SNIA DEVELOPER CONFERENCE

SDC 25

By Developers FOR Developers

Hyatt Regency Santa Clara, CA
September 15-17, 2025

Cloud Data Management Interface 3.0

Standardized Management of any URI-accessible Resource

www.sniadeveloper.org

Why we are here:

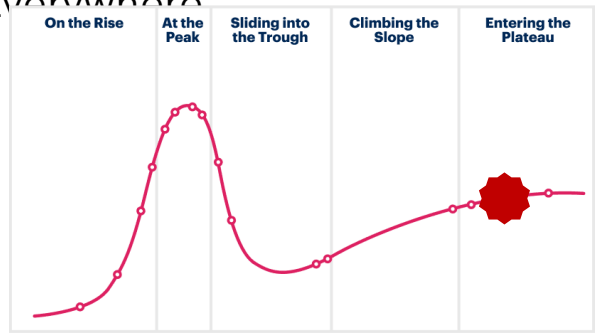
Object Storage Wave

2006: AWS S3

2010: Big Data

2012: Data Lakes

2015: Object Storage Everywhere



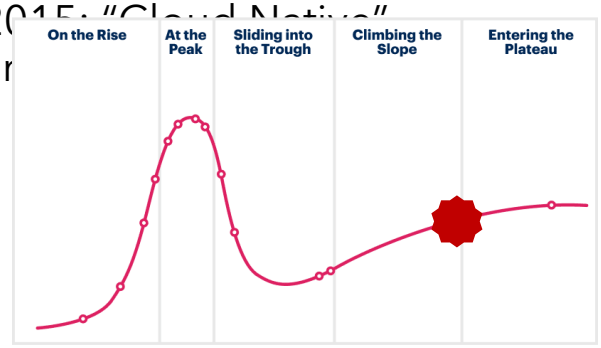
Cloud Wave

2006/2008/2010: AWS/GCP/Azure

2010: Large-Scale Deployments

2012: "Born in the Cloud"

2015: "Cloud Native" ar



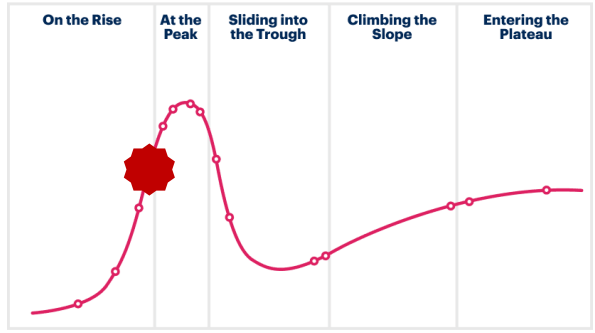
AI Wave

2022: ChatGPT

2024: Training Data Explosion

2025: Inference-in-Everything

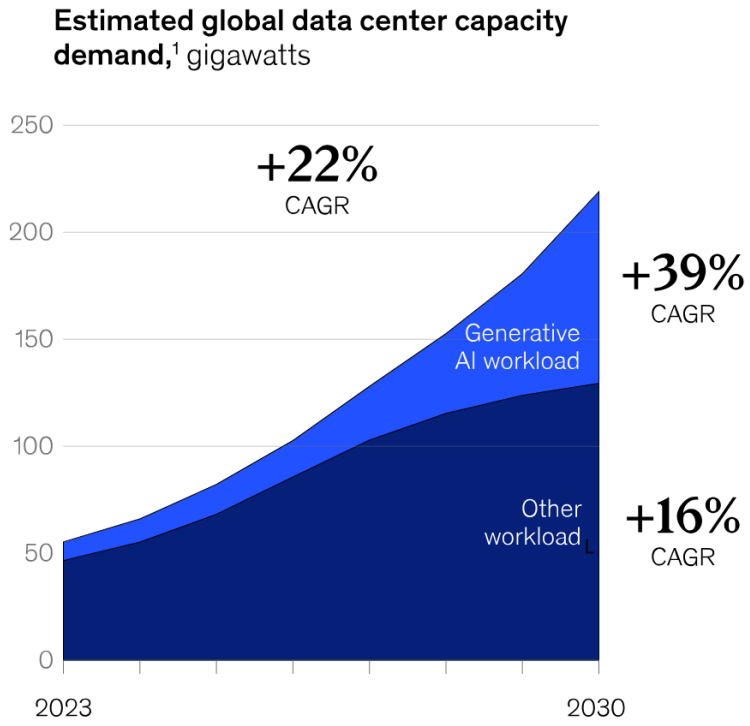
2026: Agentic AI Everywhere



- The Data Explosion continues to accelerate
- Data Management is more important then ever before

New and Emerging Use Cases

- **NO DATA, NO AI:**
  - LLMs require lots of data, and LLMs generate enormous amounts of new data
  - Data required for (and is generated by) AI experiments, AI benchmarking, AI training and AI inference, etc.
  - Next-generation AI technologies (Agentic AI, World Model AI, Physical AI, etc.) require and generate even more data
- All this data must be actively managed:
  - Trend towards "datasets" that are formally managed (emergence of self-describing data assets and data catalogs)
  - Requirements for AI reproducibility and traceability (emergence of self-describing data lineage and provenance)
  - Aspirations for AI data portability across systems and providers (E.g. EU "Dataspaces")
- These new AI systems are using cloud technology stacks (e.g. k8s) and cloud concepts (e.g. API-driven microservices), even when on-prem
- With 85% of AI expected to be hosted by Cloud Service Providers and other Hyperscalers[1] by 2030, data will increasingly move between clouds and on-prem systems

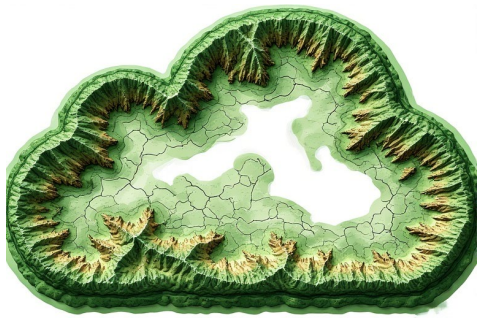


[1] McKinsey 2024 Data Center Demand Model

Why does the Cloud Data Management Interface (CDMI) exist?

1. Many different data access protocols are used in cloud (and on-prem) environments:

Category	Examples
File	NFS, NFS-over-RDMA, SMB, WebDAV, FTP, Google Drive, …
Object	HTTP, S3, S3-over-RDMA, Azure Blob, Google Cloud Storage Object, …
Block	NVMe-oF, iSCSI, FCP, …
Stream	Kafka, AMQT, MQTT, …
Database	SQL, PostgreSQL, GQL, SPARQL, …



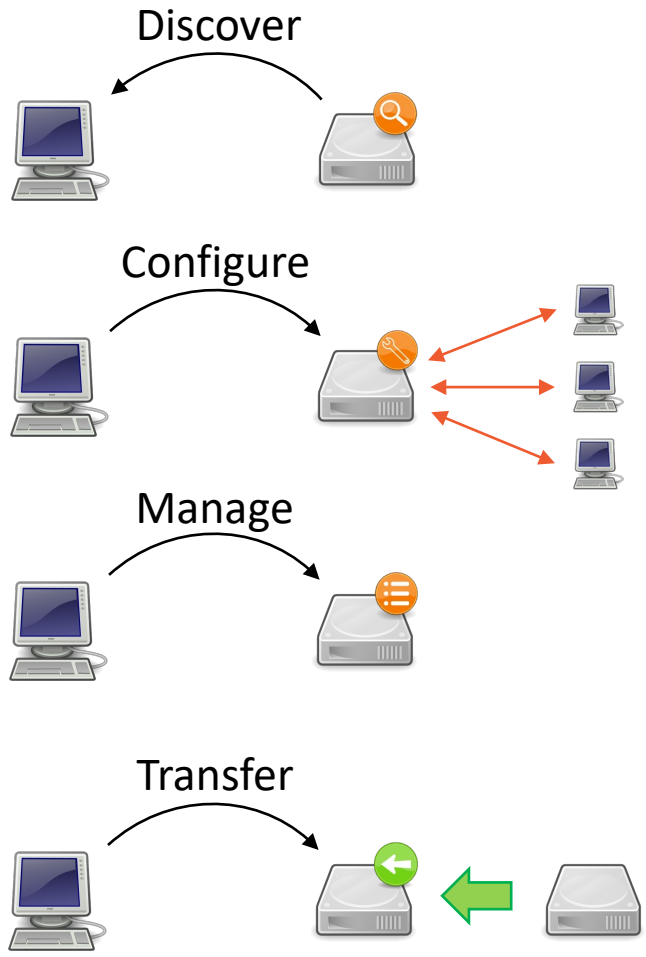
   - **It is not possible to discover data available via these data across protocols from within these protocols**
   - **It is not possible to manage data across protocols from within these protocols**

2. Cloud management protocols are specific to each cloud (AWS, Azure, Google Cloud, etc.)
   - **It is not possible to create, manage and export data in a standardized way**

3. Cloud management protocols assume data stays within a given cloud
   - **It is not possible to move or access data across clouds in a portable way**

What is CDMI's mandate?

- The SNIA Cloud Data Management Interface (CDMI), also known as ISO/IEC 17826, provides a vendor-neutral standard interface for multi-protocol **discovery, configuration, management and portable data movement**.
- Implementers should consider using CDMI when applications require **protocol-independent data management**.
- CDMI allows clients to use a lightweight and consistent management interface **independent of data access protocols**.
- By enabling management operations outside each specific data access protocol, **CDMI works across multiple data access protocols and multiple deployment models**, including on-premises, hybrid, and cloud deployment.



Why is CDMI valuable to developers and users?

- It's an international, formal, unencumbered, and open standard:
  - CDMI 2.0 is an international standard (ISO/IEC 17826)
  - CDMI has a formal specification (https://www.snia.org/sites/default/files/technical-work/cdmi/release/CDMI-v2.0.0.pdf)
  - Contributions from all major storage venders under SNIA's IP policy
  - SNIA does not require licenses, and CDMI is open to anyone to implement
  - SNIA welcomes all interested parties to contribute and participate in the future development of CDMI
- It's easy for developers to implement:
  - Implement only what you need – the only required part of the standard is capabilities (discovering what you implement)
  - Extend as you need – CDMI provides many mechanisms for vendor-specific extensions
  - Vendor extensions welcomed – SNIA welcomes vendor contributions to extend CDMI
  - Works over all existing authentication systems, with no additional development required
  - Works with vendor's existing URI-based data addressing schemes
- It's simple for users:
  - All clients need is curl (for scripts), or the basic HTTP libraries included as part of every programming language

Who has realized value from CDMI?

- Organizations that require long-term ISO/IEC interface stability and backwards compatibility
  - Governments use or mandate CDMI for long-term digital data preservation.
  - Cross-national scientific organizations use CDMI for data interchange and to specify quality of service
- Software developers that require specific features not supported by data access APIs
  - International scientific data management systems use CDMI to enable discovery and management of quality of service
  - Healthcare organizations use CDMI to provide secure access to patient data across untrusted clouds
- Cloud vendors who want to enable data portability between cloud instances
  - Private cloud instances that allow for bulk data movement between clouds
  - Cloud proxies that enable real-time management of data stored in public clouds
- AI data repository operators
  - MCP-based AI dataset discovery, management and access (NEW USE CASE)
  - Discovery and management of RDMA-based data access (NEW US
  - Self-description, portability and movement of AI datasets (NEW US

What are the complaints we heard about CDMI 2.0?

New and existing implementers have a lot to say about how we can improve CDMI:

1. Lack of clarity on the goals of CDMI and how it can be used to accomplish specific tasks

   "It wasn't clear to me why I would use CDMI until I attended this webinar, this didn't come through from reading the spec"

   "I thought CDMI was just an S3 competitor"

   "Too often I'll discover something that CDMI does and wished I knew about it earlier. There's so much functionality in here, but it's often not immediately clear from reading the standard."

2. Lack of formal implementation guidance for managing S3 objects, database tables, query interfaces, stream protocols, graphs and other more "data science"-y interfaces

   "I wish that the specification was more protocol focused, so I could easily ignore all the parts that aren't required"

   "We need guidance to make it easier to add new protocols that can be managed by CDMI"

3. Minor annoyances in an increasingly frictionless software world
   - The formal specification (our strength) is also long and imposing (our weakness)
   - Lack of immediately usable examples (e.g. curl or Python)
   - Some mandates hindered implementers, such as requiring an IANA Enterprise ID to use object identifiers
   - Lack of alignment with newer IETF RFCs

What are the gaps we heard in CDMI 2.0?

Emerging workloads and use cases:

1. Helping find the best way to access data
   - If a given dataset can be accessed in multiple ways (e.g. CIFS, NFS, S3, …), how can this be discovered and managed?
   - If a given data item can be accessed as different representations (formats), how can this be discovered and managed?
   - If a given dataset can be queried, how can this be discovered and managed?
2. Helping find how data is related
   - No standardized way to express graph relationships between datasets, data items, and metadata (important for catalog use cases)
3. Incorporation into AI-driven workflows
   - Support for Model Context Protocol (MCP)
   - Guidance for supporting new and emerging data access protocols (S3-over-RDMA, NVMe-over-UEC, etc.)
   - Managing GPU-initiated data access

Introducing CDMI 3.0

- Based on this feedback, suggestions and contributions, the Cloud Storage Technical Work Group (CSTWG) has spent much of 2025 identifying a scope of work for CDMI 3.0, the next major revision of the Cloud Data Management Interface standard.
- Seven new major improvements will be added in CDMI 3.0:
  1. New MCP representation for CDMI, allowing standardized and open AI-driven dataset management
  2. New guidance for object, query, stream, table and graph data access and query protocols
  3. New "exports_provided" field to enable export discovery
  4. New "rel" field to enable graph relationships
  5. New "representations" data system metadata to handle multi-representational data
  6. The CDMI 3.0 specification will be reorganized into three parts:
     - An informative introduction that focuses on the value of CDMI
     - A normative per-protocol section that allows implementers to focus on the data access protocols that they are interested in
     - A normative per-use-case section that illustrates how CDMI enables the implementation of common management functionality
  7. The CDMI 3.0 specification will be updated to bring it into alignment with newer RFCs
- And more? CDMI is an open standard, bring your challenges and use cases to the Work Group!

Improvement #1 – Adding CDMI over Model Context Protocol (MCP)

- RFC 6208 defines a set of CDMI JSON representations that are transported over HTTP, including:
  - application/cdmi-object – Operations on a data item
  - application/cdmi-container – Operations on a data namespace
  - application/cdmi-queue – Operations on a data queue
- CDMI defines the contents and meanings of these JSON representations

CDMI HTTP Request

```
PATCH /cdmi/myfiles HTTP/1.1
Host: example.com
Content-Type: application/cdmi-container

{
    "snapshot" : "MySnapshot"
}
```

- For example, to take a snapshot of a namespace, the HTTP operation shown on the left would be sent to a CDMI server
- This is read as:
  - For the namespace (container) at the path "/cdmi/myfiles", create a new snapshot with the name "MySnapshot".

Improvement #1 – Adding CDMI over Model Context Protocol (MCP)

- CDMI maps directly to MCP, which provides a JSON-based RPC mechanism for an AI client (the MCP client) to talk to a service provider (the MCP Server)
- The defined CDMI representations map into the MCP "arguments" field, with the method, path and content-type mapping to function, representation and uri fields, respectively.

CDMI HTTP Request

```
PATCH /cdmi/myfiles HTTP/1.1
Host: example.com
Content-Type: application/cdmi-container

{
    "snapshot" : "MySnapshot"
}
```

CDMI MCP Request

```
{
    "jsonrpc": "2.0",
    "id": 4,
    "method": "tools/call",
    "params": {
        "function": "patch",
        "representation": "cdmi-container",
        "uri": "/cdmi/myfiles",
        "arguments": {
            "snapshot" : "MySnapshot"
        }
    }
}
```


Improvement #1 – Adding CDMI over Model Context Protocol (MCP)

- This means that anything that can be done through CDMI can be performed by an MCP client
- For example, an MCP client could request that a new directory be created:

CDMI HTTP Request

```
PUT /cdmi/myfiles/newdir HTTP/1.1
Host: example.com
Content-Type: application/cdmi-container

{
}
```

CDMI MCP Request

```
{
    "jsonrpc": "2.0",
    "id": 4,
    "method": "tools/call",
    "params": {
        "function": "put",
        "representation": "cdmi-container",
        "uri": "/cdmi/myfiles/newdir",
        "arguments": {
        }
    }
}
```


Improvement #1 – Adding CDMI over Model Context Protocol (MCP)

- An MCP client could ask for the size of a file:

CDMI MCP Request

```
{
    "jsonrpc": "2.0",
    "id": 4,
    "method": "tools/call",
    "params": {
        "function": "get",
        "representation": "cdmi-object",
        "uri": "/cdmi/myfiles/file?metadata/cdmi_size",
        "arguments": {
        }
    }
}
```

CDMI MCP Response

```
{
    "jsonrpc": "2.0",
    "id": 4,
    "result": {
        "function": "get",
        "representation": "cdmi-object",
        "uri": "/cdmi/myfiles/file?metadata/cdmi_size",
        "results": {
            "metadata" : {
                "cdmi_size" : "37"
            }
        }
    }
}
```


Improvement #1 – Adding CDMI over Model Context Protocol (MCP)

- An MCP client could ask for a listing of files (or objects, or LUNs, etc.) in a directory:

CDMI MCP Request

```
{
    "jsonrpc": "2.0",
    "id": 4,
    "method": "tools/call",
    "params": {
        "function": "get",
        "representation": "cdmi-container",
        "uri": "/cdmi/myfiles/?children",
        "arguments": {
        }
    }
}
```

CDMI MCP Response

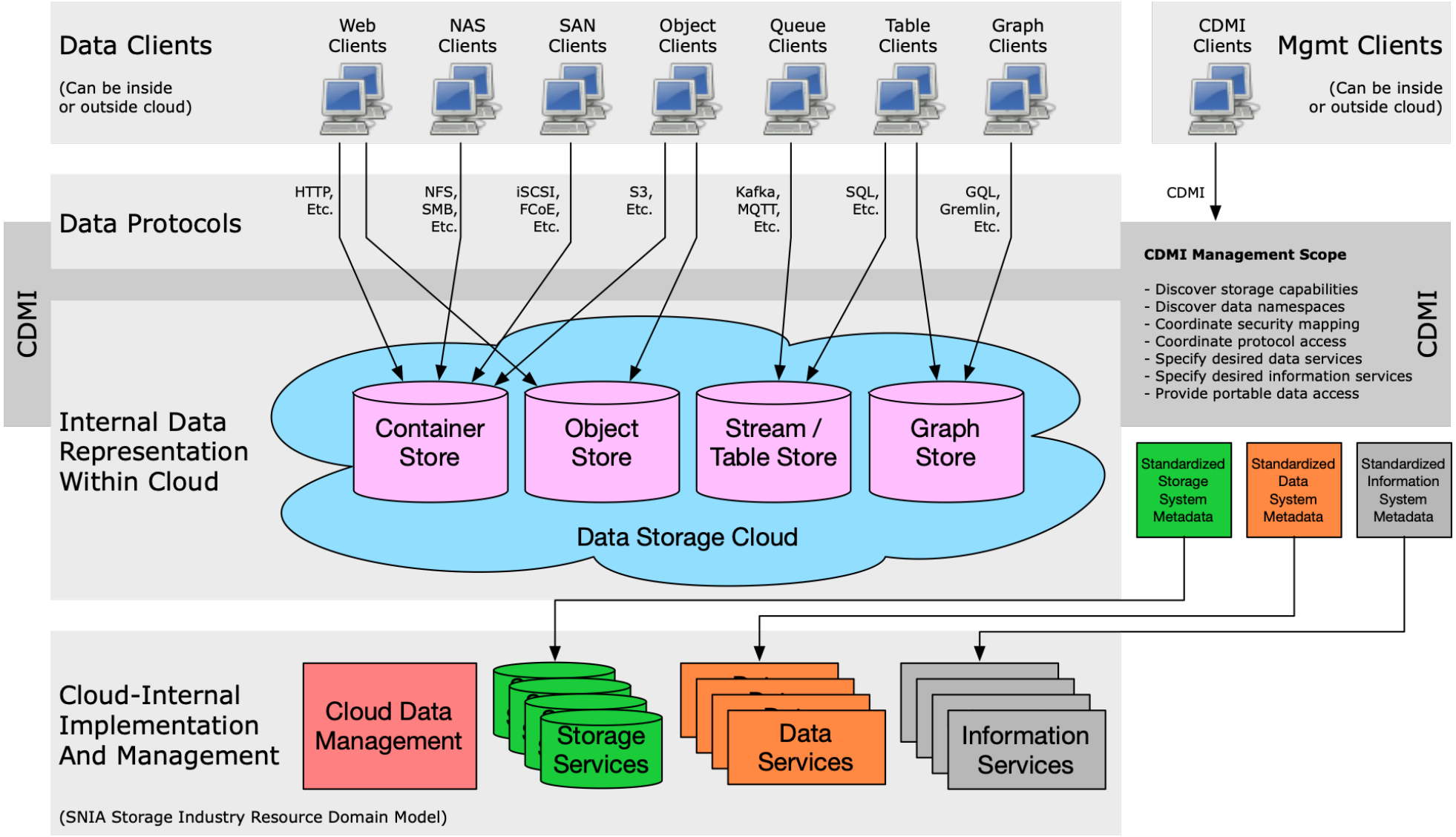
```
{
    "jsonrpc": "2.0",
    "id": 4,
    "result": {
        "function": "get",
        "representation": "cdmi-container",
        "uri": "/cdmi/myfiles/file",
        "results": {
            "children" : [
                "file",
                "data.txt"
            ]
        }
    }
}
```


Improvement #1 – Adding CDMI over Model Context Protocol (MCP)

- Why is this so exciting and powerful? Everything expressible in CDMI can be performed by an agent
- **This means that MCP + CDMI allows for agent-driven discovery, access, management and manipulation of stored cloud data in a standardized way**
- MCP and CDMI are very well aligned:
  - Both CDMI and MCP are JSON-based
  - MCP supports capabilities and tool listing, which correspond to CDMI capabilities
  - MCP supports long-running queries, which correspond to CDMI queues and query queues
  - MCP supports roots, which correspond to CDMI root URIs
  - MCP supports resources, which correspond to CDMI Namespace child listing
- MCP can further enhance CDMI:
  - MCP Prompts allow for a CDMI server to advertise how it can be used
  - MCP tool listing can provide examples of use to an agent, building on foundation model's native understanding of CDMI
  - CDMI can enable discovery and management of MCP servers for data in the cloud

Come get involved in standardizing CDMI over MCP,
and help define the future of AI agent-driven storage management!

Improvement #2 – Add guidance for data access and query protocols



Improvement #2 – Add guidance for data access and query protocols

- CDMI 2.0 already contains examples of exports for common file and block protocols (NFS, SMB, iSCSI, etc.)
  - However, there is not specific guidance for implementers who want to add additional protocol support.
- CDMI 3.0 will also add sections for additional protocols that are widely used in the cloud, or are emerging for AI use cases:
  - File: NFS-over-RDMA
  - Block: NVMe-over-UEC
  - Object: S3, S3-over-RDMA
  - Stream: Kafka, MQTT
  - Query: SQL, GQL
- CDMI 3.0 will also add guidance for implementers who want to add additional protocols.

Come get involved in standardizing data access protocol discovery and provisioning, and help connect data to AI!

Improvement #3 – Add export discovery (“exports_provided”) field

- One common use case of CDMI is to determine which protocols can be used to access a given namespace
- In CDMI 2.0, client needs to look in multiple places to identify which “exports” apply to a given namespace
- By adding an “exports-provided”, following the same pattern used for data system metadata, this process can be simplified
- The “exports-provided” field can also indicate additional configuration information not specified in the “exports” fields

```
{
    "exports-provided": {
        "nfsv4": {
            "client_uri": "nfs://example/docs/",
            "client_protocol": "NFSv4.1",
            "client_protocol_transport": "TCP/IP",
            "client_flags": [ "hard", "port=2049", "sec=krb5p" ],
            "export_definition_uri": "/shares/docs/"
        },
        "smb": {
            "client_uri": "smb://example/docs/",
            "client_protocol": "SMB 3.1",
            "client_protocol_transport": "TCP/IP",
            "client_flags": [ "vers=3.1.1", "multiuser", "sec=krb5", "mfsymlinks" ],
            "export_definition_uri": "/shares/docs/"
        },
        "s3": {
            "client_uri": "https://docs.example.com/",
            "client_protocol": "S3",
            "client_protocol_transport": "TCP/IP",
            "client_flags": [ ],
            "export_definition_uri": "/shares/docs/"
        }
    }
}
```

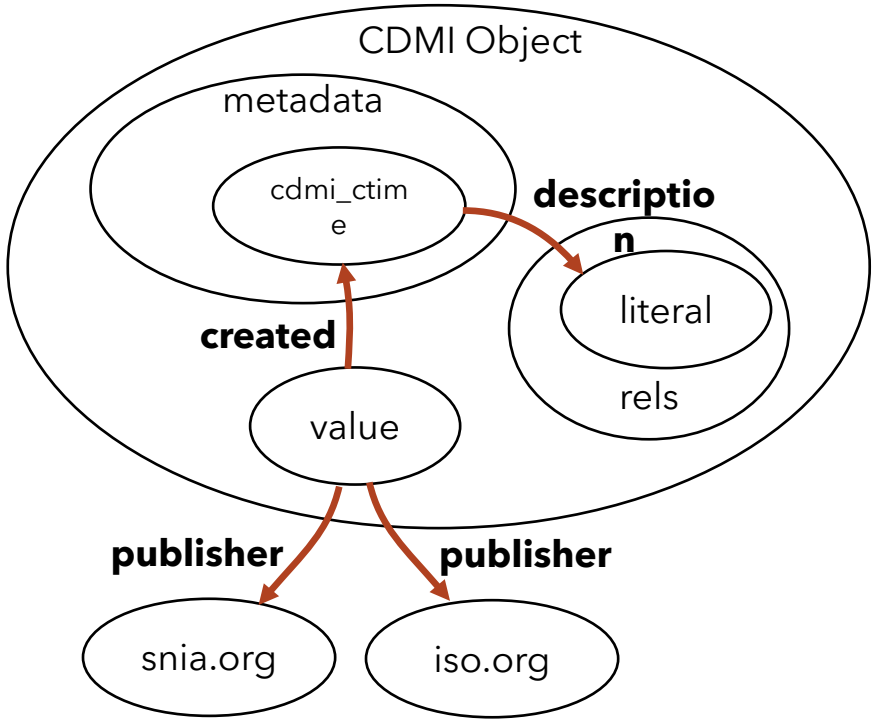

Improvement #4 – Add relationships (“rel”) field

- When managing data, it is often valuable to specify relationships between data, metadata and external systems
- In CDMI 2.0, there is no standardized way to specify relationships. Metadata can be used, but is application-dependent.
- By adding a new “rel” field that builds on W3 RDF, arbitrary typed knowledge graph relationships can be added to stored namespaces and data items
- Graph relationships can link within, across and outside these objects

```
{
    "objectName": "CDMI 3.0 Standard.pdf",
    "rel": {
        ".#value": {
            "http://purl.org/dc/terms/publisher": [ {
                "value": "https://snia.org/",
                "type": "uri"
            }, {
                "value": "https://iso.org/",
                "type": "uri"
            } ],
            "http://purl.org/dc/terms/created": [ {
                "value": ".#metadata/cdmi_ctime",
                "type": "uri"
            } ]
        },
        ".#metadata/cdmi_ctime": {
            "http://purl.org/dc/terms/description": [ {
                "value": "The creation time of the standard",
                "type": "literal"
            } ]
        }
    }
}
```


Improvement #4 – Add relationships (“rel”) field

- This creates the following graph:




```
{
    "objectName": "CDMI 3.0 Standard.pdf",
    "rel": {
        ".#value": {
            "http://purl.org/dc/terms/publisher": [ {
                "value": "https://snia.org/",
                "type": "uri"
            }, {
                "value": "https://iso.org/",
                "type": "uri"
            } ],
            "http://purl.org/dc/terms/created": [ {
                "value": ".#metadata/cdmi_ctime",
                "type": "uri"
            } ]
        },
        ".#metadata/cdmi_ctime": {
            "http://purl.org/dc/terms/description": [ {
                "value": "The creation time of the standard",
                "type": "literal"
            } ]
        }
    }
}
```

- And because it’s CDMI, these graphs can be accessed by AI agents using MCP

Improvement #5 – Add “representations” data system metadata

- Some types of stored data has multiple representations
  - For example, an image may have a JPEG representation and a PNG representation
- Adding a “representations_provided” metadata item allows clients to discover which representations a storage system is capable of providing.
- The means of selecting a representation is data access protocol specific
  - E.g. in HTTP, you would use the “Accept” header to specify “image/jpg” or “image/png” media types.

```
{
    "metadata": {
        "cdmi_representations_provided": [
            "image/png",
            "image/jpg"
        ],
        "cdmi_representations": [
            "image/png"
        ]
    }
}
```


Improvement #6 – Reorganizing the Specification

CDMI 2.0 Specification

Table of Contents

I CDMI Preamble
Clause 1: Scope
Clause 2: Normative references
Clause 3: Terms, acronyms, and definitions
Clause 4: Conventions
Clause 5: Overview of Cloud Storage

II Basic Cloud Storage
Clause 6: Data Object Resource Operations using HTTP
Clause 7: Container Object Resource Operations using HTTP

III CDMI Core
Clause 8: Data Object Resource Operations using CDMI
Clause 9: Container Object Resource Operations using CDMI

IV CDMI Advanced
Clause 10: Domain object resource operations using CDMI
Clause 11: Queue object resource operations using CDMI
Clause 12: Capability object resource operations using CDMI
Clause 13: Exported Protocols
Clause 14: CDMI Snapshots
Clause 15: Serialization/deserialization
Clause 16: Metadata
Clause 17: Access Control
Clause 18: Retention and Hold Management
Clause 19: Scope specification
Clause 20: Results specification
Clause 21: Notification queues
Clause 22: Query queues
Clause 23: Encrypted objects
Clause 24: Delegated access control
Clause 25: Data object versions

CDMI 3.0 Specification

Contents

Improvement #7 – RFC Alignment

- RFC 6901 - JavaScript Object Notation (JSON) Pointer
  - This RFC defines a standard mechanism to refer to values within a JSON document
  - Specifying the use of this RFC replaces several places in the CDMI specification where similar path-style references are custom-defined, and provides additional clarity and consistency for implementers
- RFC 6902 - JavaScript Object Notation (JSON) Patch
  - This RFC defines a mechanism to make partial changes to JSON objects (such as CDMI metadata), without having to retrieve and replace the entire JSON object
  - Adding this RFC as an option for CDMI PATCH operations improves client flexibility, reduces unnecessary data transport over the wire, and provides additional clarity and formalism for update behaviors. Adding JSON Patch also eliminates possible data loss when there are concurrent updates.
- RFC 7396 - JSON Merge Patch
  - This RFC defines how JSON request bodies provided as part of a PATCH operation should be merged into an existing JSON document
  - Specifying the use of this RFC provides additional clarity and consistency for implementers
- RFC 8615 - Well-Known Uniform Resource Identifiers (URIs)
  - This RFC defines a well-known URI location for service discovery (and a registry for namespace management)
  - Providing guidance for how to place CDMI capabilities within RFC 8515 ".well-known" URIs reduces client bootstrapping problems
- RFC 9457 - Problem Details for HTTP APIs
  - This RFC defines a JSON-based response body for HTTP-based APIs that indicates when problems have occurred.
  - Having a standard way to consistently indicate problems with CDMI requests, such as incorrectly formed requests, objects not present, permission errors, etc., to the client improves interoperability and compatibility, as it allows for clients to reliably branch on specific errors

Call to Action

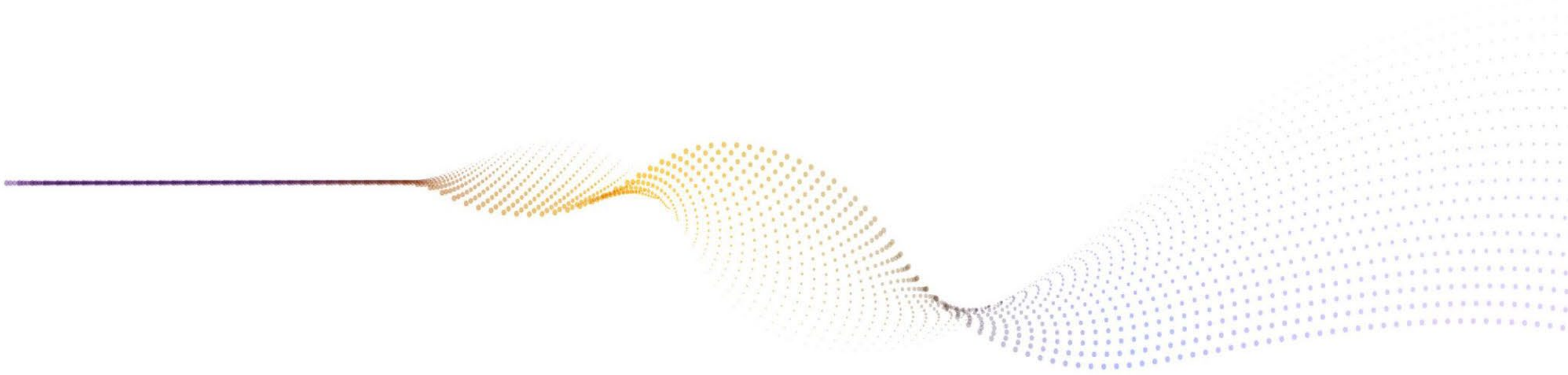
- Interested in AI, agents and storage management?
  - Join us at the Cloud Storage Technical Work Group to help standardize storage management using MCP
- Interested in data access protocol discovery and specification?
  - Join us to collaborate on standardizing protocol exports
- Interested in AI cluster accelerated access to data?
  - Join us to collaborate on discovery of RDMA and scale-up/scale-out data access methods
- Interested in knowledge graph annotations of data?
  - Join us to collaborate on adding graph relationships to CDMI

How to Join and Contribute

- Three steps to join:
  1. If your company is not a member of SNIA, you will need to first join SNIA:
     - https://www.snia.org/member_com
  2. Once your company is a member, if you do not have an account, you will need to create one:
     - https://members.snia.org/user/register
  3. Once you have an account, you need to join the Cloud Storage Technical Work Group (TWG):
     - https://members.snia.org/workgroup/index

- The TWG meets every week on Friday mornings.
- Time is split between new work items (contributor-initiated) and CDMI 3.0 work items

Let’s Work Together to Connect Clouds, Data and AI



Thank you for attending!

Please remember to rate this session. You get access the presentations at
http://sniadeveloper.org/conference

