EMC ELASTIC CLOUD STORAGE (ECS)

Overview and Architecture

ABSTRACT

This document provides an in-depth architecture overview of EMC® Elastic Cloud Storage (ECS™), a turnkey software-defined cloud-scale object storage platform that combines the cost advantages of commodity infrastructure with the reliability, availability and serviceability of traditional arrays.

January, 2016

WHITE PAPER

Part Number H14071.4

TABLE OF CONTENTS

INTRODUCTION

EMC® Elastic Cloud Storage™ (ECS) is a software-defined, cloud-scale, object storage platform that combines the cost advantages of commodity infrastructure with the reliability, availability and serviceability of traditional arrays. With ECS, any organization can deliver scalable and simple public cloud services with the reliability and control of a private-cloud infrastructure.

ECS provides comprehensive protocol support for unstructured (Object and File) workloads on a single, cloud-scale storage platform. With ECS, you can easily manage your globally distributed storage infrastructure under a single global namespace with anywhere access to content. ECS features a flexible software-defined architecture that is layered to promote limitless scalability. Each layer is completely abstracted and independently scalable with high availability and no single points of failure. Any organization can now realize true cloud-scale economics in their own data centers.

AUDIENCE

This paper is intended for EMC field personnel and customers who are interested in understanding the value and architecture of ECS. It provides an in-depth overview of ECS software, hardware, networking and services. It also provides links to other ECS information.

SCOPE

This document focuses primarily on ECS architecture. It does not cover installation, administration, and upgrade procedures for ECS software or hardware. It also does not cover specifics on using and creating applications with the ECS APIs. It does provide an overview of available connectors and integrations and links to more information.

VALUE OF ECS

ECS provides significant value for enterprises and service providers seeking a platform architected to support rapid data growth. The main advantages of ECS that enable enterprises to globally manage and store distributed content at scale include:

- **Cloud-scale** – ECS provides a simple, flexible, cloud-scale architecture that offers unparalleled levels of availability, protection and geo-scale. Large multinationals, web companies and service providers can accelerate multi-petabyte deployments for Web-scale applications and Big Data analytics.

- **Multi-Protocol Access** – supports Object, NFS and HDFS workloads on a single platform to meet the broadest range of application needs and to simplify development.

- **Active-Active Read/Write Architecture** – ECS features a multi-site, active-active architecture that features strong consistency semantics to ensure applications always access the most recent copy of data. A single global namespace enables ubiquitous access to content from anywhere. Built-in metadata management and metadata search allow developers to access metadata directly without having to write additional code to connect and query a separate metadata database.

- **Geo-Replicated Data Protection** – ECS geo-protection provides enhanced protection against site failures. It implements mirroring and erasure coding for data protection to avoid single points of failures. In addition, ECS stores files efficiently with very low overhead. ECS features < 1.8x storage overhead when distributing objects over 4 sites. You get the best of both worlds and do not have to make the trade-off between file access and storage efficiency.

- **Multi-tenancy** – ECS multi-tenant capabilities provide the elasticity needed to expand or amend storage service offerings to meet the demands of a diversified clientele and to ensure the integrity of your data stored on common arrays. Not only does ECS enable organizations to securely segregate customers within a single storage system, it also has monitoring and reporting capabilities that can be customized to provide tenants and other users with unique tailored views of their multi-data-type storage environments and chargeback reports for billing.

- **Performance** – ECS' unique "box-carting" capability ingests and reads large volumes of small files efficiently and provides faster performance for analytics. ECS also features geo-caching to provide great performance across multiple sites, maintain high availability, and decrease latency—this is especially useful for organizations with a global footprint.

- **Flexible** –Organizations can build their own customized scale-out object storage platform by deploying ECS Software on certified commodity hardware – or – drop in a purpose-built, cloud-ready turnkey appliance in your data center today.

- **Monitoring and Diagnostics** – ECS includes key storage engine and geo-replication metrics, including reporting over a 30-day interval, monitoring and diagnostics through the REST API or the ECS Portal , as well as drill down graphics, trend-line charts, and an overall system-level dashboard. This increases system visibility to measure usage and provide reports to tenants whether they're internal or external customers.

The design of ECS is optimized for the following primary use cases:

- **Global Content Repository** –enables any organization to consolidate multiple storage systems into a single, globally accessible and efficient content repository.

- **Modern Application Platform** –provides a single turnkey platform designed for modern application development, management and analytics.

- **Data Lake Foundation** –establishes a data lake foundation for organizations of any size. ECS fully maximizes user data with its powerful HDFS service, making Big Data applications and analytics a production reality.

- **Cold Archive** - serves as a secure and affordable on-premise cloud for archival and long-term retention purposes.

- **'Internet of Things' Cloud Storage Platform** –offers an efficient 'IoT' architecture for unstructured data collection at massive scale.

ARCHITECTURE

ECS is architected with certain design principles, such as global namespace with strong consistency; limitless capacity and scale-out, secure multi-tenancy; and superior performance for both small and large objects. ECS was built as a completely distributed system following the principle of cloud applications. In this model, each hardware node is identical to other nodes in the system. The ECS software running on those nodes forms the underlying cloud storage, providing protection, geo replication, and data access. This section will go in-depth into the ECS architecture and design of both the software and hardware.

OVERVIEW

ECS provides a software-defined cloud storage platform that can be deployed on a set of qualified commodity hardware or a turnkey storage appliance. At a high level ECS is composed of the following main components:

- **ECS Portal and Provisioning Services** – provides a Web-based portal that allows self-service, automation, reporting and management of ECS nodes. It also handles licensing, authentication, multi-tenancy, and provisioning services.

- **Data Services** – provides services, tools and APIs to support Object, and HDFS and NFSv3.

- **Storage Engine** – responsible for storing and retrieving data, managing transactions, and protecting and replicating data.

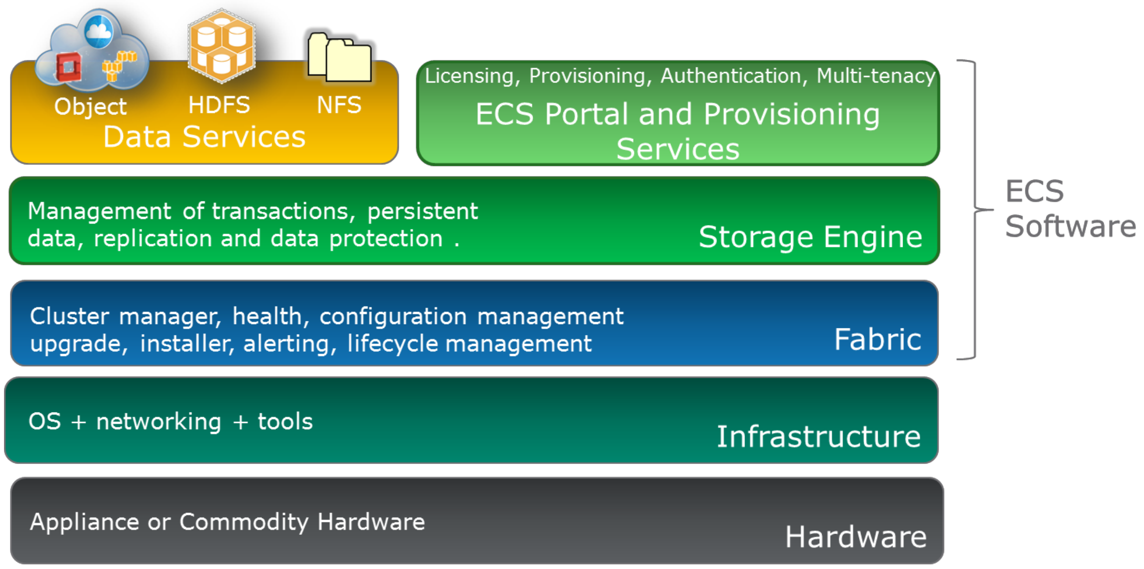
- **Fabric** – provides clustering, health, software and configuration management as well as upgrade capabilities and alerting.

- **Infrastructure** – uses SUSE Linux Enterprise Server 12 as the base operating system for the turnkey appliance or qualified Linux operating systems for a DIY commodity hardware configuration.

- **Hardware** – offers a turnkey appliance or qualified commodity hardware.

Figure 1 shows a graphical view of these layers. Each layer is described in more detail in the subsections that follow.

Figure 1 - ECS Architecture Overview


Object
HDFS
NFS
Data Services
Licensing, Provisioning, Authentication, Multi-tenacy
ECS Portal and Provisioning Services
Management of transactions, persistent data, replication and data protection .
Storage Engine
Cluster manager, health, configuration management upgrade, installer, alerting, lifecycle management
Fabric
OS + networking + tools
Infrastructure
Appliance or Commodity Hardware
Hardware
ECS Software

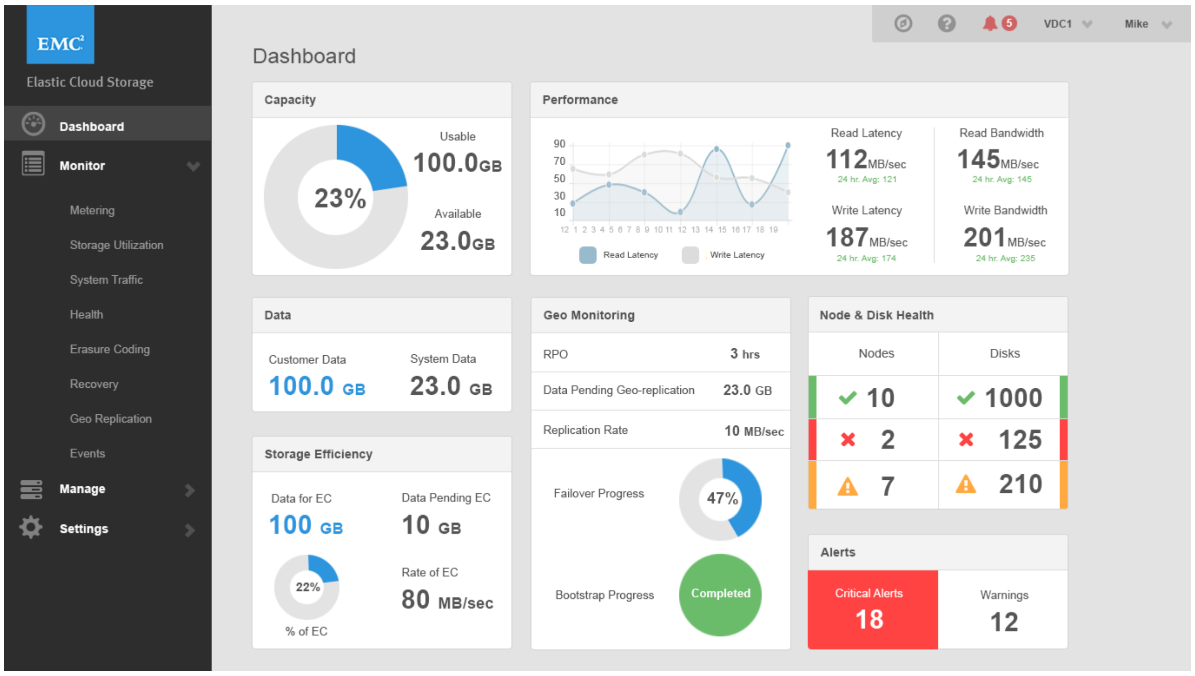
ECS PORTAL AND PROVISIONING SERVICES

ECS management is done through the ECS Portal and provisioning services. ECS provides a Web-based GUI that allows you to manage, license, and provision ECS nodes. The portal has comprehensive reporting capabilities that include:

- capacity utilization per site, storage pool, node and disk
- performance monitoring on latency, throughput, transactions per second, and replication progress and rate
- diagnostic information, such as node and disk recovery status and per-node statistics on hardware and process health, which helps identify performance and system bottlenecks.

The ECS dashboard provides overall system-level health and performance information. This unified view enhances overall system visibility. Alerts notify users about critical events, such as capacity limits, quota limits, disk or node failures, or software failures. A screenshot of the ECS Dashboard appears in Figure 2.

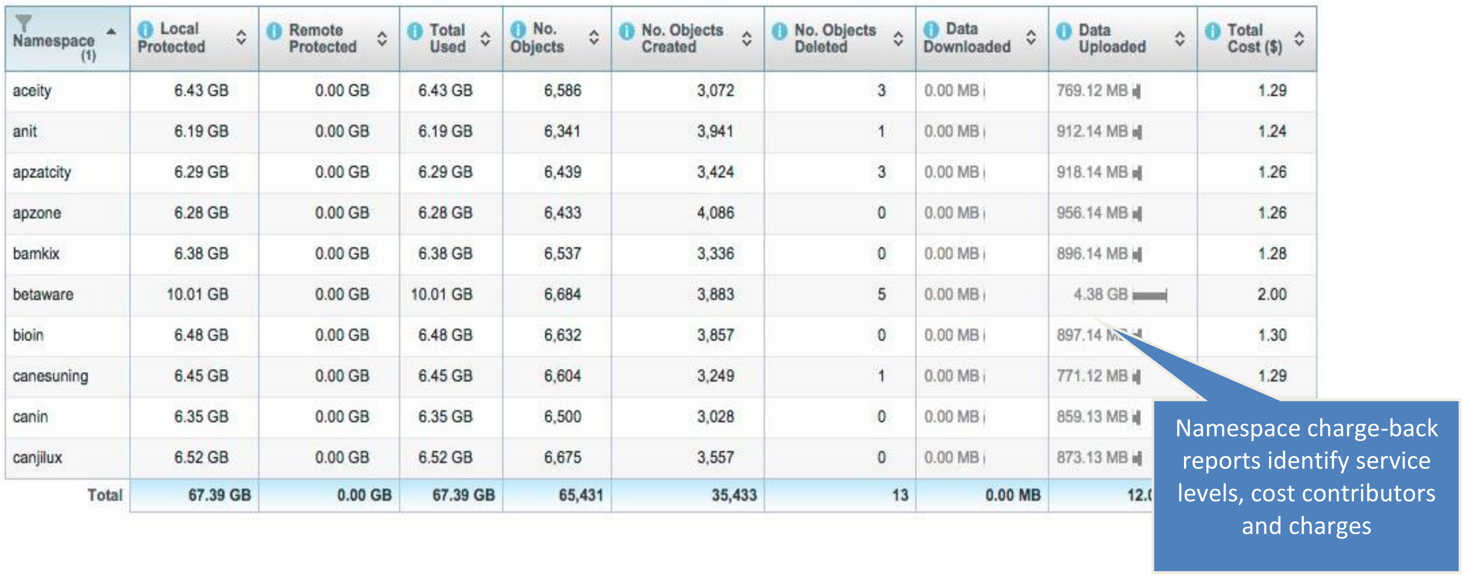
Figure 2 - ECS Dashboard.



ECS also provides a command-line interface to install, upgrade, and monitor ECS. Access to nodes for command-line usage is done via SSH. You can also manage ECS using RESTful APIs, which allow users to administer the ECS system within their own tools, scripts or existing applications. The ECS Portal and command-line tools were built using the ECS REST Management APIs.

In ECS 2.0 and later, ECS is integrated into VIPR SRM and provides dashboards and reports relating to object utilization. For instance, object dashboards now include a summary of configured usable capacity, capacity trends, and configurable usable capacity by service level. The inventory namespace report provides detailed information on quota status and used capacity by namespace. Namespace chargeback reports show total used capacity for local and remote systems and the total number of objects per namespace to identify service levels, cost contributors and charges. Bucket-level reports provide details on the number of objects as well as used capacity and the percentage of quota used by a bucket. Through ViPR SRM you also can view performance and capacity trends over a specified period of time.   Please refer to ViPR SRM on EMC's website for more information. Figure 3 shows a screenshot of a Namespace Chargeback Report in ViPR SRM.

Figure 3 - Namespace Chargeback Reports


Namespace charge-back reports identify service levels, cost contributors and charges

Namespace (1)	Local Protected	Remote Protected	Total Used	No. Objects	No. Objects Created	No. Objects Deleted	Data Downloaded	Data Uploaded	Total Cost ($)
aceity	6.43 GB	0.00 GB	6.43 GB	6,586	3,072	3	0.00 MB	769.12 MB	1.29
anit	6.19 GB	0.00 GB	6.19 GB	6,341	3,941	1	0.00 MB	912.14 MB	1.24
apzatcity	6.29 GB	0.00 GB	6.29 GB	6,439	3,424	3	0.00 MB	918.14 MB	1.26
apzone	6.28 GB	0.00 GB	6.28 GB	6,433	4,086	0	0.00 MB	956.14 MB	1.26
bamkix	6.38 GB	0.00 GB	6.38 GB	6,537	3,336	0	0.00 MB	896.14 MB	1.28
betaware	10.01 GB	0.00 GB	10.01 GB	6,684	3,883	5	0.00 MB	4.38 GB	2.00
bioin	6.48 GB	0.00 GB	6.48 GB	6,632	3,857	0	0.00 MB	897.14 M	1.30
canesuning	6.45 GB	0.00 GB	6.45 GB	6,604	3,249	1	0.00 MB	771.12 MB	1.29
canin	6.35 GB	0.00 GB	6.35 GB	6,500	3,028	0	0.00 MB	859.13 MB	
canjilux	6.52 GB	0.00 GB	6.52 GB	6,675	3,557	0	0.00 MB	873.13 MB	
Total	67.39 GB	0.00 GB	67.39 GB	65,431	35,433	13	0.00 MB		12.

DATA SERVICES

Access to data stored in ECS is through Object, HDFS and NFS v3 protocols. In general, ECS provides multi-protocol access, meaning data ingested through one protocol can be accessed through another. For example, data can be ingested through S3 and modified through NFSv3 or HDFS (or vice versa).   There are some exceptions to this multi-protocol access due to protocol semantics and representations of how the protocol was designed. Table 2 highlights the Object APIs and protocol supported and which protocols interoperate.

Table 2 - ECS Supported Data Services

Protocols		Supported	Interoperability
Object	S3	Additional capabilities like Byte Range Updates and Rich ACLS	HDFS, NFS
	Atmos	Version 2.0	N/A
	Swift	V2 APIs and Swift Authentication	HDFS, NFS
	CAS	SDK v3.1.544 or later	N/A
HDFS		Hadoop 2.7 compatibility	S3, Swift, NFS
NFS		NFSv3	S3, Swift, HDFS

The data services, which are also referred to as head services, are responsible for taking client requests, extracting required information, and passing it to the storage engine for further processing (e.g. read, write, etc.). There are one or more processes running on the infrastructure layer to handle each of the protocols. For example, a process called hdfssvc, cassvc, objectheadsvc, and filesvc runs on the infrastructure layer. The processes are further encapsulated within a Docker container named object-main, which runs on every node within the ECS system. The Infrastructure section of this document covers this topic in more detail. Also, in order to access objects via the above protocols, certain firewall ports need to be opened. For more information on ports refer to the ECS Security Configuration Guide.

OBJECT

For object access, ECS provides industry-standard object APIs. ECS supports S3, Atmos, Swift and CAS APIs. With the exception of CAS, objects or data are written, retrieved, updated, and deleted via HTTP or HTTPS calls of GET, POST, PUT, DELETE and HEAD. For CAS, standard TCP communications and specific access methods and calls are used.

In addition, ECS provides a facility for metadata search of objects. This enables ECS to maintain an index of the objects in a bucket, based on their associated metadata, allowing S3 object clients to search for objects within buckets based on the indexed metadata using a rich query language. Search indexes can be up to 5 metadata fields per bucket and are configured at the time of bucket creation through the ECS Portal, ECS Management REST API, or S3 REST API.

For CAS objects, CAS query API provides similar ability to search for objects based on metadata that is maintained for CAS objects, and does not need to be enabled explicitly.

For more information on ECS APIs and APIs for metadata search, see the ECS Data Access Guide. For Atmos and S3 SDKs, refer to the GitHub site: EMC Data Services SDK. For CAS, refer to the Centera Community site. You can also access numerous examples, resources and assistance for developers in the ECS Community.

If you have access to an ECS system, there are existing client applications that provide a way to quickly test or access data stored in ECS—for example, the S3 Browser and Cyberduck. Another option is to use the ECS Test Drive, which allows you to gain access to ECS's object protocols for testing and development purposes. After registering for ECS Test Drive, you receive REST endpoints and can create and/or manage user credentials for each of the object protocols. With the endpoints, you can use clients like S3 Browser and Cyberduck to read and write objects.

HDFS

ECS can store Hadoop file system data, which allows organizations to create Big Data repositories on ECS that Hadoop analytics can consume and process. The HDFS data service is compatible with Apache Hadoop 2.7, with support for fine-grained ACLs and extended filesystem attributes.

In ECS 2.2 and later, ECS has been integrated with Ambari, which allows you to more easily deploy ECS HDFS clients and specify ECS HDFS as the default filesystem in a Hadoop cluster. Other enhancements added in ECS 2.2 include the following:

- Proxy User Authentication – impersonation for Hive, HBase, and Oozie.
- Security - server-side ACL enforcement and addition of Hadoop superuser and superuser group as well as default group on buckets.

ECS 2.2 has been validated and tested with Hortonworks (HDP 2.3), Cloudera (CDH 5.4) and Pivotal (PHD 3.0). It also has support for services such as YARN, MapReduce, Pig, Hive/Hiveserver2, HBase, Zookeeper, Flume, Spark and Sqoop.

FILE

In ECS 2.2 and later includes native file support with NFSv3. The main features for the NFSv3 file data service include:

- Rich ACLs – support for access control lists.
- Global namespace – ability to access the file from any node at any site (with a load balancer).
- Global locking – ability to lock files from any node at any site (shared and exclusive locks, range based locks, and mandatory locks)
- Multi-protocol access – ability to access data created by object (S3 and Swift), HDFS, and NFS.

NFSv3 is configurable via the ECS Portal. NFS exports, permissions, and user group mappings are created through the API and/or the portal. In addition, NFS clients such as Solaris, Linux, and Windows can mount the export specified in the ECS Portal using namespace and bucket names. For example: *mount –t nfs –o vers=3 <ip of node or DNS name>:<export path(ie. /namespace/bucket)>.* To achieve client transparency during a node failure, a load balancer must be used.

ECS has tightly integrated the other NFS server implementations, such as lockmgr, statd, nfsd, and mountd, hence, these services are not dependent on the infrastructure layer (host operating system) to manage.

NFS v3 support has the following features:

- No design limits on the number of files or directories
- File size can be up to 4TB
- Ability to scale across 8 sites with a single global namespace/share
- Support for Kerberos and AUTH_SYS Authentication

NFS file services process NFS requests coming from clients; however, data is stored as objects, similar to the object data service. An NFS file handle is mapped to an object id. Since the file is basically mapped to an object, NFS has features similar to the object data service, including:

- Quota Management at the bucket level
- Retention (WORM)
- Encryption at the object level

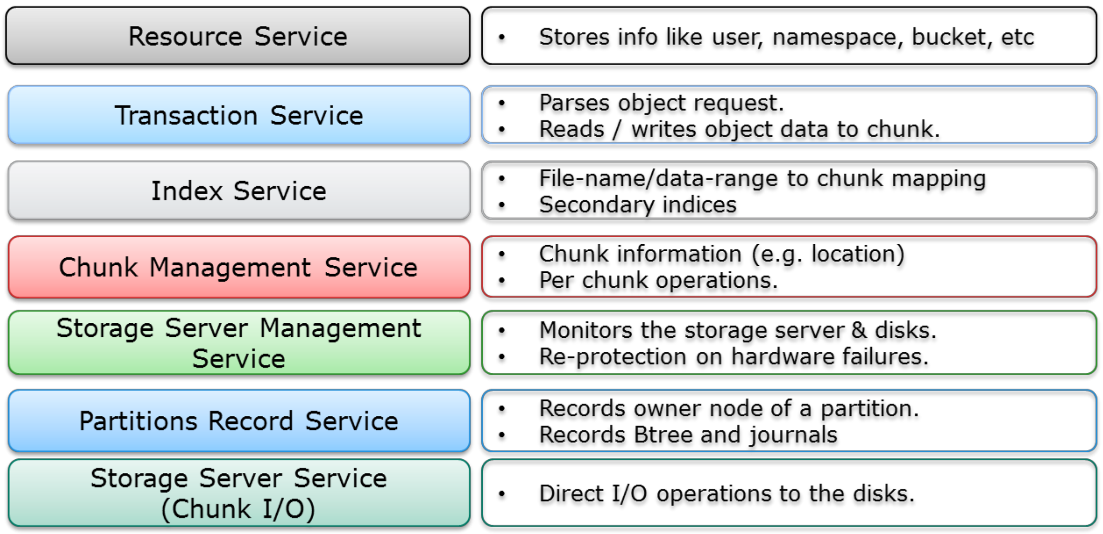
STORAGE ENGINE

At the core of ECS is the storage engine. This layer contains the main components that are responsible for processing requests as well as storing, retrieving, protecting and replicating data. This section describes the design principles and the way that data is represented and handled internally. Data flow for reads and writes is also described.

SERVICES

ECS has a layered architecture, with every function in the system built as an independent layer. This design principle makes each layer horizontally scalable across all nodes in the system and ensures high availability. The ECS storage engine includes the layers shown in Figure 4, which run on top of the infrastructure and hardware component.

Figure 4 - Storage Engine Layers


Resource Service
• Stores info like user, namespace, bucket, etc
Transaction Service
• Parses object request.
• Reads / writes object data to chunk.
Index Service
• File-name/data-range to chunk mapping
• Secondary indices
Chunk Management Service
• Chunk information (e.g. location)
• Per chunk operations.
Storage Server Management Service
• Monitors the storage server & disks.
• Re-protection on hardware failures.
Partitions Record Service
• Records owner node of a partition.
• Records Btree and journals
Storage Server Service (Chunk I/O)
• Direct I/O operations to the disks.

The services of the Storage Engine are encapsulated within a Docker container and installed on each ECS node, providing a distributed and shared service.

DATA AND METADATA

The details of data stored in ECS can be summarized as follows:

- **Data** – the actual content that needs to be stored (image, document, text, etc.)

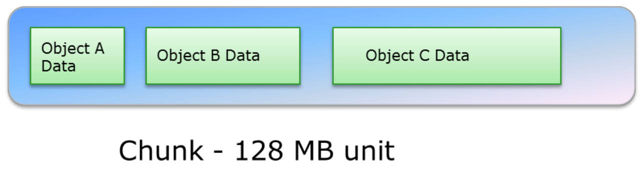
- **System Metadata** – information and attributes relating to the data. System metadata can be categorized as follows:
  - *Identifiers and descriptors* — A set of attributes used internally to identify objects and their versions. Identifiers are either numeric ids or hash values which are not of use outside the ECS software context. Descriptors define information such as type of content and encoding.

- *Encryption keys in encrypted format* – Object encryption keys are included as part of system metadata after being encrypted by KEKs(key encryption keys)
- *Internal flags* – A set of flags to track if byte range updates or encryption are enabled, as well as to coordinate object locks, caching and deletion
- *Location information* – A set of attributes with index and data location information such as byte offsets.
- *Timestamps* – A set of attributes that track time of object creation, modification and expiration .
- *Configuration/tenancy information* – Object names, namespace name , zone/vdc name and access control information for objects.

- **Custom User metadata** –user-defined metadata, which provides further information or categorization of the data that is being stored. Custom metadata is formatted as key-value pairs that are sent with a write request (e.g. Client=EMC, Event=EMCWorld, ID=123).

All types of data, including system and custom metadata, are stored in "**chunks**". A chunk is a 128MB logical container of contiguous space. Note that each chunk can have data from different objects, as shown in Figure 5. ECS uses indexing to keep track of all the parts of an object that may be spread across different chunks. Chunks are written in an append-only pattern, meaning, an application cannot modify/delete existing data within a chunk but rather updated data is written in a new chunk. Therefore, no locking is required for I/O and no cache invalidation is required. The append-only design also simplifies data versioning. Old versions of the data are maintained in previous chunks. If S3 versioning is enabled and an older version of the data is needed, it can be retrieved or restored to a previous version using the S3 API.

Figure 5 - Chunk (can store data from different objects)



All chunks are triple-mirrored to multiple nodes to protect data against drive or node failures. When data-only chunks fill up to 128MB and/or are sealed, the chunk is erasure coded for storage efficiency and spread across multiple nodes for data protection. Once a chunk has been erasure coded and protected across the nodes and disks, the mirrored copies are discarded. The chunks containing index and metadata are triple-mirrored and are not erasure coded. A more detailed description of triple mirroring and erasure coding are provided in the Data Protection section of this document.

DATA MANAGEMENT (INDEX AND PARTITION RECORDS)

ECS uses logical tables to keep track of where data and metadata chunks are stored on disk. The tables hold key-value pairs to store information relating to the objects. A hash function is used to do fast lookups of values associated with a key. These key-value pairs are eventually stored in a B+ tree for fast indexing of data locations. By storing the key-value pair in a balanced, searched tree like a B+ Tree, the location of the data and metadata can be accessed quickly. Entries in these logical tables are first recorded in journal logs on disks that are also stored in triple-mirrored chunks. The journals keep track of index transactions not yet committed to the B+ tree. A background process eventually processes these entries and stores the key-value pairs in a B+ tree on disk.

For instance, the object table in Figure 6 contains the name of an object and the chunk location (Chunk 1) at a certain offset and length within that chunk. In this table, the object name is the key to the index and the value is the chunk location. The index layer within the Storage Engine is responsible for the object-name-to-chunk mapping.

Figure 6 – Object Table

Object Name	Chunk Location
ImgA	C1:offset:length
FileB	C2:offset:length C3:offset:length

Another table referred to as the Chunk Table records which node the chunk resides on. For data integrity, the chunks are triple-mirrored to different disks on different nodes and this table keeps track of where the chunk location resides. This table keeps track of which nodes the chunk resides on, the disk within the node, the file within the disk, the offset within that file and the length of the data, as shown in Figure 7.

Figure 7 – Chunk Table

Chunk ID	Location
C1	Node1:Disk1:File1:Offset1:Length Node2:Disk2:File1:Offset2:Length Node3:Disk2:File6:Offset:Length

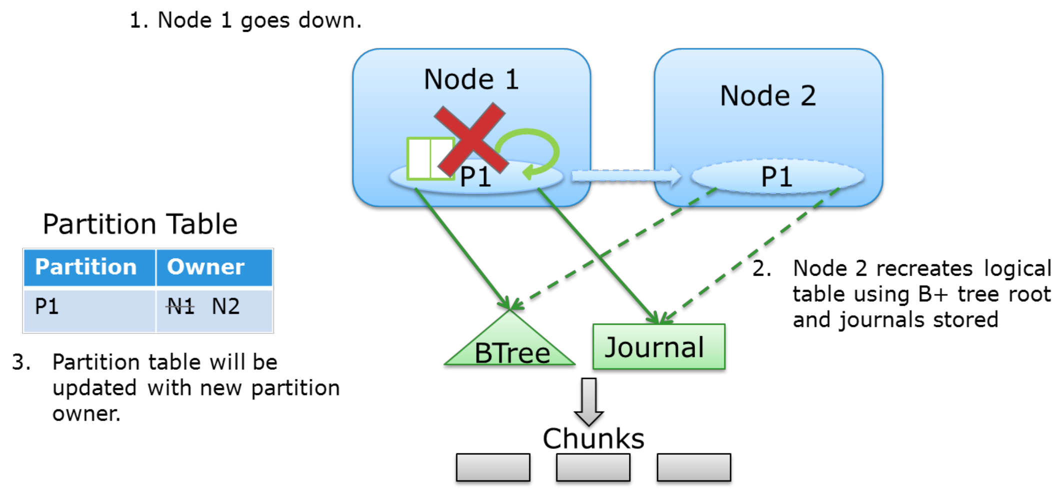
As mentioned previously, ECS was designed to be a distributed system such that storage and access of data are spread across the nodes. Since these tables can get very large, these logical tables are divided into partitions and assigned to different nodes. The node then becomes the owner of that partition or section of the table. So to get the location of a chunk would require retrieving the information from the partition owner. A partition record table is illustrated in Figure 8 below.

Figure 8 - The Partition Records Table

Partition ID	Owner
P1	Node 1
P2	Node 2
P3	Node 3

If the node goes down, another node takes ownership of the partition. The logical tables owned by the unavailable node get recreated on a new node and that node becomes the partition owner. The table is recreated using the B+ tree root stored on disk and replaying the journals also stored on disk. As previously mentioned, the B+ tree and journals are stored in chunks (just like data) and are triple-mirrored. Figure 9 shows the failover of partition ownership.

Figure 9 - Failover of Partition Ownership


1. Node 1 goes down.
Node 1
P1
Node 2
P1
2. Node 2 recreates logical table using B+ tree root and journals stored
BTree
Journal
Chunks
Partition Table
Partition | Owner
P1 | ~~N1~~ N2
3. Partition table will be updated with new partition owner.

DATA FLOW

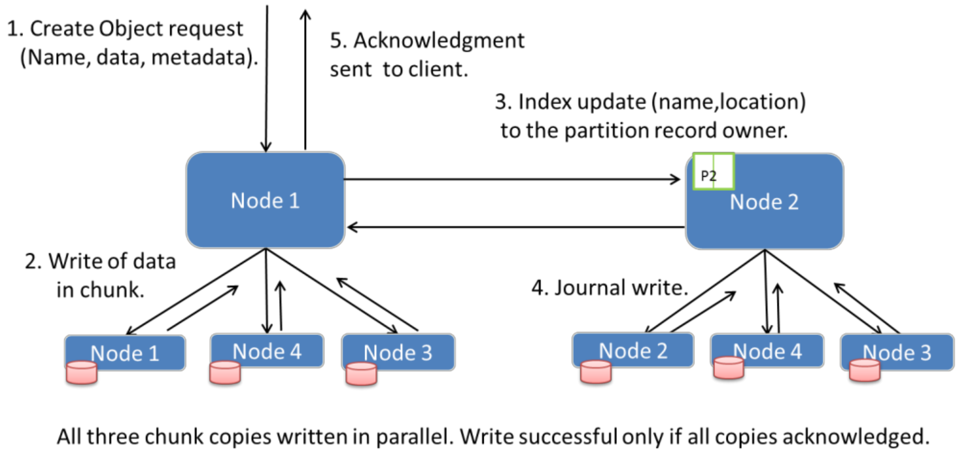
Data can be accessed by any of the ECS nodes and written to three different nodes (triple-mirrored) within ECS for data protection and resiliency. Also, prior to writing data to disk, ECS runs a checksum function and stores the result and data is compressed. Similarly with reads, data is decompressed and checksum is validated. The data flow of reads and writes are described below.

The data flow for writes is as follows (shown in Figure 10):

1. A create object is requested from client and is directed to one of the nodes (Node 1 in this case) who will process the request.
2. The data are written to a new chunk or appended to an existing chunk and triple-mirrored to three different nodes in parallel.
3. Once the three copies are acknowledged to have been written, an update to the logical tables (index) of partition owner occurs with name and chunk location.
4. The partition owner of index ("Node 2" in this case) records this write transaction into the journal logs which is also written into chunks on disk and triple mirrored to three different nodes in parallel.
5. Once the transaction has been recorded in the journal logs, an acknowledgement is sent to the client.

In a background process, journal entries are processed and persisted onto disk in B+ trees as chunks and triple-mirrored.

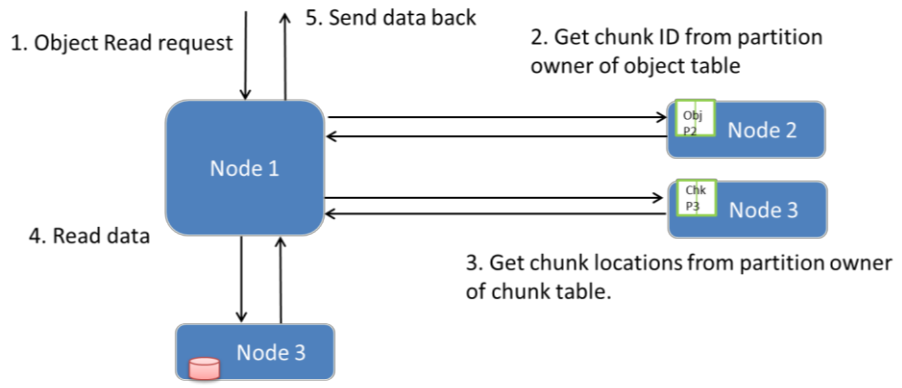
Figure 10 - Data Write Flow


1. Create Object request (Name, data, metadata).
5. Acknowledgment sent to client.
3. Index update (name,location) to the partition record owner.
Node 1
P2
Node 2
2. Write of data in chunk.
4. Journal write.
Node 1
Node 4
Node 3
Node 2
Node 4
Node 3
All three chunk copies written in parallel. Write successful only if all copies acknowledged.

The data flow for reads include the following (shown in Figure 11):

1. An object read request is received and directed to any available node to process. Node 1 in this example.
2. As mentioned in the Data Management section of this paper, there are logical tables (key-value pairs) containing information to the location of the object, like object table and chunk table. These tables are partitioned and are distributed among the nodes to own and manage. So, object request will be processed by Node 1 in this case. Node 1 will utilize a hash function using the object name ID to determine which node is the partition owner of the object table in which this object information resides. In this example, Node 2 is owner and thus Node 2 will do a lookup in the object table to get chunk ID. Chunk ID is returned to Node 1.
3. Node 1 will once again determine the partition owner of the chunk ID by using a hash function. In this case, Node 3 is the partition owner of the chunk table that holds the location of this chunk. It will do a lookup in the chunk table to get physical location of object and location information is returned to Node 1 to further retrieve the data. In this example, there are several levels of indirection to get the data; however, if object is requested often, the location of the chunk is cached locally and will not require this step.
4. From the previous step, location of chunk is provided to Node 1 who will then issue a byte offset read request to the node that holds the data, Node 3 in this example, and will send data to Node 1.
5. Node 1 will send data to requesting client.

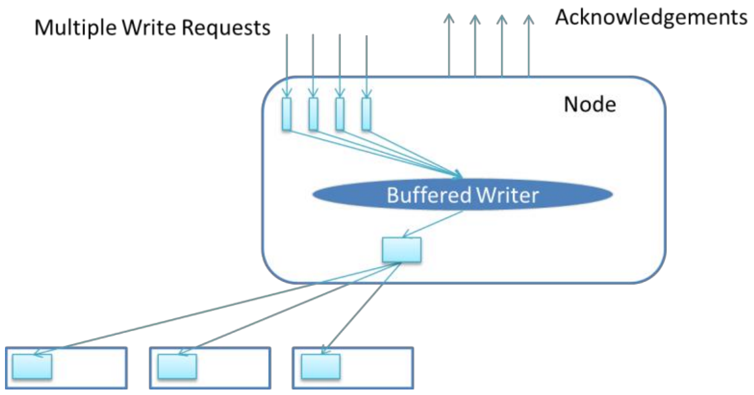
Figure 11 - Data Read Flow


1. Object Read request
5. Send data back
2. Get chunk ID from partition owner of object table
Obj P2
Node 2
Node 1
Chk P3
Node 3
4. Read data
3. Get chunk locations from partition owner of chunk table.
Node 3

BOX-CARTING

ECS has a unique and inherent feature called ***box-carting*** which helps performance for data writes. Box-carting aggregates multiple small data objects queued in memory and then write them in a single disk operation, up to 2MB of data. This improves performance by reducing the number of roundtrips to process individual writes to storage. Figure 12 provides an illustration of box-carting.

Figure 12 - Box-Carting



As for writes of large objects, all nodes within ECS can process write requests for the same object simultaneously and write to different set of disks, taking advantage of all the spindles in the ECS cluster. Thus, ECS can ingest and store small and large objects efficiently.

FABRIC

The Fabric layer provides clustering, system health, software management, configuration management, upgrade capabilities, and alerting. It is responsible for keeping the services running and managing resources such as disks, containers, the firewall, and the network. It tracks and reacts to environment changes such as failure detection and provides alerts related to system health. The fabric layer has the following components to manage the overall system:

- **Node agent** – manages host resources (disks, the network, containers, etc.) and system processes.
- **Lifecyle manager** – application lifecycle management which involves starting services, recovery, notification, and failure detection.
- **Persistence Manager –** provides the coordination and synchronization of the ECS distributed environment.

- **Registry –** stores all the Docker images for ECS.
- **Event Library –** holds the set of events occurring on the system.

NODE AGENT

The node agent is a lightweight agent written in Java that runs natively on every ECS node. Its main duties include managing and controlling host resources (Docker containers, disks, the firewall, the network), and monitoring system processes. Examples of management include formatting and mounting disks, opening required ports, ensuring all processes are running, and determining public and private network interfaces. It also has an event stream that provides ordered events to a lifecycle manager to indicate events occurring on the system. A Fabric CLI allows you to diagnose issues and look at overall system state.

LIFECYCLE MANAGER

Lifecycle manager runs on a subset of nodes (three to five instances) and manages the lifecycle of applications (object-main) running on various nodes. Each lifecycle manager is responsible for tracking a number of nodes. Its main goal is to manage the entire lifecycle of the ECS application from boot to deployment, including failure detection, recovery, notification and migration. It looks at the node agent streams and drives the agent to handle the situation. When a node is down, it responds to failures or inconsistencies in the state of the node by restoring the system to a known good state. If a lifecycle manager instance is down, another one takes its place if available.

PERSISTENCE MANAGER

ECS 2.2 fabric uses zookeeper as its persistence manager. In a later release it will be replaced with vNest, which is a zookeeper alternative with dynamic membership used by services within the object-main container. The persistence manager provides a centralized service for coordinating and synchronizing distributed processes, configuration information, groups, and naming services. Three or five nodes within the ECS system will have an instance of the persistence manager, named fabric-zookeeper, running inside the Docker container.

REGISTRY

The registry contains all the ECS Docker images used during installation, upgrade and node replacement. A Docker container called fabric-registry runs on one node within the ECS rack and holds the repository of ECS Docker images and information required for installations and upgrades. Although the registry is available on one node, all Docker images are locally cached on all nodes.

EVENT LIBRARY

The event library is used within the Fabric layer to expose the lifecycle and node agent event streams. Events generated by the system are persisted onto shared memory and disk to provide historical information on the state and health of the ECS system. These ordered event streams can be used to restore the system to a specific state by replaying the ordered events stored. Some examples of events include node events such as started, stopped or degraded.

INFRASTRUCTURE

Each ECS node runs a specific operating system. ECS Appliance currently runs SuSE Linux Enterprise 12 which acts as the ECS infrastructure. For the DIY solution on commodity hardware, which is currently in limited support status and availability, the operating system can be RedHat Enterprise Linux (RHEL) or CoreOS. Docker is also installed on the infrastructure to deploy the encapsulated ECS layers described in the previous sections. Because ECS software is written in Java, the Java Virtual Machine (JVM) is installed as part of the infrastructure.

DOCKER

ECS runs on top of the operating system as a Java application and is encapsulated within several Docker containers. The containers are isolated but share the underlying operating system resources and hardware. Some parts of ECS software run on all nodes and some run on one or some nodes. The components running within a Docker container include:

- **object-main** – contains the resources and processes relating to the data services, storage engine, portal and provisioning services. Runs on every node in ECS.

- **fabric-lifecycle** – contains the processes, information and resources required for system-level monitoring, configuration management and health management. Depending on the number of nodes in the system, there will be three instances running on a four-node system and five instances for an eight-node system.

- **fabric-zookeeper** – centralized service for coordinating and synchronizing distributed processes, configuration information, groups and naming services. It is referred to as the persistence manager, which runs on three or five nodes depending on the number of nodes deployed within the ECS system.

- **fabric-registry** – location or registry of the ECS Docker images. Only one instance runs per ECS rack.

There are other processes and tools that run outside of a Docker container namely the Fabric node agent and hardware abstraction layer tools. Figure 13 below provides an example of how ECS is run on a four node deployment:

Figure 13 - Docker Containers and Agents

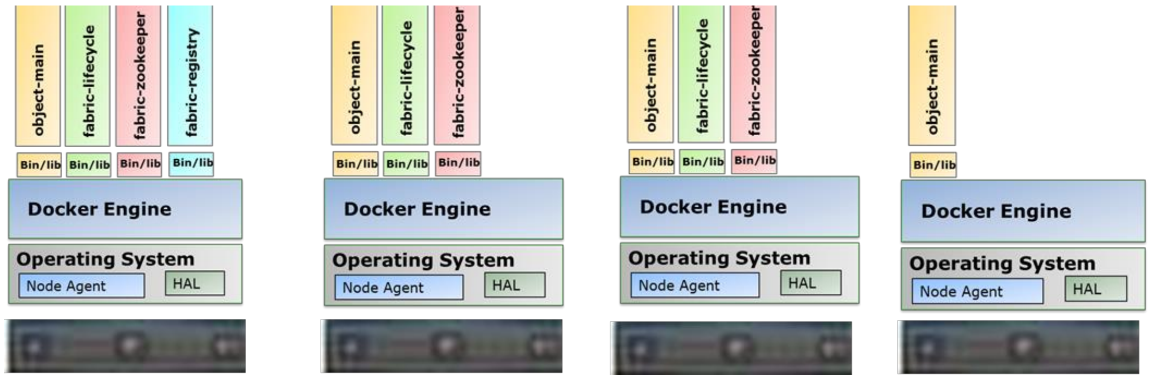
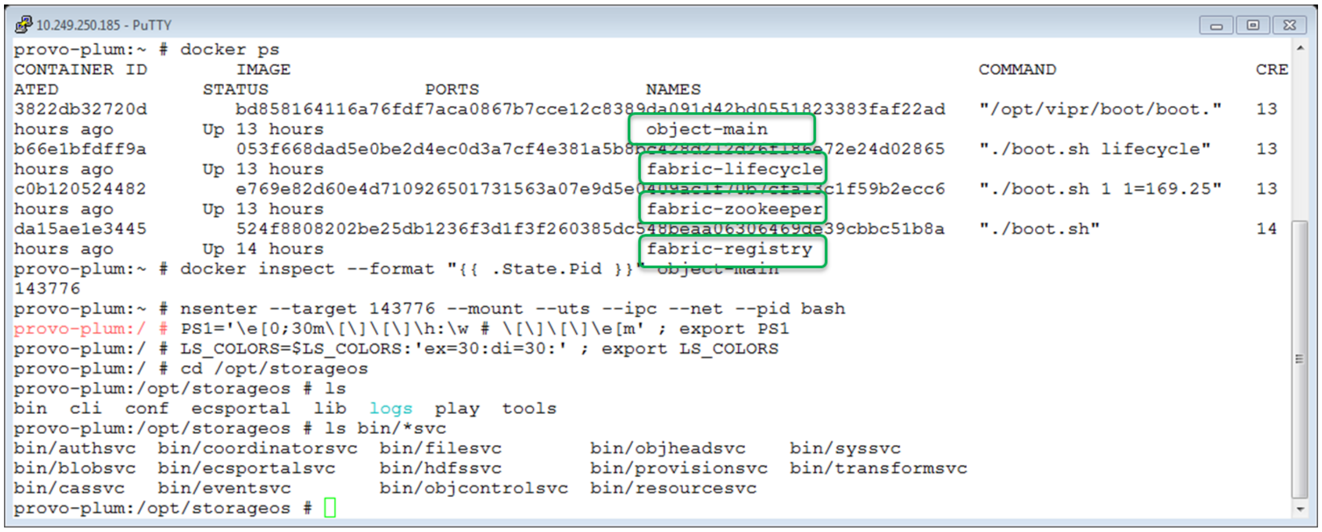


Figure 14 shows a view of the object-main container in one of the nodes of an ECS system. The listing provides the some of the services running in the Docker container.

Figure 14 - Processes, resources, tools and binaries in object-main container

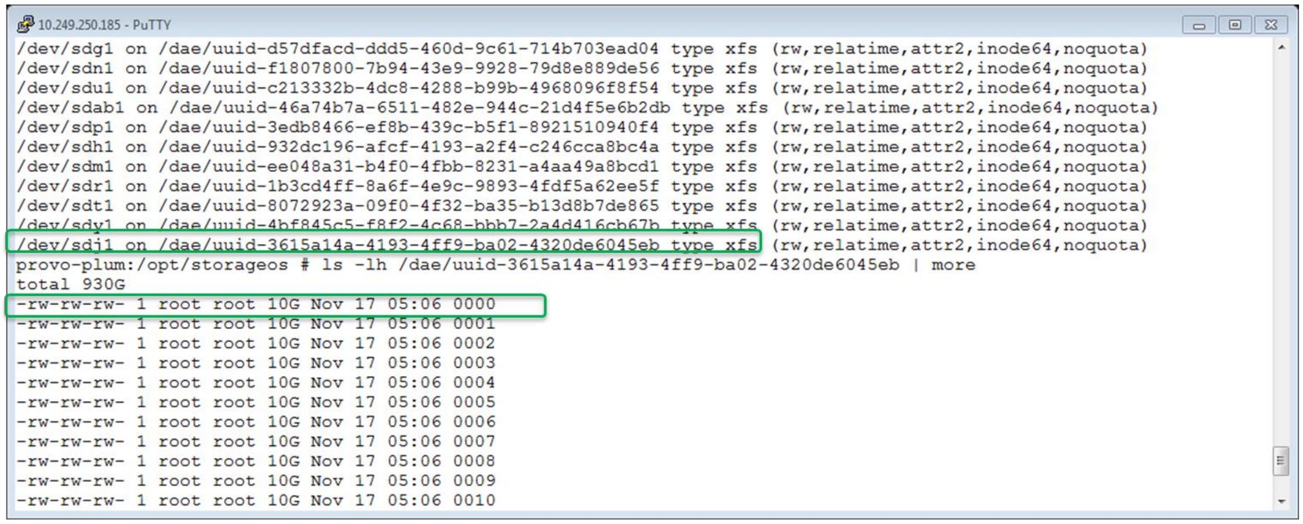


```
provo-plum:~ # docker ps
CONTAINER ID        IMAGE                                                            COMMAND                CRE
ATED           STATUS              PORTS               NAMES
3822db32720d        bd858164116a76fdf7aca0867b7cce12c8389da091d42bd0551823383faf22ad   "/opt/vipr/boot/boot."   13
hours ago           Up 13 hours                             object-main
b66e1bfdff9a        053f668dad5e0be2d4ec0d3a7cf4e381a5b8bc428d212d26f186e72e24d02865   "./boot.sh lifecycle"    13
hours ago           Up 13 hours                             fabric-lifecycle
c0b120524482        e769e82d60e4d710926501731563a07e9d5e0409ac1f70b7cfa13c1f59b2ecc6   "./boot.sh 1 1=169.25"   13
hours ago           Up 13 hours                             fabric-zookeeper
da15ae1e3445        524f8808202be25db1236f3d1f3f260385dc548beaa06306469de39cbbc51b8a   "./boot.sh"              14
hours ago           Up 14 hours                             fabric-registry
provo-plum:~ # docker inspect --format "{{ .State.Pid }}" object-main
143776
provo-plum:~ # nsenter --target 143776 --mount --uts --ipc --net --pid bash
provo-plum:/ # PS1='\e[0;30m\[\]\[\]\h:\w # \[\]\[\]\e[m' ; export PS1
provo-plum:/ # LS_COLORS=$LS_COLORS:'ex=30:di=30:' ; export LS_COLORS
provo-plum:/ # cd /opt/storageos
provo-plum:/opt/storageos # ls
bin  cli  conf  ecsportal  lib  logs  play  tools
provo-plum:/opt/storageos # ls bin/*svc
bin/authsvc  bin/coordinatorsvc  bin/filesvc        bin/objheadsvc    bin/syssvc
bin/blobsvc  bin/ecsportalsvc    bin/hdfssvc        bin/provisionsvc  bin/transformsvc
bin/cassvc   bin/eventsvc        bin/objcontrolsvc  bin/resourcesvc
provo-plum:/opt/storageos #
```

FILESYSTEM LAYOUT

Each ECS node is directly connected to a set of commodity disks. Each disk is formatted and mounted as an XFS filesystem and has a unique identifier (UUID). Data are stored in chunks and the chunks are stored in files within the filesystem. Each disk partition or filesystem is filled with files that are 10GB in size. These files are created during installation such that contiguous blocks are allocated for the files. The number of files within each disk depends on the disk size. For instance, a 1TB disk will have 100, 10GB files. The names of the files inside the disk are "0000", "0001", "0002", and so forth. Figure 15 provides a screenshot of the XFS filesystems and 10GB files populated on one node of the ECS system.

Figure 15- Mounted XFS and 10G Files Storing the Chunks



```
/dev/sdg1 on /dae/uuid-d57dfacd-ddd5-460d-9c61-714b703ead04 type xfs (rw,relatime,attr2,inode64,noquota)
/dev/sdn1 on /dae/uuid-f1807800-7b94-43e9-9928-79d8e889de56 type xfs (rw,relatime,attr2,inode64,noquota)
/dev/sdu1 on /dae/uuid-c213332b-4dc8-4288-b99b-4968096f8f54 type xfs (rw,relatime,attr2,inode64,noquota)
/dev/sdab1 on /dae/uuid-46a74b7a-6511-482e-944c-21d4f5e6b2db type xfs (rw,relatime,attr2,inode64,noquota)
/dev/sdp1 on /dae/uuid-3edb8466-ef8b-439c-b5f1-8921510940f4 type xfs (rw,relatime,attr2,inode64,noquota)
/dev/sdh1 on /dae/uuid-932dc196-afcf-4193-a2f4-c246cca8bc4a type xfs (rw,relatime,attr2,inode64,noquota)
/dev/sdm1 on /dae/uuid-ee048a31-b4f0-4fbb-8231-a4aa49a8bcd1 type xfs (rw,relatime,attr2,inode64,noquota)
/dev/sdr1 on /dae/uuid-1b3cd4ff-8a6f-4e9c-9893-4fdf5a62ee5f type xfs (rw,relatime,attr2,inode64,noquota)
/dev/sdt1 on /dae/uuid-8072923a-09f0-4f32-ba35-b13d8b7de865 type xfs (rw,relatime,attr2,inode64,noquota)
/dev/sdy1 on /dae/uuid-4bf845c5-f8f2-4c68-bbb7-2a4d416cb67b type xfs (rw,relatime,attr2,inode64,noquota)
/dev/sdj1 on /dae/uuid-3615a14a-4193-4ff9-ba02-4320de6045eb type xfs (rw,relatime,attr2,inode64,noquota)
provo-plum:/opt/storageos # ls -lh /dae/uuid-3615a14a-4193-4ff9-ba02-4320de6045eb | more
total 930G
-rw-rw-rw- 1 root root 10G Nov 17 05:06 0000
-rw-rw-rw- 1 root root 10G Nov 17 05:06 0001
-rw-rw-rw- 1 root root 10G Nov 17 05:06 0002
-rw-rw-rw- 1 root root 10G Nov 17 05:06 0003
-rw-rw-rw- 1 root root 10G Nov 17 05:06 0004
-rw-rw-rw- 1 root root 10G Nov 17 05:06 0005
-rw-rw-rw- 1 root root 10G Nov 17 05:06 0006
-rw-rw-rw- 1 root root 10G Nov 17 05:06 0007
-rw-rw-rw- 1 root root 10G Nov 17 05:06 0008
-rw-rw-rw- 1 root root 10G Nov 17 05:06 0009
-rw-rw-rw- 1 root root 10G Nov 17 05:06 0010
```

HARDWARE

ECS software can run on a turn-key EMC appliance or on commodity hardware. Flexible entry points exist with the ability to rapidly scale to petabytes and exabytes of data. With minimal business impact, you can scale ECS linearly in both capacity and performance by simply adding nodes and disks to your environment. The basic hardware required to run ECS includes a minimum of four compute nodes with data disks, a pair of 10GbE switches for data, and a 1GbE management switch.

ECS APPLIANCE

The ECS appliance currently supports two models: U-Series and C-Series. The main difference between the models is that the U-Series is more disk-dense and contain a separate Disk Array Enclosure (DAE) for the disks. The C-Series is more compute-dense and has disks integrated within the ECS server nodes. This section will only highlight the hardware components required to run ECS software. For additional documentation on installing and cabling ECS hardware see the following documents:

- EMC ECS 2.x Product Documentation Index
  - Plan an Installation
  - ECS Hardware and Cabling Guide
  - ECS System Administrator’s Guide
- SolVe Desktop (Procedure Generator) which has information on Switch Configuration Files and Installing ECS 2.x Software Guide. This link may require access to the EMC support website.

U- Series

A standard U-Series appliance contains two 10GbE switches, one 1GbE switch, and four or eight x86 server nodes and disk array enclosures (DAE). Each DAE is connected to one x86 Server by SAS; a four-server model has four disk array enclosures and an eight-server model has 8 disk enclosures. A DAE can have up to 60 disks.

Figures 16 and 17 provide a view of the current U-Models (Gen-1 and Gen-2). Please refer to the ECS Appliance specification sheet for more information on ECS models. The data sheets also contain information about power requirements for each model.

Figure 16 - ECS Appliance Gen-1 Models

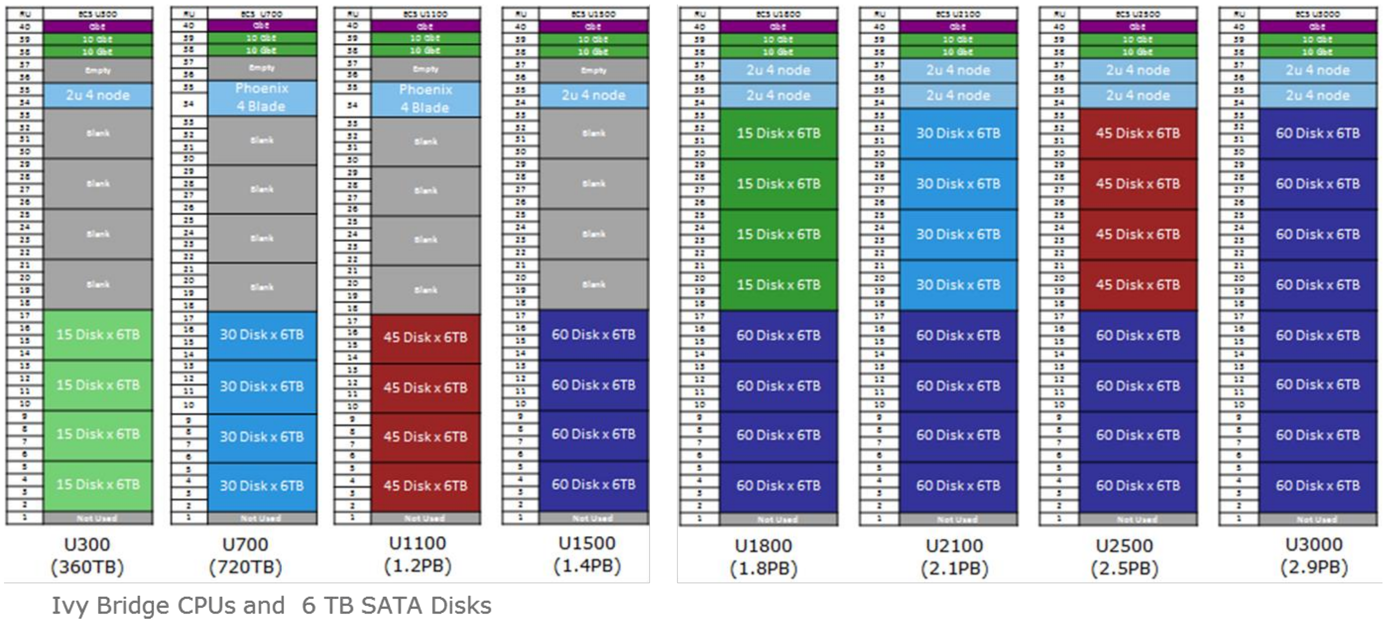
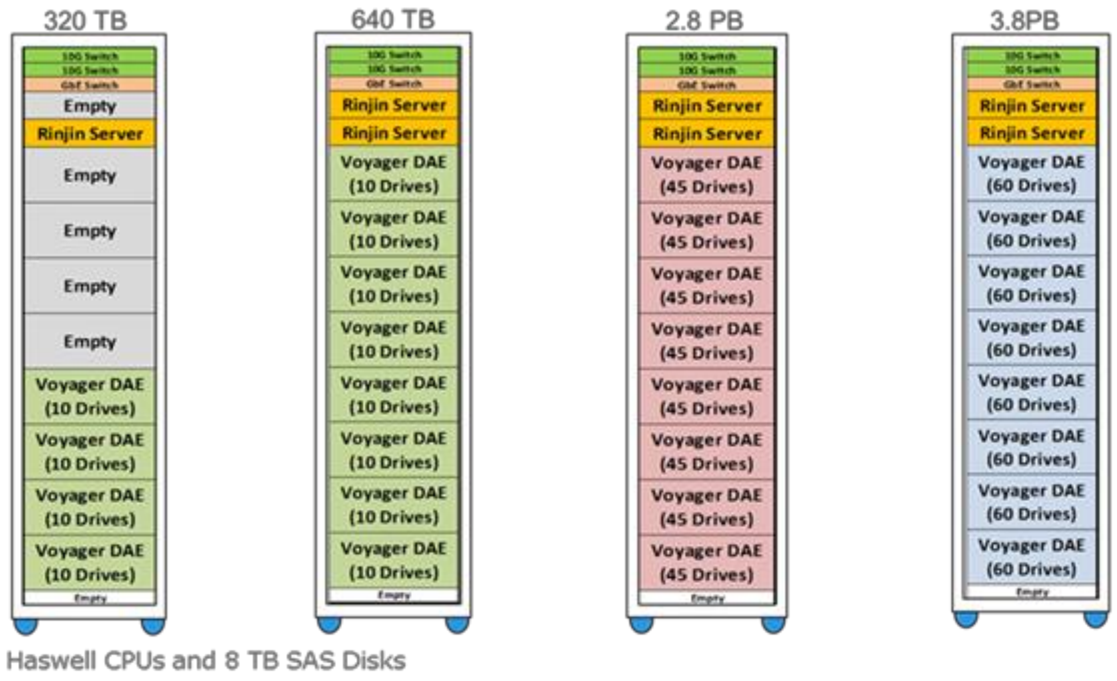


Figure 17 - ECS Appliance Gen-2 Models



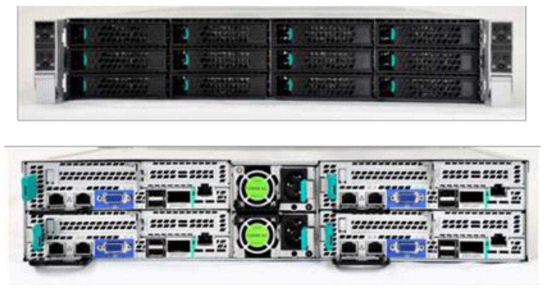
ECS Appliance Server Nodes

ECS comes with 4-blade servers in a 2U chassis. Server specifications are as follows:

- 4 node servers in 2U with two CPUs per node
- 2.4 GHz four-core Ivy Bridge or Haswell CPUs
- 64GB memory per node
- Dual 10Gb Ethernet (NIC-bonding makes them appear as a single port)
- Dual 1Gb Ethernet
- Dedicated 1Gb Ethernet remote diagnostics
- 4 x 2.5" drive slots per node
- Single DOM for OS boot
- Two SAS interfaces per node

Figure 18 shows a front and rear view of the ECS 4 node blade server.

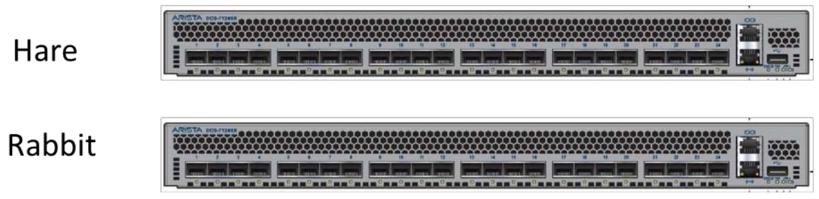
Figure 18 - 4 Node Blade Servers
(Front and Rear View)



10GbE Switches – Data

Two 10GbE, 24-port Arista switches are used for data transfer to and from customer applications as well as for internal node-to-node communications. These switches are connected to the ECS nodes in the same rack. The internal names for these two 10Gb switches are Hare (the top one) and Rabbit (the bottom one). The switches employ the Multi-Chassis Link Aggregation (MLAG) feature, which logically links the switches and enables active-active paths between the nodes and customer applications. This configuration results in higher bandwidth while preserving resiliency and redundancy in the data path. Any networking device supporting static LAG or IEEE 802.3ad LACP can connect to this MLAG switch pair. Finally, because the switches are configured as MLAG, these two switches appear and act as one large switch. Figure 19 displays an example of the front view of these 2 switches.

Figure 19 – An Example of 10GBe Arista Switches (Front View)



1GbE Switch - Management

The 48-port 1Gb Arista switch -- known within EMC as "Turtle" for its slower speed -- is used by ECS for node management and out-of-band management communication between the customer's network and the RMM ports of the individual nodes. The main purpose of this switch is for remote management and console, install manager (PXE booting), and enables rack management and cluster wide management and provisioning. Figure 20 shows a front view of this management switch.

Figure 20 - 1GBE Arista Switch (Front View)

Disk Array Enclosure (DAE)

DAEs are 4U enclosures that can have a maximum of 60 disks. Each DAE is connected to one server in the ECS node by one (Gen-1) or two (Gen-2) SAS connection. Thus, there are four DAEs in a 4-node ECS rack, and eight DAEs in an 8-node ECS rack. Currently, the supported drives are 6TB 7200 RPM SATA drives for Gen-1 and 8TB 7200 RPM SAS drives for Gen-2.

Each DAE includes a hot-swappable LCC. The LCC's main function is to be a SAS expander and provide enclosure services for all drive slots. The LCC independently monitors the environmental status of the entire enclosure and communicates the status to the ECS Element Manager. Note that the LCC is a hot-swappable component, but it cannot be replaced non-disruptively. The DAE has just one LCC, therefore, the disks in the DAE need to be brought offline before the LCC is replaced.

ECS Node Connectivity

There are four servers or ECS nodes in a 2U blade server, and each ECS node has one or two SAS connections to a DAE (depending on the hardware generation used) and four network connections -- one link to each of the two 10GbE switches and two links to the 1GbE switch.

Each ECS node has two 10GbE ports, which appear to the outside world as one port via NIC bonding. Each 10GbE port connects to one port in the Hare switch and one port in the Rabbit switch . These public ports on the ECS nodes get their IP addresses from the customer's network, either statically or via a DHCP server.

The 1GbE management port in an ECS node connects to an appropriate port in the 1GbE switch (Turtle) and has a private address of 192.168.219.X. Each node also has a connection between its RMM port and a port in the 1GbE switch, which in turn has access to a customer's network to provide out-of-band management of the servers. To enable access for the RMM ports to the customer's network, Ports 51 and/or 52 in Turtle are linked to the customer's network either directly or through the 10GbE top-of-the-rack switches. The RMM port is used by EMC field service personnel for monitoring, troubleshooting and installation.

Customer applications connect to ECS by using the (10GbE) public IP addresses of the ECS nodes.

You can expand an ECS rack by daisy-chaining one or more ECS racks to an existing rack. The 1GbE switches in the racks are used for serially linking the racks. The 10GbE switches in each rack will be connected to a customer-provided switch or backplane. Thus the data and chunk traffic will flow through the 10GbE network. For more detailed information on connectivity, please refer to the ECS Hardware and Cabling Guide and other references stated earlier in this section. Figures 21 and 22, which depict the network cabling, as well as the figures in the hardware section also appear in the ECS Hardware and Cabling Guide.

Figure 21 - 10 GbE Network Cabling

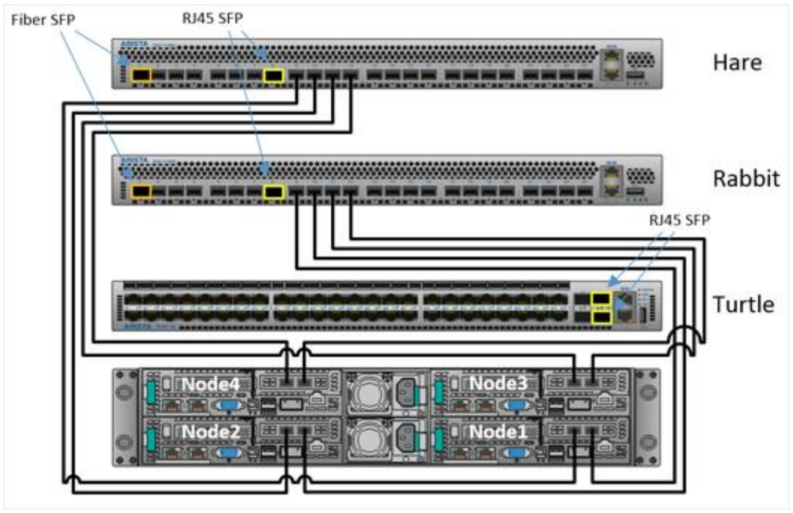
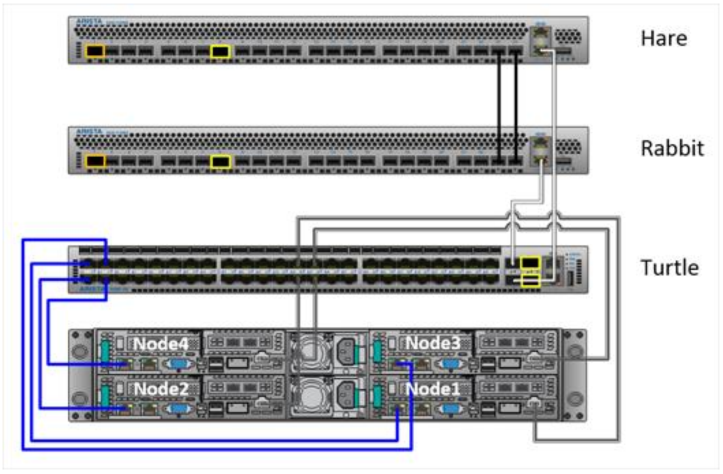


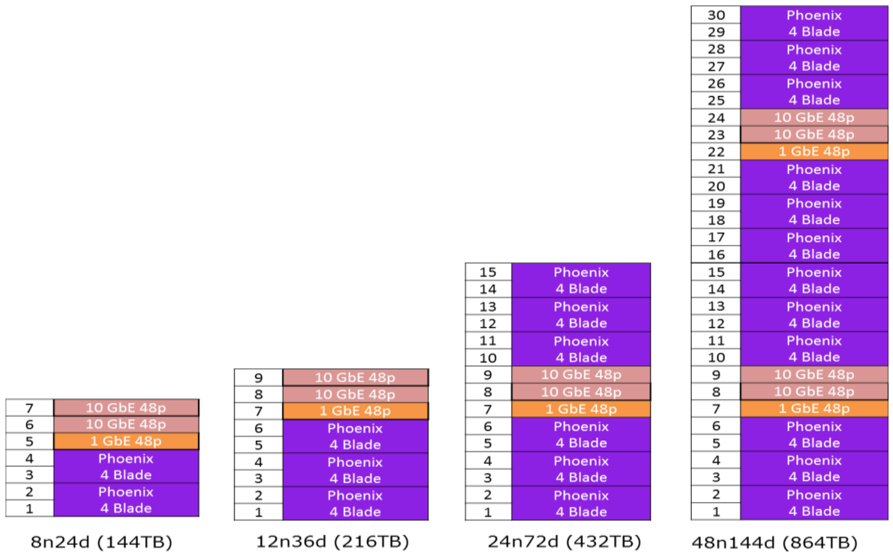
Figure 22 - 1GbE Network Cabling



C- Series

The C-models are basically ECS nodes without the DAEs—the nodes have disk drives that fit in the internal drive slots. Three drives per node are used for data. Fewer drives in these ECS models equates to very high server-to-disk ratios. The drives are 6TB HDD. The C70 models are sold with a minimum of 8 nodes—2 Phoenix blade servers each with 12 integrated disks for a total of 24 disks and a raw capacity of 144TB (24 x 6TB). The C70 model is available with a starter configuration of 144TB. This model may be installed into an existing 3rd party rack. Figure 23 provides some of the C-Series configurations.

Figure 23 – Some of the C-Series Configurations



Upgrading ECS

You can upgrade ECS by adding extra disks to existing DAEs or adding pairs of servers and DAEs (4 servers at a time). Each DAE can store maximum 60 disk drives, but disks are upgraded 10 or 15 at a time depending on model. Please refer to the ECS Hardware and Cabling Guide for more information on the upgrade paths for each series of ECS. Also, check with EMC's sales tools or representative to verify what upgrades are possible for a given configuration.

COMMODITY (DO-IT-YOURSELF)

ECS Software can be installed on EMC-approved 3rd-party commodity hardware. Do-It-Yourself (DIY) configurations require a qualification process and must go through the ASD help desk. EMC-qualified commodity hardware is currently targeted for a select number of customers, with broader support planned in the future.

SUPPORT

Depending on your whether you run ECS on an ECS appliance or DIY, the support models differ. Table 3 specifies support model offered for each type of hardware deployment.

Table 3 - Software and Hardware Support

Type	ECS Appliance	Commodity (EMC Qualified)
Software Package	ECS Software	ECS Software
Node Operating System	Managed By EMC	EMC qualified and approved
Hardware Infrastructure	ECS provided hardware rack	EMC qualified and approved hardware
Network Infrastructure	EMC provides fully managed rack	Customer provided and managed
Support Model	Fully supported by EMC	Joint support between customer and EMC

SECURITY

ECS security is implemented at the administration, transport and data levels. User and administration authentication is achieved via Active Directory, LDAP methods, or within the ECS portal. Data level security is done via HTTPS for data in motion and/or server-side encryption for data at rest.

AUTHENTICATION

ECS supports both Active Directory and LDAP authentication methods to provide access to manage and configure ECS; however, limitations exist as shown in Table 4. For more information on security, see the ECS Security Configuration Guide.

Table 4- Authentication Methods

Authentication Method	Supported
Active Directory	• AD Groups are supported for management users • Supported for Object Users • Multi-domain is supported.
LDAP	• LDAP Groups are not supported for management users • LDAP is supported for Object Users • Multi-domain is supported.

DATA SERVICES AUTHENTICATION

Object access using RESTful APIs is secured over HTTPS via specific ports, depending on the protocol. All incoming requests are authenticated using defined methods such as Hash-based Message Authentication Code (HMAC), Kerberos, or token authentication methods. Table 5 below presents the different methods used for each protocol.

Table 5 - Data Services Authentication

Protocols		Authentication Methods
Object	S3	V2 HMAC-SHA1
	Swift	Token – Keystone v2, SWAuth v1
	Atmos	HMAC-SHA1
	CAS	Secret Key PEA file
HDFS		Kerberos
NFS		Kerberos , AUTH_SYS

DATA AT REST ENCRYPTION

ECS version 2.2 and later supports server-side encryption. Server-side encryption follows the **FIPS-140-2 Level 1 compliance**, AES256 algorithm. Key features of ECS Data-at rest-encryption include:

- Low touch encryption at rest – enabled easily through the ECS Portal
- Namespace- and bucket-level control with transitivity – encryption can occur at namespace or bucket level
- Automated key management
- S3 encryption semantics support – Server Side Encryption (SSE) constructs can be used to allow object encryption, e.g. x-amz-server-side-encryption.

A valid license is required to enable server-side encryption via the ECS Portal or through the ECS REST API. Each namespace, bucket and object has an associated key that is auto-generated at the time of creation. Only data and user-defined metadata will be encrypted inline prior to being stored on disks.

Key Management

Encryption can be enabled at the namespace, bucket and/or object levels. There are two types of keys:

- **User provided keys via S3 Header –** key is provided by user through the header in the S3 API

- **System-generated keys** – randomly generated and hierarchically structured

For S3 users and applications using the S3 REST APIs, object-level encryption can be done by either system-generated or user-specified keys. If the S3 encryption header provides the key, then encryption of object is done using the user-provided key. ECS validates the key provided for updates, appends, and reads is the same as the key used for object creation. If the encryption header is provided but without a key, then the system-generated keys are used to encrypt the data.

System-generated keys at each level are auto-generated and encrypted using the key of its immediate parent. The master key is encrypted by utilizing asymmetric public-private encryption. The public and private keys are used to encrypt and decrypt the master key. Keys are generated and encrypted in a hierarchical order:

- **Public-Private Key pair** – a pair of public-private keys generated for each ECS system. The private key is stored in the system root disk of the node and the public key is stored with the master key on the ECS commodity disks.

- **Master Key** – randomly generated key encrypted using the public key.

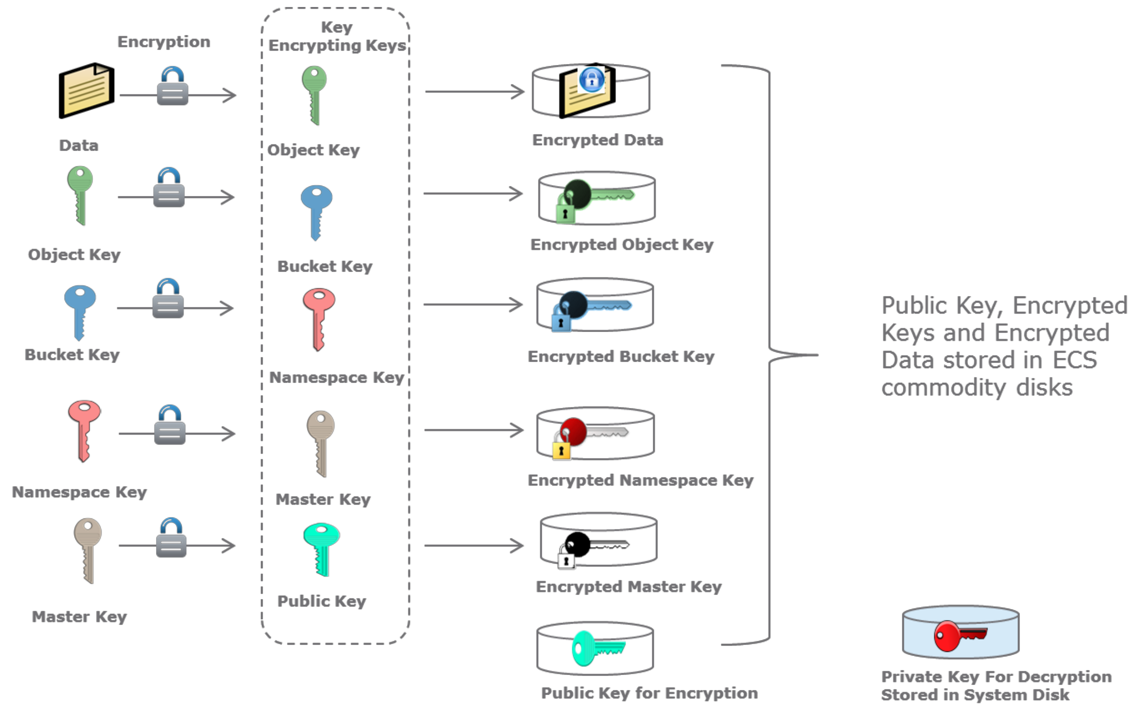
- **Namespace Key** – randomly generated namespace key encrypted using the master key.

- **Bucket Key** – randomly generated bucket key encrypted using the namespace key.

- **Object Key** – randomly generated object key encrypted using the bucket key and object id.

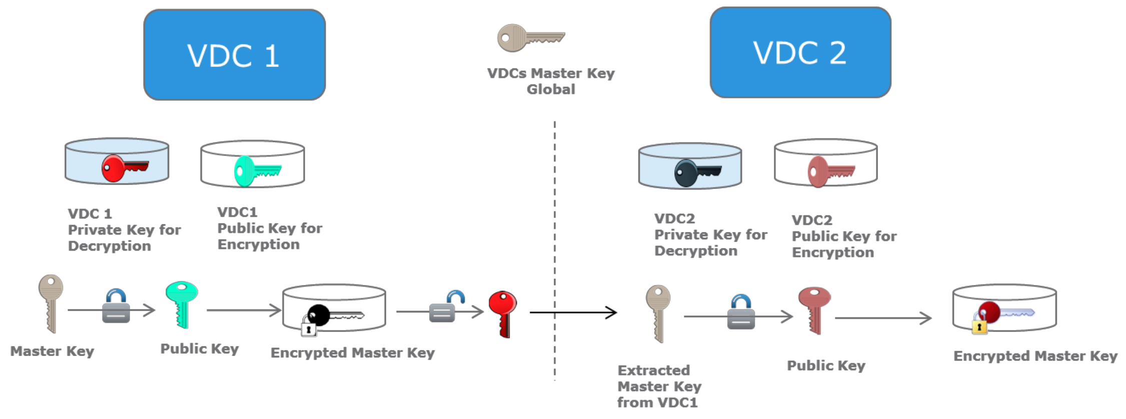
The private key for decryption of master key is stored on system disks (managed by vNest) and the other encrypted keys are stored in logical tables and in chunks, similar to data. They are also triple-mirrored, like data. When an object read or write request comes in, the node servicing the request traverses the key hierarchy to encrypt or decrypt the object. It uses the private key that's common to all nodes to decrypt the master key. Figure 24 provides a pictorial view of the key hierarchy.

Figure 24 - Data Encryption using System Generated Keys



In a geo-replicated environment, when a new ECS system joins an existing system (referred to as a federation), the master key is extracted using the public-private key of the existing system and encrypted using the new public-private key pair generated from the new system that joined the federation. From this point on, the master key is global and known to both systems within the federation. In Figure 25, the ECS system labeled VDC 2 joins the federation and the master key of VDC 1 (the existing system) is extracted and passed to VDC 2 for encryption with the public-private key randomly generated by VDC 2.

Figure 25 - Encryption of Master Key in Geo Replicated Environment.


VDC 1
VDCs Master Key Global
VDC 2
VDC 1 Private Key for Decryption
VDC1 Public Key for Encryption
VDC2 Private Key for Decryption
VDC2 Public Key for Encryption
Master Key
Public Key
Encrypted Master Key
Extracted Master Key from VDC1
Public Key
Encrypted Master Key

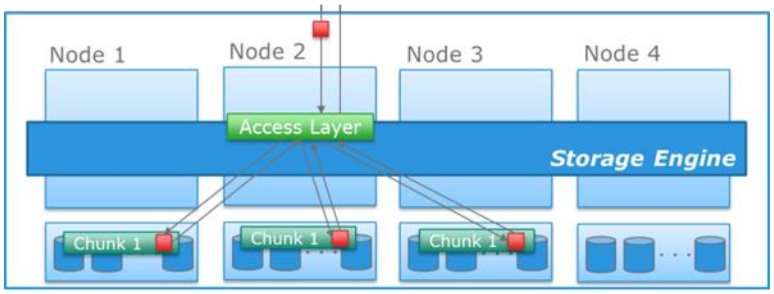
DATA INTEGRITY AND PROTECTION

The most common approach for data protection within storage systems is RAID. ECS, however, does not utilize RAID; it employs a hybrid of triple mirroring of data, meta-data, and index and also erasure coding of data for enhanced data protection and reduction of storage overhead.

TRIPLE-MIRRORED

All types of information relating to objects, such as data, metadata, and index (B+ tree and journal logs) are written to chunks. At ingest, the chunks are triple mirrored to three different nodes within the ECS system as shown in Figure 26. This technique of triple mirroring allows for data protection of the data in case of a node or disk failure.

Figure 26 - Triple Mirroring of Chunks



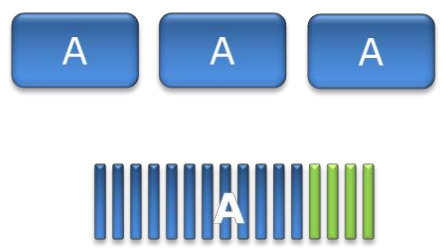
ERASURE CODING

Erasure coding provides enhanced data protection from a disk or node failure in a storage efficient fashion compared with conventional protection schemes. The ECS storage engine implements the Reed Solomon 12+4 erasure-coding scheme, in which a chunk is broken into 12 data fragments and 4 coding (parity) fragments. The resulting 16 fragments are dispersed across nodes at the local site. The data and coding fragments of each chunk are equally distributed across nodes in the cluster. For example, with 8 nodes, each node has 2 fragments (out of 16 total). The storage engine can reconstruct a chunk from any 12 of the 16 fragments. ECS 2.2 and later includes a cold archive option, for which a 10+2 erasure coding scheme is used. In this protection scheme, each chunk is broken into 10 data fragments and 2 coding (parity) fragments, which allows for better storage efficiencies for particular use cases.

All data in ECS is erasure coded except the index and system metadata. The index provides location to objects and chunks and is frequently accessed; hence, it is always kept in triple-mirrored chunks for protection.

ECS requires a minimum of four nodes to be able to conduct erasure coding and six nodes for the cold archive option, in which a 10+2 scheme is used instead of 12+4. Erasure coding stops when one node goes down in a four node cluster. When a chunk is full (128MB) or after a set period of time, it's sealed and erasure-coded.   Erasure coding is conducted as a background process. After erasure coding completes, the mirrored copies are discarded and a single erasure coded copy persists. In the erasure-coded copy, the original chunk data remains as a single copy that consists of 16 data fragments dispersed throughout the cluster, i.e., ECS can read that chunk directly without any decoding or reconstruction. For small objects, ECS doesn't need to read all the fragments of the data -- it can directly read the object from the fragment which contains it. ECS only uses the code fragments for chunk reconstruction when a failure occurs. For objects greater than 128MB in size, erasure coding is done in-line and erasure-coded data is directly written to the disk. This is done to enhance performance and decrease disk usage. Figure 27 gives a view of how a triple mirrored chunk is replaced by an erasure coded chunk.

Figure 27 - Triple Chunks Replaced by Erasure Coding



In ECS 2.2 and later, when additional nodes are added into a configuration to upgrade capacity, ECS re-distributes erasure-coded chunks in the background. This will have minimal impact on system performance; however, there will be an increase in inter-node traffic during rebalancing. Redistribution enhances local protection by leveraging all of the resources within the infrastructure.

CHECKSUMS

Another mechanism ECS uses to ensure data integrity is to store the checksum for data written. Checksums are done per write-unit, up to 2 MB. So, checksums can occur for one object fragment for large object writes or on a per-object basis for small object writes of less than 2MB. During write operations, the checksum is calculated in memory and then written to disk. On reads, data is read along with the checksum, and then the checksum is calculated in memory from the data read and compared with the checksum stored in disk to determine data integrity. Also, the storage engine runs a consistency checker periodically in the background and does checksum verification over the entire data set.

SPACE RECLAMATION

ECS writes data to chunks in an append-only pattern. Thus, during updates and deletes, there will be blocks of data that are no longer referenced and used. When a container chunk is empty, the unused chunk is reclaimed by a background task and the space can then be reused. This process is referred to as garbage collection.

DEPLOYMENT

ECS can be deployed as a single site or in a multi-site configuration. The building blocks of an ECS deployment include:

- **Virtual Data Center (VDC)** – a geographical location defined as a single ECS deployment within a site. Multiple VDCs can be managed as a unit.
- **Storage Pool** – a storage pool can be thought of as a subset of nodes and its associated storage belonging to a VDC. An ECS node can belong to only one storage pool; a storage pool can have any number of nodes, the minimum being four. A storage pool can be used as a tool for physically separating data belonging to different applications.
- **Replication Group** – defines where storage pool content is protected and locations from which data can be read or written. Local replication groups protect objects within the same VDC against disk or node failures. Global replication groups span multiple VDCs and protect objects against disk, node, and site failures.
- **Namespace** - a namespace, which is conceptually the same as a "tenant," is a logical construct. The key characteristic of a namespace is that users from one namespace cannot access objects belonging to another namespace. Namespaces can represent a department within an organization or a group within a department.
- **Buckets** – container for object data. Buckets are created in a namespace to give applications access to data stored within ECS. In S3, these containers are called "buckets" and this term has been adopted by ECS. In Atmos, the equivalent of a bucket is a "subtenant"; in Swift, the equivalent of a bucket is a "container", and for CAS, a bucket is a "CAS pool". Buckets are global resources in ECS. Where the replication group spans multiple sites, a bucket is similarly replicated across sites.

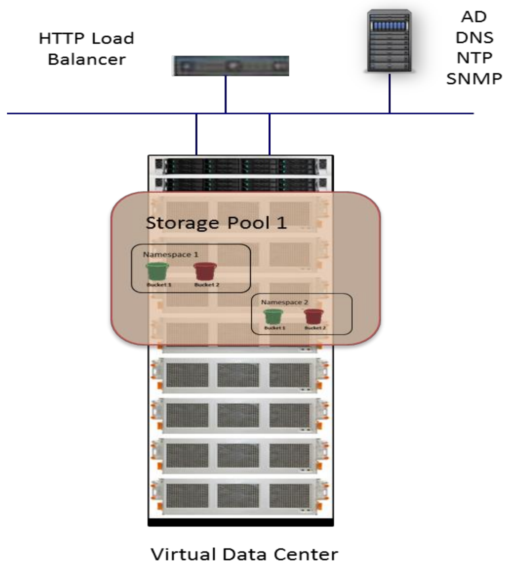
In order to be able to deploy ECS, certain infrastructure requirements need to be reachable by the ECS system.

- **Authentication Providers** – users (system admin, namespace admin and object users) can be authenticated using Active Directory or LDAP
- **DNS Server –** Domain Name server or forwarder
- **NTP Server** – Network Time Protocol server. Please refer to the NTP best practices for guidance on optimum configuration
- **SMTP Server** – (optional) Simple Mail Transfer Protocol Server is used for sending alerts from the ECS rack.
- **DHCP server –** only if assigning IP addresses via DHCP
- **Load Balancer** - (optional but highly recommended) evenly distributes loads across all data services nodes. Load balancers can use simple algorithms such as random choice or round robin. More sophisticated load balancers may take additional factors into account, such as a server's reported load, response times, up/down status, number of active connections, geographic location and so on. The customer is responsible for implementing load balancers; customers have several options including Manual IP allocation, DNS Round Robin, Client-Side Load Balancing, Load Balancer Appliances, and Geographic Load Balancers. The following are brief descriptions of each of those methods:
    - **Manual IP Allocation -** Data node IP addresses are manually distributed to applications. This is not recommended because it does not evenly distribute loads between the nodes and does not provide any fault-tolerance if a node fails.
    - **DNS Round-Robin -** With DNS Round-Robin, a DNS name is created for ECS and includes all of the IP addresses for the data nodes. The DNS server will randomly return the IP addresses when queried and provide some pseudo-load balancing. This generally does not provide fault-tolerance because you would need to remove the IP addresses from DNS to keep them out of rotation. Even after removing them, there is generally some TTL (time-to-live) issues where there is a delay to propagate the removal. Also, some operating systems like Windows will cache DNS lookups and can cause "stickiness," where a client keeps binding to the same IP address, reducing the amount of load distribution to the data nodes.
    - **Physical or Virtual load balancing-** This option is the most common approach to load balancing. In this mode, an appliance (hardware or software) receives the HTTP request and forwards it on to the data nodes. The appliance keeps track of the state of all of the data nodes (up/down, # of connections) and can intelligently distribute load amongst the nodes. Generally, the appliance will proactively "health check" the node (e.g. GET/?ping on the S3 head) to ensure the node is up and available. If the node becomes unavailable it will immediately be removed from rotation until it passes a health check. Another advantage to this kind of load balancing is SSL termination. You can install the SSL certificate on the load balancer and have the load balancer handle the SSL negotiation. The connection between the load balancer and the data node is then unencrypted. This reduces the load on the data nodes because they do not have to handle the CPU-intensive task of SSL negotiation.
    - **Geographic load balancing -** Geographic load balancing takes Physical or Virtual Load Balancing one step further: it adds load balancing into the DNS infrastructure. When the DNS lookups occur, they are routed via an "NS" record in DNS to delegate the lookups to a load balancing appliance like the Riverbed SteelApp. The load balancer can then use Geo-IP or some other mechanism to determine which site to route the client to. If a site is detected to be down, the site can be removed quickly from DNS and traffic will be routed to surviving sites.

SINGLE-SITE DEPLOYMENT

In a single site, storage pools are defined, then the Virtual Data Center (VDC) is created with namespaces and buckets. Figure 28shows one storage pool in a VDC with two namespaces each containing two buckets.

Figure 28 –Single Site Deployment Example



MULTI-SITE DEPLOYMENT

In a multisite deployment, more than one VDC is managed as a federation and/or geo-replicated. In a geo-replicated deployment, data can be read or written from any site within the defined replication group. This section describes this type of deployment.

GEO-FEDERATION

In essence, geo-federation means managing a geographically distributed environment as a single logical resource. Inside ECS, the term refers to the ability to federate multiple sites or VDCs. The obvious benefits are ease of management and the ability to use resources from multiple data centers.

To manage or monitor a single site, the administrator simply logs into one of the nodes in the VDC. This is equivalent to having a single-site federation. The administrator can subsequently add other VDCs to the federation by navigating to the VDC in the ECS portal at the first VDC. Further, customers can choose what data is replicated across sites for enhanced data protection.

To perform some administrative tasks such as creating storage pools or viewing performance statistics, one has to log into each VDC individually.

GEO-REPLICATION

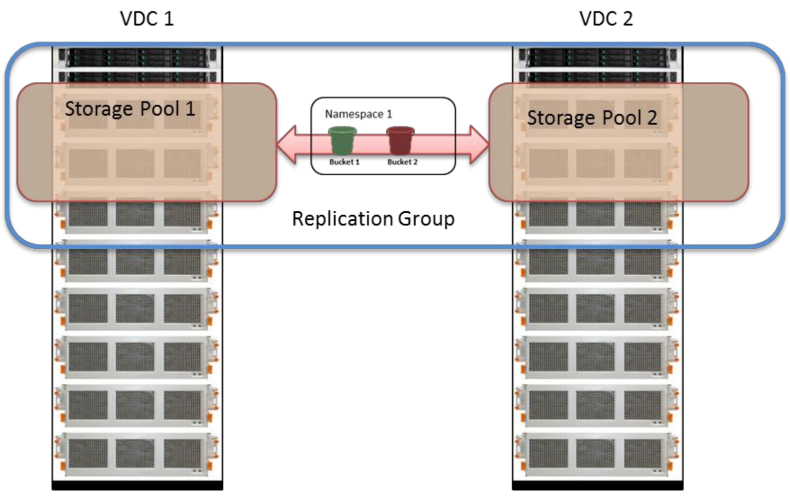
As discussed earlier, ECS offers erasure coding to help provide enhanced data durability without the overhead of storing multiple copies of the data. However, this does not protect against site failures/outages. Geo-replication provides enhanced protection against site failures by having multiple copies of the data, i.e., a primary copy of the data at the original site and a secondary copy of the data at a remote site/VDC. Both the primary and replicated copies of data at other sites are individually protected via erasure coding or triple-mirrored chunks. This means each copy has protection from local failures, such as disk or node failure. Replication to the other site is an asynchronous process. Data is first encrypted (AES256) and then sent to other site via HTTP.

Geo-replication ensures that data is protected against site failures/disasters. Unlike other solutions in the marketplace, ECS does not generate WAN traffic while recovering from local disk failures. ECS gives customers the option to link geographically dispersed systems and bi-directionally replicate data among these sites across the WAN. Several strategies, such as geo-caching as well as accessing the physically closest array, reduce WAN traffic for data access.

A replication group (RG) is defined in a geo-replicated deployment and acts as the logical container that defines what namespace maps to what physical resources, i.e. storage pool(s), to store data for the namespace. A single-site RG defines the storage pool where the data mapped to that RG resides. A multi-site RG defines a set of storage pools within which the primary and the secondary copies of data reside. The primary data copy resides at the site the write came into and the other storage pools jointly store a single secondary data copy.

Figure 29 shows an overview of the mapping from logical model to physical infrastructure, or storage pool. Customer applications access a bucket that belongs to a namespace. One or more namespaces are defined within a replication group that is a logical grouping of storage pools from different VDCs.

Figure 29 - Geo Replicated Environment Example



ACTIVE-ACTIVE ACCESS

One key feature of ECS is the ability to read and write data from any VDC within a replication group. In ECS, data is replicated asynchronously to multiple VDCs within a replication group. The challenge this poses is consistency of data across sites or VDCs. ECS ensures strong consistency by fetching the metadata from the VDC that created the object data. Thus, if an object is created in site 1 and is read from site 2, ECS checks with site 1, who is the object owner, and validates that the copy replicated at site 2 is the latest version of the data. If not, it fetches the data from site 1; otherwise, it uses the data from site 2.

GEO-CACHING

Customers with multi-site access patterns, specifically a replication group composed of three or more sites, could experience slow performance if data is always fetched from primary site, i.e., where the data was originally written. Consider a geo-federated environment with Sites 1, 2 and 3. An object, Object A, is written to Site 1 and the secondary copy of the object resides at Site 2. In this scenario, a read for the object at Site 3 needs to fetch the object data from either Site 1 or Site 2 to honor the read. This leads to elongated response times, especially in environments with multi-site access patterns. ECS alleviates the response time impact by using some pre-designated percentage of disk space to cache objects that do not exist locally. For frequently accessed objects you would see a reduced response time after the initial copy of the object is cached locally. While the data replication is asynchronous, the metadata replication is synchronous and ensures that the system never responds to a read at a remote site with a stale version of the data, thereby being true to the tenant of strong consistency. The cache is an LRU implementation and cache size is adjusted when nodes/disks are added to the storage pool.

TEMPORARY SITE OUTAGE (ACCESS DURING OUTAGE)

Temporary site failure refers to either a failure of the WAN connection between two sites or a failure of an entire site (such as a natural disaster or power failure). ECS can detect and automatically handle any such temporary site failures. VDCs in a geo-replicated environment establish a heartbeat mechanism. Sustained loss of heartbeats for a preset duration is indicative of a network outage and the system adjusts its behavior accordingly. Without the TSO (Temporary Site Outage) functionality, the inability to communicate with the node that owns the object (from a metadata perspective) leads to the system rejecting the read and write requests for the object. This TSO feature aligns with the principle of strong consistency that the ECS system adheres to.

With "Access During Outage" enabled on a bucket and upon detecting a temporary outage, the system reverts to an eventual consistency model, i.e., reads/writes from a secondary (non-owner) site are accepted and honored. Further, a write to a secondary site during a network outage causes the secondary site to take ownership of the object. This allows each VDC to continue to read and write objects from buckets in a shared namespace. Finally, the new version of the object becomes the authoritative version of the object during reconciliation even if another application updates the object on the "owner" VDC.

Although many object operations continue during a network outage, certain operations are not be permitted, such as creating new buckets, namespaces, or users.

When network connectivity between two VDCs is restored, the heartbeat mechanism automatically detects connectivity, restores service and reconciles objects from the two VDCs. If the same object is updated on both VDC A and VDC B, the copy on the "non-owner" VDC is the authoritative copy. So, if an object that is owned by VDC B is updated on both VDC A and VDC B during synchronization, the copy on VDC A will be the authoritative copy that is kept, and the other copy will be overwritten. Table 6 and 7 capture the list of operations permitted in different system states for 2 and 3 sites.

Table 6 - Temporary Site Outage (Access During Outage for 2 Sites)

System State	**Supported Operations (Site 1 | Site 2 | Site 3)**												**Consistency**
	Read Object, List Objects in Bucket			Create/Update Object			List Buckets in namespace			Create/Edit Bucket Create Namespace Create User			
	S1	S2	S3	S1	S2	S3	S1	S2	S3	S1	S2	S3	
Normal All zones connected	√	√	√	√	√	√	√	√	√	√	√	√	Strong
Temp Site Failover System detects temp site outage and completes temp site failover	√	√ AO	√ AO	x	√	√	√ LB	√ LB √ S3	√ LB √ S2	x	x	x	Eventual Consistency
Site Rejoin Resume normal operation after resync; Objects/Buckets updated at both sites are reconciled with the state of non-owner as the final version	√	√	√	√	√	√	√	√	√	√	√	√	Strong

√ AO: **Acquires Object Ownership** √ LB: **Locally Owned Buckets**

Table 7 - Temporary Site Outage (Access During Outage for 3 Sites: Site 2 and Site 3 takes ownership)

System State	**Supported Operations (Site 1 | Site 2 | Site 3)**												**Consistency**
	Read Object, List Objects in Bucket			Create/Update Object			List Buckets in namespace			Create/Edit Bucket Create Namespace Create User			
	S1	S2	S3	S1	S2	S3	S1	S2	S3	S1	S2	S3	
Normal All zones connected	√	√	√	√	√	√	√	√	√	√	√	√	Strong
Temp Site Failover System detects temp site outage and completes temp site failover	√	√ AO	√ AO	x	√	√	√ LB	√ LB √ S3	√ LB √ S2	x	x	x	Concurrent writes transition system to eventual consistency
Site Rejoin Resume normal operation after resync; Objects/Buckets updated at both sites are reconciled with the state of non-owner as the final version	√	√	√	√	√	√	√	√	√	√	√	√	Strong

√ AO: **Acquires Object Ownership** √ LB: **Locally Owned Buckets**

When more than two VDCs are part of a replication group, and if network connectivity is interrupted between one VDC and the other two, then write/update/ownership operations continue just as they would with two VDCS; however, the process for responding to read requests is more complex, as described below.

If an application requests an object that is owned by a VDC that is not reachable, ECS sends the request to the VDC with the secondary copy of the object. However, the secondary site copy might have been subject to a data contraction operation, which is an XOR between two different data sets that produces a new data set. Therefore, the secondary site VDC must first retrieve the chunks of the object included in the original XOR operation and it must XOR those chunks with the "recovery" copy. This operation will return the contents of the chunk originally stored on the failed VDC. The chunks from the recovered object can then be reassembled and returned. When the chunks are reconstructed, they are also cached so that the VDC can respond more quickly to subsequent requests. Note reconstruction is time consuming. More VDCs in a replication group imply more chunks that must be retrieved from other VDC's, and hence reconstructing the object takes longer.

If a disaster occurs, an entire VDC can become unrecoverable. ECS treats the unrecoverable VDC as a temporary site failure. If the failure is permanent, the system administrator must permanently failover the VDC from the federation to initiate fail over processing, which initiates resynchronization and re-protection of the objects stored on the failed VDC. The recovery tasks run as a background process. You can review the recovery process in the ECS Portal.

FAILURE TOLERANCE

In a single site or geo configuration, there are several types of failures that can be tolerated before ECS is no longer accessible or able to serve data. Table 8 describes the tolerances for each deployment type.

Table 8 – Failure Tolerance

Failure Model	Tolerance
Single Site Failure	In a 4 Node: • Loss of 1 node: Erasure coding stops • Loss of 2 nodes: Writing stops/subset of reads will fail • Loss of 3 nodes: Writing stops/subset of reads will fail In an 8 Node: • Loss of 2 nodes: Subset of reads may fail • Loss of 3 nodes: Subset of reads will fail • Loss of 5 nodes: Erasure coding stops • Loss of 6 nodes: Writing stops.
Geo-Replicated Failure	1 Site Failure
Temporary Site Failure	When filesystem is enabled, read-only access to files/HDFS and any other object protocols.

STORAGE EFFICIENCY

ECS uses erasure coding for data protection. Although this is more storage efficient than other forms of protection, such as mirroring, it does incur some storage overhead. ECS provides a mechanism in which storage efficiency increases as three or more sites are used. In a geo-replicated setup with multiple sites/ VDCs, ECS replicates chunks from the primary VDC to a remote site in order to provide high availability. However, this simple replication can lead to a large overhead of disk space. To prevent this, ECS uses a smart technique to reduce overhead while preserving high availability features. This can be illustrated with a simple example. Consider 3 VDC's in a multi-site environment - VDC1, VDC2 and VDC3, and that VDC1 has chunk C1 and VDC2 has chunk C2. With simple replication, a secondary copy of C1 and a secondary copy of C2 may be placed in VDC3. Since all chunks are of the same size, this will result in a total of 4 x 128MB of space being used to store 2 x 128MB of objects.

In this situation ECS can perform an XOR operation of C1 and C2 (mathematically, written as C1 ⊕ C2) and place it in VDC3 and get rid of individual secondary copies of C1 and C2. Thus, rather than using 2 x 128MB of space in VDC3, ECS now uses only 128MB (the XOR operation results in a new chunk of the same size).

In this case, if VDC1 goes down, ECS can reconstruct C1 by using C2 from VDC2 and the (C1 ⊕ C2) data from VDC3. Similarly, if VDC2 goes down, ECS can reconstruct C2 by using C1 from VDC1 and the (C1 ⊕ C2) data from VDC3.

Counterintuitively, as the number of linked sites increase, the ECS algorithm is more efficient in reducing the overhead. Table 8 provides information on the storage overhead based on the number of sites for normal erasure coding of 12+4 and cold archive erasure coding of 10+2, and it illustrates how ECS becomes more storage efficient as more sites are linked. To obtain the lower overhead, the same amount of data must be written at each site.

Table 8 - Storage Overhead

Number of Sites in Replication Group	Default Use Case (Erasure Code: 12+4)	Cold Archive Use Case (Erasure Code: 10+2)
1	1.33	1.2
2	2.67	2.4
3	2.00	1.8
4	1.77	1.6
5	1.67	1.5
6	1.60	1.44
7	1.55	1.40
8	1.52	1.37

In some scenarios, replication may be desired on all sites for increased data protection and enhanced read performance. Enabling this feature would disable the XOR capability for storage efficiency just described. Replication in all sites is available in ECS 2.2 and later.

CONNECTORS AND GATEWAYS

Several third-party software products have the capability to access ECS object storage. Independent Software Vendors (ISVs) such as Panzura, Ctera, and Syncplicity create a layer of services that offer client access to ECS object storage via traditional protocols, such as CIFS, NFS, and/or iSCSI. For more information on ISV products and use cases, see ECS Appliance Ecosystem Partners. You can also access ECS storage with EMC products: EMC CloudArray offers iSCSI or NAS services and can be used as a front-end to backup and archive data to ECS object storage; Isilon Cloudpools provides smart tiering of data to ECS object storage.

CONCLUSION

Private and hybrid clouds greatly interest customers, who are facing ever-increasing amounts of data and storage costs, particularly in the public cloud space. ECS's scale-out and geo-distributed architecture delivers an on-premise cloud platform that scales to exabytes of data with a TCO (Total Cost of Ownership) that's significantly less than public cloud storage. ECS is a great solution because of its versatility, hyper-scalability, powerful features, and use of low-cost commodity hardware.

REFERENCES

There are links referred within this document. This section collates all these links in one area for reference.

- APIs and SDKs
  - Data Access Guides - ECS Rest APIs
  - EMC Data Services SDK – Atmos and S3 SDKs
- ECS Community
- Object Browser Downloads
  - S3 Browser
  - Cyberduck
- ECS Test Drive

- EMC ECS 2.x Product Documentation
  - Plan an Installation
  - ECS Hardware and Cabling Guide
  - SolVe Desktop (Procedure Generator)
  - ECS System Administrator’s Guide
  - ECS Security Configuration Guide

- ECS Appliance Ecosystem Partners