An Introduction to CDMI

DAVID SLIK

David Slik is the Technical Director for Object Storage at NetApp and is a co-chair of the SNIA Cloud Storage Technical Working Group. He participated in the creation of the XAM and CDMI standards and has architected and developed large-scale distributed object storage systems deployed worldwide as a key component of mission-critical enterprise applications.
dslik@netapp.com

In early 2009, recognizing the growing importance of cloud storage, and building on top of the earlier XAM object storage standard [1], the Storage Networking Industry Association (SNIA) [2] started a new technical working group with the primary goal of creating an industry standard for cloud storage. The result was the Cloud Data Management Interface (CDMI) [3], which was released as a formal industry standard in 2011 and is currently in the process of becoming an ISO/IEC standard.

The creation of the CDMI standard has been a collaborative effort, with contributions from over a hundred storage vendors, end users, and university researchers. All of the major enterprise storage vendors contributed to the standard, including Dell, Cisco, EMC, HP, Hitachi Data Systems, Huawei, IBM, Intel, LSI, NetApp, Oracle, Symantec, and VMware [4]. This represents a unique cross-industry endorsement of cloud storage, and the results are clearly visible in the breadth of use cases that CDMI is able to address.

In the year following the initial publication of CDMI, the SNIA has published an errata release of the standard and held four plugfests to demonstrate interoperability and conformance of open source, research, and commercial implementations. In the coming year, additional milestones will be reached, with major storage vendors bringing CDMI-compatible systems to market, and work ongoing to add CDMI to open source platforms, including OpenStack [5].

Design Principles

The following principles guided the design of the CDMI protocol:

- **Complementary**—A key design principle is that CDMI is designed to complement, not replace, existing NAS, SAN, and object protocols. Traditional file systems and LUNs can be managed out-of-band using CDMI, in conjunction with access via NAS and SAN protocols. CDMI can also be used as a self-service management and/or access protocol alongside other object protocols, such as OpenStack's Swift. CDMI adopts many best-practice designs from existing protocols, such as NFSv4 ACLs, XAM globally unique identifiers, RESTful HTTP, and JSON structured metadata. Furthermore, as a protocol specification, CDMI places minimal restrictions on how servers are implemented, allowing it to be easily added to existing file, object, and cloud servers.
- **Simple**—In order to foster adoption and reduce the cost required to implement CDMI, the protocol is designed to be as simple as possible. By building on top

of HTTP, standard libraries and language constructs can be used, reducing the need for cloud libraries and allowing direct access by JavaScript browser-based clients. Using standard HTTP authentication and security mechanisms avoids complex header calculations. And providing the ability to start simple and add complexity only when needed reduces the learning curve and simplifies client code. Storing and retrieving your first CDMI object is as easy as:

```
demo$ curl -X PUT -d 'Hello CDMI World' -k http://127.0.0.1:18080/hello.txt
demo$ curl -X GET -v -k http://127.0.0.1:18080/hello.txt
```

This simplicity makes CDMI very script-friendly, allowing it to be easily used to create structured data repositories. At a recent coding challenge, a distributed CDMI-based temperature monitoring and reporting system was developed in hours, complete with a Web-based JavaScript front-end that retrieved data directly from the repository.

- **Extensible**—Recognizing that cloud storage is still in its infancy and that custom features are often required and desired, CDMI was designed from the ground up to allow functionality to be added to the standard without breaking client interoperability. CDMI allows clients to dynamically discover what features a server implements, and it allows clients to discover profiles of capabilities required to perform common use cases. The SNIA also has defined a process by which emerging extensions can be documented, and once multi-vendor implementations have been demonstrated, they can then be incorporated into the next version of the standard.

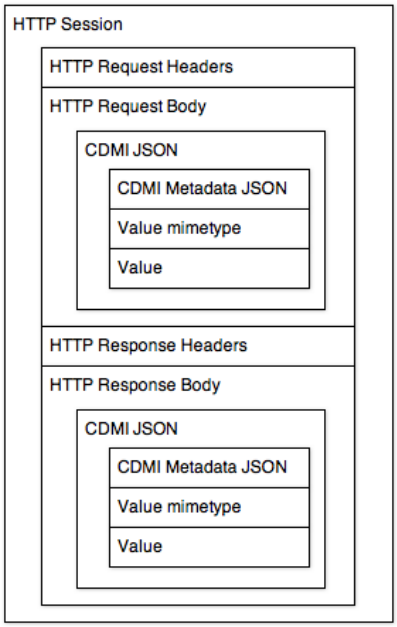

Figure 1: CDMI requests and responses are embedded in HTTP sessions.

CDMI as a Storage Protocol

CDMI is an encapsulation protocol based around RESTful HTTP. Representational State Transfer, or REST, was initially described by Roy Fielding in Chapter five of his PhD dissertation [6], and codifies a series of architectural patterns for the creation of Web-scale distributed systems. The key principles of RESTful architectures include stateless communication, idempotent operations with minimal side effects resulting from repeating a given transaction, and the use of negotiated "representations" for entities that are transferred between clients and servers.

CDMI defines five basic representations (content-types), described in RFC 6208, which are transferred between a client and a server in HTTP Request Bodies and HTTP Response Bodies, as illustrated in Figure 1.

While CDMI 1.0 defines JSON-based representations, the standard is structured such that XML representations can easily be added.

CDMI also defines "Non-CDMI" interactions, where the value is directly transferred in the HTTP request and response body. This provides the ability for a CDMI server to act as a standard Web server to unmodified Web clients and browsers.

A Non-CDMI Request for a stored data object returns a standard HTTP response:

```
demo$ curl -X GET -v -k http://127.0.0.1:18080/hello.txt
* Connected to 127.0.0.1 (127.0.0.1) port 18080 (#0)
> GET /hello.txt HTTP/1.1
> User-Agent: curl/7.19.7 (universal-apple-darwin10.0) libcurl/7.19.7
OpenSSL/0.9.8r zlib/1.2.3
```

```
> Host: 127.0.0.1:18080
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: text/plain
< Content-Length: 16
<
Hello CDMI World
* Closing connection #0
```

A CDMI Request for the same stored data object returns the CDMI JSON representation, which includes additional information about the stored object:

```
demo$ curl -X GET -v --header 'Accept: application/cdmi-object'--header
'X-CDMI-Specification-Version: 1.0.1' -k http://127.0.0.1:18080/hello.txt
* Connected to 127.0.0.1 (127.0.0.1) port 18080 (#0)
> GET /hello.txt HTTP/1.1
> User-Agent: curl/7.19.7 (universal-apple-darwin10.0) libcurl/7.19.7
OpenSSL/0.9.8r zlib/1.2.3
> Host: 127.0.0.1:18080
> Accept: */*
> Content-Type: application/cdmi-object
> X-CDMI-Specification-Version: 1.0.1
>
< HTTP/1.1 200 OK
< Content-Type: application/cdmi-object
< Content-Length: 1033
< Connection: close
< X-CDMI-Specification-Version: 1.0.1
<
{
  "objectType": "application/cdmi-object",
  "objectID": "00007ED90012E02F466C7574746572736879",
  "objectName": "hello.txt",
  "parentURI": "/",
  "parentID": "00007ED9001OC2A44D4C503A46694D21",
  "domainURI": "/cdmi_domains/",
  "capabilitiesURI": "/cdmi_capabilities/dataobject/",
  "completionStatus": "Complete",
  "mimetype": "text/plain",
  "metadata": {
    "cdmi_ctime": "2012-03-20T18:53:46.238543",
    "cdmi_mtime": "2012-03-20T18:53:46.238543",
    "cdmi_mcount": "1",
    "cdmi_owner": "root",
    "cdmi_group": "root",
    "cdmi_acl": [
      {
        "identifier": "OWNER@",
        "acetype": "ALLOW",
        "aceflags": "OBJECT_INHERIT, CONTAINER_INHERIT, INHERITED",
        "acemask": "ALL_PERMS"
```

```
    },
    {
      "identifier": "AUTHENTICATED@",
      "acetype": "ALLOW",
      "aceflags": "OBJECT_INHERIT, CONTAINER_INHERIT, INHERITED",
      "acemask": "READ"
    }
   ],
   "cdmi_size": "16"
  },
  "valuerange": "0-15",
  "valuetransferencoding": "utf-8",
  "value": "Hello CDMI World"
}
* Closing connection #0
```

In the above data object retrieval example, the meaning of the JSON fields in the HTTP response body is listed in Table 1.

JSON Field	**Description**
objectType	Indicates the type of object described in the JSON body. CDMI mimetypes are defined in RFC 6208.
objectID	Every CDMI object has a globally unique identifier that remains constant over the life of the object.
objectName	The name of the object. Present only if the object is stored in a container.
parentURI	The URI of the container where the object is stored. Present only if the object is stored in a container.
parentID	The object ID of the parent container when stored in a container.
domainURI	The URI of a domain object corresponding to the administrative domain that owns the object.
capabilitiesURI	The URI to a capabilities object describing what can be done to the object.
completionStatus	Indicates whether the object is complete or is in the process of being created. This is used for long-running operations.
mimetype	Indicates the mimetype of the value of the data object.
metadata	System and user-provided metadata, in JSON format. Examples of metadata include system properties such as creation time, size, owner, ACLs. Additional user-specified metadata is also stored.

valuerange	Indicates the byte range returned in the value field.
valuetransferencoding	Indicates the encoding of the value field. CDMI supports both UTF-8 and base64 encodings.
value	The data stored in the object.

Table 1: JSON fields returned in an example CDMI Data Object retrieval

Each CDMI object type defines different JSON fields that, in turn, define how objects are set and retrieved, with data objects defined in clause 8, containers defined in clause 9, domains defined in clause 10, queues defined in clause 11, and capabilities defined in clause 12.

Example Client Use Cases

To provide a real-world example, let us suppose that we have been tasked with creating a distributed temperature monitoring system. Our requirements are to sample the temperature of the processor of each of our servers, storing second granular samples every minute to a repository, and providing a Web-based front-end allowing users to visualize temperature across the datacenter.

Using CDMI, a small daemon would be written that runs on each server. This daemon collects 60 samples of data aligned to a minute, and stores it as a CDMI object, including user metadata for the start time, end time, server name, system load average, and processor type.

A JavaScript-based Web page is also served from the CDMI server. When accessed, the JavaScript program is run within the browser and performs a CDMI query based on the user metadata stored in the objects. For example, if a user wishes to see temperature for a given time range, the metadata is used to return only the temperature values within those time ranges. Various visualizations are then generated based on the temperature values returned in the query results.

A second example is a scalable cloud-based OCR system. Multiple scanning workstations scan documents and store them as a data objects. Once scanned, the object ID for each data object is enqueued into a CDMI queue object.

Multiple OCR engines are then instantiated within one or more compute clouds, with the number of instances dynamically varying based on the current size of the CDMI queue. Each OCR engine checks out a scan from the queue, performs OCR processing, and generates a new data object containing a PDF. Based on notifications of the creation of these PDFs, email notifications are then sent to the originator of the scan, or the PDF shows up in the user's home directory.

CDMI Functionality

The following sections provide a survey of the functionality defined by the CDMI standard. To learn more, additional details and examples can be found in the CDMI standard document [3].

Client-to-Cloud Data Transfer

The first area of scope for the CDMI standard is providing standardized methods for clients to exchange data for storage in clouds.

CAPABILITIES DISCOVERY

The CDMI standard mandates that every CDMI server shall provide the ability for clients to discover what optional parts of the standard are implemented in a given server. As the CDMI standard addresses many different cloud-related use cases, allowing an implementer to select the subset of CDMI's functionality specific to their target applications avoids imposing additional development costs for unneeded functionality. For example, a read-only cloud service is free to only implement functionality related to retrieval of stored data, whereas a cloud using CDMI to manage block storage LUNs would only need to implement containers and the ability to define exports.

Clients discover which parts of the CDMI standard are implemented by inspecting published "capabilities." Profiles are also defined to allow clients to determine if logical sets of related capabilities are implemented.

DATA OBJECTS

CDMI data objects are similar to files, and store a value along with metadata. Data objects can be accessed by ID and/or name and support partial retrievals and updates.

CONTAINERS

CDMI container objects are similar to directories and contain named children that can be listed, along with metadata. Containers can be accessed by ID and/or name and support partial listing of children. Traditional hierarchies can be created using sub-containers.

QUEUES

CDMI queue objects are similar to data objects, where multiple values can be stored in a first-in/first-out manner. Queues are typically used to provide persistent inter-process communication structures between distributed programs running in the cloud, and are also used as a foundation for advanced CDMI functionality such as query and notification.

NOTIFICATION

CDMI allows clients to request that when stored objects are created, retrieved, updated, or deleted, notifications of these changes are enqueued into a client-created CDMI queue. Clients can specify the characteristics of the objects for which notifications are generated, based on metadata matching criteria, and can specify which events are of interest and what information is to be returned in each generated notification. Notifications allow powerful workflows to be created, where loosely coupled programs can react to events in the cloud, such as performing transcoding, format conversion, sending notifications, and synchronizing between multiple storage systems.

QUERY

CDMI allows clients to perform a query to find all stored objects that match a set of client-specified metadata matching criteria. Clients can specify which objects are included in the query results and what information from each object is to be included for each query result found. Query allows clients to quickly locate stored

objects, which can be used for further processing or displayed as results to end users.

ACCESS CONTROL

Access to CDMI objects is controlled by ACLs, which define the visibility, read, write, and deletion privileges. The mapping of client credentials to the ACL principal is managed via CDMI domains, which allows content administered by different organizations to co-exist within a single namespace.

Client-to-Cloud Management

The second area of scope for the CDMI standard is providing standardized methods for clients to manage data stored in clouds.

ADMINISTRATIVE DOMAINS

CDMI introduces the concept of Cloud Domains, which permit clients to manage credential to identity mapping (think nsswitch for the cloud) and provide accounting and summary usage information. Domains are hierarchical, which permits tenant and subtenant models, along with delegated administration. Every stored object belongs to a single domain, which controls how the object is accessed and determines who has administrative control over the object.

DATA SYSTEM METADATA

In order to provide a channel that enables clients to express the data services they desire for content stored in the cloud and to give cloud storage system feedback to a client indicating which services are being offered, CDMI introduces a specialized type of metadata known as Data System Metadata (DSM). Instead of providing low-level details about storage, such as RAID level, DSM is expressed in terms of service level objectives, such as desired throughput, latency, and protection.

A client specifies the desired DSM characteristics on individual objects or on containers of objects, which then propagate their DSM to all child containers and data objects. This provides hints about how data should be stored internally within the cloud, allowing a cloud to optimize its internal operations and to charge based on services requested and thus delivered.

The cloud can then create corresponding DSM feedback items, known as "_provided" metadata items, that indicate to a client which actual service the client is receiving. For example, if a client requests three-way replication by setting the "cdmi_data_redundancy" DSM to the value "3", but the system can only provide two-way replication, the "cdmi_data_redundancy_provided" DSM would have the value "2".

A complete list of standardized DSM items can be found in clause 16.4 of the CDMI standard.

RETENTION

CDMI defines a series of DSM that allow restrictions to be placed on stored objects for compliance, eDiscovery, and regulatory purposes. Objects can be placed under retention, meaning they cannot be altered or deleted; can be placed under legal hold

(preventing deletion or modification); and can be automatically deleted when they are no longer under retention periods or any holds.

SNAPSHOTS

CDMI allows clients to trigger the creation of snapshots on a container-based granularity. Snapshots can be accessed through the CDMI interface, and provide read-only access.

EXPORTS

DMI defines the ability to export CDMI containers via standard NAS or SAN protocols. The same approach can be extended to export CDMI namespaces via other cloud protocols, or export queues via other queuing protocols such as AMQP [7]. When combined with cloud computing standards such as OCCI [8] and CIMI [9], CDMI can provide full storage management services for both traditional block and file services accessed by cloud computing resources.

LOGGING

CDMI defines a standardized queue-based mechanism by which clients can receive cloud logging and audit data. This is especially important when a cloud acts as a proxy or broker and logging data must be aggregated or translated. The CDMI standard does not define the contents of log messages originating from clouds.

Cloud-to-Cloud Interactions

The third and final area of scope for the CDMI standard is providing standardized methods for clouds to transfer data with other clouds, both as a result of client requests and automatically.

GLOBALLY UNIQUE IDENTIFIERS

Every CDMI object has a globally unique identifier that remains constant for the life of the object, even as objects are moved or replicated across systems provided by different vendors. This enables location-independent access and allows content to be migrated and replicated without requiring updates to the client's knowledge about how to access the stored data, as the identifier remains constant.

SERIALIZATION/DESERIALIZATION

CDMI objects can be serialized into a JSON format that can be used to transport objects and their metadata between systems. This provides a portable representation for backup and restore, as well as cloud-to-cloud transfer, even if it entails shipping hard drives or tapes storing the data.

CLOUD-TO-CLOUD DATA MOVEMENT

CDMI defines primitives that allow a client to request that a new object be created from an existing object in the same or a different cloud. This allows the destination cloud to retrieve the object directly from the source cloud (using credentials from the CDMI domain, or distributed authentication systems such as OAuth), avoiding the need to transfer the data to and from the client. This also enables clouds to provide transparent proxy and broker functions, while still allowing client access to the underlying clouds themselves.

AUTHENTICATION DELEGATION

CDMI allows resolution of user credentials and mapping to ACL principals to be delegated, allowing CDMI systems to be easily interfaced both with local identity management systems such as AD and LDAP and with emerging federated identity systems.

Future Directions

The SNIA Cloud Technical Working Group encourages interested parties to join the group, participate in plugfests, and submit extensions to the CDMI protocol. Following review by the technical working group, extensions are published for public review. Once two interoperable implementations of an extension are demonstrated at a plugfest, the extension can then be voted on for incorporation into the next version of the CDMI standard.

CDMI extensions currently under public review can be found at the SNIA Web site [10].

References

[1] XAM standard: http://snia.org/forums/xam/technology/specs.

[2] Storage Networking Industry Association: http://www.snia.org/.

[3] CDMI standard: http://snia.org/sites/default/files/CDMI_SNIA_Architecture_v1.0.1.pdf.

[4] Cloud Storage Initiative member companies: http://www.snia.org/forums/csi.

[5] CDMI submission to Swift: https://blueprints.launchpad.net/swift/+spec/cdmi.

[6] REST architecture style: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.

[7] Advanced Message Queuing Protocol: http://www.amqp.org/.

[8] OCCI standard: http://occi-wg.org/about/specification/.

[9] CIMI standard: http://dmtf.org/sites/default/files/standards/documents/DSP0263_1.0.0c.pdf.

[10] Draft CDMI extensions: http://snia.org/tech_activities/publicreview/cdmi.