Cloud Storage Reference Model

Version 0.3 rev 0

Publication of this Trial-Use Draft Specification for trial use and comment has been approved by the SNIA Technical Council and the Cloud Storage TWG. Distribution of this draft specification for comment shall not continue beyond 6 months from the date of publication. It is expected, but not certain that following this 6 month period, this draft specification, revised as necessary will be submitted to SNIA Technical Council for final approval. Discussions concerning this document should be held on snia-cloud@googlegroups.com, suggestions for revision and concrete content addition should be directed to http://snia.org/feedback.

Trial-Use Draft

June 8, 2009

Revision History

Date	Version	By:	Comments
6/2/09	0.1	Mark Carlson	Initial Draft version 0.1
6/4/09	0.2	Mark Carlson	TWG Edits from face to face
6/5/09	0.3	Mark Carlson	Changed to a single interface, incorporated interface model

Table of Contents

Abstract

.

This paper sets out a model of cloud storage elements that describes a logical view of their functions and capabilities using a descriptive taxonomy. The purpose of this model is to form a basis upon which industry efforts can be organized, needed standards identified and vendor products can be described by vendor independent terminology. In addition, this model is used to describe standard interfaces for cloud storage.

Introduction

When discussing cloud storage and standards, it is important to distinguish the various resources that are being offered as services. These resources are exposed to clients as Functional interfaces (Data Path) and are managed by Management interfaces (Control Path). We explore the various types of interfaces that are part of offerings today and show how they are related. We propose a model for the interfaces that can be mapped to the various offerings as well as form the basis for rich cloud storage interfaces into the future.

Another important concept explored in this paper is that of *MetaData.* When managing large amounts of data with differing requirements, metadata is a convenient mechanism to express those requirements in such a way that underlying data services can differentiate their treatment of the data to meet those requirements.

The appeal of cloud storage is due to some of the same attributes that define other cloud services: pay as you go, the illusion of infinite capacity (elasticity), and the simplicity of use/management. It is therefore important that any interface for cloud storage support these attributes, while allowing for a multitude of business cases and offerings, long into the future.

What is Cloud Storage?

The use of the term *cloud* in describing these new models arose from architecture drawings that typically used a cloud as the dominant networking icon. The cloud conceptually represented any to any connectivity in a network, but also an abstraction of concerns such the actual connectivity and the services running in the network that accomplish that connectivity with little manual intervention.

This abstraction of complexity and promotion of simplicity is what primarily constitutes a cloud of resources, regardless of type. An important part of the cloud model in general is the concept of a pool of resources that is drawn from upon demand in small increments (smaller than what you would typically purchase by buying equipment). The recent innovation that has made this possible is virtualization.

Thus cloud storage is simply the delivery of virtualized storage on demand. The formal term we propose for this is *Data Storage as a Service* (**DaaS**).

Data Storage as a Service

Delivery of virtualized storage on demand.

Data Storage as a Service

By abstracting data storage behind a set of service interfaces and delivering it on demand, a wide range of actual offerings and implementations are possible. The only type of storage that is excluded from this definition is that which is delivered, not based on demand, but on fixed capacity increments.

An important part of any DaaS offering is the support of legacy clients. This is accommodated with existing standard protocols such as iSCSI for block and CIFS/NFS or WebDAV for file network storage as shown below:

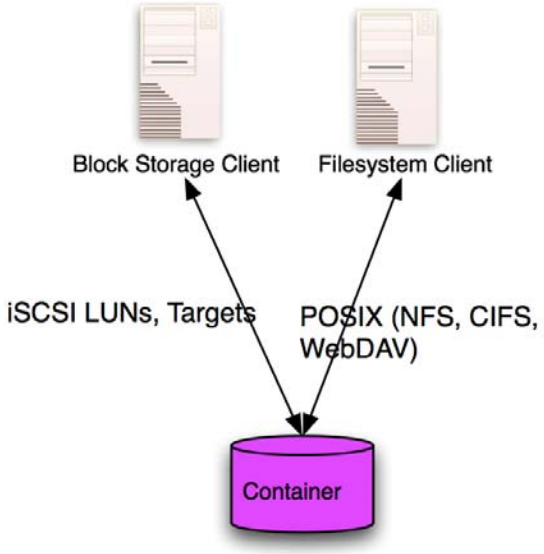

Figure 1: Existing Data Storage Interface Standards

The difference between the purchase of a dedicated appliance and that of cloud storage is not the functional

interface, but merely the fact that the storage is delivered on demand. The customer pays for either what they actually use or in other cases, what they have allocated for use. In the case of block storage, a LUN or virtual volume is the granularity of allocation. For file protocols, a filesystem is the unit of granularity. In either case, the actual storage space can be thin provisioned and billed for based on actual usage. Data services such as compression and deduplication can be used to further reduce the actual space consumed.

The management of this storage is typically done out of band of these standard Data Storage interfaces, either through an API, or more commonly, though an administrative browser based user interface. This interface may be used to invoke other data services as well, such as snapshot and cloning.

In this model we abstract the underlying storage space exposed by these interfaces using the notion of a container. A container is not only a useful abstraction for storage space, but also serves as a grouping of the data stored in it, and a point of control for applying data services in the aggregate.

Another type of DaaS offering is one of simple table space storage, allowing for horizontal scaling of database-like operations needed by certain applications. Rather than virtualizing relational database instances, these offerings offer a new data storage interface of limited functionality with the emphasis on scalability rather than features. This allows the tables to be partitioned across multiple nodes based on common key values, affording horizontal scalability at the expense of functions that can typically only be implemented by a vertically scaled relational database.

There is a great deal of innovation and change happening in these interfaces, and the offerings each have their own unique proprietary interface as shown below:

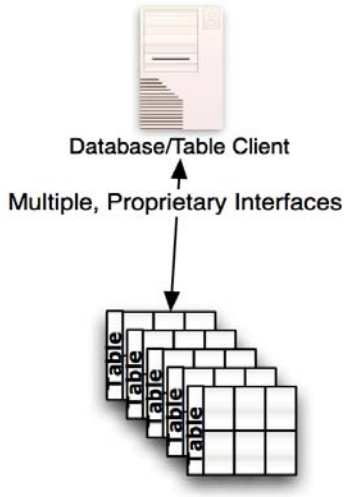

Figure 2: Database/Table Data Storage Interfaces

Due to the rapid innovation in this space, it is probably best to wait for further development of this type of cloud storage before trying to standardize a functional interface for this type of storage.

There is a third category of functional interface for Data Storage that has emerged. This type of interface treats every data object as accessible via a unique URI. It can then be fetched using the standard HTTP protocol and a browser can be used to invoke the appropriate application to deal with the data.

Each data object is Created, Retrieved, Updated and Deleted (CRUD) as a separate resource. In this type of interface, a container, if used, is a simple grouping of data objects for convenience. There is nothing preventing the concept of containers in this case from being hierarchical, although any given implementation might support only a single level of such. We call this type of container a “soft” container as shown below:

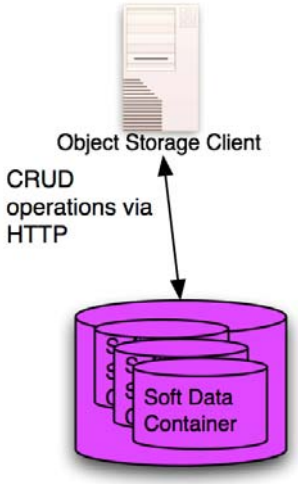

Figure 3: CRUD/HTTP Data Storage Interfaces

While there are several proprietary examples of this type of interface, they all pretty much support the same set of operations. This, then, is an area ripe for standardization.

Managing Data in the Cloud

Many of the initial offerings of cloud storage focused on a kind of "best effort" quality of storage service with very little offering of additional data services for that as additional data services are applied, and the implication that these services need to be managed.

Fortunately, the SNIA Resource Domain Model gives us way to minimize this complexity and address the need of cloud storage to remain simple. By using the different types of metadata discussed in that model for a cloud storage interface, we can create an interface that allows offerings to meet the requirements of the data without adding undo complexity to the management of that data.

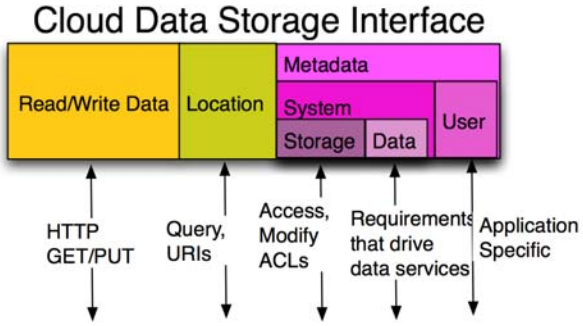

Figure 4: Using the Resource Domain Model

By supporting metadata in a cloud storage interface standard and proscribing how the storage and data system metadata is interpreted to meet the requirements of the data, we can retain the simplicity required by the cloud storage paradigm, and yet still address the requirements of enterprise applications and their data.

Managing Data and Containers

There is no reason that the management of data and the management of containers should involve different paradigms. We therefore propose extending the use of metadata from applying to individual data elements into applying to containers of data as well. Thus any data placed into a container essentially *inherits* the metadata of the container it was placed into. Creating a new container within an existing container would similarly inherit its parent's metadata settings. Of course the data. In order to address the needs of enterprise applications with cloud storage, however, there is increasing pressure to offer better quality of service and the deployment of additional data services.

The danger, of course, is that cloud storage loses its benefit of simplicity and the abstraction of complexity metadata can be overridden at the container or individual data element level as desired.

Even if the functional interface provided by the offering does not support this type of metadata on individual data elements, it can still be applied to the containers even though it cannot be overridden on the basis of individual data elements. For file-based interfaces that support extended attributes (i.e. CIFS, NFSv4), these extended attributes can be used to specify the Data System Metadata to override that specified for the container. The mapping of extended attribute names and values to individual file data requirements as supported by cloud storage will be done as a follow on effort.

Reference Model for Cloud Storage Interfaces

Putting it all together we have the model as shown below:

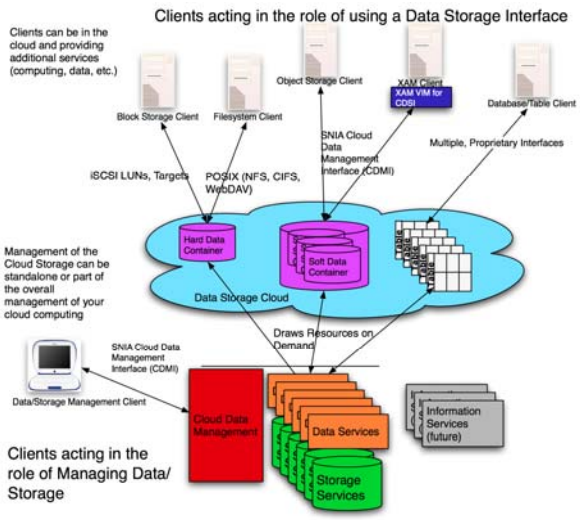

Figure 5: Cloud Storage Reference Model

This model shows multiple types of cloud data storage interfaces able to support both legacy and new applications. All of the interfaces allow storage to be

provided on demand, drawn from a pool of resources. The capacity is drawn from a pool of storage capacity provided by storage services. The data services are applied to individual data elements as determined by the data system metadata. Metadata specifies the data requirements on the basis of individual data elements or on groups of data elements (containers).

SNIA Cloud Data Management Interface

As shown in Figure 5, the SNIA Cloud Data Management Interface (CDMI) is the functional interface that applications will use to create, retrieve, update and delete data elements from the cloud. As part of this interface the client will be able to discover the capabilities of the cloud storage offering and use this interface to manage containers and the data that is placed in them. In addition, metadata can be set on containers and their contained data elements through this interface.

It is expected that the interface will be able to be implemented by the majority of existing cloud storage offerings today. This can be done with an adapter to their existing proprietary interface, or by implementing the interface directly. In addition, existing client libraries such as XAM can be adapted to this interface as show in Figure 5.

This interface is also used by administrative and management applications to manage containers, accounts, security access and monitoring/billing information, even for storage that is accessible by other protocols. The capabilities of the underlying storage and data services are exposed so that clients can understand the offering.

Conformant cloud offerings may offer a subset of either interface as long as they expose the limitations in the capabilities part of the interface.

Object Model for CDMI

The model behind the Cloud Data Management Interface is shown below in Figure 6:

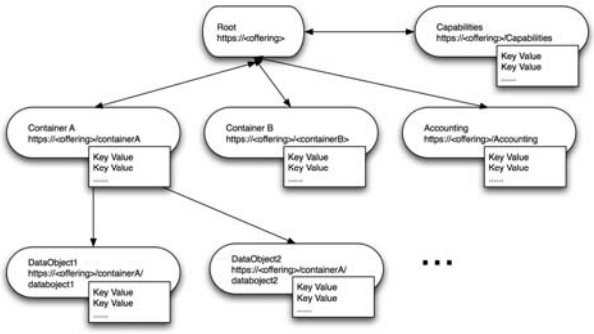

Figure 6: CDMI Interface Model

For Data Storage operations, the client of the interface only needs to know about Containers and Data Objects. All implementations are required to support at least one level of Containers - a sort of grouping of Data Objects. The Client may do a PUT to the Container URI as shown above and create a new container with the specified name. The KEY/VALUE metadata shown is optional. Once a Container is created, a Client may do a PUT to a Data Object URI as shown. A subsequent GET will fetch the actual Data Object. The only Metadata KEY/VALUE required on the Data Object PUT is Content Type (MIME). Other KEY/VALUE pairs can be used to specify the data requirements at the data object or container level. This metadata is defined in the CDMI specification.

The CDMI need not be used as the data path, and indeed is applicable to cloud storage exposed as standard interfaces. In this case, the client can stop at creating the Container. The metadata is used to configure the exported interface that the container exposes, such as a block protocol or a file protocol.

A cloud offering can also support an Accounting URI that allows cloud storage accounts and credentials to be managed. Accounts can also be hierarchical, allowing for corporate accounts with multiple children accounts for departments or individuals. The Accounting resource is also used for billing, metering and monitoring of usage information.

Lastly there is a Capabilities resource and associated URI that allows a client to discover the capabilities of the offering and it's implementation of CDMI. This is a required resource of the interface, but can be satisfied by static pages listing what exactly is implemented.

Cloud Peering

The concept of cloud peering is similar to the peering of networks in an Internet. The idea is that a cloud can act as either/both a client or a server side of the interface and form an agreement with another offering to share

data and storage space. This could be done for economic, technical, geographic, legal or other reasons. This is show below in Figure 7:

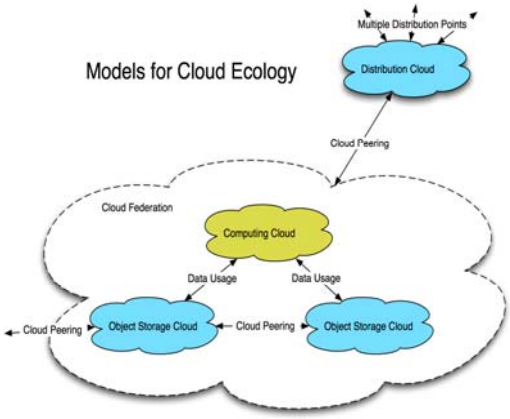

Figure 7: Cloud Storage Peering

The Distribution Cloud in this picture is a specialized object storage cloud, used to reduce latency to the cloud consumer by geographically positioning the data closer to the consumer. This is an example of a non-symmetric peer. Peering can also be used to migrate or mirror data on the fly from vendor to vendor. There are multiple use cases that become possible due to a standard API.