89 Fifth Avenue, 7th Floor

New York, NY 10003

www.TheEdison.com

@EdisonGroupInc

212.367.7400

Dell EMC Elastic Cloud Storage and SwiftStack

Comparing Object Storage Solutions

Printed in the United States of America

This report was commissioned by Dell EMC.

The information contained in this document is based on Dell EMC provided materials and independent research and was aggregated and validated for Edison Group, Inc. by the Edison Group Analyst team.

First Publication: November 2017

Produced by: Harold Kreitzman, VP, Strategic Advisory Services

EDISON

Table of Contents

Executive Summary

This white paper explores the differences between Dell EMC Elastic Cloud Storage (ECS) and the SwiftStack implementation of the OpenStack Swift object storage module.

It is clear that ECS has distinct advantages over SwiftStack as ECS is built from the ground up producing a fully-integrated, enterprise-grade object storage solution while SwiftStack added functionality on top of Swift to make it a viable, supported, commercial-grade product. The benefits of using ECS over SwiftStack is evident in the following areas detailed below:

- Scalability
- Installation and Provisioning
- Management
- Storage Efficiency and Data Protection
- Data Access
- Support and Upgrades
- Data Security and Metadata Search

OpenStack

OpenStack provides organizations with a framework to deploy and manage their own private cloud. OpenStack's modular architecture allows for easy incorporation of new features. Each new capability is built into its own modular component. Swift is the OpenStack module that provides object storage used to hold disk and server images, as well as other data that is critical but not performance dependent.

OpenStack has enjoyed quick feature development and innovation. The perceived low cost of entry has made it a popular choice for organizations considering a private or hybrid cloud. As with many open source projects, a high level of expertise and understanding of OpenStack's internal workings is required to successfully deploy an installation. Even the most successful OpenStack installations agree that significant IT resources are necessary to successfully operate OpenStack. OpenStack has, thus far, not suffered from the abandonment that many open source projects have faced once the initial excitement and interest wanes.

Swift and SwiftStack

As mentioned earlier, Swift is the object storage component of OpenStack but can also be deployed as an independent object storage solution.

Many open-source solutions, including Swift, require a high level of expertise to implement. This is generally because developers are often more excited about features than ease of use. Swift installation and operations require a high level of Linux expertise, Python and an understanding of the underlying hardware. Management of Swift and is still primarily done through the command line interface (CLI).

Swift, like most open source offerings, is only available as software for installation. No pre-loaded or appliance option is available so all users are left to install the software on user obtained hardware.

SwiftStack is a company that was created to support Swift as a commercial product. The product is also called SwiftStack. As with many such open source commercialization efforts, SwiftStack heavily contributes to the open-source project it originated from, OpenStack Swift, and influences the community, but nominally does not have direct control over project or source code. New features and capabilities are subject to contributions and interest from a community that they have no direct control over. Bug fixes must be reconciled with other potential code provided by the community.

SwiftStack also provides some capabilities not found in the open-source version of Swift. The SwiftStack implementation employs a separate controller for easier management and installation automation. However, this controller requires additional installation and configuration of the controller itself, management of the controller's security issues, as well as maintaining an additional PostgreSQL database to manage the controller data. This controller can be an additional single point of failure for the SwiftStack cluster. For an additional cost, SwiftStack provides their controller as a cloud service that they host and manage.

Dell EMC Elastic Cloud Storage (ECS)

ECS is Dell EMC's third generation object storage solution, incorporating lessons learned from EMC's previous Centera and Atmos object storage platforms. It was designed to provide customers with the benefits of public cloud storage, e.g. Amazon AWS S3, deployed within the customer's data center while delivering additional features above what Amazon S3 offers.

For maximum flexibility, ECS is available in a variety of configurations:

- **Software Only** - As a well-supported and simple-to-install software-only solution, users can choose to install ECS on their own systems, fully tested and certified systems available from Dell EMC and other popular vendors, or select pre-configured appliances for quick integration.
- **Fully Provisioned Appliance** - ECS can easily be integrated into existing data center or added to existing ECS installation for expansion.
- **ECS Dedicated Cloud** - Provides the benefits of a wholly owned ECS installation without having to house and maintain the hardware.
- **Multitenant Storage Solution** – is offered through several cloud vendors.

ECS is designed as an enterprise storage solution and provides not only features and functionality, but also keeps manageability and usability in mind.

ECS is designed to function as a geo-replicated data store. Several sites can be configured to share data, enabling applications to access data from anywhere in the world. By replicating data to several global sites, enterprises can bring the data closer to the user for more efficient access. Strong consistency ensures that the data accessed is always the most current copy regardless of the site being accessed. Geo-replication also inherently enables each site to serve as a disaster recovery site for the other sites.

OpenStack Swift interfaces with the rest of OpenStack using a purpose built RESTful object protocol (also called Swift) and authentication using Keystone. For compatibility, ECS is also able to interface with OpenStack using the same mechanism. ECS is compatible with V2 APIs and Swift and Keystone V3 authentication, making it a drop-in replacement for Swift/SwiftStack for OpenStack.

Platform Use Comparisons

Scalability

The following table summarizes function/feature differences:

Functions/Features	ECS	SwiftStack
Scaling Limitations	No inherent limits	Required SQLite database limits scalability
Adding Capacity	Nodes are homogeneous and can simply be added to a cluster as needed	The ratio of proxy nodes to object nodes required for your cluster's workload must be determined and provisioned for accordingly

Table 1- ECS/SwiftStack Scalability Comparison

When provisioning SwiftStack, it is necessary to be mindful of its internal architecture - understanding the relationships between proxy and object nodes to ensure the correct ratio of the two to meet workload requirements. SwiftStack relies on a SQLite database on disk to manage the metadata. Scaling the SwiftStack environment is limited by the ability to scale this database.

Unlike SwiftStack, which is limited by its SQLite database for metadata, ECS is designed with no coded limits to its scalability. Scaling an ECS cluster is simply a matter of adding additional nodes to the cluster. ECS features built-in metadata search with no additional hardware and no single point of failure since each ECS node participates in metadata management.

Instead of having a variety of different node types, performing different functions and requiring the customer to configure and manage them differently, ECS nodes are homogenous. Every node is, from the user perspective, the same. ECS nodes automatically coordinate with each other to determine which nodes run additional cluster functions. This makes it easier to scale since users can simply add another node for expansion without having to worry about corresponding changes to a proxy node or an object node or specifying which services need to be added. ECS nodes automatically determine which additional services are required by the cluster to accommodate the added nodes and capacity.

Takeaway

ECS has no inherent scalability limits. SwiftStack is limited by its dependence on the SQLite database. The fact that ECS nodes are homogeneous means that capacity can be added by just adding more nodes. There is no need to determine the proper mix of different node types before provisioning more capacity.

Installation and Provisioning

The following table summarizes function/feature differences:

Functions/Features	ECS	SwiftStack
Deployment options	Appliance Software Only Dedicated Cloud Multi-tenant Cloud	Bare metal Container Virtual Machine
Configuration	Add nodes with desired amount of storage Only web based configuration required	Determine proxy node to object node ratio Modify configuration files Some command line operations on the OS and the software
Installation	Browser based configuration	Browser based configuration

Table 2 - ECS/SwiftStack Installation/Provisioning Comparison

SwiftStack can be deployed on bare metal, on a container, or in a virtual machine. Unlike basic OpenStack Swift, SwiftStack provides a SwiftStack controller to help automate installation. Users still must understand and manage the complication of determining how many proxy vs object nodes they must provision. For a completely on-premises installation, users must have intimate knowledge of the underlying Linux OS, SwiftStack and the SwiftStack controller. Initial setup requires modifying several configuration files and command line operations on the OS and the software.

ECS is intended to provide customers with the ease of deployment expected by enterprise customers, while providing them with the flexibility of choosing how they want to consume object storage. ECS is available in a software-only option which can be installed on commodity hardware. ECS installs easily as Docker containers which the installation script automatically provisions. There is no need to determine the ratio of different types of nodes since ECS nodes automatically take on the required functions based on the cluster needs. The administrator does not need to be an expert on the ECS architecture to efficiently manage the system.

Many enterprises are time constrained for their deployments. The appliance option for ECS allows quick and easy deployment. The appliance is already constructed and installed before it arrives at the datacenter. Having the system already installed means that minimal work is needed on site to connect and configure the appliance to the existing environment.

The dedicated cloud option for ECS makes things even easier. Since the customer owned system is fully hosted and managed by Dell EMC, the only thing customers need to do is connect their existing data center to the ECS appliance at the co-located datacenter.

Takeaway

ECS and Swift both have simplified installations. ECS has an easier architecture to manage and options for turnkey deployment as well as cloud based options. For customer who prefer not to take the time or obtain expertise to install a software only solution. ECS provides the added flexibility to obtain ECS as a pre-installed appliance for ease of deployment in your datacenter or as a hosted cloud from Dell EMC through Virtustream.

Management

The following table summarizes function/feature differences:

Functions/Features	ECS	SwiftStack
Interface	Web based interface	Web interface for basic features Advanced features only from the command line
Troubleshooting	Web based interface	Execute *ssontrol* from the command line with options

Table 3 - ECS/SwiftStack Management Comparison

While SwiftStack now includes a web-based interface to manage the most basic functions, many features still require the use of the Linux command line interface. The SwiftStack interface stems from its Linux-centric developer origins with a web based overlay added in an effort to make it more approachable.

Many advanced SwiftStack management tasks still require the administrator to launch a command line process. Troubleshooting commands, for example, often require executing `ssontrol` with various options to query or modify existing information. Users must be aware of which options exist for the command.

ECS is designed to be managed primarily through the GUI. ECS is designed for users to be able to manage the system with relatively little knowledge of the underlying systems. All major capabilities can be accessed through the GUI without the needing to know much about the internal workings of ECS. Storage administrators only need to be experts on storage and their own environment, not the specific intricacies of the storage platform.

Takeaway

Command line interfaces are, by their nature, much less intuitive than a GUI. They require users to have foreknowledge of what options are available. GUI interfaces require only the understanding of the functions without having to remember all the options since they are presented in the interface. ECS GUI management interface means that the administrator has a much lower barrier to entry thereby easing the administration burden. For enterprises, being able to manage storage from a GUI means that it can be managed by storage administrators who understand the organization's specific storage needs without requiring vendor specific knowledge.

Storage Efficiency and Data Protection

The following table summarizes function/feature differences:

Functions/Features	ECS	SwiftStack
Data Protection	Data Erasure coded at 12+4 or 10+2 schemes	Triple redundant copies Optional erasure coding
Erasure Coding Scheme	Data is organized into 128MB chunks which are then erasure coded	SwiftStack postprocesses each object and erasure codes regardless of size.
Small File Handling	Writes are aggregated to reduce the number of disk operations	Files are erasure coded regardless of size. For small files, this could be very inefficient
Large File Handling	Large files are erasure coded into 128MB chunks as it is ingested Very large files can be broken into multiple parts and distributed to different nodes to be uploaded in parallel	Large files are erasure coded as an entire file after it is ingested.
Geo-replication	Each site erasure codes its own data Data from other sites are XOR'ed to improve efficiency. Multiple sites can be protected with the overhead of just one additional site.	Each site erasure codes its own data Data from other sites are replicated. The overhead of protecting the data from other sites is the sum of the space required by all sites.

Functions/Features	ECS	SwiftStack
Data Consistency	ECS implements strong consistency which ensures that the latest data is provided regardless of where the data is queried	SwiftStack implements eventual consistency which means that each location will provide the information it has at the given time of query. This data may be inconsistent with other locations if it has not yet have an opportunity to update.

Table 4 - ECS/SwiftStack Efficiency/Data Protection Comparison

Erasure coding (EC) is a data protection technique that involves segmenting data into pieces and creating additional error correction pieces that can be used to re-constitute any lost or damaged data fragments. By distributing the data and error correction fragments across the cluster, this technique ensures that any loss of a single disk or node will not result in lost data since there are enough remaining pieces to rebuild. The typical EC nomenclature is M+N where M represents the number of data pieces and N represents the number of error correction pieces. Example: 12+4 means this erasure coding scheme segments the data into 12 pieces, generating 4 error correction pieces. Up to 4 fragments of any type could be lost and the system will still recover all data.

SwiftStack typically uses triple replicated redundant system. This is an extremely inefficient scheme since it requires 3x the amount of storage to protect data. These scheme gets even worse when replicated across multiple geographies. Where a single site would require 3x the amount of space available to protect data, replicating across 2 sites results in 6x the amount of space required. Three sites require 9x the amount of storage, etc.

SwiftStack also supports erasure coding as an optional replacement for triple replication. However, instead of providing defined erasure coding options, SwiftStack leaves it to the user to determine what the data to parity fragment ratio will be. SwiftStack's erasure coding scheme operates on every object individually regardless of size. For many small objects, this could lead to very poor performance since it will encode, fragment, and write all the small fragments separately. For a given bucket, there is no way to specify limits or parameters on which objects will get erasure coded and which will not for performance. For performance reasons, if many small objects are anticipated, even if large objects will also be stored, erasure coding should not be used and the less storage efficient mirroring policy should be used. The configuration of the erasure coding policy must also consider the layout of the SwiftStack cluster in order to ensure that the erasure coded fragments are properly distributed across the cluster to ensure sufficient data durability and recoverability.

Across geographies, SwiftStack, like most object storage systems, supports *eventual* consistency. This means that data written to one location will eventually propagate to other sites. However, there is a window where users may see different versions of the data they're trying to access depending on which site they contact. The burden is on the application developer to ensure that users and applications do not see inconsistent data based on access location.

ECS was designed with erasure coding at its core and supports two erasure coding schemes. The 12+4 scheme is used for most typical instances and is the default setting. It provides superior performance and data protection and requires about 33% data overhead. The other, 10+2, scheme is only used for cold data archiving. This scheme allows for greater data efficiency, 20% overhead, but slightly less data resiliency. For cold archive data that is seldom accessed, decreased resiliency is a fair trade-off for increased data efficiency. ECS automatically ensures data durability by intelligently distributing the fragments across the cluster.

ECS has optimized data-handling capabilities for small and large objects. Many small objects are aggregated into larger single writes, reducing the number of round trips to process individual writes to disk, thereby greatly improving small file performance. ECS will aggregate data into 128MB chunks before erasure coding the chunk. For large files, ECS breaks the files down into 128MB chunks and erasure codes each chunk for greater efficiency.

Starting with ECS 3.1, for extremely large files, ECS allows for multi-part uploads. This feature allows users to break up the file and upload the parts in parallel to multiple nodes simultaneously. This not only allows the file transfer to be parallelized for better performance, but it also makes the upload more resilient. If a single part experiences an error during upload, only that part must be re-uploaded. The other upload parts do not need to be re-uploaded.

For geo-distributed environments, ECS asynchronously replicates data across multiple locations. This enables organizations to allow more local access of the same data across multiple geographies. A user in one geography can access data that was originally written in another geography by accessing their local ECS installation. This reduces the amount of latency the user experiences and reduces the dependence on long, slow WAN traffic. ECS follows a unique strong consistency data policy that ensures that the copy of the data the user receives from their local ECS installation is always the latest copy and is consistent regardless of which geography the data is accessed from. ECS achieves this by having each local site confirm that it has the latest version of the data when it services the user's data request, thereby relieving the applications developer of ensuring that the user always gets the latest copy of the data.

ECS implements a unique XOR data protection scheme that allows remote sites to provide disaster recovery protection for the entire geo-distributed environment while only requiring the overhead of a single additional site across each existing site. This method enables ECS to achieve greater storage efficiency as the number of sites in the environment increases. For example, using the 12+4 erasure coding scheme, a single site requires 1.33 the amount of storage to provide data protection. For two sites, the space required would be 2.67X to provide disaster recover protection. However, at three sites, the storage required would be 2.0X and at 4 sites, the storage requirements drop to 1.77X. The efficiency improves as more sites are added. This not only saves on space but also results in less WAN traffic required to maintain DR protection.

Takeaway

SwiftStack supports erasure coding as a supplement to their less efficient triple-data redundancy system. The Swift implementation was added onto the existing system as an afterthought, requiring adaptations to data flow. As a supplement, Swift erasure coding is complicated to configure and is not appropriate for all data types. ECS, on the other hand, was designed from the start around erasure coding for data protection. There is no additional configuration required and ECS handles small and large files efficiently and easily with erasure coding. Data protection is automatically handled without the risk that the user could compromise data durability through misconfiguring the cluster. Beyond erasure coding, ECS does not demand high bandwidth and low latency from WANs to function across data centers nor does it require the user to determine what erasure coding scheme to implement. Across multiple geographies, ECS' strong consistency implementation ensures that data accessed is always the latest version. With SwiftStack, the same data accessed from different data centers can vary, replying on applications developers to ensure consistency across geographies. ECS also implements an innovative XOR data protection system across multiple geographies that become more efficient as the number of data centers increases.

Data Access

The following table summarizes function/feature differences:

Functions/Features	ECS	SwiftStack
Access Protocol	S3, Swift, NFS, HDFS, CAS, Atmos CIFS via gateway	Swift S3 emulation NFS & CIFS via gateway

Table 5 - ECS/SwiftStack Data Access Comparison

SwiftStack's primary data access protocol is the OpenStack Swift object protocol. It is also able to emulate the more popular S3 object protocol from Amazon. This is implemented as a middleware layer, not integral to the Swift data architecture. Because of mapping issues, clients may see mismatches in "folders" when transferring data between S3 and Swift protocols.

SwiftStack also provides a filesystem gateway to handle NFS and CIFS requests. The gateway works by creating a share of its own filesystem and exporting that to the file based clients. When data is written to the gateway, it writes the data to its filesystem, then communicates with a proxy node to write that data to an object node in Swift. When data is read, it checks its own filesystem first and provides that data. If the data is not in its filesystem, it retrieves the data from Swift and provides it to the client.

While this gateway provides file based functionality to Swift, it also requires additional hardware. The gateway is a separate server that must be provisioned and maintained to provide this functionality. In addition, this gateway becomes an additional point of failure since the share structure is based on the local filesystem of the gateway. SwiftStack claims that additional gateways can be employed to manage the load and reduce the chance of a single failure. However, this not only adds additional hardware, but also additional complexity. Since each gateway relies on its own filesystem for caching, and serves up the data in its cache first when requested. This makes it possible that two writes to the same file on different gateways will mean that two potentially different files can be returned depending on which gateway the client accesses and how long it takes the caches to clear and reconcile. SwiftStack does not currently support HDFS for analytics workloads.

ECS on the other hand, natively supports S3, Swift, NFS, HDFS (Hadoop Data File System), Centera CAS, and Atmos protocols. All these protocols are handled by ECS Data Services running on all ECS nodes. Every ECS supported protocol has direct access to the ECS storage engine for efficient data handling. S3 and Swift data remain mutually consistent with no emulation required.

For NFS, since there are no gateways, there is no reliance on a local filesystem. This allows the ECS NFS to have capabilities beyond the typical NFS implementation. It has no limits on the number of files or directories it can support, can write files up to 4TB, and can make use of ECS' built in quota and encryption infrastructure. Also, since the data and file structure is stores in ECS data structure, there is no single point of failure.

ECS also supports the HDFS protocol providing direct access for big data analytics. Hadoop users can directly connect their compute resources to ECS without having to employ direct attached storage for each compute resource for more efficient analytics. Centera CAS and Atmos protocols for compatibility with the previous Dell EMC object systems.

The same data can be accessed concurrently on ECS via multiple protocols. For example, data written in NFS or HDFS can be accessed through S3 or Swift and vice versa. This provides an easy way for applications developers who are looking to migrate their applications from files based NFS to object based S3 while maintaining functionality and data consistency for both.

Takeaway

SwiftStack's native protocol is the Swift object protocol. All other protocols, are built on top of this and rely on some form of translation or gateway. ECS data access for S3, Swift, NFS, HDFS, Atmos, and CAS are all native to ECS. Each of these protocols communicate to the ECS storage engine as peers providing efficient and consistent access to the data. This enables easier transition from older applications to newer web-based applications.

Support and Upgrades

The following table summarizes function/feature differences:

Functions/Features	ECS	SwiftStack
Self Help	Dell EMC web-based knowledge base customer self-help tools	IRC, Twitter, OpenStack forum
Paid Support	30-minute response time 24/7 4-hour onsite support	8am-5pm PST 24/7 urgent callback

Table 6 - ECS/SwiftStack Support/Upgrades Comparison

Swift is community supported through IRC and Twitter, as well as through the OpenStack forums and OpenStack IRC channels. Paid support is available from SwiftStack which offers standard support from 8am-5pm PST. 24/7 urgent callback support is also available as is online support.

As a tier 1 enterprise supplier, Dell EMC offers a variety of support options. The most common option offers 30-minute response times on a 24x7 basis. This option with 4-hour onsite support is also available for ECS appliances. Dell EMC web-based knowledge base and customer self-help tools are also available 24/7 from the Dell EMC website.

Takeaway

Dell EMC offers enterprise grade 24/7 support for ECS. While SwiftStack support may be appropriate for consumer level requirements, enterprise customers require Dell EMC's 24/7, 30-minute response times to ensure prompt support and minimal downtime.

Data Security and Metadata Search

The following table summarizes function/feature differences:

Functions/Features	ECS	SwiftStack
D@RE	Integrated encryption FIPS-140-2 Level 1 compliant No additional HW required No unencrypted internode traffic Automatic key generation and management S3 encryption semantics supported	Encryption implemented in middleware Internode traffic not encrypted Sensitive to object path Additional configuration required for container sync
Metadata search	Works with encryption	Not compatible with encrypted data

Table 7 - ECS/SwiftStack Security/Metadata Search Comparison

SwiftStack recently added at-rest encryption. Encryption is implemented as a pair of middleware add-ons, encryption and key master middleware modules. Both must be enabled and configured correctly by the user and implemented in the correct place in the proxy node's data path and individually across all proxy nodes.

SwiftStack encryption is restricted to only data being processed by the proxy node. It does not consider any inter-node traffic. SwiftStack's documentation regarding encryption specifically states: "This feature is not intended to protect against an attacker that gains access to Swift's internal network connections, or gains access to key material or is able to modify the Swift code running on Swift nodes"

If SwiftStack container sync is implemented, additional configuration changes must be made to that service to work with encryption. SwiftStack encryption is also sensitive to the object paths

and expects them to remain unchanged. SwiftStack recommends against implementing features which may change the object path when using encryption.

Another limitation of encryption for SwiftStack is that the metadata search for SwiftStack cannot be used with encryption. Since metadata search is implemented by a separate SQLite database and encryption is implemented within the proxy nodes, the two systems are independent of each other and the SQLite database has no access to the encrypted data.

ECS implements Data-at-Rest encryption (D@RE) that is FIPS-140-2 Level 1 compliant as part of its storage engine. FIPS-140-2 is the Federal standard that governs cryptography for use by the US Government and other regulated industries, such as healthcare and finance. Level 1specifies the standard appropriate for software only implementations. No additional hardware is required to enable D@RE in ECS and D@RE can be easily enabled at the namespace, and bucket level through simple GUI switches, and at the object level through S3 encryption semantics. Since encryption is implemented on the node as the data is stored, there is no concern about data being intercepted within the ECS cluster while unencrypted. D@RE support in ECS is transparent to the client and clients only need to concern themselves with encryption if they are using S3 encryption semantics directly. Keys are automatically managed by ECS and stored internally.

Beginning with ECS 3.1, metadata search remains functional with D@RE. Metadata search in ECS is implemented as an integral part of the storage engine without the need to run any additional database and allows searching of up to 30 pre-defined fields. Since it is integral to ECS, metadata search is managed by each of the nodes and scales along with the ECS installation as nodes are added.

ECS also offers advanced retention management capabilities with SEC 17-A4 compliance. With this level of compliance, ECS meets regulatory requirements for data retention, indexing, and accessibility outlined by the SEC for the management of data related to trade, brokering, and financial securities such as stock, bonds, and futures. ECS also offers retention control based on litigation hold, event based retention, and min/max governance. Litigation hold enables the application to temporarily prevent deletion or modification of an object that is subject to an investigation or legal action. Event based retention allows the application to specify retention periods based on specific events. With min/max governance, the administrator can specify a minimum and maximum value for the default retention period.

Takeaway

ECS offers FIPS-140-2 Level 1 complaint Data-at-Rest Encryption (D@RE) which offers greater data security and flexibility than SwiftStack's implementation of encryption. There is no additional installation or configuration required to enable D@RE in ECS as opposed to the many installation and configuration steps for SwiftStack. Since encryption is implemented within the storage engine of the ECS node, there is no data vulnerability due to inter-node network communication such as that which exists in SwiftStack. Metadata search remains functional with ECS encryption but not with SwiftStack's implementation. ECS also provides SEC 17-A4 compliant advanced retention management capabilities which SwiftStack lacks.

Conclusion

ECS and Swift are both object storage solutions that easily integrate with OpenStack and function as standalone storage solutions. Both interface with OpenStack using the Swift object data and Keystone authentication interfaces. Swift began as an object storage module just for OpenStack and was later made into the commercial product SwiftStack. ECS is the third-generation object storage solution from Dell EMC and was developed from-the-ground-up to be an enterprise-quality object storage solution.

Swift, as with many open-source solutions, has grown over time to include many new features that were not initially planned. The addition of these new features into the architecture varies in complexity and elegance, depending on how well the underlying architecture can adapt to these new features. Implementations can require complex installation procedures or compromises to feature capabilities to accommodate these extensions.

SwiftStack *is an effort* to commercialize the open-source Swift module. SwiftStack attempts to address some of the more glaring difficulties of installing Swift by providing a GUI interface. The GUI interface, however, requires setting up a separate server to run it or subscribing to a cloud based service which does the same thing. SwiftStack, as with many open source solutions, offers the promise of lower investment costs in software licenses and/or hardware. This, however, is often accompanied with higher requirements for administration and complexity which bring additional costs in support, training, and expertise through additional hiring or consultation.

ECS hides complexity of the system behind well thought out graphical user interfaces and is built keeping in mind that enterprise users do not want to experiment and may not be interested in having to understand the internal workings of their storage. ECS features are implemented and built by the same team that developed the architecture and thus is able to tightly integrate these features with the core storage engine.

ECS is available as a turnkey appliance for convenience, a software defined solution for flexibility, and as a cloud-based solution for those who prefer not to manage their own hardware or would prefer an OPEX vs CAPEX model. ECS features are designed with integration and business needs in mind, including certification and regulatory requirements. As an enterprise IT supplier, Dell EMC provides the reliability, support, and services that enterprises expect from a tier 1 supplier.

The origin of companies like SwiftStack were made possible because OpenSource applications do not have sufficient Enterprise-Grade capabilities. While it makes sense to explore OpenSource-based applications, Edison has found that in many cases there are trade-offs where performance, operational efficiencies, and functionality are sacrificed for cost. After a detailed comparison of ECS to SwiftStack, it is clear that ECS is the better alternative to managing object storage.