EMC ATMOS CLOUD STORAGE ARCHITECTURE

ABSTRACT

This white paper explains a high level architecture of the EMC Atmos cloud storage platform to efficiently manage, protect, and store distributed unstructured content at scale.

September, 2014

REDEFINE

EMC²

TABLE OF CONTENTS

INTRODUCTION

Content-rich applications such as those that store medical images, video, collaboration content, and advanced analytics, combined with the growth of user-generated content are rapidly challenging the traditional approach to storing data at scale. In typical Enterprise environments these workloads can combine to routinely create hundreds of billions of new objects in a form that is both unstructured and diverse as it can encompass documents, digital media assets and related materials.

This combination of massive growth, new information types, new deployment styles, and the need to serve multiple locations and users around the world, has led to new market requirements for information management at a global scale. These new storage dynamics have routinely proven to be a challenge when supported with traditional file systems which have relied on two confining concepts.

First is a cyclical pattern of requests for storage resources, followed by a provisioning cycle ranging in duration from minutes to months, and finally a consumption of the amount provisioned. This cyclical pattern can impede operational efficiency in the face of unpredictable and often explosive content growth patterns.

Next, in this provisioning cycle, storage is often presented to a consumer which can vary in form from an application to an end user but despite their differences, each consumer traditionally accesses storage through a rigid, structured, path-based construct of \\servername\sharename for example. As applications continue to scale and the underlying storage supporting those apps needs to scale, its growth in a single site must follow this cyclical step-wise storage provisioning process. Then, as apps and their content need to grow and scale geographically, their content must be replicated through a variety of methods each of which typically involve some added set of replication and distribution tasks. These storage-based constraints are often overshadowed by increasing pressures on IT departments to contain costs, eliminate management and storage silos to optimize a shared infrastructure and, ultimately, deliver storage resources in a service-based environment to add business value.

To overcome these constraints, EMC has introduced a new class of storage technology referred to as cloud-optimized. In this new class, the Atmos product family is positioned to deliver a more modern technology than traditional file or block based constructs that delivers the linear scale, elasticity, geographic distribution of unstructured that eliminates the cyclical need to provision and consume storage by making capacity available based largely on-demand as well as accessing an entire infrastructure collectively rather than in a frame-by-frame monolithic approach common with file or block based storage technologies.

Next, Atmos solves the constraints of rigid, mount-point based interaction between storage and consumer by presenting a singular accesspoint to the entire storage infrastructure. This enables storage to be automatically distributed across the infrastructure in an active/active everywhere methodology.

Next, to streamline the delivery of storage as a service on a shared infrastructure, Atmos and the Atmos Cloud Delivery Platform leverage a built-in multi-tenancy model where all users and consumers of storage can be logically isolated from one another without any additional software or complete silos of infrastructure to provide isolation as has been the case with legacy storage technologies.

Finally, as an optional module, the Atmos Cloud Delivery Platform delivers all the components and services needed to deliver turn-key, self-service, fully metered access to storage resources thereby easily enabling the delivery of storage as a service.

NEXT GENERATION STORAGE WITH EMC ATMOS FEATURE OVERVIEW

EMC Atmos is a cloud storage platform that enables enterprises and service providers to store, manage, and protect globally distributed, unstructured content at scale. It is the first exabyte-scale, global information management solution specifically designed to automate and manage data placement, protection, and access for rich, unstructured content as a single system across distributed storage environments.

Atmos operates as a single entity, regardless of how it is physically distributed which distributes content in an active/active paradigm rather than in a hierarchical approach common with file system-based structures. Unlike other systems, Atmos uses customizable, value-driven metadata to drive storage placement, protection and lifecycle policies.   This ensures information get's to the right location, at the right time - automatically.   Atmos can operate as the foundation of a Cloud infrastructure, natively serving and metering isolated tenants (Multi-tenancy) from a single system to maximize utilization across multiple customers and applications.

These qualities of Cloud-optimized storage architecture increase operational efficiency, reduce management complexity, and reduce lifecycle cost. Specific Atmos features that drive these benefits include:

- Massively scalable infrastructure into multiple petabytes with support for billions of objects across a globally distributed infrastructure.

- Unified namespace eliminates capacity, file number, location and other file system limitations.

- Policy-based management: Metadata and policy-based information management capabilities combine to intelligently drive information placement, protection and other information services, optimizing availability and cost based on the customer's SLO.

- Data Protection and Recovery: Atmos offers two flexible policy-based options to choose from. GeoMirror provides traditional synchronous or asynchronous copies that are distributed across locations. GeoParity lets you split up objects into multiple encoded fragments that are distributed across components for increased content durability.

- Integrated Data Services: Atmos policies also allow you to set and automate data services including compression, de-duplication, spin down, striping. Reduce administration time and permit Atmos to be efficiently managed globally.

- Multi-tenancy: Enables multiple applications to be securely served from the same infrastructure. Each application is securely partitioned and data is neither co-mingled nor accessible by other tenants. This feature is ideal for businesses providing cloud services for multiple customers or departments within large enterprises.

- Flexible Access Methods: REST and SOAP web service APIs, as well as file-based access provides convenient integration to virtually any application, and easy access over the LAN or WAN. Sync & Share with mobile devices, windows, and Linux.

- Storage-as-a-Service: The Atmos Cloud Delivery Platform is add-on software product that enables enterprises and service providers to deliver and manage storage-as-a-service to an Atmos cloud. Enables self-service access and management by tenant.

HIGH-LEVEL ARCHITECTURE

The above features can be abstracted into a basic architecture illustrated below. In this context, you have quite simply client requests coming in through a firewall to a front-end node. This node can be either physical or virtual. The Atmos distributed services runs on each node, with each node connected to commodity or blob storage.
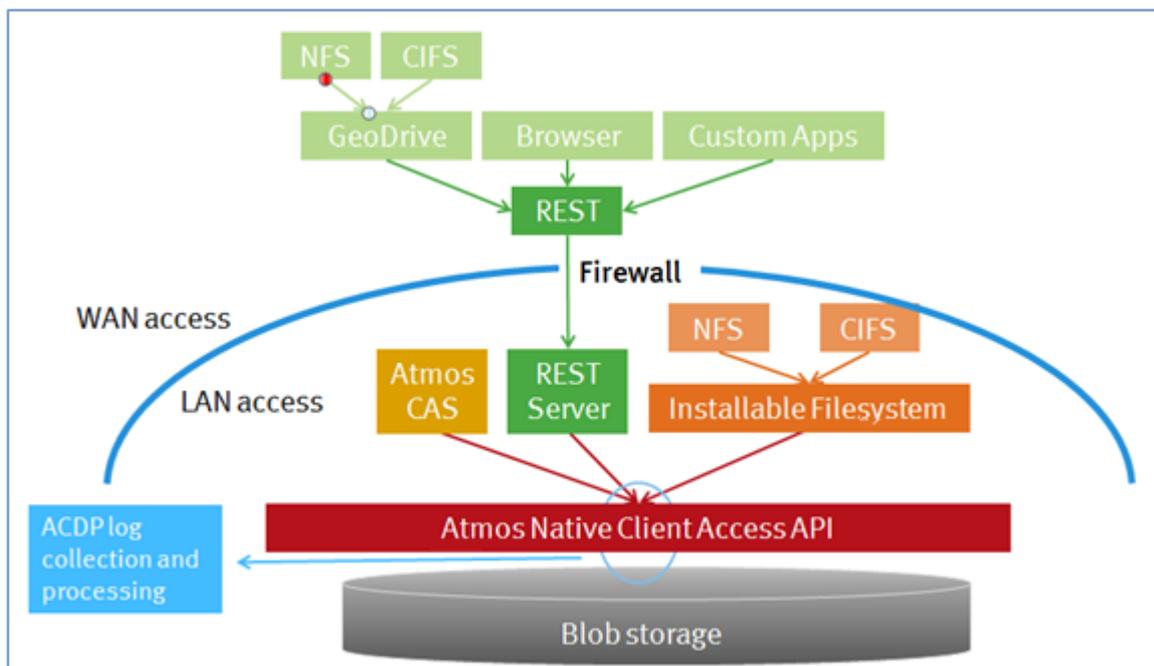

Figure 1. EMC Atmos high level architecture

SERVICES

At its core, Atmos is delivered as a set of distributed, redundant software services that interact with one another to provide global information management. This collection of services:

- Provides web services and file presentation interfaces
- Tracks availability and location of all other services
- Maintains an index of objects
- Stores the policy for objects
- Stores the user and system metadata
- Responds to all I/O requests
- Writes to physical disk
- Manages background replication tasks

The Atmos high-level services architecture can be illustrated as follows:
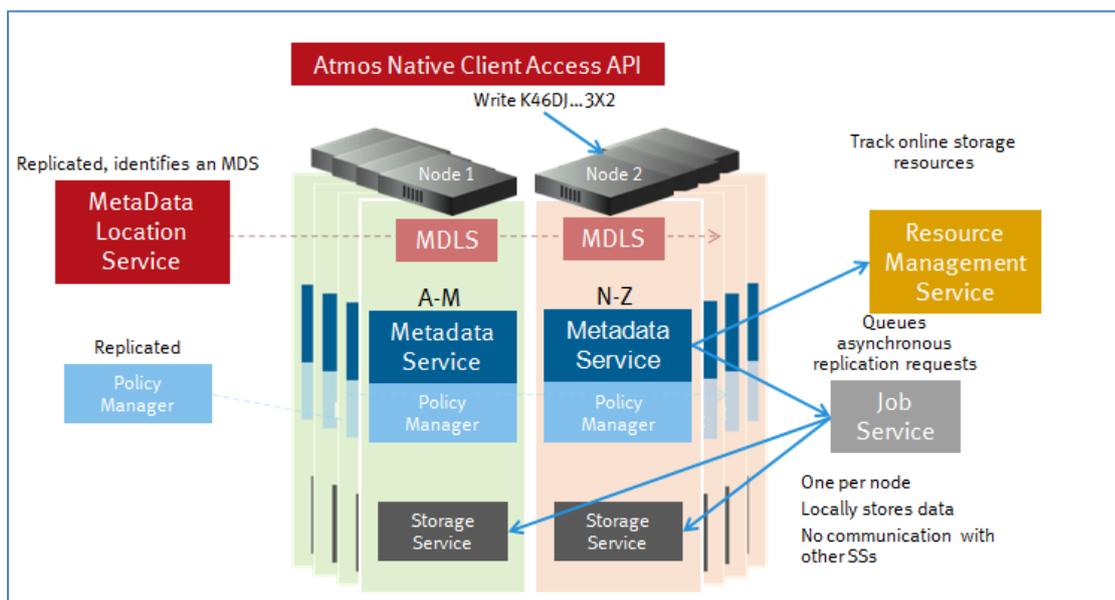
Figure 2. EMC Atmos high level services architecture

SERVICE ARCHITECTURE

Atmos leverages "Staged Event Driven Architecture (SEDA)" to drive its processing. All the services are designed using this framework.

SEDA is an approach to building scalable services that attempts to combine the best aspects of event-driven processing with multi-threaded processing.

Event-driven basics

In a traditional event-driven (not SEDA) server, client requests are modeled as events which are placed on a number of queues. Processing proceeds in a single thread that removes an event from a queue and performs processing appropriate to that particular event type. Each event carries with it some amount of state. If the processing state reaches a blocking condition (e.g. waiting for a device or I/O), it sends the request to the device in an asynchronous mode, updates the state in the event, and stores the event on a queue for processing later. The process continues by selecting an event from the next queue to begin processing.

7

Unlike threaded programming, event-driven programs need no synchronization surrounding critical sections. Events are processed serially, so there is by definition no parallelism within a critical section.

The main advantage of event-driven processing over a multi-threaded approach is that event-driven processing tends to maximize request throughput by reducing processor overhead due to thread synchronization (locks, context switching, etc.) The main disadvantage of event-driven processing vs. a multi-threaded approach is that an event-driven program has no way to take advantage of the extra processing power provided by multi-core architectures.

Staged event-driven basics

The goal of the staged event-driven approach is to combine the modularity and low synchronization overhead of an event-driven style of programming, with the parallelism provided by multi-threaded programming. In a SEDA approach, the programming task for a request is broken down into logically separable stages. Very simply, a stage is defined by a queue of incoming events and one (or more) thread(s) which process each event in turn. In the Atmos SEDA model, each stage can have its own private pool of threads, or can share a pool of threads with other stages. Each stage has a defined service it performs for the event, after which the event can be discarded, or placed in the queue of another stage for additional processing.

DATA AND METADATA

Atmos stores content as objects, and divides objects into two parts: metadata and user data.

- **Metadata**, which is further divided into:
- System metadata – This includes filename, file size, modification date, creation date, access-control lists, and object ID ("OID")
- User metadata – This comprises arbitrary, custom, name-value pairs. Examples of user metadata are artist name (for music data) and customer type
- **User data** – This is application data, such as image files, text files, videos and audio files.

Every object in Atmos has information associated with it that includes an object ID, system and user metadata, Atmos object layout information, and parent/child information (for objects saved through file system interfaces).

Atmos uses metadata to provide greater context for the user data. User metadata can be used to logically group objects. Data management policies can then be applied to these logical groupings. System metadata can also be used to trigger policies based on MIME type and similar system attributes, but user metadata allows the end user and end user application greater control in grouping objects by more abstract concepts like user type (e.g. objects associated with a new user to a web application).

Data flow

Atmos stores content as objects and is designed to handle billions of objects across a logical storage cloud, independent of physical location. The way Atmos creates and reads objects neatly illustrates the SEDA design points and the service interaction.

ATMOS DATA FLOW

Atmos create request

Object creation starts with a query for resources, allowing End User Applications to traverse both logical services and physical resources, eliminating traditional file system and node limitations. All requests are broken in stages and processed, so that at peak load an optimal mapping of requests threads and resources is achieved.
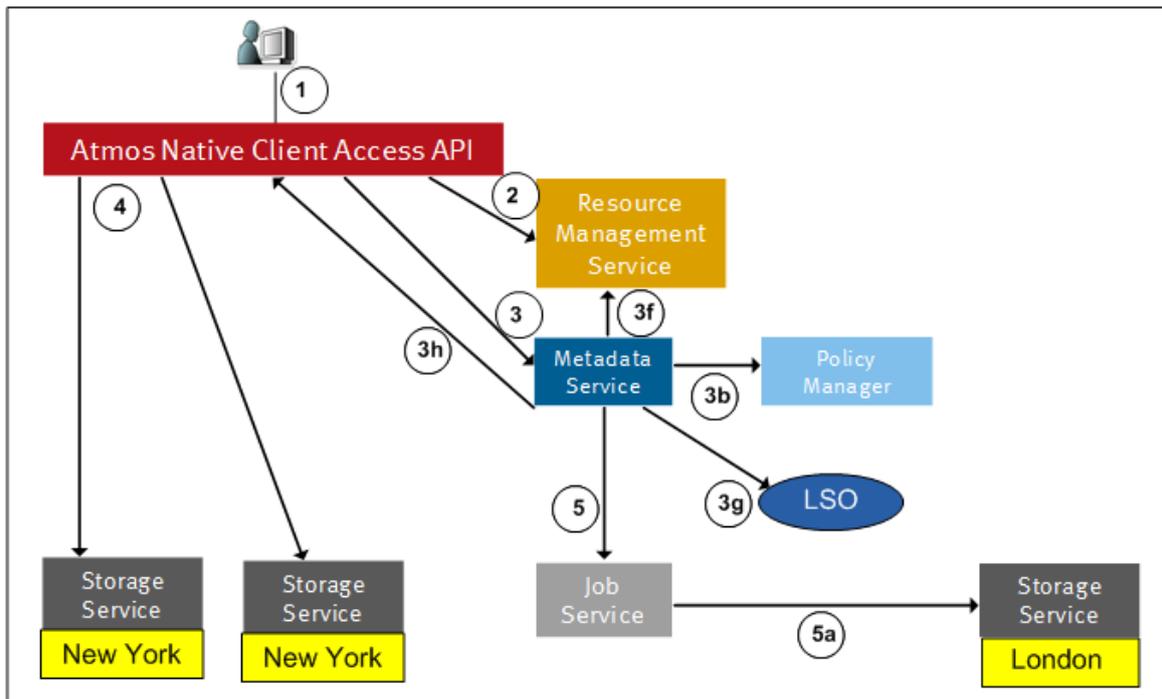
Figure 3. EMC Atmos create request workflow

1.	Client Access Layer (CL) gets a Create request, and invokes Create Event

2.	CL Goes to RMS, and asks for a local MDS

	a.	RMS returns MDS back to CS

3.	CL sends a request to the local MDS master service for Create Object request.

	a.	MDS generates a unique object id.

	b.	MDS consults PM and generates the layout of the object.

	c.	PM goes through assigned Policies for that sub-tenant, and return a policy layout for the object in XML format to MDS (E.G. – 2 local sync replicas, and 1 remote async replica, based on system metadata trigger)

	d.	MDS fills in the system metadata.

	e.	MDS fills in any user metadata coming in through the request.

	f.	MDS takes layout and goes to RMS for available Storage Servers to satisfy layout

	g.	MDS generates LSO (Logical Storage Object), with assigned Storage Servers. These are the exact nodes and storage disks where the replicas will be written to

	h.	MDS returns the entire structure back to the CL

4.	CL then sends data objects to the SS for any synchronous writes in parallel.

	a.	SS creates a backend data object for the data and returns back the OSD ID.

	b.	CL fills out the PSO (Physical Storage Object) ID information associated with each SS.

	c.	CL sends an update to the MDS

5.	If any asynchronous replicas are associated with the object, MDS generates asynchronous copy jobs and forwards it to the JS.

	a.	The JS sends the Asynchronous replica to the designated SS at the remote site

Atmos read request

Atmos read requests were designed so that any read request scales throughout an Atmos system to satisfy a client read request. This is accomplished with Atmos' distributed services architecture, described below.
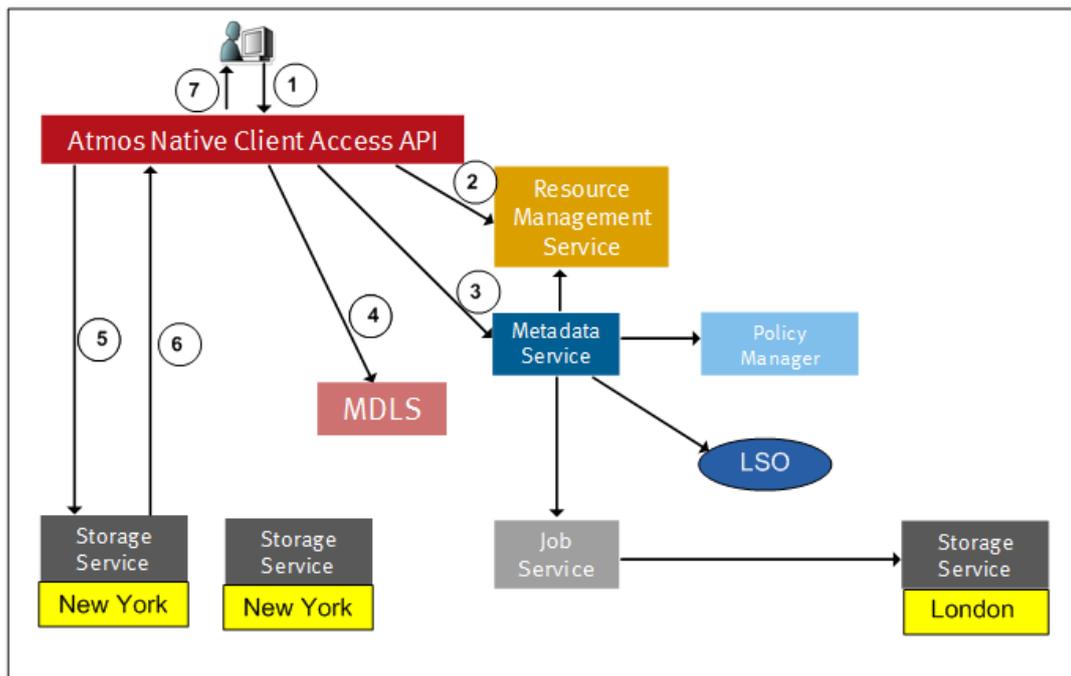

Figure 4. EMC Atmos read request workflow

1. Client Access Layer (CL) gets a Read request, and invokes Read Event

2. Client on reception of a read request, using a given Object ID, consults with RMS for a local MDLS.

3. Client then consults with MDLS and determines the owner MDS replication set (Primary, Slave, and Remote MDS services for the specific object).

4. Client talks to the primary, secondary, or remote MDS to retrieve the object metadata and also opens the object for read.

a. Client gets back the object layout.

5. Client interprets the object layout and determines the closest SS and issues a read request to the SS by sending the PSO ID associated with the SS.

a. SS interprets the OSDID and determines the path on the backend ext3 file system.

6. SS Reads the data content and returns back the data to the Client

7. Client in turn returns the data back to the client.

CREATING A SYSTEM WITH THE ATMOS BUILDING BLOCKS

As illustrated above, Atmos is a set of distributed software services which can be deployed on three bundled physical solutions based on commodity hardware, or on VMware virtual machines.

The Atmos packaging consists of two elements: Atmos front-end nodes and Disk Array Enclosures (DAE). As shown in the picture below, Atmos front-end nodes run the Atmos software, while the Atmos DAEs provide very dense, economical storage. Every node runs the Common Appliance Platform "CAP", which is an internal EMC Linux distribution based on a Red Hat kernel. The Atmos software is layered on top of CAP, and is considered a closed appliance model.
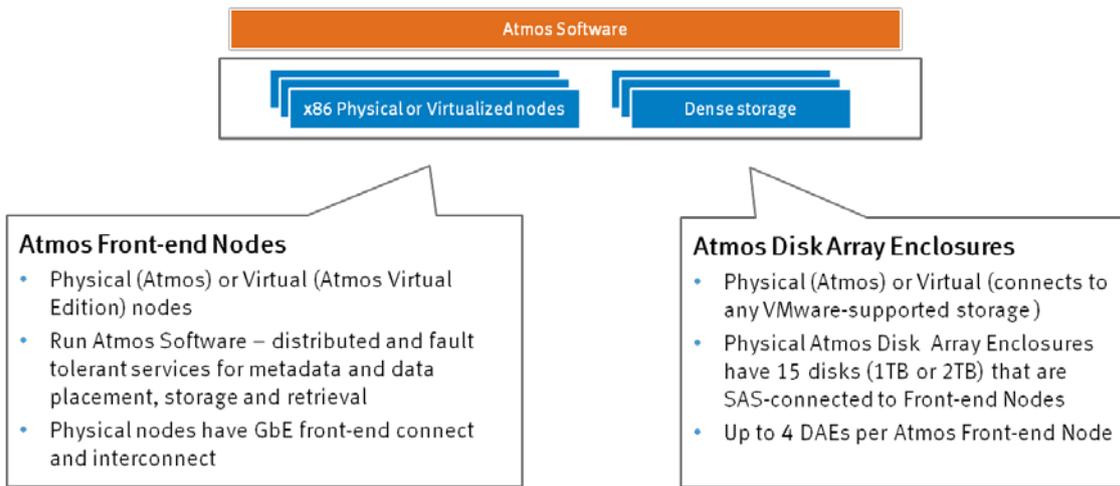
Figure 5. EMC Atmos building blocks

As shown in the picture below, a group of front-end nodes and storage is referred to as an Install Segment "IS". A site, which can consist of multiple Install Segments, is considered a Resource Management Group "RMG", which is a logical construct. Each RMG has a location-based metadata tag associated with it. Every node in the RMG will get the location tag during installation.
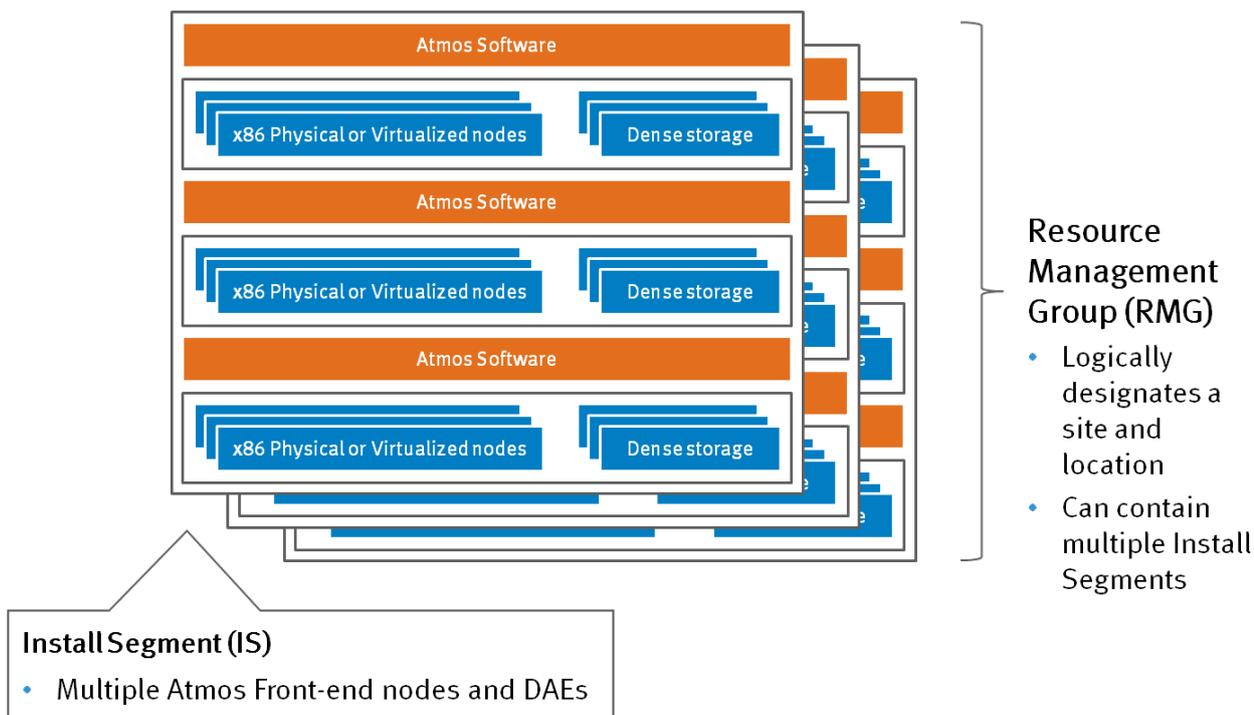
Figure 6. Installation segment

The picture below shows a high-level mapping of the abstract packaging to the Atmos physical rack:
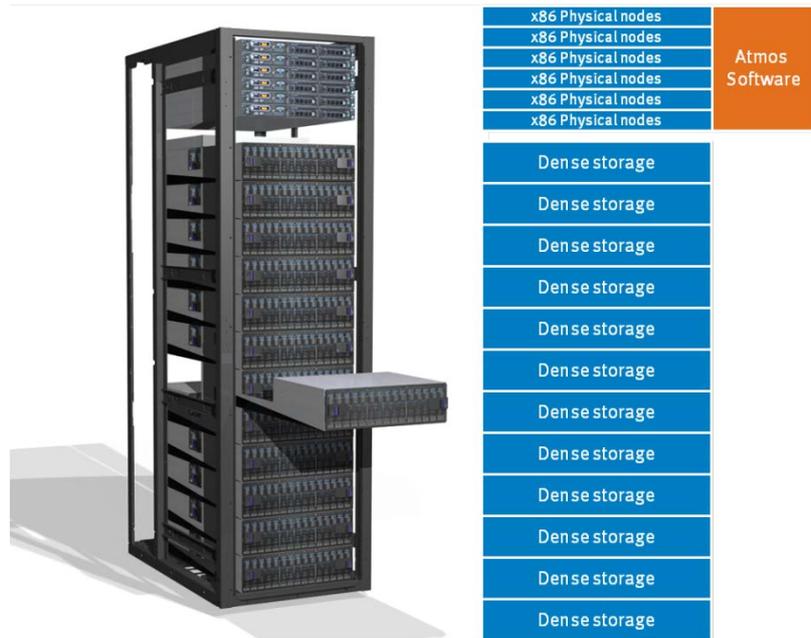
Figure 7. EMC Atmos physical rack

The picture below shows the 4 bundled physical solutions, being a G3-FLEX-180, G3-FLEX-240/360, and G3-DENSE-480 respectively. Currently, each physical node is an x86 commodity server, with 2 quad core CPUs, and 2 onboard NIC ports. The first NIC port, eth0, is connected to a private network with a bundled internal-only switch, to allow for Management, PXE and IMPI traffic between the nodes. There are 2 10G ports which can be connected to a single or dual (HA configuration) 10G switches for external network access (I/O). Each node is connected via a serial-attached SCSI (SAS) cable to a disk enclosure (DAEs). Depending on the model, one or two servers may connect to a single DAE and each DAE may be populated with 30 or 60 SATA drives. The drive capacities available are 1TB, 2TB, 4TB, and 6TB at 7200 RPM spindle speeds.
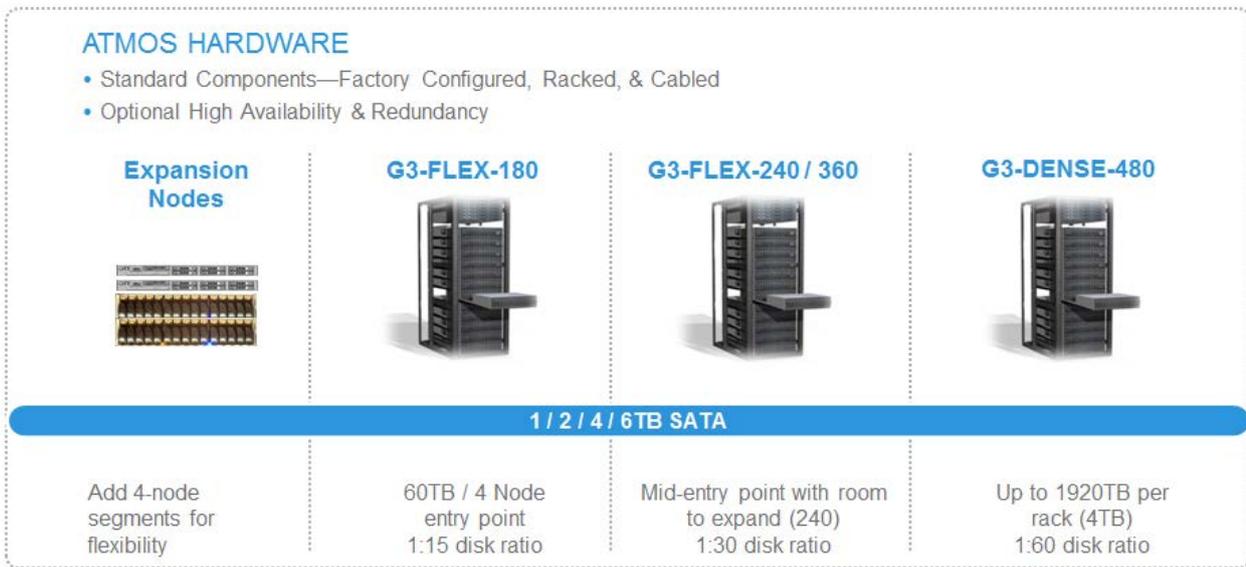
Figure 8. EMC Atmos deployment options

When Atmos is installed, the first node is installed physically via CD drive, or via an ISO file for the *Atmos Virtual Edition. The remaining nodes in the IS are then PXE-booted off of the initial node, and will self-configure according to a simply storage profile that is set.

*New deployments of Atmos VE are intended for Proof of Concept and Test QA / Development systems only. For new production deployments, please refer to the Atmos appliance for a hardware solution or ViPR for a software-defined solution.

PROVISIONING A SYSTEM WITH MULTI-TENANCY

One of the core features of Atmos is the implementation of secure multi-tenancy. A tenant is a logical compartmentalization of data and resources.   A tenant has their own access nodes, security control, storage policies, and access to the data. A tenant is not aware of another tenant's resources, and has no access to other tenant's resources. A system administrator creates a tenant, along with a tenant admin, and then assigns specific front-end nodes for web services access, as well as CIFS or NFS access. This is done on a node-by-node basis. Once a tenant gets created, a default subtenant with the same name as the tenant gets automatically created. The tenant admin can then create additional subtenants.

A subtenant owns a distinct storage environment, with its own users, shares, and objects. Each subtenant also has its own namespace. Thus, there is inherent isolation of data between subtenants.

Atmos data management policies, when created by the tenant, can be assigned to one or more specific subtenants, allowing either unique policy profiles to be assigned to specific subtenants. Any number of the same policies to be re-used and assigned to multiple subtenants.

CIFS shares and NFS exports get assigned at the subtenant level, allowing for efficient file services design, especially when it comes to multiple subtenants. These file services approaches are discussed in further detail in the next section.

In this context, subtenants can equate to specific customers, divisions or departments, all with different business rules governing the application of their data. In general, it is recommended to configure one or 2 tenants in a system, and leverage multiple subtenants underneath each tenant. User IDs at the subtenant level can then identify an application, or an individual user.

Lastly, as a general guideline, Atmos limits the number of tenants in a single system to the number of physical nodes available (as front-end nodes are assigned to a specific tenant). For optimal flexibility, most end-users will have 1 or 2 tenants within a system, followed by many subtenants and users.

The illustration below describes the relationship between subtenants and users, all in the same tenant.
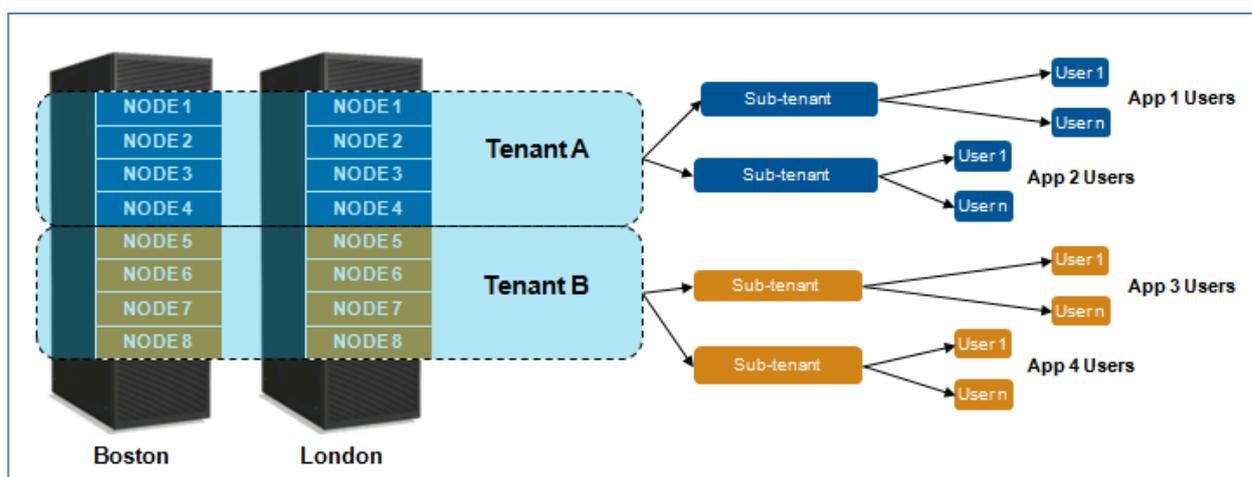
Figure 9. EMC Atmos multi-tenancy

ACCESSING A SYSTEM FOR ACCESS METHODS AND END USERS

The Atmos architecture attempts to offer the broadest range of access mechanisms possible. Most end users will interact with Atmos through pre-integrated applications, custom or packaged, so the end user will not be aware of what is storing and managing the information. Application developers at Enterprises and ISVs can best take advantage of Atmos via the Atmos Web Services interfaces (REST and SOAP), which allow ubiquitous, scalable and full featured access to Atmos. Most of the increasing number of EMC and third party integrated packaged applications use the Atmos REST API. For certain use cases, end users can also take advantage of Atmos as a mounted drive via the Atmos Installable File System (Linux) and traditional file system interfaces like NFS (Network File System for Linux) or CIFS (Common Internet File System for Windows).

Web services—REST and SOAP

Atmos provides a REST (Representational State Transfer) and SOAP (Simple Object Access Protocol)-based object storage API that lets developers write platform-independent applications to create, read, update, and delete objects, and to manipulate their metadata.

13

NFS—Network File System

Atmos supports NFS by its nature of being based on Linux. NFS is part of the core of Linux, and enabling it in Atmos follows the same rules and methods as any other Linux system. When NFS is enabled, each Atmos node is unique and configured separately from every other Atmos node with NFS enabled.

The NFS implementation also support a High Availability (HA) option. This is instantiated with a pair of Atmos nodes, configured with a virtual IP address for clients to connect to. A network heartbeat is then run over both the private and public interfaces. Should both heartbeats time out, and the primary node cannot be reached, the client connections are automatically failed over to the secondary node, without client disruption.

CIFS—Common Internet File System

When CIFS is enabled, each Atmos node acts as a CIFS server, which can be configured in stand-alone mode, or joined to an Active Directory domain. The CIFS implementation also supports the HA option, in the same manner as NFS above.

IFS—Installable File System

The IFS driver provides a file system presentation layer that allows applications access to Atmos objects through a traditional file system interface. It functions as an Installable File System using the FUSE (File-system in User SpacE) extension on Red Hat Linux clients. With this implementation, we have bundled the Atmos Client Libraries directly into an RPM package that gets installed on the Red Hat 5 server. Some configuration steps tie that client directly to a specific tenant:subtenant namespace implementation. Thus, the Linux client then becomes an extension of the Atmos system itself, as it will talk directly to the running Atmos services via TCP/IP. Given this context, the IFS client is inherently clustered with the Atmos implementation. For example, if the IFS client was communicating with an MDS service on a particular Atmos node that then becomes unavailable on the network, the IFS client will simply communicate with a different available MDS service.

The Atmos Installable File System has three main areas of functionality:

1. A human readable namespace — File systems organize files into hierarchical trees, with directories as interior nodes and files as leaves. Each file is uniquely identified by a full pathname, which represents a traversal of the tree from the root directory to a leaf node. The COS file system allows for the creation of such a namespace, then maps individual files in the namespace onto individual COS objects.
2. File-system metadata and access control: File systems typically store standard metadata about each file, such as its size, last modification time, and owner. Also, file systems control user access to directories and files via fine-grained access-control mechanisms like modification bits and access-control lists. The Atmos IFS associates standard file-system metadata and access-control information with each Atmos object named in the IFS namespace.
3. File I/O: File systems allow applications to read from, write to, and append data to files. Typically, I/O transfers may be of any size. The Atmos IFS allows efficient, scalable I/O on Atmos objects. By default, the Atmos IFS provides close-to-open consistency of files.

Atmos GeoDrive

Atmos GeoDrive is an optional add-on software product that provides a single user or application instant access to an Atmos private, public or hybrid cloud, from any Windows or Linux desktop or server.

Unified namespace and object space

Atmos is implemented as an object-based storage system that uses a single object namespace across multiple sites. The benefits of this architecture results in an active/active configuration distributing objects across all nodes in the infrastructure. In this context, every file or folder lives in the system as an object. To create a client readable namespace by traditional access methods, the architecture is required to keep a consistent directory structure across multiple sites. This is known as using the namespace implementation. To that end, it is important to denote the difference between using the Web Services Object API, versus the Namespace API. Note that the "Namespace API" encompasses both file system (NFS, CIFS, IFS) and the Web Services REST Namespace API option.
The current metadata architecture for Atmos places two read / write copies of the metadata of an object at one site, but on different nodes. Assuming MDS Replication is enabled, a read-only copy of the metadata is placed at the other site.

If an application is only using the REST Object API, there is no directory structure required. Hence, an application can make a new "Create Object" API call, and through a Load Balancer, transparently create the new object on a surviving site when the primary is not available.

MANAGING DATA IN THE SYSTEM VIA POLICY

This section provides an overview of the policy manager implementation in Atmos.

The Atmos Policy Manager is responsible for classifying all objects that are ingested into the system. When an object is classified, the appropriate layout and features can be applied to the object when it is written, or upon user or system metadata updates. These layout and feature options include such things as the number, type and location of replicas, as well as options such as striping, compression, or de-duplication. With policy, objects from different applications, or of different types, may be treated differently by the Atmos system, ensuring that the service level objective appropriate to the type of object is applied.

For example, a high value customer or process may be defined as "priority" and therefore require different protection or access – such as highly available copies in more locations. With Atmos you can satisfy this requirement, while also managing things such as age-based policies for compression, de-duplicated, fewer copies in fewer locations, etc.   These and many other polices can be set by tenant, driven by data, and are automated to minimize operating costs while still meeting the service level agreements.

Policy manager

The Atmos Policy Manager is responsible for classifying all objects that are ingested into the system. When an object is classified, the appropriate layout and features can be applied to the object when it is written, or upon user or system metadata updates. These layout and feature options include such things as the number, type and location of replicas, as well as options such as striping, compression, or de-duplication. With policy, objects from different applications, or of different types, may be treated differently by the Atmos system, ensuring that the service level objective appropriate to the type of object is applied.

Policy selection

The Policy Selector defines the exact system or metadata trigger with which to apply a specific Policy Specification to an object. In this context, a policy can trigger on both an object Create, and an object Update. The create function and update function can also each have different Policy Specifications applied. For example, a JPG file can have an initial object Create policy of 2 local synchronous replicas defined by a system metadata trigger of ".jpg" in the object name. The user can then define a second Policy on an Update Selector adding a remote asynchronous replica. The Update can be triggered an Access time (atime) system metadata update, or by adding user-defined metadata.

In addition, age-based retention and automatic deletion can also be applied at the Policy level. An administrator can define a policy that makes an object immutable for 30 days, and then deletes the object automatically five days later, for example.
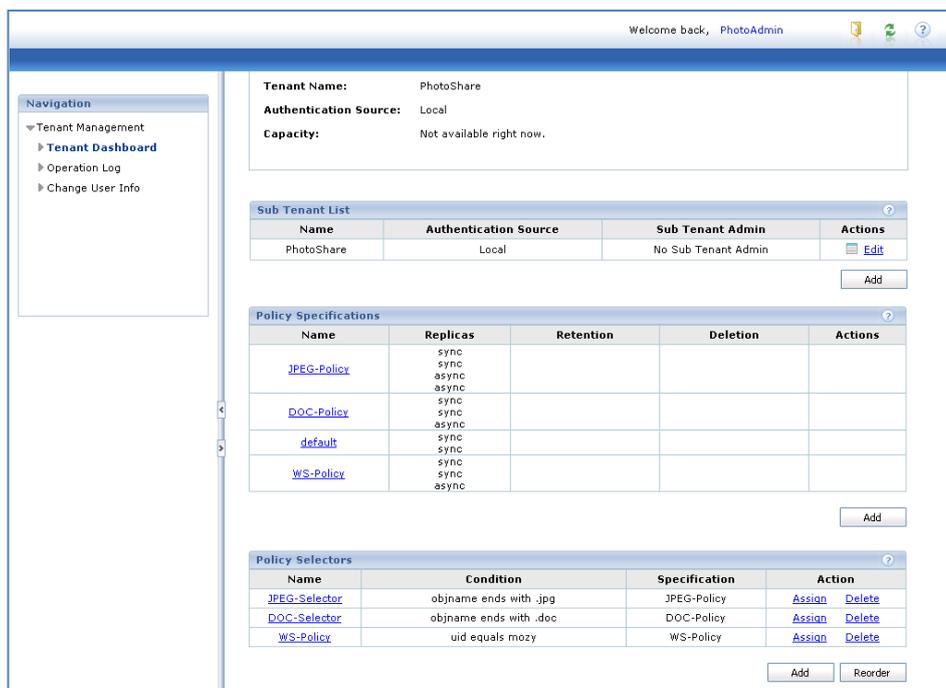
Policy specification

Let's continue our example from the Selector section. In our example, we have classified .jpg files and .doc files and associated each with a specification. The specification defines exactly how to lay out the objects of each type. The administrator and architects have decided that photos should have four replicas, 2 synchronous in one location, and 2 asynchronous in a second location. These replicas should be on Storage Servers that are configured as optimal, and do not support any data transformations. The JPG Specification would define this requirement in Atmos specific terminology, and the user would create this specification via the Policy Specification screen within the Tenant dashboard of the Atmos UI. The DOC Specification would describe a different layout, perhaps a smaller number of replicas that should be used for the placement of word documents.

As a photo.jpg enters the Atmos system, to be created, it will be categorized as such by the Selector, which associates it with the proper specification. At this point, the Policy Manager must find resources that satisfy the constraints of the specification. To accomplish this task, the Resource Manager must be queried. For a JPG, the policy manager must ask the resource manager for four distinct storage servers, 2 in one location and 2 in a second specific location, none of which support compression or de-duplication, and that support the layout option of optimal.

Finally, Atmos offers the capability of automating an object lifecycle even further, through the use of a Policy Transition Specification. With this feature, an administrator can define one policy on policy ingest, then automatically converting to a different policy after a defined time period. For example, I may have a type of object that is "hot" to begin with, and then becomes more stagnant some time later. In this context, I may want to have 2 copies of the object initially at each of my 4 geographic sites, and then after 90 days, keep a single Geoparity replica that has a lifetime retention set.

Policy "features" for specification

Policies are set through the Atmos User Interface, or system management CLI. Specifically, setting policy comes under the domain of the tenant administrator. A tenant administrator defines the policy selectors and specifications applicable for various levels of data protection, and in turn assigns those to subtenants.

Figure 10. EMC Atmos tenant dashboard

As illustrated above, the policy configuration is broken out into two steps. Policy Specifications must be defined, and then a Policy Condition must be written and assigned a specification for use. The tenant dashboard offers an at-a-glance view of configured polices and the specification associated with each. For example, we can tell above the JPEG-Selector is querying for objects with an objname that ends with .jpg. Objects of that class are associated to the JPEG-Policy Specification. From the Policy Specifications pane, the JPEG-Policy specification defines four replicas: 2 synchronous and 2 asynchronous. Clicking on the link for the specification provides more details around the locations of those replicas and the type of storage required for each.

When defining the specification, it is important to understand each selection, and the specific implementation options for each feature, including replication, storage server layout options, storage server transformation options and optional lifecycle features.

PROTECTING DATA IN THE SYSTEM WITH REPLICATION

Atmos supports two forms of replication: synchronous replication and asynchronous replication. For a particular object, both types of replication can be specified, with synchronous replicas being tied to certain location and a number of asynchronous replicas in other locations.

Replica definition for an object class is configured at the policy specification screen by the tenant administrator. It is the first step in determining where copies of an object will reside. These replicas, or copies, can be used to drive availability, disaster recover, content distribution and read performance. Replicas ensure that data is available despite hardware failures, and that users, regardless of location, may have a copy of the data close to them to read from. It is important to determine how many, which location(s) and which type of replicas need to be defined for each class of object.

Synchronous replicas are those which the Atmos client services writes before returning an acknowledgement of successful write to the requesting application. The client services perform all synchronous writes, to all specified destinations, in parallel. For this reason, it is important to ensure that adequate bandwidth and tolerable latencies exist between sites across which synchronous replication may occur. In general, it is best practice to limit synchronous replication to within a data center, or to campus-wide distribution.

Asynchronous replicas are used in policy with one or more synchronous replicas, and are usually placed in a separate location (RMG). Asynchronous replicas are placed in a Job Queue, and destaged as the queue empties, whether locally or remote.

Replicas, whether synchronous or asynchronous, utilize the metadata location tag identified upon install for each location. Through policy, each replica can then be assigned a specific location (E.g. – "sameAs" or "otherThan" "Boston"), or can utilize the "$client" tag. This tag signifies the Atmos node where the write comes into. In this context, if there are multiple sites, a policy can then place one or more replicas at the site where the write occurred, and will thus scale a single policy on a global scale. The "$client" tag can also utilize the "sameAs" or "otherThan" parameter within policy as well.

GeoParity

GeoParity is a policy-based replication feature that is available in Atmos as of version 1.3. Its name has nothing to do with deleting objects, but rather comes from the concept of "erasure channels", a model that has been traditionally used in the telecommunications industry. For Erasure Coding in Atmos, the idea is to split up an object into m data fragments and k coding fragments. The coding fragments can be thought of as parity fragments, with the associated data fragments. These fragments are then placed in independent fault domains, residing on different disks, on different nodes, and can even be on different RMGs within a system. This overall process is referred to as "encoding" the object. When an object is then requested by a client, Atmos will then "decode" the fragments back into the original object. One point of note is that a decode operation only needs a minimal subset of the overall m + k fragments to satisfy the client request.

Atmos implements GeoParity with the Cauchy Reed-Solomon algorithm, and uses two different implementations. The first is a 9/12 configuration, where an object is split into 9 data fragments and 3 coding fragments. This effectively tolerates up to 3 drive failures, and gives the object 5 nines of data availability. In addition, the storage overhead is only 33%, compared with 100% for a RAID-1 configuration, or 25% for a RAID-5 configuration (using k/m to calculate overhead: 3/9, 1/1 or ¼). Such benefits come at a price, in this case some performance overhead is required for the encoding and decoding operations.

The second configuration is a 10/16 option, where there are 10 data fragments and 6 coding fragments. This configuration will tolerate up to 6 drive failures, has a 60% storage overhead, but at a cost of additional performance overhead than the 9/12 configuration.

There is an inherent MD5 inline integrity checksum that is calculated with each encoding operation on the fragments for erasure coding. Once a client requests the object, the checksum is calculated again against the stored checksum. If the checksum does not match for any fragments, that fragment or fragments is dropped.

Finally, it is recommended to implement replicas with Erasure Coding on data stored for archival or long-tail purposes. In this example, the write performance would only be a concern on the initial ingest, but this can be controlled with a migration plan for the data. Thereafter, this type of data is usually expected to have only reads – not updates – and the implementation receives the benefit of higher fault tolerance with significant storage overhead savings. For remote disaster recovery or read access purposes, consider adding a remote asynchronous replica.

DEPLOYING AND GROWING A SYSTEM WITH AUTOMATION

Scaling

Capacity can be added seamlessly in two methods: either by expanding capacity within an existing datacenter by deploying new cabinets of purpose-built, preconfigured nodes and disk enclosures, or by expanding incrementally by simply adding nodes and disk enclosures to existing cabinets. In addition to flexible growth, Atmos can add this new capacity seamlessly to an up and running cloud. Once new capacity is added to the cloud, it is immediately available for use without any provisioning needed. In this context, Atmos scales horizontally as rack models can be mixed and matched together, both within a site and between sites. As long as MDS symmetry is met for MDS replication between sites, the deployment options are very flexible.

Auto configuration

As mentioned earlier, Atmos is installed by physically installing the first node in each Installation Segment. The remaining nodes in the segment are then pxe-booted off of the initial node, and self-install according to a user-defined template. Both the configuration of the nodes and the services are automatically executed by the software.

Software auto healing

Atmos is implemented as a software storage system with multiple distributed services running on networked hardware front-end nodes. As depicted in the graphic below, the majority of these services (MDS, RMS, CS, JS, SS) run on every node in every rack at every site, with the exception of the MDLS service, which runs on the first two nodes of the first rack at every site. With this design, there is no single point of failure in regard to services or processes. If one of the services goes down, if the node becomes unavailable, or an entire rack or site becomes unavailable, there are redundant services that will facilitate normal operations. In addition, Atmos will also periodically retry a service that has stopped or gone inactive.
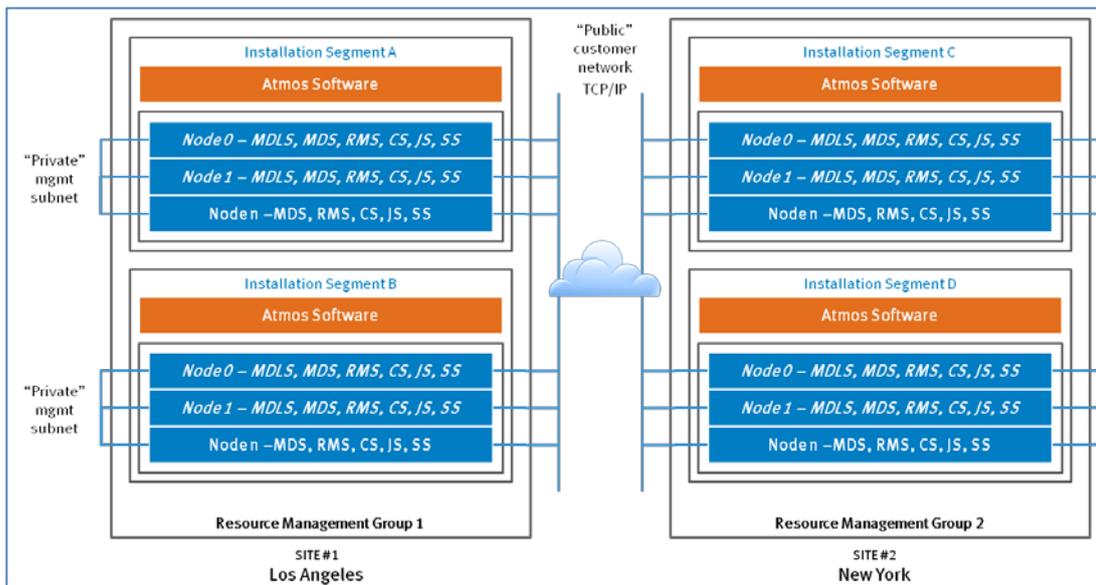
Figure 11. EMC Atmos self-healing services

Hardware auto healing

Atmos storage does not use hardware RAID or disk-level protection. Rather, Atmos utilizes a policy engine to enable the user to classify unstructured data based on user-defined service levels. Thus, when Atmos encounters a hardware failure, the defined policies for each sub-tenant, and how the subsequent data, plays a large role. For example, if I define a policy to keep one copy of a certain type of data, I am defining a specific Service Level for that data. If the disk, DAE, or node where that data resides on encounters a failure, the data is no longer available. However, if I define a policy to keep multiple copies of a certain type of data at multiple sites; I am defining a higher level of resiliency and Service Level for that data.

VIEWING THE ENTIRE STACK FROM APP TO SPINDLE

To tie together the concepts of the service architecture, data flow, packaging and deployment, the high level diagram below illustrates how a read request flows from a user to an application server down to Atmos in a geographically distributed deployment.
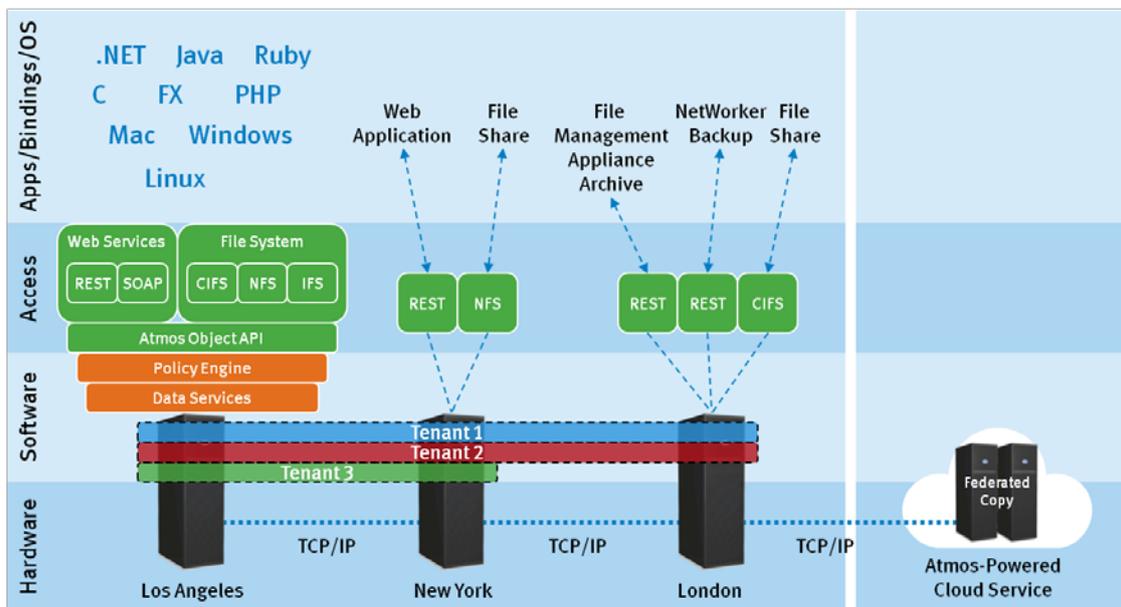

Figure 12. EMC Atmos service architecture in a distributed environment

EMC also understands that not everyone is capable or interested in developing to an API. For these customers EMC offers traditional file sharing integrations like NFS, CIFS, IFS and packaged applications and solutions from our ISV partners that are more turnkey.

Moving down from the interfaces is a set of distributed Atmos software services that manage users and information in a highly automated fashion. This includes data protection and efficiency by policy, a unified namespace, and the ability to split the namespace into multiple tenants. All of this appears as one system, and is managed as one system, regardless of how distributed the physical hardware may be. This in turn enables much greater scalability and lower management overhead than traditional systems.

CONCLUSION

In closing, EMC Atmos helps you more efficiently manage distributed unstructured big data in the cloud. Atmos cloud storage platform provides a single system across your entire storage environment you can:

• Aggregate distributed big data to drive business decisions
• Seamlessly and elastically expand storage and applications without the need to write or rewrite code which means zero impact to availability
• Automate storage tasks through policy to reduce time spent on mundane / non-valued tasks
• Make Big Data Smart to help drive business decisions and explore new opportunities
• Make Big Data Accessible from any device, network, or location
• Offer storage-as-a-service to give IT the power to meter and manage capacity across tenants, and provide self-service for users to manage their own storage.

When it comes to Big Data, EMC Atmos provides you the right platform to manage distributed big data in the cloud.

To learn more about the Atmos Product family, see http://www.emc.com/atmos

APPENDIX

Terms and Definitions

Atmos Resource Management Group (RMG)
RMG. A collection of installation segments that share a single domain. In almost all cases, this is equivalent to a subnet on the "public," customer network. You can create multiple RMGs on the same subnet, as long as each RMG has a unique address. RMGs are responsible for monitoring and discovering nodes within the subnet.

Installation Segment (IS)
A set of nodes that share the same "private," management subnet. It is often referred to a single physical or virtual storage rack.

Resource Management Service (RMS)
• Tracks availability, status and attributes of all Storage Servers
• Tracks availability and location of all other services

Metadata Services (MDS)
• Multiple databases
• Stores the user, system and layout for objects

Metadata Location Services (MDLS)
• Maintains index of objects to owning MDS

Atmos Cloud Delivery Portal (ACDP)
EMC Atmos Cloud Delivery Platform is an add-on software product that enables enterprises and service providers to deliver and manage storage-as-a-service to an Atmos cloud.
Client Services (CS)
• Provides presentation interfaces over eth1
• File access (NFS, CIFS, IFS) or Web Services

Client Layer (CL)
• Runs the Client Services API to interact with the other services

Storage Service (SS)
- Responsible for writing to physical disk
- Responds to all I/O requests

Job Services (JS)
- Responsible for background replication tasks

Policy Manager (PM)
- Determines what policy gets assigned to incoming object writes

Web Services (WS)
- Includes Web Services protocols of SOAP and REST for client access

Object ID (OID)
- This is the unique identifier that is assigned to every object within Atmos. An example of an OID could be 4924264aa10573d404924281caf51f049242d810edc8.

File Services (FS)
- Covers legacy protocols such as CIFS and NFS for client access
- Also includes Atmos' Installable File System (IFS).

Installable File System (IFS)
- The Atmos IFS client bundles the Atmos Client Libraries into a RedHat RPM package, to allow any RHEL5 server to communicate to the Atmos system, by directly mounting a specific Tenant: Subtenant namespace on the RHEL server as a standard mount point.

User ID (UID)
- A user defined in the Web Services (SOAP or REST) protocols.