# CONTENTS

**INTRODUCTION**  
3

**SOLUTION OVERVIEW**  
3

**USE CASES**  
4

- Enterprise Backup  
4
- Active Archive (e.g., Media Archives, Genetic Sequencing)  
4
- Web Application Data  
4
- File Sync-and-Share  
4
- Metapod and Other Private Cloud Storage  
4

**CISCO UNIFIED COMPUTING SYSTEM**  
5

- Cisco UCS C220 Rack Server  
6
- Cisco UCS C240 Rack Server  
6
- Cisco UCS C3160/C3260 Rack Server  
7

**SWIFTSTACK OBJECT STORAGE SOFTWARE**  
7

- SwiftStack Object System Components  
7
- General SwiftStack Features  
8
- Controller  
9
- OpenStack Swift  
9

**USING SWIFTSTACK AND CISCO UCS TOGETHER**  
9

- Hardware Configuration  
9
- Networking  
12
- Installation and Software Configuration  
13

**INTEGRATION VERIFICATION TESTING**  
14

- Functional Tests  
14
- Performance Tests  
15

**CONCLUSION**  
18
INTRODUCTION

SwiftStack is designed to empower enterprises to harness the power of object storage in their own data centers. SwiftStack enables enterprise users to manage, deploy, scale, upgrade, and monitor single- and multi-site clusters using the OpenStack Swift object storage engine plus additional capabilities—including LDAP and Active Directory integration, CIFS/NFS access with the SwiftStack Filesystem Gateway, and 24x7 enterprise support.

The SwiftStack Object Storage System can span multiple geographically distributed data centers with a single namespace—thus providing built-in disaster recovery and flexibility for data storage. Administrators use the SwiftStack Controller, which provides a unified view across all clusters, to dynamically tune and optimize performance and non-disruptively upgrade the entire storage infrastructure.

SOLUTION OVERVIEW

By combining the power and flexibility of SwiftStack software with the versatility and density of Cisco UCS hardware, customers can achieve a truly scalable, manageable, cost-effective, and easy-to-deploy object storage solution.

This reference design document provides an overview of the joint SwiftStack and Cisco UCS software-defined object storage solution. It provides both detailed reference architectures and supporting performance information. The intent of this document is to enable the reader to understand how SwiftStack software is deployed on Cisco UCS servers running common distributions of the Linux operating system. The document describes the setup and observed performance on a common entry-level configuration in Cisco’s lab. For load generation and testing, SwiftStack used the freely available ssbench load generator. Details about the test setup and parameters used can be found in the performance section. For this testing, Cisco UCS C3160 servers were chosen for their density and balanced cost-to-performance ratio.

This combined SwiftStack and Cisco UCS solution can be used for many use cases—including enterprise backup and archive, media and large-data archives in the Media & Entertainment and Life Sciences industries, file-sync-and-share, private cloud storage for CloudStack and OpenStack, and many web applications leveraging object storage.
USE CASES

Enterprise Backup

SwiftStack provides a resilient and easily scaled storage target for backups of VMs, files, databases using popular applications like CommVault Simpana, Veritas NetBackup, IBM Spectrum Protect, Percona, Trillio, and more. These backups can be configured to be automatically distributed to multiple data centers by SwiftStack software for additional availability, which can be a significant simplification over traditional primary/DR replication strategies; built-in data replication ensures that backups remain available even if a server, site, or network fails. As a backup target, SwiftStack can provide a much lower-cost solution than other on-premise storage options or public cloud storage, and with the features in applications like CommVault Simpana, it is possible to add SwiftStack alongside existing backup targets to accomplish strategies like disk-to-cloud or disk-to-disk-to-cloud and/or age out the other infrastructure over time. Many additional details, demonstrations, and case studies are available at https://swiftstack.com/backup/.

Web Application Data

SwiftStack is used to build high-volume content repositories for many web and mobile applications. High concurrency capability supports the workload that millions of users place on a storage system, and a single, web-addressable namespace is much simpler to use than requiring the application to have knowledge of the storage infrastructure. These applications access data via native HTTP RESTful APIs (AWS S3 and OpenStack Swift), and developers leverage broad API support for programming languages such as Java, C, C#, Python, PHP, and Ruby.

File Sync-and-Share

SwiftStack makes it simple to store and share unlimited documents and unstructured data and access it easily via the SwiftStack Filesystem Gateway or a wide range of third-party sync-and-share document management applications—including OwnCloud, CloudBerry, Storage Made Easy, Gladinet, and more.

Active Archive (e.g., Media Archives, Genetic Sequencing)

SwiftStack is used to store and archive large files across several industries that produce massive amounts of data—including media & entertainment, life sciences, and other research efforts.

Throughout the data lifecycle, SwiftStack provides necessary flexibility—high-throughput data ingest, multi-region data distribution, and cost-effective replica and erasure coding options to balance data availability, performance, and raw-to-usable space ratios.

Metapod and Other Private Cloud Storage

SwiftStack makes it fast and simple to roll out on-premises object storage, easily connect it to your Metapod managed cloud or your OpenStack or CloudStack cloud, integrate with your AD, LDAP, or Keystone authentication systems, and monitor all of your multi-site storage from an easy-to-use browser interface.
Cisco UCS is a next-generation data center platform that combines computing, networking, and storage access resources. The platform is optimized for various enterprise workloads using open industry-standard technologies and aims to reduce TCO and increase business agility. The system integrates a low-latency, lossless 10 Gigabit Ethernet unified network fabric with enterprise-class, x86-architecture servers. It is an integrated, scalable, multi-chassis platform in which all resources participate in a unified management domain.

Cisco UCS C-Series Rack Servers deliver unified computing in an industry-standard form factor to reduce total cost of ownership and increase agility. Each product addresses varying workload challenges through a balance of processing, memory, I/O, and internal storage resources.

Cisco UCS C-Series Rack Servers benefits include:

- Form-factor-agnostic entry point into Cisco UCS
- Simplified and fast deployment of applications
- Extension of unified computing innovations and benefits to rack servers
- Increased customer choice with unique benefits in a familiar rack package
- Reduction in TCO and an increase in business agility
Cisco UCS C220 Rack Server

The Cisco UCS C220 M4 Rack Server is one of the most versatile, high-density, general-purpose enterprise infrastructure and application servers in the industry today. It delivers best-in-class performance for a wide range of enterprise workloads, including virtualization, collaboration, and scale-out applications.

The enterprise-class UCS C220 M4 server extends the capabilities of the Cisco Unified Computing System (UCS) portfolio in a one rack-unit (1RU) form-factor. It provides:

- Dual Intel® Xeon® E5-2600 v3 processors for improved performance suitable for nearly all 2-socket applications
- Next-generation double-data-rate 4 (DDR4) memory and 12 Gbps SAS throughput
- Innovative Cisco UCS virtual interface card (VIC) support in PCIe or modular LAN on motherboard (MLOM) form factor

Cisco UCS C240 Rack Server

The Cisco UCS C240 M4 Rack Server is an enterprise-class server designed to deliver exceptional performance, expandability, and efficiency for storage and I/O-intensive infrastructure workloads. This includes big data analytics, virtualization, and graphics-rich and bare-metal applications.

The UCS C240 M4 Rack Server delivers outstanding levels of expandability and performance for standalone or UCS-managed environments in a two rack-unit (2RU) form factor. It provides:

- Dual Intel® Xeon® E5-2600 v3 processors for improved performance suitable for nearly all two-socket applications
- Next-generation double-data-rate 4 (DDR4) memory and 12 Gbps SAS throughput
- Innovative Cisco UCS virtual interface card (VIC) support in PCIe or modular LAN-on-motherboard (mLOM) form factor
Cisco UCS C3160/C3260 Rack Server

The Cisco UCS® C3160 Rack Server is a modular, high-density rack server ideal for service providers, enterprises, and industry-specific environments. The Cisco UCS C3160 addresses the need for highly scalable computing with high-capacity local storage. Designed for a new class of cloud-scale applications like SwiftStack, it is simple to deploy and excellent for software-defined storage environments, unstructured data repositories, backup and archival, media streaming, and content distribution.

Extending the capability of the Cisco UCS portfolio, the new Cisco UCS C3160 Rack Server is an advanced, modular rack server with extremely high storage density. Based on the Intel Xeon processor E5-2600 v2 series, it offers up to 360 TB of local storage in a compact 4-rack-unit (4RU) form factor.

Because all its hard-disk drives are individually hot-swappable, the Cisco UCS C3160 helps you achieve the highest levels of data availability. Unlike typical high-density rack servers that require extended depth racks, the Cisco UCS C3160 has no such requirement and can comfortably fit in a standard-depth rack, such as the Cisco UCS R42610.

SWIFTSTACK OBJECT STORAGE SOFTWARE

SwiftStack Object System Components

- **SwiftStack Controller**: Easily manage the entire storage system via the Controller, which sits out-of-band from the data path and enables administrators to easily deploy, scale, upgrade, and monitor the storage system across multiple sites.
• **SwiftStack Nodes:** SwiftStack Node software is installed on industry-standard x86 hardware with standard Linux distributions. The system enables administrators to mix-and-match hardware from multiple vendors, matching capital investments directly to storage needs.

• **Filesystem Gateway (Optional):** Provides flexibility to store and retrieve files via NFS or CIFS protocols. Unlike most other file system Gateways, files stored via the Gateway can also be accessed via RESTful HTTP API as objects and vice versa.

**General SwiftStack Features**

• **Central Management Interface:** The SwiftStack Controller lets you manage all of your object storage clusters from a single centralized management interface, reducing operational complexity, conserving IT resources, and saving costs.

• **Supports Existing Application Workflows:** Apps access SwiftStack storage via a RESTful HTTP API and a broad set of client libraries for PHP, Python, Ruby, Java, and other languages. And we also support the S3 API and file system protocols like NFS and CIFS.

• **Flexible Data Protection:** Storage policies let you create multiple tiers of storage within the same namespace, giving admins and applications full control over where their data is located and how many replicas or erasure-coded segments are kept for availability.

• **Enterprise Integrations:** SwiftStack easily integrates with your existing enterprise authentication infrastructure, including Active Directory and LDAP, speeding up the provisioning and management of storage users.

• **Non-disruptive Upgrades:** The SwiftStack Controller lets you update your storage cluster while it is running, with no disruption or downtime. One click and all of the steps needed to update nodes happens automatically in the background.

• **Utilization and Reporting:** Build-out your storage-as-a-service. Chargeback and report on storage utilization to users, departments and/or applications. Utilization is tracked on a per-storage policy basis for both data stored and transfers in/out.

• **Monitoring and Alerts:** SwiftStack proactively monitors the health of each storage node and drive to ensure availability and performance. Alerts can be integrated with external monitoring tools such as Nagios and via SNMP.

• **Automatic Data Distribution:** A SwiftStack object storage cluster can be spread over multiple, geographically distributed data centers with a single namespace. This means built-in disaster recovery and more flexibility for your data.

• **24x7 Enterprise Support**
Controller

The SwiftStack Controller provides a single pane of glass dashboard to easily manage your entire SwiftStack Object Storage deployment, whether a few terabytes at one site or hundreds of petabytes distributed world-wide. The Controller enables Enterprises to manage, deploy, scale, upgrade, and monitor the object storage system via APIs and a browser based dashboard. Deployment options for the Controller include: on-premises behind your firewall or As-a-Service from cloud infrastructure managed by SwiftStack. Regardless of deployment choice, your data does not leave your data center(s).

OpenStack Swift

OpenStack Object Storage (code named “Swift”) is an open source engine for object storage, available for use under the Apache 2 open source license through the OpenStack Foundation. Swift is the engine that runs the world’s largest storage clouds, and is designed to operate on industry-standard x86 server hardware and scale out to store billions of files and hundreds of petabytes without any single point of failure.

USING SWIFTSTACK AND CISCO UCS TOGETHER

Hardware Configuration

SwiftStack’s object storage engine, at a high level, consists of the following primary services:

- **Proxy services**: Communicate with the external clients and coordinate read/write requests
- **Account services**: Provide metadata for the individual accounts and a list of the containers within an account
- **Container services**: Provide metadata for the individual containers and a list of objects within a container
- **Object services**: Provide a blob storage service that can store, retrieve, and delete objects on the node’s drives
- **Consistency services**: Find and correct errors caused by data corruption or hardware failures

For a SwiftStack cluster using Cisco UCS servers, Swift services—including Proxy, Account, Container, and Object—can run together on “PACO nodes” or separately on proxy “PAC” nodes plus “O” nodes. For small configurations, PACO nodes are commonly C240 servers; larger configurations use the C3160 or C3260 servers as PACO nodes; and the multi-PB configurations tend to use C220 servers as PAC nodes and C3160 servers as O nodes (or C240s for O nodes in highest-performance configurations that require higher CPU-to-spindle ratios).
The following four reference architectures provide a good starting point for SwiftStack on Cisco UCS hardware. In addition to the hardware listed, be sure to account for data center racks, top-of-rack network switches, and load balancers (if not already in place). Contact SwiftStack for assistance with these components and/or to identify additional configurations for significantly different use cases or much larger capacities.

### Reference Architecture #1: <1 PB for Test/Dev & General Purpose Use Cases

<table>
<thead>
<tr>
<th>Node Model/Role</th>
<th>CPU/RAM</th>
<th>Drives</th>
<th>Network</th>
</tr>
</thead>
</table>
| C240 PACO       | • 2x CPU (e.g., E5-2620 v3)  
|                 | • 64 GB       | • 2x boot drives  
|                 |             | • 12x 3.5” HDD    | • Multiple 1Gb NICs or at least 1x 10Gb NIC |

**Notes:**
- For large object counts (e.g., 1-10 million objects per container), add 1x SSD per C240 for account and container data.
- For CPUs, choose models such that the ratio of (sockets*cores*speed) : HDDs is at least 2:3.
- Testing environments commonly start with three nodes; production environments commonly start with nine (three zones of three nodes each) and expand three nodes at a time.

### Reference Architecture #2: 1-10 PB for General Purpose Use Cases

<table>
<thead>
<tr>
<th>Node Model/Role</th>
<th>CPU/RAM</th>
<th>Drives</th>
<th>Network</th>
</tr>
</thead>
</table>
| C220 PAC        | • 2x CPU (e.g., E5-2620 v3)  
|                 | • 128 GB    | • 2x boot drives  
|                 |             | • SSD (2 per object node)  | • At least 2x 10Gb NIC ports |
| C3160 Object    | • 2x CPU (e.g., E5-2650 v3)  
|                 | • 128 GB    | • 2x boot drives  
|                 |             | • HDDs       | • At least 2x 10Gb NIC ports |

**Notes:**
- Include 2x SSD per C3160 in the C220 PAC nodes for account and container data.
- For CPUs in the C3160, choose models such that the ratio of (sockets*cores*speed) : HDDs is at least 2:3.
- Configurations commonly start with two or three data center racks containing two PAC nodes and two or three C3160 object nodes in each rack; capacity is added by adding C3160s until the racks are full and then adding racks (each rack contains two PAC nodes).
### Reference Architecture #3: 1-10 PB for Backup/Archive Use Cases

<table>
<thead>
<tr>
<th>Node Model/Role</th>
<th>CPU/RAM</th>
<th>Drives</th>
<th>Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>C220 or C240 PAC</td>
<td>2x CPU (e.g., E5-2620 v3) 64 GB</td>
<td>2x boot drives SSD (2 per object node)</td>
<td>At least 2x 10Gb NIC ports</td>
</tr>
<tr>
<td>C3160 Object</td>
<td>2x CPU (e.g., E5-2620 v3) 128 GB</td>
<td>2x boot drives HDDs</td>
<td>At least 2x 10Gb NIC ports</td>
</tr>
</tbody>
</table>

**Notes:**
- Include 2x SSD per C3160 in the C220 PAC nodes for account and container data.
- For CPUs in the C3160, choose models such that the ratio of (sockets*cores*speed) : HDDs is at least 1:3.
- Configurations commonly start with two or three data center racks containing one PAC node and two or three C3160 object nodes in each rack; capacity is added by adding C3160s until the racks are full and then adding racks (each rack contains one PAC node).

---

### Reference Architecture #4: 1-10 PB for High-Performance Use Cases

<table>
<thead>
<tr>
<th>Node Model/Role</th>
<th>CPU/RAM</th>
<th>Drives</th>
<th>Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>C220 or C240 PAC</td>
<td>2x CPU (e.g., E5-2620 v3) 64 GB</td>
<td>2x boot drives SSD (2 per object node)</td>
<td>At least 2x 10Gb NIC ports</td>
</tr>
<tr>
<td>C3160 Object</td>
<td>2x CPU (e.g., E5-2620 v3) 128 GB</td>
<td>2x boot drives HDDs</td>
<td>At least 2x 10Gb NIC ports</td>
</tr>
<tr>
<td>C240 Object</td>
<td>2x CPU (e.g., E5-2620 v3) 128 GB</td>
<td>2x boot drives HDDs</td>
<td>At least 2x 10Gb NIC ports</td>
</tr>
</tbody>
</table>

**Notes:**
- Include 1x SSD per C240 object node in the C220 AC nodes for account and container data.
- For CPUs in the C240 object nodes, choose models such that the ratio of (sockets*cores*speed) : HDDs is at least 1:1.
- Configurations commonly start with two or three data center racks containing four proxy nodes, four AC nodes, and approximately 16 object nodes in each rack; capacity is added by adding full racks.
Networking

**Cluster Networks**


A typical SwiftStack deployment will have an “outward-facing” network and an internal “cluster-facing” network. When designing the network capacity for a SwiftStack deployment using the default three-replica storage policy, keep in mind that writes fan-out in triplicate in the storage network. Since there are three copies of each object, an incoming write sent to a proxy node is then “forwarded” to three object nodes. Therefore, network capacity for writes needs to be considered in proportion to overall workload.

Starting from the client’s perspective, the “client-facing” IP is the IP address (or DNS record) that clients connect to. Typically, that would be a WAN IP on a load balancer or on a firewall.

The “outward-facing” IP(s) are the IP(s) on the proxy node(s). It is important to note that “outward-facing” is not the same as a public or WAN IP. These “outward-facing” IPs are simply the IP addresses of the proxy nodes facing out—towards the load balancer. Thus, outward-facing IPs are the IPs that should be included in the load balancing pool.

All intra-cluster traffic (between the proxy and object nodes, for example) occurs through the “cluster-facing” IP(s).

In summary, while all networking in a SwiftStack cluster is done via Layer-3, a SwiftStack cluster will have these network segments:

- **Outward-facing Network**: Used for client traffic (i.e., API access). If an external load balancer is used, it will exist in the outward-facing network.
- **Cluster-facing Network**: Used for intra-cluster SwiftStack node communication.
• **Replication Network (optional):** Used to separate intra-cluster replication traffic (which occurs when a drive fails or capacity is added to or removed from the cluster) from other intra-cluster traffic for normal client traffic.

• **Management Network:** Used for IPMI, iLO, etc., for hardware management.

---

**Load Balancer(s)**

For a SwiftStack cluster, a dedicated load-balancing solution can use any method from simple round-robin DNS to layer-7 load balancing. Open-source load balancers such as HAProxy or commercial solutions from companies like F5 and A10 can be used. For SwiftStack, you will need to load-balance across your Swift proxy nodes to divide the client request load between your proxy servers.

The outward-facing IPs of the proxy nodes need to be included in the load-balancing pool, and the load balancer must be assigned a virtual IP address (VIP). Load balancers should be set up to run health checks against the proxy nodes in their pools so that a proxy will be automatically excluded if it is unresponsive. To remove a failed proxy node from the load balancing pool, configure the loadbalancer to check the proxy node’s health check URL. A healthy SwiftStack proxy node should respond on http://<outward-ip-address>/healthcheck, such as http://192.168.11.81/healthcheck.

---

**Installation and Software Configuration**

The following high-level steps are involved in deploying these reference architectures:

1. **Install a SwiftStack controller.** This can be done by leveraging the hosted SwiftStack Management Service or by deploying an on-premises instance of this controller on physical hardware or in a virtual machine. To sign up for an account on the SwiftStack Management Service, visit https://swiftstack.com/try-it-now/, or for an on-premises instance, contact sales@swiftstack.com.

2. **Install hardware**—including UCS servers, network switches, and a load balancer (if needed).

3. **Install Linux.** SwiftStack supports RHEL, Ubuntu, and CentOS. Install the standard server edition; assign a hostname; and assign IP addresses to network interfaces.

4. **Configure SwiftStack nodes.** The SwiftStack management software greatly simplifies the process of preparing and allocating servers and drives for use. Only a single command is required on each server, and the rest of the configuration is done in SwiftStack’s simple-to-use browser interface. Details and step-by-step screenshots can be found in SwiftStack’s Quick-Start Guide: https://swiftstack.com/docs/install/index.html

   a. Install the SwiftStack software on each node (`curl https://platform.swiftstack.com/install | bash`).

   b. Claim the new node(s) to your SwiftStack cluster in the browser (i.e., copy and paste one line from the terminal window to your browser window).

   c. Create a new SwiftStack cluster if desired for your new node(s).

   d. Allocate drives and network interfaces for the new node(s), and identify in which zone and region each node belongs.

   e. Create your first user to access the cluster (or alternatively, integrate with LDAP, AD, etc.).

   f. Click “Deploy,” and begin using your new SwiftStack cluster!
# INTEGRATION VERIFICATION TESTING

Both functional and performance tests were performed jointly by Cisco and SwiftStack engineers in Cisco’s San Jose “OpenStack Engineering Lab” to demonstrate the capabilities of SwiftStack software running on a relatively small configuration of UCS hardware. Details of the test cases and results are provided here.

## Functional Tests

The following test cases were used to demonstrate basic functionality of the combined SwiftStack and Cisco UCS solution. Details of performance testing can be found in the following section.

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Success Criteria</th>
<th>Result</th>
</tr>
</thead>
</table>
| Perform SwiftStack installation from hosted controller. | • Pre-install RHEL on each UCS server.  
• Install SwiftStack with a single command on each UCS server.  
• Complete configuration using the SwiftStack controller’s web interface. | Pass |
| Perform SwiftStack installation from on-premises controller. | • Install the SwiftStack controller as a VM in Cisco’s lab.  
• Pre-install RHEL on each UCS server.  
• Install SwiftStack with a single command on each UCS server.  
• Complete configuration using the SwiftStack controller’s web interface. | Pass |
| Demonstrate ease of configuration. | Using the SwiftStack controller’s web interface:  
• Create a cluster.  
• Configure outward-facing and cluster-facing networks.  
• Assign drives to storage policies.  
• Assign nodes to zones and regions.  
• Create a user or integrate with existing LDAP/AD infrastructure.  
• Press “Deploy” to enable the cluster. | Pass |
| Demonstrate commonly used features. | Using the SwiftStack controller’s web interface:  
• Observe the health and status of all cluster nodes.  
• Observe the graphical analytics of cluster-wide and node-specific workload and utilization.  
• Observe the capacity-planning data and graphs.  
• Enable S3 API compatibility middleware.  
• Perform a software upgrade while the cluster is running. | Pass |
| Demonstrate resilience to hardware changes, data corruption, and flexibility to add/remove hardware online. | In all cases, observe that the cluster remains online for data access:  
• Add and remove drives.  
• Add and remove a node.  
• Simulate multiple drive failures.  
• Simulate a node failure.  
• Simulate data corruption of an object at rest; observe SwiftStack repair the corrupted data. | Pass |
<table>
<thead>
<tr>
<th>Test Case</th>
<th>Success Criteria</th>
<th>Result</th>
</tr>
</thead>
</table>
| Read/Write data from multiple applications. | • Drag-and-drop files into the SwiftStack Web Console.  
• Upload/Download objects using the python-swiftclient command-line tool.  
• Connect Cyberduck or similar cloud storage explorer.  
• Perform a backup from CommVault Simpana or similar. | Pass |

Performance Tests

**Benchmarking Strategy**

It is often helpful to understand the capabilities of a reference architecture when placed under simulated client load. SwiftStack and Cisco performed initial benchmark testing to simulate a relatively common “enterprise backup” use case. For the purposes of this benchmark, three client servers were used to execute the ssbench tests as described below against a three-node (3x C3160) SwiftStack cluster.

**Preparing for Testing**

Minimal configuration and preparation was required to prepare for benchmark testing. Specifically, the number of Swift “workers” (i.e., small pieces of software that run in parallel to share the responsibility of a concurrent application workload) was set according to the best practices in SwiftStack’s documentation: [https://swiftstack.com/docs/admin/cluster_management/ss_tuning.html](https://swiftstack.com/docs/admin/cluster_management/ss_tuning.html)  

In addition, the four 10Gb physical network links in each C3160 PACO node were set up using Cisco’s standard two-VNICs-per-physical-NIC configuration such that two VNICs participated in the “outward-facing network,” and the remaining six VNICs participated in the “cluster-facing network” as shown in the diagram below. Specifically, the VNICs were configured as shown here:

Outward-facing (bond0) - Bonding Mode: IEEE 802.3ad Dynamic link aggregation  
Cluster-facing (bond1) - Bonding Mode: load balancing (xor)  
(S-Mac XOR D-Mac) % Slaves

**Important Note:** The network bonding (i.e., link aggregation) settings used were suboptimal for the small number of benchmark systems and SwiftStack nodes in the test cluster. This resulted in an artificial network bottleneck during these benchmark tests (i.e., only some of the available physical network links were utilized for data traffic). In larger configurations with more SwiftStack nodes and more clients systems, these settings would be perfectly appropriate, but the ideal setting in this small lab environment would have been a “layer 3+4” load-balancing algorithm on each network link aggregation. Cisco’s VNIC software did not support this on RHEL 6.x (which was used in this testing), but it is now supported on RHEL 7.x. If you are deploying a small SwiftStack cluster on Cisco UCS hardware, contact SwiftStack to help optimize your network configuration.
Tests Performed

ssbench Tool

SwiftStack has enhanced a quick and effective tool—called “ssbench”—for benchmarking object storage clusters that support the Swift API; it is freely available at https://github.com/swiftstack/ssbench. Prior to each test execution, a predetermined number of containers and objects were pre-populated in the cluster, then the tests performed a maximum number of PUTS to the cluster or GETS from the cluster across a range of object sizes with varying levels of concurrent client connections—each for a 180-second period of time. Two examples of “scenario files” used to determine how a benchmark test will execute are provided here:

Example: Testing reads of 1kB objects

```json
{
  "name": "1KB-GET test scenario",
  "sizes": [
    {
      "name": "1KB-GET",
      "size_min": 1024,
      "size_max": 1024
    }
  ],
  "initial_files": {
    "1KB-GET": 1000
  },
  "run_seconds": 180,
  "crud_profile": [0, 1, 0, 0],
  "user_count": 100,
  "container_base": "ssbench-1KB",
  "container_count": a100,
  "container_concurrency": 10
}
```

Example: Testing writes of 1kB objects

```json
{
  "name": "1KB-PUT test scenario",
  "sizes": [
    {
      "name": "1KB-PUT",
      "size_min": 1024,
      "size_max": 1024
    }
  ],
  "initial_files": {
    "1KB-PUT": 100
  },
  "run_seconds": 180,
  "crud_profile": [1, 0, 0, 0],
  "user_count": 100,
  "container_base": "ssbench-1KB",
  "container_count": 100,
  "container_concurrency": 10
}
```
Simulating Backup, File Sync-and-Share, and Web Applications

Specifically, for this benchmark, PUTs and GETs of objects with sizes ranging from 1KB to 500MB were made using a range of concurrent workers ranging from 100 to 5,000 clients. Combinations of tens to hundreds of clients with large object sizes would be typical of many backup and archive use cases, while thousands of clients with smaller object sizes would be more typical of web applications or file sync-and-share use cases. Results captured include the total number of objects written or read, latency, throughput, and more.

**Interpretation of Results: Meeting Real-World Requirements**

It is often best to interpret performance numbers in the context of real-world requirements. As an example, this is a recent SwiftStack customer’s stated requirement:

“One of our big customers, who is into backup and archive, has a target capacity of ~1PB. At 1PB, they expect a daily change rate (8 hour window) of ~1.5-3%, which means they need an Object Store that can handle between 1.8TB and 3.6TB per hour of sustained writes, i.e., around 1GB/s.”

As seen in the charts below, despite the suboptimal network link aggregation settings, client systems were able to write over 1,000 small objects per second and over 1.3 GBytes/sec of large-object write throughput and read over 7,000 small objects per second and nearly 3.3 GBytes/sec of large-object read throughput to and from the cluster—more than sufficient for the real-world customer. *(Note, with the same hardware and preferred link aggregation settings, you can expect about twice the write and read throughput observed in these tests. See the “important note” above for more details.)*

**Write Performance**

<table>
<thead>
<tr>
<th>Object Size</th>
<th>Concurrent Clients</th>
<th>Total Requests Per Second</th>
<th>Total Throughput from Client (MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1B</td>
<td>500</td>
<td>1,015</td>
<td>0.001</td>
</tr>
<tr>
<td>1KB</td>
<td>500</td>
<td>1,014</td>
<td>1.0</td>
</tr>
<tr>
<td>1MB</td>
<td>500</td>
<td>621</td>
<td>621</td>
</tr>
<tr>
<td>10MB</td>
<td>500</td>
<td>129</td>
<td>1,291*</td>
</tr>
<tr>
<td>100MB</td>
<td>500</td>
<td>14.0</td>
<td>1,399*</td>
</tr>
<tr>
<td>500MB</td>
<td>500</td>
<td>2.7</td>
<td>1,355*</td>
</tr>
</tbody>
</table>

*Write throughput was limited by bonding mode; expect nearly double with preferred VNIC settings on RHEL 7.x. See “important note” above for more details.*
### Read Performance

<table>
<thead>
<tr>
<th>Object Size</th>
<th>Concurrent Clients</th>
<th>Total Requests Per Second</th>
<th>Total Throughput from Client (MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1B</td>
<td>500</td>
<td>6,661</td>
<td>0.006</td>
</tr>
<tr>
<td>1KB</td>
<td>500</td>
<td>6,732</td>
<td>6.6</td>
</tr>
<tr>
<td>1MB</td>
<td>500</td>
<td>3,342</td>
<td>3,342*</td>
</tr>
<tr>
<td>10MB</td>
<td>500</td>
<td>330</td>
<td>3,304*</td>
</tr>
<tr>
<td>100MB</td>
<td>500</td>
<td>31.7</td>
<td>3,173*</td>
</tr>
<tr>
<td>500MB</td>
<td>500</td>
<td>5.2</td>
<td>2,605*</td>
</tr>
</tbody>
</table>

*Read throughput was limited by bonding mode; expect nearly double with preferred VNIC settings on RHEL 7.x. See “important note” above for more details.

### CONCLUSION

In conclusion, by combining the power and flexibility of SwiftStack software with the versatility and density of Cisco UCS hardware, customers can achieve a truly scalable, cost-effective, and easy-to-deploy-and-manage object storage solution for many of today’s data-heavy use cases—including enterprise backup and archive, media and large-data archives in the Media & Entertainment and Life Sciences industries, file-sync-and-share, private cloud storage for CloudStack and OpenStack, and many web applications leveraging object storage. Descriptions of the component technologies were discussed; details of four reference architectures were provided; and benchmark results were described.

More information can be found on our website:

https://swiftstack.com/cisco/