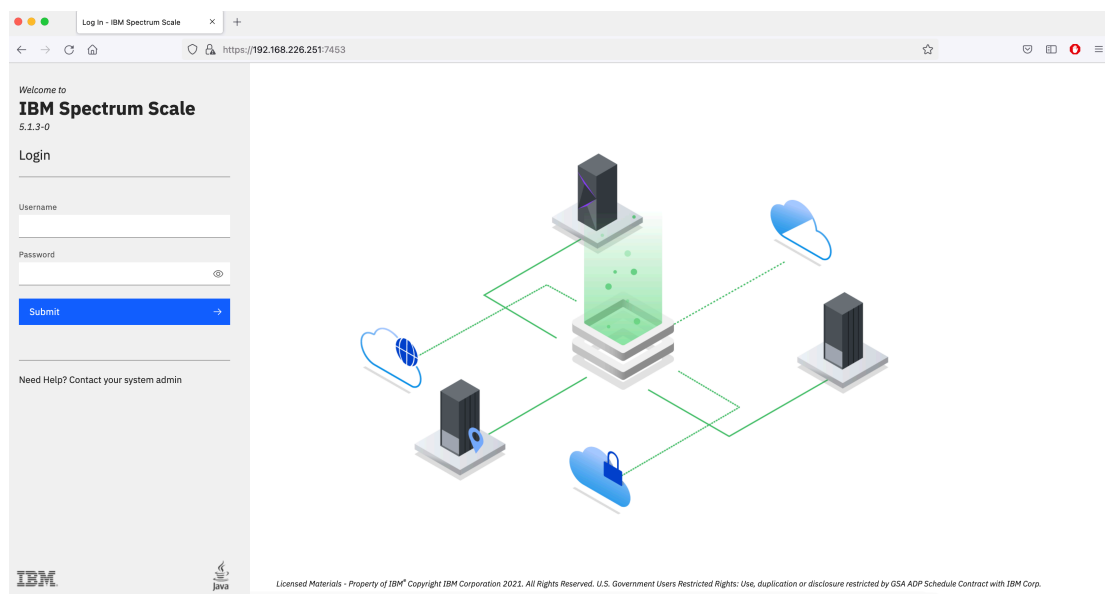


Figure 5: Access the Spectrum Scale GUI using custom ports

A pre-requisite to implement HAProxy is to ensure we have a SSL certificate and a private key that matches our domain name or IP address. Since we are using wildcard DNS with multiple domains, we need to ensure our SSL certificate matches our configuration. As an example of how to accomplish this, we will first generate a self-signed certificate that covers multiple domains and use that for the HAProxy configuration.

In order to generate a self-signed certificate to support multiple domains, create a configuration file as follows:

```
[root@objectcluster certs]# cat newcert.txt
[req]
default_bits = 2048
default_md = sha256
distinguished_name = req_distinguished_name
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
x509_extensions = v3_req
prompt = no
[req_distinguished_name]
C = ZA
ST = Gauteng
L = Johannesburg
O = Acme Ltd
OU = Storage Management
CN = s3server.scale.com
[v3_req]
keyUsage = digitalSignature, keyEncipherment, nonRepudiation, keyCertSign
extendedKeyUsage = serverAuth
subjectAltName = @alt_names
[alt_names]
DNS.1 = *.scale.com
DNS.2 = *.s3server.scale.com
[root@objectcluster certs]#
```

For primary domain (CN) you can use the CES FQDN. We can then specify additional domains in the `alt_names` stanza. This configuration file will allow us to create a single SSL certificate that covers multiple domains. With wildcard DNS enabled, we are going to be addressing buckets in the form of BUCKETNAME.s3server.scale.com where BUCKETNAME will vary based on which bucket the user/application is addressing hence the need to specify an `*` in the configuration file.

Once the configuration file is built, create the required self-signed certificate and convert it to a PEM file for use with HAProxy. Note, adjust the validity period of your certificate as required (i.e. days parameter).

```
[root@objectcluster certs]# openssl req -new -nodes -x509 -days 365 -keyout scalenew.key -out scalenew.crt -config
./newcert.txt -addext 'basicConstraints = critical,CA:TRUE'
Generating a RSA private key
.....+++++
.....+++++
writing new private key to 'scalenew.key'
-----
[root@objectcluster certs]# cat scalenew.crt scalenew.key > scalenew.pem
[root@objectcluster certs]#
```

Note, to prevent “*curl: (60) SSL certificate problem: unable to get local issuer certificate*”, we are using the Basic Constraints extension to indicate that the certificate is indeed a CA certificate (not self-signed). You can see if your certificate has been properly created as follows. We are looking specifically for CA:TRUE and also for Alternative Names to be set to wildcard DNS entries.

```
[root@objectcluster certs]# openssl x509 -noout -text -in scalenew.pem
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            73:16:26:e3:a1:52:e7:eb:37:bb:44:fd:e1:3f:ff:3f:21:91:cf:57
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = ZA, ST = Gauteng, L = Johannesburg, O = Acme Ltd, OU = Storage Management, CN =
s3server.scale.com
        Validity
            Not Before: Apr  2 09:07:36 2022 GMT
            Not After : Apr  2 09:07:36 2023 GMT
        Subject: C = ZA, ST = Gauteng, L = Johannesburg, O = Acme Ltd, OU = Storage Management, CN =
s3server.scale.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public-Key: (2048 bit)
            Modulus:
                00:b6:fe:78:1a:f2:f4:f4:8d:22:98:06:6d:3b:06:
                18:22:0b:87:0d:e8:f7:ca:a9:57:05:54:12:e9:65:
                fa:15:1b:a5:18:9d:b2:11:8c:4d:a4:15:59:3b:e7:
                09:ad:8d:ba:c7:59:1f:fa:04:d9:ea:57:fd:48:15:
                64:06:26:14:8a:e7:e0:d5:4e:48:7f:71:a5:ec:03:
                19:4d:ee:43:bb:0e:3d:f5:88:78:0e:21:d7:32:2f:
                8a:f3:f6:80:8f:95:89:26:16:e0:ba:d6:a5:48:c6:
                b7:25:b3:1c:4e:a3:a0:17:2e:5a:f1:60:1c:da:9e:
                8c:aa:73:38:99:a1:99:ab:42:b5:c1:e6:c0:38:6a:
                df:51:c9:32:88:19:c9:18:f5:21:dc:62:a4:7a:04:
                b7:eb:80:3f:7c:b5:e5:a3:2f:ed:fe:4c:7b:5c:1e:
                3d:21:cb:c8:bf:04:c4:05:c1:d0:e9:9e:5c:16:ac:
                f3:70:97:1f:02:ef:7c:43:12:bc:49:69:7c:25:f1:
                e9:7a:d6:37:71:00:e5:ef:59:39:1c:a4:b1:bd:c9:
                fd:6b:35:85:4d:e4:69:d3:a7:d4:4a:c0:58:84:7e:
```


Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
f6:a3:ac:36:64:60:84:c0:f4:a0:ba:aa:ff:dd:4a:
f4:fa:6b:e6:88:67:a4:53:74:04:20:0d:70:b7:96:
99:99
    Exponent: 65537 (0x10001)
X509v3 extensions:
    X509v3 Key Usage:
        Digital Signature, Non Repudiation, Key Encipherment, Certificate Sign
    X509v3 Extended Key Usage:
        TLS Web Server Authentication
    X509v3 Subject Alternative Name:
        DNS:*.scale.com, DNS:s3server.scale.com
    X509v3 Basic Constraints: critical
        CA:TRUE
Signature Algorithm: sha256WithRSAEncryption
79:04:bd:37:3f:0f:a2:c4:84:4b:44:3d:a0:28:68:36:03:28:
bb:72:64:60:0d:cc:1d:16:b6:fe:1b:36:b1:51:7b:9a:96:71:
29:ee:d4:f8:65:f3:fb:74:ba:0f:8d:c2:b7:be:f4:51:30:08:
c3:d5:5b:64:1c:45:14:15:95:68:e1:87:31:f8:84:f0:4e:b3:
44:ff:b9:96:eb:7f:1e:68:a0:b7:ce:86:c6:6f:a6:1e:2f:fd:
bd:33:a3:cd:6e:33:4d:f2:69:34:17:94:42:b6:ab:6f:49:43:
7e:a8:af:db:60:94:1c:d4:a2:76:db:22:e4:bc:c8:57:48:8d:
10:c1:7c:83:3f:60:d2:a9:cb:34:12:61:03:15:d5:07:b1:52:
98:65:d7:ca:a4:32:ca:02:43:f1:11:18:46:46:be:fe:b0:a4:
53:22:1c:14:79:58:8f:d0:aa:a2:14:2d:7c:8c:06:d6:1f:8a:
1e:10:d0:42:7c:80:18:52:c7:39:02:71:06:eb:c8:7f:d4:7e:
35:fd:87:d9:05:0e:c4:60:2a:2a:55:e9:5d:ea:1c:ef:64:eb:
64:ca:d4:58:dc:3a:c4:04:89:a7:d8:10:6d:63:0a:7e:75:70:
97:cd:f8:33:1c:b1:2f:b7:79:e8:64:9b:88:38:60:13:99:69:
1f:d6:fd:37
[root@objectcluster certs]
```

If we try and verify the certificate it fails as expected.

```
[root@objectcluster certs]# openssl verify scalenew.crt
C = ZA, ST = Gauteng, L = Johannesburg, O = Acme Ltd, OU = Storage Management, CN = s3server.scale.com
error 18 at 0 depth lookup: self signed certificate
error scalenew.crt: verification failed
[root@objectcluster certs]#
```

We need to import this into our OS trusted keystore.

```
[root@objectcluster certs]# cp scalenew.crt /etc/pki/ca-trust/source/anchors/
[root@objectcluster certs]# update-ca-trust enable; update-ca-trust; update-ca-trust extract
```

Now it should appear in the trusted key list.

```
[root@objectcluster certs]# trust list | more
pkcs11:id=%36%F2%BA%60%72%24%4E%BC%5B%9E%ED%85%BF%95%F7%2E%24%F8%05%9B;type=cert
type: certificate
label: s3server.scale.com
trust: anchor
category: authority
.
.
```

We should be able to verify the certificate now without issues.

```
[root@objectcluster certs]# openssl verify scalenew.crt
scalenew.crt: OK
[root@objectcluster certs]# openssl verify scalenew.pem
scalenew.pem: OK
[root@objectcluster certs]#
```

Now that we have our SSL certificate, we can go ahead and install HAProxy “yum install haproxy”. Once HAProxy is installed, we need to modify the configuration file /etc/haproxy/haproxy.cfg as follows.

```
[root@objectcluster haproxy]# cat haproxy.cfg
#-----
# Example configuration for a possible web application.  See the
# full configuration options online.
#
# https://www.haproxy.org/download/1.8/doc/configuration.txt
#
#-----
#
# Global settings
#-----
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
global
# to have these messages end up in /var/log/haproxy.log you will
# need to:
#
# 1) configure syslog to accept network log events. This is done
#    by adding the '-r' option to the SYSLOGD_OPTIONS in
#    /etc/sysconfig/syslog
#
# 2) configure local2 events to go to the /var/log/haproxy.log
#    file. A line like the following can be added to
#    /etc/sysconfig/syslog
#
#    local2.*                                /var/log/haproxy.log
#
log                127.0.0.1 local2

chroot             /var/lib/haproxy
pidfile            /var/run/haproxy.pid
maxconn            4000
user               haproxy
group              haproxy
daemon

# turn on stats unix socket
stats socket /var/lib/haproxy/stats

# utilize system-wide crypto-policies
ssl-default-bind-ciphers PROFILE=SYSTEM
ssl-default-server-ciphers PROFILE=SYSTEM

#-----
# common defaults that all the 'listen' and 'backend' sections will
# use if not designated in their block
#-----
defaults
    mode                http
    log                 global
    option               httplog
    option               dontlognull
    option http-server-close
    option forwardfor    except 127.0.0.0/8
    option               redispatch
    retries              3
    timeout http-request 10s
    timeout queue        1m
    timeout connect      10s
    timeout client       1m
    timeout server       1m
    timeout http-keep-alive 10s
    timeout check        10s
    maxconn              3000

frontend www-http
    bind 192.168.226.252:80
    reqadd X-Forwarded-Proto:\ http
    default_backend www-backend

frontend www-https
    bind 192.168.226.252:443 ssl crt /root/certs/scalenev.pem
    reqadd X-Forwarded-Proto:\ https
    default_backend www-backend

backend www-backend
    server s3server 192.168.226.252:8080 check
[root@objectcluster haproxy]#
```

We have 2 FRONTEND sections to cover HTTP and HTTPS using the same BACKEND. For HTTPS, we point it to the PEM file we generated above. If we have more than one protocol node, we would add multiple server parameters in the BACKEND section and “*balance roundrobin*” to enable load balancing across our protocol nodes. Once the above file is created, enable and start HAProxy.

```
[root@objectcluster haproxy]# systemctl enable haproxy
Created symlink /etc/systemd/system/multi-user.target.wants/haproxy.service →
/usr/lib/systemd/system/haproxy.service.

[root@objectcluster haproxy]# systemctl start haproxy
[root@objectcluster haproxy]# systemctl status haproxy
● haproxy.service - HAProxy Load Balancer
   Loaded: loaded (/usr/lib/systemd/system/haproxy.service; enabled; vendor preset: disabled)
   Active: active (running) since Mon 2022-03-21 01:26:06 SAST; 10s ago
     Process: 111795 ExecStartPre=/usr/sbin/haproxy -f $CONFIG -c -q $OPTIONS (code=exited, status=0/SUCCESS)
    Main PID: 111798 (haproxy)
       Tasks: 2 (limit: 23527)
      Memory: 7.0M
     CGroup: /system.slice/haproxy.service
            └─111798 /usr/sbin/haproxy -Ws -f /etc/haproxy/haproxy.cfg -p /run/haproxy.pid
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
└─111800 /usr/sbin/haproxy -Ws -f /etc/haproxy/haproxy.cfg -p /run/haproxy.pid
Mar 21 01:26:06 objectcluster.scale.com systemd[1]: Starting HAProxy Load Balancer...
Mar 21 01:26:06 objectcluster.scale.com haproxy[111798]: [WARNING] 079/012606 (111798) : parsing
[/etc/haproxy/haproxy.cfg:70] : 'bind 192.168.226.252:443' :
Mar 21 01:26:06 objectcluster.scale.com haproxy[111798]: unable to load default 1024 bits DH parameter for
certificate '/etc/haproxy/scale.pem'.
Mar 21 01:26:06 objectcluster.scale.com haproxy[111798]: , SSL library will use an automatically generated DH
parameter.
Mar 21 01:26:06 objectcluster.scale.com haproxy[111798]: [WARNING] 079/012606 (111798) : Setting tune.ssl.default-
dh-param to 1024 by default, if your workload permits it you should se>
Mar 21 01:26:06 objectcluster.scale.com systemd[1]: Started HAProxy Load Balancer.
[root@objectcluster haproxy]#
```

Note the WARNING in the HAProxy status. Our certificate is 2048 bits so we need to set the *tune.ssl.default-dh-param* to 2048 in the HAProxy configuration file to get rid of this error.

```
.
.
.
# utilize system-wide crypto-policies
ssl-default-bind-ciphers PROFILE=SYSTEM
ssl-default-server-ciphers PROFILE=SYSTEM
tune.ssl.default-dh-param 2048
.
.
```

Restart HAProxy and the error should be gone.

```
[root@objectcluster haproxy]# systemctl restart haproxy
[root@objectcluster haproxy]# systemctl status haproxy
● haproxy.service - HAProxy Load Balancer
   Loaded: loaded (/usr/lib/systemd/system/haproxy.service; enabled; vendor preset: disabled)
   Active: active (running) since Mon 2022-03-21 01:29:19 SAST; 2s ago
     Process: 114687 ExecStartPre=/usr/sbin/haproxy -f $CONFIG -c -q $OPTIONS (code=exited, status=0/SUCCESS)
    Main PID: 114691 (haproxy)
       Tasks: 2 (limit: 23527)
      Memory: 6.9M
     CGroup: /system.slice/haproxy.service
             └─114691 /usr/sbin/haproxy -Ws -f /etc/haproxy/haproxy.cfg -p /run/haproxy.pid
               └─114694 /usr/sbin/haproxy -Ws -f /etc/haproxy/haproxy.cfg -p /run/haproxy.pid

Mar 21 01:29:19 objectcluster.scale.com systemd[1]: haproxy.service: Succeeded.
Mar 21 01:29:19 objectcluster.scale.com systemd[1]: Stopped HAProxy Load Balancer.
Mar 21 01:29:19 objectcluster.scale.com systemd[1]: Starting HAProxy Load Balancer...
Mar 21 01:29:19 objectcluster.scale.com systemd[1]: Started HAProxy Load Balancer.
[root@objectcluster haproxy]#
```

You can check if HAProxy is listening for connections on port 80 and 443.

```
[root@objectcluster haproxy]# netstat -an | grep LISTEN | more
.
.
tcp        0      0 192.168.226.252:443    0.0.0.0:*              LISTEN
.
.
tcp        0      0 192.168.226.252:80     0.0.0.0:*              LISTEN
```

And use curl to check if we can communicate to those ports and also that we don't get the curl SSL error.

```
[root@objectcluster t]# curl http://s3server.scale.com
<html><h1>Not Found</h1><p>The resource could not be found.</p></html>[root@objectcluster t]#
[root@objectcluster t]# curl https://s3server.scale.com
<html><h1>Not Found</h1><p>The resource could not be found.</p></html>[root@objectcluster t]#
[root@objectcluster t]#
```

So, both wildcard DNS and HAProxy are setup to correspond to our test environment. Before we can finally complete the last step and configure the S3 compliant Object store in IBM Spectrum Scale, we need to verify that our HAProxy is working. You can perform this step using the Swift CLI or you can use CURL. To make use of CURL, we first need to get an authentication token. We will do so for one of our Swift Projects called admin (note, we could have also used HAProxy for Keystone authentication though for this example, we only used it for HTTP/HTTPS so need to bypass the HAProxy to request a token). Below is a script called `get_token.sh` that makes it easy to obtain a v3 authentication token.

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

You just need to populate it with info from the openrc file. Note, 35357 is the internal Keystone port - best practice would be to use the public port which is 5000.

```
[root@objectcluster scripts]# cat get_token.sh
#!/bin/sh

#####
# Get this from "source openrc" output and populate these variables
#####

export OS_PROJECT_NAME="admin"
export OS_PROJECT_DOMAIN_NAME="Default"
export OS_USERNAME="ksAdmin"
export OS_USER_DOMAIN_NAME="Default"
export OS_PASSWORD="Passw0rd"
export OS_AUTH_URL="http://192.168.226.252:35357/v3/auth/tokens"

#####
# Temporary JSON file to use for the CURL
#####

JSONFILE=/tmp/auth.json

#####
# Construct the JSON file to use for the CURL#
#####

cat >${JSONFILE} <<EOF
{
  "auth": {
    "identity": {
      "methods": ["password"],
      "password": {
        "user": {
          "name": "${OS_USERNAME}",
          "domain": {"name": "${OS_USER_DOMAIN_NAME}"},
          "password": "${OS_PASSWORD}"
        }
      }
    },
    "scope": {
      "project": {
        "name": "${OS_PROJECT_NAME}",
        "domain": {"name": "${OS_PROJECT_DOMAIN_NAME}"}
      }
    }
  }
}
EOF

#####
# Do the CURL
#####
curl -i -H "Content-Type: application/json" -d @$JSONFILE $OS_AUTH_URL

[root@objectcluster scripts]#
```

Run the script to get the X-Auth-Token ID (it is the value X-Subject-Token in the output).

```
[root@objectcluster scripts]# ./get_token.sh
HTTP/1.1 201 CREATED
Date: Mon, 21 Mar 2022 02:53:00 GMT
Server: Apache
Content-Length: 1851
X-Subject-Token: gAAAAABiN-iM4tDj0GoJzmpIbUNQ_27wGT01MBdf-bM1ZQjUBWdyBx18HQZBgEaaBULM-5wPdPm8tp44CRxuNlqWY5p1ZrrsYRaA5uTIammGEB0C6paqg2ETy1fvc8MZRwhWx1k0LdGxTqeG-VYZB7TxXfa8_hLL89eT5Pf4cZo9hs4UbJHH8T8
Vary: X-Auth-Token
x-openstack-request-id: req-d9c754a8-4f7e-4754-a38d-882e63b0f5dc
Content-Type: application/json

{"token": {"methods": ["password"], "user": {"domain": {"id": "default", "name": "Default"}, "id": "96db1ab24b464c7e8d0948bdaaa41722", "name": "ksAdmin", "password_expires_at": null, "audit_ids": ["D_04YT8SSwKYcithvEI_gA"], "expires_at": "2022-04-20T02:53:00.000000Z", "issued_at": "2022-03-21T02:53:00.000000Z", "project": {"domain": {"id": "default", "name": "Default"}, "id": "434caa7f2cbe4e038b30917d956238c7", "name": "admin", "is_domain": false, "roles": [{"id": "ef507f8e0c1340d2b30f7e44ad5e9c53", "name": "member"}, {"id": "57d271a0ebb44ccd816db3827571cdcb", "name": "reader"}, {"id": "25fea0bc18d04e64b79be9fecba2540e", "name": "admin"}], "catalog": [{"endpoints": [{"id": "0118383f15c04e6dbaad7c37ae174573", "interface": "public", "region_id": null, "url": "http://s3server.scale.com:5000/", "region": null}, {"id": "e8c70c04ac0d4b9897f53cda2ed3501e", "interface": "internal", "region_id": null, "url": "http://s3server.scale.com:35357/", "region": null}, {"id": "e3c2bd46af864346bc3a93795eb43d60", "interface": "admin", "region_id": null, "url": "http://s3server.scale.com:35357/", "region": null}], "id": "99789591b37d46dcb51964540b0535ea", "type": "identity",
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
"name": "keystone"}, {"endpoints": [{"id": "86263b30adbf41139367febec288d0ca", "interface": "public", "region_id": "RegionOne", "url": "http://s3server.scale.com:8080/v1/AUTH_434caa7f2cbe4e038b30917d956238c7", "region": "RegionOne"}, {"id": "00463af2be364dcb884aef22f29e0bc", "interface": "internal", "region_id": "RegionOne", "url": "http://s3server.scale.com:8080/v1/AUTH_434caa7f2cbe4e038b30917d956238c7", "region": "RegionOne"}, {"id": "253746ebec024e7b964b2ca68930f7be", "interface": "admin", "region_id": "RegionOne", "url": "http://s3server.scale.com:8080", "region": "RegionOne"}], "id": "ee8f841de5f74036b965d8ba27e53f96", "type": "object-store", "name": "swift"}]]}[root@objectcluster scripts]#
```

Now we can use curl to test HAProxy. We will create 2 containers, one called *httpcontainer* and one call *httpscontainer* by issuing a POST command and using the X-Auth-Token ID we obtained above.

```
[root@objectcluster scripts]# curl -i
http://s3server.scale.com/v1/AUTH_434caa7f2cbe4e038b30917d956238c7/httpcontainer?format=json -X PUT -H "X-Auth-Token: gAAAAABiN-iM4tDj0GoJzmpIbUNQ_27wGT01MBdf-bM1ZQjUBWdyBx18HQZBgEaaBULM-5wPdPm8tp44CRxuNlqWY5p1ZrrsYRaA5uTIammGEB0C6paqg2ETylfvc8MZRwhWx1k0LdGxTqeG-VYZB7TxXfa8_hLL89eT5Pf4cZo9hs4UbJHH8T8"
HTTP/1.1 201 Created
Content-Type: text/html; charset=UTF-8
Content-Length: 0
X-Trans-Id: txb7ab15cb82f544729c4ce-006237e944
X-OpenStack-Request-Id: txb7ab15cb82f544729c4ce-006237e944
Date: Mon, 21 Mar 2022 02:56:04 GMT

[root@objectcluster scripts]#

[root@objectcluster scripts]# curl -i
https://s3server.scale.com/v1/AUTH_434caa7f2cbe4e038b30917d956238c7/httpscontainer?format=json -X PUT -H "X-Auth-Token: gAAAAABiN-iM4tDj0GoJzmpIbUNQ_27wGT01MBdf-bM1ZQjUBWdyBx18HQZBgEaaBULM-5wPdPm8tp44CRxuNlqWY5p1ZrrsYRaA5uTIammGEB0C6paqg2ETylfvc8MZRwhWx1k0LdGxTqeG-VYZB7TxXfa8_hLL89eT5Pf4cZo9hs4UbJHH8T8"
HTTP/1.1 201 Created
Content-Type: text/html; charset=UTF-8
Content-Length: 0
X-Trans-Id: tx3960a3c558e04409ab9f8-006237e980
X-OpenStack-Request-Id: tx3960a3c558e04409ab9f8-006237e980
Date: Mon, 21 Mar 2022 02:57:04 GMT

[root@objectcluster scripts]#
```

Notice for the HTTP request we didn't have to specify port 8080 (HAProxy is configured to route port 80 to port 8080 on our protocol node). We can verify this by using the Swift CLI (remember to source the openrc file or set the environment variables manually if not running this from a protocol node).

```
[root@objectcluster ~]# source openrc
[root@objectcluster ~]# swift list
httpcontainer
httpscontainer
[root@objectcluster ~]#
```

We have just verified that we HAProxy is working and that any requests to port 80 or 443 will be redirected to port 8080 on our protocol node. In a real-world scenario, you would most likely also add keystone authentication to HAProxy. On our test setup, since HAProxy is running on the protocol node, there would be a conflict of port numbers (5000/35357) if we implemented this. For reference, here are the stanzas you would add to */etc/haproxy/haproxy.cfg* for keystone authentication.

```
frontend keystone-http
    bind <HAPROXY_IP>:35357
    reqadd X-Forwarded-Proto:\ http
    default_backend keystone-backend

frontend keystone-https
    bind <HAPROXY_IP>:5000 ssl crt /root/certs/scalenew.pem
    reqadd X-Forwarded-Proto:\ https
    default_backend keystone-backend

backend keystone-backend
    balance roundrobin
    server ces1 <CES_IP1>:35357 check inter 5s
    server ces2 <CES_IP2>:35357 check inter 5s
    server ces3 <CES_IP3>:35357 check inter 5s
```

Note, you probably do not want to expose the internal Keystone port which is 35357. Remove this if required.

Setting up the S3 API Emulation on IBM Spectrum Scale Object

As a first step, it would be prudent to create a new project on our OpenStack Swift implementation specifically to use for S3 access. As detailed earlier, the default configuration after enabling IBM Spectrum Scale Object results in a single Swift domain with 2 default projects (admin/service). A project holds containers and objects so we will use logical separation between Swift containers and S3 containers (you can in theory mix them).

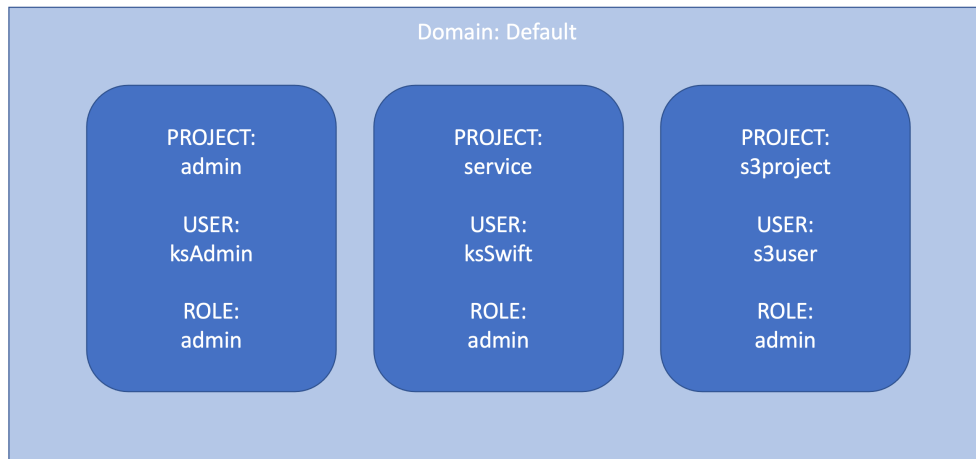


Figure 6: Swift configuration to support S3 emulation

By default, we support S3 V2 protocol. Also, the default region is set to us-east-1 for Spectrum Scale V5.1+ (previously it was set to US) but can be changed (the region needs to match the client applications if using V4 protocol). The other point to note is that bucket creation date is always reported as “2009-02-03 10:45:09” due to limitations in Swift. To get the actual creation date you would need to query this via the Swift protocol.

Since Swift and AWS S3 use different authentication methods, we have to create EC2 credentials as well (s3user in Figure 6). More information about how to do this can be found here:

<https://www.ibm.com/docs/en/spectrum-scale/5.1.3?topic=storage-configuring-openstack-ec2-credentials>

Some limitations for S3 emulation and more information are documented here:

<https://www.ibm.com/docs/en/spectrum-scale/5.1.3?topic=storage-how-manage-openstack-s3-api>

As documented in the URL above, to use the S3 API in Spectrum Scale, you must have a role that is defined for the swift project. Let's start by creating a new project/user/role for S3 use before binding them to the EC2 credentials.

```
[root@objectcluster ~]# source ./openrc
[root@objectcluster ~]# openstack project create --domain default --description "S3 Project" --enable s3project
+-----+-----+
| Field | Value |
+-----+-----+
| description | S3 Project |
| domain_id | default |
| enabled | True |
| id | 224f53da12e04b7893783821a34c171f |
| is_domain | False |
| name | s3project |
| options | {} |
+-----+-----+
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
| parent_id | default |
| tags      | []      |
+-----+
```

Now we need to create a dedicated user and assign them to our s3project.

```
[root@objectcluster ~]# openstack user create --domain default --project s3project --password Passw0rd --
description "S3 user" --enable s3user
+-----+
| Field | Value |
+-----+
| default_project_id | 224f53da12e04b7893783821a34c171f |
| description        | S3 user |
| domain_id          | default |
| enabled            | True |
| id                 | 7ed3923b2df64a5d94af66f1cfd6512c |
| name              | s3user |
| options            | {} |
| password_expires_at | None |
+-----+
```

We need to create and assign roles to the new user or we can assign an existing role. For now, we will just assign the admin role to the s3user.

```
[root@objectcluster ~]# openstack role add --project s3project --user s3user admin
```

We can check if everything is fine by checking the role assignment:

```
[root@objectcluster ~]# openstack role assignment list --names
+-----+
| Role | User | Group | Project | Domain | System | Inherited |
+-----+
| admin | ksAdmin@Default | | admin@Default | | | False |
| admin | ksSwift@Default | | service@Default | | | False |
| admin | s3user@Default | | s3project@Default | | | False |
| admin | ksAdmin@Default | | | | all | False |
| reader | ksAdmin@Default | | | | all | False |
+-----+
```

We can now create the S3 credentials (the equivalent of the S3 access and secret keys). We will use our newly created project and user for the credentials as follows. Make sure to do this as the s3user by copying the openrc file and modifying it to match the s3user information.

```
[root@objectcluster ~]# cat openrc_s3
# Sun Mar 20 03:17:09 SAST 2022
export OS_AUTH_URL="http://127.0.0.1:35357/v3"
export OS_IDENTITY_API_VERSION=3
export OS_AUTH_VERSION=3
export OS_USERNAME='s3user'
export OS_PASSWORD='Passw0rd'
export OS_USER_DOMAIN_NAME='Default'
export OS_PROJECT_NAME=s3project
export OS_PROJECT_DOMAIN_NAME=Default
export OS_SYSTEM_SCOPE=all

#export OS_REGION_NAME="RegionOne"
[root@objectcluster ~]# source openrc_s3
[root@objectcluster ~]# openstack credential create --type ec2 --project s3project s3user '{"access": "s3access",
"secret": "s3secret"}'
+-----+
| Field | Value |
+-----+
| blob | {"access": "s3access", "secret": "s3secret"} |
| id | 171960ecd22ee0c26af9dc4e55a8a8a99ca031ac35c52229235a41cafb5760d4 |
| project_id | 224f53da12e04b7893783821a34c171f |
| type | ec2 |
| user_id | 7ed3923b2df64a5d94af66f1cfd6512c |
+-----+
```

You can verify if the S3 credentials were properly created as follows.

```
[root@objectcluster ~]# openstack credential list --type ec2
+-----+
| ID | Project ID | Type | User ID | Data |
+-----+
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
| 171960ecd22ee0c26af9dc4e55a8a8a99ca031ac35c5229235a41cafb5760d4 | ec2 | 7ed3923b2df64a5d94af66f1cfd6512c |
{"access": "s3access", "secret": "s3secret"} | 224f53da12e04b7893783821a34c171f |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
[root@objectcluster ~]# openstack ec2 credentials list
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Access | Secret | Project ID | User ID |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| s3access | s3secret | 224f53da12e04b7893783821a34c171f | 7ed3923b2df64a5d94af66f1cfd6512c |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+

```

Note: You can use the Spectrum Scale GUI to manage accounts, users and roles. You can also monitor object performance metrics. You need to first add an Object Administrator via the Services panel of the GUI. You can add an administrative user that has the Keystone administrator role (admin) from any project that belongs to the default domain.

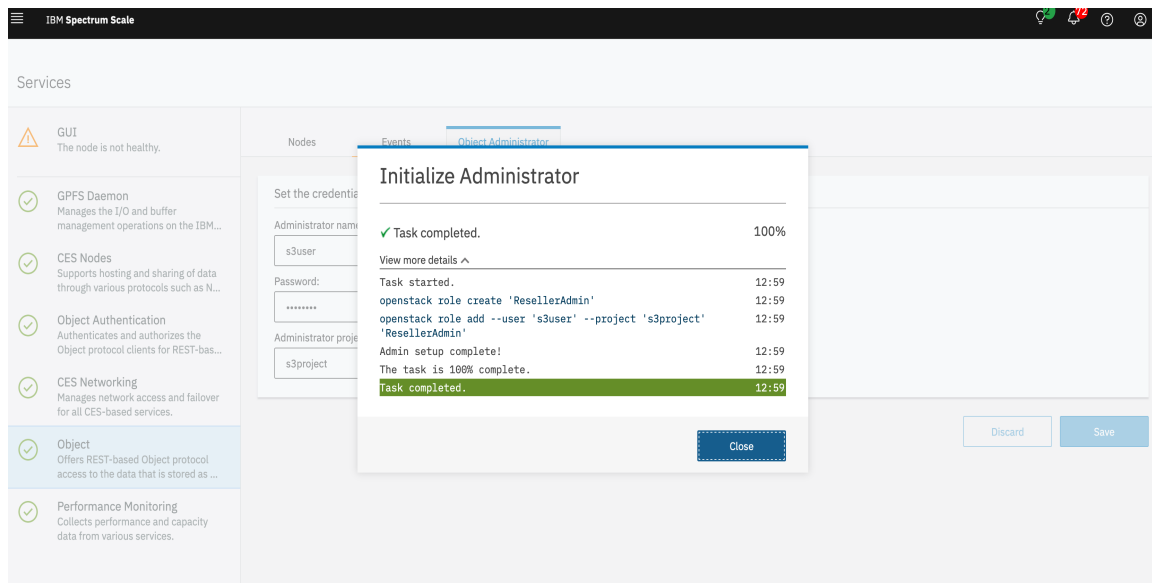


Figure 7: Adding an object administrator on the Spectrum Scale GUI

We can now connect to the Spectrum Scale Object store using any S3 client with the access key “s3access” and secret “s3secret”. We can test this using any S3 client (e.g. AWS CLI/s3curl/s3cmd/S3Browser). For the purposes of this document we will use the AWS CLI and s3cmd which is installed on the object client VM called scalecluster.

The latest version of AWS CLI can be found here: <https://aws.amazon.com/cli/>. Once AWS CLI is installed we need to configure it as follows:

```
[root@scalecluster ~]# aws configure
AWS Access Key ID [None]: s3access
AWS Secret Access Key [None]: s3secret
Default region name [None]: af-south-1
Default output format [None]:
[root@scalecluster ~]#
```

AWS CLI uses S3 V4 protocol by default. Therefore, your region needs to match the default region on Swift which is US. Alternatively, if you need to modify the region to suit a specific application, you can do so as follows:

```
[root@objectcluster ~]# mmobj config change --ccrfile proxy-server.conf --section filter:s3api --property location
--value af-south-1
```

You need to stop and start Object services (mmces service stop/start OBJ) or restart Spectrum Scale if you make modifications to the Swift proxy-server config file. Before testing with AWS CLI, you need

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

to copy the self-signed certificate PEM file we created earlier for use by HAProxy and import it into your OS certificate trust store on your client VM. If you don't do this, you need to issue the `--no-verify-ssl` with AWS CLI or use `--insecure` with curl for example.

```
[root@scalecluster ~]# scp s3server.scale.com:/root/certs/scalenew.pem /etc/pki/ca-trust/source/anchors/
root@s3server.scale.com's password:
scalnew.pem
100% 3144 3.0MB/s 00:00
[root@scalecluster ~]# update-ca-trust enable; update-ca-trust; update-ca-trust extract
[root@scalecluster ~]# trust list | more
pkcs11:id=%AC%84%A9%34%01%F3%4E%70%6D%CE%31%E3%BE%75%6E%C7%F5%D7%75%94;type=cert
  type: certificate
  label: s3server.scale.com
  trust: anchor
  category: authority
:
.
.
[root@scalecluster anchors]# openssl verify /etc/pki/ca-trust/source/anchors/scalenew.pem
/etc/pki/ca-trust/source/anchors/scalenew.pem: OK
[root@scalecluster anchors]#
```

Once that is done, you need to set an AWS CLI environment variable to point to this certificate.

```
[root@objectcluster .aws]# export AWS_CA_BUNDLE=/etc/pki/ca-trust/source/anchors/scalenew.pem
```

Now we can test basic S3 API compatibility using AWS CLI as follows:

Create Bucket:

```
[root@scalecluster ~]# aws s3api --endpoint-url https://s3server.scale.com create-bucket --bucket awsbucket
{
  "Location": "/awsbucket"
}
```

List Bucket:

```
[root@scalecluster ~]# aws s3api --endpoint-url https://s3server.scale.com list-buckets
{
  "Buckets": [
    {
      "Name": "awsbucket",
      "CreationDate": "2009-02-03T16:45:09+00:00"
    }
  ],
  "Owner": {
    "DisplayName": "s3project:s3user",
    "ID": "s3project:s3user"
  }
}
```

Notice the `CreationDate` value reported. This is due to a limitation in Swift as noted earlier.

Put Object:

```
[root@scalecluster ~]# aws s3api --endpoint-url https://s3server.scale.com put-object --bucket awsbucket --key
file1 --body ./testfile
{
  "ETag": "\"54fb6627dbaa37721048e4549db3224d\""
}
```

List Objects:

```
[root@scalecluster ~]# aws s3api --endpoint-url https://s3server.scale.com list-objects --bucket awsbucket
{
  "Contents": [
    {
      "Key": "file1",
      "LastModified": "2022-03-22T14:07:28.821Z",
      "ETag": "\"54fb6627dbaa37721048e4549db3224d\"",
      "Size": 158,
      "StorageClass": "STANDARD",
      "Owner": {
        "DisplayName": "s3project:s3user",
        "ID": "s3project:s3user"
      }
    }
  ]
}
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
}  
}  
]  
}
```

Get Object:

```
[root@scalecluster ~]# aws s3api --endpoint-url https://s3server.scale.com get-object --bucket awsbucket --key  
file1 newfile1  
{  
  "LastModified": "2022-04-02T09:31:29+00:00",  
  "ContentLength": 158,  
  "ETag": "\"54fb6627dbaa37721048e4549db3224d\"",  
  "ContentType": "application/octet-stream",  
  "Metadata": {}  
}  
[root@scalecluster ~]# ls newfile1  
newfile1
```

Delete Object / Bucket:

```
[root@scalecluster ~]# aws s3api --endpoint-url https://s3server.scale.com delete-object --bucket awsbucket --key  
file1  
[root@scalecluster ~]# aws s3api --endpoint-url https://s3server.scale.com delete-bucket --bucket awsbucket  
  
[root@scalecluster ~]# aws s3api --endpoint-url https://s3server.scale.com list-buckets  
{  
  "Buckets": [],  
  "Owner": {  
    "DisplayName": "s3project:s3user",  
    "ID": "s3project:s3user"  
  }  
}
```

If we use s3 (AWS CLI s3api commands are a 1:1 mapping with the low level S3 API. AWS CLI s3 commands offer higher-level functionality).

```
[root@scalecluster ~]# aws s3 ls --endpoint-url https://s3server.scale.com  
2009-02-03 18:45:09 awsbucket  
[root@scalecluster ~]# aws s3 cp testfile s3://awsbucket --endpoint-url https://s3server.scale.com  
upload: ./testfile to s3://awsbucket/testfile  
[root@scalecluster ~]# aws s3 ls s3://awsbucket --endpoint-url https://s3server.scale.com  
2022-04-02 11:40:50      158 testfile  
[root@scalecluster ~]#
```

If you issue the above AWS CLI commands with the `-debug` flag, then you will notice that it is defaulting to Path-style addressing.

```
.  
.  
2022-04-02 11:42:09,081 - MainThread - botocore.hooks - DEBUG - Event before-sign.s3.ListObjectsV2: calling handler  
<bound method S3EndpointSetter.set_endpoint of <botocore.utils.S3EndpointSetter object at 0x7ff154f4bd60>>  
2022-04-02 11:42:09,082 - MainThread - botocore.utils - DEBUG - Using S3 path style addressing.  
2022-04-02 11:42:09,082 - MainThread - botocore.auth - DEBUG - Calculating signature using v4 auth.  
2022-04-02 11:42:09,082 - MainThread - botocore.auth - DEBUG - CanonicalRequest:  
GET  
/awsbucket  
delimiter=%2F&encoding-type=url&list-type=2&prefix=  
host:s3server.scale.com  
.  
.
```

Now that we have confirmed basic compatibility using Path-style addressing, if we try and use Virtual-hosted style bucket addressing it should fail. To demonstrate this, we will use s3cmd which is a popular and powerful S3 client with virtual-hosted style bucket addressing enabled by default. Using s3cmd fails as expected. We need to enable this on IBM Spectrum Scale first in order for this to work. The latest version of S3CMD can be found here: <https://s3tools.org/s3cmd>. Before we begin, we need to make sure the following options are specified at installation (i.e. when you issue the “s3cmd –configure” command).

```
.s3cfg important options:  
access_key = s3access  
bucket_location = af-south-1
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
check_ssl_certificate = True
check_ssl_hostname = True
ca_certs_file = /etc/ssl/certs/scale.pem
host_base = s3server.scale.com
host_bucket = %(bucket)s.s3server.scale.com
use_https = True
multipart_chunk_size_mb = 15
secret_key = s3secret
```

```
[root@s3client ~]# s3cmd put hosts s3://awsbucket
upload: 'hosts' -> 's3://awsbucket/hosts' [1 of 1]
 376 of 376 100% in 0s 13.83 kB/s done
ERROR: S3 error: 400 (MalformedXML): The XML you provided was not well-formed or did not validate against our published schema.

[root@s3client ~]# s3cmd mb s3://awsbucket3
ERROR: S3 error: 405 (MethodNotAllowed): The specified method is not allowed against this resource.
```

As expected, these requests fail. If we issue these commands with `--debug` option we can see `s3cmd` is trying to use Virtual-hosted style bucket naming which is currently not enabled. Our underlying OpenStack Swift implementation is not able to differentiate the bucket name in the access URL. Also, we haven't configured `s3cmd` to use an explicit certificate file as our OS trust store already trusts our Object endpoint.

```
.
.
DEBUG: ConnMan.get(): creating new connection: https://awsbucket.s3server.scale.com
DEBUG: Using ca_certs_file None
DEBUG: Using ssl_client_cert_file None
DEBUG: Using ssl_client_key_file None
DEBUG: httplib.HTTPSConnection() has both context and check_hostname
DEBUG: non-proxied HTTPSConnection(awsbucket.s3server.scale.com, None)
.
.
```

Enabling Virtual-hosted Style Bucket Addressing

IBM Spectrum Scale V5.1.3 has the OpenStack Swift V2.23.1 (TRAIN) release bundled with it. This version uses the S3API middleware to emulate S3. A sample `proxy-server.conf` file can be found here <https://github.com/openstack/swift/blob/train-em/etc/proxy-server.conf-sample>. If you look at some of the options for the S3API middleware, we are especially interested in the following:

```
.
.
# Set whether to enforce DNS-compliant bucket names. Note that S3 enforces
# these conventions in all regions except the US Standard region.
# dns_compliant_bucket_names = true
.
# Set a region name of your Swift cluster. Note that Swift3 doesn't choose a
# region of the newly created bucket actually. This value is used only for the
# GET Bucket location API.
# location = US
.
# Specify a host name of your Swift cluster. This enables virtual-hosted style
# requests.
# storage_domain =
.
.
```

We need to enable DNS compliant bucket names and need to set `storage_domain` to our CES IP FQDN (for our test environment this is `s3server.scale.com`). Using `s3server.scale.com` will allow the middleware to treat any prefix to this value as the bucket name.

Let's query the existing Swift3 middleware values from our `proxy-server`.

```
[root@objectcluster ~]# mmobj config list --ccrfile proxy-server.conf --section filter:s3api
[filter:s3api]
use = egg:swift#s3api
s3_acl = true
allow_no_owner = true
dns_compliant_bucket_names = false
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
location = af-south-1
[root@objectcluster ~]#
```

We need to change `dns_compliant_bucket_names` to true and add `storage_domain`. This is done as follows:

```
[root@objectcluster ~]# mmobj config change --ccrfile proxy-server.conf --section filter:s3api --property
dns_compliant_bucket_names --value true
[root@objectcluster ~]# mmobj config change --ccrfile proxy-server.conf --section filter:s3api --property
storage domain --value s3server.scale.com
[root@objectcluster ~]# mmobj config list --ccrfile proxy-server.conf --section filter:s3api
[filter:s3api]
use = egg:swift#s3api
s3_acl = true
allow_no_owner = true
dns_compliant_bucket_names = true
location = af-south-1
storage_domain = s3server.scale.com
[root@objectcluster ~]#
```

We need to stop and start the OBJECT services for the changes to take effect.

```
[root@objectcluster ~]# mmces service stop OBJ
Keystone DB: service successfully stopped.
Keystone: service successfully stopped.
Swift: service successfully stopped.
[root@objectcluster ~]# mmces service start OBJ
Swift: service successfully started.
Keystone DB: service successfully started.
Keystone: service successfully started.
[root@objectcluster ~]#
```

Make sure the proxy-server is running.

```
[root@objectcluster ~]# mmces service list -v
Enabled services: OBJ
OBJ is running
  OBJ:openstack-swift-object-updater          is running
  OBJ:openstack-swift-object-expirer          is running
  OBJ:ibmobjectizer                           is not running
  OBJ:openstack-swift-object                  is running
  OBJ:openstack-swift-account                 is running
  OBJ:openstack-swift-container               is running
  OBJ:memcached                              is running
  OBJ:openstack-swift-proxy                   is running
  OBJ:openstack-swift-object-replicator       is running
  OBJ:openstack-swift-account-reaper          is running
  OBJ:openstack-swift-account-auditor         is running
  OBJ:openstack-swift-container-auditor       is running
  OBJ:openstack-swift-container-updater       is running
  OBJ:openstack-swift-account-replicator      is running
  OBJ:openstack-swift-container-replicator    is running
  OBJ:openstack-swift-object-sof              is not running
  OBJ:postgresql-obj                         is running
  OBJ:httpd (keystone)                       is running
[root@objectcluster ~]#
```

Now, we if try our S3CMD commands that previously failed, we get:

List Buckets:

```
root@s3client:~# s3cmd ls
```

Create Bucket:

```
root@s3client:~# s3cmd mb s3://testbucket
Bucket 's3://testbucket/' created
```

Put Object:

```
root@s3client:~# s3cmd put testfile s3://testbucket
upload: 'testfile' -> 's3://testbucket/testfile' [1 of 1]
 1440 of 1440   100% in    0s    3.13 KB/s  done
```

List Object:

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
root@s3client:~# s3cmd ls s3://testbucket
2022-03-21 14:13      1440  s3://testbucket/testfile
```

Get Object:

```
root@s3client:~# s3cmd get s3://testbucket/testfile testfile2
download: 's3://testbucket/testfile' -> 'testfile2' [1 of 1]
1440 of 1440 100% in 0s 8.22 KB/s done
```

Verify GET Object worked:

```
root@s3client:~# ls -al testfile*
-rw-r--r-- 1 root root 1440 Mar 21 14:56 testfile
-rw-r--r-- 1 root root 1440 Mar 21 14:13 testfile2
```

Delete Object:

```
[root@scalecluster ~]# s3cmd del s3://testbucket/testfile
delete: 's3://testbucket/testfile'
```

Delete Bucket:

```
[root@scalecluster ~]# s3cmd rb s3://testbucket
Bucket 's3://testbucket/' removed
```

S3CMD has some additional functionality that might come in handy.

Copy a file across Buckets:

```
[root@scalecluster ~]# s3cmd cp s3://testbucket1/testfile s3://testbucket2/
remote copy: 's3://testbucket1/testfile' -> 's3://testbucket2/testfile' [1 of 1]
[root@scalecluster ~]# s3cmd ls s3://testbucket2
2022-03-21 18:06      158  s3://testbucket2/testfile
```

Sync a local directory to a Bucket:

```
[root@scalecluster ~]# s3cmd sync syncdir/ s3://testbucket2
remote copy: 'testfile' -> 'testfile10' [1 of 9]
remote copy: 'testfile' -> 'testfile2' [2 of 9]
remote copy: 'testfile' -> 'testfile3' [3 of 9]
remote copy: 'testfile' -> 'testfile4' [4 of 9]
remote copy: 'testfile' -> 'testfile5' [5 of 9]
remote copy: 'testfile' -> 'testfile6' [6 of 9]
remote copy: 'testfile' -> 'testfile7' [7 of 9]
remote copy: 'testfile' -> 'testfile8' [8 of 9]
remote copy: 'testfile' -> 'testfile9' [9 of 9]
Done. Uploaded 0 bytes in 1.6 seconds, 0.00 B/s.
[root@scalecluster ~]# s3cmd ls s3://testbucket2
2022-03-21 18:06      158  s3://testbucket2/testfile
2022-03-21 18:09      158  s3://testbucket2/testfile10
2022-03-21 18:09      158  s3://testbucket2/testfile2
2022-03-21 18:09      158  s3://testbucket2/testfile3
2022-03-21 18:09      158  s3://testbucket2/testfile4
2022-03-21 18:09      158  s3://testbucket2/testfile5
2022-03-21 18:09      158  s3://testbucket2/testfile6
2022-03-21 18:09      158  s3://testbucket2/testfile7
2022-03-21 18:09      158  s3://testbucket2/testfile8
2022-03-21 18:09      158  s3://testbucket2/testfile9
```

Remove a Bucket and all its Objects:

```
[root@scalecluster ~]# s3cmd rb --recursive s3://testbucket2
WARNING: Bucket is not empty. Removing all the objects from it first. This may take some time...
delete: 's3://testbucket2/testfile'
delete: 's3://testbucket2/testfile10'
delete: 's3://testbucket2/testfile2'
delete: 's3://testbucket2/testfile3'
delete: 's3://testbucket2/testfile4'
delete: 's3://testbucket2/testfile5'
delete: 's3://testbucket2/testfile6'
delete: 's3://testbucket2/testfile7'
delete: 's3://testbucket2/testfile8'
delete: 's3://testbucket2/testfile9'
Bucket 's3://testbucket2/' removed
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

We can create a script to PUT an object into a bucket using Virtual-hosted style addressing. This will prove that our Object Store is in fact supporting Virtual-hosted style bucket addressing. Below is a rudimentary script to do just that.

```
[root@scalecluster ~]# cat vh.sh
# Script to test Virtual Hosted Style Bucket Addressing
# PUT a given file into a given bucket

# Variables
S3_ACCESS_KEY=s3access
S3_SECRET_KEY=s3secret
BUCKET="awsbucket"
FILE="testfile"
DATEVALUE=`date -R`
FILEPATH="/${BUCKET}/${FILE}"

# Curl Metadata
CONTENTTYPE="application/x-compressed-tar"
DATEVALUE=`date -R`
SIGNATURE_STRING="PUT\n\n${CONTENTTYPE}\n${DATEVALUE}\n${FILEPATH}"

# Create signature hash to be sent in Authorization header
SIGNATURE_HASH=`echo -en ${SIGNATURE_STRING} | openssl sha1 -hmac ${S3_SECRET_KEY} -binary | base64`

# Clean out awsbucket
s3cmd rb --recursive --force s3://awsbucket >/dev/null 2>&1
s3cmd mb s3://awsbucket >/dev/null 2>&1

# Output to screen
echo
echo "Script to demonstrate Virtual-hosted style bucket addressing"
echo
echo "Listing bucket prior to PUT"
s3cmd ls s3://awsbucket
echo
echo "Performing a PUT to the following URL https://${BUCKET}.s3server.scale.com/${FILE}"
# curl command to do PUT operation to our S3 endpoint
curl -X PUT -T "${FILE}" \
  -H "Host: ${BUCKET}.s3server.scale.com" \
  -H "Date: ${DATEVALUE}" \
  -H "Content-Type: ${CONTENTTYPE}" \
  -H "Authorization: AWS ${S3_ACCESS_KEY}:${SIGNATURE_HASH}" \
  https://${BUCKET}.s3server.scale.com/${FILE}
echo
echo "Listing bucket after PUT"
s3cmd ls s3://awsbucket

[root@scalecluster ~]#
```

Running the script we get:

```
[root@scalecluster ~]# ./vh.sh

Script to demonstrate Virtual-hosted style bucket addressing

Listing bucket prior to PUT

Performing a PUT to the following URL https://awsbucket.s3server.scale.com/testfile

Listing bucket after PUT
2022-04-14 18:54          158  s3://awsbucket/testfile
[root@scalecluster ~]#
```

And now we have Virtual-hosted style addressing working in our cluster! This should ensure compatibility with the widest variety of applications as we support both the S3 and SWIFT Object protocols on IBM Spectrum Scale.

S3CMD is the most versatile of S3 CLI clients. MINIO also offers a S3 CLI client which is detailed below. Note, when we setup our alias we set the PATH style addressing to OFF.

```
[root@scalecluster ~]# wget https://dl.min.io/client/mc/release/linux-amd64/mc
--2022-04-03 20:29:38-- https://dl.min.io/client/mc/release/linux-amd64/mc
Resolving dl.min.io (dl.min.io)... 178.128.69.202, 138.68.11.125
Connecting to dl.min.io (dl.min.io)|178.128.69.202|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 23334912 (22M) [application/octet-stream]
Saving to: 'mc'

mc
100%[=====>] 22.25M 2.25MB/s in 15s
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
2022-04-03 20:29:55 (1.46 MB/s) - 'mc' saved [23334912/23334912]

[root@scalecluster ~]# chmod +x mc
[root@scalecluster ~]# mc alias set SCALE https://s3server.scale.com s3access s3secret --api S3v4 --path off
Added `SCALE` successfully.
[root@scalecluster ~]# mc alias ls
SCALE
  URL      : https://s3server.scale.com
  AccessKey : s3access
  SecretKey : s3secret
  API      : S3v4
  Path     : off
.
```

Some simple commands to create a bucket and sync local files to the bucket:

```
[root@scalecluster ~]# mc mb SCALE/miniobucket
Bucket created successfully `SCALE/miniobucket`.
[root@scalecluster ~]# mc ls SCALE/miniobucket
[root@scalecluster ~]# mc mirror syncdir/ SCALE/miniobucket
/root/syncdir/testfile9: 1.54 KiB / 1.54 KiB | 2.95 KiB/s 0s
[root@scalecluster ~]# mc ls SCALE/miniobucket
[2022-04-03 20:44:12 SAST] 158B STANDARD testfile
[2022-04-03 20:44:12 SAST] 158B STANDARD testfile10
[2022-04-03 20:44:12 SAST] 158B STANDARD testfile2
[2022-04-03 20:44:12 SAST] 158B STANDARD testfile3
[2022-04-03 20:44:12 SAST] 158B STANDARD testfile4
[2022-04-03 20:44:12 SAST] 158B STANDARD testfile5
[2022-04-03 20:44:12 SAST] 158B STANDARD testfile6
[2022-04-03 20:44:12 SAST] 158B STANDARD testfile7
[2022-04-03 20:44:12 SAST] 158B STANDARD testfile8
[2022-04-03 20:44:12 SAST] 158B STANDARD testfile9
[root@scalecluster ~]# mc stat SCALE/miniobucket
Name      : miniobucket/
Size      : 0 B
Type      : folder
Metadata  :
  Versioning: Un-versioned
  Location: af-south-1
  Policy:
[root@scalecluster ~]#
```

You can also write your own scripts to interface with our S3 compliant Object store using the python boto3 package. As a simple example of how to do this we will create a script to list buckets and a script to list objects in a specific bucket.

BOTO3 script to List Buckets:

```
[root@scalecluster botostuff]# cat listbuckets.py
import boto3

# Use the higher level resource api
# https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/s3.html#bucket
s3 = boto3.resource('s3',
    aws_access_key_id='s3access',
    aws_secret_access_key='s3secret',
    endpoint_url='http://s3server.scale.com',
)
print (' Test script to demonstrate how to use BOTO3 to interface with an S3 Object Store ')
print

# List all objects in a container
buckets = s3.buckets.all()
for bucket in buckets:
    print(' ->', bucket)

[root@scalecluster botostuff]#
```

```
[root@scalecluster botostuff]# python3 listbuckets.py
Test script to demonstrate how to use BOTO3 to interface with an S3 Object Store
-> s3.Bucket(name='aclcontainer')
-> s3.Bucket(name='awsacl')
-> s3.Bucket(name='awsbucket')
-> s3.Bucket(name='compressbucket')
-> s3.Bucket(name='miniobucket')
-> s3.Bucket(name='mybucket')
-> s3.Bucket(name='share-me')
-> s3.Bucket(name='sharing')
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
-> s3.Bucket(name='test1')
-> s3.Bucket(name='test10')
-> s3.Bucket(name='test11')
-> s3.Bucket(name='test12')
-> s3.Bucket(name='test13')
-> s3.Bucket(name='test14')
-> s3.Bucket(name='test15')
-> s3.Bucket(name='test2')
-> s3.Bucket(name='test3')
-> s3.Bucket(name='test4')
-> s3.Bucket(name='test5')
-> s3.Bucket(name='test6')
-> s3.Bucket(name='test7')
-> s3.Bucket(name='test8')
-> s3.Bucket(name='test9')
-> s3.Bucket(name='testacl')
-> s3.Bucket(name='testnew1')
-> s3.Bucket(name='testnew10')
-> s3.Bucket(name='testnew11')
-> s3.Bucket(name='testnew12')
-> s3.Bucket(name='testnew13')
-> s3.Bucket(name='testnew14')
-> s3.Bucket(name='testnew15')
-> s3.Bucket(name='testnew2')
-> s3.Bucket(name='testnew3')
-> s3.Bucket(name='testnew4')
-> s3.Bucket(name='testnew5')
-> s3.Bucket(name='testnew6')
-> s3.Bucket(name='testnew7')
-> s3.Bucket(name='testnew8')
-> s3.Bucket(name='testnew9')
-> s3.Bucket(name='webbucket')
[root@scalecluster botostuff]#
```

BOTO3 script to List Objects:

```
[root@scalecluster botostuff]# cat listobjects.py
import boto3

print (' Test script to demonstrate how to use BOTO3 to interface with an S3 Object Store ')
print

# Use the higher level resource api
# https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/s3.html#bucket
s3 = boto3.resource('s3',
    aws_access_key_id='s3access',
    aws_secret_access_key='s3secret',
    endpoint_url='http://s3server.scale.com',
)

# List all objects in a container
bucket = s3.Bucket('mybucket')
for obj in bucket.objects.all():
    print(' ->', obj)

[root@scalecluster botostuff]#
```

```
[root@scalecluster botostuff]# python3 listobjects.py
Test script to demonstrate how to use BOTO3 to interface with an S3 Object Store
-> s3.ObjectSummary(bucket_name='mybucket', key='file13')
-> s3.ObjectSummary(bucket_name='mybucket', key='file14')
-> s3.ObjectSummary(bucket_name='mybucket', key='file15')
-> s3.ObjectSummary(bucket_name='mybucket', key='file16')
-> s3.ObjectSummary(bucket_name='mybucket', key='file17')
[root@scalecluster botostuff]#
```

And that concludes the section on setting up an S3 Compliant Object in IBM Spectrum Scale. In the next section we will look at a practical use case for your S3 Compliant Object store.

Practical Use Case for an S3 Compliant Object Store

Using S3CMD or the MinIO client or other similar applications (e.g. RClone is also quite popular and can be found here <https://rclone.org>) could allow you to mirror or sync fileserver information to your Object store as a form of backup. Using Object versioning (which we turned off) can also allow you to recover previous versions of files (which are stored as objects) similar to how backup applications work. You could also provide persistent storage to Kubernetes clusters (though for Spectrum Scale,

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

making use of the CSI driver is the best option for this requirement). As an example of a practical use case, we will use s3fs to mount a bucket as a local Linux filesystem similar to how a NAS appliance would present an NFS store or SMB share to clients. The s3fs GitHub project can be found here <https://github.com/s3fs-fuse/s3fs-fuse>.

For RHEL8.5, we need to install Fedora's Extra Packages for Enterprise Linux (EPEL) repository. You can do this as follows.

```
[root@scalecluster ~]# yum install https://dl.fedoraproject.org/pub/epel/epel-release-latest-8.noarch.rpm
Updating Subscription Management repositories.
Last metadata expiration check: 0:05:05 ago on Mon 04 Apr 2022 20:55:53 SAST.
epel-release-latest-8.noarch.rpm
15 kB/s | 23 kB    00:01
Dependencies resolved.
=====
Package                               Architecture      Version
Repository                            Size
=====
Installing:
epel-release                          noarch            8-15.el8
@commandline                          23 k
Transaction Summary
=====
Install 1 Package

Total size: 23 k
Installed size: 32 k
Is this ok [y/N]: y
Downloading Packages:
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing      :
1/1
Installing      : epel-release-8-15.el8.noarch
1/1
Running scriptlet: epel-release-8-15.el8.noarch
1/1
Verifying       : epel-release-8-15.el8.noarch
1/1
Installed products updated.

Installed:
  epel-release-8-15.el8.noarch

Complete!
[root@scalecluster ~]#
```

Now we can install s3fs.

```
[root@scalecluster ~]# yum install s3fs-fuse
Updating Subscription Management repositories.
Extra Packages for Enterprise Linux 8 - x86_64
4.1 MB/s | 11 MB    00:02
Extra Packages for Enterprise Linux Modular 8 - x86_64
16 kB/s | 1.0 MB    01:04
Last metadata expiration check: 0:00:02 ago on Mon 04 Apr 2022 21:03:43 SAST.
Dependencies resolved.
=====
Package                               Architecture      Version
Repository                            Size
=====
Installing:
s3fs-fuse                             x86_64            1.91-1.el8
epel                                  272 k
Transaction Summary
=====
Install 1 Package

Total download size: 272 k
Installed size: 648 k
Is this ok [y/N]: y
Downloading Packages:
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
s3fs-fuse-1.91-1.el8.x86_64.rpm
1.9 MB/s | 272 kB      00:00
-----
Total
149 kB/s | 272 kB      00:01
Extra Packages for Enterprise Linux 8 - x86_64
1.6 MB/s | 1.6 kB      00:00
Importing GPG key 0x2F86D6A1:
  Userid      : "Fedora EPEL (8) <epel@fedoraproject.org>"
  Fingerprint: 94E2 79EB 8D8F 25B2 1810 ADF1 21EA 45AB 2F86 D6A1
  From        : /etc/pki/rpm-gpg/RPM-GPG-KEY-EPEL-8
Is this ok [y/N]: y
Key imported successfully
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing      :
1/1
  Installing     : s3fs-fuse-1.91-1.el8.x86_64
1/1
  Running scriptlet: s3fs-fuse-1.91-1.el8.x86_64
1/1
  Verifying      : s3fs-fuse-1.91-1.el8.x86_64
1/1
Installed products updated.

Installed:
  s3fs-fuse-1.91-1.el8.x86_64

Complete!
[root@scalecluster ~]#
```

We need to create a file to store our S3 access secret keys.

```
[root@scalecluster ~]# echo "s3access:s3secret" > /etc/passwd-s3fs
[root@scalecluster ~]# chmod 600 /etc/passwd-s3fs
```

We can mount an existing bucket or create a new one. We will create a new one and also a mount point for the new bucket.

```
[root@scalecluster ~]# s3cmd mb s3://s3fs-fuse
Bucket 's3://s3fs-fuse/' created
[root@scalecluster ~]# mkdir /s3fs-fuse
```

Now we need to mount the S3 Bucket to our Linux filesystem. We will use the following format:

```
sudo s3fs <BUCKETNAME> <MOUNTPPOINT> -o passwd_file=/etc/passwd-s3fs -o allow_other -o url=https://<OBJECT_ENDPOINT>
```

Where:

- <BUCKETNAME> is the name of the bucket that we want to mount.
- <MOUNTPPOINT> is the empty directory we created where we want to mount the bucket (it must already exist).
- /etc/passwd-s3fs is the location of the global credential file that we created earlier.
- -o allow_other allows non-root users to access the mount. Otherwise, only the root user will have access to the mounted bucket.
- -o url specifies the network endpoint for the Object Storage. In our case, this will point to our HAProxy HTTPS Object endpoint address

Note, s3fs uses virtual hosted style bucket addressing by default. You need to explicitly specify the use of path style addressing using the following parameter (we don't need to as our Object store now supports virtual hosted style bucket addressing).

```
-o use_path_request_style (use legacy API calling style)
Enable compatibility with S3-like APIs which do not support the virtual-host request style, by using the older path request style.
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

Using the above mentioned format, we can now mount our bucket as a filesystem on on Linux client.

```
[root@scalecluster ~]# s3fs s3fs-fuse /s3fs-fuse/ -o passwd_file=/etc/passwd-s3fs -o allow_other -o url=https://s3server.scale.com

[root@scalecluster ~]# df -hT
Filesystem      Type      Size  Used Avail Use% Mounted on
devtmpfs        devtmpfs  1.8G   0    1.8G   0% /dev
tmpfs           tmpfs     1.9G   0    1.9G   0% /dev/shm
tmpfs           tmpfs     1.9G   9.6M  1.9G   1% /run
tmpfs           tmpfs     1.9G   0    1.9G   0% /sys/fs/cgroup
/dev/nvme0n1p3  xfs       18G   12G   6.3G   65% /
/dev/nvme0n1p1  xfs      295M  255M   40M   87% /boot
tmpfs           tmpfs     371M  12K   371M   1% /run/user/42
tmpfs           tmpfs     371M  4.0K  371M   1% /run/user/1000
s3fs            fuse.s3fs 16E    0    16E    0% /s3fs-fuse
[root@scalecluster ~]# mount
.
.
s3fs on /s3fs-fuse type fuse.s3fs (rw,nosuid,nodev,relatime,user id=0,group id=0,allow other)
.
```

You can now read and write files to the mounted Bucket.

```
[root@scalecluster ~]# cp /etc/hosts /s3fs-fuse
[root@scalecluster ~]# ls -al /s3fs-fuse/
total 1
drwxrwxrwx  1 root root  0 Jan  1  1970 .
dr-xr-xr-x  18 root root 261 Apr  4 21:13 ..
-rw-r--r--  1 root root 158 Apr  4 21:34 hosts
[root@scalecluster ~]# s3cmd ls s3://s3fs-fuse
2022-04-04 19:34          158  s3://s3fs-fuse/hosts
[root@scalecluster ~]# s3cmd get s3://s3fs-fuse/hosts myhosts
download: 's3://s3fs-fuse/hosts' -> 'myhosts' [1 of 1]
158 of 158 100% in 0s 2.46 KB/s done
[root@scalecluster ~]# cat myhosts
127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4
::1      localhost localhost.localdomain localhost6 localhost6.localdomain6
[root@scalecluster ~]# echo "NEW STUFF" >> /s3fs-fuse/hosts
[root@scalecluster ~]# cat /s3fs-fuse/hosts
127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4
::1      localhost localhost.localdomain localhost6 localhost6.localdomain6
NEW STUFF
[root@scalecluster ~]#
```

Lastly, if you want to mount the s3fs filesystem at boot, you need to add an entry to the /etc/fstab file as follows.

```
s3fs-fuse /s3fs-fuse fuse.s3fs _netdev,allow_other,passwd_file=/etc/passwd-s3fs,url=https://s3server.scale.com/ 0 0
```

Note, objects are NOT encrypted by default. They are stored in plain text in our Object store. In addition to the limitations documented on the s3fs GitHub project page, its important to note each object has a maximum size of 5GB which is the default Large Object Support setting for OpenStack Swift. You can however, use a number of smaller objects to construct a large object. This is taken care of by the SLO and DLO middleware that is part of your proxy-server.conf pipeline. The default IBM Spectrum Scale OpenStack Swift proxy-server.conf file includes both SLO and DLO in the pipeline with all the defaults enabled. You can use the mmobj command to adjust this if required. When you upload a large object (based on the SLO and DLO middleware settings) Swift will divide the large file into a set of segments, and then upload these segments in parallel. Once all the parts are uploaded, the swift client tool will create a manifest file for these parts to ensure that these parts can be downloaded as a single object. You can easily check this as a new bucket is created with all the segments usually called <BUCKETNAME>+segments.

A simple test of how this works.

```
[root@scalecluster s3fs-fuse]# dd if=/dev/zero of=2GB_file count=2000 bs=1M
2000+0 records in
2000+0 records out
2097152000 bytes (2.1 GB, 2.0 GiB) copied, 54.6395 s, 38.4 MB/s
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
[root@scalecluster s3fs-fuse]# s3cmd ls s3://s3fs-fuse
2022-04-04 20:01 2097152000 s3://s3fs-fuse/2GB_file
2022-04-04 19:38 169 s3://s3fs-fuse/hosts
[root@scalecluster s3fs-fuse]#

[root@objectcluster ~]# source openrc_s3
[root@objectcluster ~]# swift list | grep s3fs
s3fs-fuse
s3fs-fuse+segments
[root@objectcluster ~]# swift list s3fs-fuse+segments
2GB_file/NjQxODkyYTYtOGVjNC00Zjc1LWE4YTUtY2ViY2U2MWNjOTZj/1
2GB_file/NjQxODkyYTYtOGVjNC00Zjc1LWE4YTUtY2ViY2U2MWNjOTZj/10
2GB_file/NjQxODkyYTYtOGVjNC00Zjc1LWE4YTUtY2ViY2U2MWNjOTZj/100
2GB_file/NjQxODkyYTYtOGVjNC00Zjc1LWE4YTUtY2ViY2U2MWNjOTZj/101
2GB_file/NjQxODkyYTYtOGVjNC00Zjc1LWE4YTUtY2ViY2U2MWNjOTZj/102
2GB_file/NjQxODkyYTYtOGVjNC00Zjc1LWE4YTUtY2ViY2U2MWNjOTZj/103
2GB_file/NjQxODkyYTYtOGVjNC00Zjc1LWE4YTUtY2ViY2U2MWNjOTZj/104
2GB_file/NjQxODkyYTYtOGVjNC00Zjc1LWE4YTUtY2ViY2U2MWNjOTZj/105
2GB_file/NjQxODkyYTYtOGVjNC00Zjc1LWE4YTUtY2ViY2U2MWNjOTZj/106
.
.
.
```

This is all done transparent to the user. Typically, you don't need to change the default settings. We have now demonstrated a practical application for our S3 Compliant Object store using s3fs. More detail on the s3fs options can be obtained by issuing “*man s3fs*”.

Another potential use case for our S3 Compliant Object store would be to host a static website. AWS S3 supports configuring your bucket to function as a website. On a static website, individual webpages include static content and can contain client-side scripts. To check if this feature is implemented in the Swift S3API middleware let us try and create a static website via an S3 client.

```
[root@scalecluster ~]# aws s3 website s3://awsbuckets3server.scale.com/ --index html/index.html --error-document
html/404.html --endpoint-url=https://s3server.scale.com

An error occurred (NotImplemented) when calling the PutBucketWebsite operation: The requested resource is not
implemented
[root@scalecluster ~]#

[root@scalecluster ~]# s3cmd ws-create --ws-index=index.html --ws-error=404.html s3://awswebbucket
ERROR: S3 error: 501 (NotImplemented): The requested resource is not implemented
[root@scalecluster ~]#
```

So neither the AWS CLI or s3cmd can create the bucket. The error code (*NotImplemented*) implies that our S3API middleware has no support for this function. If we issue the s3cmd with the `-debug` flag we can confirm this.

```
.
.
.
DEBUG: non-proxied HTTPSConnection(webbucket.s3server.scale.com, None)
DEBUG: format_uri(): /?website
DEBUG: Sending request method_string='PUT', uri='/?website', headers={'x-amz-date': '20220507T092845Z',
'Authorization': 'AWS4-HMAC-SHA256 Credential=s3access/20220507/af-south-1/s3/aws4_request,SignedHeaders=host;x-
amz-content-sha256;x-amz-date,Signature=bdf8e1ee750cfc88ff94c07ea9048cac52e4b1ff0a69c77d2bf8eff9636b1018', 'x-amz-
content-sha256': 'd9b635030955bea091cdd4363723eb9be76ea320f9752f1f033f76dd7b56014b'}, body=(217 bytes)
DEBUG: ConnMan.put(): connection put back to pool (https://webbucket.s3server.scale.com#1)
DEBUG: Response:
{'data': b'<?xml version='1.0' encoding='UTF-8'?>\n<Error><Code>NotImplement"
b'ed</Code><Message>The requested resource is not implemented</Message>
b'><RequestId>tx707ef25c5639420eb4d9d-0062763bcd</RequestId></Error>',
'headers': {'content-type': 'application/xml',
'date': 'Sat, 07 May 2022 09:28:45 GMT',
'transfer-encoding': 'chunked',
'x-amz-id-2': 'tx707ef25c5639420eb4d9d-0062763bcd',
'x-amz-request-id': 'tx707ef25c5639420eb4d9d-0062763bcd',
'x-openstack-request-id': 'tx707ef25c5639420eb4d9d-0062763bcd',
'x-trans-id': 'tx707ef25c5639420eb4d9d-0062763bcd'},
'reason': 'Not Implemented',
'status': 501}
DEBUG: S3Error: 501 (Not Implemented)
DEBUG: HttpHeader: content-type: application/xml
DEBUG: HttpHeader: x-amz-id-2: tx707ef25c5639420eb4d9d-0062763bcd
.
.
.
```

A quick check of the Swift S3 Compatibility matrix and sure enough this feature is not supported.

| | | |
|---|----------------|----|
| Public website 7 8 9 10 | Public Website | No |
|---|----------------|----|

Figure 8: AWS S3 static website support is not available in OpenStack Swift

OpenStack Swift however does have this functionality built in. Whilst the ability to support this function is not available via the S3API Swift middleware we will take a quick look at how to achieve this in Swift. The ability to host a static website from a container is provided by the Static Web middleware and servers data with a specified index file, error file and optional file listings. The requirement is for the Static Web middleware to be added to the Swift pipeline in our `/etc/swift/proxy-server.conf` file and for us to have a Static Web middleware configuration section. Let us check our `proxy-server.conf` file for the required middleware and configuration section.

```
[root@objectcluster ~]# grep static /etc/swift/proxy-server.conf
pipeline = catch_errors healthcheck proxy-logging cache container_sync formpost tempurl authtoken s3api s3token
keystoneauth container-quotas account-quotas staticweb bulk slo dlo proxy-logging proxy-server
[filter:staticweb]
use = egg:swift#staticweb
[root@objectcluster ~]#
```

Luckily for us, the default Swift `proxy-server.conf` file in Spectrum Scale comes with this middleware already enabled (we could add this manually using the `mmobj` command if it was missing). Let us take a quick look at how to create a static webpage in Swift.

First we create a container to host our website and configure the read ACL to allow read access and optionally to allow file listing.

```
[root@objectcluster ~]# swift post website
[root@objectcluster ~]# swift post -r '.*,.rlistings' website
```

Next we upload our files we wish to host. In our case, just the `index.html` and `404.html` files.

```
[root@objectcluster ~]# swift upload website index.html 404.html
404.html
index.html
[root@objectcluster ~]#
```

We want to allow listing of all files in the container by. Enabling web-listings.

```
[root@objectcluster ~]# swift post -m 'web-listings: true' website
```

Let us just validate the configuration and ACLs.

```
[root@objectcluster ~]# swift stat website
Account: AUTH_224f53da12e04b7893783821a34c171f
Container: website
Objects: 2
Bytes: 250
Read ACL: .*,.rlistings
Write ACL:
Sync To:
Sync Key:
Meta Web-Listings: true
Content-Type: application/json; charset=utf-8
X-Timestamp: 1651917134.64304
Last-Modified: Sat, 07 May 2022 09:58:43 GMT
Accept-Ranges: bytes
X-Storage-Policy: policy-0
X-Container-Sharding: False
X-Trans-Id: txbla4f0b131084eb79e4ea-0062764330
X-Openstack-Request-Id: txbla4f0b131084eb79e4ea-0062764330
[root@objectcluster ~]#
```

We should be able to access a file listing using the following URL:

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

https://s3server.scale.com/v1/AUTH_224f53da12e04b7893783821a34c171f/website

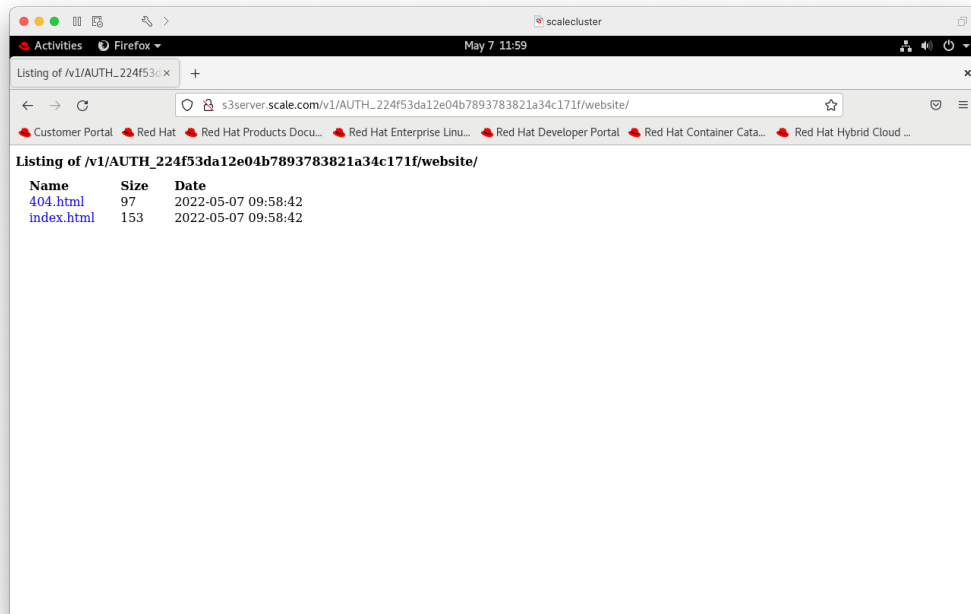


Figure 9: Static website file listing in Swift using a container to host our website

If we want our container to work as a full website, we need to enable the index and error settings.

```
[root@objectcluster ~]# swift post -m 'web-index:index.html' website
[root@objectcluster ~]# swift post -m 'web-error:404.html' website
```

Accessing our container again we should be directed to our index.html automatically.

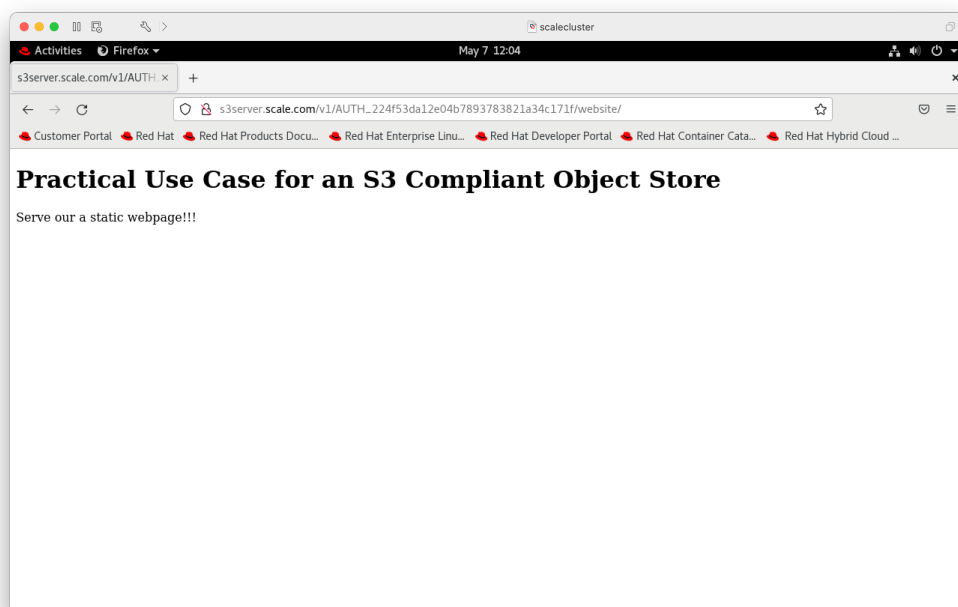


Figure 10: Static website functionality in Swift using a container to host our website

As mentioned above, the ability to host a static website via your S3 bucket is not currently supported in the S3API Swift middleware though we have managed to demonstrate this functionality using Swift native commands.

Sharing Buckets with different Users

A common scenario that you might come across is the requirement to share containers and objects with different users. This section will go through the steps to share an object using OpenStack Swift and S3 protocols.

For OpenStack Swift, we will create a new bucket call *aclbucket* which is owned by the account *s3project* and user *s3user*. We will then create a new user called *acluser* in the *s3project* account which is part of the default Swift domain and a new role called *_member_* and assign this role to the *acluser*. We will then grant access to an object in the new *aclbucket* to the user *acluser* and try and access it. Note, you can create users if you have the admin or swift operator roles assigned to you.

Create a new container and upload an object to it. We will also query the container properties. Note, the account owner is *s3project* and we don't have any ACLs set.

```
[root@objectcluster ~]# source openrc_s3
[root@objectcluster ~]# swift post aclcontainer
[root@objectcluster ~]# swift upload aclcontainer /etc/hosts
etc/hosts
[root@objectcluster ~]# swift stat aclcontainer
  Account: AUTH_224f53da12e04b7893783821a34c171f
  Container: aclcontainer
  Objects: 1
    Bytes: 344
  Read ACL:
  Write ACL:
  Sync To:
  Sync Key:
  Content-Type: application/json; charset=utf-8
  X-Timestamp: 1648393779.07658
  Last-Modified: Sun, 27 Mar 2022 16:20:11 GMT
  Accept-Ranges: bytes
  X-Storage-Policy: policy-0
  X-Container-Sharding: False
  X-Trans-Id: txfl1287316585d46e28a64d-0062408ecd
X-OpenStack-Request-Id: txfl1287316585d46e28a64d-0062408ecd
[root@objectcluster ~]# swift stat aclcontainer etc/hosts
  Account: AUTH_224f53da12e04b7893783821a34c171f
  Container: aclcontainer
  Object: etc/hosts
  Content Type: application/octet-stream
  Content Length: 344
  Last Modified: Sun, 27 Mar 2022 16:20:12 GMT
    ETag: e16d4fa2b8f51ba5ebde8901fe260ce5
  Meta Mtime: 1647760482.106204
  X-Timestamp: 1648398011.38034
  Accept-Ranges: bytes
  X-Trans-Id: tx4a683d3ce2a543ef99244-0062408ed8
X-OpenStack-Request-Id: tx4a683d3ce2a543ef99244-0062408ed8
```

Now create the *acluser*, the new role called *_member_* and assign it to *acluser*.

```
[root@objectcluster ~]# openstack user create --project s3project --password Passw0rd acluser
+-----+-----+
| Field | Value |
+-----+-----+
| default_project_id | 224f53da12e04b7893783821a34c171f |
| domain_id | default |
| enabled | True |
| id | 3c00f89ce91b44d0b595cc1468cbb814 |
| name | acluser |
| options | {} |
| password_expires_at | None |
+-----+-----+
[root@objectcluster ~]# openstack role create _member_
+-----+-----+
| Field | Value |
+-----+-----+
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
+-----+
| description | None
| domain_id   | None
| id          | f399ca552ed14328b4223b99b2638933
| name        | _member_
| options     | {}
+-----+
[root@objectcluster ~]# openstack role add --project s3project --user acluser _member_
[root@objectcluster ~]# openstack role assignment list --names
+-----+-----+-----+-----+-----+-----+-----+
| Role      | User              | Group | Project          | Domain | System | Inherited |
+-----+-----+-----+-----+-----+-----+-----+
| admin     | ksAdmin@Default   |        | admin@Default    |         |         | False     |
| admin     | ksSwift@Default   |        | service@Default  |         |         | False     |
| admin     | s3user@Default    |        | s3project@Default|         |         | False     |
| ResellerAdmin | s3user@Default    |        | s3project@Default|         |         | False     |
| _member_  | acluser@Default   |        | s3project@Default|         |         | False     |
| admin     | ksAdmin@Default   |        |                   |         | all    | False     |
| reader    | ksAdmin@Default   |        |                   |         | all    | False     |
+-----+-----+-----+-----+-----+-----+-----+
```

We need to now add an ACL to allow *acluser* access to the container and object. Note, we do this as the *s3user* so you need to source the *openrc_s3* profile.

```
[root@objectcluster ~]# swift post -r 's3project:acluser' -w 's3project:acluser' aclcontainer
[root@objectcluster ~]# swift stat aclcontainer
Account: AUTH_224f53da12e04b7893783821a34c171f
Container: aclcontainer
Objects: 1
Bytes: 344
Read ACL: s3project:acluser
Write ACL: s3project:acluser
Sync To:
Sync Key:
Content-Type: application/json; charset=utf-8
X-Timestamp: 1648393779.07658
Last-Modified: Sun, 27 Mar 2022 16:24:49 GMT
Accept-Ranges: bytes
X-Storage-Policy: policy-0
X-Container-Sharding: False
X-Trans-Id: txbbb8cf06b9704a6d8fbf7-0062408fda
X-OpenStack-Request-Id: txbbb8cf06b9704a6d8fbf7-0062408fda
[root@objectcluster ~]#
```

Now we can copy our *openrc_s3* file and modify it with the *acluser* username and password.

```
[root@objectcluster ~]# cat openrc_acluser
# Sun Mar 20 03:17:09 SAST 2022
export OS_AUTH_URL="http://127.0.0.1:35357/v3"
export OS_IDENTITY_API_VERSION=3
export OS_AUTH_VERSION=3
export OS_USERNAME='acluser'
export OS_PASSWORD='Passw0rd'
export OS_USER_DOMAIN_NAME='Default'
export OS_PROJECT_NAME=s3project
export OS_PROJECT_DOMAIN_NAME=Default
export OS_SYSTEM_SCOPE=all
export OS_TENANT_NAME=s3user

#export OS_REGION_NAME="RegionOne"
[root@objectcluster ~]#
```

After sourcing our *acluser* openrc profile, we can try and access the container and its objects.

```
[root@objectcluster ~]# source openrc_acluser
[root@objectcluster ~]# swift list aclcontainer
etc/hosts
[root@objectcluster ~]# swift stat aclcontainer
Account: AUTH_224f53da12e04b7893783821a34c171f
Container: aclcontainer
Objects: 1
Bytes: 344
Read ACL:
Write ACL:
Sync To:
Sync Key:
Content-Type: application/json; charset=utf-8
X-Timestamp: 1648393779.07658
Last-Modified: Sun, 27 Mar 2022 16:24:49 GMT
Accept-Ranges: bytes
X-Storage-Policy: policy-0
X-Trans-Id: txdidc585fb9054a278e07c-006240906c
X-OpenStack-Request-Id: txdidc585fb9054a278e07c-006240906c
```


Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
[root@objectcluster ~]# swift upload aclcontainer t.pol
Warning: failed to create container 'aclcontainer': 403 Forbidden: <html><h1>Forbidden</h1><p>Access was denied to
this resource
t.pol
[root@objectcluster ~]# swift list aclcontainer
etc/hosts
t.pol
[root@objectcluster ~]#
[root@objectcluster ~]# swift download aclcontainer etc/hosts
etc/hosts [auth 0.800s, headers 0.904s, total 0.912s, 0.003 MB/s]
[root@objectcluster ~]#
```

That concludes a simple example of how to share containers and objects on OpenStack Swift. Now we will turn our attention to attempting this via the S3 API. For AWS S3, Amazon recommends the use of bucket policies or IAM policies for access control. The use of bucket ACLs is a legacy access control mechanism that is discouraged (<https://aws.amazon.com/blogs/security/iam-policies-and-bucket-policies-and-acls-oh-my-controlling-access-to-s3-resources/>). The use of S3 ACLs is similar to how we shared the container and object with OpenStack Swift). Note, Swift ACLs and S3 ACLs are not compatible and should not be mixed on a specific bucket or container.

A simple bucket policy to allow read access to anyone would look like this.

```
{
  "Version": "2012-10-17",
  "Id": "policy-read-any",
  "Statement": [
    {
      "Sid": "read-any",
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "*"
        ]
      },
      "Action": [
        "s3:ListBucket",
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::*"
      ]
    }
  ]
}
```

Trying to apply this policy though results in the following error:

```
[root@scalecluster ~]# s3cmd setpolicy bucket.pol s3://aclcontainer
ERROR: S3 error: 501 (NotImplemented): The requested resource is not implemented
[root@scalecluster ~]#
```

Sure enough, checking the S3API compatibility matrix we can see that S3 Bucket ACL policy is not supported.

| Bucket policy | Advanced ACLs | No |
|-------------------------------|---------------|----|
|-------------------------------|---------------|----|

Figure 11: S3API middleware Compatibility Matrix support for Bucket Policy

So, we will need to resort to using the legacy Bucket ACL method. Firstly, let's create a new bucket, upload an object to it and query its properties.

```
[root@scalecluster ~]# s3cmd mb s3://share-me
Bucket 's3://share-me/' created
[root@scalecluster ~]# s3cmd info s3://share-me
s3://share-me/ (bucket):
  Location:  af-south-1
  Payer:      none
  Expiration Rule: none
  Policy:     none
  CORS:       none
  ACL:        s3project:s3user: FULL_CONTROL
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
[root@scalecluster ~]#  
[root@scalecluster ~]# s3cmd put /etc/hosts s3://share-me/hosts  
upload: '/etc/hosts' -> 's3://share-me/hosts' [1 of 1]  
158 of 158 100% in 0s 1232.66 B/s done  
[root@scalecluster ~]#
```

Now we can set the bucket ACL to allow our new *acluser* read and write access (full_control).

```
[root@scalecluster ~]# s3cmd setacl s3://share-me --acl-grant=full_control:s3project:acluser  
s3://share-me/: ACL updated  
[root@scalecluster ~]# s3cmd setacl s3://share-me/hosts --acl-grant=full_control:s3project:acluser  
s3://share-me/hosts: ACL updated
```

Query the bucket and object properties to ensure our new ACLs are in effect. Our *acluser* now has full control on the bucket and its associated object.

```
[root@scalecluster ~]# s3cmd info s3://share-me  
s3://share-me/ (bucket):  
Location: af-south-1  
Payer: none  
Expiration Rule: none  
Policy: none  
CORS: none  
ACL: s3project:s3user: FULL_CONTROL  
ACL: s3project:acluser: FULL_CONTROL  
[root@scalecluster ~]# s3cmd info s3://share-me/hosts  
s3://share-me/hosts (object):  
File size: 158  
Last mod: Mon, 28 Mar 2022 12:39:09 GMT  
MIME type: application/xml  
Storage: STANDARD  
MD5 sum: 54fb6627dbaa37721048e4549db3224d  
SSE: none  
Policy: none  
CORS: none  
ACL: s3project:s3user: FULL_CONTROL  
ACL: s3project:acluser: FULL_CONTROL  
x-amz-meta-s3cmd-attribs:  
atime:1648460461/ctime:1646667149/gid:0/gname:root/md5:54fb6627dbaa37721048e4549db3224d/mode:33188/mtime:1536580263  
/uid:0/uname:root  
[root@scalecluster ~]#
```

Before we can attempt access, we need to generate EC2 credentials for our *acluser* in OpenStack Swift. You need to generate credentials with the admin role or swift operator. In our case, *s3user* has the admin role assigned.

```
[root@objectcluster ~]# source openrc_s3  
[root@objectcluster ~]# openstack ec2 credentials create --project s3project --user acluser --user-domain default --project-domain default  
+-----+  
| Field      | Value  
|-----+  
| access     | 0c2bb825a6e6466db8b1846e29446315  
| links      | {'self': 'http://s3server.scale.com:5000/v3/users/3c00f89ce91b44d0b595cc1468cbb814/credentials/OS-EC2/0c2bb825a6e6466db8b1846e29446315'} |  
| project_id | 224f53da12e04b7893783821a34c171f  
| secret     | 6c3e4614814d4aeb748cd4aef0b6fee  
| trust_id   | None  
| user_id    | 3c00f89ce91b44d0b595cc1468cbb814  
+-----+  
+-----+  
+-----+  
+-----+
```

Once we generate an access and secret key for our user *acluser*, let us verify the new credentials as the *acluser*.

```
[root@objectcluster ~]# source openrc_acluser  
[root@objectcluster ~]# openstack ec2 credentials list  
+-----+  
| Access      | Secret      | Project ID      | User ID      |  
|-----+-----+-----+-----+  
|
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
+-----+-----+-----+-----+
| 0c2bb825a6e6466db8b1846e29446315 | 6c3e4614814d4ae1b748cd4aef0b6fee | 224f53da12e04b7893783821a34c171f |
| 3c00f89ce91b44d0b595cc1468cbb814 |                                     |                                     |
+-----+-----+-----+-----+
```

We can now test access to the S3 Bucket using the access key and secret for *acluser*. In our case, we will just copy *.s3cfg* and modify the access key and secret to match the newly generated ones for *acluser*.

Query all buckets as *acluser*.

```
[root@scalecluster ~]# s3cmd ls
2009-02-03 16:45 s3://aclcontainer
2009-02-03 16:45 s3://compressbucket
2009-02-03 16:45 s3://mybucket
2009-02-03 16:45 s3://share-me
2009-02-03 16:45 s3://sharing
2009-02-03 16:45 s3://test1
2009-02-03 16:45 s3://test10
2009-02-03 16:45 s3://test11
2009-02-03 16:45 s3://test12
2009-02-03 16:45 s3://test13
2009-02-03 16:45 s3://test14
2009-02-03 16:45 s3://test15
2009-02-03 16:45 s3://test2
2009-02-03 16:45 s3://test3
2009-02-03 16:45 s3://test4
2009-02-03 16:45 s3://test5
2009-02-03 16:45 s3://test6
2009-02-03 16:45 s3://test7
2009-02-03 16:45 s3://test8
2009-02-03 16:45 s3://test9
2009-02-03 16:45 s3://testacl
2009-02-03 16:45 s3://testnew1
2009-02-03 16:45 s3://testnew10
2009-02-03 16:45 s3://testnew11
2009-02-03 16:45 s3://testnew12
2009-02-03 16:45 s3://testnew13
2009-02-03 16:45 s3://testnew14
2009-02-03 16:45 s3://testnew15
2009-02-03 16:45 s3://testnew2
2009-02-03 16:45 s3://testnew3
2009-02-03 16:45 s3://testnew4
2009-02-03 16:45 s3://testnew5
2009-02-03 16:45 s3://testnew6
2009-02-03 16:45 s3://testnew7
2009-02-03 16:45 s3://testnew8
2009-02-03 16:45 s3://testnew9
[root@scalecluster ~]#
```

Try and access a bucket for which we were not granted access.

```
[root@scalecluster ~]# s3cmd ls s3://testnew9
ERROR: Access to bucket 'testnew9' was denied
ERROR: S3 error: 403 (AccessDenied): Access Denied.
[root@scalecluster ~]#
```

Query the share-me bucket for which we were granted access. If that works, get an object from the share-me bucket and upload a new object to the bucket.

```
[root@scalecluster ~]# s3cmd ls s3://share-me
2022-03-28 12:39      158 s3://share-me/hosts
[root@scalecluster ~]# s3cmd get s3://share-me/hosts
download: 's3://share-me/hosts' -> './hosts' [1 of 1]
158 of 158 100% in 0s 4.14 KB/s done
[root@scalecluster ~]# s3cmd put object2.txt s3://share-me
upload: 'object2.txt' -> 's3://share-me/object2.txt' [1 of 1]
573462 of 573462 100% in 0s 7.02 MB/s done
[root@scalecluster ~]# s3cmd ls s3://share-me
2022-03-28 12:39      158 s3://share-me/hosts
2022-03-28 12:43    573462 s3://share-me/object2.txt
[root@scalecluster ~]#
```

Lastly, our current Swift role (*_member_*) should not allow us to create or delete buckets. We can check this as follows.

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
[root@scalecluster ~]# s3cmd mb s3://mystuff
ERROR: Access to bucket 'mystuff' was denied
ERROR: S3 error: 403 (AccessDenied): Access Denied.
[root@scalecluster ~]#
```

The above methods serve just to showcase how to share information across different users using both the Swift protocol and S3 protocol. One other key feature of AWS S3 is the ability to make an object public so that the object can be accessed from anyone over the internet using the bucket/object URL. This is demonstrated below.

```
root@scalecluster ~]# aws s3api --endpoint-url=https://s3server.scale.com put-object-acl --key hosts --bucket
awsacl --acl public-read
[root@scalecluster ~]# s3cmd info s3://awsacl/hosts
s3://awsacl/hosts (object):
  File size: 158
  Last mod:  Wed, 30 Mar 2022 13:07:14 GMT
  MIME type: application/octet-stream
  Storage:   STANDARD
  MD5 sum:   54fb6627dbaa37721048e4549db3224d
  SSE:       none
  Policy:    none
  CORS:      none
  ACL:       *anon*: READ
  ACL:       s3project:s3user: FULL_CONTROL
  URL:       http://awsacl.s3server.scale.com/hosts
[root@scalecluster ~]#
```

In theory, anyone should be able to access the object using the URL displayed above. In our case it is <http://awsacl.s3server.scale.com/hosts>. Trying this URL though doesn't work currently via the Swift S3API middleware. Attempting to access it on our Object store results in Bad URL.

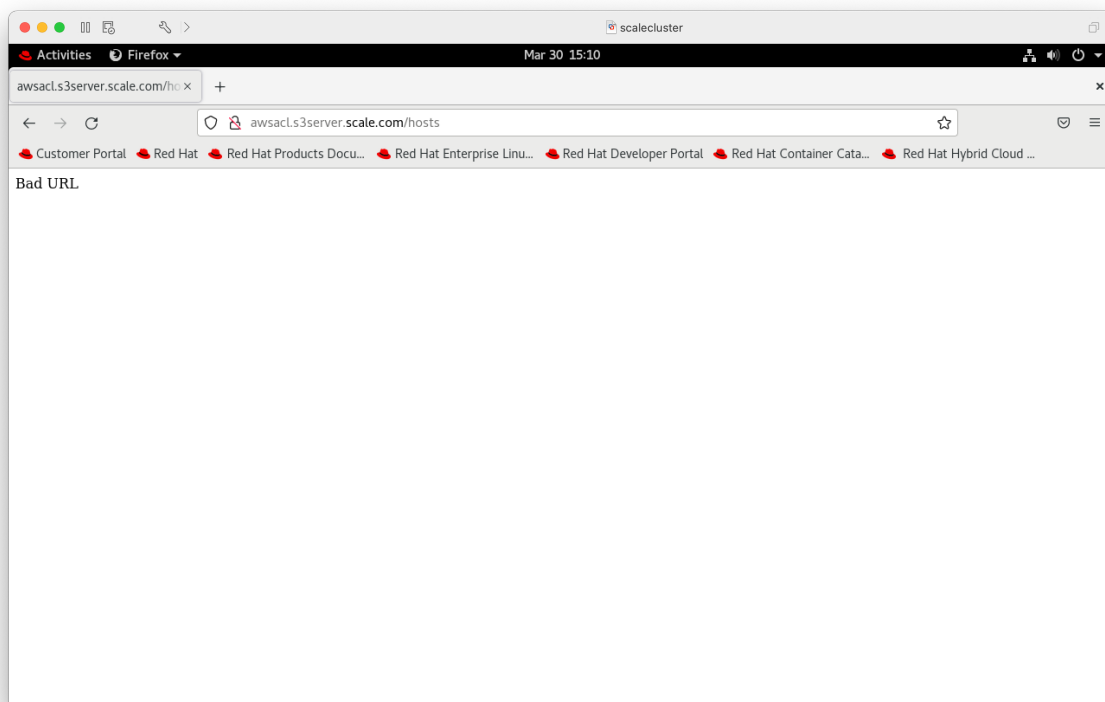


Figure 12: Trying to access a public S3 Object via HTTP

Alternatively, we can share this object on S3 using presigned URLs. By default, all S3 objects are private with only the object owner able to access them. The object owner however, can optionally share objects with others by creating a presigned URL using their own security credentials to grant time-limited permission to download objects. Anyone who receives the presigned URL can then

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

access the object. Below is an example of how this works. We will create a bucket and upload a test file to it and then generate a presigned URL and access the object with a web browser. We will choose to set the time-limited access to 3000 seconds.

```
[root@scalecluster html]# s3cmd mb s3://awsbucket
Bucket 's3://awsbucket/' created
[root@scalecluster ~]# cat testfile
127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4
::1 localhost localhost.localdomain localhost6 localhost6.localdomain6
[root@scalecluster ~]#
[root@scalecluster ~]# s3cmd put testfile s3://awsbucket/testfile
upload: 'testfile' -> 's3://awsbucket/testfile' [1 of 1]
 158 of 158 100% in 0s 550.85 B/s done
[root@scalecluster ~]# s3cmd ls
2009-02-03 16:45 s3://awsbucket
[root@scalecluster ~]# s3cmd ls s3://awsbucket
2022-05-07 06:56 158 s3://awsbucket/testfile
[root@scalecluster ~]# s3cmd signurl s3://awsbucket/testfile +3000
http://awsbucket.s3server.scale.com/testfile?AWSAccessKeyId=s3access&Expires=1651909880&Signature=XmvBnspYy3Bce%2BJEeN2IN2iVYpU%3D
[root@scalecluster ~]#
```

Using the presigned URL, we can now access the object via a standard web browser.

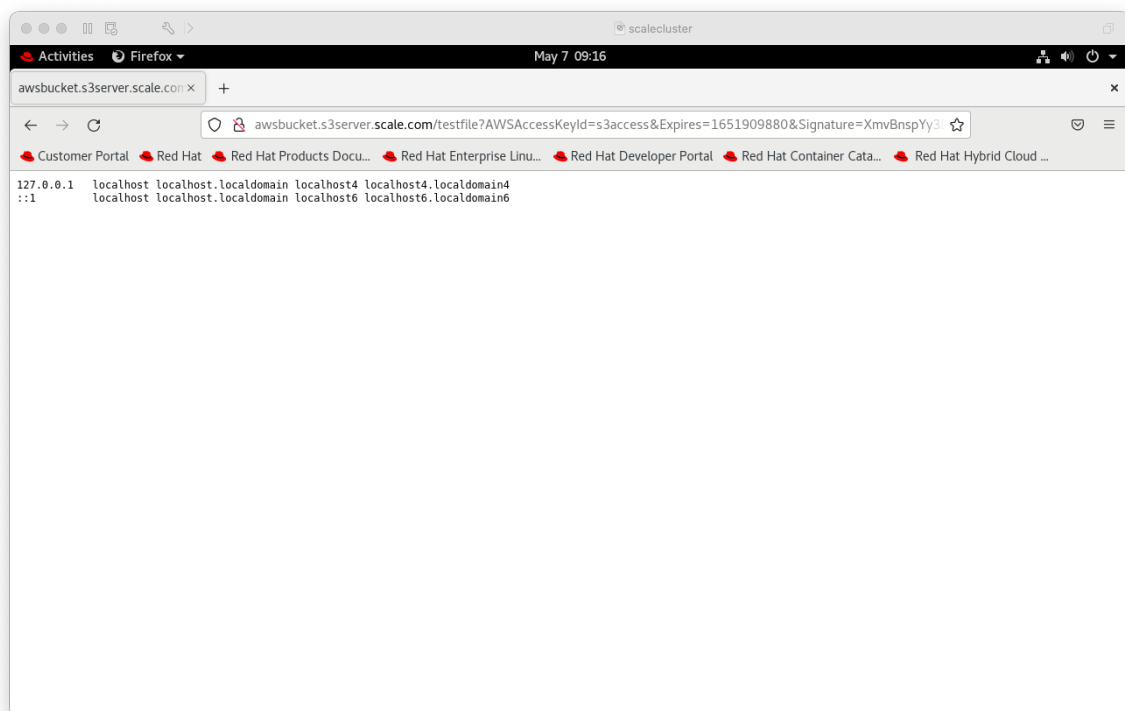


Figure 13: Accessing an object using a S3 presigned URL

Swift has similar functionality if you require it. It is arguably more powerful than the S3 presigned URL as we can also allow upload access via PUT. This functionality on Swift is achieved via the *formpost* and *tempurl* middleware. Luckily, the IBM Spectrum Scale default `proxy-server.conf` file includes both middleware in the pipeline which means we can make use of this function with little effort (or you could add it if it wasn't included by default).

```
[root@objectcluster ~]# mmobj config list --ccrfile proxy-server.conf | grep temp
pipeline = catch_errors healthcheck proxy-logging cache container_sync formpost tempurl authtoken s3api s3token
keystoneauth container-quotas account-quotas staticweb bulk slo dlo proxy-logging proxy-server
[filter:tempurl]
use = egg:swift#tempurl
```

Let us create a new container and upload and object to it.

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
[root@objectcluster ~]# source openrc_s3
[root@objectcluster ~]# swift post webbucket
[root@objectcluster ~]# swift upload webbucket hosts
hosts
[root@objectcluster ~]# swift list webbucket
hosts
```

If we query the container, we need to note the storage URL field. You can also see that there is no secret key set yet.

```
[root@objectcluster ~]# swift stat -v webbucket
URL: http://s3server.scale.com:8080/v1/AUTH_224f53da12e04b7893783821a34c171f/webbucket
Auth Token: gAAAAABiQf7ECjquEftjuV7o0DnxxVR2yG0-
XjHqcgkd3ami5_V9mmTuPSEVTk0pjY4_8hzM3EDI1PMaKYJ4IfAmxStsRB8k4y_WV421QGPR7uwcKo7zj56yJNr6yTC8kYm-7gTiLkXjnp6XKjm33-
Kdo-hrUyLwYn-SWEpTicLdZR86vopPQAY
Account: AUTH_224f53da12e04b7893783821a34c171f
Container: webbucket
Objects: 1
Bytes: 344
Read ACL:
Write ACL:
Sync To:
Sync Key:
Content-Type: application/json; charset=utf-8
X-Timestamp: 1648492125.42192
Last-Modified: Mon, 28 Mar 2022 18:29:45 GMT
Accept-Ranges: bytes
X-Storage-Policy: policy-0
X-Container-Sharding: False
X-Trans-Id: txb22ef95cd9264006a05dc-006241fec4
X-OpenStack-Request-Id: txb22ef95cd9264006a05dc-006241fec4
[root@objectcluster ~]#
```

Let us set a secret key (mysecret123) and query the container again. Notice “Meta Temp-Url-Key” is set to our secret.

```
[root@objectcluster ~]# swift post -m 'Temp-URL-Key:mysecret123' webbucket
[root@objectcluster ~]# swift stat -v webbucket
URL: http://s3server.scale.com:8080/v1/AUTH_224f53da12e04b7893783821a34c171f/webbucket
Auth Token: gAAAAABiQf8V5NHVO8cQ4Ch17WQaG_JZiNn7wEoTFsBQfuP_3bViWg_MLzPhzTF7-
NJ_WoukVsLANvgGDa6a8zOMHRWOh0SoA_ISRC89QzxgMnFn_Cw8M42_FHbiU28FY4a-
125cr99qAqmDjwopd81Fy8qzxc9ebmA64wopYgMAh74twpjerKs
Account: AUTH_224f53da12e04b7893783821a34c171f
Container: webbucket
Objects: 1
Bytes: 344
Read ACL:
Write ACL:
Sync To:
Sync Key:
Meta Temp-Url-Key: mysecret123
Content-Type: application/json; charset=utf-8
X-Timestamp: 1648492125.42192
Last-Modified: Mon, 28 Mar 2022 18:31:44 GMT
Accept-Ranges: bytes
X-Storage-Policy: policy-0
X-Container-Sharding: False
X-Trans-Id: tx150e104bdb914c328e254-006241ff15
X-OpenStack-Request-Id: tx150e104bdb914c328e254-006241ff15
[root@objectcluster ~]#
```

Now we just need to generate the tempurl using our OpenStack Swfit client. The usage for tempurl is as follows:

```
[root@objectcluster ~]# swift tempurl
Usage: swift tempurl [--absolute] [--prefix-based] [--iso8601]
      <method> <time> <path> <key>
Generates a temporary URL for a Swift object.

Positional arguments:
  <method>          An HTTP method to allow for this temporary URL.
                    Usually 'GET' or 'PUT'.
  <time>            The amount of time the temporary URL will be
                    valid. The time can be specified in two ways:
                    an integer representing the time in seconds or an
                    ISO 8601 timestamp in a specific format.
                    If --absolute is passed and time
                    is an integer, the seconds are interpreted as the Unix
                    timestamp when the temporary URL will expire. The ISO
                    8601 timestamp can be specified in one of following
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
formats:

i) Complete date: YYYY-MM-DD (eg 1997-07-16)

ii) Complete date plus hours, minutes and seconds:

    YYYY-MM-DDThh:mm:ss

    (eg 1997-07-16T19:20:30)

iii) Complete date plus hours, minutes and seconds with
    UTC designator:

    YYYY-MM-DDThh:mm:ssZ

    (eg 1997-07-16T19:20:30Z)

Please be aware that if you don't provide the UTC
designator (i.e., Z) the timestamp is generated using
your local timezone. If only a date is specified,
the time part used will equal to 00:00:00.

<path>      The full path or storage URL to the Swift object.
Example: /v1/AUTH_account/c/o
or: http://saio:8080/v1/AUTH_account/c/o
<key>      The secret temporary URL key set on the Swift cluster.
To set a key, run 'swift post -m
"Temp-URL-Key:b3968d0207b54ece87cccc06515a89d4"'

Optional arguments:
--absolute      Interpret the <time> positional argument as a Unix
                timestamp rather than a number of seconds in the
                future. If an ISO 8601 timestamp is passed for <time>,
                this argument is ignored.
--prefix-based  If present, a prefix-based temporary URL will be
                generated.
--iso8601       If present, the generated temporary URL will contain an
                ISO 8601 UTC timestamp instead of a Unix timestamp.
--ip-range      If present, the temporary URL will be restricted to the
                given ip or ip range.

[root@objectcluster ~]#
```

We will generate a tempurl for our object (hosts) for a duration of 3600 seconds after which access will be removed. For this we will need the Storage URL using the swift stat command and our secret.

```
[root@objectcluster ~]# swift tempurl GET 3600
http://s3server.scale.com:8080/v1/AUTH_224f53da12e04b7893783821a34c171f/webbucket/hosts mysecret123
```

Output for the above command is as follows:

```
http://s3server.scale.com:8080/v1/AUTH_224f53da12e04b7893783821a34c171f/webbucket/hosts?temp_url_sig=1f93a7754e7412
b553990eeb3c94574f6d068027&temp_url_expires=1648495981
```

We can now use the output URL to access our object via the internet.

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

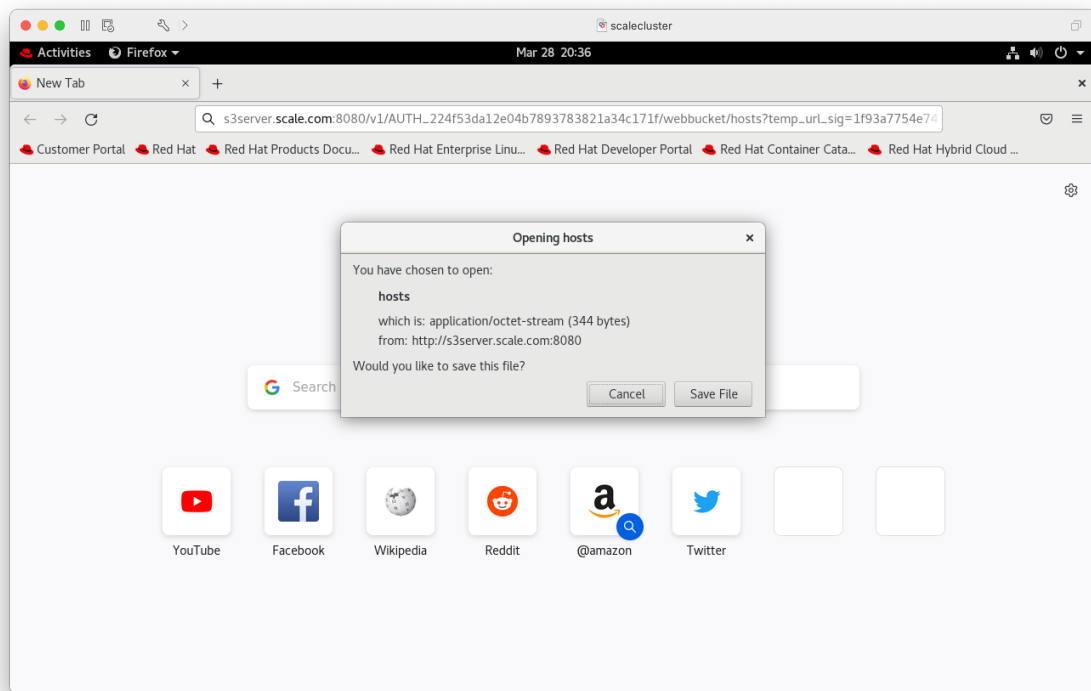


Figure 14: Accessing an object using tempurl via http or https

And there you have it. The ability to share objects via the internet. You might need to do this to share information with 3rd parties for example. You can also grant them the ability to PUT an object into your bucket. Refer to the *tempurl* usage posted above.

Swift Object Storage and IBM Spectrum Scale

Before we dive into the details of how to enable Object compression or perform ILM on Objects in Spectrum Scale, we need to first understand some basic concepts of OpenStack Swift and how they are implemented in Spectrum Scale. This understanding of how Swift storage elements are mapped in Spectrum Scale will also help us when we cover Object tiering later in this document. Most of the information detailed below was sourced from the Swift publicly available documentation which can be found at the following URL https://docs.openstack.org/swift/latest/overview_architecture.html.

The Swift Object storage system organizes data in a hierarchy as follows:

Account – this is the top level of the hierarchy and is synonymous with a project or tenant. An account defines the namespace for containers. You can have the same container names in different accounts. It is probably easier to think of a Swift Account as a storage location. It is not to be confused with an identity (e.g. a Swift user).

Container – A container provides a namespace for Objects. You can have the same objects in two different containers. In Swift, you can define as many containers as you like within an account. You can define access control and storage policies at the container level.

Objects – An Object stores data content like documents and images etc. You can also store custom metadata with each object. You can define as many objects as you like within a container.

Swift uses the concept of a ring to map the name of entities stored on disk and their physical location. There are separate rings for accounts and containers (though these can be combined). There is also one ring per object storage policy. The Ring maintains this mapping through the use of regions, zones, devices, partitions and replicas.

A Swift Proxy server ties the entire Swift architecture together. For each client request, it will look up the location of the account, container or object in the ring and route the request accordingly. Partitions store objects, account and container databases and help manage locations where data lives in the cluster. Basically, the object namespace is mapped to a number of partitions and each partition holds one or more objects. There are 3 servers that run on storage nodes in a Swift cluster. The first is the Object Server which stores, retrieves and deletes objects. Next you have the container server which lists objects for a specific container (this information is stored in a SQLite database). Lastly, you have the account server which lists containers for a specific account (this information is also stored in a SQLite database). The entire architecture can be depicted as follows:

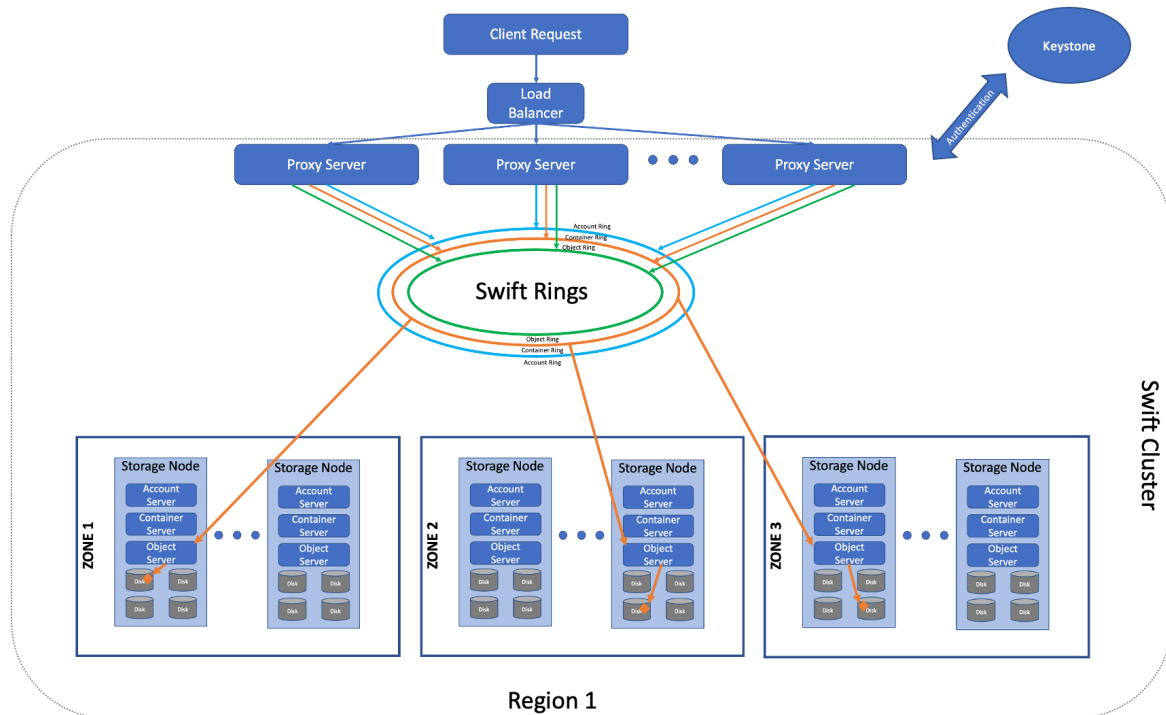


Figure 15: OpenStack Swift High-level Architecture

Swift Storage Policies allow for segmenting the cluster for various purposes by creating multiple object rings. Different devices can belong to different object rings. Once created, you can associate a Storage Policy during container creation. A container can only belong to a single Storage Policy and once created, a policy cannot be deleted. Essentially, Storage Policies allow for segregating object storage within a cluster. You might want to do this for different reasons which include:

- Different levels of durability – 2 replicas versus 3
- Performance – Use SSDs for account and container rings or for a very low-latency object ring
- Grouping Storage Nodes – place specific data only within a specific geographic location
- Different Storage Implementations - Group storage nodes that use a different Diskfile (e.g. GlusterFS) and direct traffic just to those nodes
- Different read and write affinity settings – Proxy servers can be used to define different read and write affinity settings for each policy

In the context of Spectrum Scale, a new object ring is created for each new Storage Policy. The same Spectrum Scale filesystem is used but a new independent fileset is created to house this new object ring. Typically, you would create new storage policies in Spectrum Scale if you want to make use of compression, encryption or make use of unified file access. Let's take a more detailed look at how Spectrum Scale maps out the Swift storage layers. By default, Spectrum Scale creates 3 separate rings. One each for the account, containers and objects.

Let's look at the Spectrum Scale implementation of Swift and the configuration of each ring. Firstly, the location of each ring. For the account ring, the location is our *object filesystem/object fileset/ac* directory. The account/container/object configuration files are located in the */etc/swift* directory.

```
[root@objectcluster swift]# cat account-server.conf
[DEFAULT]

# Make sure your swift-ring-builder arguments match the bind_ip and bind_port.
# You almost certainly do not want to listen just on loopback unless testing.
# However, you want to keep port 6202 if SELinux is enabled.
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
bind_ip = 0.0.0.0
bind_port = 6202

workers = 2
devices = /ibm/objectfs/object_fileset/ac
mount_check = false
log_level = ERROR

[pipeline:main]
pipeline = account-server

[app:account-server]
use = egg:swift#account

[account-replicator]

[account-auditor]

[account-reaper]
```

For the container ring, we have the same location as the account ring (which implies they are combined). The location is *object filesystem/object fileset/ac* directory.

```
[root@objectcluster swift]# cat container-server.conf
[DEFAULT]

# Make sure your swift-ring-builder arguments match the bind_ip and bind_port.
# You almost certainly do not want to listen just on loopback unless testing.
# However, you want to keep port 6201 if SELinux is enabled.
bind_ip = 0.0.0.0
bind_port = 6201

workers = 2
devices = /ibm/objectfs/object_fileset/ac
mount_check = false
log_level = ERROR

[pipeline:main]
pipeline = container-server

[app:container-server]
use = egg:swift#container

[container-replicator]

[container-updater]

[container-auditor]

[container-sync]

[container-sharder]
```

Lastly, for the object ring the location is *object filesystem/object fileset/o* directory.

```
[root@objectcluster swift]# cat /etc/swift/object-server.conf
[DEFAULT]

# Make sure your swift-ring-builder arguments match the bind_ip and bind_port.
# You almost certainly do not want to listen just on loopback unless testing.
# However, you want to keep port 6200 if SELinux is enabled.
bind_ip = 0.0.0.0
bind_port = 6200

workers = 3
devices = /ibm/objectfs/object_fileset/o
mount_check = false
log_level = ERROR

[pipeline:main]
pipeline = object-server

[app:object-server]
use = egg:swift#object

[object-replicator]
reclaim_age = 30
run_pause = 14400
sync_method = ssync

[object-updater]

[object-auditor]

[object-reconstructor]
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

Note the workers parameter in the above configuration files. We will cover the optimal tuning values later in this paper.

Let us now look at the exact ring layout. The list of devices has the following keys:

| | | |
|--------|---------|---|
| id | integer | The index into the list of devices. |
| zone | integer | The zone in which the device resides. |
| region | integer | The region in which the zone resides. |
| weight | float | The relative weight of the device in comparison to other devices. This usually corresponds directly to the amount of disk space the device has compared to other devices. For instance a device with 1 terabyte of space might have a weight of 100.0 and another device with 2 terabytes of space might have a weight of 200.0. This weight can also be used to bring back into balance a device that has ended up with more or less data than desired over time. A good average weight of 100.0 allows flexibility in lowering the weight later if necessary. |
| ip | string | The IP address or hostname of the server containing the device. |
| port | int | The TCP port on which the server process listens to serve requests for the device. |
| device | string | The on-disk name of the device on the server. For example: sdb1 |
| meta | string | A general-use field for storing additional information for the device. This information isn't used directly by the server processes, but can be useful in debugging. For example, the date and time of installation and hardware manufacturer could be stored here. |

Figure 16: Swift Ring Device Keys (https://docs.openstack.org/swift/latest/overview_ring.html)

For the account and container rings (which are combined), we can see that this ring has been created with a partition power of 14 (16384 partitions), a replica value of 1 and 0 hours minimum time between moving a partition more than once. A single replica is used in Spectrum Scale as we assume the NSDs that make up the object filesystem are RAID protected and combined with AFM or Spectrum Scale stretched clustering will provide enough resilience for our object implementation. IBM Spectrum Scale Object also supports a multi-region configuration should this be required for extra availability though this is not covered in this document. More detail on multi-region support in Spectrum Scale can be found here <https://www.ibm.com/docs/en/spectrum-scale/5.1.3?topic=deployment-planning-multi-region-object>.

```
[root@objectcluster ~]# /usr/bin/swift-ring-builder /etc/swift/account.builder
/etc/swift/account.builder, build version 129, id cc300d883c3f4750a1431977081210c9
16384 partitions, 1.000000 replicas, 1 regions, 1 zones, 128 devices, 0.00 balance, 0.00 dispersion
The minimum number of hours before a partition can be reassigned is 0 (0:00:00 remaining)
The overload factor is 0.00% (0.000000)
Ring file /etc/swift/account.ring.gz is up-to-date
Devices:  id region zone      ip address:port  replication ip:port  name weight partitions balance flags meta
          0      1      1 192.168.226.252:6202 192.168.226.252:6202 z1device1  1.00      128      0.00
          9      1      1 192.168.226.252:6202 192.168.226.252:6202 z1device10 1.00      128      0.00
         99      1      1 192.168.226.252:6202 192.168.226.252:6202 z1device100 1.00      128      0.00
        100      1      1 192.168.226.252:6202 192.168.226.252:6202 z1device101 1.00      128      0.00
        101      1      1 192.168.226.252:6202 192.168.226.252:6202 z1device102 1.00      128      0.00
        102      1      1 192.168.226.252:6202 192.168.226.252:6202 z1device103 1.00      128      0.00
          .
          .
          .
         95      1      1 192.168.226.252:6202 192.168.226.252:6202 z1device96  1.00      128      0.00
         96      1      1 192.168.226.252:6202 192.168.226.252:6202 z1device97  1.00      128      0.00
         97      1      1 192.168.226.252:6202 192.168.226.252:6202 z1device98  1.00      128      0.00
         98      1      1 192.168.226.252:6202 192.168.226.252:6202 z1device99  1.00      128      0.00
```

The container ring is basically the same as above.

```
[root@objectcluster ~]# /usr/bin/swift-ring-builder /etc/swift/container.builder
/etc/swift/container.builder, build version 129, id 8d6b8b4ca5fe49e88419917c9e11e700
16384 partitions, 1.000000 replicas, 1 regions, 1 zones, 128 devices, 0.00 balance, 0.00 dispersion
The minimum number of hours before a partition can be reassigned is 0 (0:00:00 remaining)
The overload factor is 0.00% (0.000000)
Ring file /etc/swift/container.ring.gz is up-to-date
Devices:  id region zone      ip address:port  replication ip:port  name weight partitions balance flags meta
          0      1      1 192.168.226.252:6201 192.168.226.252:6201 z1device1  1.00      128      0.00
          9      1      1 192.168.226.252:6201 192.168.226.252:6201 z1device10 1.00      128      0.00
         99      1      1 192.168.226.252:6201 192.168.226.252:6201 z1device100 1.00      128      0.00
        100      1      1 192.168.226.252:6201 192.168.226.252:6201 z1device101 1.00      128      0.00
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

| | | | | | | | | |
|-----|------|---|----------------------|----------------------|-------------|------|-----|------|
| 101 | 1 | 1 | 192.168.226.252:6201 | 192.168.226.252:6201 | z1device102 | 1.00 | 128 | 0.00 |
| 102 | 1 | 1 | 192.168.226.252:6201 | 192.168.226.252:6201 | z1device103 | 1.00 | 128 | 0.00 |
| . | | | | | | | | |
| . | | | | | | | | |
| 128 | 0.00 | | | | | | | |
| 96 | 1 | 1 | 192.168.226.252:6201 | 192.168.226.252:6201 | z1device97 | 1.00 | 128 | 0.00 |
| 97 | 1 | 1 | 192.168.226.252:6201 | 192.168.226.252:6201 | z1device98 | 1.00 | 128 | 0.00 |
| 98 | 1 | 1 | 192.168.226.252:6201 | 192.168.226.252:6201 | z1device99 | 1.00 | 128 | 0.00 |

Lastly, the object ring is configured as follows.

```
[root@objectcluster swift]# /usr/bin/swift-ring-builder /etc/swift/object.builder
/etc/swift/object.builder, build version 129, id e37cbe05236c451094502206a709874d
16384 partitions, 1.000000 replicas, 1 regions, 1 zones, 128 devices, 0.00 balance, 0.00 dispersion
The minimum number of hours before a partition can be reassigned is 0 (0:00:00 remaining)
The overload factor is 0.00% (0.000000)
Ring file /etc/swift/object.ring.gz is up-to-date
Devices:  id region zone      ip address:port  replication ip:port  name weight partitions balance flags meta
          0      1      1 192.168.226.252:6200 192.168.226.252:6200 z1device1  1.00      128      0.00
          9      1      1 192.168.226.252:6200 192.168.226.252:6200 z1device10 1.00      128      0.00
         99      1      1 192.168.226.252:6200 192.168.226.252:6200 z1device100 1.00      128      0.00
        100      1      1 192.168.226.252:6200 192.168.226.252:6200 z1device101 1.00      128      0.00
        101      1      1 192.168.226.252:6200 192.168.226.252:6200 z1device102 1.00      128      0.00
        102      1      1 192.168.226.252:6200 192.168.226.252:6200 z1device103 1.00      128      0.00
          .
          .
          .
        128      0.00
          96      1      1 192.168.226.252:6200 192.168.226.252:6200 z1device97  1.00      128      0.00
          97      1      1 192.168.226.252:6200 192.168.226.252:6200 z1device98  1.00      128      0.00
          98      1      1 192.168.226.252:6200 192.168.226.252:6200 z1device99  1.00      128      0.00
```

The Swift Object directory structure is defined as follows:

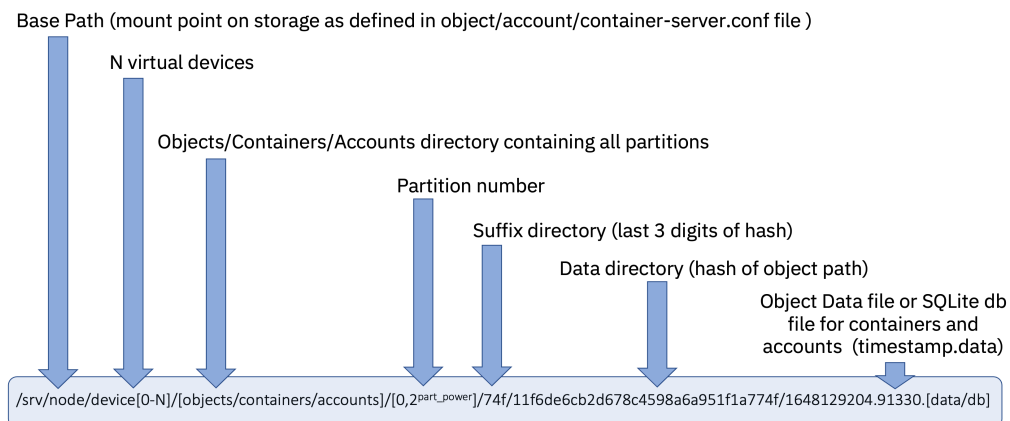


Figure 17: Swift directory structure for accounts/containers and objects

In the context of Spectrum Scale, the base path for the account and container rings is /ibm/objectfs/object_fileset/ac and the base path for the object ring is /ibm/objectfs/object_fileset/o. If we explore the directory structure on Spectrum Scale for the account and container rings we get:

```
[root@objectcluster object_fileset]# tree ac
ac
├── z1device1
│   └── containers
│       └── tmp
├── z1device10
│   └── containers
│       ├── 9826
│       │   └── e22
│       │       ├── 99888d0c028fc696a199aea94c74ee22
│       │       ├── 99888d0c028fc696a199aea94c74ee22.db
│       │       └── 99888d0c028fc696a199aea94c74ee22.db.pending
│       └── tmp
├── z1device100
├── z1device101
│   └── containers
│       ├── 1056
│       │   └── d1e
│       │       └── 10821f8b09b51cb652901ba88595cd1e
```

```

├───┬── 10821f8b09b51cb652901ba88595cd1e.db
│   └── 10821f8b09b51cb652901ba88595cd1e.db.pending
├── tmp
├── zldevice102
│   ├── containers
│   │   └── tmp
├── zldevice103
│   ├── containers
│   │   └── tmp
├── zldevice104
│   └── containers
│       └── tmp
.
.
.
├── zldevice75
│   ├── accounts
│   │   ├── 7997
│   │   │   └── d03
│   │   │       └── 7cf4fe52291c81cd1b70f1988c048d03
│   │   │           ├── 7cf4fe52291c81cd1b70f1988c048d03.db
│   │   │           └── 7cf4fe52291c81cd1b70f1988c048d03.db.pending
│   └── tmp
├── zldevice76
│   ├── containers
│   │   ├── 14299
│   │   │   └── aa0
│   │   │       └── df6cd23997422fbe0708c99cd60f4aa0
│   │   │           ├── df6cd23997422fbe0708c99cd60f4aa0.db
│   │   │           └── df6cd23997422fbe0708c99cd60f4aa0.db.pending
│   └── tmp
.
.
.
```

```
[root@objectcluster object_filesset]# tree o
o
├── zidevice1
│   └── objects
│       ├── 11405
│       │   ├── 9e8
│       │   │   └── b2368985dc8805277a79dfc8a62149e8
│       │   │       └── 1648370442.47661.data
│       │   ├── hashes.invalid
│       │   └── hashes.pkl
│       ├── 11456
│       │   ├── hashes.invalid
│       │   └── hashes.pkl
│       ├── 1149
│       │   ├── hashes.invalid
│       │   └── hashes.pkl
│       ├── 11733
│       │   ├── f82
│       │   │   ├── b757b56ebd11b7f784f6ad19a593ff82
│       │   │   └── 1648370267.78878.data
│       │   ├── hashes.invalid
│       │   └── hashes.pkl
│       ├── 11922
│       │   ├── hashes.invalid
│       │   └── hashes.pkl
│       ├── 11938
│       │   ├── a23
│       │   │   ├── ba89c1bf9277f892e51d6664cce67a23
│       │   │   └── 1648369197.52914.data
│       │   ├── hashes.invalid
│       │   └── hashes.pkl
└── .
└── .
```

```
[root@scalecluster ~]# s3cmd mb s3://testbucket
Bucket 's3://testbucket/' created
[root@scalecluster ~]# s3cmd ls
2009-02-03 16:45 s3://testbucket
[root@scalecluster ~]#
```

61

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
[root@scalecluster ~]# echo "This is a test object" > object.txt ; cat s3cmd-2.2.0.tar >> object.txt
[root@scalecluster ~]# ls -al object.txt
-rw-r--r--. 1 root root 573462 Mar 25 22:12 object.txt
[root@scalecluster ~]# cat object.txt | more
This is a test object
[root@scalecluster ~]# s3cmd put object.txt s3://testbucket
upload: 'object.txt' -> 's3://testbucket/object.txt' [1 of 1]
 573462 of 573462 100% in 0s 3.00 MB/s done
[root@scalecluster ~]# s3cmd ls s3://testbucket
2022-03-25 20:13 573462 s3://testbucket/object.txt
[root@scalecluster ~]#
```

To find the exact location of our object, we need to use the OpenStack Swift CLI. We firstly need to source the correct Swift profile and get our account name.

```
[root@objectcluster ~]# source openrc_s3
[root@objectcluster ~]# swift stat
Account: AUTH_224f53da12e04b7893783821a34c171f
Containers: 1
Objects: 1
Bytes: 22
Containers in policy "policy-0": 1
Objects in policy "policy-0": 1
Bytes in policy "policy-0": 22
Content-Type: application/json; charset=utf-8
X-Timestamp: 1647870922.58997
Accept-Ranges: bytes
X-Trans-Id: txf2eaf23f3be64a689833b-00623e20d1
X-OpenStack-Request-Id: txf2eaf23f3be64a689833b-00623e20d1
```

Now that we have the required information, we can use the swift-get-nodes tool to locate where a particular account, container or object is located within the Swift cluster. The location of the object is reported in the NON-HANDOFF location.

```
[root@objectcluster ~]# swift-get-nodes /etc/swift/object.ring.gz AUTH_224f53da12e04b7893783821a34c171f testbucket
object.txt

Account AUTH_224f53da12e04b7893783821a34c171f
Container testbucket
Object object.txt

Partition 9203
Hash 8fce87e26840ea2b3c73604c40c58bca

Server:Port Device 192.168.226.252:6200 z1device50
Server:Port Device 192.168.226.252:6200 z1device71 [Handoff]

curl -g -I -XHEAD
"http://192.168.226.252:6200/z1device50/9203/AUTH_224f53da12e04b7893783821a34c171f/testbucket/object.txt"
curl -g -I -XHEAD
"http://192.168.226.252:6200/z1device71/9203/AUTH_224f53da12e04b7893783821a34c171f/testbucket/object.txt" #
[Handoff]

Use your own device location of servers:
such as "export DEVICE=/srv/node"
ssh 192.168.226.252 "ls -lah ${DEVICE:-/srv/node*}/z1device50/objects/9203/bca/8fce87e26840ea2b3c73604c40c58bca"
ssh 192.168.226.252 "ls -lah ${DEVICE:-/srv/node*}/z1device71/objects/9203/bca/8fce87e26840ea2b3c73604c40c58bca" #
[Handoff]

note: `/srv/node*' is used as default value of `devices`, the real value is set in the config file on each storage
node.
[root@objectcluster ~]# swift-get-nodes /etc/swift/object.ring.gz AUTH_224f53da12e04b7893783821a34c171f testbucket
object.txt

Account AUTH_224f53da12e04b7893783821a34c171f
Container testbucket
Object object.txt

Partition 9203
Hash 8fce87e26840ea2b3c73604c40c58bca

Server:Port Device 192.168.226.252:6200 z1device50
Server:Port Device 192.168.226.252:6200 z1device71 [Handoff]

curl -g -I -XHEAD
"http://192.168.226.252:6200/z1device50/9203/AUTH_224f53da12e04b7893783821a34c171f/testbucket/object.txt"
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
curl -g -I -XHEAD
"http://192.168.226.252:6200/zldevice71/9203/AUTH_224f53da12e04b7893783821a34c171f/testbucket/object.txt" #
[Handoff]

Use your own device location of servers:
such as "export DEVICE=/srv/node"
ssh 192.168.226.252 "ls -lah ${DEVICE}:/srv/node*/z1device50/objects/9203/bca/8fce87e26840ea2b3c73604c40c58bca"
ssh 192.168.226.252 "ls -lah ${DEVICE}:/srv/node*/z1device71/objects/9203/bca/8fce87e26840ea2b3c73604c40c58bca" #
[Handoff]

note: `/srv/node*' is used as default value of `devices`, the real value is set in the config file on each storage
node.
[root@objectcluster ~]#
```

The location of our test object (object.txt) is highlighted above. If we navigate to this directory location we should be able to find our object (remember, our base path is OBJECTFS/OBJECT FILESET/o).

```
[root@objectcluster object_fileset]# cd
/ibm/objectfs/object_fileset/o/z1device50/objects/9203/bca/8fce87e26840ea2b3c73604c40c58bca
[root@objectcluster 8fce87e26840ea2b3c73604c40c58bca]# ls -l
total 568
-rwxr-xr-x. 1 swift swift 573462 Mar 25 22:13 1648239193.58174.data
[root@objectcluster 8fce87e26840ea2b3c73604c40c58bca]# cat 1648239193.58174.data
This is a test object
[root@objectcluster 8fce87e26840ea2b3c73604c40c58bca]#
```

We now know how to locate an object in our Spectrum Scale filesystem. This ability will come in handy later when we cover object compression and ILM. We will now discuss how Storage Policies are implemented in Spectrum Scale. You might also notice from the above example that by default objects are store unencrypted in Spectrum Scale.

As discussed earlier, each new Storage Policy requires a new object ring. In Spectrum Scale, the new object ring is located on the same filesystem as existing rings but a new independent fileset is created for the new policy. This separation of policies using different independent filesets will allow us to implement advanced Spectrum Scale functions like ILM, however it is not natively supported in Swift. To get around this, Spectrum Scale uses soft links. Remember the base path for object rings as defined in the `/etc/swift/object-server.conf` file is `/ibm/objectfs/object_fileset/o`. For new policies, Swift will create them by default in the same fileset as existing policies unless we split them out. For example, in Spectrum Scale, policies are located as follows:

| | | |
|---------------------|------------------------------|----------------------|
| <object filesystem> | <object policy fileset><dir> | <device folders> |
| /ibm/objectfs | /object_fileset/o | /zXdeviceX |
| /ibm/objectfs | /SP2/o | /<SP2index>zXdeviceX |

In our example, each policy is located in a separate independent fileset and each subsequent fileset we use is linked to the original fileset as defined in the `/etc/swift/object-server.conf` file. So in our example:

Initial Storage Policy (policy-0)

```
/ibm/objectfs/object_fileset/o/zXdeviceX
```

New Storage Policy which is called `CompressionObjectfs` with a fileset name of `obj_CompressionObjectfs`

```
/ibm/objectfs/object_fileset/o/<SP2index>zXdeviceX links to our independent fileset that was created for this new
object ring which is /ibm/objectfs/obj_CompressionObjectfs/<SP2index>zXdeviceX.
```

This is what the `/ibm/objectfs/object_fileset/o` (which is the base path defined in the `/etc/swift/object-server.conf`) looks like with this new policy:

```
[root@objectcluster o]# ls -al | more
total 145
drwxr-x---. 130 swift swift 16384 Mar 25 22:23 .
```


Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
drwxr-x--- 5 swift swift 4096 Mar 27 09:15 ..
lrwxrwxrwx. 1 swift swift 59 Mar 25 22:23 s18722203250z1device1 -> /ibm/obj
ectfs/obj_CompressionObjectfs/s18722203250z1device1
lrwxrwxrwx. 1 swift swift 60 Mar 25 22:23 s18722203250z1device10 -> /ibm/ob
jectfs/obj_CompressionObjectfs/s18722203250z1device10
lrwxrwxrwx. 1 swift swift 61 Mar 25 22:23 s18722203250z1device100 -> /ibm/o
bjectfs/obj_CompressionObjectfs/s18722203250z1device100
lrwxrwxrwx. 1 swift swift 61 Mar 25 22:23 s18722203250z1device101 -> /ibm/o
bjectfs/obj_CompressionObjectfs/s18722203250z1device101
.
.
drwxr-xr-x. 3 swift swift 4096 Mar 20 03:20 z1device90
drwxr-xr-x. 3 swift swift 4096 Mar 20 03:20 z1device91
drwxr-xr-x. 3 swift swift 4096 Mar 20 03:20 z1device92
drwxr-xr-x. 3 swift swift 4096 Mar 20 03:20 z1device93
drwxr-xr-x. 3 swift swift 4096 Mar 20 03:20 z1device94
drwxr-xr-x. 3 swift swift 4096 Mar 20 03:20 z1device95
drwxr-xr-x. 3 swift swift 4096 Mar 20 03:20 z1device96
drwxr-xr-x. 3 swift swift 4096 Mar 20 03:20 z1device97
drwxr-xr-x. 3 swift swift 4096 Mar 20 03:20 z1device98
drwxr-xr-x. 3 swift swift 4096 Mar 20 03:20 z1device99
[root@objectcluster o]#
```

And the independent fileset for our second storage policy directory listing looks like:

```
[root@objectcluster obj_CompressionObjectfs]# ls -al | more
total 337
drwxr-xr-x. 130 swift swift 16384 Mar 25 22:23 .
drwxr-xr-x. 5 root root 262144 Mar 27 09:02 ..
drwxr-xr-x. 3 swift swift 4096 Mar 25 22:24 s18722203250z1device1
drwxr-xr-x. 3 swift swift 4096 Mar 25 22:24 s18722203250z1device10
drwxr-xr-x. 3 swift swift 4096 Mar 25 22:24 s18722203250z1device100
drwxr-xr-x. 3 swift swift 4096 Mar 25 22:24 s18722203250z1device101
drwxr-xr-x. 3 swift swift 4096 Mar 25 22:24 s18722203250z1device102
drwxr-xr-x. 3 swift swift 4096 Mar 25 22:24 s18722203250z1device103
.
.
.
```

Now that we understand how Spectrum Scale maps out Swift Storage Policies, we are able to firstly know where objects are located and secondly to perform advanced Spectrum Scale ILM functions if required.

Spectrum Scale Object Compression

Object compression is slightly different to normal Spectrum Scale file compression. Firstly, object compression is available only through Cluster Export Services (CES) on protocol nodes and is implemented via the `mmobj` command. File Compression is performed on NSD servers and is policy driven using the `mmpolicy` command or manually using the `mmchattr` command. Another key difference is that you can defer file compression or decompression until a time where the system is not loaded with processes and I/O. We will use the `mmobj` command which has the following syntax.

```
mmobj policy create PolicyName [-f FilesetName]
[--file-system FilesystemName] [-i MaxNumInodes]
{[--enable-compression --compression-schedule CompressionSchedule]}
{[--enable-encryption --encryption-keyfile EncryptionKeyFileName [--force-rule-append]]}
[--enable-file-access]
```

Note, you can specify the existing Object fileset using the `-f` command though this will prevent you from being able to implement some ILM functions. Let us now create a new Storage Policy that will enable the use of object compression. We will define a new fileset for the object ring as `CompressionObjectfs` and set the compression schedule to every minute (in a real world environment the recommended frequency is at least one hour or longer).

```
[root@objectcluster ~]# mmobj policy create CompressionObjectfs --enable-compression --compression-schedule
"*.:*:*"
[I] It is recommended to use a compression schedule frequency that is one hour or longer.
[I] Getting latest configuration from ccr
[I] Creating fileset objectfs:obj_CompressionObjectfs
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
Attention: the specified maximum number of files (8000000) is too high.
The amount of available disk space allows for a maximum of 4102144 files.
[I] Creating new unique index and building the object rings
[I] Updating the configuration
[I] Uploading the changed configuration
[root@objectcluster ~]#
```

We can check that new policy has been created successfully as follows:

```
[root@objectcluster 8fce87e26840ea2b3c73604c40c58bca]# mmobj policy list -v
```

| Index | Name | Default | Deprecated | Fileset | Fileset Path |
|-------------|--|---------|------------|-------------------------|---------------------------------------|
| Functions | Function Details | | | File System | |
| 0 | policy-0 | yes | | object_fileset | /ibm/objectfs/object_fileset |
| 18722203250 | CompressionObjectfs | | | obj_CompressionObjectfs | /ibm/objectfs/obj_CompressionObjectfs |
| compression | compression-schedule="*:*:*";regions="1" | | | objectfs | |

```
[root@objectcluster 8fce87e26840ea2b3c73604c40c58bca]#
```

Note, our new policy is not set to be the default policy just yet so all new containers will still belong to Policy-0 which is the standard policy created when the object protocol is enabled in Spectrum Scale. This policy has a policy index of 0 which maps to the object ring that is available in all Swift installations (which is `/etc/swift/object.ring.gz`). Any policy can be set to the default which means Swift will use this policy for all new container creations. Policy-0 is slightly different to the default policy in that it is used for accessing all pre-storage-policy containers which don't have a policy – in this case we would not use the default as it might not have the same policy as legacy containers. When no other policies are defined, Swift will always choose Policy-0 as the default. Back to our example, our Storage Policy index for our new policy is 18722203250. Let's look at the fileset definition that was created and the directory structure of our object filesystem.

```
[root@objectcluster ~]# mmlsfileset objectfs
Filesets in file system 'objectfs':
Name      Status  Path
root      Linked  /ibm/objectfs
object_fileset  Linked  /ibm/objectfs/object_fileset
obj_CompressionObjectfs  Linked  /ibm/objectfs/obj_CompressionObjectfs
[root@objectcluster ~]# mmlsfileset objectfs obj_CompressionObjectfs -X
Filesets in file system 'objectfs':

Attributes for fileset obj_CompressionObjectfs:
=====
Status      Linked
Path        /ibm/objectfs/obj_CompressionObjectfs
Id          2
Root inode  262147
Parent Id   0
Created     Fri Mar 25 22:23:07 2022
Comment
Inode space 2
Maximum number of inodes 4102144
Allocated inodes 67584
Permission change flag  chmodAndSetacl
IAM mode     off
afm-associated  No
Permission inherit flag  inheritAclOnly
-:-:[root@objectcluster ~]#
```

We now have two filesets to cater for the 2 object rings required by the different Storage Policies.

```
[root@objectcluster objectfs]# ls -al
total 274
drwxr-xr-x.  5 root root 262144 Mar 25 22:24 .
drwxr-xr-x.  4 root root   43 Mar 20 02:47 ..
drwxr-xr-x.  2 root root  4096 Mar 25 22:42 .mmSharedTmpDir
drwxr-xr-x. 130 swift swift 16384 Mar 25 22:23 obj_CompressionObjectfs
drwxr-xr-x.  4 swift swift  4096 Mar 20 03:15 object_fileset
dr-xr-xr-x.  2 root root   8192 Jan  1 1970 .snapshots
[root@objectcluster objectfs]#
```

The `mmobj` command created a new object ring for us. Let us query this new object ring. The ring configuration file should be `/etc/swift/object-<POLICY_INDEX>.ring.gz`.

```
[root@objectcluster obj_CompressionObjectfs]# /usr/bin/swift-ring-builder /etc/swift/object-18722203250.ring.gz
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
Note: using /etc/swift/object-18722203250.builder instead of /etc/swift/object-18722203250.ring.gz as builder file
/etc/swift/object-18722203250.builder, build version 129, id 19c4902501f84459af6f90eb286c8b77
16384 partitions, 1.000000 replicas, 1 regions, 1 zones, 128 devices, 0.00 balance, 0.00 dispersion
The minimum number of hours before a partition can be reassigned is 0 (0:00:00 remaining)
The overload factor is 0.00% (0.000000)
Ring file /etc/swift/object-18722203250.ring.gz is up-to-date
Devices:  id region zone      ip address:port  replication ip:port  name weight partitions balance flags meta
          0      1      1 192.168.226.252:6200 192.168.226.252:6200 s18722203250z1device1 1.00      128      0.00
          9      1      1 192.168.226.252:6200 192.168.226.252:6200 s18722203250z1device10 1.00      128
0.00
          99      1      1 192.168.226.252:6200 192.168.226.252:6200 s18722203250z1device100 1.00      128
0.00
          100      1      1 192.168.226.252:6200 192.168.226.252:6200 s18722203250z1device101 1.00      128
0.00
          101      1      1 192.168.226.252:6200 192.168.226.252:6200 s18722203250z1device102 1.00      128
0.00
          102      1      1 192.168.226.252:6200 192.168.226.252:6200 s18722203250z1device103 1.00      128
0.00
          .
          .
          95      1      1 192.168.226.252:6200 192.168.226.252:6200 s18722203250z1device96 1.00      128
0.00
          96      1      1 192.168.226.252:6200 192.168.226.252:6200 s18722203250z1device97 1.00      128
0.00
          97      1      1 192.168.226.252:6200 192.168.226.252:6200 s18722203250z1device98 1.00      128
0.00
          98      1      1 192.168.226.252:6200 192.168.226.252:6200 s18722203250z1device99 1.00      128
0.00
[root@objectcluster obj_CompressionObjectfs]#
```

This is as expected. We have the option now of changing our new policy to be the default so that all new containers will use this policy (and be compressed) or we can choose to use this policy by explicitly specifying X-Storage-Policy in our header at container creation. For our environment, we will use our new policy by explicitly specifying the X-Storage-Policy at container creation. You might want to do this if you don't want to compress all your containers by setting this policy as the default. Let's get started and obtain our X-Auth-Token using the script provided earlier called `get_token.sh`. We just need to make sure we populate the script variables with our `s3project` and `s3user` information.

```
[root@objectcluster obj_CompressionObjectfs]# cat openrc_s3

#####
# Get this from "source openrc" output and populate these variables
#####

export OS_PROJECT_NAME="s3project"
export OS_PROJECT_DOMAIN_NAME="Default"
export OS_USERNAME="s3user"
export OS_USER_DOMAIN_NAME="Default"
export OS_PASSWORD="Passw0rd"
export OS_AUTH_URL="http://192.168.226.252:35357/v3/auth/tokens"

[root@objectcluster scripts]# ./get_token.sh
HTTP/1.1 201 CREATED
Date: Fri, 25 Mar 2022 21:22:18 GMT
Server: Apache
Content-Length: 1923
X-Subject-Token: gAAAAABiPjKKlagVt4S2cfKQ_JmHjOJDvqnxB-
RboKMeMefedWvIkOb_EuoTgjUdOI55MZhyDa_ugn3E8q7PAdHaYcCruzL0uD_tNBokWc3xzwerS1K76_9u6hsRsvjYGBxU9Xd9oQA90PqhoerJiGqVV
Ns4CetlFVR_if-Cigt0A6cooNw0Fpw
Vary: X-Auth-Token
x-openstack-request-id: req-2be38324-b6fc-417c-8727-9b80a4b3042b
Content-Type: application/json

{"token": {"methods": ["password"], "user": {"domain": {"id": "default", "name": "Default"}, "id":
"7ed3923b2df64a5d94af66f1cfd6512c", "name": "s3user", "password expires at": null}, "audit ids":
["L9TYy19nSz26gIv2M_MWLA"], "expires_at": "2022-04-24T21:22:18.000000Z", "issued_at": "2022-03-
25T21:22:18.000000Z", "project": {"domain": {"id": "default", "name": "Default"}, "id":
"224f53dal2e04b7893783821a34c171f", "name": "s3project", "is_domain": false, "roles": [{"id":
"638ade0ff03146a48e6694585f72b4ec", "name": "ResellerAdmin"}, {"id": "ef507f8e0c1340d2b30f7e44ad5e9c53", "name":
"member"}, {"id": "57d271a0ebb44ccd816db3827571cdbc", "name": "reader"}, {"id": "25fea0bc18d04e64b79be9fecba2540e",
"name": "admin"}], "catalog": [{"endpoints": [{"id": "0118383f15c04e6dbaad7c37a6174573", "interface": "public",
"region_id": null, "url": "http://s3server.scale.com:5000/", "region": null}, {"id":
"e8c70c04ac0d4b9897f53cda2ed3501e", "interface": "internal", "region_id": null, "url":
"http://s3server.scale.com:35357/", "region": null}, {"id": "e3c2bd46af864346bc3a93795eb43d60", "interface":
"admin", "region_id": null, "url": "http://s3server.scale.com:35357/", "region": null}], "id":
"99789591b37d46dcb51964540b0535ea", "type": "identity", "name": "keystone"}, {"endpoints": [{"id":
"86263b30adb4f1139367febec288d0ca", "interface": "public", "region_id": "RegionOne", "url":
"http://s3server.scale.com:8080/v1/AUTH_224f53dal2e04b7893783821a34c171f", "region": "RegionOne"}, {"id":
"00463af2be364dcb884aef22f29e0bc", "interface": "internal", "region_id": "RegionOne", "url":
"http://s3server.scale.com:8080/v1/AUTH_224f53dal2e04b7893783821a34c171f", "region": "RegionOne"}, {"id":
"253746ebec024e7b964b2ca68930f7be", "interface": "admin", "region_id": "RegionOne", "url":
"http://s3server.scale.com:8080", "region": "RegionOne"}], "id": "ee8f841de5f74036b965d8ba27e53f96", "type":
"object-store", "name": "swift"}]}}[root@objectcluster scripts]#
```

Now we can use our X-Auth-Token to create a new container called compresscontainer which will be associated with our compression Storage Policy using the X-Storage-Policy header as follows:

```
[root@objectcluster scripts]# curl -i -X PUT -H "X-Auth-Token: gAAAAABiPjKKlagVt4S2cfKQ_JmHjOJDvqnxB-RboKMeMefeDWvik0b_EuoTgjUdOI55MZhYDa_ugN3E8q7PAdHaYcCruzL0uD_tNBokWc3xzwerS1K76_9u6hsRsvjYGBxU9Xd9oQA90PqhoerJiGqVVNs4CetlFVr_if-CiqtoA6cooNw0Fpw" -H "X-Storage-Policy: CompressionObjectfs" http://s3server.scale.com/v1/AUTH_224f53da12e04b7893783821a34c171f/compresscontainer
HTTP/1.1 201 Created
Content-Type: text/html; charset=UTF-8
Content-Length: 0
X-Trans-Id: tx68b29c3bdba7469bb8289-00623e32fc
X-OpenStack-Request-Id: tx68b29c3bdba7469bb8289-00623e32fc
Date: Fri, 25 Mar 2022 21:24:12 GMT
```

Using the Swift CLI, we can check if the container was correctly created and associated with our compression policy.

```
[root@objectcluster scripts]# swift list
compresscontainer
testbucket
[root@objectcluster scripts]# swift stat compresscontainer
bash: swift: command not found...
[root@objectcluster scripts]# swift stat compresscontainer
Account: AUTH_224f53da12e04b7893783821a34c171f
Container: compresscontainer
Objects: 0
Bytes: 0
Read ACL:
Write ACL:
Sync To:
Sync Key:
Content-Type: application/json; charset=utf-8
X-Timestamp: 1648243452.31523
Last-Modified: Fri, 25 Mar 2022 21:24:13 GMT
Accept-Ranges: bytes
X-Storage-Policy: CompressionObjectfs
X-Container-Sharding: False
X-Trans-Id: tx5a30612511144c8abb44e-00623e334a
X-OpenStack-Request-Id: tx5a30612511144c8abb44e-00623e334a
[root@objectcluster scripts]#
```

As an alternative, you can use the OpenStack Swift CLI to create containers with a specific storage policy as follows.

```
[root@objectcluster ~]# swift post -H "X-Storage-Policy:CompressionObjectfs" compressbucket
[root@objectcluster ~]# swift list
compressbucket
[root@objectcluster ~]# swift stat compressbucket
Account: AUTH_224f53da12e04b7893783821a34c171f
Container: compressbucket
Objects: 0
Bytes: 0
Read ACL:
Write ACL:
Sync To:
Sync Key:
Content-Type: application/json; charset=utf-8
X-Timestamp: 1648285938.16526
Last-Modified: Sun, 27 Mar 2022 05:57:35 GMT
Accept-Ranges: bytes
X-Storage-Policy: CompressionObjectfs
X-Container-Sharding: False
X-Trans-Id: txab93cd4f386345fb9782d-00623ffcde
X-OpenStack-Request-Id: txab93cd4f386345fb9782d-00623ffcde
[root@objectcluster ~]#
```

You cannot create a container via the S3API with an explicit Swift Storage Policy. AWS S3 implements storage classes on objects whereas Swift does it at the container level. AWS S3 uses pre-defined Storage Classes (e.g. Standard, Intelligent Tiering, S3 Glacier etc.). The best option is to pre-create all the required containers using curl (as we did above) that belong to a specific storage policy and then access it as per normal via the S3 API. If we query our newly created compresscontainer via the S3 API, there is no concept of Storage Policy.

```
[root@scalecluster ~]# aws s3api list-objects --endpoint-url=https://s3server.scale.com --bucket compresscontainer
{
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
"Contents": [
  {
    "Key": "object.txt",
    "LastModified": "2022-03-25T21:30:39.244Z",
    "ETag": "\"ef3d567699acee9280d917a52c83dc94\"",
    "Size": 573462,
    "StorageClass": "STANDARD",
    "Owner": {
      "DisplayName": "s3project:s3user",
      "ID": "s3project:s3user"
    }
  }
]
}
[root@scalecluster ~]#
```

To check if compression is working, let us upload a TXT file to our new container and then locate the file on the Spectrum Scale Object filesystem to check its attributes to make sure that it is compressed. Remember, we used the `swift-get-nodes` command to locate an object earlier.

```
[root@scalecluster ~]# s3cmd put object.txt s3://compresscontainer
upload: 'object.txt' -> 's3://compresscontainer/object.txt' [1 of 1]
573462 of 573462 100% in 0s 1200.73 KB/s done
[root@scalecluster ~]# s3cmd ls s3://compresscontainer
2022-03-25 21:30 573462 s3://compresscontainer/object.txt
[root@scalecluster ~]#
```

We now need to locate the object we uploaded via the S3 API on Swift. First, we get our account name.

```
[root@objectcluster ~]# swift stat
Account: AUTH_224f53da12e04b7893783821a34c171f
Containers: 2
Objects: 2
Bytes: 1146924
Containers in policy "policy-0": 1
Objects in policy "policy-0": 1
Bytes in policy "policy-0": 573462
Containers in policy "compressionobjectfs": 1
Objects in policy "compressionobjectfs": 1
Bytes in policy "compressionobjectfs": 573462
Content-Type: application/json; charset=utf-8
X-Timestamp: 1647870922.58997
Accept-Ranges: bytes
X-Trans-Id: tx8c97c5e1aad45fd11bd-00623e38f7
X-OpenStack-Request-Id: tx8c97c5e1aad45fd11bd-00623e38f7
[root@objectcluster ~]#
```

Our Storage Policy *CompressionObjectfs* has a policy index of 18722203250 so we need to query the `/etc/swift/object-18722203250.ring.gz` file as follows (`/etc/swift/object.ring.gz` refers to policy-0 as discussed earlier):

```
[root@objectcluster ~]# swift-get-nodes /etc/swift/object-18722203250.ring.gz AUTH_224f53da12e04b7893783821a34c171f
compresscontainer object.txt

Account AUTH_224f53da12e04b7893783821a34c171f
Container compresscontainer
Object object.txt

Partition 1436
Hash 167344ad0641c85e52524877d35a4502

Server:Port Device 192.168.226.252:6200 s18722203250z1device91
Server:Port Device 192.168.226.252:6200 s18722203250z1device70 [Handoff]

curl -g -I -XHEAD
"http://192.168.226.252:6200/s18722203250z1device91/1436/AUTH_224f53da12e04b7893783821a34c171f/compresscontainer/object.txt" -H "X-Backend-Storage-Policy-Index: 18722203250"
curl -g -I -XHEAD
"http://192.168.226.252:6200/s18722203250z1device70/1436/AUTH_224f53da12e04b7893783821a34c171f/compresscontainer/object.txt" -H "X-Backend-Storage-Policy-Index: 18722203250" # [Handoff]

Use your own device location of servers:
such as "export DEVICE=/srv/node"
ssh 192.168.226.252 "ls -lah ${DEVICE}:/srv/node*/s18722203250z1device91/objects-18722203250/1436/502/167344ad0641c85e52524877d35a4502"
ssh 192.168.226.252 "ls -lah ${DEVICE}:/srv/node*/s18722203250z1device70/objects-18722203250/1436/502/167344ad0641c85e52524877d35a4502" # [Handoff]
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
note: `/srv/node*` is used as default value of `devices`, the real value is set in the config file on each storage node.
[root@objectcluster ~]#
```

The location of our object *object.txt* is reported in the NON-HANDOFF line. If we navigate to this directory and query our object file with `mmlsattr` we get:

```
[root@objectcluster ~]# cd /ibm/objectfs/obj_CompressionObjectfs/s18722203250z1device91/objects-18722203250/1436/502/167344ad0641c85e52524877d35a4502
[root@objectcluster 167344ad0641c85e52524877d35a4502]# ls -al
total 161
drwxr-xr-x. 2 swift swift  4096 Mar 25 23:30 .
drwxr-xr-x. 3 swift swift  4096 Mar 25 23:30 ..
-rwxr-xr-x. 1 swift swift 573462 Mar 25 23:30 1648243839.24486.data
[root@objectcluster 167344ad0641c85e52524877d35a4502]#

[root@objectcluster 167344ad0641c85e52524877d35a4502]# mmlsattr -L ./1648243839.24486.data
file name:                ./1648243839.24486.data
metadata replication:     1 max 2
data replication:         1 max 2
immutable:                no
appendOnly:               no
flags:
storage pool name:        system
fileset name:              obj_CompressionObjectfs
snapshot name:
creation time:             Fri Mar 25 23:30:39 2022
Misc attributes:           ARCHIVE COMPRESSION (library z)
Encrypted:                 no
[root@objectcluster 167344ad0641c85e52524877d35a4502]#
```

The `mmlsattr` command displays the **COMPRESSION** flag in the Misc attributes output line. The flag is followed in parentheses by the name of the compression library that was used to compress the file. If this flag is present, it indicates that the file is compressed. Spectrum Scale uses the `zlib` compression library by default which favours compression over speed. As of Spectrum Scale V5, IBM offers the `lz4` compression library which is intended for primary or active data as this library favours speed over compression. You can specify the compression library to use explicitly via the `mmchattr` command or via `mmappypolicy`. The `mmobj` command used when creating a new compression storage policy defaults to `zlib` with no option to specify `lz4`.

The size of a compressed file after it has been compressed as reported by operating system commands like “`ls -l`” display the uncompressed size. You can use `du` or the `mmdf` command to display the actual compressed size. You can also make use of the `stat()` system call to find out how many blocks the file occupies. In our example, the file is reported as 573462 bytes or 560KB by the operating system using the `ls -al` command.

```
[root@objectcluster 167344ad0641c85e52524877d35a4502]# ls -al
total 161
drwxr-xr-x. 2 swift swift  4096 Mar 25 23:30 .
drwxr-xr-x. 3 swift swift  4096 Mar 25 23:30 ..
-rwxr-xr-x. 1 swift swift 573462 Mar 25 23:30 1648243839.24486.data
[root@objectcluster 167344ad0641c85e52524877d35a4502]#
```

The actual compressed size as reported by `du` is 160KB which translates to a compression ratio of about 3.45:1 (the file is a TXT file so we expect to get good compression).

```
[root@objectcluster ~]# du -k /ibm/objectfs/obj_CompressionObjectfs/s18722203250z1device91/objects-18722203250/1436/502/167344ad0641c85e52524877d35a4502/1648243839.24486.data
160      /ibm/objectfs/obj_CompressionObjectfs/s18722203250z1device91/objects-18722203250/1436/502/167344ad0641c85e52524877d35a4502/1648243839.24486.data
[root@objectcluster ~]#
```

If you want to set this new Storage Policy as the default, you can issue “`mmobj policy change CompressionObjectfs -default`”. This will deprecate Policy-0 and ensure all new containers are created as compressed. Remember, you cannot delete a Storage Policy once it is created. You can however, promote a policy to be the default and deprecate the unwanted policy.

As a quick test of compressed versus uncompressed performance, a script is included in Appendix A: testobj.sh which does the following:

- Create 2 temporary buckets using s3cmd (via S3API middleware on Swift)
- Create some temporary files for the testing
- Use PUT to upload the files to your chosen Object endpoint and measure the time taken and throughput
- Copy the objects from one container to another and measure the time taken and throughput
- Use GET to download the files from your chosen Object endpoint and measure the time taken and throughput
- Perform a cleanup of the temporary containers and object
- Generate a log file with the results

Note, there are some options available for testobj.sh one of which specifies the multipart chunk size (s3cmd by default uses a 15MB chunk size). As mentioned earlier, Swift can support a file size up to 5GB after which it will need to break the file up into segments or parts. By default, the OpenStack Swift proxy-server.conf file used for Spectrum Scale sets the largest chunk size to 1GB via the SLO middleware. You can override this if required. Using multipath uploads is a key feature of the AWS S3 API that allows you to upload a single object as a set of parts. Each part is a contiguous portion of the object's data. You can upload these parts independently and in any order. If transmission of any part fails, you can retransmit that part without affecting other parts. After all the parts are uploaded, AWS S3 assembles these parts and creates an object. Multipart uploads provide the following benefits:

- Improved throughput – you can upload parts in parallel to improve performance
- Quick Recovery from network issues – smaller part sizes minimize the impact of restarting a failed upload due to network errors
- Pause and Resume object uploads – you can upload objects over a time period. After you initiate a multipart upload, there is no expiry. You need to explicitly complete or stop the multipart upload
- Begin an upload before you know the file final object size – you can upload an object as you are creating it

If you have a stable network you can maximise your network bandwidth by using multipart uploads. If you have an unreliable network, you can use multipart uploads to avoid restarting uploads due to network errors. You can retry uploading only the parts that failed instead of the entire object. If you use the multipart upload capability via the S3API, you will find that Swift stores the files a bit differently. A manifest file is created in the target container and a new container is created for the various parts of the object. The manifest file allows for the download of the object as a single file. This new container is typically called [BUCKET_NAME]+segments. This is to ensure that listings of the container are not polluted with all the segment names. As an example, below is the output when querying the buckets created by the testobj.sh script.

```
[root@objectcluster obj_CompressionObjectfs]# swift list
testobj-5b548ef1
testobj-b175f758
testobj-b175f758+segments
[root@objectcluster obj_CompressionObjectfs]#
```

If we query the bucket testobj-b175f758 we will see our objects as a single entity. However, if we query the segments bucket which is auto created for us on a multipart upload via the S3API, then we will see all the individual parts that make up the objects. These parts are typically stored as [BUCKET_NAME]+segments/UPLOAD_ID/PART_NUMBER.

```
[root@objectcluster obj_CompressionObjectfs]# swift list testobj-b175f758
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
file1
file2
file3
file4
file5
[root@objectcluster obj_CompressionObjectfs]# swift list testobj-b175f758+segments
file1/ZjE4MGZjYTQtnZvJMC00NTMzLWEzMjQtnThmNTVjNDU2NTVh/1
file1/ZjE4MGZjYTQtnZvJMC00NTMzLWEzMjQtnThmNTVjNDU2NTVh/10
file1/ZjE4MGZjYTQtnZvJMC00NTMzLWEzMjQtnThmNTVjNDU2NTVh/2
file1/ZjE4MGZjYTQtnZvJMC00NTMzLWEzMjQtnThmNTVjNDU2NTVh/3
file1/ZjE4MGZjYTQtnZvJMC00NTMzLWEzMjQtnThmNTVjNDU2NTVh/4
file1/ZjE4MGZjYTQtnZvJMC00NTMzLWEzMjQtnThmNTVjNDU2NTVh/5
file1/ZjE4MGZjYTQtnZvJMC00NTMzLWEzMjQtnThmNTVjNDU2NTVh/6
file1/ZjE4MGZjYTQtnZvJMC00NTMzLWEzMjQtnThmNTVjNDU2NTVh/7
file1/ZjE4MGZjYTQtnZvJMC00NTMzLWEzMjQtnThmNTVjNDU2NTVh/8
file1/ZjE4MGZjYTQtnZvJMC00NTMzLWEzMjQtnThmNTVjNDU2NTVh/9
file2/MGQ3ZjJhMjQtnOTg2Yy00ZGVjLWJhMjMtYWU3ODdkMDY3ZTZj/1
file2/MGQ3ZjJhMjQtnOTg2Yy00ZGVjLWJhMjMtYWU3ODdkMDY3ZTZj/10
file2/MGQ3ZjJhMjQtnOTg2Yy00ZGVjLWJhMjMtYWU3ODdkMDY3ZTZj/2
file2/MGQ3ZjJhMjQtnOTg2Yy00ZGVjLWJhMjMtYWU3ODdkMDY3ZTZj/3
file2/MGQ3ZjJhMjQtnOTg2Yy00ZGVjLWJhMjMtYWU3ODdkMDY3ZTZj
.
.
.
file5/YzUzYzg2NDQtmWE4NC00ZDBjLWI5MTQtnZTQ0MWFhYjA1NWIl/4
file5/YzUzYzg2NDQtmWE4NC00ZDBjLWI5MTQtnZTQ0MWFhYjA1NWIl/5
file5/YzUzYzg2NDQtmWE4NC00ZDBjLWI5MTQtnZTQ0MWFhYjA1NWIl/6
file5/YzUzYzg2NDQtmWE4NC00ZDBjLWI5MTQtnZTQ0MWFhYjA1NWIl/7
file5/YzUzYzg2NDQtmWE4NC00ZDBjLWI5MTQtnZTQ0MWFhYjA1NWIl/8
file5/YzUzYzg2NDQtmWE4NC00ZDBjLWI5MTQtnZTQ0MWFhYjA1NWIl/9
[root@objectcluster obj_CompressionObjectfs]#
```

Back to the comparison of compressed versus uncompressed performance, the testobj.sh script was modified to insert a pause to allow the Spectrum Scale compression schedule to take effect (remember we set it to run every minute). You can check when the schedule runs by looking at the /var/log/messages file on RHEL. An example is provided below.

```
[root@objectcluster log]# grep compressionrule messages
.
.
.
Apr 19 11:21:28 objectcluster mmfs[96572]: CLI root root [ENTRY, CHANGE] 'mmapplypolicy
/ibm/objectfs/obj_CompressionObjectfs --scope fileset -P /var/mmfs/tmp/mmcesobj/compressionrule.96394 -N
objectcluster.scale.com -I yes -L 1'
Apr 19 11:21:28 objectcluster mmfs[96788]: CLI root root [EXIT, CHANGE] 'mmapplypolicy
/ibm/objectfs/obj_CompressionObjectfs --scope fileset -P /var/mmfs/tmp/mmcesobj/compressionrule.96394 -N
objectcluster.scale.com -I yes -L 1' RC=0
Apr 19 11:22:28 objectcluster mmfs[97803]: CLI root root [ENTRY, CHANGE] 'mmapplypolicy
/ibm/objectfs/obj_CompressionObjectfs --scope fileset -P /var/mmfs/tmp/mmcesobj/compressionrule.97625 -N
objectcluster.scale.com -I yes -L 1'
Apr 19 11:22:29 objectcluster mmfs[98019]: CLI root root [EXIT, CHANGE] 'mmapplypolicy
/ibm/objectfs/obj_CompressionObjectfs --scope fileset -P /var/mmfs/tmp/mmcesobj/compressionrule.97625 -N
objectcluster.scale.com -I yes -L 1' RC=0
.
.
.
```

Below is the output of the testobj.sh script (the modified testobj_compress.sh is to allow a pause to ensure the test files were in fact compressed). It was run with a multipart chunk size of 50MB.

```
[root@scalecluster ~]# ./testobj_compress.sh -c 50
testobj.sh run on Sat Mar 26 01:48:17 SAST 2022

Checking to see if s3cmd is installed... PASS

Checking to see if at least 2500MB is available in /root... PASS

Source directory /root already exists... PASS

500+0 records in
500+0 records out
524288000 bytes (524 MB, 500 MiB) copied, 0.210699 s, 2.5 GB/s

Creating source file file1... PASS

500+0 records in
500+0 records out
524288000 bytes (524 MB, 500 MiB) copied, 0.38634 s, 1.4 GB/s

Creating source file file2... PASS
```


Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
500+0 records in
500+0 records out
524288000 bytes (524 MB, 500 MiB) copied, 0.211079 s, 2.5 GB/s

Creating source file file3...          PASS

500+0 records in
500+0 records out
524288000 bytes (524 MB, 500 MiB) copied, 0.205398 s, 2.6 GB/s

Creating source file file4...          PASS

500+0 records in
500+0 records out
524288000 bytes (524 MB, 500 MiB) copied, 0.202562 s, 2.6 GB/s

Creating source file file5...          PASS

s3cmd is able to connect to the object endpoint...    PASS

Created testobj-b175f758 successfully...    PASS

Created testobj-5b548ef1 successfully...    PASS

Starting upload of source files using a chunk siize of 50MB for multipart uploads...    PASS

upload: '/root/testobjfiles/file1' -> 's3://testobj-b175f758/file1' [part 1 of 10, 50MB] [1 of 5]
52428800 of 52428800 100% in 1s 45.98 MB/s done
upload: '/root/testobjfiles/file1' -> 's3://testobj-b175f758/file1' [part 2 of 10, 50MB] [1 of 5]
52428800 of 52428800 100% in 1s 47.24 MB/s done
upload: '/root/testobjfiles/file1' -> 's3://testobj-b175f758/file1' [part 3 of 10, 50MB] [1 of 5]
52428800 of 52428800 100% in 1s 45.06 MB/s done
upload: '/root/testobjfiles/file1' -> 's3://testobj-b175f758/file1' [part 4 of 10, 50MB] [1 of 5]
52428800 of 52428800 100% in 1s 39.28 MB/s done
upload: '/root/testobjfiles/file1' -> 's3://testobj-b175f758/file1' [part 5 of 10, 50MB] [1 of 5]
52428800 of 52428800 100% in 1s 29.76 MB/s done
upload: '/root/testobjfiles/file1' -> 's3://testobj-b175f758/file1' [part 6 of 10, 50MB] [1 of 5]
52428800 of 52428800 100% in 1s 27.61 MB/s done
upload: '/root/testobjfiles/file1' -> 's3://testobj-b175f758/file1' [part 7 of 10, 50MB] [1 of 5]
52428800 of 52428800 100% in 1s 37.33 MB/s done
upload: '/root/testobjfiles/file1' -> 's3://testobj-b175f758/file1' [part 8 of 10, 50MB] [1 of 5]
52428800 of 52428800 100% in 1s 25.45 MB/s done
upload: '/root/testobjfiles/file1' -> 's3://testobj-b175f758/file1' [part 9 of 10, 50MB] [1 of 5]
52428800 of 52428800 100% in 1s 35.72 MB/s done
upload: '/root/testobjfiles/file1' -> 's3://testobj-b175f758/file1' [part 10 of 10, 50MB] [1 of 5]
52428800 of 52428800 100% in 1s 38.08 MB/s done
upload: '/root/testobjfiles/file2' -> 's3://testobj-b175f758/file2' [part 1 of 10, 50MB] [2 of 5]
52428800 of 52428800 100% in 1s 36.37 MB/s done
upload: '/root/testobjfiles/file2' -> 's3://testobj-b175f758/file2' [part 2 of 10, 50MB] [2 of 5]
52428800 of 52428800 100% in 1s 45.96 MB/s done
upload: '/root/testobjfiles/file2' -> 's3://testobj-b175f758/file2' [part 3 of 10, 50MB] [2 of 5]
52428800 of 52428800 100% in 1s 43.06 MB/s done
upload: '/root/testobjfiles/file2' -> 's3://testobj-b175f758/file2' [part 4 of 10, 50MB] [2 of 5]
52428800 of 52428800 100% in 1s 45.26 MB/s done
upload: '/root/testobjfiles/file2' -> 's3://testobj-b175f758/file2' [part 5 of 10, 50MB] [2 of 5]
52428800 of 52428800 100% in 1s 41.66 MB/s done
upload: '/root/testobjfiles/file2' -> 's3://testobj-b175f758/file2' [part 6 of 10, 50MB] [2 of 5]
52428800 of 52428800 100% in 1s 44.52 MB/s done
upload: '/root/testobjfiles/file2' -> 's3://testobj-b175f758/file2' [part 7 of 10, 50MB] [2 of 5]
52428800 of 52428800 100% in 1s 38.78 MB/s done
upload: '/root/testobjfiles/file2' -> 's3://testobj-b175f758/file2' [part 8 of 10, 50MB] [2 of 5]
52428800 of 52428800 100% in 1s 43.45 MB/s done
upload: '/root/testobjfiles/file2' -> 's3://testobj-b175f758/file2' [part 9 of 10, 50MB] [2 of 5]
52428800 of 52428800 100% in 1s 42.88 MB/s done
upload: '/root/testobjfiles/file2' -> 's3://testobj-b175f758/file2' [part 10 of 10, 50MB] [2 of 5]
52428800 of 52428800 100% in 1s 41.71 MB/s done
upload: '/root/testobjfiles/file3' -> 's3://testobj-b175f758/file3' [part 1 of 10, 50MB] [3 of 5]
52428800 of 52428800 100% in 1s 40.93 MB/s done
upload: '/root/testobjfiles/file3' -> 's3://testobj-b175f758/file3' [part 2 of 10, 50MB] [3 of 5]
52428800 of 52428800 100% in 1s 37.77 MB/s done
upload: '/root/testobjfiles/file3' -> 's3://testobj-b175f758/file3' [part 3 of 10, 50MB] [3 of 5]
52428800 of 52428800 100% in 1s 40.41 MB/s done
upload: '/root/testobjfiles/file3' -> 's3://testobj-b175f758/file3' [part 4 of 10, 50MB] [3 of 5]
52428800 of 52428800 100% in 1s 43.15 MB/s done
upload: '/root/testobjfiles/file3' -> 's3://testobj-b175f758/file3' [part 5 of 10, 50MB] [3 of 5]
52428800 of 52428800 100% in 1s 42.79 MB/s done
upload: '/root/testobjfiles/file3' -> 's3://testobj-b175f758/file3' [part 6 of 10, 50MB] [3 of 5]
52428800 of 52428800 100% in 1s 41.56 MB/s done
upload: '/root/testobjfiles/file3' -> 's3://testobj-b175f758/file3' [part 7 of 10, 50MB] [3 of 5]
52428800 of 52428800 100% in 1s 39.37 MB/s done
upload: '/root/testobjfiles/file3' -> 's3://testobj-b175f758/file3' [part 8 of 10, 50MB] [3 of 5]
52428800 of 52428800 100% in 1s 39.38 MB/s done
upload: '/root/testobjfiles/file3' -> 's3://testobj-b175f758/file3' [part 9 of 10, 50MB] [3 of 5]
52428800 of 52428800 100% in 1s 41.93 MB/s done
upload: '/root/testobjfiles/file3' -> 's3://testobj-b175f758/file3' [part 10 of 10, 50MB] [3 of 5]
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
52428800 of 52428800 100% in 1s 38.66 MB/s done
upload: '/root/testobjfiles/file4' -> 's3://testobj-b175f758/file4' [part 1 of 10, 50MB] [4 of 5]
52428800 of 52428800 100% in 1s 38.19 MB/s done
upload: '/root/testobjfiles/file4' -> 's3://testobj-b175f758/file4' [part 2 of 10, 50MB] [4 of 5]
52428800 of 52428800 100% in 1s 37.88 MB/s done
upload: '/root/testobjfiles/file4' -> 's3://testobj-b175f758/file4' [part 3 of 10, 50MB] [4 of 5]
52428800 of 52428800 100% in 1s 35.84 MB/s done
upload: '/root/testobjfiles/file4' -> 's3://testobj-b175f758/file4' [part 4 of 10, 50MB] [4 of 5]
52428800 of 52428800 100% in 1s 30.22 MB/s done
upload: '/root/testobjfiles/file4' -> 's3://testobj-b175f758/file4' [part 5 of 10, 50MB] [4 of 5]
52428800 of 52428800 100% in 1s 35.04 MB/s done
upload: '/root/testobjfiles/file4' -> 's3://testobj-b175f758/file4' [part 6 of 10, 50MB] [4 of 5]
52428800 of 52428800 100% in 1s 39.07 MB/s done
upload: '/root/testobjfiles/file4' -> 's3://testobj-b175f758/file4' [part 7 of 10, 50MB] [4 of 5]
52428800 of 52428800 100% in 1s 40.65 MB/s done
upload: '/root/testobjfiles/file4' -> 's3://testobj-b175f758/file4' [part 8 of 10, 50MB] [4 of 5]
52428800 of 52428800 100% in 1s 36.37 MB/s done
upload: '/root/testobjfiles/file4' -> 's3://testobj-b175f758/file4' [part 9 of 10, 50MB] [4 of 5]
52428800 of 52428800 100% in 1s 36.28 MB/s done
upload: '/root/testobjfiles/file4' -> 's3://testobj-b175f758/file4' [part 10 of 10, 50MB] [4 of 5]
52428800 of 52428800 100% in 1s 35.79 MB/s done
upload: '/root/testobjfiles/file5' -> 's3://testobj-b175f758/file5' [part 1 of 10, 50MB] [5 of 5]
52428800 of 52428800 100% in 1s 36.43 MB/s done
upload: '/root/testobjfiles/file5' -> 's3://testobj-b175f758/file5' [part 2 of 10, 50MB] [5 of 5]
52428800 of 52428800 100% in 1s 32.49 MB/s done
upload: '/root/testobjfiles/file5' -> 's3://testobj-b175f758/file5' [part 3 of 10, 50MB] [5 of 5]
52428800 of 52428800 100% in 1s 32.57 MB/s done
upload: '/root/testobjfiles/file5' -> 's3://testobj-b175f758/file5' [part 4 of 10, 50MB] [5 of 5]
52428800 of 52428800 100% in 1s 32.56 MB/s done
upload: '/root/testobjfiles/file5' -> 's3://testobj-b175f758/file5' [part 5 of 10, 50MB] [5 of 5]
52428800 of 52428800 100% in 1s 32.32 MB/s done
upload: '/root/testobjfiles/file5' -> 's3://testobj-b175f758/file5' [part 6 of 10, 50MB] [5 of 5]
52428800 of 52428800 100% in 1s 33.01 MB/s done
upload: '/root/testobjfiles/file5' -> 's3://testobj-b175f758/file5' [part 7 of 10, 50MB] [5 of 5]
52428800 of 52428800 100% in 1s 35.72 MB/s done
upload: '/root/testobjfiles/file5' -> 's3://testobj-b175f758/file5' [part 8 of 10, 50MB] [5 of 5]
52428800 of 52428800 100% in 1s 32.59 MB/s done
upload: '/root/testobjfiles/file5' -> 's3://testobj-b175f758/file5' [part 9 of 10, 50MB] [5 of 5]
52428800 of 52428800 100% in 1s 25.01 MB/s done
upload: '/root/testobjfiles/file5' -> 's3://testobj-b175f758/file5' [part 10 of 10, 50MB] [5 of 5]
52428800 of 52428800 100% in 1s 33.49 MB/s done

PUT 5 files of 500MB each... PASS

Runtime for PUT is 73 secondsi...
Throughput for PUT is 34 MB/s...
>>>> PLEASE CONFIRM FILES ARE COMPRESSED.. PRESS ANY KEY TO CONTINUE..

Starting copy of source files from container testobj-b175f758 to container testobj-5b548ef1... PASS

remote copy: 's3://testobj-b175f758/file1' -> 's3://testobj-5b548ef1/file1' [1 of 5]
remote copy: 's3://testobj-b175f758/file2' -> 's3://testobj-5b548ef1/file2' [2 of 5]
remote copy: 's3://testobj-b175f758/file3' -> 's3://testobj-5b548ef1/file3' [3 of 5]
remote copy: 's3://testobj-b175f758/file4' -> 's3://testobj-5b548ef1/file4' [4 of 5]
remote copy: 's3://testobj-b175f758/file5' -> 's3://testobj-5b548ef1/file5' [5 of 5]

Copy 5 files from container testobj-b175f758 to container testobj-5b548ef1... PASS

Runtime for CP is 36 secondsi...
Throughput for CP is 69 MB/s...

Starting GET source files from container testobj-b175f758 to /root/testobjfiles... PASS

download: 's3://testobj-b175f758/file1' -> '/root/testobjfiles/file1' [1 of 5]
524288000 of 524288000 100% in 10s 47.77 MB/s done
download: 's3://testobj-b175f758/file2' -> '/root/testobjfiles/file2' [2 of 5]
524288000 of 524288000 100% in 10s 47.29 MB/s done
download: 's3://testobj-b175f758/file3' -> '/root/testobjfiles/file3' [3 of 5]
524288000 of 524288000 100% in 10s 45.64 MB/s done
download: 's3://testobj-b175f758/file4' -> '/root/testobjfiles/file4' [4 of 5]
524288000 of 524288000 100% in 10s 49.53 MB/s done
download: 's3://testobj-b175f758/file5' -> '/root/testobjfiles/file5' [5 of 5]
524288000 of 524288000 100% in 10s 47.01 MB/s done
GET of 5 files from container testobj-b175f758 to /root/testobjfiles... PASS

Listing retrieved files from container

total 2572228
drwxr-xr-x. 2 root root 71 Mar 26 01:20 .
dr-xr-x---. 11 root root 4096 Mar 26 01:38 ..
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
-rw-r--r--. 1 root root 524288000 Mar 25 23:48 file1
-rw-r--r--. 1 root root 524288000 Mar 25 23:48 file2
-rw-r--r--. 1 root root 524288000 Mar 25 23:49 file3
-rw-r--r--. 1 root root 524288000 Mar 25 23:49 file4
-rw-r--r--. 1 root root 524288000 Mar 25 23:49 file5

Runtime for GET is 54 secondsi...
Throughput for GET is 46 MB/s...

Skipping.. rm /root/testobjfiles/file*

Cleaned up /root/testobjfiles          PASS
Skipping rm -fd /root/testobjfiles

Removed testobjfiles from our working directory          PASS
Skipping.. /usr/local/bin/s3cmd rb --recursive --force s3://testobj-b175f758 > /dev/null 2>&1

Deleted testobj-b175f758 and all its contents successfully...          PASS
Skipping.. /usr/local/bin/s3cmd rb --recursive --force s3://testobj-5b548ef1 > /dev/null 2>&1

Deleted testobj-5b548ef1 and all its contents successfully...          PASS

Created output file /root/testobj-2022-03-26-02-20-02.log...          PASS
Writing results to /root/testobj-2022-03-26-02-20-02.log...          PASS

testobj.sh completed on Sat Mar 26 02:20:02 SAST 2022

[root@scalecluster ~]#
```

If we compare the test results for the two runs (one with compression and one without), we get:

COMPRESSED:

```
[root@scalecluster ~]# cat testobj-2022-03-26-02-20-02.log

*****
testobj.sh run on Sat Mar 26 02:20:02 SAST 2022

PUT runtime was 73s and throughput was 34MB/s
CP runtime was 36s and throughput was 69MB/s
GET runtime was 54s and throughput was 46MB/s

*****
```

UNCOMPRESSED:

```
[root@scalecluster ~]# cat testobj-2022-03-24-16-12-48.log

*****
testobj.sh run on Thu Mar 24 16:12:48 SAST 2022

PUT runtime was 75s and throughput was 33MB/s
CP runtime was 44s and throughput was 56MB/s
GET runtime was 62s and throughput was 40MB/s

*****
```

Granted the test files created by the dd command using /dev/zero as input result in highly compressible files. Performance with highly compressible data should be better than reading or writing uncompressed data. As an example, to check our compression ratio, during the testobj.sh script run, if we query one of the test objects (file1) and use the swift-get-nodes utility to find its location we will see the target bucket has only the manifest file which is 2511 bytes.

```
[root@objectcluster obj_CompressionObjectfs]# swift list
testobj-5b548ef1
testobj-b175f758
testobj-b175f758+segments
[root@objectcluster obj_CompressionObjectfs]#

[root@objectcluster obj_CompressionObjectfs]# swift list testobj-b175f758
file1
file2
file3
file4
file5
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
[root@objectcluster obj_CompressionObjectfs]# swift list testobj-b175f758+segments
file1/ZjE4MGZjYTQtnZvJMC00NTMzLWEzMjQtnThmNTVjNDU2NTVh/1
file1/ZjE4MGZjYTQtnZvJMC00NTMzLWEzMjQtnThmNTVjNDU2NTVh/10
file1/ZjE4MGZjYTQtnZvJMC00NTMzLWEzMjQtnThmNTVjNDU2NTVh/2
file1/ZjE4MGZjYTQtnZvJMC00NTMzLWEzMjQtnThmNTVjNDU2NTVh/3
file1/ZjE4MGZjYTQtnZvJMC00NTMzLWEzMjQtnThmNTVjNDU2NTVh/4
file1/ZjE4MGZjYTQtnZvJMC00NTMzLWEzMjQtnThmNTVjNDU2NTVh/5
file1/ZjE4MGZjYTQtnZvJMC00NTMzLWEzMjQtnThmNTVjNDU2NTVh/6
file1/ZjE4MGZjYTQtnZvJMC00NTMzLWEzMjQtnThmNTVjNDU2NTVh/7
file1/ZjE4MGZjYTQtnZvJMC00NTMzLWEzMjQtnThmNTVjNDU2NTVh/8
file1/ZjE4MGZjYTQtnZvJMC00NTMzLWEzMjQtnThmNTVjNDU2NTVh/9
.
.
.

[root@objectcluster obj_CompressionObjectfs]# swift-get-nodes /etc/swift/object-18722203250.ring.gz
AUTH_224f53da12e04b7893783821a34c171f testobj-b175f758 file1

Account AUTH_224f53da12e04b7893783821a34c171f
Container testobj-b175f758
Object file1
.
.
.
Use your own device location of servers:
such as "export DEVICE=/srv/node"
ssh 192.168.226.252 "ls -lah $(DEVICE:~/srv/node*)/s18722203250z1device87/objects-
18722203250/15996/706/f9f198d4d1b00e921dce7c853da80706/1648252119.06961.data
-rwxr-xr-x. 1 swift swift 2511 Mar 26 01:48 /ibm/objectfs/obj_CompressionObjectfs/s18722203250z1device87/objects-
18722203250/15996/706/f9f198d4d1b00e921dce7c853da80706/1648252119.06961.data" # [Handoff]

note: `~/srv/node` is used as default value of `devices`, the real value is set in the config file on each storage
node.
[root@objectcluster obj_CompressionObjectfs]# ls -l
/ibm/objectfs/obj_CompressionObjectfs/s18722203250z1device87/objects-
18722203250/15996/706/f9f198d4d1b00e921dce7c853da80706/1648252119.06961.data
-rwxr-xr-x. 1 swift swift 2511 Mar 26 01:48 /ibm/objectfs/obj_CompressionObjectfs/s18722203250z1device87/objects-
18722203250/15996/706/f9f198d4d1b00e921dce7c853da80706/1648252119.06961.data
[root@objectcluster obj_CompressionObjectfs]#
```

If we query one of the multipath chunk files as reported by manifest file, we expect it to be compressed. We can just cat the manifest file for our object to find its associated chunks and their names.

```
[root@objectcluster obj_CompressionObjectfs]# cat
/ibm/objectfs/obj_CompressionObjectfs/s18722203250z1device87/objects-
18722203250/15996/706/f9f198d4d1b00e921dce7c853da80706/1648252119.06961.data
[{"name": "/testobj-b175f758+segments/file1/ZjE4MGZjYTQtnZvJMC00NTMzLWEzMjQtnThmNTVjNDU2NTVh/1", "bytes": 52428800,
"hash": "25e317773f308e446cc84c503a6d1f85", "content_type": "application/octet-stream", "last_modified": "2022-03-
25T23:48:26.000000"}, {"name": "/testobj-
b175f758+segments/file1/ZjE4MGZjYTQtnZvJMC00NTMzLWEzMjQtnThmNTVjNDU2NTVh/2", "bytes": 52428800, "hash":
"25e317773f308e446cc84c503a6d1f85", "content_type": "application/octet-stream", "last_modified": "2022-03-
25T23:48:27.000000"}, {"name": "/testobj-
b175f758+segments/file1/ZjE4MGZjYTQtnZvJMC00NTMzLWEzMjQtnThmNTVjNDU2NTVh/3", "bytes": 52428800, "hash":
"25e317773f308e446cc84c503a6d1f85", "content_type": "application/octet-stream", "last_modified": "2022-03-
25T23:48:28.000000"}, {"name": "/testobj-
b175f758+segments/file1/ZjE4MGZjYTQtnZvJMC00NTMzLWEzMjQtnThmNTVjNDU2NTVh/4", "bytes": 52428800, "hash":
"25e317773f308e446cc84c503a6d1f85", "content_type": "application/octet-stream", "last_modified": "2022-03-
25T23:48:29.000000"}, {"name": "/testobj-
b175f758+segments/file1/ZjE4MGZjYTQtnZvJMC00NTMzLWEzMjQtnThmNTVjNDU2NTVh/5", "bytes": 52428800, "hash":
"25e317773f308e446cc84c503a6d1f85", "content_type": "application/octet-stream", "last_modified": "2022-03-
25T23:48:30.000000"}, {"name": "/testobj-
b175f758+segments/file1/ZjE4MGZjYTQtnZvJMC00NTMzLWEzMjQtnThmNTVjNDU2NTVh/6", "bytes": 52428800, "hash":
"25e317773f308e446cc84c503a6d1f85", "content_type": "application/octet-stream", "last_modified": "2022-03-
25T23:48:32.000000"}, {"name": "/testobj-
b175f758+segments/file1/ZjE4MGZjYTQtnZvJMC00NTMzLWEzMjQtnThmNTVjNDU2NTVh/7", "bytes": 52428800, "hash":
"25e317773f308e446cc84c503a6d1f85", "content_type": "application/octet-stream", "last_modified": "2022-03-
25T23:48:34.000000"}, {"name": "/testobj-
b175f758+segments/file1/ZjE4MGZjYTQtnZvJMC00NTMzLWEzMjQtnThmNTVjNDU2NTVh/8", "bytes": 52428800, "hash":
"25e317773f308e446cc84c503a6d1f85", "content_type": "application/octet-stream", "last_modified": "2022-03-
25T23:48:37.000000"}, {"name": "/testobj-
b175f758+segments/file1/ZjE4MGZjYTQtnZvJMC00NTMzLWEzMjQtnThmNTVjNDU2NTVh/9", "bytes": 52428800, "hash":
"25e317773f308e446cc84c503a6d1f85", "content_type": "application/octet-stream", "last_modified": "2022-03-
25T23:48:37.000000"}, {"name": "/testobj-
b175f758+segments/file1/ZjE4MGZjYTQtnZvJMC00NTMzLWEzMjQtnThmNTVjNDU2NTVh/10", "bytes": 52428800, "hash":
"25e317773f308e446cc84c503a6d1f85", "content_type": "application/octet-stream", "last_modified": "2022-03-
25T23:48:38.000000"}] [root@objectcluster obj_CompressionObjectfs]#
```

Once we have the name of a chunk, we can use the swift-get-nodes command again to find its location so we can check if it is in fact compressed. Let us take the first chunk for file1.

```
[root@objectcluster obj_CompressionObjectfs]# swift-get-nodes /etc/swift/object-18722203250.ring.gz
AUTH_224f53da12e04b7893783821a34c171f testobj-b175f758+segments
file1/ZjE4MGZjYTQtnZvJMC00NTMzLWEzMjQtnThmNTVjNDU2NTVh/1
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
Account AUTH_224f53da12e04b7893783821a34c171f
Container testobj-b175f758+segments
Object file1/ZjE4MGZjYTQtNzVjMC00NTMzLWEzMjQtNTVjNDU2NTVh/1
.
.
.
Use your own device location of servers:
such as "export DEVICE=/srv/node"
ssh 192.168.226.252 "ls -lah ${DEVICE}:/srv/node*/s18722203250z1device120/objects-
18722203250/57/785/00e51b072cf5d2f0068bf099a4490785/1648252105.16591.data"
ssh 192.168.226.252 "ls -lah ${DEVICE}:/srv/node*/s18722203250z1device83/objects-
18722203250/57/785/00e51b072cf5d2f0068bf099a4490785/" # [Handoff]

note: '/srv/node*' is used as default value of 'devices', the real value is set in the config file on each storage
node.
```

The size of the chunk is 50MB as expected.

```
[root@objectcluster obj_CompressionObjectfs]# ls -alh s18722203250z1device120/objects-
18722203250/57/785/00e51b072cf5d2f0068bf099a4490785/1648252105.16591.data
-rwxr-xr-x. 1 swift swift 50M Mar 26 01:48 s18722203250z1device120/objects-
18722203250/57/785/00e51b072cf5d2f0068bf099a4490785/1648252105.16591.data
```

If we query the file attributes we expect to find ARCHIVE COMPRESSION in the “Misc attributes” field to indicate the file is indeed compressed.

```
[root@objectcluster obj_CompressionObjectfs]# mmlsattr -L s18722203250z1device120/objects-
18722203250/57/785/00e51b072cf5d2f0068bf099a4490785/1648252105.16591.data
file name: s18722203250z1device120/objects-
18722203250/57/785/00e51b072cf5d2f0068bf099a4490785/1648252105.16591.data
metadata replication: 1 max 2
data replication: 1 max 2
immutable: no
appendOnly: no
flags:
storage pool name: system
fileset name: obj_CompressionObjectfs
snapshot name:
creation time: Sat Mar 26 01:48:25 2022
Misc attributes: ARCHIVE COMPRESSION (library z)
Encrypted: no
[root@objectcluster obj_CompressionObjectfs]#
```

The actual size of the 50M chunk is 8192 kilobytes or 8MB. So we got about an 84% compression ratio hence why the testobj.sh output is slightly better for the COMPRESS run.

```
[root@objectcluster obj_CompressionObjectfs]# du -k s18722203250z1device120/objects-
18722203250/57/785/00e51b072cf5d2f0068bf099a4490785/1648252105.16591.data
8192 s18722203250z1device120/objects-18722203250/57/785/00e51b072cf5d2f0068bf099a4490785/1648252105.16591.data
[root@objectcluster obj_CompressionObjectfs]#
```

We have now covered the use of compression which is a feature of Spectrum Scale that is extended to our OpenStack Swift Object configuration which doesn't have native compression functionality. This is an example of one of the benefits of using Spectrum Scale over other Object solutions.

Object Information Lifecycle Management (ILM)

One of the major benefits offered by public cloud providers like Amazon S3 is different tiers of storage which vary in cost, performance and reliability. Amazon S3 offers multiple different storage classes (<https://aws.amazon.com/s3/storage-classes/>) which include classes that do intelligent tiering (moving of objects between different tiers of storage based on access patterns). We will cover the equivalent functionality in Spectrum Scale in the next section. For now, we will concentrate on how to setup different tiers of storage for your Spectrum Scale Swift implementation and how to ensure objects are placed in the correct tier. You might for example want to have an SSD tier for high-performance object workloads and a disk or tape tier for other object workloads. Having discussed how Swift Storage Policy works and how the Swift architecture is implemented in Spectrum Scale, this section will discuss how to incorporate different tiers of storage to match different Storage Policies on Spectrum Scale.

Spectrum Scale Object is implemented on a single Spectrum Scale filesystem. As discussed above, the initial fileset created when the Object protocol is enabled houses the account and container ring and the object ring that corresponds to Policy-0. Additional Swift Storage Policies are implemented as separate independent filesets within this filesystem. We therefore need a way to include additional storage tiers into our existing object filesystem and then find a mechanism of moving data based on some criteria to the correct storage tier. We might also want to find a way to migrate only data belonging to a specific Swift Storage Policy to a different tier (so only objects that belong to a specific object ring on a specific fileset). Luckily for us, Spectrum Scale filesystems are made up of internal and external pools and it has a powerful policy engine built in that will help us achieve our objective.

A graphical representation of what we want to achieve in our example is depicted below.

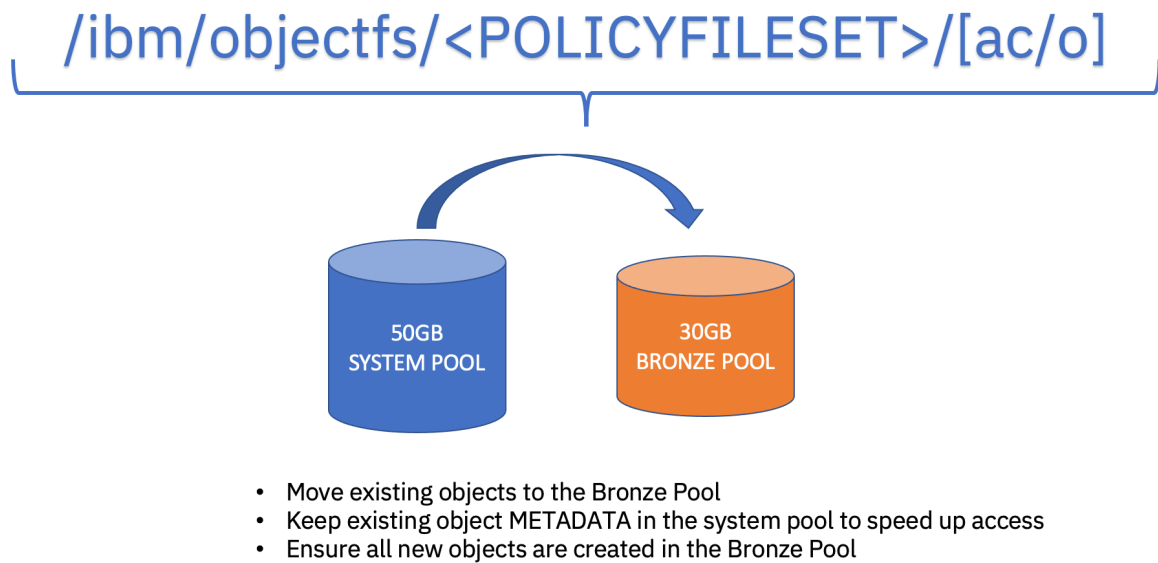


Figure 18: Spectrum Scale Object Directory layout

We currently have the following configuration:

```
[root@objectcluster objectfs]# mmobj policy list
```

| Index | Name | Default | Deprecated | Fileset | Functions | File System |
|-------------|---------------------|---------|------------|-------------------------|-------------|-------------|
| 0 | policy-0 | yes | | object_fileset | | objectfs |
| 18722203250 | CompressionObjectfs | | | obj_CompressionObjectfs | compression | objectfs |

```
[root@objectcluster objectfs]#
```

Where:

- /ibm/objectfs – this is our base Spectrum Scale filesystem for our Swift implementation
- /ibm/objectfs/object_fileset – is the initial fileset created when the object protocol is enabled that belongs to Storage Policy “Policy-0”
- /ibm/object/obj_CompressionObjectfs – is a fileset created to cater for our Compression Storage Policy “Compressionfs”
- /ibm/object/object_fileset/ac – is a directory that represents the combined account and container rings
- /ibm/object/object_fileset/o – is a directory that represents the object ring for our Storage Policy “Policy-0”

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

- `/ibm/object/obj_CompressionObjectfs/o` – is a directory that represents the object ring for our Storage Policy “CompressionObjectfs”

Our object filesystem currently has only a single NSD which belongs to the default system pool.

```
[root@objectcluster ansible-toolkit]# mmdf objectfs
disk      disk size  failure holds   holds      free in KB      free in KB
name      in KB      group metadata data      in full blocks  in fragments
-----
Disks in storage pool: system (Maximum disk size allowed is 457.49 GB)
nsd2      52428800      1 yes      yes      47210496 ( 90%)      263760 ( 1%)
-----
(pool total)      52428800      47210496 ( 90%)      263760 ( 1%)
=====
(total)      52428800      47210496 ( 90%)      263760 ( 1%)

Inode Information
-----
Total number of used inodes in all Inode spaces:      16612
Total number of free inodes in all Inode spaces:      185116
Total number of allocated inodes in all Inode spaces:  201728
Total of Maximum number of inodes in all Inode spaces: 4369408
[root@objectcluster ansible-toolkit]#

[root@objectcluster ~]# mmlspool objectfs
Storage pools in file system at '/ibm/objectfs':
Name      Id      BlkSize Data Meta Total Data in (KB)  Free Data in (KB)  Total Meta in (KB)  Free
Meta in (KB)
system      0      4 MB  yes  yes  52428800  50085888 ( 96%)  52428800  50343936 (
96%)
[root@objectcluster ~]#
```

Before we cover object migration, all existing containers and objects were deleted and two new containers belonging to our *policy-0* and *Compressionfs* Storage Policies were created. New objects were then uploaded to them. This was done so that its easier to demonstrate the migration and placement policies covered in this section.

Clean out the current containers and objects.

```
[root@objectcluster ~]# swift stat
Account: AUTH_224f53da12e04b7893783821a34c171f
Containers: 0
Objects: 0
Bytes: 0
Containers in policy "compressionobjectfs": 0
Objects in policy "compressionobjectfs": 0
Bytes in policy "compressionobjectfs": 0
Containers in policy "policy-0": 0
Objects in policy "policy-0": 0
Bytes in policy "policy-0": 0
Content-Type: application/json; charset=utf-8
X-Timestamp: 1647870922.58997
Accept-Ranges: bytes
X-Trans-Id: tx33986621a30149da8105f-00623ffca4
X-OpenStack-Request-Id: tx33986621a30149da8105f-00623ffca4
[root@objectcluster ~]#
```

Create two new containers, one belonging to our default policy (*policy-0*) and the other to our *Compressionfs* policy.

```
[root@objectcluster ~]# swift post -H "X-Storage-Policy:CompressionObjectfs" compressbucket
[root@objectcluster ~]# swift post mybucket
[root@objectcluster ~]# swift list
Compressbucket
mybucket
```

Now upload the same test files to each bucket using `s3cmd`.

```
[root@scalecluster ~]# s3cmd put --recursive --multipart-chunk-size-mb=500 testfiles/ s3://mybucket
upload: 'testfiles/file13' -> 's3://mybucket/file13' [1 of 5]
524288000 of 524288000 100% in 21s 23.11 MB/s done
upload: 'testfiles/file14' -> 's3://mybucket/file14' [2 of 5]
524288000 of 524288000 100% in 19s 26.12 MB/s done
upload: 'testfiles/file15' -> 's3://mybucket/file15' [3 of 5]
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
524288000 of 524288000 100% in 15s 33.29 MB/s done
upload: 'testfiles/file16' -> 's3://mybucket/file16' [4 of 5]
524288000 of 524288000 100% in 13s 38.08 MB/s done
upload: 'testfiles/file17' -> 's3://mybucket/file17' [5 of 5]
524288000 of 524288000 100% in 15s 33.15 MB/s done
[root@scalecluster ~]# s3cmd put --recursive --multipart-chunk-size-mb=500 testfiles/ s3://compressbucket
upload: 'testfiles/file13' -> 's3://compressbucket/file13' [1 of 5]
524288000 of 524288000 100% in 12s 41.11 MB/s done
upload: 'testfiles/file14' -> 's3://compressbucket/file14' [2 of 5]
524288000 of 524288000 100% in 13s 36.61 MB/s done
upload: 'testfiles/file15' -> 's3://compressbucket/file15' [3 of 5]
524288000 of 524288000 100% in 12s 39.60 MB/s done
upload: 'testfiles/file16' -> 's3://compressbucket/file16' [4 of 5]
524288000 of 524288000 100% in 12s 38.52 MB/s done
upload: 'testfiles/file17' -> 's3://compressbucket/file17' [5 of 5]
524288000 of 524288000 100% in 13s 36.09 MB/s done
[root@scalecluster ~]#
```

Check that we have 1 container and 5 files in each Storage Policy and that all our data is stored in the default system pool.

```
[root@objectcluster ~]# swift stat
Account: AUTH_224f53dal2e04b7893783821a34c171f
Containers: 2
Objects: 5
Bytes: 2621440000
Containers in policy "policy-0": 1
Objects in policy "policy-0": 5
Bytes in policy "policy-0": 2621440000
Containers in policy "compressionobjectfs": 1
Objects in policy "compressionobjectfs": 5
Bytes in policy "compressionobjectfs": 2621440000
Content-Type: application/json; charset=utf-8
X-Timestamp: 1647870922.58997
Accept-Ranges: bytes
X-Trans-Id: tx25d8395870a24d9f81565-00623fff7c
X-OpenStack-Request-Id: tx25d8395870a24d9f81565-00623fff7c
[root@objectcluster ~]# mmlspool objectfs
Storage pools in file system at '/ibm/objectfs':
Name      Id  BlkSize Data Meta Total Data in (KB)  Free Data in (KB)  Total Meta in (KB)  Free
Meta in (KB)
system    0   4 MB  yes  yes  52428800  47259648 ( 90%)  52428800  47517696 (
91%)
[root@objectcluster ~]#
```

Note, OpenStack Swift CLI command stat does not report the compressed size as it is unaware of the underlying Spectrum Scale compression that is being used.

```
[root@objectcluster ~]# du -sh /ibm/objectfs/object_fileset
2.5G /ibm/objectfs/object_fileset
[root@objectcluster ~]# du -sh /ibm/objectfs/obj_CompressionObjectfs
261M /ibm/objectfs/obj_CompressionObjectfs
[root@objectcluster ~]#
```

Our objective is to move all objects from the system pool to a lower cost pool. The first thing we need to do is add a new pool which represents a lower cost and performance storage tier. For our purposes, we will assume the system pool is equivalent to a high-performance storage tier (e.g. like SSD) in which we want to cache our account and container rings (remember, every Object API request needs to query the Swift account, container and object rings to locate the object so keeping the account and container rings on our high-performance tier is important to ensure optimal client performance). To achieve this, we allocate a new virtual disk /dev/nvme04 to our existing object filesystem but in a separate pool called bronze. We can use the Spectrum Scale Installation Toolkit that we used to deploy the cluster to make any modifications (like adding additional nodes, or new filesystems or additional NSDs etc.). Let us add the new NSD to the bronze pool for our Object filesystem using the Installation Toolkit. Note, only the system pool can hold metadata so any NSD added to a non-system pool needs to be tagged as “dataOnly”.

```
[root@objectcluster ansible-toolkit]# ./spectrumsscale nsd add -p objectcluster.scale.com -fs objectfs -po bronze -u
dataOnly /dev/nvme0n4
[ INFO ] Connecting to objectcluster.scale.com to check devices and expand wildcards.
[ INFO ] Looking up details of /dev/nvme0n4.
[ INFO ] Adding NSD nsd3 on objectcluster.scale.com using device /dev/nvme0n4.
[ INFO ] Configuration updated
```


Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
[ INFO ] Tip : If all node designations and any required protocol configurations are complete, proceed to check
the installation configuration: ./spectrumscale install --precheck
[root@objectcluster ansible-toolkit]# ./spectrumscale nsd list
[ INFO ] Name FS Size(GB) Usage FG Pool Device Servers
[ INFO ] nsd1 cesSharedRoot 5.0 dataAndMetadata 1 system /dev/nvme0n2 objectcluster.scale.com
[ INFO ] nsd2 objectfs 50.0 dataAndMetadata 1 system /dev/nvme0n3 objectcluster.scale.com
[ INFO ] nsd3 objectfs 30.0 dataOnly 1 bronze /dev/nvme0n4 objectcluster.scale.com
[root@objectcluster ansible-toolkit]#
```

As we did previously, we will run the install pre-check.

```
[root@objectcluster ansible-toolkit]# ./spectrumscale install -pr
[ INFO ] Logging to file: /usr/lpp/mmfs/5.1.3.0/ansible-toolkit/logs/INSTALL-PRECHECK-27-03-2022_08:16:13.log
[ INFO ] Validating configuration
[ WARN ] Ensure that base OS repositories are configured and enabled so that package dependencies can be satisfied
during installation.
[ WARN ] objectcluster.scale.com is specified as both an NSD server and a protocol node. It is recommended that a
node not be both an NSD server and a protocol node
[ WARN ] Only one GUI server specified. The Graphical User Interface will not be highly available.
[ INFO ] Install toolkit will not configure file audit logging as it has been disabled.
[ INFO ] Performing GPFS checks.
[ INFO ] Running environment checks
.
.
.
[ INFO ] ok: [objectcluster.scale.com]
[ INFO ] TASK [core/common : rpm key]
*****
[ INFO ] ok: [objectcluster.scale.com]
[ INFO ] PLAY RECAP
*****
[ INFO ] localhost : ok=4 changed=1 unreachable=0 failed=0 skipped=0 rescued=0
ignored=0
[ INFO ] objectcluster.scale.com : ok=14 changed=0 unreachable=0 failed=0 skipped=48 rescued=0
ignored=0
[ INFO ] Pre-check successful for install.
[ INFO ] Tip : ./spectrumscale install
[root@objectcluster ansible-toolkit]#
```

If this completes successfully we can continue with the actual install.

```
[root@objectcluster ansible-toolkit]# ./spectrumscale install
[ INFO ] Logging to file: /usr/lpp/mmfs/5.1.3.0/ansible-toolkit/logs/INSTALL-27-03-2022_08:19:06.log
[ INFO ] Validating configuration
[ WARN ] Ensure that base OS repositories are configured and enabled so that package dependencies can be satisfied
during installation.
[ WARN ] objectcluster.scale.com is specified as both an NSD server and a protocol node. It is recommended that a
node not be both an NSD server and a protocol node
[ WARN ] Only one GUI server specified. The Graphical User Interface will not be highly available.
[ INFO ] Install toolkit will not configure file audit logging as it has been disabled.
[ INFO ] Running pre-install checks
[ INFO ] Running environment checks
[ INFO ] Installed license validation passed. The installed license on GPFS cluster matching with the installer
license version.
[ WARN ] With 1 node specified functionality will be limited
.
.
.
[ INFO ] changed: [objectcluster.scale.com]
[ INFO ] PLAY RECAP
*****
[ INFO ] localhost : ok=4 changed=1 unreachable=0 failed=0 skipped=0 rescued=0
ignored=0
[ INFO ] objectcluster.scale.com : ok=155 changed=25 unreachable=0 failed=0 skipped=325 rescued=0
ignored=0
[ INFO ] Destroying repository...
[ INFO ] Checking for a successful install
[ INFO ] Checking state of GPFS
[ INFO ] Checking state of GPFS on all nodes
[ INFO ] GPFS active on all nodes
[ INFO ] GPFS ACTIVE
[ INFO ] Checking state of NSDs
[ INFO ] NSDs ACTIVE
[ INFO ] Checking state of Performance Monitoring
[ INFO ] Running Performance Monitoring post-install checks
[ INFO ] pmcollector running on all nodes
[ INFO ] pmsensors running on all nodes
[ INFO ] Performance Monitoring ACTIVE
[ INFO ] Checking state of GUI
[ INFO ] Running Graphical User Interface post-install checks
[ INFO ] Graphical User Interface running on all GUI servers
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
[ INFO ] Enter one of the following addresses into a web browser to access the Graphical User Interface:
objectcluster.scale.com
[ INFO ] GUI ACTIVE
[ INFO ] SUCCESS
[ INFO ] All services running
[ INFO ] StanzaFile and NodeDesc file for NSD, filesystem, and cluster setup have been saved to /usr/lpp/mmfs
folder on node: objectcluster.scale.com
[ INFO ] Installation successful. 1 GPFS node active in cluster objectcluster.scale.com. Completed in 6 minutes 17
seconds.
[ INFO ] Tip :If all node designations and any required protocol configurations are complete, proceed to check the
deploy configuration:./spectrumscale deploy --precheck
[root@objectcluster ansible-toolkit]#
```

Before you continue, you might want to ensure the post-install check also completes successfully.

```
[root@objectcluster ansible-toolkit]# ./spectrumscale install -po
[ INFO ] Logging to file: /usr/lpp/mmfs/5.1.3.0/ansible-toolkit/logs/INSTALL-POSTCHECK-27-03-2022_08:25:40.log
[ WARN ] objectcluster.scale.com is specified as both an NSD server and a protocol node. It is recommended that a
node not be both an NSD server and a protocol node
[ WARN ] Only one GUI server specified. The Graphical User Interface will not be highly available.
[ INFO ] Install toolkit will not configure file audit logging as it has been disabled.
[ INFO ] Checking state of GPFS
[ INFO ] Checking state of GPFS on all nodes
[ INFO ] GPFS active on all nodes
[ INFO ] GPFS ACTIVE
[ INFO ] Checking state of NSDs
[ INFO ] NSDs ACTIVE
[ INFO ] Checking state of Performance Monitoring
[ INFO ] Running Performance Monitoring post-install checks
[ INFO ] pmcollector running on all nodes
[ INFO ] pmsensors running on all nodes
[ INFO ] Performance Monitoring ACTIVE
[ INFO ] Checking state of GUI
[ INFO ] Running Graphical User Interface post-install checks
[ INFO ] Graphical User Interface running on all GUI servers
[ INFO ] Enter one of the following addresses into a web browser to access the Graphical User Interface:
objectcluster.scale.com
[ INFO ] GUI ACTIVE
[ INFO ] SUCCESS
[ INFO ] All services running
[ INFO ] StanzaFile and NodeDesc file for NSD, filesystem, and cluster setup have been saved to /usr/lpp/mmfs
folder on node: objectcluster.scale.com
[root@objectcluster ansible-toolkit]#
```

We can now verify the new NSD is added successfully to our object filesystem as a separate pool.

```
[root@objectcluster ansible-toolkit]# mmdf objectfs
disk          disk size  failure holds   holds          free in KB          free in KB
name          in KB      group metadata data          in full blocks          in fragments
-----
Disks in storage pool: system (Maximum disk size allowed is 457.49 GB)
nsd2          52428800          1 yes         yes          47243264 ( 90%)          260120 ( 0%)
-----
(pool total)          52428800          47243264 ( 90%)          260120 ( 0%)

Disks in storage pool: bronze (Maximum disk size allowed is 274.49 GB)
nsd3          31457280          1 no          yes          31330304 (100%)          15768 ( 0%)
-----
(pool total)          31457280          31330304 (100%)          15768 ( 0%)

=====
(data)          83886080          78573568 ( 94%)          275888 ( 0%)
(metadata)          52428800          47243264 ( 90%)          260120 ( 0%)
=====
(total)          83886080          78573568 ( 94%)          275888 ( 0%)

Inode Information
-----
Total number of used inodes in all Inode spaces:          16612
Total number of free inodes in all Inode spaces:          185116
Total number of allocated inodes in all Inode spaces:          201728
Total of Maximum number of inodes in all Inode spaces:          4369408
[root@objectcluster ansible-toolkit]#

[root@objectcluster ansible-toolkit]# mmlspool objectfs
Storage pools in file system at '/ibm/objectfs':
Name          Id      BlkSize Data Meta Total Data in (KB)  Free Data in (KB)  Total Meta in (KB)  Free
Meta in (KB)
system          0          4 MB  yes  yes  52428800  47243264 ( 90%)  52428800  47501312 (
91%)
bronze         131073          4 MB  yes  no   31457280  31383552 (100%)          0          0 (
0%)
[root@objectcluster ansible-toolkit]#
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

Swift account and container information is stored as .db files in their respective rings and Objects are stored as .data files also in its respective ring. Based on our example, now that we have a lower cost storage tier, let us use the Spectrum Scale policy engine to move the required data for us (which in our case are the objects themselves). The Spectrum Scale policy engine has 2 types of data movement policies. The first is MIGRATION where files are migrated based on a specific criteria or if we set a threshold on the pool (e.g. when the pool reaches 70% utilisation migrate data to a different pool). The second is a file PLACEMENT policy where files are placed in the specific pool when they are created. We will make use of both types of policies.

Note, it's possible to define Spectrum Scale ILM policies via the GUI or via the command line.

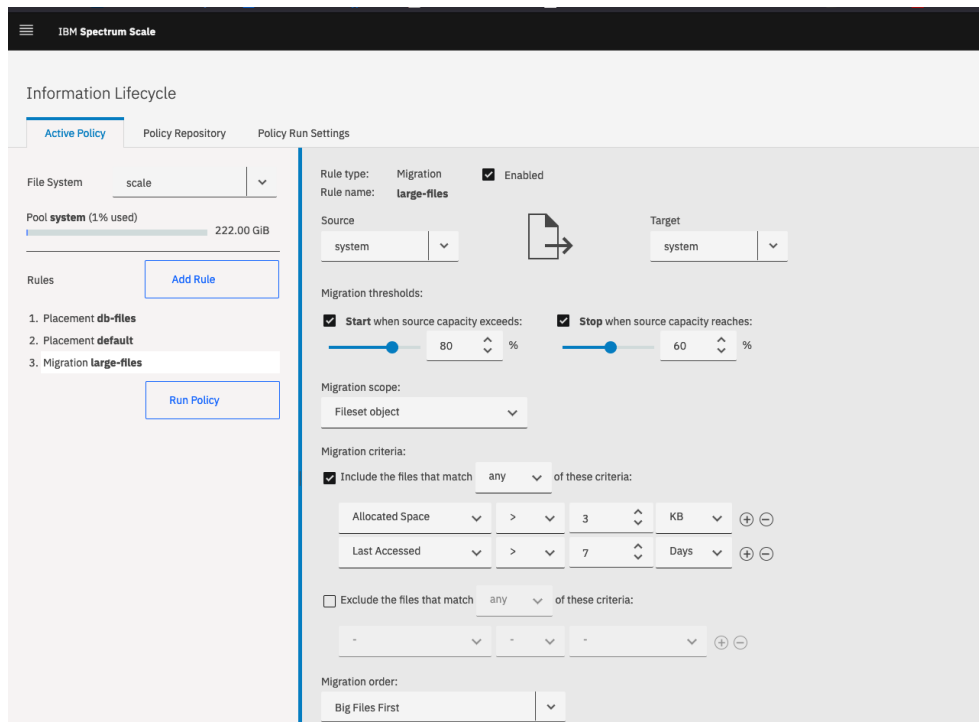


Figure 19: Defining ILM policies via the Spectrum Scale GUI

Prior to adding an explicit policy for a filesystem, the default policy configured by Spectrum Scale is as follows:

```
[root@objectcluster ~]# mmlspolicy objectfs -L
/* DEFAULT */
/* Store data in the first data pool or system pool */
[root@objectcluster ~]#
```

If you need to re-instate the default policy, you can issue the `mmchpolicy <FSNAME> DEFAULT`. In our example, once the bronze storage pool was created, all NEW data will be directed to it based on the DEFAULT policy (since bronze is the first data pool). We don't however want our account and container data to be directed to the bronze pool (we want to keep it in the system pool which represents a high-performance storage tier) so we will have to find a way to achieve this. For now, let's create an explicit PLACEMENT policy that ensures all new objects are stored in the bronze pool for both our Storage Policies.

```
[root@objectcluster ~]# cat policyfile.pol
/* PLACEMENT policy 1 - put objects in the bronze pool upon creation */
RULE 'obj data files' SET POOL 'bronze' FOR FILESET('object fileset') WHERE NAME LIKE 'tmp%' OR NAME LIKE '%.data'
/* PLACEMENT policy 2 - put objects in the bronze pool upon creation for our custom storage policy */
RULE 'obj data files custom' SET POOL 'bronze' FOR FILESET('obj_CompressionObjectfs') WHERE NAME LIKE 'tmp%' OR NAME LIKE '%.data'
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
/* Default placement policy rule */
RULE 'default' SET POOL 'bronze'
[root@objectcluster ~]#
```

Note we explicitly look for files beginning with tmp and files ending with .data. The OpenStack Swift Object Server usually creates a temporary directory (e.g. /srv/node/device/tmp) and temporary file (e.g. /srv/node/device/tmp/tmpABCDEF) for each new object which is eventually renamed and moved to the correct location in the object ring (as per the Object directory structure depicted in Figure 12). The default rule for any other file is to reside in the bronze pool (remember, the account and container db files need to be separated out which we will cover later). Also, we are applying our policy at a fileset level so its important to create independent filesets for each new Swift Storage Policy (when creating a new storage policy you can in theory use the existing policy-0 fileset though this will limit us in terms of implementing ILM).

A filesystem can only have one active policy (usually a file PLACEMENT and/or MIGRATION THRESHOLD type policy) and one or more scheduled policies (usually MIGRATION policies based on some migration rules). Let us test our policy and then activate it.

```
[root@objectcluster ~]# mmchpolicy objectfs policyfile.pol -I test
Validated policy 'policyfile.pol': Parsed 3 policy rules.
[root@objectcluster ~]# mmchpolicy objectfs policyfile.pol -I yes
Validated policy 'policyfile.pol': Parsed 3 policy rules.
Policy 'policyfile.pol' installed and broadcast to all nodes.
[root@objectcluster ~]# mmlspolicy objectfs -L
/* PLACEMENT policy 1 - put objects in the bronze pool upon creation */
RULE 'obj data files' SET POOL 'bronze' FOR FILESET('object_fileset') WHERE NAME LIKE 'tmp%' OR NAME LIKE '%.data'
/* PLACEMENT policy 2 - put objects in the bronze pool upon creation for our custom storage policy */
RULE 'obj data files custom' SET POOL 'bronze' FOR FILESET('obj_CompressionObjectfs') WHERE NAME LIKE 'tmp%' OR
NAME LIKE '%.data'
/* Default placement policy rule */
RULE 'default' SET POOL 'bronze'
[root@objectcluster ~]#
```

Now that we have our PLACEMENT policy in effect which should cover all newly created files, we need to create a MIGRATION policy to move all the objects from the default system pool to the bronze pool (that is, objects that were created prior to us adding the bronze pool to the object filesystem). The MIGRATION policy is as follows:

```
[root@objectcluster ~]# cat migratepolicy.pol
/* policy rule to migrate existing objects from the system pool to the bronze pool */
RULE 'obj_migration_policy'
MIGRATE FROM POOL system TO POOL bronze FOR FILESET('object_fileset') WHERE NAME LIKE 'tmp%' OR NAME LIKE '%.data'
[root@objectcluster ~]#
```

Essentially we want to move all files beginning with tmp or ending with .data to the bronze pool. Let's test and run this policy. Note, this policy only covers the default Storage Policy (*policy-0* which is bound to the *object_fileset* fileset). A recap of our fileset definitions:

```
[root@objectcluster ~]# mmlsfileset objectfs
Filesets in file system 'objectfs':
Name      Status  Path
root      Linked  /ibm/objectfs
object_fileset  Linked  /ibm/objectfs/object_fileset
obj_CompressionObjectfs  Linked  /ibm/objectfs/obj_CompressionObjectfs
[root@objectcluster ~]#
```

We will create another MIGRATION policy for our *obj_CompressionObjectfs* fileset which is bound to our *CompressionObjectfs* Storage Policy. Let's test our MIGRATION policy for *object_fileset*:

```
[root@objectcluster ~]# mmapplypolicy objectfs -P migratepolicy.pol -I test
[I] GPFS Current Data Pool Utilization in KB and %
Pool Name      KB_Occupied      KB_Total      Percent_Occupied
bronze          73728          31457280          0.234375000%
system         5185536          52428800          9.890625000%
[I] 16616 of 4369408 inodes used: 0.380280%.
[I] Loaded policy rules from migratepolicy.pol.
Evaluating policy rules with CURRENT_TIMESTAMP = 2022-03-27@06:44:20 UTC
Parsed 1 policy rules.
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
/* policy rule to migrate existing objects from the system pool to the bronze pool */
RULE 'obj_migration_policy'
MIGRATE FROM POOL system TO POOL bronze FOR FILESET('object_fileset') WHERE NAME LIKE '%.tmp' OR NAME LIKE '%.data'
[I] 2022-03-27@06:44:21.157 Directory entries scanned: 12597.
[I] Directories scan: 8327 files, 4142 directories, 128 other objects, 0 'skipped' files and/or errors.
[I] 2022-03-27@06:44:24.979 Parallel-piped sort and policy evaluation. 12597 files scanned.
[I] 2022-03-27@06:44:24.996 Piped sorting and candidate file choosing. 5 records scanned.
[I] Summary of Rule Applicability and File Choices:
Rule#      Hit_Cnt      KB_Hit      Chosen      KB_Chosen      KB_Ill      Rule
0          5          2560000      5          2560000      0          RULE 'bronze'
MIGRATE FROM POOL 'system' TO POOL 'bronze' FOR FILESET(.) WHERE(.)

[I] Filesystem objects with no applicable rules: 12580.

[I] GPFS Policy Decisions and File Choice Totals:
Chose to migrate 2560000KB: 5 of 5 candidates;
Predicted Data Pool Utilization in KB and %:
Pool_Name      KB_Occupied      KB_Total      Percent_Occupied
bronze          2654208          31457280      8.437500000%
system          2625536          52428800      5.007812500%
[root@objectcluster ~]#
```

We know there are 5 objects that belong to the Storage Policy using this fileset (*policy-0*). As per the above, the MIGRATION policy identified the 5 objects to move. We can now perform the MIGRATION.

```
[root@objectcluster ~]# mmapplypolicy objectfs -P migratepolicy.pol -I yes
[I] GPFS Current Data Pool Utilization in KB and %
Pool_Name      KB_Occupied      KB_Total      Percent_Occupied
bronze          135168          31457280      0.429687500%
system          5185536          52428800      9.890625000%
[I] 16616 of 4369408 inodes used: 0.380280%.
[I] Loaded policy rules from migratepolicy.pol.
Evaluating policy rules with CURRENT_TIMESTAMP = 2022-03-27@06:44:50 UTC
Parsed 1 policy rules.
/* policy rule to migrate existing objects from the system pool to the bronze pool */
RULE 'obj_migration_policy'
MIGRATE FROM POOL system TO POOL bronze FOR FILESET('object_fileset') WHERE NAME LIKE '%.tmp' OR NAME LIKE '%.data'
[I] 2022-03-27@06:44:50.856 Directory entries scanned: 12587.
[I] Directories scan: 8317 files, 4142 directories, 128 other objects, 0 'skipped' files and/or errors.
[I] 2022-03-27@06:44:51.081 Parallel-piped sort and policy evaluation. 12587 files scanned.
[I] 2022-03-27@06:44:51.109 Piped sorting and candidate file choosing. 5 records scanned.
[I] Summary of Rule Applicability and File Choices:
Rule#      Hit_Cnt      KB_Hit      Chosen      KB_Chosen      KB_Ill      Rule
0          5          2560000      5          2560000      0          RULE 'bronze'
MIGRATE FROM POOL 'system' TO POOL 'bronze' FOR FILESET(.) WHERE(.)

[I] Filesystem objects with no applicable rules: 12572.

[I] GPFS Policy Decisions and File Choice Totals:
Chose to migrate 2560000KB: 5 of 5 candidates;
Predicted Data Pool Utilization in KB and %:
Pool_Name      KB_Occupied      KB_Total      Percent_Occupied
bronze          2658304          31457280      8.450520833%
system          2625536          52428800      5.007812500%
[I] 2022-03-27@06:44:51.740 Policy execution. 5 files dispatched.
[I] A total of 5 files have been migrated, deleted or processed by an EXTERNAL EXEC/script;
0 'skipped' files and/or errors.
[root@objectcluster ~]#
```

Let's now do the same MIGRATION for the *obj_CompressionObjectfs* fileset which corresponds to our *CompressionObjectfs* storage policy.

```
[root@objectcluster ~]# cat migratepolicy_compress.pol
/* policy rule to migrate existing objects from the system pool to the bronze pool for our custom storage policy */
RULE 'obj_migration_policy_custom'
MIGRATE FROM POOL system TO POOL bronze FOR FILESET('obj_CompressionObjectfs') WHERE NAME LIKE 'tmp%' OR NAME LIKE '%.data'
[root@objectcluster ~]#
[root@objectcluster ~]# mmapplypolicy objectfs -P migratepolicy_compress.pol -I test
[I] GPFS Current Data Pool Utilization in KB and %
Pool_Name      KB_Occupied      KB_Total      Percent_Occupied
bronze          2678784          31457280      8.515625000%
system          2625536          52428800      5.007812500%
[I] 16616 of 4369408 inodes used: 0.380280%.
[I] Loaded policy rules from migratepolicy_compress.pol.
Evaluating policy rules with CURRENT_TIMESTAMP = 2022-03-27@06:46:01 UTC
Parsed 1 policy rules.
/* policy rule to migrate existing objects from the system pool to the bronze pool for our custom storage policy */
RULE 'obj_migration_policy_custom'
MIGRATE FROM POOL system TO POOL bronze FOR FILESET('obj_CompressionObjectfs') WHERE NAME LIKE 'tmp%' OR NAME LIKE '%.data'
[I] 2022-03-27@06:46:02.063 Directory entries scanned: 12597.
[I] Directories scan: 8327 files, 4142 directories, 128 other objects, 0 'skipped' files and/or errors.
[I] 2022-03-27@06:46:02.224 Parallel-piped sort and policy evaluation. 12597 files scanned.
[I] 2022-03-27@06:46:02.233 Piped sorting and candidate file choosing. 5 records scanned.
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
[I] Summary of Rule Applicability and File Choices:
Rule#      Hit_Cnt      KB_Hit      Chosen      KB_Chosen      KB_Ill      Rule
0           5           266240      5           266240         0      RULE 'bronze'
MIGRATE FROM POOL 'system' TO POOL 'bronze' FOR FILESET(.) WHERE(.)

[I] Filesystem objects with no applicable rules: 12577.

[I] GPFS Policy Decisions and File Choice Totals:
Chose to migrate 266240KB: 5 of 5 candidates;
Predicted Data Pool Utilization in KB and %:
Pool Name      KB_Occupied      KB_Total      Percent_Occupied
bronze          2940928        31457280        9.348958333%
system         2359296        52428800        4.500000000%
[root@objectcluster ~]#
```

We know that we had 5 objects in the compressed bucket that we created for this test and the MIGRATION policy test identified 5 files as expected. We can now perform the actual MIGRATION.

```
[root@objectcluster ~]# mmapplypolicy objectfs -P migratepolicy_compress.pol -I yes
[I] GPFS Current Data Pool Utilization in KB and %
Pool Name      KB_Occupied      KB_Total      Percent_Occupied
bronze          2633728        31457280        8.372395833%
system         2625536        52428800        5.007812500%
[I] 16616 of 4369408 inodes used: 0.380280%.
[I] Loaded policy rules from migratepolicy_compress.pol.
Evaluating policy rules with CURRENT_TIMESTAMP = 2022-03-27@06:46:18 UTC
Parsed 1 policy rules.
/* policy rule to migrate existing objects from the system pool to the bronze pool for our custom storage policy */
RULE 'obj_migration_policy_custom'
MIGRATE FROM POOL system TO POOL bronze FOR FILESET('obj_CompressionObjectfs') WHERE NAME LIKE 'tmp%' OR NAME LIKE
'%.data'
[I] 2022-03-27@06:46:19.007 Directory entries scanned: 12597.
[I] Directories scan: 8327 files, 4142 directories, 128 other objects, 0 'skipped' files and/or errors.
[I] 2022-03-27@06:46:19.576 Parallel-piped sort and policy evaluation. 12597 files scanned.
[I] 2022-03-27@06:46:19.600 Piped sorting and candidate file choosing. 5 records scanned.
[I] Summary of Rule Applicability and File Choices:
Rule#      Hit_Cnt      KB_Hit      Chosen      KB_Chosen      KB_Ill      Rule
0           5           266240      5           266240         0      RULE 'bronze'
MIGRATE FROM POOL 'system' TO POOL 'bronze' FOR FILESET(.) WHERE(.)

[I] Filesystem objects with no applicable rules: 12577.

[I] GPFS Policy Decisions and File Choice Totals:
Chose to migrate 266240KB: 5 of 5 candidates;
Predicted Data Pool Utilization in KB and %:
Pool Name      KB_Occupied      KB_Total      Percent_Occupied
bronze          2932736        31457280        9.322916667%
system         2359296        52428800        4.500000000%
[I] 2022-03-27@06:46:19.744 Policy execution. 5 files dispatched.
[I] A total of 5 files have been migrated, deleted or processed by an EXTERNAL EXEC/script;
0 'skipped' files and/or errors.
[root@objectcluster ~]#
```

If we query the storage pools after the MIGRATION policies were run, we should see our bronze pool populate and a reduction in used capacity for the system pool.

```
[root@objectcluster ~]# mmlspool objectfs
Storage pools in file system at '/ibm/objectfs':
Name      Id      BlkSize Data Meta Total Data in (KB) Free Data in (KB) Total Meta in (KB) Free
Meta in (KB)
system      0        4 MB yes yes 52428800 50069504 ( 96%) 52428800 50327552 (
96%)
bronze     131073    4 MB yes no 31457280 28557312 ( 91%) 0 0 (
0%)
[root@objectcluster ~]#
```

As expected, our free data in KB has gone up for the system pool and down for the bronze pool. All our existing objects are migrated to the bronze pool and our FILE PLACEMENT policy will ensure that all new objects will also be stored in our bronze pool. Our account and container information stored as .db files still reside in the system pool on the *object_files* fileset which was the fileset specified when deploying the OBJ protocol (Spectrum Scale uses the same fileset for account/container and object rings as discussed earlier). The problem we now have is that the DEFAULT rule in our FILE PLACEMENT policy means that ANY new data (whether object, container or account) is directed to the bronze pool which is not what we want. We want all new object data to be stored in the bronze pool but anything else (which would be account and container information) be directed to the high-performance pool or system pool in our example. The easiest way to achieve this is to move our account and container ring

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

to a separate fileset and create another FILE PLACEMENT rule to ensure that any file with a .db extension is directed to the new fileset which resides in the high-performance storage pool or system pool in our example. Let's try and move account and container ring which is located by default in the /ibm/objectfs/object_fileset/ac directory to a separate fileset as follows:

Note, if you chose not to use the Installation Toolkit for protocol deployment and did it manually, you could also ensure that the account and container ring is separated to its own fileset at installation.

```
[root@objectcluster object_fileset]# pwd
/ibm/objectfs/object_fileset
[root@objectcluster object_fileset]# ls
ac  o
[root@objectcluster object_fileset]#
```

First we need to stop the OBJ protocol services to ensure that all the data on the filesystem is consistent.

```
[root@objectcluster object_fileset]# mmces service stop OBJ
Keystone DB: service successfully stopped.
Keystone: service successfully stopped.
Swift: service successfully stopped.
[root@objectcluster object_fileset]#
```

Now we move our ac directory and then create a new fileset for the account and container ring and link it to the old location. Note, this fileset is a dependent fileset.

```
[root@objectcluster object_fileset]# mv ac ac_new
[root@objectcluster object_fileset]# mmcrfileset objectfs obj ac --inode-space object_fileset
Fileset obj_ac created with id 3 root inode 139777.

[root@objectcluster object_fileset]# mmlinkfileset objectfs obj ac -J /ibm/objectfs/object_fileset/ac
Fileset obj_ac linked at /ibm/objectfs/object_fileset/ac
[root@objectcluster object_fileset]#
```

We should have a new fileset created just for our account and container ring.

```
[root@objectcluster object_fileset]# mmlsfileset objectfs
Filesets in file system 'objectfs':
Name                Status    Path
root                Linked   /ibm/objectfs
object_fileset      Linked   /ibm/objectfs/object_fileset
obj_CompressionObjectfs  Linked   /ibm/objectfs/obj_CompressionObjectfs
obj_ac              Linked   /ibm/objectfs/object_fileset/ac
[root@objectcluster object_fileset]#
```

Now that we have separated our account and container ring, we need to copy the original contents of the ac directory to this new fileset. Remember to ensure the directory ownership and permissions are restored to their original values.

```
[root@objectcluster object_fileset]# chown -R swift:swift ac
[root@objectcluster object_fileset]# chmod -R 750 ac
[root@objectcluster object_fileset]# ls -al
total 282
drwxr-x---.  5 swift swift  4096 Mar 27 09:15 .
drwxr-xr-x.  5 root  root  262144 Mar 27 09:02 ..
drwxr-x---.  2 swift swift  4096 Mar 27 09:15 ac
drwxr-x---. 130 swift swift  8192 Mar 20 03:17 ac_new
drwxr-x---. 130 swift swift 16384 Mar 25 22:23 o
dr-xr-xr-x.  2 root  root   8192 Jan  1 1970 .snapshots
[root@objectcluster object_fileset]#
root@objectcluster ac_new]# cp -pr * /ibm/objectfs/object_fileset/ac
[root@objectcluster ac_new]#
```

Before restarting the OBJ protocol service let's do a quick sanity check to make sure everything copied across.

```
[root@objectcluster object_fileset]# du -hs ac
9.2M    ac
[root@objectcluster object_fileset]# du -hs ac_new
9.2M    ac_new
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
[root@objectcluster object_fileset]# ls -lR ac | wc -l
3282
[root@objectcluster object_fileset]# ls -lR ac_new | wc -l
3282
```

Let's restart the OBJ protocol service and do a basic test to download objects from one of our existing containers.

```
[root@objectcluster ~]# mmces service start OBJ
Swift: service successfully started.
Keystone DB: service successfully started.
Keystone: service successfully started.
[root@objectcluster ~]# mmces service list -v
Enabled services: OBJ
  OBJ is running
    OBJ:openstack-swift-object-updater          is running
    OBJ:openstack-swift-object-expirer          is running
    OBJ:ibmobjectizer                           is not running
    OBJ:openstack-swift-object                  is running
    OBJ:openstack-swift-account                 is running
    OBJ:openstack-swift-container               is running
    OBJ:memcached                               is running
    OBJ:openstack-swift-proxy                   is running
    OBJ:openstack-swift-object-replicator       is running
    OBJ:openstack-swift-account-reaper          is running
    OBJ:openstack-swift-account-auditor         is running
    OBJ:openstack-swift-container-auditor       is running
    OBJ:openstack-swift-container-updater       is running
    OBJ:openstack-swift-account-replicator      is running
    OBJ:openstack-swift-container-replicator    is running
    OBJ:openstack-swift-object-sof              is not running
    OBJ:postgresql-obj                         is running
    OBJ:htpdp (keystone)                       is running
[root@objectcluster ~]#
```

From our object client VM we will GET the objects from our mybucket container.

```
[root@scalecluster ~]# s3cmd ls
2009-02-03 16:45  s3://compressbucket
2009-02-03 16:45  s3://mybucket

[root@scalecluster testfiles]# s3cmd get --recursive --force s3://mybucket
download: 's3://mybucket/file13' -> './file13' [1 of 5]
 524288000 of 524288000 100% in 11s 44.94 MB/s done
download: 's3://mybucket/file14' -> './file14' [2 of 5]
 524288000 of 524288000 100% in 12s 41.52 MB/s done
download: 's3://mybucket/file15' -> './file15' [3 of 5]
 524288000 of 524288000 100% in 12s 41.54 MB/s done
download: 's3://mybucket/file16' -> './file16' [4 of 5]
 524288000 of 524288000 100% in 20s 24.52 MB/s done
download: 's3://mybucket/file17' -> './file17' [5 of 5]
 524288000 of 524288000 100% in 15s 33.20 MB/s done
[root@scalecluster testfiles]#
```

Everything looks good as we can access our containers and objects created prior to splitting out the account and container ring into a separate fileset. Before adding a new FILE PLACEMENT rule, additional objects were uploaded to our Object store so that they are directed by the DEFAULT placement rule to the bronze pool. This means we have account and container data that needs to be MIGRATED to the system pool. This was done just to showcase how we can apply a MIGRATION policy for our account and container files. In reality, just a FILE PLACEMENT policy would suffice for the *obj_ac* fileset to ensure all account and container information is restricted to the high-performance storage pool or system pool in our example.

We will now add new rule to our existing FILE PLACEMENT policy called “*object db files*” and test and activate the new FILE PLACEMENT policy as follows:

```
[root@objectcluster ~]# cat policyfile.pol
/* PLACEMENT policy 1 - put account and container files in system */
RULE 'object db files' SET POOL 'system' FOR FILESET('obj_ac')
/* PLACEMENT policy 2 - put objects in the bronze pool upon creation */
RULE 'obj data files' SET POOL 'bronze' FOR FILESET('object_fileset') WHERE NAME LIKE 'tmp%' OR NAME LIKE '%.data'
/* PLACEMENT policy 3 - put objects in the bronze pool upon creation for our custom storage policy */
RULE 'obj data files custom' SET POOL 'bronze' FOR FILESET('obj_CompressionObjectfs') WHERE NAME LIKE 'tmp%' OR NAME LIKE '%.data'
/* Default placement policy rule */
RULE 'default' SET POOL 'bronze'
[root@objectcluster ~]# mmchpolicy objectfs policyfile.pol -I test
```


Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
Validated policy 'policyfile.pol': Parsed 4 policy rules.
[root@objectcluster ~]# mmchpolicy objectfs policyfile.pol -I yes
Validated policy 'policyfile.pol': Parsed 4 policy rules.
Policy 'policyfile.pol' installed and broadcast to all nodes.
[root@objectcluster ~]# mmlspolicy objectfs -L
/* PLACEMENT policy 1 - put account and container files in system */
RULE 'object db files' SET POOL 'system' FOR FILESET('obj_ac')
/* PLACEMENT policy 2 - put objects in the bronze pool upon creation */
RULE 'obj data files' SET POOL 'bronze' FOR FILESET('object_fileset') WHERE NAME LIKE 'tmp%' OR NAME LIKE '%.data'
/* PLACEMENT policy 3 - put objects in the bronze pool upon creation for our custom storage policy */
RULE 'obj data files custom' SET POOL 'bronze' FOR FILESET('obj_CompressionObjectfs') WHERE NAME LIKE 'tmp%' OR
NAME LIKE '%.data'
/* Default placement policy rule */
RULE 'default' SET POOL 'bronze'
[root@objectcluster ~]#
```

Effectively, any new files that need to be written to the `ac` directory (which we have separated into a separate fileset called `obj_ac`) should reside in our high-performance storage pool or system pool in our example. For the account and container information that was directed to the bronze pool (we uploaded additional objects to the Object store before creating the FILE PLACEMENT rule for our `obj_ac` fileset) we need to now MIGRATE them to the system pool. We can do this by creating a MIGRATION policy as follows:

```
[root@objectcluster ~]# cat dbpol.pol
/* policy rule to migrate existing objects from the bronze pool to the system pool */
RULE 'obj_migration_policy'
MIGRATE FROM POOL bronze TO POOL system FOR FILESET('obj_ac')
[root@objectcluster ~]#
```

We need to test the new MIGRATION policy.

```
[root@objectcluster ~]# mmapplypolicy objectfs -P dbpol.pol -I test
[I] GPFS Current Data Pool Utilization in KB and %
Pool_Name      KB_Occupied      KB_Total      Percent_Occupied
bronze          2957312          31457280          9.401041667%
system          2367488          52428800          4.515625000%
[I] 26100 of 4369408 inodes used: 0.597335%.
[I] Loaded policy rules from dbpol.pol.
Evaluating policy rules with CURRENT_TIMESTAMP = 2022-03-27@08:07:52 UTC
Parsed 1 policy rules.
/* policy rule to migrate existing objects from the system pool to the bronze pool */
RULE 'obj_migration_policy'
MIGRATE FROM POOL bronze TO POOL system FOR FILESET('obj_ac')
[I] 2022-03-27@08:07:55.889 Directory entries scanned: 22081.
[I] Directories scan: 12873 files, 9080 directories, 128 other objects, 0 'skipped' files and/or errors.
[I] 2022-03-27@08:07:56.118 Parallel-piped sort and policy evaluation. 22081 files scanned.
[I] 2022-03-27@08:07:56.143 Piped sorting and candidate file choosing. 412 records scanned.
[I] Summary of Rule Applicability and File Choices:
Rule#      Hit_Cnt      KB_Hit      Chosen      KB_Chosen      KB_ILL      Rule
0           412           8976          412          8976           0      RULE 'system'
MIGRATE FROM POOL 'bronze' TO POOL 'system' FOR FILESET(.)

[I] Filesystem objects with no applicable rules: 21669.

[I] GPFS Policy Decisions and File Choice Totals:
Chose to migrate 8976KB: 412 of 412 candidates;
Predicted Data Pool Utilization in KB and %:
Pool_Name      KB_Occupied      KB_Total      Percent_Occupied
bronze          2968816          31457280          9.437611898%
system          2376464          52428800          4.532745361%
```

It has identified 412 hits that need to be migrated. We can now perform the actual MIGRATION.

```
[root@objectcluster ~]# mmapplypolicy objectfs -P dbpol.pol -I yes
[I] GPFS Current Data Pool Utilization in KB and %
Pool_Name      KB_Occupied      KB_Total      Percent_Occupied
bronze          2994176          31457280          9.518229167%
system          2367488          52428800          4.515625000%
[I] 26100 of 4369408 inodes used: 0.597335%.
[I] Loaded policy rules from dbpol.pol.
Evaluating policy rules with CURRENT_TIMESTAMP = 2022-03-27@08:08:26 UTC
Parsed 1 policy rules.
/* policy rule to migrate existing objects from the system pool to the bronze pool */
RULE 'obj_migration_policy'
MIGRATE FROM POOL bronze TO POOL system FOR FILESET('obj_ac')
[I] 2022-03-27@08:08:30.074 Directory entries scanned: 22081.
[I] Directories scan: 12873 files, 9080 directories, 128 other objects, 0 'skipped' files and/or errors.
[I] 2022-03-27@08:08:30.277 Parallel-piped sort and policy evaluation. 22081 files scanned.
[I] 2022-03-27@08:08:30.297 Piped sorting and candidate file choosing. 412 records scanned.
[I] Summary of Rule Applicability and File Choices:
Rule#      Hit_Cnt      KB_Hit      Chosen      KB_Chosen      KB_ILL      Rule
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
0          412          8976          412          8976          0          RULE 'system'
MIGRATE FROM POOL 'bronze' TO POOL 'system' FOR FILESET(.)

[I] Filesystem objects with no applicable rules: 21669.

[I] GPFS Policy Decisions and File Choice Totals:
  Chose to migrate 8976KB: 412 of 412 candidates;
  Predicted Data Pool Utilization in KB and %:
  Pool_Name          KB_Occupied          KB_Total          Percent_Occupied
  bronze             2952432             31457280           9.385528564%
  system             2376464             52428800           4.532745361%
[I] 2022-03-27@08:08:30.724 Policy execution. 412 files dispatched.
[I] A total of 412 files have been migrated, deleted or processed by an EXTERNAL EXEC/script;
    0 'skipped' files and/or errors.
[root@objectcluster ~]#
```

As a last test to check if our FILE PLACEMENT policy is working, we will create a bunch of containers and upload some objects to them and check to see if the account/container info and objects go to the correct storage pools. First let's create some new containers.

```
[root@scalecluster testfiles]# for BUCKET in {1..15}; do s3cmd mb s3://test${BUCKET}; done
Bucket 's3://test1/' created
Bucket 's3://test2/' created
Bucket 's3://test3/' created
Bucket 's3://test4/' created
Bucket 's3://test5/' created
Bucket 's3://test6/' created
Bucket 's3://test7/' created
Bucket 's3://test8/' created
Bucket 's3://test9/' created
Bucket 's3://test10/' created
Bucket 's3://test11/' created
Bucket 's3://test12/' created
Bucket 's3://test13/' created
Bucket 's3://test14/' created
Bucket 's3://test15/' created
[root@scalecluster testfiles]#
```

And some test files to upload.

```
root@scalecluster smallfiles]# for FILE in {1..100}; do dd if=/dev/zero of=smallfile${FILE} bs=20K count=1; done
.
.
20480 bytes (20 kB, 20 KiB) copied, 9.0687e-05 s, 226 MB/s
1+0 records in
1+0 records out
20480 bytes (20 kB, 20 KiB) copied, 9.4887e-05 s, 216 MB/s
[root@scalecluster smallfiles]#
```

Now let's use s3cmd to upload them as objects to the newly created containers.

```
[root@scalecluster testfiles]# for BUCKET in {1..15}; do s3cmd put --recursive smallfiles/ s3://test${BUCKET}; done
upload: 'smallfiles/smallfile1' -> 's3://test1/smallfile1' [1 of 100]
 20480 of 20480 100% in 0s 163.19 KB/s done
upload: 'smallfiles/smallfile10' -> 's3://test1/smallfile10' [2 of 100]
 20480 of 20480 100% in 0s 145.72 KB/s done
upload: 'smallfiles/smallfile100' -> 's3://test1/smallfile100' [3 of 100]
 20480 of 20480 100% in 0s 117.73 KB/s done
upload: 'smallfiles/smallfile11' -> 's3://test1/smallfile11' [4 of 100]
 20480 of 20480 100% in 0s 248.31 KB/s done
upload: 'smallfiles/smallfile12' -> 's3://test1/smallfile12' [5 of 100]
 20480 of 20480 100% in 0s 240.12 KB/s done.
.
.
upload: 'smallfiles/smallfile97' -> 's3://test15/smallfile97' [98 of 100]
 20480 of 20480 100% in 0s 305.53 KB/s done
upload: 'smallfiles/smallfile98' -> 's3://test15/smallfile98' [99 of 100]
 20480 of 20480 100% in 0s 277.75 KB/s done
upload: 'smallfiles/smallfile99' -> 's3://test15/smallfile99' [100 of 100]
 20480 of 20480 100% in 0s 296.26 KB/s done
[root@scalecluster testfiles]#
```

If the FILE PLACEMENT policy is working for account and container files with a .db extension, we should have NO files to MIGRATE from the bronze pool.

```
[root@objectcluster ~]# mmapplypolicy objectfs -P dbpol.pol -I test
[I] GPFS Current Data Pool Utilization in KB and %
Pool_Name          KB_Occupied          KB_Total          Percent_Occupied
bronze             3002368             31457280           9.544270833%
system             2375680             52428800           4.531250000%
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
[I] 28080 of 4369408 inodes used: 0.642650%.
[I] Loaded policy rules from dbpol.pol.
Evaluating policy rules with CURRENT_TIMESTAMP = 2022-03-27@08:42:32 UTC
Parsed 1 policy rules.
/* policy rule to migrate existing objects from the system pool to the bronze pool */
RULE 'obj_migration_policy'
MIGRATE FROM POOL bronze TO POOL system FOR FILESET('obj ac')
[I] 2022-03-27@08:42:41.472 Directory entries scanned: 24061.
[I] Directories scan: 13814 files, 10119 directories, 128 other objects, 0 'skipped' files and/or errors.
[I] 2022-03-27@08:42:42.291 Parallel-piped sort and policy evaluation. 24061 files scanned.
[I] 2022-03-27@08:42:42.440 Piped sorting and candidate file choosing. 0 records scanned.
[I] Summary of Rule Applicability and File Choices:
Rule#      Hit_Cnt      KB_Hit      Chosen      KB_Chosen      KB_Ill      Rule
0          0          0          0          0          0      RULE 'system'
MIGRATE FROM POOL 'bronze' TO POOL 'system' FOR FILESET(.)

[I] Filesystem objects with no applicable rules: 24061.

Predicted Data Pool Utilization in KB and %:
Pool_Name      KB_Occupied      KB_Total      Percent_Occupied
bronze         2990080          31457280      9.505208333%
system         2375680          52428800      4.531250000%
[root@objectcluster ~]#
```

Sure enough, no new account or container information was found in the bronze pool which indicates that our account and container information all resides in the high-performance storage pool or system pool in our example and that our FILE PLACEMENT policy for object db files is working correctly. That concludes this section on how to implement ILM for Spectrum Scale Object. This might be required if you want to reduce costs by using different tiers of storage for your object filesystem. The above examples showcase how you can use Spectrum Scale's ILM policy engine to achieve this goal.

Object Tiering

As mentioned previously, some public cloud providers like Amazon S3 offer object tiering where objects that are frequently accessed are kept in the highest-performance storage tier or class and objects that are less frequently accessed are migrated to a lower performance storage tier or class automatically. Amazon S3 refers to this storage class as Intelligent Tiering. This functionality carries with it an additional software cost. In Spectrum Scale, we can offer similar functionality where we move files based on access patterns across different storage pools that make up the filesystem. This will ensure for example, that frequently accessed or “hot” objects always reside in the highest-performance storage pool and less frequently accessed or “cold” objects are migrated to a lower performance storage pool. We achieve this using Spectrum Scale File Heat tiering policies which is discussed here <https://www.ibm.com/docs/en/spectrum-scale/5.1.3?topic=scale-file-heat-tracking-file-access-temperature>.

Objects with a .data extension are ideal candidates for heatmap tiering. In our example, we want to tier objects from the lower cost bronze pool to the higher performance system pool and vice versa based on their access pattern where frequently accessed objects will reside in the system pool and less frequently accessed objects will reside in the bronze pool. In a real world environment you might want to tier objects across more than just two storage pools (including to tape via Spectrum Archive).

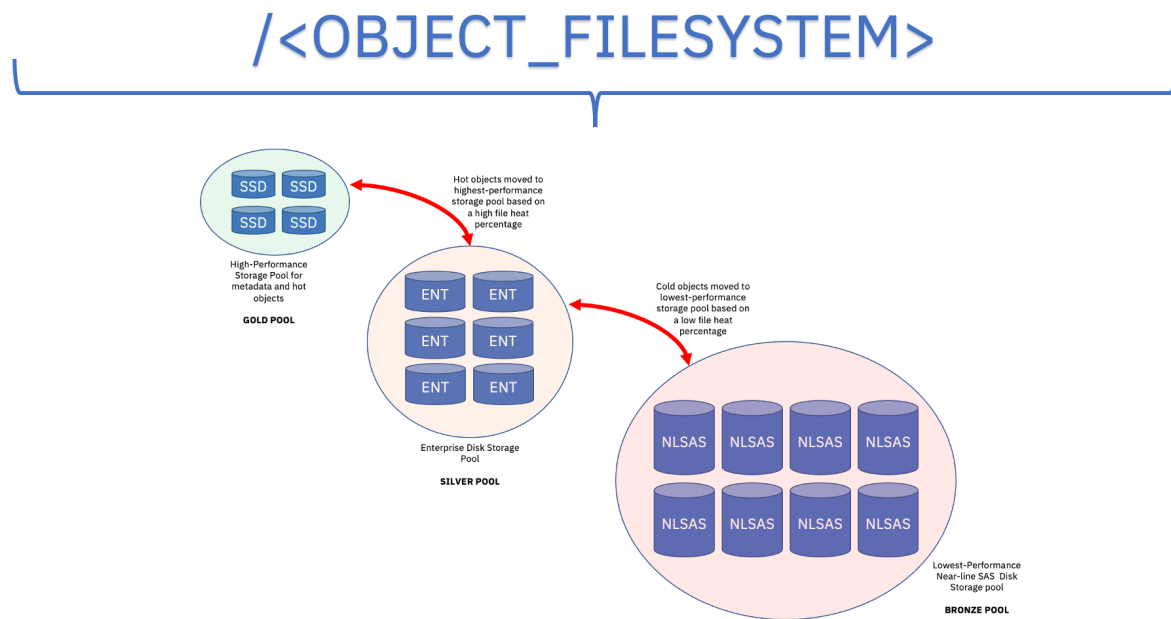


Figure 20: Example Object tiering configuration

There are a few rules before enabling the built-in FILEHEAT attribute for Spectrum Scale files. These are documented here <https://www.ibm.com/docs/en/spectrum-scale/5.1.3?topic=scale-file-heat-tracking-file-access-temperature>. The most important rules to remember is that once enabled, Spectrum Scale needs to be restarted for this to take effect. Also, you need to ensure that suppression of file access time or atime is not suppressed. Lastly, there are two parameters that you need to specify when enabling this feature which are *fileHeatPeriodMinutes* and *fileHeatLossPercent*. The *fileHeatPeriodMinutes* parameter basically specifies the frequency with which file heat attributes are updated and the *fileHeatLossPercent* parameter specifies the percent of file access heat that an unaccessed file loses at the end of each tracking interval (the default value is 10 percent). We will use a 5 minute tracking interval and leave the default heat loss percent at 10.

Firstly, let us enable file heatmap tracking and shutdown our cluster.

```
[root@objectcluster ~]# mmchconfig fileheatperiodminutes=5,fileheatlosspercent=10
mmchconfig: Command successfully completed
mmchconfig: mmsdrfs propagation completed.
[root@objectcluster ~]# mmshutdown
Shutting down the following quorum nodes will cause the cluster to lose quorum:
objectcluster.scale.com
Do you wish to continue [yes/no]: yes
Sun Mar 27 03:47:46 SAST 2022: mmshutdown: Starting force unmount of GPFS file systems
Sun Mar 27 03:48:21 SAST 2022: mmshutdown: Shutting down GPFS daemons
Sun Mar 27 03:48:27 SAST 2022: mmshutdown: Finished
[root@objectcluster ~]#
```

We need to set the atime attribute on our object filesystem to NOT suppress the file access time and restart our cluster. Both `mmchfs <FILESYSTEM> -S no` or `-S relatime` will suffice.

```
[root@objectcluster ~]# mmchfs objectfs -S no
mmchfs: mmsdrfs propagation completed.
[root@objectcluster ~]# mmstartup
```

Once the cluster is back up, we will try to generate some activity and check if the heatmap file attribute is in fact being tracked and updated. You could issue the `dd` command for example to perform reads on existing objects. We will basically just use `s3cmd` to generate some read activity to two objects and

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

then find them in our object filesystem and query their heatmap attribute. Firstly, let us use s3cmd to GET some objects (smallfile98 and smallfile99).

```
[root@scaletcluster smallfiles]# for COUNT in {1..1000}; do s3cmd get --force s3://test15/smallfile98;s3cmd get --force s3://test15/smallfile99;done
download: 's3://test15/smallfile98' -> './smallfile98' [1 of 1]
20480 of 20480 100% in 0s 288.97 KB/s done
download: 's3://test15/smallfile99' -> './smallfile99' [1 of 1]
20480 of 20480 100% in 0s 308.18 KB/s done
download: 's3://test15/smallfile98' -> './smallfile98' [1 of 1]
20480 of 20480 100% in 0s 399.90 KB/s done
download: 's3://test15/smallfile99' -> './smallfile99' [1 of 1]
20480 of 20480 100% in 0s 440.07 KB/s done
download: 's3://test15/smallfile98' -> './smallfile98' [1 of 1]
20480 of 20480 100% in 0s 454.23 KB/s done
download: 's3://test15/smallfile99' -> './smallfile99' [1 of 1]
20480 of 20480 100% in 0s 468.63 KB/s done
download: 's3://test15/smallfile98' -> './smallfile98' [1 of 1]
20480 of 20480 100% in 0s 411.66 KB/s done
download: 's3://test15/smallfile99' -> './smallfile99' [1 of 1]
20480 of 20480 100% in 0s 424.62 KB/s done
download: 's3://test15/smallfile98' -> './smallfile98' [1 of 1]
20480 of 20480 100% in 0s 415.28 KB/s done
.
.
.
```

We need to locate them in our filesystem so we will use the swift-get-nodes utility as we did before.

```
[root@objectcluster ~]# swift-get-nodes /etc/swift/object.ring.gz AUTH_224f53dal2e04b7893783821a34c171f test15
smallfile98

Account AUTH_224f53dal2e04b7893783821a34c171f
Container test15
Object smallfile98

Partition 12041
Hash bc2659c98cba07f6399929262b10c969

Server:Port Device 192.168.226.252:6200 z1device54
Server:Port Device 192.168.226.252:6200 z1device63 [Handoff]

.
.
.
Use your own device location of servers:
such as "export DEVICE=/srv/node"
ssh 192.168.226.252 "ls -lah ${DEVICE:-/srv/node*}/z1device54/objects/12041/969/bc2659c98cba07f6399929262b10c969"
ssh 192.168.226.252 "ls -lah ${DEVICE:-/srv/node*}/z1device63/objects/12041/969/bc2659c98cba07f6399929262b10c969" #
[Handoff]

note: '/srv/node*' is used as default value of 'devices', the real value is set in the config file on each storage
node.
[root@objectcluster ~]# swift-get-nodes /etc/swift/object.ring.gz AUTH_224f53dal2e04b7893783821a34c171f test15
smallfile99

Account AUTH_224f53dal2e04b7893783821a34c171f
Container test15
Object smallfile99

Partition 4525
Hash 46b797c5fe5e08b296e7709b4c346fd0

Server:Port Device 192.168.226.252:6200 z1device90
Server:Port Device 192.168.226.252:6200 z1device60 [Handoff]

.
.
.
Use your own device location of servers:
such as "export DEVICE=/srv/node"
ssh 192.168.226.252 "ls -lah ${DEVICE:-/srv/node*}/z1device90/objects/4525/fd0/46b797c5fe5e08b296e7709b4c346fd0"
ssh 192.168.226.252 "ls -lah ${DEVICE:-/srv/node*}/z1device60/objects/4525/fd0/46b797c5fe5e08b296e7709b4c346fd0" #
[Handoff]

note: '/srv/node*' is used as default value of 'devices', the real value is set in the config file on each storage
node.
[root@objectcluster ~]#
```

Once we have located them, we can query their attributes to see if they include the gpfs.FileHeat.

```
[root@objectcluster ~]# ls -al
/ibm/objectfs/object_fileset/o/z1device54/objects/12041/969/bc2659c98cba07f6399929262b10c969/1648370448.95369.data
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
-rwxr-xr-x. 1 swift swift 20480 Mar 27 10:40
/ibm/objectfs/object_fileset/o/zldevice54/objects/12041/969/bc2659c98cba07f6399929262b10c969/1648370448.95369.data
[root@objectcluster ~]# ls -al
/ibm/objectfs/object_fileset/o/zldevice90/objects/4525/fd0/46b797c5fe5e08b296e7709b4c346fd0/1648370449.02771.data
-rwxr-xr-x. 1 swift swift 20480 Mar 27 10:40
/ibm/objectfs/object_fileset/o/zldevice90/objects/4525/fd0/46b797c5fe5e08b296e7709b4c346fd0/1648370449.02771.data
[root@objectcluster ~]# mmlsattr -d -X
/ibm/objectfs/object_fileset/o/zldevice54/objects/12041/969/bc2659c98cba07f6399929262b10c969/1648370448.95369.data
file name:
/ibm/objectfs/object_fileset/o/zldevice54/objects/12041/969/bc2659c98cba07f6399929262b10c969/1648370448.95369.data
security.selinux
user.swift.metadata
user.swift.metadata_checksum
gpfs.FileHeat

[root@objectcluster ~]# mmlsattr -d -X
/ibm/objectfs/object_fileset/o/zldevice90/objects/4525/fd0/46b797c5fe5e08b296e7709b4c346fd0/1648370449.02771.data
file name:
/ibm/objectfs/object_fileset/o/zldevice90/objects/4525/fd0/46b797c5fe5e08b296e7709b4c346fd0/1648370449.02771.data
security.selinux
user.swift.metadata
user.swift.metadata_checksum
gpfs.FileHeat

[root@objectcluster ~]#
```

Both of our objects now have the FileHeat attribute. You can query the actual value as follows:

```
[root@objectcluster ~]# mmlsattr -d -X -L
/ibm/objectfs/object_fileset/o/zldevice90/objects/4525/fd0/46b797c5fe5e08b296e7709b4c346fd0/1648370449.02771.data
file name:
/ibm/objectfs/object_fileset/o/zldevice90/objects/4525/fd0/46b797c5fe5e08b296e7709b4c346fd0/1648370449.02771.data
metadata replication: 1 max 2
data replication: 1 max 2
immutable: no
appendOnly: no
flags:
storage pool name: bronze
fileset name: object_fileset
snapshot name:
creation time: Sun Mar 27 10:40:49 2022
Misc attributes: ARCHIVE
Encrypted: no
security.selinux: 0x73797374656D5F753A6F626A6563745F723A73776966745F646174615F743A733000
user.swift.metadata:
0x80027D710028635F636F646563730A656E636F64650A7101580B000000582D54696D657374616D70710258060000006C6174696E317103867
10452710568015810000000313634383337303434392E3032373731710668038671075271086801580C000000436F6E74656E742D5479706571
09680386710A52710B680158180000006170706C69636174696F6E2F6F637465742D73747265616D710C680386710D52710E6801580E0000004
36F6E74656E742D4C656E677468710F680386711052711168015805000000323034383071126803867113527114680158040000004554616771
1568038671165271176801582000000064616131303064663665363731313930366236316339616235616131363033327118680386711952711
A6801581A000000582D4F626A6563742D5379736D6574612D53334170692D41636C711B680386711C52711D680158610000007B224F776E6572
223A22733370726F6A6563743A733375736572222C224772616E74223A5B7B225065726D697373696F6E223A2246554C4C5F434F4E54524F4C2
22C224772616E746565223A22733370726F6A6563743A733375736572227D5D7D711E680386711F52712068015819000000582D4F626A656374
2D4D6574612D5333436D642D417474727371216803867122527123680158840000006174696D653A313634383337303232342F6374696D653A3
13634383337303136362F6769643A302F676E616D653A726F6F742F6D64353A6461613130306466366536373131393036623631633961623561
6131363033327F6D6F64653A32333138382F6D74696D653A313634383337303136362F7569643A302F756E616D653A726F6F747124680386712
5527126680158040000006E616D6571276803867128527129680158390000002F415554485F3232346635336461313265303462373839333738
3338323161333463313731662F7465737431352F736D616C6C66696C653939712A680386712B52712C752E
user.swift.metadata_checksum: 0x6563313564633834323766656431623063643530643832613561623261333261
gpfs.FileHeat: 0x0000000100010014
[root@objectcluster ~]#

[root@objectcluster ~]# mmlsattr -d -X -L
/ibm/objectfs/object_fileset/o/zldevice54/objects/12041/969/bc2659c98cba07f6399929262b10c969/1648370448.95369.data
file name:
/ibm/objectfs/object_fileset/o/zldevice54/objects/12041/969/bc2659c98cba07f6399929262b10c969/1648370448.95369.data
metadata replication: 1 max 2
data replication: 1 max 2
immutable: no
appendOnly: no
flags:
storage pool name: bronze
fileset name: object_fileset
snapshot name:
creation time: Sun Mar 27 10:40:48 2022
Misc attributes: ARCHIVE
Encrypted: no
security.selinux: 0x73797374656D5F753A6F626A6563745F723A73776966745F646174615F743A733000
user.swift.metadata:
0x80027D710028635F636F646563730A656E636F64650A7101580B000000582D54696D657374616D70710258060000006C6174696E317103867
10452710568015810000000313634383337303434382E3935333639710668038671075271086801580C000000436F6E74656E742D5479706571
09680386710A52710B680158180000006170706C69636174696F6E2F6F637465742D73747265616D710C680386710D52710E6801580E0000004
36F6E74656E742D4C656E677468710F680386711052711168015805000000323034383071126803867113527114680158040000004554616771
1568038671165271176801582000000064616131303064663665363731313930366236316339616235616131363033327118680386711952711
A6801581A000000582D4F626A6563742D5379736D6574612D53334170692D41636C711B680386711C52711D680158610000007B224F776E6572
223A22733370726F6A6563743A733375736572222C224772616E74223A5B7B225065726D697373696F6E223A2246554C4C5F434F4E54524F4C2
22C224772616E746565223A22733370726F6A6563743A733375736572227D5D7D711E680386711F52712068015819000000582D4F626A656374
2D4D6574612D5333436D642D417474727371216803867122527123680158840000006174696D653A313634383337303232342F6374696D653A3
13634383337303136362F6769643A302F676E616D653A726F6F742F6D64353A6461613130306466366536373131393036623631633961623561
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
6131363033322F6D6F64653A33333138382F6D74696D653A313634383337303136362F7569643A302F756E616D653A726F6F747124680386712
5527126680158040000006E616D6571276803867128527129680158390000002F415554485F3232346635336461313265303462373839333738
3338323161333463313731662F7465737431352F736D616C6C66696C653938712A680386712B52712C752E
user.swift.metadata checksum: 0x3935313663313634383534316233323430653539363536653030636464363234
gpfs.FileHeat: 0x0000000100010014
[root@objectcluster ~]#
```

Now that we have confirmed that heatmap tracking is enabled and working, we can create a Spectrum Scale MIGRATION policy to migrate objects based on this attribute. Note, in the policy we create we define the storage tiers in the order of highest to lowest performance (we only have system and bronze in our example). We also set a migration threshold of 70 percent so this pool will be populated until it hits 70 percent of its capacity. A more detailed explanation of defining the policy can be found here <https://www.ibm.com/docs/en/spectrum-scale/5.1.3?topic=storage-enabling-object-heatmap-policy>.

Define our heatmap tiering policy. Note, we want to migrate files based on their heatmap attribute for both our object filesets which correspond to our two Storage Policies.

```
[root@objectcluster ~]# cat fileheat.pol
/* Define pool group using two pools */
RULE 'DefineTiers' GROUP POOL 'TIERS' IS 'system' LIMIT(70) THEN 'bronze'
RULE 'Rebalance' MIGRATE FROM POOL 'TIERS' TO POOL 'TIERS' WEIGHT(FILE HEAT) FOR
FILESET('object_fileset','obj_CompressionObjectfs') WHERE NAME LIKE '%.data'
[root@objectcluster ~]#
```

Test the new policy. It has identified 1885 files to migrate. Basically it wants to put all the files we previously migrated from the system pool to the bronze pool back into the system pool (you can confirm this by looking at the Percent Occupied for each pool before and after).

```
[root@objectcluster ~]# mmapplypolicy objectfs -P fileheat.pol -I test
[I] GPFS Current Data Pool Utilization in KB and %
Pool_Name      KB_Occupied      KB_Total      Percent_Occupied
bronze          3018752          31457280      9.596354167%
system          2375680          52428800      4.531250000%
[I] 28080 of 4369408 inodes used: 0.642650%.
[I] Loaded policy rules from fileheat.pol.
Evaluating policy rules with CURRENT_TIMESTAMP = 2022-03-27@09:23:21 UTC
Parsed 2 policy rules.
/* Define pool group using two pools */
RULE 'DefineTiers' GROUP POOL 'TIERS' IS 'system' LIMIT(70) THEN 'bronze'
RULE 'Rebalance' MIGRATE FROM POOL 'TIERS' TO POOL 'TIERS' WEIGHT(computeFileHeat(CURRENT_TIMESTAMP-
ACCESS_TIME,xattr('gpfs.FileHeat'),KB_ALLOCATED)) FOR FILESET('object_fileset','obj_CompressionObjectfs') WHERE
NAME LIKE '%.data'
[I] 2022-03-27@09:23:27.920 Directory entries scanned: 24061.
[I] Directories scan: 13814 files, 10119 directories, 128 other objects, 0 'skipped' files and/or errors.
[I] 2022-03-27@09:23:28.560 Parallel-piped sort and policy evaluation. 24061 files scanned.
[I] 2022-03-27@09:23:28.658 Piped sorting and candidate file choosing. 1885 records scanned.
[I] Summary of Rule Applicability and File Choices:
Rule#      Hit_Cnt      KB_Hit      Chosen      KB_Chosen      KB_ILL      Rule
0           1885          2862240      1510          2862240          0          RULE 'TIERS'
MIGRATE FROM POOL 'TIERS' WEIGHT(.) TO POOL 'TIERS' FOR FILESET(.) WHERE(.)

[I] Filesystem objects with no applicable rules: 22176.

[I] GPFS Policy Decisions and File Choice Totals:
Chose to migrate 2862240KB: 1510 of 1885 candidates;
[I] File Migrations within Group Pools
Group      Pool      Files_Out      KB_Out      Files_In      KB_In
TIERS      system      0              0            1510          2862240
TIERS      bronze     1510           2862240      0              0
Predicted Data Pool Utilization in KB and %:
Pool_Name      KB_Occupied      KB_Total      Percent_Occupied
bronze          164704           31457280      0.523579915%
system          5237920          52428800      9.990539551%
[root@objectcluster ~]#
```

Let us run the policy.

```
[root@objectcluster ~]# mmapplypolicy objectfs -P fileheat.pol -I yes
[I] GPFS Current Data Pool Utilization in KB and %
Pool_Name      KB_Occupied      KB_Total      Percent_Occupied
bronze          3035136           31457280      9.648437500%
system          2375680           52428800      4.531250000%
[I] 28080 of 4369408 inodes used: 0.642650%.
[I] Loaded policy rules from fileheat.pol.
Evaluating policy rules with CURRENT_TIMESTAMP = 2022-03-27@09:23:35 UTC
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
Parsed 2 policy rules.
/* Define pool group using two pools */
RULE 'DefineTiers' GROUP POOL 'TIERS' IS 'system' LIMIT(70) THEN 'bronze'
RULE 'Rebalance' MIGRATE FROM POOL 'TIERS' TO POOL 'TIERS' WEIGHT(computeFileHeat(CURRENT_TIMESTAMP-
ACCESS_TIME,xattr('gpfs.FileHeat'),KB_ALLOCATED)) FOR FILESET('object_fileset','obj_CompressionObjectfs') WHERE
NAME LIKE '%.data'
[I] 2022-03-27@09:23:42.522 Directory entries scanned: 24061.
[I] Directories scan: 13814 files, 10119 directories, 128 other objects, 0 'skipped' files and/or errors.
[I] 2022-03-27@09:23:42.890 Parallel-piped sort and policy evaluation. 24061 files scanned.
[I] 2022-03-27@09:23:42.941 Piped sorting and candidate file choosing. 1885 records scanned.
[I] Summary of Rule Applicability and File Choices:
  Rule#      Hit_Cnt      KB_Hit      Chosen      KB_Chosen      KB_Ill      Rule
    0         1885      2862240      1510      2862240         0      RULE 'TIERS'
MIGRATE FROM POOL 'TIERS' WEIGHT(.) TO POOL 'TIERS' FOR FILESET(.) WHERE(.)

[I] Filesystem objects with no applicable rules: 22176.

[I] GPFS Policy Decisions and File Choice Totals:
Chose to migrate 2862240KB: 1510 of 1885 candidates;
[I] File Migrations within Group Pools
  Group      Pool      Files_Out      KB_Out      Files_In      KB_In
  TIERS      system         0         0         1510      2862240
  TIERS      bronze      1510      2862240         0         0
Predicted Data Pool Utilization in KB and %:
Pool_Name      KB_Occupied      KB_Total      Percent_Occupied
bronze          160608      31457280      0.510559082%
system          5237920      52428800      9.990539551%
[I] 2022-03-27@09:23:50.312 Policy execution. 1510 files dispatched.
[I] A total of 1510 files have been migrated, deleted or processed by an EXTERNAL EXEC/script;
    0 'skipped' files and/or errors.
[root@objectcluster ~]#
```

Just to make sure nothing is left we can test the policy again. You can schedule this policy to run via cron for example based on how frequent you want to rebalance the tiers.

```
[root@objectcluster ~]# mmapplypolicy objectfs -P fileheat.pol -I test
[I] GPFS Current Data Pool Utilization in KB and %
Pool Name      KB Occupied      KB Total      Percent Occupied
bronze         106496      31457280      0.338541667%
system         5234688      52428800      9.984375000%
[I] 28080 of 4369408 inodes used: 0.642650%.
[I] Loaded policy rules from fileheat.pol.
Evaluating policy rules with CURRENT_TIMESTAMP = 2022-03-27@09:25:14 UTC
Parsed 2 policy rules.
/* Define pool group using two pools */
RULE 'DefineTiers' GROUP POOL 'TIERS' IS 'system' LIMIT(70) THEN 'bronze'
RULE 'Rebalance' MIGRATE FROM POOL 'TIERS' TO POOL 'TIERS' WEIGHT(computeFileHeat(CURRENT_TIMESTAMP-
ACCESS_TIME,xattr('gpfs.FileHeat'),KB_ALLOCATED)) FOR FILESET('object_fileset','obj_CompressionObjectfs') WHERE
NAME LIKE '%.data'
[I] 2022-03-27@09:25:20.056 Directory entries scanned: 24061.
[I] Directories scan: 13814 files, 10119 directories, 128 other objects, 0 'skipped' files and/or errors.
[I] 2022-03-27@09:25:20.488 Parallel-piped sort and policy evaluation. 24061 files scanned.
[I] 2022-03-27@09:25:20.543 Piped sorting and candidate file choosing. 1885 records scanned.
[I] Summary of Rule Applicability and File Choices:
  Rule#      Hit_Cnt      KB_Hit      Chosen      KB_Chosen      KB_Ill      Rule
    0         1885      2862240         0         0         0      RULE 'TIERS'
MIGRATE FROM POOL 'TIERS' WEIGHT(.) TO POOL 'TIERS' FOR FILESET(.) WHERE(.)

[I] Filesystem objects with no applicable rules: 22176.

[I] GPFS Policy Decisions and File Choice Totals:
Chose to migrate 0KB: 0 of 1885 candidates;
[I] File Migrations within Group Pools
  Group      Pool      Files_Out      KB_Out      Files_In      KB_In
  TIERS      system         0         0         0         0
  TIERS      bronze         0         0         0         0
Predicted Data Pool Utilization in KB and %:
Pool_Name      KB_Occupied      KB_Total      Percent_Occupied
bronze          135168      31457280      0.429687500%
system          5234688      52428800      9.984375000%
[root@objectcluster ~]#
```

You can define a LIST ILM policy to query the fileheat of all files if you are interested.

```
[root@objectcluster ~]# cat testheat.pol
define(DISPLAY_NULL,[CASE WHEN ($1) IS NULL THEN ' _NULL_' ELSE varchar($1) END])
rule fh2 list 'fh' weight(FILE_HEAT) show( DISPLAY_NULL(FILE_HEAT) || ' | ' || varchar(file_size) )
[root@objectcluster ~]#
```

Run the LIST policy.

```
[root@objectcluster ~]# mmapplypolicy objectfs -P testheat.pol -L 6 > fh.out
```


Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
[W] Attention: In RULE 'fh' LIST name 'fh' appears but there is no corresponding "EXTERNAL LIST 'fh' EXEC ... OPTS ..." rule to specify a program to process the matching files.
[I] 2022-03-27@10:01:27.467 Directory entries scanned: 24051.
[I] 2022-03-27@10:01:28.257 Parallel-piped sort and policy evaluation. 24051 files scanned.
[I] 2022-03-27@10:01:28.286 Piped sorting and candidate file choosing. 13804 records scanned.
[I] 2022-03-27@10:01:28.543 Policy execution. 13804 files dispatched.
[root@objectcluster ~]#
```

If we query the output file, we will see the heatmap value for each file (this is the weight value).

```
[root@objectcluster ~]# cat fh.out
.
.
<1>
/ibm/objectfs/object_fileset/o/zldevice74/objects/6519/2ab/65dd764401fb3cbad8496a91ce20f2ab/1648370448.36994.data
[2022-03-27@08:40:48 160 160 20480 system 2022-03-27@08:40:48 24 object_fileset] RULE 'fh' LIST 'fh'
WEIGHT(0.000000) SHOW( +0.000000000000000E+000 | 20480)
<1>
/ibm/objectfs/object_fileset/o/zldevice119/objects/1740/55f/1b32606355ba618abf6a9542f021855f/1648370448.74201.data
[2022-03-27@08:40:48 160 160 20480 system 2022-03-27@08:40:48 24 object_fileset] RULE 'fh' LIST 'fh'
WEIGHT(0.000000) SHOW( +0.000000000000000E+000 | 20480)
<1> /ibm/objectfs/object_fileset/o/zldevice46/objects/9141/.lock [2022-03-24@07:03:41 160 160 0 system 2022-
03-24@07:03:41 0 object_fileset] RULE 'fh' LIST 'fh' WEIGHT(0.000000) SHOW( +0.000000000000000E+000 | 0)
<1> /ibm/objectfs/object_fileset/o/zldevice46/objects/9141/hashe.invalid [2022-03-25@20:24:26 160 160 0
system 2022-03-27@07:24:22 0 object_fileset] RULE 'fh' LIST 'fh' WEIGHT(0.000000) SHOW( +0.000000000000000E+000 | 0)
<1>
/ibm/objectfs/object_fileset/o/zldevice54/objects/12041/969/bc2659c98cba07f6399929262b10c969/1648370448.95369.data
[2022-03-27@08:40:48 160 160 20480 system 2022-03-27@09:27:07 24 object_fileset] RULE 'fh' LIST 'fh'
WEIGHT(0.833333) SHOW( +8.333333333333333E-001 | 20480)
<1>
/ibm/objectfs/object_fileset/o/zldevice64/objects/7441/fd6/744421ee1f2ddef42a0d1c5f1594dfd6/1648370415.45968.data
[2022-03-27@08:40:15 160 160 20480 system 2022-03-27@08:40:15 24 object_fileset] RULE 'fh' LIST 'fh'
WEIGHT(0.000000) SHOW( +0.000000000000000E+000 | 20480)
<1>
/ibm/objectfs/object_fileset/o/zldevice1/objects/8212/b89/80534db58445f9ed8791ae9f584efb89/1648370416.45306.data
[2022-03-27@08:40:16 160 160 20480 system 2022-03-27@08:40:16 24 object_fileset] RULE 'fh' LIST 'fh'
WEIGHT(0.000000) SHOW( +0.000000000000000E+000 | 20480)
.
.
.
```

That concludes this section on Object tiering using Spectrum Scale file heatmap attributes. This functionality is intended to ensure optimal performance for hot objects and cost savings in that we can offload cold objects to more cost-effective storage pools.

Spectrum Scale Object Performance Tuning

By default, the IBM Spectrum Scale installation sets the number of workers for object services to a low number. You most likely want to adjust this upwards for production workloads. Refer to the following URL for guidance.

<https://www.ibm.com/docs/en/spectrum-scale/5.1.3?topic=service-performance-tuning-object-services>

In addition to the above Object tuning, you might want to consider the following tunables. These can be set with mmchconfig (See here for more information <https://www.ibm.com/docs/en/spectrum-scale/5.1.3?topic=configuring-parameters-performance-tuning-optimization>):

- Ensure you are using jumbo frames if your network supports it
- Set the pagepool size to at least 8GB or more if you can afford too
- Set MaxMBPS to twice the I/O throughput the node can support
- Set prefetchPct to 30
- Set maxFilestoCache to 128K
- Set maxStatCache to 128K
- Set ignorePrefetchLUNCount to yes
- Set workerThreads to 1024

- Use either a 8MB or 16MB Blocksize for your Object filesystem
- Verify `/proc/sys/net/ipv4/tcp_window_scaling` is enabled
- Tune the TCP window settings by adding these lines to the `/etc/sysctl.conf` (you need to issue the `sysctl -p /etc/sysctl.conf` file and recycle Spectrum Scale)

```
# increase Linux TCP buffer limits
net.core.rmem_max = 8388608
net.core.wmem_max = 8388608
# increase default and maximum Linux TCP buffer sizes
net.ipv4.tcp_rmem = 4096 262144 8388608
net.ipv4.tcp_wmem = 4096 262144 8388608
```

- Use smaller luns to make up your Object filesystem to ensure you take advantage of host per disk I/O queues – more luns equates to more queues which will yield better performance
- Ensure lun queue depths are set according to your Storage Vendor's recommendation

Spectrum Scale Object Backup and Restore

A detailed explanation of how-to backup and restore your Object installation is provided here <https://www.ibm.com/docs/en/spectrum-scale/5.1.3?topic=storage-backing-up-object>. There are a few other considerations though. Firstly, the above procedure does not backup the Swift Keystone database and Swift configuration files and secondly, if you have multiple Swift Storage Policies you will need to ensure you back them up as well (as they are implemented as separate filesets).

Let's take a look at what needs to be backed up in our example. We have the Swift Keystone database located in the `/ibm/cesSharedRoot/keystone` directory and we have our Object filesystem with our account, container and object rings (a separate Object ring for each Storage Policy). We also need to backup the `/etc/swift` directory which contains our Swift builder and ring files. If your main concern is just operational recovery, then the link provided above should suffice though you would need to ensure that you also perhaps include a consistent backup of the PostgreSQL keystone database. You can do this by shutting down Object services and taking a snapshot or manually backing up the `/ibm/cesSharedRoot/keystone` directory. If you don't make use of an external backup application (e.g. IBM Spectrum Protect), then the use of Spectrum Scale snapshots is a good alternative.

To cater for a complete recovery of your Object implementation to a new cluster is outside the scope of this document (for example in the case of a catastrophic failure or site loss). However, you would need to perform similar functions as described in the links below:

- Object Information that is required for a complete rebuild

<https://www.ibm.com/docs/en/spectrum-scale/5.0.5?topic=recovery-object-data-required-protocols-cluster-dr>

- Failover Steps (or rebuild steps to a new cluster) to recover your Object configuration

<https://www.ibm.com/docs/en/spectrum-scale/5.0.5?topic=odrpdc-failover-steps-object-configuration-if-you-are-using-local-authentication-object>

Using AFM with Spectrum Scale Object

A good example of how to use AFM with Spectrum Scale Object can be found here <https://www.ibm.com/docs/en/spectrum-scale/5.1.3?topic=scale-using-afm-object>. Essentially the example presented in the above link entails replicating data from a home cluster to a cache cluster (using Independent Writer AFM mode) which has smaller capacity and this cache cluster is used primarily to serve Object clients. Data is replicated to a home cluster with much larger capacity. Failover from the cache to the home cluster is detailed so that Object clients can still get access to their data and Failback from the home to the cache cluster is also discussed. The example includes all the steps to synchronize the Swift configuration between the two sites so is a good read to understand exactly what this entails and how it is implemented. Another key consideration would be to ensure that you make use of an external keystone server to facilitate the failover/failback process.

Summary

This document has highlighted the steps required to enable an S3 compliant Object Store in IBM Spectrum Scale. We have also covered the steps required to enable Virtual-hosted style bucket addressing which will be the AWS S3 standard going forward. Virtual-hosted style bucket addressing requires wildcard DNS support (which we setup using dnsmasq). This document also covered the installation of HAProxy to enable HTTPS access to our Object Store via SSL termination. Additional Spectrum Scale features like Object compression, Object ILM and Object tiering were also covered to showcase the benefits of using IBM Spectrum Scale for your Object storage requirements compared to other offerings on the market.

Appendix A: testobj.sh

This script is used to perform testing of your Object storage endpoint. A full description of what this script does is detailed on page 65.

```
[root@scalecluster ~]# cat testobj.sh
#####
#
# Script to PUT/GET some files to the designated object store using s3cmd and
# measure the time taken to do so
#
# Written by Nishaan Docrat, 2022
#
# This program is distributed in the hope that it will be useful, but WITHOUT ANY
# WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
# PARTICULAR PURPOSE. See the GNU General Public License for more details.
#
#####

#####
#
# VARIABLES
#
#####

S3CMD=/usr/local/bin/s3cmd
FILES="file1 file2 file3 file4 file5"

#####
#
# FUNCTIONS
#
#####

Help()
{
    # Display Help
    echo "Script to PUT/GET multiples files to an object store using s3cmd."
    echo
    echo "Syntax: testobj.sh [-d]"
    echo "options:"
    echo "-d      Working directory."
    echo "-s      Size of source files in MB."
    echo "-c      Multipart chunk size in MB."
    echo
    exit 1
}

#####
#
# MAIN
#
#####

while getopts d:s:c: flag
do
    case $flag in
        d) WORKING_DIR=${OPTARG};;
        s) FILESIZE=${OPTARG};;
        c) CHUNKSIZE=${OPTARG};;
        *) Help
           exit;;
    esac
done

echo "testobj.sh run on `date`"
echo

# Check to see if s3cmd is installed
if [ -e $S3CMD ]
then
    echo "Checking to see if s3cmd is installed...PASS"
    echo
else
    echo "ERROR: Could not find $S3CMD. Exiting.."
    exit 1
fi

# Check to see if source file size is specified or not
if [ -z $FILESIZE ]
then
    FILESIZE=500
fi
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
# Work out total size
TOTALSIZE=$((FILESIZE * 5))

# Check to see if working directory is set or not
if [ -z $WORKING_DIR ]
then
    WORKING_DIR=$PWD
fi

# Check to see if enough free space exists for out testing in the specified working directory
FREESPACE=`df --block-size=M $WORKING_DIR | tail -1 | awk '{print $4}' | sed "s/[TMGK]//g"`
if [ $FREESPACE -gt $TOTALSIZE ]
then
    echo "Checking to see if at least ${TOTALSIZE}MB is available in ${WORKING_DIR}... PASS"
    echo
else
    echo "ERROR: Not enough free space in $WORKING_DIR.. Exiting.."
    exit 1
fi

# Create source directory for copy if it doesn't exist
if [ -e $WORKING_DIR/testobjfiles ]
then
    echo "Source directory $WORKING_DIR already exists... PASS"
    echo
else
    mkdir $WORKING_DIR/testobjfiles
    if [ $? -eq 0 ]
    then
        echo "Created source directory... PASS"
        echo
    else
        echo "ERROR: Could not create source directory... Exiting"
        exit 1
    fi
fi

# Creating testfiles
for i in $FILES
do
    echo
    dd if=/dev/zero of=${WORKING_DIR}/testobjfiles/${i} bs=1M count=${FILESIZE}
    if [ $? -eq 0 ]
    then
        echo
        echo "Creating source file ${i}... PASS"
    else
        echo "ERROR: Could not create source file ${i}... Exiting"
        exit 1
    fi
done

# Check to see if s3cmd is configured correctly to object endpoint
echo
$S3CMD ls > /dev/null 2>&1
if [ $? -eq 0 ]
then
    echo
    echo "s3cmd is able to connect to the object endpoint... PASS"
else
    echo "ERROR: s3cmd cannot connect to the object endpoint. Make sure s3cmd is configured correctly... Exiting"
    exit 1
fi

# Make two containers
CONTAINER1="testobj-`echo $RANDOM | md5sum | head -c8`"
CONTAINER2="testobj-`echo $RANDOM | md5sum | head -c8`"

$S3CMD mb s3://${CONTAINER1} > /dev/null 2>&1
if [ $? -eq 0 ]
then
    echo
    echo "Created $CONTAINER1 successfully... PASS"
else
    echo
    echo "ERROR: Could not create container ${CONTAINER1}... Exiting"
    exit 1
fi

$S3CMD mb s3://${CONTAINER2} > /dev/null 2>&1
if [ $? -eq 0 ]
then
    echo
    echo "Created $CONTAINER2 successfully... PASS"
else
    echo
    echo "ERROR: Could not create container ${CONTAINER2}... Exiting"
    exit 1
fi
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
# Check to see if multipart chunk size is set or not
if [ -z $CHUNKSIZE ]
then
    echo
    echo "Multipart chunk size is not set... Using default 15MB...   PASS"
    CHUNKSIZE=15
fi

# PUT source files into CONTAINER1

echo
echo
echo
echo
echo "Starting upload of source files using a chunk size of ${CHUNKSIZE}MB for multipart uploads...   PASS"
echo
echo
echo
PUTSTART=`date +%s`
$S3CMD put --recursive --multipart-chunk-size-mb=${CHUNKSIZE} ${WORKING_DIR}/testobjfiles/ s3://${CONTAINER1}
if [ $? -eq 0 ]
then
    echo
    echo "PUT 5 files of ${FILESIZE}MB each...   PASS"
else
    echo
    echo "ERROR: Could not upload source files... Exiting"
fi
echo
echo
PUTEND=`date +%s`

# Calculate throughput for PUT

PUTRUNTIME=$((PUTEND-PUTSTART))
PUTTHROUGHPUT=$((TOTALSIZE/PUTRUNTIME))
echo "Runtime for PUT is $PUTRUNTIME secondsi...   "
echo "Throughput for PUT is $PUTTHROUGHPUT MB/s...   "

# CP files from CONTAINER1 to CONTAINER2

echo
echo
echo
echo
echo "Starting copy of source files from container $CONTAINER1 to container $CONTAINER2...   PASS"
echo
echo
echo
CPSTART=`date +%s`
$S3CMD cp --recursive s3://${CONTAINER1} s3://${CONTAINER2}
if [ $? -eq 0 ]
then
    echo
    echo "Copy 5 files from container $CONTAINER1 to container $CONTAINER2...   PASS"
else
    echo
    echo "ERROR: Could not copy files from container ${CONTAINER1} to container ${CONTAINER2}... Exiting"
fi
echo
echo
CPEND=`date +%s`

# Calculate throughput for PUT

CPRUNTIME=$((CPEND-CPSTART))
CPTHROUGHPUT=$((TOTALSIZE/CPRUNTIME))
echo "Runtime for CP is $CPRUNTIME secondsi...   "
echo "Throughput for CP is $CPTHROUGHPUT MB/s...   "

# GET files from CONTAINER1 to working directory

echo
echo
echo
echo
echo "Starting GET source files from container $CONTAINER1 to ${WORKING_DIR}/testobjfiles...   PASS"
echo
echo
echo
GETSTART=`date +%s`
$S3CMD get --recursive --force s3://${CONTAINER1} ${WORKING_DIR}/testobjfiles
if [ $? -eq 0 ]
then
    echo "GET of 5 files from container $CONTAINER1 to ${WORKING_DIR}/testobjfiles...   PASS"
    echo
    echo "Listing retrieved files from container $CONTAINER"
    echo
    ls -l $WORKING_DIR/testobjfiles
    echo
else
    echo
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
        echo "ERROR: Could not copy files from container ${CONTAINER1} to ${WORKING_DIR}/testobjfiles... Exiting"
    fi
    echo
    echo
    GETEND=`date +%s`

    # Calculate throughput for PUT

    GETRUNTIME=$((GETEND-GETSTART))
    GETTHROUGHPUT=$((TOTALSIZE/GETRUNTIME))
    echo "Runtime for GET is $GETRUNTIME secondsi...      "
    echo "Throughput for GET is $GETTHROUGHPUT MB/s...    "
    echo

    # Cleaning up our working directory
    rm ${WORKING_DIR}/testobjfiles/file*
    if [ $? -eq 0 ]
    then
        echo
        echo "Cleaned up ${WORKING_DIR}/testobjfiles      PASS"
    else
        echo
        echo "ERROR: Cleaning up working director ${WORKING_DIR}/testobjfiles... EXITING"
        exit 1
    fi

    # Removing testobjfiles from our working directory
    rm -fd ${WORKING_DIR}/testobjfiles
    if [ $? -eq 0 ]
    then
        echo
        echo "Removed testobjfiles from our working directory      PASS"
    else
        echo
        echo "ERROR: Removing ${WORKING_DIR}/testobjfiles... EXITING"
        exit 1
    fi

    # Cleanup containers and objects

    $S3CMD rb --recursive --force s3://${CONTAINER1} > /dev/null 2>&1
    if [ $? -eq 0 ]
    then
        echo
        echo "Deleted ${CONTAINER1} and all its contents successfully...  PASS"
    else
        echo
        echo "ERROR: Could not delete container ${CONTAINER1} and its contents... Exiting"
        exit 1
    fi

    $S3CMD rb --recursive --force s3://${CONTAINER2} > /dev/null 2>&1
    if [ $? -eq 0 ]
    then
        echo
        echo "Deleted ${CONTAINER2} and all its contents successfully...  PASS"
    else
        echo
        echo "ERROR: Could not delete container ${CONTAINER2} and its contents... Exiting"
        exit 1
    fi

    # Write to output file the results
    OFILE=${WORKING_DIR}/testobj-`date "+%Y-%m-%d-%H-%M-%S"`.log
    echo
    touch $OFILE > /dev/null 2>&1
    if [ $? -eq 0 ]
    then
        echo
        echo "Created output file ${OFILE}...      PASS"
        echo
        echo "Writing results to $OFILE...  PASS"
        echo >> $OFILE
        echo "*****" >> $OFILE
        echo "testobj.sh run on `date`" >> $OFILE
        echo >> $OFILE
        echo "PUT runtime was ${PUTRUNTIME}s and throughput was ${PUTTHROUGHPUT}MB/s" >> $OFILE
        echo "CP runtime was ${CPRUNTIME}s and throughput was ${CPTHROUGHPUT}MB/s" >> $OFILE
        echo "GET runtime was ${GETRUNTIME}s and throughput was ${GETTHROUGHPUT}MB/s" >> $OFILE
        echo >> $OFILE
        echo "*****" >> $OFILE
    else
        echo
        echo "ERROR: Could not create output file ${OFILE}... Exiting"
        exit 1
    fi

    echo
    echo "testobj.sh completed on `date`"
```

Setting up an S3 Compliant Object Store in IBM Spectrum Scale with Object Compression, ILM and Tiering

```
echo
```

```
[root@scalecluster ~]#
```