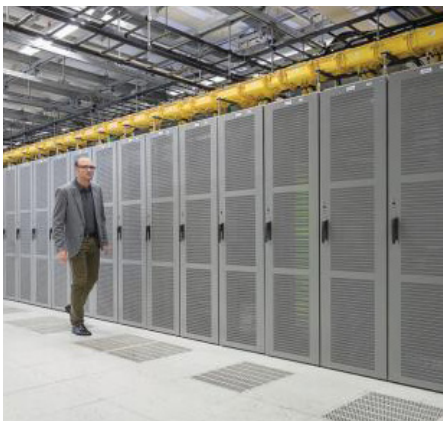
Solution Brief
OpenStack Swift
Intelligent Storage

Look Inside.™

OpenStack Swift 2.0: Storage Policies Open Up Broader Horizons

New, intelligent features in OpenStack Swift help meet today's storage demands



The OpenStack Swift community has released Swift 2.0, its first-ever major revision bump milestone. Swift 2.0 is available immediately and all of its key features and functionality are part of the OpenStack *Juno* release. The cornerstone feature: Storage Policies, one of the most important developments in OpenStack Swift since the project was open-sourced.

OpenStack software is used by hundreds of companies to build public, private, and hybrid clouds, and Swift object-based storage is a key element of OpenStack. Now, the addition of Storage Policies to Swift creates a whole new set of usage models for enterprise users, cloud operators, and application developers. It enables Swift to meet different storage requirements for performance, durability, and geographic location, while also ensuring container consistency.

Building on a Solid Foundation

Storage Policies are built on an already successful open-source storage model. By deploying Swift, organizations can say goodbye to traditional silos of hard-to-manage and difficult-to-scale storage systems. Swift works by abstracting the storage volumes to enable the scalability, availability, and global access that many applications now demand. Swift has a flexible architecture and is able to scale in two different directions. If the cluster needs to support more performance or simultaneous connections, then more proxy servers can be added in the Access Tier. If additional capacity is required instead, more storage nodes can be added in the Storage Tier (see Figure 1).

Swift follows the conventions of the Representational State Transfer (REST) style of network architecture, making it consistent with the most popular web-based applications. That means organizations can easily read and write their data over HTTP to a Swift cloud storage cluster located practically anywhere.

Removing Barriers for Users and Providers

Swift uses a replication framework to protect data against the failure of any one drive. Users specify some level of replication of objects—such as 2x or 3x—with the copies placed on different drives in the cluster. But until now, the user's entire cluster had to adhere to one setting; there was no way to set some nodes at 2x and others at 3x to meet different data protection requirements. There was also no efficient way to make use of differentiated hardware within the cluster such as nodes with newer or faster characteristics. Swift 2.0 removes these barriers by introducing Storage Policies.

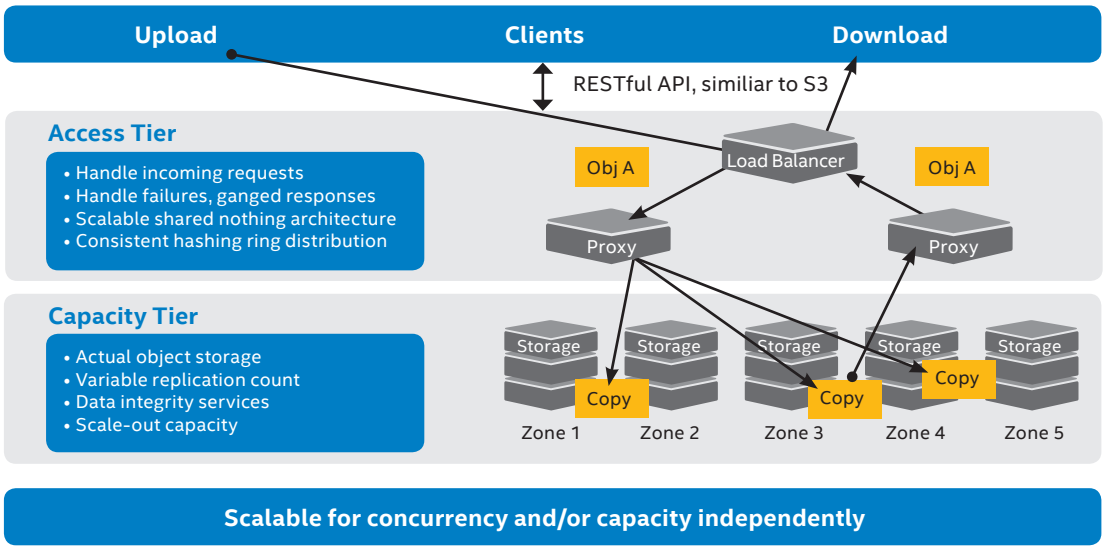


Figure 1: OpenStack Swift storage architecture

Storage Policies leverage the hashing ring technology that Swift already uses to determine where it should store and retrieve data within the cluster. The hashing process works by feeding each unique URL for an object into a hash algorithm, producing an index of values. Swift then uses this index to look up which nodes that particular URL and object should occupy in the data structure of the ring. A ring used for this data placement is called an *object ring*.

Where a Swift cluster previously supported only one object ring, now it can take advantage of many rings to specify data placement. That means a cluster can have multiple redundancy settings: the user can require triple replication on one ring and reduced (double) replication on another. Alternatively, a ring can be used to isolate specific performance features, such as faster storage nodes with solid-state drives (SSDs). Rings can also be used to assure geographic containment of specific drives within a large cluster to meet policy and governance requirements. These rings are built in the same way the original object ring was built before Storage Policies were introduced. When cluster administrators want to build a policy for low latency, for example, they simply specify only SSDs for that policy's ring. When an incoming request looks up where an object should be placed and uses the low-latency policy, it will query the ring associated with the low-latency policy, and that ring will place the object only on SSDs.

Capitalizing on Swift Containers

Another addition to Swift 2.0 builds on the existing Swift *container* concept. Swift allows for one level of hierarchy: a root directory and one level of subdirectories. A container is a subdirectory, and each container can be associated with a different Storage Policy. When a container is created with a particular policy, all objects stored in that container will then be handled according to that policy. This capability provides significant advantages for Swift service providers, who can easily use the containers and Storage Policies to offer differentiated services based on data placement and protection.

In Swift 2.0, a new daemon called the *container reconciler* has been added to Swift to handle error scenarios. Swift is what is called an "eventually consistent" system—it favors availability over consistency. That means an update to a file is available immediately on some storage servers, but may take time to propagate to other servers throughout a system.

For example, a network outage might prevent the update from getting to a particular server. When that particular machine came back on line, it would not immediately have the update. Later, after Swift realized that the server was up and running, the update would occur and all storage nodes would be consistent. Swift does not support "strongly consistent" implementations; if it did, the initial upload would fail and availability would suffer because the storage system would require consistent views of all nodes before considering the operation successful.

The "eventually consistent" nature of Swift is relevant for Storage Policies. Consider, for example, a scenario where an application creates a container with a storage policy, but a network outage prevents some nodes from knowing about it. Then suppose that another application, still able to get to the nodes that don't know about the first container, creates its own container with the same name but a different storage policy.

When the network outage is restored, there will be two containers of the same name in the cluster, each containing valid data but stored in a potentially very different manner on different nodes. The new Swift container reconciler is designed to search the cluster for situations like this and ensure that the data is stored according to the correct storage policy.

Introducing New Swift Use Cases

New usage models enabled by Swift Storage Policies include reduced redundancy options, performance tiers, and geographical tagging (see Figure 2).

Reduced Redundancy

Storage Policies can be created to enable different replication factors to be used in the same cluster, depending on the type of data that needs to be stored. For example, image thumbnails require less durability than other data. If the original resolution image is stored with three copies, then a resampled image can be stored with just two copies—any data loss is mitigated by the ability to re-create the thumbnail from the original.

Used at scale, this reduced-redundancy Storage Policy can save significant hard drive capacity, thereby lowering the user's cost and helping service providers attract and retain more cost-sensitive customers.

Performance Tier

Performance tiering enables users to specify storage servers with higher-performance SSDs for their most critical applications, while placing less performance-dependent or less frequently accessed data on conventional hard drives. With more choices, the user can more closely match the storage configuration to the application's requirements.

A Year in the Making

Swift Storage Policies were in development for a year, and are the product of broad collaboration across the OpenStack community—more than three dozen developers at 15 companies contributed. Lead contributing companies by lines of code include SwiftStack, Intel, Red Hat, IBM, Rackspace, HP, and multiple independent developers. The development process involved multiple informal face-to-face gatherings of small teams, lots of IRC discussion, and two separate full-team hackathons in Austin, Texas and Boulder, Colorado.

The new Storage Policy features are available now from the open-source repository and technology providers in the OpenStack ecosystem, and are integrated into the *Juno* release of the OpenStack software.

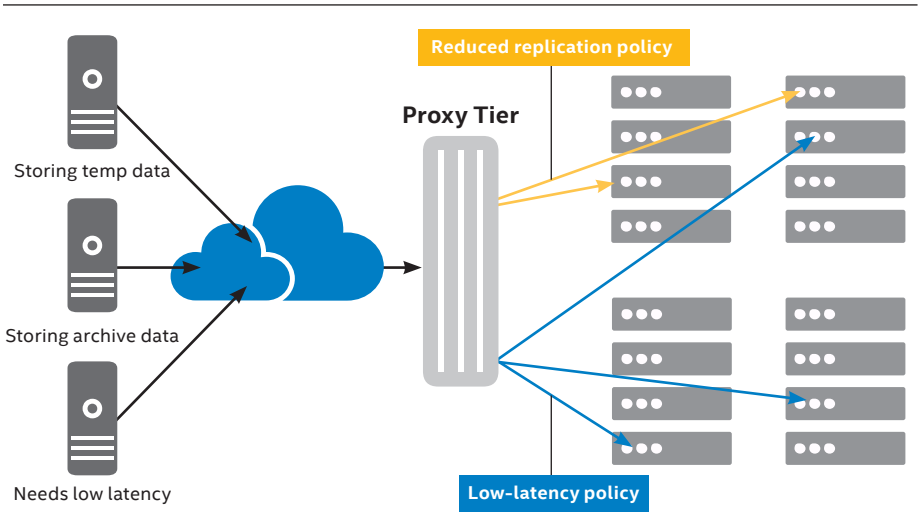


Figure 2: Configuration for reduced replication and low-latency policies

Building a new cluster entirely with SSDs is possible for the highest performance, but placing one or two SSDs in each node is an economical approach that allows the provider to offer different service levels at different rates. Alternatively, operators can make sure all drives in a policy map to a specific set of faster servers, or servers with special capabilities.

Geotagging

Swift users can also create a container Storage Policy that specifies a country, a region, or a specific data center. The user is then guaranteed that anything written to that container will stay within that geographical location, using only the physical machines residing there. Specifying location can be important for a variety of reasons, ranging from regulatory compliance and security concerns to simply placing data as close as possible to users. For example, with geotagging, a business can create policies to ensure branch offices have their own data locally available for fast lookup, while the headquarters office has a copy of everything that is being stored in the branches.

Meeting New Demands

With Swift 2.0 and Swift Storage Policies, organizations can better meet the requirements of modern, on-demand applications. They can specify how their data is stored in different geographical regions, and control replication policies that can improve the economy of different types of storage without compromising availability. Look for new Storage Policy capabilities, such as erasure coding, coming soon.

For More Information

To learn more about Swift and the new Swift Storage Policies, visit: www.SwiftStack.com

SwiftStack powers enterprises with a software-defined storage platform, built on the OpenStack Swift object storage engine.

SwiftStack

Printed in USA 1014/CM/TDA/XX/PDF ♻ Please Recycle 331194-001US