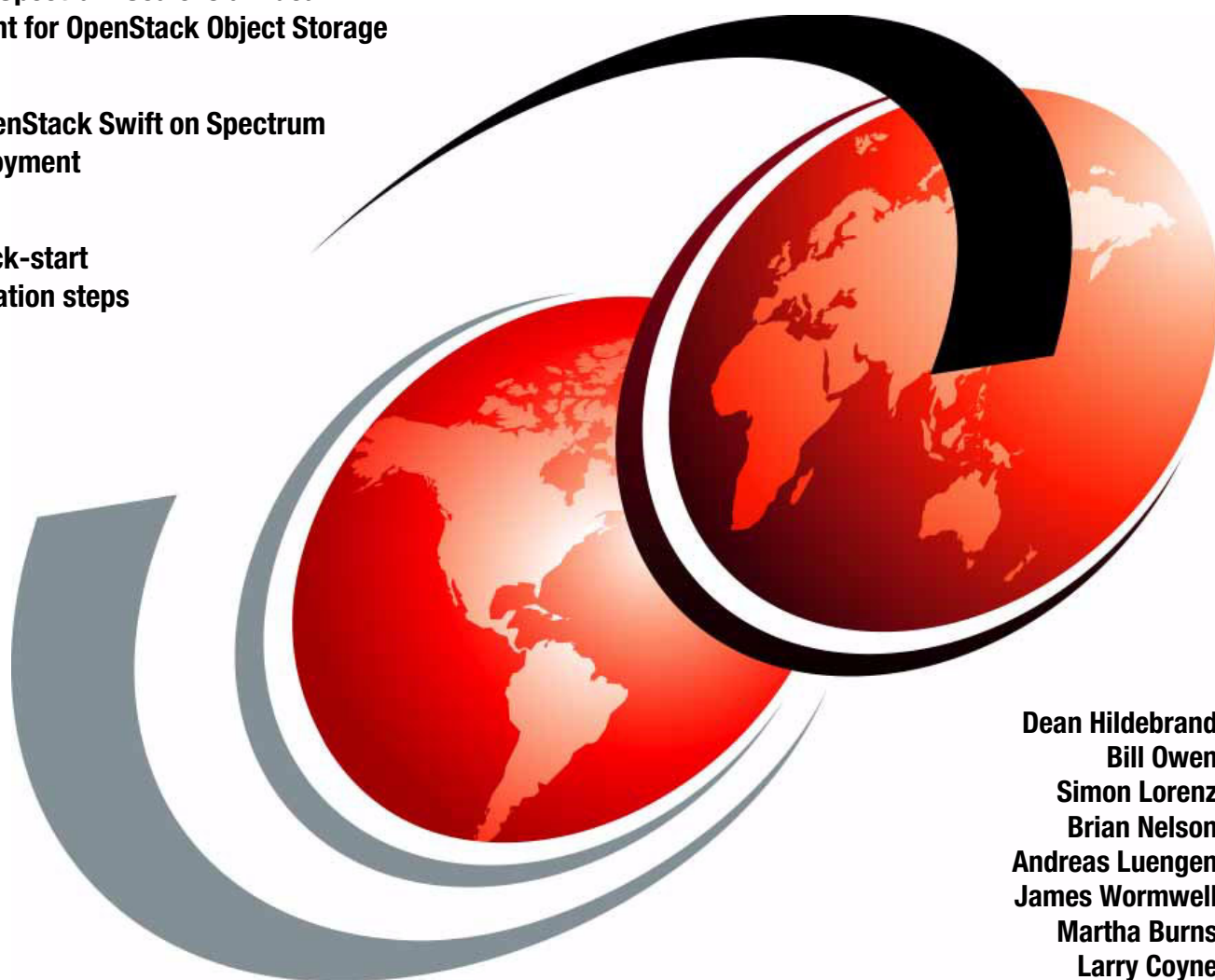


A Deployment Guide for IBM Spectrum Scale Object

Learn why Spectrum Scale is an ideal environment for OpenStack Object Storage

Plan an OpenStack Swift on Spectrum Scale deployment

Follow quick-start implementation steps



Dean Hildebrand
Bill Owen
Simon Lorenz
Brian Nelson
Andreas Luengen
James Wormwell
Martha Burns
Larry Coyne



International Technical Support Organization

A Deployment Guide for IBM Spectrum Scale Object

March 2015

Note: Before using this information and the product it supports, read the information in “Notices” on page v.

Third Edition (March 2015)

This edition applies to Version 4, Release 1, of IBM General Parallel File System.

© Copyright International Business Machines Corporation 2014, 2015. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	v
Trademarks	vi
Preface	vii
Authors	vii
Now you can become a published author, too!	x
Comments welcome	x
Stay connected to IBM Redbooks	x
Summary of changes	xi
September 2016, update	xi
March 2015, Third Edition	xi
December 2014, Second Edition	xi
Chapter 1. Spectrum Scale Object	1
1.1 Introduction	2
1.2 Assumptions	3
1.3 Key concepts and terminology	4
1.4 Introduction to Spectrum Scale Object	7
1.4.1 Spectrum Scale features	7
1.4.2 Use cases	7
1.4.3 High-level architecture	8
1.4.4 Benefits	10
1.4.5 Detailed architecture	10
Chapter 2. Planning for a Spectrum Scale Object deployment	13
2.1 Spectrum Scale architecture	14
2.1.1 Networking	14
2.1.2 Placement of Object Store components in Spectrum Scale	14
2.1.3 Authentication	15
2.1.4 Data protection	15
2.1.5 Performance	15
2.1.6 High availability	16
2.1.7 Spectrum Scale and Swift interaction	17
2.1.8 Administration tools	17
2.2 Tested configurations	17
Chapter 3. Spectrum Scale Object configuration overview	19
3.1 Swift replication	20
3.2 Swift rings	20
3.3 Swift services	20
Chapter 4. Spectrum Scale Object installation.	23
4.1 Installation overview	24
4.2 Installation prerequisites	25
4.3 Spectrum Scale installation and configuration	25
4.4 Object software installation and configuration	27
4.5 Postinstallation	33
4.5.1 Verifying the installation	33

4.5.2 Tuning considerations	33
4.6 Removing the installation	33
Chapter 5. System administration considerations.	35
5.1 Managing Swift services	36
5.2 Adding a GPFS Object Node	36
5.3 Removing a GPFS Object Node	37
5.4 Account and container data placement policies	38
5.5 Process monitoring	39
5.6 Security-Enhanced Linux considerations	40
5.7 Port security considerations	40
5.8 Configuring rsync to limit host access	41
5.9 Virtual Network Computing port conflict	41
5.10 Software maintenance	41
Chapter 6. Swift feature overview	43
Chapter 7. Backup and restore	45
7.1 GPFS independent filesets	46
7.2 Snapshots	46
7.3 Backing up and restoring the object store	46
7.3.1 Backup procedure	47
7.3.2 Restore procedure	49
7.3.3 Automating the backup and restore procedures	52
Chapter 8. Summary	55
8.1 Future investigation	56
8.2 Conclusion	56
Appendix A. Additional material	57
Locating the web material	57
Using the web material	57
Downloading and extracting the web material	57
Related publications	59
IBM Redbooks	59
Other publications	59
Online resources and references	59
Help from IBM	62

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.


Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at <http://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

AIX®
GPFS™
IBM®
IBM Spectrum™

IBM Spectrum Protect™
IBM Spectrum Scale™
Redbooks®
Redpaper™

Redbooks (logo) ®
Storwize®
Tivoli®

The following terms are trademarks of other companies:

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Preface

Because of the explosion of unstructured data that is generated by individuals and organizations, a new storage paradigm called “Object Storage” has been developed. Object Storage stores data in a flat namespace that scales to trillions of objects. The design of object storage also simplifies how users access data, supporting new types of applications and allowing users to access data by various methods, including mobile devices and web applications. Data distribution and management are also simplified, allowing greater collaboration across the globe.

OpenStack Swift is an emerging open source object storage software platform that is widely used for cloud storage. IBM® Spectrum Scale™ (based on IBM General Parallel File System (GPFS™) technology, which also is known formerly as code name Elastic Storage) is a high-performance and proven product that is used to store data for thousands of mission-critical commercial installations worldwide. (Throughout this IBM Redpaper™ publication, Spectrum Scale is used to refer to GPFS V4.1.)

Spectrum Scale also automates common storage management tasks, such as tiering and archiving at scale. Together, Spectrum Scale and OpenStack Swift provide an enterprise-class object storage solution that efficiently stores, distributes, and retains critical data.

This paper provides instructions about how to set up and configure Swift with Spectrum Scale. It also provides an initial set of preferred practices to ensure optimal performance and reliability.

The goal of this paper is to describe the benefits of using Spectrum Scale as the underlying file system with OpenStack Swift, guide an administrator through the installation and configuration of Spectrum Scale Object, and describe the general set of configurations and scenarios that have been validated. It is intended for administrators who are familiar with Spectrum Scale and OpenStack Swift components.

Authors

This paper was produced by a team of specialists from around the world working with the International Technical Support Organization, Tucson Center.

Dean Hildebrand is a Master Inventor and the Manager of the Cloud Storage Software Research group at the IBM Almaden Research Center and a recognized expert in the field of distributed and parallel file systems. He has authored numerous scientific publications, created over 24 patents, and chaired and sat on the program committee of numerous conferences. Dr. Hildebrand pioneered pNFS, demonstrating the feasibility of providing standard and scalable access to any parallel file system. He received a B.Sc. degree in Computer Science from the University of British Columbia in 1998 and M.S. and PhD. degrees in Computer Science from the University of Michigan in 2003 and 2007.

Bill Owen is a Senior Engineer with the IBM Spectrum™ Scale development team. He is responsible for the integration of OpenStack with Spectrum Scale, focusing on the Swift object, Cinder block, and Manila file storage components of OpenStack. He has worked in various development roles within IBM for over 15 years. Before joining IBM, Bill developed and deployed grid management systems for electric utilities. Bill holds B.Sc. and M.S. degrees in Electrical Engineering from New Mexico State University.

Simon Lorenz is an IT Architect in IBM Research and Development in Mainz, Germany. He joined IBM Germany in 1993 and worked on productivity and manufacturing quality improvements within IBM Disk Drive Manufacturing Management software. During international assignments, he helped to improve fully automated chip factories in the US and Asia. Simon has held various positions within IBM Research and Development. Since 2009, he has worked on Storage Systems Management software and has been responsible for subcomponents, such as system health reporting, cluster configuration management, and recovery. Simon joined the IBM General Parallel File System development team in 2014.

Brian Nelson is a Software Engineer with the IBM Spectrum Scale development team. He is responsible for providing OpenStack Swift integration into Spectrum Scale and Elastic Storage Server (ESS). He has worked within IBM for over 18 years on projects, such as IBM AIX®, IBM Director, and PowerVC. Brian holds a B.Sc. in Computer Science from Trinity University.

Andreas Luengen is a Master Certified IT specialist currently working in the Software Defined Systems (SDS) development team in Mainz, Germany. He joined IBM in 1999 and worked for over 10 years in Advanced Technical Support for mid-range Storage support before joining the NAS development team that introduced SoFS, Scale Out Network Attached Storage, IFS, and Elastic Storage. In his development role, he provided documentation for service offerings and products such as SoFS and Scale Out Network Attached Storage/V7000U before taking over component ownership responsibilities focusing on backup/restore and space management (ILM/HSM) of SoNAS/V7000U. Andreas co-authored *Configuration and Tuning GPFS for Digital Media Environments*, SG24-6700. He holds a degree in Electrical Engineering from the University of Hannover, Germany.

James Wormwell is a Software Engineer with the IBM Spectrum Scale protocols development team. He is responsible for automating the installation of Object and File protocol software in a Spectrum Scale environment. He has worked within IBM for over 3 years. James previously held roles within the IBM Storwize® development, build, and test teams. James holds a B.Sc degree in Computer Science from the University of Salford.

Martha Burns is an Information Development planner and writer for the IBM User Technologies organization in Poughkeepsie, NY. She has been a technical writer, programmer, and communications specialist for over 30 years. Martha has written customer documentation for the IBM clustered-server and high-performance computing software products since 1996, focusing on GPFS and Spectrum Scale since 2009. She holds a B.A. degree in English Language and Literature from the College of William and Mary and an M.S. degree in Technical Writing from Rensselaer Polytechnic Institute.

Larry Coyne is a Project Leader at the International Technical Support Organization (ITSO), Tucson, Arizona, center. He has 33 years of IBM experience, with 23 years in IBM storage software management. He holds degrees in Software Engineering from the University of Texas at El Paso, and Project Management from George Washington University. His areas of expertise include client relationship management, quality assurance, development management, and support management for IBM Tivoli® Storage Software.

Thanks to the following people for their contributions to this project:

Montserrat Ariday
Balderas Alba
Juergen Beicht
April Brown
Steve Buller
Ulrich Busch
Patrick Byrne
Doris Conti
Fiona Crowther
Steve Delillo
Mathias Dietz
John Dorfner
Steve Duersch
Tobias Fleming
Steven Frankl
Julian Cachua Fruchier
Sanjay Gandhi
Michael Garwood
Lyle Gayne
Deepak Ghuge
Imogen Gough
Zi Qiang (John) Gu
David Ibarra
Karsten Jancke
Radha Kandadai
Ross Keeping
Werner Kuehn
John Langlois
Christina Lara
Chen Lei
Kerry McLaughlin
Varun Mittal
John T. Olson
Sandeep Ramesh Patil
Jose Perez
Bonnie Pulver
Krystal Rothaupt
Gautam Shah
Joseph Taylor
Michael L. Taylor
IBM Systems

Renu Tewari
IBM Research - Parallel File Systems

Patrik Hysky
International Technical Support Organization, Poughkeepsie Center

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our papers to be as helpful as possible. Send us your comments about this paper or other IBM Redbooks® publications in one of the following ways:

- ▶ Use the online **Contact us** review Redbooks form found at:

ibm.com/redbooks

- ▶ Send your comments in an email to:

redbooks@us.ibm.com

- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- ▶ Find us on Facebook:

<http://www.facebook.com/IBMRedbooks>

- ▶ Follow us on Twitter:

<http://twitter.com/ibmredbooks>

- ▶ Look for us on LinkedIn:

<http://www.linkedin.com/groups?home=&gid=2130806>

- ▶ Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

<https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm>

- ▶ Stay current on recent Redbooks publications with RSS Feeds:

<http://www.redbooks.ibm.com/rss.html>

Summary of changes

This section describes the technical changes that are made in this edition of the paper and in previous editions. This edition might also include minor corrections and editorial changes that are not identified.

Summary of Changes
for A Deployment Guide for IBM Spectrum Scale Object
as created or updated on September 13, 2016.

September 2016, update

Updated the Redpaper with the following note.

Note: The installation and maintenance procedures described in Chapter 4 and 5 of this Redpaper are not compatible with the Spectrum Scale protocol installation toolkit, provided in Spectrum Scale 4.1.1 and later versions.

The steps in this paper are valid, and should be only be executed on Spectrum Scale nodes that are not designed as Cluster Export Service (CES) protocol nodes.

March 2015, Third Edition

This revision reflects the addition, deletion, or modification of new and changed information that is described below.

Changed information

- ▶ Added the new IBM Spectrum Scale product name (based upon IBM General Parallel File System (GPFS), which also is known formerly as code name Elastic Storage)
- ▶ Added the new IBM Spectrum Protect™ product name (based on IBM Tivoli Storage Manager)

December 2014, Second Edition

This revision reflects the addition, deletion, or modification of new and changed information that is described below.

New information

- ▶ Introduced a new automated installation procedure using the Chef configuration management toolset
- ▶ Described in more detail the steps that are required to back up your object store configuration and data to protect against disaster scenarios and how to recover from that saved data

Changed information

- ▶ Added numerous clarifications and tips that we added based on our own and customer experience using Swift in a GPFS environment
- ▶ Removed the installation appendixes and made installation information available for download at the following website:

<ftp://www.redbooks.ibm.com/redbooks/REDP5113>

For more information about requirements for the download, see Appendix A, “Additional material” on page 57.



Spectrum Scale Object

This chapter provides an introduction to Spectrum Scale Object, business problems that are solved with Spectrum Scale, a high-level architecture, and the scope and intent of this paper.

1.1 Introduction

Data centers are currently struggling to efficiently and cost-effectively store and manage vast amounts of data. The increasing number of application domains, such as analytics, online transaction processing (OLTP), and high-performance computing (HPC), have created silos of storage within data centers. With each new application, a new storage system can be required, forcing system administrators to become experts in numerous storage management tools.

In addition, the set of applications that includes mobile and web-based applications, archiving, backup, and cloud storage has recently created yet another type of storage system for the system administrator to manage: Object Storage. With Object Storage, data is accessed by using a unique identifier over a simple RESTful HTTP interface, objects cannot be updated after they are created (although they can be replaced, versioned, or removed), and in many cases the objects are accessible in an eventually consistent manner. These types of semantics are ideal for images, videos, text documents, virtual machine (VM) images, and other similar files.

The OpenStack open source Object Storage project, which is known as *OpenStack Swift*, uses the Swift API (and also provides Amazon Simple Storage Service [S3] API emulation) to store data in a distributed storage system. For more information, see Amazon Simple Storage Services (S3):

<http://aws.amazon.com/documentation/s3/>

Note: For more information about this project and other OpenStack open source projects, which allow an organization to deploy infrastructure as a service (IaaS) solutions, see this website:

<http://www.openstack.org>

OpenStack Swift is emerging as a dominant object storage solution due to its extreme scalability, extensibility, and resilience. Despite its benefits, however, OpenStack Swift still follows the model of deploying new storage systems for new application domains.

Spectrum Scale Object, the combination of Spectrum Scale and Swift, aims to change this model by consolidating File and Object under a single shared storage infrastructure. The global namespace eliminates the physical client-to-server mappings and makes this an ideal platform to perform common storage management tasks, such as automated storage tiering and user transparent data migration. Spectrum Scale Object simplifies data management even further by creating a flat namespace and eliminating the hassle of organizing data in a hierarchical namespace.

This new version of this paper introduces a new automated installation procedure that uses the Chef configuration management toolset. This paper describes in more detail the steps that are required to back up your object store configuration and data to protect against disaster scenarios, and how to recover from that saved data. Finally, we have added numerous clarifications and tips based on our own and customer experience using Swift in a Spectrum Scale environment.

Note: The installation procedure and sample scripts that are provided with this paper are valid for this paper only. Command names and functions do not reflect the command names and functions that will be available in future product versions.

This paper describes the following topics:

- ▶ How to take advantage of the benefits of Spectrum Scale when building an object storage solution
- ▶ How to install OpenStack Swift on Spectrum Scale, providing preferred practices for configuring and tuning both Swift and Spectrum Scale

Note: When using Spectrum Scale Object in a production environment, it is important to follow the instructions in this paper to maintain the integrity of the solution and the data.

- ▶ The Swift and Spectrum Scale features that have been tested in our labs as part of Spectrum Scale Object

Questions and support: If you have questions or comments regarding the information in this document, contact IBM at gpfs@us.ibm.com. For current Spectrum Scale (based on General Parallel File System) product support information or to open a new service request, contact IBM support at the following website:

<http://www.ibm.com/support>

1.2 Assumptions

The goal of this paper is to describe the benefits of using Spectrum Scale as the underlying file system with OpenStack Swift, guide the administrator through the installation and configuration of Spectrum Scale Object, and describe the general set of configurations and scenarios that have been validated.

The following assumptions are made:

- ▶ The reader is already familiar with OpenStack Swift components. The paper covers high-level concepts only to establish the required context for follow-on topics. For further details, see the resources that are listed in “Related publications” on page 59 or the respective product documentation.
- ▶ The reader is already familiar with Spectrum Scale because this paper is not intended to serve as a comprehensive Spectrum Scale overview. For a comprehensive understanding of Spectrum Scale, see “Related publications” on page 59.
- ▶ The paper focuses on the benefits of Spectrum Scale Object and does not attempt to make any direct comparisons with using Swift with other underlying file systems, such as XFS.

1.3 Key concepts and terminology

This section defines key concepts and terminology that are associated with Spectrum Scale and OpenStack Swift as they relate to Spectrum Scale Object.

Spectrum Scale

The following terms relate to Spectrum Scale:

- ▶ **Spectrum Scale software:** The software that is used to mount and access GPFS file systems. In this paper, it is assumed that this software is installed on every GPFS Object Node and is used as the file system that underpins OpenStack Swift. For more information, see “General Parallel File System” at this website:
http://www.ibm.com/support/knowledgecenter/SSFKCN/gpfs_welcome.html
- ▶ **GPFS Object Node:** The GPFS node where OpenStack object services are running. All client requests to store and retrieve data are made to these nodes.
- ▶ **Elastic Storage Server (ESS):** The ESS combines the Spectrum Scale software with storage enclosures, drives, and networking components to provide a high-capacity, highly scalable storage solution. ESS includes the unique and innovative GPFS Native RAID capability, which provides extreme data integrity and reduced latency with faster rebuild times and enhanced data protection.

Note: In GPFS 4.1 documentation, Elastic Storage Server (ESS) is referred to as GPFS Storage Server (GSS).

See Elastic Storage Server planning and service information at the following URLs:

http://www.ibm.com/support/knowledgecenter/POWER8/p8ehc/p8ehc_storage_landing.htm

http://www.ibm.com/common/ssi/cgi-bin/ssialias?subtype=SP&infotype=PM&appname=S TGE_DC_ZQ_USEN&htmlfid=DCD12377USEN&attachment=DCD12377USEN.PDF#loaded

See GPFS Native RAID information at the following URLs:

http://www.ibm.com/support/knowledgecenter/SSYSP8_2.5.0/sts25_welcome.html

http://www.ibm.com/support/knowledgecenter/SSFKCN_4.1.0/com.ibm.cluster.gpfs.v4r1.gpfs200.doc/blladv_introduction.htm

- ▶ **Network Shared Disk (NSD):** A logical unit number (LUN) that is provided by a storage subsystem (for example, ESS) for use in a GPFS file system.
- ▶ **Storage pool:** A collection of NSDs. GPFS storage pools in Spectrum Scale allow the grouping of storage devices within a file system, based on performance, locality, or reliability characteristics. Storage pools provide a method to partition file system storage to offer several benefits, including improved price-performance, reduced contention on premium resources, seamless archiving, and advanced failure containment.

For example, one pool can be an enterprise-class storage system that uses high-performance flash devices, and another pool might consist of numerous disk controllers that host a large set of economical Serial ATA (SATA) or Near-Line SAS disks.

There are two types of GPFS storage pools: internal and external. For internal storage pools, Spectrum Scale manages the data storage. For external storage pools, however, Spectrum Scale handles the policy processing but leaves data management to an external application, such as Spectrum Protect (formerly IBM Tivoli Storage Manager).

Storage pools are declared as an attribute of the disk and file system.

- ▶ **Fileset:** A subtree of a file system namespace. This document suggests placing the Swift object store in an independent fileset to allow fileset-level administrative operations (for example, snapshot or backup) to apply only to the files in the object store (rather than to the entire file system). The fileset is identified as an attribute of each file and can be specified in a policy to control the initial data placement, migration, and replication of the file's data.
- ▶ **Replication:** A feature of Spectrum Scale that enables multiple copies of data and metadata for failure containment and disaster recovery. Spectrum Scale supports replication of both metadata and data independently and provides fine-grained control of the replication. Replication can incur costs in terms of disk capacity and performance and therefore must be used carefully.
- ▶ **GPFS quotas:** The amount of disk space and the number of inodes that are assigned as upper limits for a specified user, group of users, or fileset. With OpenStack Swift, GPFS user quotas are not used; instead, the system relies on OpenStack Swift quotas to provide a similar type of service. However, GPFS fileset quotas can still be defined (for example, for inodes, to limit the resources that are consumed by the fileset).
- ▶ **GPFS access control lists (ACLs):** Fine-grained access control mechanisms. With OpenStack Swift, GPFS ACLs are not used; instead, the system relies on OpenStack Swift ACLs to set permissions.

OpenStack Swift

The following terms relate to OpenStack Swift:

- ▶ **Account:** The top-level element in the object storage system hierarchy. An account contains a list of the containers in the account. In the OpenStack environment, the term “account” is synonymous with the term “tenant” (as used by Keystone).
- ▶ **Tenant:** See Account.
- ▶ **Container:** The second-level element in the hierarchy (under accounts). A container maintains a list of objects that belong to the container. The account and container provide a namespace for objects, analogous to files in a directory path. Many features, such as ACLs, versioning, and quotas, are controlled at the container level.
- ▶ **Object:** The third-level element in the hierarchy (under containers). An object stores actual data content and metadata that describes the object. In Spectrum Scale, objects are stored as files, and object metadata (shown in Example 1-1) is stored as file extended attributes.

Example 1-1 Object metadata

```
Content-Length = 14540
name = /AUTH_39150cd50dfb47e7be85280735174691/bill/testobj
ETag = 2cd0ae668a585a14e07c2ea4f264d79b
X-Timestamp = 1402267969.60475
Content-Type = application/octet-stream
```

- ▶ **Object metadata:** Key-value pairs that are stored by Swift and by Swift clients on an object through the **POST** command. These key-value pairs are stored as GPFS extended attributes.
- ▶ **Keystone:** The OpenStack service that provides identity, token, catalog, and policy services. In this document, Keystone is used as the Swift authentication service that provides role-based management of Swift accounts and containers. Keystone can be integrated with existing Lightweight Directory Access Protocol (LDAP) and Active Directory (AD) systems, but we have not integrated this function into the automated installer currently.

- ▶ **Proxy service:** The proxy service runs on the GPFS Object Nodes and accepts requests from applications on accounts, containers, or objects, and directs the request to the appropriate service within the cluster.
- ▶ **Object service:** The object service runs on the GPFS Object Nodes and accepts requests from the local Swift Proxy service to store objects in Spectrum Scale.
- ▶ **Virtual devices:** In Swift, devices typically map to physical disks. In Spectrum Scale Object, virtual devices map to a top-level directory in which the object store hierarchy is created. There can be multiple virtual devices, and these are created and managed by Swift services.

Changing the number of virtual devices must be done with care because this will require rebalancing the Swift rings, resulting in data being copied between virtual devices.

- ▶ **Partitions:** Used to group and evenly distribute objects in the file system. The number of partitions is specified during Swift configuration when the Swift rings are created and determines the expected number of objects in each one.

The number of partitions must be chosen carefully because it cannot be easily changed after it is specified.

- ▶ **Pseudo-hierarchical folders and directories:** Objects that are stored in a flat namespace but whose names include one or more forward slash (/) characters to imply levels in a hierarchy. Each forward slash (/) character creates another level in the hierarchy. Pseudo-directories do not affect how objects are placed on disk, but they can be useful when listing objects, for example, within a specific pseudo-directory.
- ▶ **SQLite:** A database in which account and container listings are stored. An account database contains a list of its containers, and a container database contains a list of its associated objects.
- ▶ **Memcached:** A distributed caching service that caches items in memory for fast retrieval. Example items that are cached are account and container database information and authentication tokens.
- ▶ **Swift access control list (ACL):** Controls access to an account or container. Swift provides role-based authentication that is mixed with fine-grained ACLs. Swift ACLs apply to accounts and containers, not to individual objects.

Note: GPFS ACLs are not used with Swift.

- ▶ **Swift quotas:** Allows specification of the amount of disk space or number of objects that can be consumed by either an account (and then all of its containers) or to an individual container. The interaction between Swift Quotas and GPFS quotas are described in more detail in Chapter 6, “Swift feature overview” on page 43.
- ▶ **Swift ring:** Determines the partition in which accounts, containers, and objects will be stored (one ring for each). For accounts and containers, each partition is mapped to a specific GPFS Object Node. For objects, every partition is accessible by every GPFS Object Node.
- ▶ **Controller Node:** The node on which the Keystone service is running. The Controller Node can be one of the GPFS Object Nodes, or it can be a separate node.
- ▶ **Eventual consistency:** All Swift items, which include accounts, containers, and objects, are ensured to *eventually* reach a consistent state, allowing applications to access the most up-to-date versions.

1.4 Introduction to Spectrum Scale Object

This section describes the product features of Spectrum Scale and the use cases, high-level architecture, and benefits of Spectrum Scale Object.

1.4.1 Spectrum Scale features

Spectrum Scale enables virtualization, analytics, file, and object use cases to be unified into a single scale-out data plane for the entire data center. As shown in Figure 1-1, Spectrum Scale can provide a single namespace for all of this data, offering a single point of management. Data can then be tiered in differentiated classes of storage and accessed around the globe, ensuring that data is always available in the correct place at the correct time.

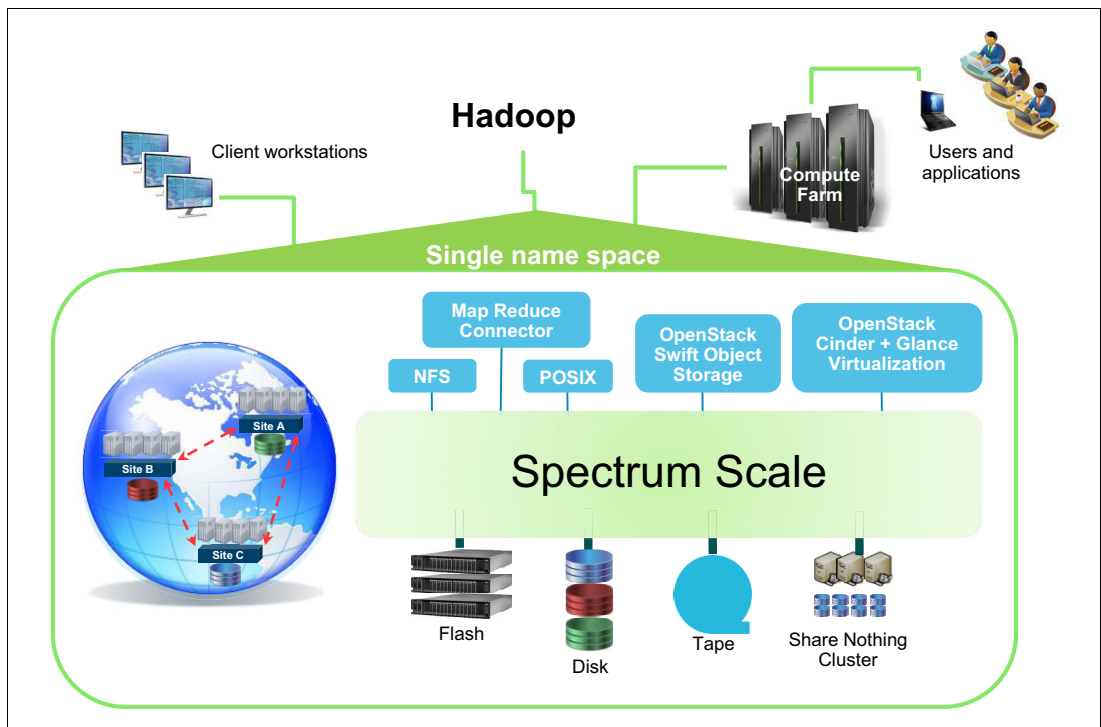


Figure 1-1 Spectrum Scale overview

1.4.2 Use cases

This section describes the types of use cases and the potential users who will benefit from Spectrum Scale Object:

- ▶ Enterprise archiving
- ▶ Backup
- ▶ Hadoop analytics
- ▶ Content storage
- ▶ Content distribution
- ▶ Video production
- ▶ Worldwide collaboration
- ▶ An unstructured data landing zone for the output of sensors, devices, and applications

Although Spectrum Scale enables a single unified namespace for data storage, it allows the different types of data to be physically placed onto different storage devices (GPFS storage pools), which can range from solid-state drives (SSDs) and Flash devices to external disk subsystems, internal disks within servers, and even tape.

Note: Spectrum Scale does not *enforce* a single namespace, and one GPFS cluster can support up to 256 file systems or namespaces, each with different characteristics.

Spectrum Scale Object is targeted toward two types of potential users:

- ▶ Sites that already use Spectrum Scale and are seeking to support Swift in the same data plane as their existing data.
- ▶ Sites that are simply seeking an enterprise-ready, cost-efficient, and high-performing object solution. For this use case, Spectrum Scale initially provides data management for the object store only, but gives users the option to extend to further types of applications as their requirements grow.

Spectrum Scale Object is initially targeting workloads that demand the following capabilities:

- ▶ Enterprise-ready features, such as snapshots and backups to prevent data loss
- ▶ High-throughput access to the objects
- ▶ High-capacity, dense storage
- ▶ Levels of data scaling that can be used in the future

1.4.3 High-level architecture

Spectrum Scale Object combines the benefits of Spectrum Scale with the most widely used open source object store today, OpenStack Swift. Spectrum Scale provides enterprise information lifecycle management (ILM) features. OpenStack Swift provides a robust object layer with an active community that is continuously adding innovative new features. To ensure compatibility with the Swift packages over time, no code changes are required to either Spectrum Scale or Swift to build the solution.

An example of Spectrum Scale Object architecture is shown in Figure 1-2 on page 9. In this example, applications perform RESTful operations (HTTP) to the object store through (at least two, for high availability) Spectrum Scale Object Nodes (also referred to as GPFS Object Nodes in this document). All objects are stored in an Elastic Storage Server (ESS).

Note: In GPFS 4.1 documentation, Elastic Storage Server is referred to as GPFS Storage Server (GSS).

See Elastic Storage Server planning and service information at the following URLs:

http://www.ibm.com/support/knowledgecenter/POWER8/p8ehc/p8ehc_storage_landing.htm

http://www.ibm.com/common/ssi/cgi-bin/ssialias?subtype=SP&infotype=PM&appName=STGE_DC_ZQ_USEN&htmlfid=DCD12377USEN&attachment=DCD12377USEN.PDF#loaded

See GPFS Native RAID information at the following URLs:

http://www.ibm.com/support/knowledgecenter/SSYSP8_2.5.0/sts25_welcome.html

http://www.ibm.com/support/knowledgecenter/SSFKCN_4.1.0/com.ibm.cluster.gpfs.v4r1.gpfs200.doc/b1ladv_introduction.htm

In this example configuration, the ESS Model GL6 solution has redundant servers and uses GPFS Native RAID (8+2P) across 696 4 TB drives. This configuration offers 2 PB of accessible storage capacity with 174 disks per server and 80% storage capacity efficiency.

Applications can access objects by using the full 40 Gbps of the available network bandwidth (in fact with storage network bandwidth to spare).

Note: This solution is just an example configuration, and both smaller and larger configurations, different networking configurations and network types, and the use of different storage controllers are possible depending on requirements.

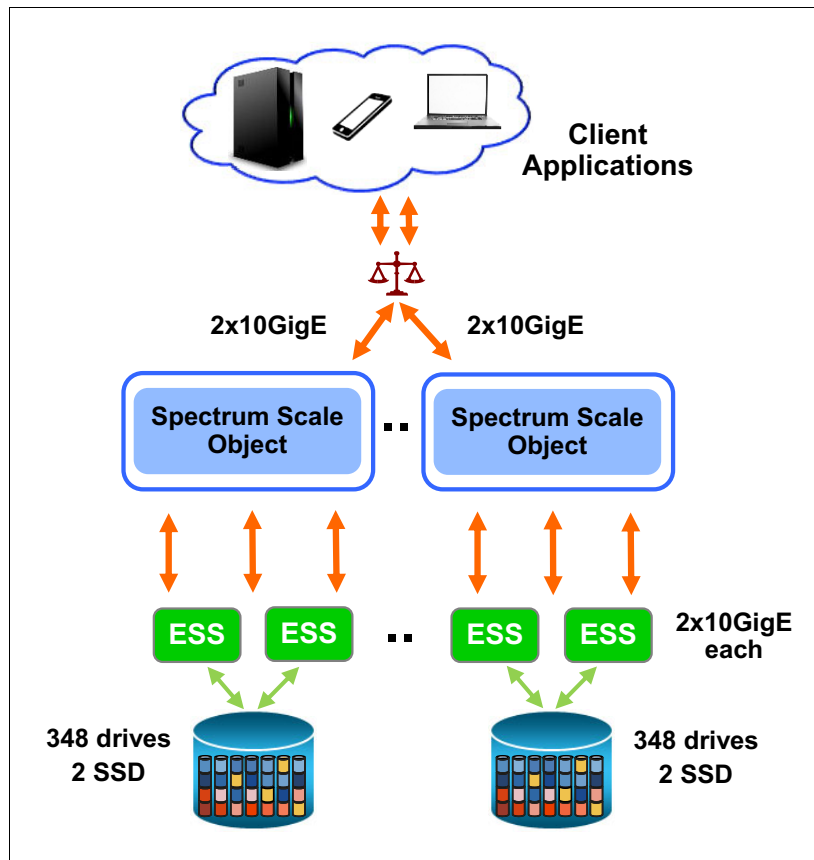


Figure 1-2 An example Spectrum Scale Object architecture that uses the Elastic Storage Server

Consider the following key points in the example architecture:

- ▶ Data protection is handled by the ESS, with Swift writing a single instance of each object in the file system.
- ▶ Additional GPFS Object Nodes provide both additional object access bandwidth and an additional level of fault tolerance in case of GPFS Object Node failure.
- ▶ Each GPFS Object Node can independently access all objects. The Swift Proxy service always accesses the Swift object service that is running on the same node, which means that objects are not transferred between GPFS Object Nodes.
- ▶ The only logical networks that are required are an application network that applications use to access the object store (typically IP-based) and a storage network (IP-based or InfiniBand) that Spectrum Scale uses to store the data.

1.4.4 Benefits

As described in the example architecture in Figure 1-2 on page 9, Spectrum Scale Object can provide an efficient storage solution for both cost and space, which is built upon commodity parts that offers high throughput. The density and performance vary with each storage solution.

The following summary lists the general benefits of Spectrum Scale Object:

- ▶ **Use of Spectrum Scale data protection:** Delegating the responsibility of protecting data to Spectrum Scale (and not by using Swift three-way replication) increases both the efficiency and performance of the system in the following ways:
 - With GPFS Native RAID, storage efficiency rises from 33% to up to 80%.
 - Disk failure recovery does not cause data to flow over the storage network. Recovery is handled transparently and with minimal impact to applications. For more information, see “GPFS-based implementation of a hyper-converged system for software defined infrastructure” by Azagury, et al, in IBM Journal of Research and Development 58 (2), 1-12, 2014.
 - Applications now realize the full bandwidth of the storage network because Spectrum Scale writes only a single copy of each object to the storage servers.
 - Increased maximum object size, up to 5 TB: With Spectrum Scale data striping, large objects do not cause capacity imbalances or server hotspots, and they do not inefficiently use available network bandwidth.
 - No separate replication network is required to replicate data within a single cluster.
 - Capacity growth is seamless because storage capacity can be increased without requiring Swift to rebalance the objects.
 - GPFS Object Node failure does not require any recovery or movement of data between nodes or disks.
- ▶ **Integration of file and object in a single system:** As described in Spectrum Scale features, applications can store a variety of application data in a single GPFS file system. (Currently, we do not support file and object in the same data set.)
- ▶ **Energy savings:** High per-server storage density and efficient use of network resources reduce energy costs.
- ▶ **Enterprise storage management features:** Spectrum Scale Object benefits from GPFS features, such as global namespace, encryption, backup, disaster recovery, ILM (auto-tiering), tape integration, and remote caching.

1.4.5 Detailed architecture

A more detailed view of the architecture is shown in Figure 1-3 on page 11. The GPFS Object Nodes run all Swift services and the GPFS client. Swift clients (users or applications) first obtain a token from the authorization service (Keystone, in our case). The token is included in all requests to Swift, and the Swift proxy service verifies the token by comparing it with cached tokens or by contacting the authorization service. See “Authentication” at this website:

<http://docs.openstack.org/api/openstack-object-storage/1.0/content/authentication-examples-curl.html>

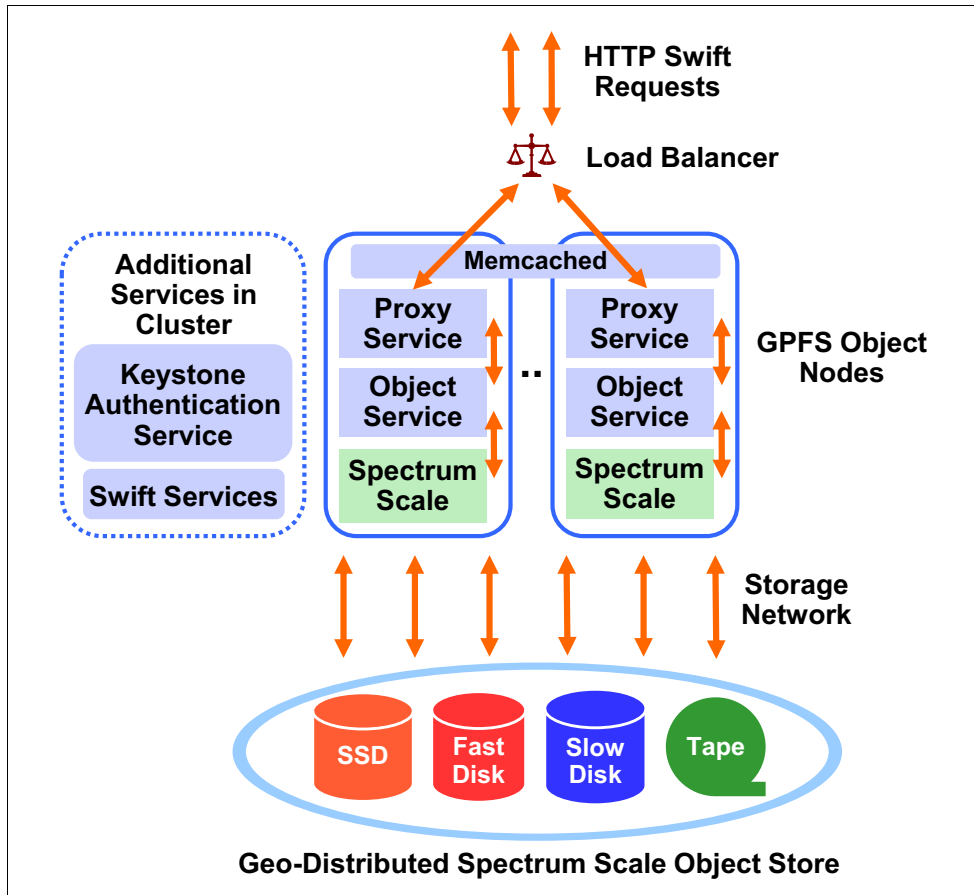


Figure 1-3 Spectrum Scale Object Store architecture


As illustrated in Figure 1-3, the GPFS Object Nodes are both active and provide a front end for the entire object store. The lines represent the I/O flow through the OpenStack Swift components and the GPFS client. The Load Balancer, which distributes HTTP requests across the GPFS Object Nodes, can be based on software or hardware.

After the applications are authenticated, they perform all object store operations (for example, storing and retrieving objects and metadata, or listing account and container information) through any of the proxy service daemons (possibly by using an HTTP Load Balancer as shown in Figure 1-3). For object requests, the proxy service then contacts its local object service, which in turn performs file system-related operations for the GPFS client. Account and container information requests are currently handled a little differently; for more information, see 2.1.6, “High availability” on page 16.

Note: Although this architecture likely works with GPFS File Placement Optimizer (FPO), this configuration has not been tested. See *GPFS 4.1.0.4: Advanced Administration Guide*, SC23-7032, at the following website:

http://www.ibm.com/support/knowledgecenter/SSFKCN_4.1.0.4/com.ibm.cluster.gpfs.v4r104.gpfs200.doc/blladv_fposettings.htm

This document focuses on the use of Spectrum Scale shared storage architectures, such as the Elastic Storage Server (ESS) or the combination of GPFS Network Shared Disk (NSD) servers with a storage controller.



Planning for a Spectrum Scale Object deployment

This chapter describes the key issues that need to be addressed when deploying Spectrum Scale Object. Where Chapter 1, “Spectrum Scale Object” on page 1 described use cases and an example configuration, this chapter digs deeper, with the goal of helping administrators understand the best configuration for their environment. This chapter also lists the tested software versions to consider when deploying Spectrum Scale Object.

Questions and support: If you have questions or comments regarding the information in this paper, contact IBM at gpfs@us.ibm.com. For current Spectrum Scale (based on General Parallel File System) product support information or to open a new service request, contact IBM support at the following website:

<http://www.ibm.com/support>

2.1 Spectrum Scale architecture

The configuration of Spectrum Scale is mostly independent of its use in Spectrum Scale Object. For existing deployments of Spectrum Scale, the Swift software can be layered on top, with its data stored in a new GPFS independent fileset. For new deployments, users need to follow the general guidelines that are described in *GPFS 4.1.0.4: Concepts, Planning, and Installation*, GA76-0441, at this website:

http://www.ibm.com/support/knowledgecenter/SSFKCN_4.1.0.4/com.ibm.cluster.gpfs.v4r104.gpfs300.doc/b11ins_top.htm

In either case though, users need to follow the configuration and tuning suggestions in this guide to help ensure the best experience.

The physical storage architecture is also largely independent, but it will affect the level of efficiency, fault tolerance, and performance. Because the storage also constitutes the bulk of storage costs, the choice of storage hardware also determines the cost/performance of the system. The current storage architecture focuses on shared storage deployments, which can be configured to cover a wide range of workload requirements (see Figure 1-2 on page 9).

2.1.1 Networking

Spectrum Scale Object simplifies the network configuration of typical Swift deployments, but many of the same considerations must be made.

The *client network* that applications use to connect to GPFS Object Nodes is typically a 1 or 10 GigE network (with one or two network interface controllers (NICs)) installed on each server. Because requests are balanced across the GPFS Object Nodes, typically every server has the same network configuration. The network bandwidth of the client network can scale with additional GPFS Object Nodes and the use of a network load balancer to balance requests across the servers. Of course, scaling this network beyond the maximum bandwidth of the storage subsystem might not yield substantial benefits unless a significant amount of data is cached on the GPFS Object Nodes. This is the only network that is visible to OpenStack, and it is therefore the only network that can be managed by OpenStack network management tools.

The *storage network* connects the GPFS Object Nodes to the storage subsystem. This network is typically 10 GigE or InfiniBand and is not visible to, and therefore not managed by, the OpenStack environment. The storage network architecture layout needs to be carefully planned based upon the available physical storage bandwidth and must follow the guidelines of the chosen storage subsystem.

Note: No additional “Swift Replication Network” is required because all object data protection management is handled by Spectrum Scale.

2.1.2 Placement of Object Store components in Spectrum Scale

Because Spectrum Scale can store data for many different applications, it is wise to place the object store data in a separate Spectrum Scale management entity that is called an “independent fileset.” By doing so, it allows Spectrum Scale information lifecycle management (ILM) (including snapshots and backup) to uniquely identify and manage the object store data.

Objects

The location of the objects must be placed in the GPFS storage pool that stores most of the data. Most Spectrum Scale deployments have a single storage pool for data, but the filesset that contains the object data can be mapped to a separate storage pool to provide performance isolation.

Account and container

Currently, we suggest storing the account and container information in the same filesset as the object data, which will ease the management of the data by having all the data in a single data management entity.

Snapshots and backup/restore

Snapshots and backup are critical tools to protect against software and user errors, and catastrophic hardware failures, and they need to be used for any and all critical data sets. By having all account, container, and object information in a single independent filesset, Spectrum Scale can then snapshot the entire object store without affecting any other workloads that might be running against data in other filessets or file systems. After a snapshot is created, it can be backed up to an external storage pool, such as tape, and subsequently restored, if required.

2.1.3 Authentication

The authentication mechanism that is confirmed to work with Spectrum Scale Object is the official OpenStack authentication mechanism that is known as *Keystone*. See “The Auth System” at this website:

http://docs.openstack.org/developer/swift/overview_auth.html#keystone-auth

2.1.4 Data protection

Spectrum Scale Object relies on the storage subsystem for data protection and realizes the same level of protection against physical storage failures (for example, RAID 5 or RAID 6). Spectrum Scale can provide an additional level of protection by replicating data within the storage subsystem, but this protection is not required when the storage subsystem is providing data protection itself. Spectrum Scale can also synchronously replicate I/O to a separate storage cluster, but handling disaster recovery is not described at this time. For more information, see “Configuring GPFS for Reliability” at this website:

[https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/General%20Parallel%20File%20System%20\(GPFS\)/page/Configuring%20GPFS%20for%20Reliability](https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/General%20Parallel%20File%20System%20(GPFS)/page/Configuring%20GPFS%20for%20Reliability)

2.1.5 Performance

The system design must be crafted to ensure that it meets the requirements of one or more of the following possible requirements:

- ▶ **Individual object throughput:** Each object is accessed through a single GPFS Object Node only. Therefore, the robustness (maximum network throughput, number of CPUs, and so on) of each node will generally determine the performance of reading and writing objects (assuming that the storage subsystem is capable of at least saturating a single node). In addition, for smaller objects, it is possible that an object is stored on a single storage device, in which the latency of the storage device might be the limiting performance factor.

- ▶ **Aggregate object throughput:** The overall performance is determined less by the capability of a single node or individual storage device than by other factors, such as the number of GPFS Object Nodes, the aggregate throughput of the storage subsystem, the client network bandwidth, and the capacity of the SQLite database.
- ▶ **Container/account access performance:** Swift uses memcached to improve the performance of retrieving account and container information by caching the data across the memory of the GPFS Object Nodes. In addition, the SQLite database performs small I/O accesses to Spectrum Scale to retrieve the information. Increasing the amount of memory on the servers helps improve account/container access performance.

In addition, several other factors affect performance:

- ▶ Tuning of Spectrum Scale is important. Although more information about Spectrum Scale tuning is provided in “Spectrum Scale installation and configuration” on page 25, a general description of tuning is available in “Tuning Parameters” at this website:
[https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/General%20Parallel%20File%20System%20\(GPFS\)/page/Tuning%20Parameters](https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/General%20Parallel%20File%20System%20(GPFS)/page/Tuning%20Parameters)
- ▶ If performance is an issue, it is important to understand which other workloads are running against the same disk subsystem that might be reducing performance.

2.1.6 High availability

To ensure the availability of Spectrum Scale Object, it is important to ensure a correct configuration that accounts for the level of fault tolerance that you want.

For objects, the GPFS Object Nodes are independent, with each additional server increasing performance and availability. Unfortunately, the same is not currently true for account and container information. Because of issues that are related to inter-node locking, the account/container information must continue to be replicated within Spectrum Scale. This situation has little negative effect on performance or disk space usage (typically, this information represents less than 1% of the total disk usage), but it does affect availability.

For a proof of concept deployment involving only a single node, high availability is not an issue. The only requirement is that the number of Swift virtual devices is equal to or greater than the replication factor chosen for the account/container rings.

For a production deployment, the number of GPFS Object Nodes must be two or more to ensure an access route to the information if a node fails. The Swift account/container replication must be set to 2 if there are two GPFS Object Nodes, or to 3 if there are a larger number of nodes available. The number of virtual devices must then be at least as large as the replication factor (we suggest 10 virtual devices per GPFS Object Node). When building the account and container rings, the virtual devices are bound to unique GPFS Object Nodes.

Note: Because account/container replicas are in Spectrum Scale, it is not an issue of protecting account/container data from failure. What is important is that the account and container rings specify a path to access data following the loss of any node.

This paper does not describe how to configure an OpenStack environment for high availability. There are some Swift services that run on a single node, and in a production environment, these services need to be monitored and automatically restarted. For more information about the high availability of OpenStack Swift, see Chapter 1. Introduction to OpenStack High Availability on the OpenStack website:

<http://docs.openstack.org/high-availability-guide/content/ch-intro.html>

The availability of the GPFS cluster also needs to be considered in the system design. For more information, see these resources:

- ▶ *GPFS 4.1.0.4: Administration and Programming Reference*, SA23-1452:
http://www.ibm.com/support/knowledgecenter/SSFKCN_4.1.0.4/com.ibm.cluster.gpfs.v4r104.gpfs100.doc/bl1adm_top.htm
- ▶ *GPFS 4.1.0.4: Concepts, Planning, and Installation*, GA76-0441:
http://www.ibm.com/support/knowledgecenter/SSFKCN_4.1.0.4/com.ibm.cluster.gpfs.v4r104.gpfs300.doc/bl1ins_top.htm
- ▶ “Configuring GPFS for Reliability” at this website:
[https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/General%20Parallel%20File%20System%20\(GPFS\)/page/Configuring%20GPFS%20for%20Reliability](https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/General%20Parallel%20File%20System%20(GPFS)/page/Configuring%20GPFS%20for%20Reliability)

2.1.7 Spectrum Scale and Swift interaction

Swift accesses data in Spectrum Scale by using the POSIX interface, but it places limits on the amount of time that it waits to store or retrieve information. Depending on the load of the system, these timeouts might need to be increased in the Swift configuration files.

2.1.8 Administration tools

It is highly advised to manage Swift by using the Swift command-line interface rather than the OpenStack Horizon GUI management tool, which requires the installation of the OpenStack Nova and Glance components. See “Chapter 9. Object Storage command-line client” on the OpenStack website:

http://docs.openstack.org/cli-reference/content/swiftclient_commands.html

2.2 Tested configurations

The following list shows configurations and software that have been tested and shown to work:

- ▶ GPFS Standard Edition Version 4.1.0.5
- ▶ RHEL7 on x86 and ppc platforms
- ▶ Security-Enhanced Linux (SELinux) disabled
- ▶ OpenStack Juno with Swift Version 2.2
- ▶ Keystone authentication
- ▶ Maximum object size: 5 TB
- ▶ Shared storage configurations only, including ESS, NSD servers, or traditional block controllers
- ▶ Single GPFS cluster (no data distribution to remote clusters or sites)



Spectrum Scale Object configuration overview

This section summarizes the configuration settings to optimize Swift while taking advantage of Spectrum Scale features wherever appropriate.

The setup requires *no changes* to Swift software. Optimization is provided through configuration settings that are made when creating Swift rings and also in Swift configuration files.

3.1 Swift replication

Spectrum Scale gives every node in the Swift cluster access to every object. If any single node fails, object data is still available from all of the remaining nodes. For objects, there is no need to use Swift replication to make data available from other nodes.

To achieve this, the Swift object ring is built with a replication factor of 1 and there is one copy of each object that is maintained by Swift. (See 2.1.6, “High availability” on page 16.) There is no need to copy objects between servers as part of replication or rebalance operations. Spectrum Scale provides data protection through General Parallel File System (GPFS) Native RAID, storage controller RAID, or GPFS replication.

In this paper, we use Swift replication for account and container data because of the way that Swift locks container and account information while it is updated. The Swift locking module is not supported by Spectrum Scale.

3.2 Swift rings

Every GPFS Object Node can access the shared file system, so all of the rings are constructed using virtual devices rather than physical devices. The virtual devices are subdirectories in the GPFS fileset that is created for Swift data. When constructing the object ring, virtual devices are added to the “localhost” node. This has the effect of giving every proxy service local access to every virtual device.

Note: Do not use a single virtual device to contain all object data due to access contention between nodes. Currently, we advise scaling the number of virtual devices. One approach is to create a reasonable number of virtual devices (for example, 10) for each GPFS Object Node that currently exists, or is expected to exist, in the configuration.

The account and container rings are created with virtual devices evenly distributed to all of the GPFS Object Nodes. This is done by assigning virtual devices for these rings to the external IP addresses of the nodes, rather than using the localhost IP address.

3.3 Swift services

Every GPFS Object Node can access every virtual device in the shared file system, and some Swift object services can be optimized to take advantage of this by running from a single GPFS Object Node:

- ▶ Even though objects are not replicated by Swift, the swift-object-replicator runs to periodically clean up tombstone files from deleted objects. It is run on a single GPFS Object Node and manages cleanup for all of the virtual devices.
- ▶ The swift-object-updater is responsible for updating container listings with objects that were not successfully added to the container when they were initially created, updated, or deleted. Like the object replicator, it is run on a single GPFS Object Node.

Note: The swift-object-auditor is responsible for comparing the checksum in the object file's extended attributes with the checksum of the object data on disk. If a discrepancy is discovered, the object file is moved to Swift's quarantine directory, with the expectation that the object-replicator will eventually replace the quarantined object with a replica object instance.

With Spectrum Scale Object, it is better to use storage controller capabilities, such as GPFS Native RAID checksums and disk scrubbing/auditing capabilities. For this reason, the swift-object-auditor service is not run on any node.

Table 3-1 lists each of the Swift services and the set of GPFS Object Nodes on which they need to be run.

Table 3-1 Swift services that are mapped to GPFS Object Nodes

Service name	Runs on GPFS Object Nodes
openstack-swift-account	All
openstack-swift-account-auditor	All
openstack-swift-account-reaper	All
openstack-swift-account-replicator	All
openstack-swift-container	All
openstack-swift-container-auditor	All
openstack-swift-container-updater	All
openstack-swift-container-replicator	All
openstack-swift-object	All
openstack-swift-object-auditor	None
openstack-swift-object-expirer	All (started only if object expiration is enabled)
openstack-swift-object-replicator	Single
openstack-swift-object-updater	Single
openstack-swift-proxy	All



Spectrum Scale Object installation

This chapter provides detailed instructions for the Spectrum Scale Object installation process by using the automated installation tool that is provided in the additional material (see Appendix A, “Additional material” on page 57) that is associated with this paper.

Note: The installation and maintenance procedures described in Chapter 4 and 5 of this Redpaper are not compatible with the Spectrum Scale protocol installation toolkit, provided in Spectrum Scale 4.1.1 and later versions.

The steps in this paper are valid, and should be only be executed on Spectrum Scale nodes that are not designed as Cluster Export Service (CES) protocol nodes.

4.1 Installation overview

This paper includes automated installation tools for installing and configuring OpenStack Swift in an existing Spectrum Scale environment. It generally follows the OpenStack Juno installation procedure that is described at the following website:

<http://docs.openstack.org/juno/install-guide/install/yum/content/>

In this paper, we modify the configuration to optimize Swift to run in a Spectrum Scale environment. Those changes are described in more detail in this paper.

You can use the installation tool to accomplish the following tasks:

- ▶ Specify an existing GPFS file system
- ▶ Create an independent fileset that is linked to that file system
- ▶ Install and configure the Juno version of OpenStack Swift software on a set of GPFS Object Nodes
- ▶ Install and configure the Juno version of OpenStack Keystone software on a selected GPFS Object Node, or link to an existing Keystone server

The code for the installation tool can be downloaded from the IBM Redbooks website. To access this code, go to <http://www.redbooks.ibm.com/>, follow the “Additional Materials” link, and find REDP5113 directory (<ftp://www.redbooks.ibm.com/redbooks/REDP5113/>). All of the materials should be downloaded and reviewed.

Installation flow

The steps in the installation process are listed here to provide an overview. The details of each step are described in detail in 4.4, “Object software installation and configuration” on page 27:

1. Ensure that the prerequisites that are defined in 4.2, “Installation prerequisites” on page 25 are satisfied.
2. Download the installation and sample code from <ftp://www.redbooks.ibm.com/redbooks/REDP5113/> and place it in a node that will be used for the installation. The node can be part of your GPFS cluster or a separate system.
3. Extract the contents of the compressed file on to the selected installation system.
4. Run the installer and complete the following steps from the command line or from the installer menu:
 - a. Specify the GPFS Object Nodes that will be configured.
 - b. Specify the passwords.
 - c. Specify the GPFS configuration.
 - d. Run the installation.
5. Perform postinstallation checking steps.

Note: The installer can also be used to add or remove Object Nodes. This process is described in 5.2, “Adding a GPFS Object Node” on page 36 and 5.3, “Removing a GPFS Object Node” on page 37.

4.2 Installation prerequisites

This section describes the installation prerequisites that must be met to use the installation tool.

SSH and network setup

The installer node and all of the GPFS Object Nodes require external internet access during the installation process. They also require SSH keys to be set up so that the installer can run remote commands without password authentication.

The installer uses port 8889 for communicating with the Chef server. This port must be reachable from all GPFS Object Nodes. The installer does not change any SELinux settings; the user must make those settings, if required. For more information about these settings, see 5.6, “Security-Enhanced Linux considerations” on page 40.

Repository setup

The following software repositories must be enabled through the Red Hat Satellite Service by the user:

- ▶ RHEL Server Extras
- ▶ Red Hat OpenStack 5 for Server 7.0

If it is not possible to enable these repositories, you can manually install the following two packages on each GPFS Object Node:

- ▶ python-greenlet v0.3.2 or higher
- ▶ python-webob v1.2.3.8 or higher

NTP setup

Network Time Protocol (NTP) must be configured on all nodes in your system to ensure that all of the nodes' clocks are synchronized. For more information about installing NTP, see the “Network Time Protocol (NTP)” topic in *OpenStack Installation Guide for Red Hat Enterprise Linux 7, CentOS 7, and Fedora 20*, found at this website:

http://docs.openstack.org/juno/install-guide/install/yum/content/ch_basic_environment.html#basics_ntp

4.3 Spectrum Scale installation and configuration

General Parallel File System (GPFS) must be installed on every GPFS Object Node in your system. For more information, see *GPFS 4.1.0.4: Concepts, Planning, and Installation*, GA76-0441, found at the following website:

http://www.ibm.com/support/knowledgecenter/SSFKCN_4.1.0.4/com.ibm.cluster.gpfs.v4r104.gpfs300.doc/bl1ins_top.htm

Note: The Spectrum Scale software is not required on the controller node that hosts the Keystone identity service if it is deployed on a separate node from the GPFS Object Nodes.

A GPFS cluster and file system are required. If this infrastructure does not exist, you must install the Spectrum Scale software and create a GPFS cluster. Next, create a GPFS file system and mount the GPFS file system on all GPFS Object Nodes. The installer creates an independent fileset in the GPFS file system that you name. This independent fileset is used only for Swift object storage.

GPFS tuning for Swift data

This section describes general GPFS tuning and also includes commands that are related to tuning. A comprehensive description of GPFS tuning can be found at this website:

[https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/General%20Parallel%20File%20System%20\(GPFS\)/page/Tuning%20Parameters](https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/General%20Parallel%20File%20System%20(GPFS)/page/Tuning%20Parameters)

Setting the number of inodes

The GPFS default maximum number of inodes is acceptable for most configurations, but for large systems that expect hundreds of millions of objects or more, the maximum number of inodes to set at fileset creation time might need to be increased. The maximum number of inodes must be larger than the sum of the maximum number of expected objects and the inodes that are required by the Swift directory structure.

The maximum number of inodes must be set to at least twice the maximum number of expected objects. So, for example, if the expected number of objects is 100 million, the maximum number of inodes must be set to 200 million.

The installer sets the maximum number of inodes to 8 million by default, and all of them are preallocated.

To modify the maximum inode limit for a fileset, run the following command:

```
mmchfileset FileSystem Fileset --inode-limit MaxNumInodes[:NumInodesToPreallocate]
```

Note: Specifying a relatively large number of pre-allocated inodes will also help improve ingest performance, and it needs to be at least as large as the number of inodes that are created by an application or benchmark.

GPFS file system cache

The GPFS *pagepool* caches file system information in memory for fast access (similar to the Linux Page Cache). Use **mmchconfig** to set this GPFS pagepool to be at least 30% - 50% of memory on each GPFS Object Node.

maxFilesToCache

This parameter limits the total number of different files that can be cached at one time. This needs to be set by using **mmchconfig** to be large enough to handle the number of concurrently open files (objects), plus allow caching of recently used files.

maxStatCache

This parameter sets aside additional pageable memory to cache attributes of files that are not currently in the regular file cache. Increasing this value can improve the performance of background Swift services that scan directories.

Note: If you change the **maxFilesToCache** value but not the **maxStatCache** value, the **maxStatCache** value will default to 4 x the **maxFilesToCache** value.

seqDiscardThreshold

The default for this value is 1 MB, which means that if you have a file that is sequentially read and is greater than 1 MB, Spectrum Scale does not keep the data in cache after consumption. Because everything in Swift is read sequentially, unless this value is increased, Spectrum Scale does not cache anything, so this value must be increased to the largest expected object size that must be cached in memory.

tokenMemLimit

This parameter sets the size of the memory that is available for manager nodes to use for caching tokens; it controls the allocation of memory that is separate from the pagepool. With many objects, or if high periods of activity are expected, increase the amount of memory to cache tokens to 2 GB or larger.

worker1Threads

This parameter represents the total number of concurrent application requests that can be processed at one time (including Swift operations). To support periods of high activity, increase this value to 1024. The `work1threads` parameter can be reduced without having to restart the GPFS daemon. Increasing the value of `worker1threads` requires a restart of the GPFS daemon.

4.4 Object software installation and configuration

After you have ensured that all of the prerequisites that are described in 4.2, “Installation prerequisites” on page 25 and 4.3, “Spectrum Scale installation and configuration” on page 25 are met, you can begin the installation. Follow the instructions in the following sections to start the installation.

Downloading and unpacking the installer software

First, download the Spectrum Scale Object installer from the IBM Redbooks FTP site by running the following command:

```
wget ftp://www.redbooks.ibm.com/redbooks/REDP5113/*
```

The following two files are downloaded:

- ▶ `disclaimer.txt`
- ▶ `esobject-installer-1.1.x.tgz`

Note: `x` represents the level of the code that is available at the IBM Redbooks web server. Update the `x` to match the current level in the REDP5113 folder.

The `esobject-installer` tar file contains all of the needed installation recipes and scripts, and Version 11.12.8 of the Chef client for both `x86_64` and `ppc64` architectures. The Chef client automatically is installed on the installer node and all of the installation target nodes. Move the files to a suitable directory on the installer node, and extract the files from the compressed file by running the following command

```
tar -xvf esobject-installer-1.1.x.tgz
```

Review the `disclaimer.txt` and `license.txt` files that are in the top-level directory of the extracted files. Also, review the `readme` file, as it contains a more detailed description of the `esobject` commands.

Using the installer

The `esobject` file is the installer driver. You can view the help page for the installer by adding the `-h` parameter to `esobject`:

```
./esobject -h
```

To view more detailed information for any command, use the `-v` parameter (verbose) with any command. For example:

```
./esobject -v <command parameter>
```

You can enter a text-based menu mode by using the `menu` parameter with `esobject`:

```
./esobject menu
```

The sequence of commands is the same for the menu mode and for the command-line interface (CLI). For simplicity, we describe the CLI mode.

Initializing the installer environment

Use the `setup` parameter to install the Chef client software on the installer node and load the required components. This step can take up to 10 minutes to complete:

```
./esobject setup
```

<output omitted>

Initialize the Chef server by using the IP address of the installer node:

```
./esobject server -s <Installer IP Address>
```

For example, if the installer IP address is 10.10.10.1, use the following command (the resulting output is shown):

```
./esobject server -s 10.10.10.1
[ INFO ] Starting chef zero
[ INFO ] Vending and uploading cookbooks...
[ INFO ] Done
[ INFO ] Uploading roles and data bags...
[ INFO ] Done
```

Adding GPFS Object Nodes

Specify the nodes that will be part of the GPFS Object deployment. Again, these nodes must be part of a GPFS cluster and all have the same GPFS file system mounted. Add each node by using the `add` parameter:

```
./esobject add <GPFS Object Node>
```

One node must be designated as the Keystone node by using the `-k` parameter:

```
./esobject add -k <GPFS Object Node>
```

For example, if you have four GPFS Object Nodes with IP addresses 10.10.10.1 - 10.10.10.4, and the node at 10.10.10.4 will host the Keystone server, enter the following commands:

```
./esobject add 10.10.10.1
[ INFO ] Added node with fqdn '10.10.10.1' and ip 10.10.10.1.
[ INFO ] Added role swift_non_master to node 10.10.10.1
[ INFO ] Saved configuration to configuration/nodes.json.
./esobject add 10.10.10.2
[ INFO ] Added node with fqdn '10.10.10.2' and ip 10.10.10.2.
```

```
[ INFO ] Added role swift_non_master to node 10.10.10.2
[ INFO ] Saved configuration to configuration/nodes.json.
./esobject add 10.10.10.3
[ INFO ] Added node with fqdn '10.10.10.3' and ip 10.10.10.3.
[ INFO ] Added role swift_non_master to node 10.10.10.3
[ INFO ] Saved configuration to configuration/nodes.json.
./esobject add 10.10.10.4
[ INFO ] Added node with fqdn '10.10.10.4' and ip 10.10.10.4.
[ INFO ] Added role swift_non_master to node 10.10.10.4
[ INFO ] Saved configuration to configuration/nodes.json.
./esobject add -k 10.10.10.4
[ INFO ] Added role keystone to node 10.10.10.4
[ INFO ] Saved configuration to configuration/nodes.json.
```

You can view the list of defined GPFS Object Nodes by using the `list` parameter as follows:

```
./esobject list
```

With our example nodes, the output look like the following output:

```
./esobject list
[ INFO ] List of nodes in current configuration:
[ INFO ] 10.10.10.1 [swift_non_master]
[ INFO ] 10.10.10.3 [swift_non_master]
[ INFO ] 10.10.10.2 [swift_non_master]
[ INFO ] 10.10.10.4 [swift_non_master, keystone]
```

If you want to link to an existing Keystone server, use the `--existing` parameter. In this case, the installer updates the Swift proxy server configuration files to use that server. No changes are made on that server.

For example, remove the local Keystone server from the configuration and add an existing external Keystone server by running the following commands:

```
[root@boneknapper-vm1 esobject_sw]# ./esobject delete 10.10.10.4
[ INFO ] Removed node with fqdn of 10.10.10.4
[ INFO ] Saved configuration to configuration/nodes.json.

[root@boneknapper-vm1 esobject_sw]# ./esobject add -k --existing 10.10.20.1
[ INFO ] Added node with fqdn '10.10.20.1' and ip 10.10.20.1.
[ INFO ] Added role keystone to node 10.10.20.1
[ INFO ] Saved configuration to configuration/nodes.json.

[root@boneknapper-vm1 esobject_sw]# ./esobject list
[ INFO ] List of nodes in current configuration:
[ INFO ] 10.10.10.1 [swift_non_master]
[ INFO ] 10.10.10.3 [swift_non_master]
[ INFO ] 10.10.10.2 [swift_non_master]
[ INFO ] 10.10.20.1 [keystone] (keystone will not be installed)
```

Setting passwords

Specify the password that is used for Keystone, and optionally, a second password that is used for the database manager by using the `swiftpw` parameter:

```
./esobject swiftpw
```

Here is the output of the command. In this case, we are using a unique password for the database:

```
./esobject swiftpw
[ INFO ] Please provide the admin password:
Password:
[ INFO ] Please reenter the admin password:
Password:
Do you want to use the same password for database root user? Enter "Yes" to
confirm,
anything else not to:no
[ INFO ] Please provide the database root password:
Password:
[ INFO ] Please reenter the database root password:
Password:
[ INFO ] Created secret key encrypted_data_bag_secret
[ INFO ] Creating data bags with password...
[ INFO ] Created encrypted data bags
```

Defining the GPFS configuration

The values for the GPFS device name, GPFS mount point, and object-independent fileset name can be entered by using the **gpfsconfig** parameter. They can be entered and updated individually or in a single command. For example:

```
./esobject gpfsconfig -d <gpfs_device_name> -g <gpfs_mount_point> \
-o <object_fileset> -i <inode_allocation>
```

The object base parameter is the independent fileset name where all object data is stored.

For example:

```
./esobject gpfsconfig -d gpfs -g /gpfs0 -o objectfs
[ INFO ] Setting GPFS Device Name to gpfs
[ INFO ] Setting GPFS Mount Point to /gpfs0
[ INFO ] Setting GPFS Fileset for Object Base to objectfs
[ INFO ] Configuration saved
```

The object fileset is created when you run the **install** command. At that time, the fileset is linked to the file system by using the mount point as the junction. By default, the object fileset is created with 8 million inodes allocated. For more information about this parameter, see “Setting the number of inodes” on page 26. To modify that value (for example, to 10 million), set a new inode setting by using the **-i** parameter:

```
./esobject gpfsconfig -i 10000000
[ INFO ] Setting GPFS Fileset inode allocation to 10000000
[ INFO ] Configuration saved
```

To view the GPFS configuration settings, use the **gpfsconfig** parameter with no additional parameters:

```
[root@boneknapper-vm1 esobject_sw]# ./esobject gpfsconfig
[ INFO ] No changes made. Current settings are as follows:
[ INFO ] GPFS Device Name is gpfs
[ INFO ] GPFS Mount Point is /gpfs0
[ INFO ] GPFS Fileset for Object Base is objectfs
[ INFO ] GPFS Fileset inode allocation is 10000000
```

Installation precheck

Before beginning the installation, optionally run the installation precheck. These checks are also run during the installation, but it can be useful to run them before starting the installation. Run the following command:

```
./esobject precheck
[ INFO ] Running environment checks
[ INFO ] ChefDK - Detected
[ INFO ] chef-zero path configuration - Confirmed
[ INFO ] Nodes - Configured
[ INFO ] Passwordless SSH - Enabled
[ INFO ] GPFS configuration - Checked
[ INFO ]
[ INFO ] Configuration checks complete
[ INFO ]
[ INFO ] RESULT: System ready for install.
```

Running the installation

The configuration is defined and you are ready to start the installation by using the `install` parameter (for details, see “Installation details”):

```
./esobject install
<output omitted>
```

Note: Summary messages are shown on the screen unless verbose output is requested. All messages are always logged in to `chef_output.log`.

Installation postcheck

After completing the installation, optionally run the installation postcheck to verify that the environment is successfully installed. These checks are also run during the installation phase, but it can be useful to run them separately following the installation.

```
./esobject postcheck
[ INFO ] Running post-install checks
[ INFO ] Checks finished: Installation completed successfully
```

All the required services are started and you can begin using the Spectrum Scale Object environment.

Installation details

The installer runs the steps that are described in the *OpenStack Juno Installation Guide*, which can be found at the following website:

<http://docs.openstack.org/juno/install-guide/install/yum/content/>

Here are the specific sections that describe the installation steps:

- ▶ 2. Basic environment
- ▶ 3. Add the Identity service
- ▶ 9. Add Object Storage

There are several configuration changes that are made to optimize operation in the Spectrum Scale environment, which are listed here:

- ▶ The replication factor for the object ring is set to 1 so that object data is not replicated by Swift. Spectrum Scale is responsible for protecting object data.
- ▶ All virtual devices are assigned to localhost in the object ring. The `bind_ip` setting in `object-server.conf` is set to `localhost` on all GPFS Object Nodes. This allows all proxy to object communication to be local to the GPFS Object Node on which the request is received.
- ▶ Account and container replication factor are set to 2 by default. The `bind_ip` setting is defined as the external IP address for the GPFS Object Node, which is typical for an OpenStack Swift deployment.
- ▶ The `partition_power` setting for all rings is set to 14 by default. This setting is increased from the original recommendation of 8 in the first edition of this paper, based on additional performance work. Having a broader directory tree reduces directory contention for larger numbers of container and objects.
- ▶ The account and container data is in a different directory path from object data. This facilitates GPFS placement policies to allow account and container data to be stored in a faster storage pool than objects. For more information, see 5.4, “Account and container data placement policies” on page 38.
- ▶ Ten virtual devices are added for each GPFS Object Node to the account, container, and object rings. As additional GPFS Object Nodes are added to or removed from the system, the virtual devices may be reassigned to maintain a balanced number of virtual devices across all nodes. For more information, see 5.2, “Adding a GPFS Object Node” on page 36 and 5.3, “Removing a GPFS Object Node” on page 37.
- ▶ An `openrc` file is created in the `$HOME` directory of each GPFS Object Node for the admin, user, and tenant. For information about how to add additional users and tenants, see the following website:
<http://docs.openstack.org/juno/install-guide/install/apt/content/keystone-users.html>
- ▶ Sample scripts are placed in the `$HOME/eso_samples` directory. These scripts are provided to help automate administrative tasks and are explained in Chapter 5, “System administration considerations” on page 35.

References for additional installation information: For general information about tuning Swift components, see “General System Tuning” at the following website:

http://swift.openstack.org/deployment_guide.html#general-system-tuning

4.5 Postinstallation

After installation, you must verify the installation, and you might need to tune the configuration.

4.5.1 Verifying the installation

For one or more GPFS Object Nodes, run the following example Swift commands to verify your installation. If the installation completed successfully, you can list all containers, upload a sample object to a container, and list that container and see the object.

- ▶ `source ~/openrc`
- ▶ `swift list`
- ▶ `date > object1.txt`
`swift upload test_container object1.txt`
- ▶ `object1.txt`
- ▶ `swift list test_container`
- ▶ `object1.txt`

4.5.2 Tuning considerations

You might need to modify the value for the **workers** parameter in Swift server configurations. This parameter is set to **auto** for a proxy server. With the **auto** setting, the worker count is automatically set to the number of cores that are detected on the node where the proxy server is running. In our example, we set the **workers** parameter to 8 in the object server and 2 in the account and container server configuration files. Depending on the memory and number of cores in your GPFS Object Nodes, you might be able to increase these settings.

To increase these settings, modify the account, container, and object server configuration files in `/etc/swift` and restart the Swift services on each GPFS Object Node. For instructions about restarting the Swift services, see 5.1, “Managing Swift services” on page 36.

The installer sets two important Swift timeout values in the proxy server configuration file. Both `node_timeout` and `conn_timeout` are set to 120 seconds to minimize timeouts during periods of heavy load.

4.6 Removing the installation

To remove the installation, run **uninstall** as follows:

```
./esobject uninstall
```

This command removes all OpenStack software, and removes the object fileset and all data that it contains.

You can remove individual GPFS Object Nodes by using the **rmnode** parameter as follows:

```
./esobject rmnode <GPFS Object Node>
```

This command removes the OpenStack software from that node, but leaves the object data unchanged. For the details of this operation, see 5.3, “Removing a GPFS Object Node” on page 37.

For example, to remove node 10.10.10.3 from the configuration, run the following command:

```
./esobject rmnode 10.10.10.3
```




System administration considerations

This chapter describes Swift administrative actions that behave differently in a Spectrum Scale Object environment, and the operational considerations that we encountered during our testing.

One of the benefits of using a clustered file system for Swift storage is simplified maintenance. Nodes can be added or replaced with minimal effort and with no need for the rebalancing or movement of data. This chapter describes the process for adding and removing a GPFS Object Node in your system.

For performance reasons, it is preferable to place account and container data into a higher performing storage pool. This chapter describes the procedure for this placement by using GPFS ILM placement policies.

Finally, this chapter provides suggestions for several other system administration areas.

Note: The installation and maintenance procedures described in Chapter 4 and 5 of this Redpaper are not compatible with the Spectrum Scale protocol installation toolkit, provided in Spectrum Scale 4.1.1 and later versions.

The steps in this paper are valid, and should be only be executed on Spectrum Scale nodes that are not designed as Cluster Export Service (CES) protocol nodes.

5.1 Managing Swift services

We provide a utility script, `manage_swift_services.sh`, to help manage Swift services running on GPFS Object Nodes. This script uses `systemctl` for starting, stopping, and displaying the status for the required Swift services.

Note: The use of the `swift-init` command is discouraged in the RHEL environment. All Swift services should be started by *using only the `systemctl` command*. This provides consistency with other system services, and proper behavior for Swift services in environments where SELinux is enabled.

The installer places the `manage_swift_services.sh` script in the `$HOME/eso_samples/Utils` directory of each GPFS Object Node. Here is the exact location:

```
$HOME/eso_samples/Utils/manage_swift_services.sh
```

Note: To simplify the running of this script, you might want to include the path to the `Utils` directory in your `PATH` variable on each GPFS Object node. For example:

```
PATH=$PATH:$HOME/eso_samples/Utils/
```

To start all required object services, run the following command on each GPFS Object Node:

```
./manage_swift_services.sh start
```

To stop all required object services, run the following command on each GPFS Object Node:

```
./manage_swift_services.sh stop
```

To check the status of all required object services, run the following command on each GPFS Object Node:

```
./manage_swift_services.sh status
```

5.2 Adding a GPFS Object Node

To add a node, you must first install the prerequisite software and GPFS software. Then, you add the node to your GPFS cluster. Finally, use the `esobject` installer to install the required software and configure the new node. The installer adds the required object software, updates the account, container, and object rings, and redistributes the rings to all nodes.

Rebalancing or data movement is not required because all virtual devices can be accessed from every GPFS Object Node.

After the new node is added to the GPFS cluster and the file system is mounted, the Swift services are started automatically and the new node can begin handling object requests.

Note: Your load balancer configuration might need to be updated to recognize the new node.

To add a node to your configuration, run the following commands from your installation server to list the set of GPFS Object Nodes, add a node, and then update the configuration to complete the installation on that node:

```
./esobject add <ip>
./esobject list
./esobject update
```

Note: With the current installer version, you must add one node and then run an update before adding more nodes. Adding two or more nodes followed by a single update is not supported currently.

For example, to add node 10.10.10.5 to our example configuration, run the following commands:

```
./esobject add 10.10.10.5
[ INFO ] Added node with fqdn '10.10.10.5' and ip 10.10.10.5.
[ INFO ] Saved configuration to configuration/nodes.json.

./esobject list
[ INFO ] List of nodes in current configuration:
[ INFO ] 10.10.10.1 [swift_non_master]
[ INFO ] 10.10.10.3 [swift_non_master]
[ INFO ] 10.10.10.2 [swift_non_master]
[ INFO ] 10.10.10.4 [swift_non_master, keystone]
[ INFO ] 10.10.10.5 [swift_non_master]

./esobject update
#<output omitted>
```

5.3 Removing a GPFS Object Node

To remove a node from your configuration, remove the node by using the `rmnode` parameter:

```
./esobject rmnode <ip>
```

The installer removes all of the object software from the node, updates the account, container, and object rings, and redistributes the rings to all nodes.

Rebalancing or data movement is not required because all virtual devices can be accessed from every GPFS Object node.

For example, to remove node 10.10.10.5 from our example configuration, run the following commands:

```
./esobject rmnode 10.10.10.5
<output omitted>

./esobject list
[ INFO ] List of nodes in current configuration:
[ INFO ] 10.10.10.1 [swift_non_master]
[ INFO ] 10.10.10.3 [swift_non_master]
[ INFO ] 10.10.10.2 [swift_non_master]
[ INFO ] 10.10.10.4 [swift_non_master, keystone]
```

5.4 Account and container data placement policies

For performance reasons, it might be preferable to place account and container data into a separate storage pool that is backed by faster solid-state device (SSD) storage. This can be easily accomplished by using GPFS placement policies.

To implement the placement policies, a new *dependent fileset* must be created to hold all account and container data. A storage pool is then created that is backed by the preferred storage. A file placement policy must then be written and deployed so that all files that are created in the new dependent fileset automatically are placed into the new storage pool. The installer sets the `devices` parameter in both account server and container server configuration files to use a directory named `ac` under the object fileset. The `devices` parameter in the object server configuration file is set to use a directory that is named `o` under the object fileset. This allows the account and container-dependent fileset to be easily linked to the `ac` directory.

Currently, however, the installer does not automatically configure the account and container fileset or create the placement policies. This task must be performed by the user, but an example is provided below.

In the following example, the GPFS file system device name is `gpfs`, the file system mount point is `/gpfs0`, and the object independent fileset is `objectfs`.

Note: Delete the `ac` directory before you start this process. If you already have account and container data in the `ac` directory, copy this data to a temporary location and then delete the `ac` directory.

To configure the account and container filesets, complete the following steps:

1. Create a stanza file that is named `nsd_file` to define the SSD devices that are used in the new storage pool. In this example, the pool is called `poolAC` and there are two devices:

```
%nsd:
device=/dev/dm100
usage=dataOnly
pool=poolAC
```

```
%nsd:
device=/dev/dm101
usage=dataOnly
pool=poolAC
```

2. Create the NSDs by running the following command:

```
mmcrnsd -F nsd_file -v no
```

3. Add the NSDs to your GPFS file system by running the following command:

```
mmadddisk gpfs -F nsd_file
```

4. Create the `object_acfs` dependent fileset for account and container data by running the following command:

```
mmcrfileset gpfs object_acfs --inode-space objectfs
Fileset object_acfs created with id 3 root inode 1085441.
```

5. Link the new dependent fileset as `ac` under the `objectfs` independent fileset by running the following command:

```
mmlinkfileset gpfs object_acfs -J /gpfs0/objectfs/ac
Fileset object_acfs linked at /gpfs0/objectfs/ac
```

6. Create a placement.policy file. Here is an example:

```
/* policy rules to place account and container files into poolAC */

/* PLACEMENT policy 1 - put account and container files in poolAC
poolAC is previously created storage pool and object_acfs is previously created
dependent fileset */

RULE 'db files' SET POOL 'poolAC' FOR FILESET('object_acfs')

/* Default placement policy rule */
RULE 'default' SET POOL 'system'
```

7. To test the policy, run the following command:

```
mmchpolicy gpfs placement.policy -I test
Validated policy 'placement.policy': Parsed 2 policy rules.
```

8. If no errors were detected, apply the policy by running the following command:

```
mmchpolicy gpfs placement.policy -I yes
Validated policy 'placement.policy': Parsed 2 policy rules.
Policy 'placement.policy' installed and broadcast to all nodes.
```

Note: If you copied any existing account and container data to a temporary location, copy it back to the new ac directory.

5.5 Process monitoring

In the context of this paper, no process monitoring is provided. We suggest that you consider the monitoring approaches suggested by OpenStack, which are available at the OpenStack website:

http://docs.openstack.org/openstack-ops/content/logging_monitoring.html

Note: Logging is not meant to be turned on in a production system or on a continual basis.

Swift services rely on the GPFS file system that is mounted and available on all GPFS Object Nodes. If Spectrum Scale is stopped for some reason while Swift is running, certain Swift services might exit abnormally. In this case, you must restart all Swift services when Spectrum Scale is restored, either individually or by using the `manage_swift_services.sh` sample script that is provided with the additional materials (see Appendix A, “Additional material” on page 57). For more information about this script, see 5.1, “Managing Swift services” on page 36.

5.6 Security-Enhanced Linux considerations

If Security-Enhanced Linux (SELinux) is enabled in your environment, you must perform the following tasks.

Note: Enabling Spectrum Scale with SELinux affects the backup operations.

1. Install the OpenStack SELinux package, which configures Swift services for SELinux, by running the following command. This task must be done on each GPFS Object Node.
2. Set the Swift data type on the GPFS directory on any one of the GPFS Object Nodes by completing the following steps. This task must be done any time that the fileset is re-created:

```
yum install openstack-selinux
```

- a. Create a file context to associate the `swift_data_t` to your swift path by running the following command:

```
semanage fcontext -a -t swift_data_t '<gpfs_object_base>(/.*)?'
```

- b. Update the SELinux security context for the fileset by running the following command:

```
restorecon -R <gpfs_object_base>
```

For example, if your GPFS mount point is `/gpfs0` and the object independent fileset is `objectfs`, the commands are the following ones:

```
semanage fcontext -a -t swift_data_t '/gpfs0/objectfs(/.*)?'
```

```
restorecon -R /gpfs0/objectfs
```

Note: You can ignore any “Operation not permitted” errors for the fileset `.snapshots` directory that follow the `restorecon` command.

3. Swift uses `rsync` for replication, which depends on the file context being correct on the fileset. You must verify that each node can successfully run `rsync` after making these changes. Run `rsync` to copy a test file to each node before using Swift to ensure that the permissions are correct. To do so, run the following commands that are shown in this example:

```
rm -f <gpfs_object_base>/testfile
rsync /etc/motd hostname::object/testfile
ls -l <gpfs_object_base>/testfile
```

5.7 Port security considerations

The Swift object, container, and account servers are configured to use ports 6200, 6201, and 6202. These ports must be reachable by the proxy server on each GPFS Object Node. In a production deployment, these ports must be blocked so that they cannot be accessed from non- GPFS Object Nodes.

Note: The Swift object, container, and account servers do not authenticate requests that they receive. For this reason, it is critical to ensure that firewall policies are set up to protect these ports from outside access in a production deployment.

For a listing of OpenStack default ports, see the “Firewalls and default ports” topic at the OpenStack website:

<http://docs.openstack.org/juno/config-reference/content/firewalls-default-ports.html>

5.8 Configuring rsync to limit host access

To limit the hosts that can interact with your **rsync** environment, add the list of all the GPFS Object Nodes to the **hosts allow** configuration parameter in the `/etc/rsyncd.conf` file. This task must be done on each node and updated any time that a node is added to or deleted from the environment. Here is an example:

```
hosts allow = 192.167.11.10, 192.167.11.11
```

This line allows the two specified hosts to connect to **rsync**, and all others are rejected. For more information about configuring **rsync**, see the “rsyncd.conf” topic at the OpenStack website:

<http://rsync.samba.org/ftp/rsync/rsyncd.conf.html>

5.9 Virtual Network Computing port conflict

In our test environment, we observed cases where a Virtual Network Computing (VNC) server created a conflict with the object, container, and account servers when using the default port values of 6000, 6001, and 6002.

If you see port conflict issues when starting your Swift services, a VNC server might be the cause. The installer uses ports 6200, 6201, and 6202 instead of the Swift defaults, so it is unlikely that you will see this problem.

5.10 Software maintenance

After you install and configure Spectrum Scale Object, it is a preferred practice to keep both Spectrum Scale and OpenStack software at the latest levels. Information about the Spectrum Scale updates can be found at the GPFS frequently asked questions (FAQ) website:

http://www.ibm.com/support/knowledgecenter/SSFKCN/com.ibm.cluster.gpfs.doc/gpfs_faqs/gpfsclustersfaq.html

OpenStack updates are listed on the OpenStack website:

<https://wiki.openstack.org/wiki/Releases>



Swift feature overview

An advantage of Swift over other object stores is its rich set of features and its extensible design, which allows users to easily add new middleware for their applications. For more information, see the “Middleware and Metadata” topic on the OpenStack website:

http://docs.openstack.org/developer/swift/development_middleware.html

There are far too many features and middleware to fully test, and more are added regularly, so this chapter describes the set of features that are currently tested to help ensure their general stability and correctness. For more information, see the “Middleware” topic on the OpenStack website:

<http://docs.openstack.org/developer/swift/middleware.html>

This paper describes the following Swift features that have been tested:

- ▶ **Quotas:** Swift quotas allow a specific amount of disk capacity to be allocated to either containers or accounts by using Swift quotas. They also allow a limit on the maximum number of objects to be specified for containers or accounts.

Note:

- ▶ Due to Swift eventual consistency semantics, these limits might be exceeded in certain circumstances.
- ▶ Although GPFS quotas do not explicitly interact with Swift quotas, it still might be useful to employ GPFS quotas to limit the amount of space or the number of inodes that is consumed by the object store. To do this task, define GPFS quotas on the top-level independent fileset by specifying the maximum size or maximum inode usage that the object store can consume.

For more information about GPFS quotas, see the following website:

http://www.ibm.com/support/knowledgecenter/SSFKCN_4.1.0.4/com.ibm.cluster.gpfs.v4r104.gpfs100.doc/blladm_edquot.htm

- ▶ **Access control lists (ACLs):** Swift ACLs and the Keystone authentication server enable user-based and role-based access to be configured for accounts, containers, and individual objects.

Note: Do not use GPFS ACLs in the Object independent fileset.

- ▶ **Very large object support:** The maximum single object size that has been tested is 5 TB. This is much larger than the typical Swift object size limit (5 GB) and is possible because of the following conditions:
 - Load balancing across GPFS Object Nodes to increase throughput is no longer necessary because each GPFS Object Node can read and write by using the full storage network bandwidth (instead of needing to share the bandwidth with the creation of one or more object replicas).
 - Object placement imbalance is not an issue with Spectrum Scale because of its shared storage architecture.

- ▶ **Swift object segmentation** can be used to store arbitrarily large objects in the object store. For more information, see the “Large Object Support” topic on the OpenStack website:

http://docs.openstack.org/developer/swift/overview_large_objects.html

- ▶ **Versioning:** Normally, when an object is updated, the new object replaces the existing object in the object store. Swift versioning can be used to retain multiple versions of an object, which are automatically created upon object update. For more information, see the “Object Versioning” topic on the OpenStack website:

http://docs.openstack.org/developer/swift/overview_object_versioning.html

- ▶ **Swift3 (S3 compatibility)** middleware can be used to allow your object store to interoperate with applications that use the S3 protocol. For information about on how to configure this middleware, see the “Configure Object Storage with the S3 API” topic on the OpenStack website:

http://docs.openstack.org/juno/config-reference/content/configuring-openstack-object-storage-with-s3_api.html

For a list of the supported S3 operations, see the “Swift/APIFeatureComparison” topic at the following website:

<https://wiki.openstack.org/wiki/Swift/APIFeatureComparison>



Backup and restore

Enterprise data requires proven and robust data management tools to ensure that data is not lost due to user errors, software errors, or even hardware failures and errors. Generic OpenStack Swift replication protects against hardware failures, but it is not an end-to-end solution allowing recovery of data that has been incorrectly modified by a user, software application, or in specific cases, hardware corruption. Object versioning can assist with this situation, but it was not designed for this purpose. For example, there is no way to instantly create a version of the entire object store; older versions cannot be deleted without deleting the latest version; and it does not include account or container database information.

You can use Spectrum Scale Object to create consistent point-in-time copies of the object store to protect against data loss or corruption. These copies can be either created and stored instantly within Spectrum Scale by using snapshots, or stored in an external mass capacity storage system by using backups.

7.1 GPFS independent filesets

Storing the account, container, and object data in a single GPFS independent fileset is required because it provides a way to manage the entire object store. The object store independent fileset might contain one or more dependent filesets. For more information, see “Information Lifecycle Management for GPFS - Filesets” in the *GPFS 4.1.0.4: Advanced Administration Guide*, SC23-7032, found at the following website:

http://www.ibm.com/support/knowledgecenter/SSFKCN_4.1.0.4/com.ibm.cluster.gpfs.v4r104.gpfs200.doc/b11adv_filesets.htm

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

7.2 Snapshots

GPFS fileset *snapshots* provide a method of creating instant point-in-time copies of the entire object store, including the account and container databases. By using fileset snapshots instead of file system snapshots, the impact of the snapshot (which requires dirty data to be flushed to disk) does not affect applications that use files that are not in the fileset. Snapshot data can be accessed through the `.snapshots` directory in the fileset.

Fileset snapshots can be used for quickly capturing periodic object store contents. For information about saving data outside of the object cluster to protect it against disaster scenarios, see 7.3, “Backing up and restoring the object store” on page 46.

For more information about snapshots, see the `mmcrsnapshot` command in the *GPFS 4.1.0.4: Administration and Programming Reference*, SA23-1452, found at the following website:

http://www.ibm.com/support/knowledgecenter/SSFKCN_4.1.0/com.ibm.cluster.gpfs.v4r1.gpfs100.doc/b11adm_mmcrsnapshot.htm

7.3 Backing up and restoring the object store

Snapshots are a good way to protect data from various errors and failures. Moving them to a separate backup storage system can provide better protection against catastrophic failures of the entire storage system and might even allow the data to be stored at a lower cost.

This section describes the manual steps that are needed to back up the object store and its configuration information.

Note: In our example, we do not back up the Keystone configuration files and database. This task is the responsibility of the user. You can use OpenStack backup procedures for this task. For more information, go to the following OpenStack website:

http://docs.openstack.org/openstack-ops/content/backup_and_recovery.html

To improve usability, we also provide two sample scripts to help automate the backup and recovery process:

- ▶ `eso_backup.sh`
- ▶ `eso_restore.sh`

The scripts are also described in this section.

Note: The Spectrum Protect backup-archive client must be installed, at the same version, on all of the nodes that are running the `mmbackup` command. For more information about Spectrum Protect requirements for the `mmbackup` command, see “Tivoli Storage Manager requirements” in *GPFS: Administration and Programming Reference*, SA23-1452, which can be found at the following website:

http://www.ibm.com/support/knowledgecenter/SSFKCN_4.1.0.4/com.ibm.cluster.gpfs.v4r104.gpfs100.doc/blladm_tsmreqs.htm

7.3.1 Backup procedure

The backup procedure consists of the following steps:

1. Storing all relevant configuration data in a safe location outside of your GPFS cluster. This data is essential to restoring your object store quickly, so you might want to store it in a site in a different geographical location for added safety.
2. Creating a file system (global) snapshot
3. Creating a snapshot-based backup by running `mmbackup`.

All GPFS Object Nodes and Spectrum Protect client nodes must be available with the object file system mounted on each one when the backup is being created. The Spectrum Protect server also must be available.

A global or file system snapshot is used in this procedure because the `mmbackup` command's smallest granularity currently is a file system snapshot.

Saving configuration data

To recover the object store in a disaster recovery case, the following configuration data must be backed up:

- ▶ Swift-related configuration files and content
- ▶ The GPFS fileset configuration that is used by the object store
- ▶ Spectrum Protect client configuration files

This configuration information should be stored in a safe location that is outside of your GPFS cluster environment. This configuration data is needed for the recovery of the GPFS independent fileset that is used for object data and Spectrum Protect client configuration. By having the configuration data readily re-created, the object store contents can be restored quickly from the Spectrum Protect server.

Note: Make sure to preserve the permissions or ACLs of the configuration files that are backed up or copied. This is essential to restoring from a disaster and starting the system without issues. Also, make sure that all Swift nodes are accessible during backup.

Manual backup procedure

To perform the backup procedure manually, complete the following steps:

1. Run the following command to extract the GPFS file system and fileset configuration and store the output file in a safe location outside of your GPFS cluster:

```
mmbackupconfig <file system device> -o <output file>
```

Here is an example of using this command with a device name of `gpfs` and an output file name of `gpfs_object.cfg`:

```
mmbackupconfig gpfs -o gpfs_object.cfg
```

2. Save the following Spectrum Protect configuration files for each Spectrum Protect client node in the same safe location that is used in step 1 on page 47:

- a. `/etc/adsm/TSM.PWD`

Contains the client password that is required to access the Spectrum Protect. Depends on the Spectrum Protect server setting of authentication. This file is present only when authentication is set to "on".

- b. `/opt/tivoli/tsm/client/ba/bin/dsm.sys` and
`/opt/tivoli/tsm/client/ba/bin/dsm.opt`:

These are the Spectrum Protect client configuration files.

3. Save the Swift configuration files for each GPFS Object Node in the same safe location that is used in step 1 on page 47:

- a. `/etc/swift`

This directory tree contains the Swift configuration files, the ring builder files (`account.builder`, `container.builder`, and `object.builder`) and ring files (`account.ring.gz`, `container.ring.gz`, and `object.ring.gz`). The contents of this directory and all subdirectories must be saved. The ring information is the same on each GPFS Object Node, but the Swift server configuration files have different settings.

- b. `/root/openrc`

This file contains OpenStack account credentials that are stored for convenience. This file can be easily re-created, but for convenience, save it.

4. Back up the object store content to a Spectrum Protect server by running `mmbackup` (based on a snapshot):

- a. Create a global snapshot by running the following command:

```
mmcrsnapshot <file system device> <snapshot name>
```

For example, create a snapshot that is named `objects_globalsnap1` by running the following command:

```
mmcrsnapshot gpfs objects_globalsnap1
```

- b. Create global and local work directories by running, for example, the following commands:

```
mkdir -p /gpfs0/.es/mmbackupglobal  
mkdir -p /gpfs0/.es/mmbackuplocal
```

- c. Start the snapshot-based backup job by running the following command (the line wrap is indicated by '\')

```
mmbackup <file system device> -t incremental -N <TSM client nodes> \  
-g <global work directory> \  
-s <local work directory> \  
-S <global snapshot name> --tsm-servers <tsm server> --noquote
```

For example:

```
mmbackup gpfs -t incremental -N 10.10.10.1,10.10.10.2 \  
-g /gpfs0/.es/mmbackupglobal \  
-s /gpfs0/.es/mmbackuplocal \  
-S objects_globalsnap1 --tsm-servers tsm1 --noquote
```

Where:

- **-N** specifies the nodes that are involved in the backup process. These nodes must be configured for the Spectrum Protect server that is being used.
- **-S** specifies the global snapshot name to be used for the backup. In our example, we use the same snapshot that was created in step a on page 48.
- **--tsm-servers** specifies which Spectrum Protect server is used as the backup target, which refers to the Spectrum Protect server name that is used in the Spectrum Protect client configuration `dsm.sys` file.

There are several other options that are available that can influence the backup process and its velocity or system load. For example, you can increase the number of backup threads per node by using the **-m** parameter. Check the man page of the **mmbackup** command for the full list of available parameters.

Note: Do not unlink any fileset that is part of the file system while the backup is occurring.

- d. Remove the snapshot that was previously created in step a on page 48 by running the following command:

```
mmdelsnapshot <file system device> <snapshot name>
```

For example:

```
mmdelsnapshot gpfs objects_globalsnap1
```

7.3.2 Restore procedure

After you have performed the prerequisite procedures, you can begin the recovery procedure.

Prerequisites

You must meet the following prerequisites to being the recovery procedure:

1. Restore the GPFS cluster with the same node names that were used previously.
2. Re-create the file system that was used for Swift.
3. Restore your OpenStack Keystone server and make sure that it is operational.
4. Install Swift software on all GPFS Object Nodes. You can accomplish this task by using the **esobject** installer that is described in Chapter 4, “Spectrum Scale Object installation” on page 23 and Chapter 5, “System administration considerations” on page 35.
5. Install the Spectrum Protect backup-archive client software on the GPFS Object Nodes that were clients previously.
6. All GPFS Object Nodes and Spectrum Protect client nodes must be available when the object store configuration and content are being restored.

Recovery procedure

To perform the recovery procedure, complete the following steps:

1. Stop the Swift services on all GPFS Object Nodes. The Swift service management script, which is described in 5.1, “Managing Swift services” on page 36, can be used for this task:

```
manage_swift_services.sh stop
```

2. Unmount the object file system on all GPFS Object Nodes.
 - For example, using the file system that is named gpfs, run the following command:


```
mmunmount <file system device> -a
```
 - For example, using the file system that is named gpfs, run the following command:


```
mmunmount gpfs -a
```
3. Restore the GPFS configuration by running the **mmrestoreconfig** command. The input to the **mmrestoreconfig** command is the file that was created previously by the **mmbackupconfig** command.


```
mmrestoreconfig <file system device> -i <output_file> -I yes -Q no
```

For example, reusing the gpfs_object.cfg output file that is created above, run the following command:

```
mmrestoreconfig gpfs -i gpfs_object.cfg -I yes -Q no
```

You can create a report of the expected configuration to review before restoring by running the following example command:

```
mmrestoreconfig gpfs -i gpfs_object.cfg -F report.out
```
4. Mount the object file system on all nodes by running the following command:


```
mmmount <file system device> -a
```

For example, using the file system that is named gpfs, run the following command:

```
mmmount gpfs -a
```
5. Restore the configuration of the Spectrum Protect client nodes by copying the saved configuration files from their saved location to each Spectrum Protect client node.
 - a. The Spectrum Protect client config files dsm.opt and dsm.sys should be restored to /opt/tivoli/tsm/client/ba/bin/.
 - b. The Spectrum Protect client password file, TSM.PWD, if saved during the backup procedure, should be restored to /etc/adsm/.
 - c. Verify that each Spectrum Protect client node can communicate with the Spectrum Protect server without prompting for a password by running the following command:


```
dsmc q sess
```
6. Restore the Swift configuration files by copying them from their location to /etc/swift on each GPFS Object Node.

Ensure that the ownership and permission settings of these configuration files are correct by running the **cp -a** command or by explicitly setting the ownership after the copy is completed by running the following command:

```
chown -R swift:swift /etc/swift
```
7. Restore the object store data from the Spectrum Protect server.
 - a. On a Spectrum Protect client node, start a no-query restore by running the **dsmc restore** command:


```
dsmc restore <GPFS Object path> -subdir=yes -disablenqr=no \
          -servername=<tsm server> -errorlogname=<error log path>
```

For example:

```
dsmc restore /gpfs0/objectfs/ -subdir=yes -disablenqr=no \
          -servername=tsm1 -errorlogname=/tmp/object_restore.log
```
 - b. When all restore jobs are completed, check the error logs. If any errors are found, correct them until all restore operations complete successfully.

8. On each GPFS Object Node, start the Swift services by running the following command:


```
manage_swift_services.sh start
```
9. Verify that basic Swift commands (**swift stat** and **swift list**) return without error, in the same manner that is described in 4.5.1, “Verifying the installation” on page 33. Also, verify that the number of containers and the number of objects within those containers are as expected.

Improving recovery time

The command that is shown in step 7a on page 50 starts a single restore job on a single node. This job might require a long period to restore all of your object data. To improve the restore performance, you can start separate restore jobs on different Spectrum Protect client nodes. To do this task, you must split the overall restore task into several smaller ones. One way to do this task is to specify your restore path for object data that is deeper in the GPFS object path.

For example, instead of starting the restore with the root of the GPFS Object path, start the object restore at the virtual devices level. If you have 40 virtual devices that are configured, you might start 40 independent restore jobs to restore the object data, and distribute the jobs to the different Spectrum Protect client nodes. Additionally, you start a single restore job for all of the files under the account and container path.

With this approach, care must be taken to not overload the Spectrum Protect client nodes or the Spectrum Protect server. You might want to experiment to determine the best mix of jobs.

For example, if there are four GPFS Object Nodes, each with the Spectrum Protect client installed and configured, you might use the following types of commands.

1. On the first GPFS Object Node, run a restore job for each of the first 10 virtual devices by running the following commands:

```
dsmc restore /gpfs0/objectfs/o/z1device0 -subdir=yes -disablenqr=no \
-serversname=tsml
dsmc restore /gpfs0/objectfs/o/z1device1 -subdir=yes -disablenqr=no \
-serversname=tsml
```

```
#<repeat for z1device2 - z1device9>
```

2. On the second node, run a restore job for each of the next 10 virtual devices. Continue the pattern on the remaining GPFS Object Nodes so that all of the virtual devices under the o subdirectory are restored. Also, start a single restore job for all of the account and container data under the ac subdirectory by running the following command:

```
dsmc restore /gpfs0/objectfs/ac -subdir=yes -disablenqr=no -serversname=tsml
```

The most efficient restore approach depends on many factors, including the number of tape drives, Spectrum Protect client configuration, and network bandwidth. You might need to experiment with your configuration to determine the optimal restore strategy.

7.3.3 Automating the backup and restore procedures

This section describes two sample scripts that are provided for automating the backup and restore procedures. The installer places these scripts in the `$HOME/eso_samples/backup` directory on each GPFS Object Node. The exact location is shown here:

```
$HOME/eso_samples/backup/eso_backup.sh  
$HOME/eso_samples/backup/eso_restore.sh
```

Setup and notes

Here are some considerations for automating the procedures:

- ▶ All GPFS Object Nodes and Spectrum Protect client nodes must be available with the object fileset mounted during both the backup and restore operations. The Spectrum Protect server must also be available.
- ▶ The `eso_backup.sh` and `eso_restore` scripts depend on the `manage_services.sh` script to control the Swift services.
- ▶ Log files for each script invocation are stored here:
`/var/log/eso-backup-restore/`

Automating the backup procedure

Run the automated backup procedure by using the `eso_backup.sh` script as follows:

```
./eso_backup.sh <fs dev> <tsm server stanza> <object_base_path>
```

The `<fs dev>` parameter specifies the file system device that is backed up.

The `<tsm server stanza>` parameter specifies which Spectrum Protect server is used as the backup target.

The `<object_base_path>` parameter specifies the root location (usually an independent fileset) of the object store, for example, `/gpfs0/objectfs`.

An example invocation is shown here:

```
./eso_backup.sh gpfs tsm1 /gpfs0/objectfs
```

The script initiates a backup of the object store configuration and data to the specified Spectrum Protect server.

The script collects the required configuration information and stores it as a compress file in two places:

1. `/tmp/eso_config_dump/eso_config_<timestamp>.tgz` on the node where the script is run.
For example:
`/tmp/eso_config_dump/eso_config_20141112-115012.tgz`
2. `<FS mountpoint>/.eso_config_dump/eso_config_<timestamp>.tgz` in the shared file system hosting the object store. For example:
`/gpfs0/.eso_config_dump/eso_config_20141112-115012.tgz`

This file serves as input for the corresponding recovery script if there is a disaster, and should be stored in a safe and easy accessible location.

When the script completes, carefully review the log file output. If any issues are found, these must be resolved and, if necessary, you must run the backup script again.

Note: Log files and the saved configuration files that are generated by this script are uniquely named, including the execution time. It is the user's responsibility to manage these files by deleting older files that are no longer needed.

Automating the restore procedure

Before running `eso_restore.sh`, you must unmount the object store file system from all GPFS Object Nodes by running the `mmunmount` command:

```
mmunmount <fs device> -a
```

For example:

```
mmunmount gpfs0 -a
```

Run the automated restore procedure by running the `eso_restore.sh` script as follows:

```
./eso_restore.sh <eso_backup_set> [-nr]
```

The `<eso_backup_set>` parameter is the configuration file of a previously taken backup, for example, `eso_config_20141112-115012.tgz`.

An example single node invocation is shown here:

```
./eso_restore.sh eso_config_20141112-115012.tgz
```

This invocation restores the previous object store configuration to all GPFS Object Nodes and also restores the Spectrum Protect client configuration. Use that configuration to restore the object store content with a single restore job.

The `[-nr]` (no restore) parameter can be specified to display the `restore` command but not run that command. The administrator can then manually split up the single restore job into multiple jobs that can be started in parallel on all Spectrum Protect client nodes for improved restore performance, as described in “Improving recovery time” on page 51.

Note: Log files that are generated by this script are uniquely named, including the execution time. It is the user's responsibility to manage these files by deleting older files that are no longer needed.



Summary

This chapter summarizes further areas of investigation and concludes with how Spectrum Scale Object solves business problems by providing a new type of cloud storage.

8.1 Future investigation

This paper focuses on the general architecture and setup of Spectrum Scale Object. To realize the vision that was described earlier in this paper, several further steps are needed, which include development and test validation.

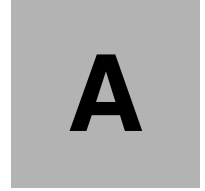
Investigation includes the following information:

- ▶ Integration of external tape pools to further decrease the cost per gigabyte.
- ▶ Suggestions for preferred practice use of Swift global clusters, Swift container synchronization, and Spectrum Scale Active File Management for disaster recovery and data distribution.
- ▶ Integration of file and object access to a single data set. This investigation includes many challenges due to the stark difference between POSIX and Swift object semantics, but the goal is to work with the community to provide a coherent and easy-to-use system for file and object access. See “SwiftOnFile” at this website:
<https://github.com/swiftonfile/swiftonfile>
- ▶ Performance optimizations through the DiskFile pluggable on-disk back end. See “Pluggable On-Disk Back-end APIs” at this website:
http://docs.openstack.org/developer/swift/development_ondisk_backends.html
- ▶ Removing the requirement to replicate account and container information by ensuring the correct update of account and container database information from multiple nodes.
- ▶ The ability to search for objects in a container or account that have specific metadata values.
- ▶ The ability to designate objects as immutable for a time duration, so that they cannot be updated or deleted during that time.
- ▶ Quality assurance of the following areas:
 - Swift expiration
 - Spectrum Scale encryption
 - File access gateways:
 - See Cyberduck at this website:
<http://cyberduck.io>
 - See Maldivica at this website:
<http://www.maldivica.com/>

Note: Any references to future investigation are for planning purposes only. IBM reserves the right to change those plans at its discretion. Any reliance on such a disclosure is solely at your own risk. IBM makes no commitment to provide additional information in the future.

8.2 Conclusion

IBM Spectrum Scale is a proven, enterprise-class file system, and OpenStack Swift is a best-of-breed object-based storage system. Spectrum Scale Object combines these technologies to provide a new type of cloud storage that includes efficient data protection and recovery, proven scalability, and performance, snapshot and backup and recovery support, and information lifecycle management. Through these features, Spectrum Scale Object can help simplify data management and allow enterprises to realize the full value of their data.



Additional material

This paper refers to additional material that can be downloaded from the Internet as described in the following sections.

Locating the web material

The web material that is associated with this paper is available in softcopy on the Internet from the IBM Redbooks web server. Point your web browser at:

<ftp://www.redbooks.ibm.com/redbooks/REDP5113>

Alternatively, you can go to the IBM Redbooks website at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the IBM Redpaper form number, REDP5113.

Using the web material

The additional web material that accompanies this paper includes the following files:

<i>File name</i>	<i>Description</i>
disclaimer.txt	Disclaimer information
esobject-installer-1.1.x.tgz	Installation tool

Note: *x* represents the level of the code that is available at the IBM Redbooks web server. Update the *x* to match the current level in the REDP5113 folder.

Downloading and extracting the web material

Create a subdirectory (folder) on your workstation, and extract the contents of the web material `esobject-installer-1.1.x.tgz` file into this folder.

Related publications

The publications that are listed in this section are considered suitable for a more detailed discussion of the topics that are covered in this paper.

IBM Redbooks

The following IBM Redbooks publications provide additional information about the topic in this document. Some publications referenced in this list might be available in softcopy only.

- ▶ *Adding an IBM LTF5 EE Tape Tier to an IBM SC5A Managed Storage Cloud*, TIPS1129
- ▶ *IBM Private, Public, and Hybrid Cloud Storage Solutions*, REDP-4873

You can search for, view, download, or order these documents and other Redbooks, Redpapers, Web Docs, draft and additional materials, at the following website:

ibm.com/redbooks

Other publications

These publications are also relevant as further information sources:

- ▶ *GPFS: Administration and Programming Reference*, SA23-1452
- ▶ *GPFS: Advanced Administration Guide*, SC23-7032
- ▶ *GPFS: Problem Determination Guide*, GA76-0443

Online resources and references

- ▶ “GPFS-based implementation of a hyper-converged system for software defined infrastructure” by Azagury, et al, in IBM Journal of Research and Development 58 (2), 1-12, 2014
- ▶ IBM Spectrum Scale
<http://www.ibm.com/systems/storage/spectrum/scale/index.html>
- ▶ IBM GPFS Storage Server (Elastic Storage Server) Announcement
http://www.ibm.com/common/ssi/rep_ca/9/897/ENUS114-149/ENUS114-149.PDF

IBM Knowledge Center topics:

- ▶ “General Parallel File System”
http://www.ibm.com/support/knowledgecenter/SSFKCN/gpfs_welcome.html
- ▶ *GPFS 4.1.0.4: Administration and Programming Reference*, SA23-1452
http://www.ibm.com/support/knowledgecenter/SSFKCN_4.1.0.4/com.ibm.cluster.gpfs.v4r104.gpfs100.doc/bl1adm_top.htm

- ▶ *GPFS 4.1.04: Concepts, Planning, and Installation*, GA76-0441
http://www.ibm.com/support/knowledgecenter/SSFKCN_4.1.0.4/com.ibm.cluster.gpfs.v4r104.gpfs300.doc/blins_top.htm
- ▶ “GPFS File Placement Optimizer” in *GPFS 4.1.0.4: Advanced Administration Guide*, SC23-7032
http://www.ibm.com/support/knowledgecenter/SSFKCN_4.1.0.4/com.ibm.cluster.gpfs.v4r104.gpfs200.doc/bl1adv_fposettings.htm
- ▶ GPFS Frequently Asked Questions (FAQ)
<http://www.ibm.com/support/knowledgecenter/SSFKCN/gpfs4104/gpfsclustersfaq.html>
- ▶ Elastic Storage Server planning and service information
http://www.ibm.com/support/knowledgecenter/POWER8/p8ehc/p8ehc_storage_landing.htm
http://www.ibm.com/common/ssi/cgi-bin/ssialias?subtype=SP&infotype=PM&appname=S TGE_DC_ZQ_USEN&htmlfid=DCD12377USEN&attachment=DCD12377USEN.PDF#loaded
- ▶ GPFS Native RAID information
http://www.ibm.com/support/knowledgecenter/SSYSP8_2.5.0/sts25_welcome.html
http://www.ibm.com/support/knowledgecenter/SSFKCN_4.1.0/com.ibm.cluster.gpfs.v4r1.gpfs200.doc/bl1adv_introduction.htm

IBM developerWorks topics:

- ▶ “Configuring GPFS for Reliability”
[https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/General%20Parallel%20File%20System%20\(GPFS\)/page/Configuring%20GPFS%20for%20Reliability](https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/General%20Parallel%20File%20System%20(GPFS)/page/Configuring%20GPFS%20for%20Reliability)
- ▶ “File System Planning”
[https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/General%20Parallel%20File%20System%20\(GPFS\)/page/File%20System%20Planning](https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/General%20Parallel%20File%20System%20(GPFS)/page/File%20System%20Planning)
- ▶ “Tuning Parameters”
[https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/General%20Parallel%20File%20System%20\(GPFS\)/page/Tuning%20Parameters](https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/General%20Parallel%20File%20System%20(GPFS)/page/Tuning%20Parameters)

OpenStack topics:

- ▶ “Authentication”
<http://docs.openstack.org/api/openstack-object-storage/1.0/content/authentication-examples-curl.html>
- ▶ “The Auth System”
http://docs.openstack.org/developer/swift/overview_auth.html#keystone-auth
- ▶ “Firewalls and default ports”
<http://docs.openstack.org/juno/config-reference/content/firewalls-default-ports.html>
- ▶ Chapter 1. Introduction to OpenStack High Availability
<http://docs.openstack.org/high-availability-guide/content/ch-intro.html>
- ▶ Chapter 9. Object Storage command-line client
http://docs.openstack.org/cli-reference/content/swiftclient_commands.html

- ▶ “Cluster Telemetry and Monitoring”
http://docs.openstack.org/developer/swift/admin_guide.html#cluster-telemetry-and-monitoring
- ▶ “Controller Setup”
http://docs.openstack.org/juno/install-guide/install/yum/content/ch_basic_environment.html#basics-database
- ▶ “General System Tuning”
http://swift.openstack.org/deployment_guide.html#general-system-tuning
- ▶ “Middleware”
<http://docs.openstack.org/developer/swift/middleware.html>
- ▶ “Middleware and Metadata”
http://docs.openstack.org/developer/swift/development_middleware.html
- ▶ “Network Time Protocol (NTP)”
<http://docs.openstack.org/icehouse/install-guide/install/yum/content/basics-ntp.html>
- ▶ “Object Versioning”
http://docs.openstack.org/developer/swift/overview_object_versioning.html
- ▶ “OpenStack Installation Guide for Red Hat Enterprise Linux 7, CentOS 7, and Fedora 20”
<http://docs.openstack.org/juno/install-guide/install/yum/content/>
- ▶ “OpenStack Packages”
http://docs.openstack.org/juno/install-guide/install/yum/content/ch_basic_environment.html#basics-packages
- ▶ “Pluggable On-Disk Back-end APIs”
http://docs.openstack.org/developer/swift/development_ondisk_backends.html
- ▶ “Verify the Identity Service installation”
<http://docs.openstack.org/juno/install-guide/install/yum/content/keystone-verify.html>
- ▶ “Verify the installation”
<http://docs.openstack.org/juno/install-guide/install/yum/content/verify-object-storage-installation.html>

Other websites:

- ▶ Amazon Simple Storage Services (S3)
<http://aws.amazon.com/documentation/s3/>
- ▶ Cyberduck
<http://cyberduck.io>
- ▶ Maldivica
<http://www.maldivica.com/>
- ▶ rsyncd.conf
<http://rsync.samba.org/ftp/rsync/rsyncd.conf.html>
- ▶ SwiftOnFile
<https://launchpad.net/gluster-swift>

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services



A Deployment Guide for IBM Spectrum Scale Object



Learn why Spectrum Scale is an ideal environment for OpenStack Object Storage

Because of the explosion of unstructured data that is generated by individuals and organizations, a new storage paradigm called “Object Storage” has been developed. Object Storage stores data in a flat namespace that scales to trillions of objects. The design of object storage also simplifies how users access data, supporting new types of applications and allowing users to access data by various methods, including mobile devices and web applications. Data distribution and management are also simplified, allowing greater collaboration across the globe.

Plan an OpenStack Swift on Spectrum Scale deployment

OpenStack Swift is an emerging open source object storage software platform that is widely used for cloud storage. IBM Spectrum Scale (based on IBM General Parallel File System (GPFS) technology, which also is known formerly as code name Elastic Storage) is a high-performance and proven product that is used to store data for thousands of mission-critical commercial installations worldwide. (Throughout this IBM Redpaper publication, Spectrum Scale is used to refer to GPFS V4.1.)

Follow quick-start implementation steps

Spectrum Scale also automates common storage management tasks, such as tiering and archiving at scale. Together, Spectrum Scale and OpenStack Swift provide an enterprise-class object storage solution that efficiently stores, distributes, and retains critical data.

This paper provides instructions about how to set up and configure Swift with Spectrum Scale. It also provides an initial set of preferred practices to ensure optimal performance and reliability.

The goal of this paper is to describe the benefits of using Spectrum Scale as the underlying file system with OpenStack Swift, guide an administrator through the installation and configuration of Spectrum Scale Object, and describe the general set of configurations and scenarios that have been validated. It is intended for administrators who are familiar with Spectrum Scale and OpenStack Swift components.

**INTERNATIONAL
TECHNICAL
SUPPORT
ORGANIZATION**

**BUILDING TECHNICAL
INFORMATION BASED ON
PRACTICAL EXPERIENCE**

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

**For more information:
ibm.com/redbooks**