Lessons Learned Implementing Cross-Protocol Compatibility for Object Storage

Scott Horan

Senior Integration Engineer

Cleversafe, Inc.

Agenda

- Overview and Motivations
- Feature Comparison Matrix
- Software Architecture for Multi-Interface Support
- Key Cross-Protocol Compatibility Drivers
  - Authentication
  - Metadata
  - ACLs
  - Versioning
  - Large Object Segmentation
  - Containers and Provisioning
- Summary and Conclusions

Overview

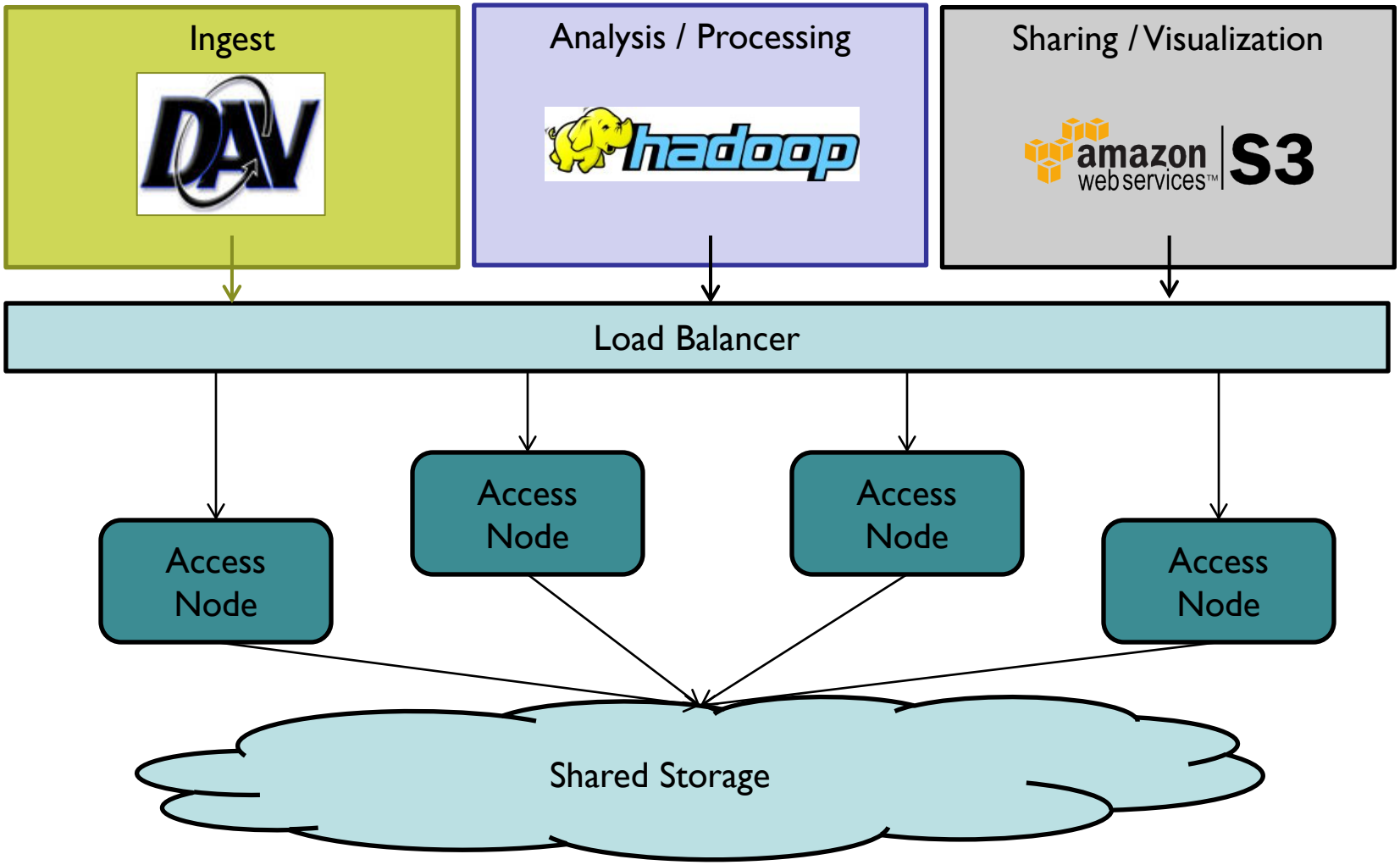
- Many API alternatives exist for accessing object storage
- Over the past year, we have implemented "compatibility APIs" for a number of cloud and object storage APIs
  - Amazon S3, OpenStack Swift, WebDAV, HDFS, CDMI
- On the surface, many similarities exist
- However, differences exist that pose challenges for implementers

Motivations

- Why is cross-protocol support important?
  - Larger potential customer base
  - Compatibility with third-party applications and tools
  - Support of legacy applications
  - Onramp from traditional file system based storage to object storage

Organizations derive value from shared access to stored data;
Enabling cross-protocol access to shared data streamlines workflows and increases data utility

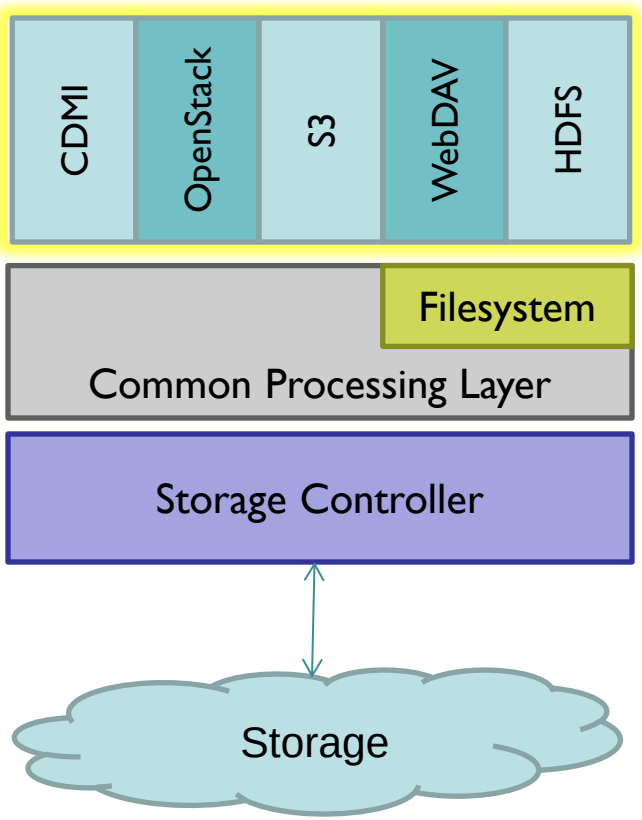
Multi-protocol Access: A simple use case



Feature Comparison Matrix

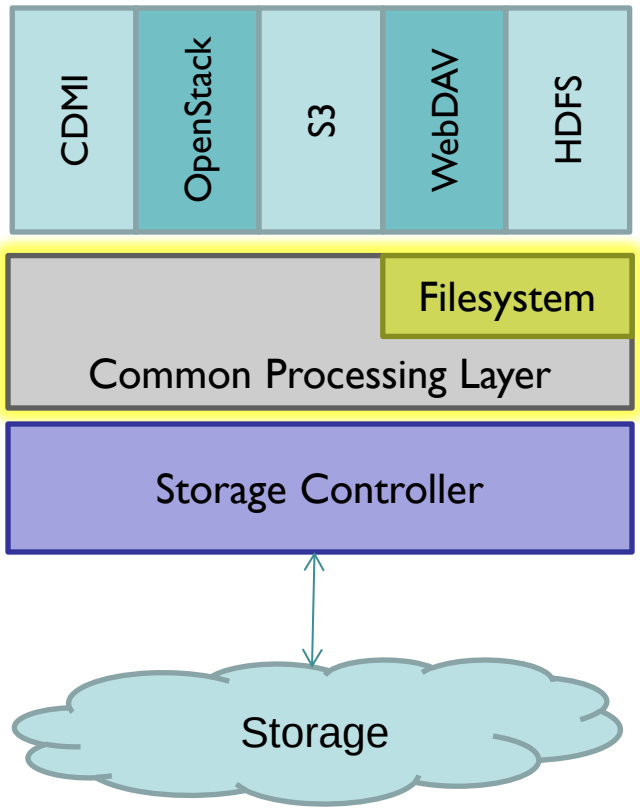
Interface	S3	OpenStack Swift	CDMI	HDFS	WebDAV
Object Read	Supported	Supported	Supported	Supported	Supported
Object Write	Supported	Supported	Supported	Supported	Supported
Object Delete	Supported	Supported	Supported	Supported	Supported
Metadata Operations	Object Metadata Bucket Metadata	Object Metadata Container Metadata	Object Metadata Container Metadata	File/Directory Attributes	PROPPATCH PROPFIND
Authentication	Keyed HMAC-based AWS Auth	Token-based Auth	Traditional HTTP Auth Mechanisms	Host OS Process Owner	Traditional HTTP Auth Mechanisms
ACLs	Bucket & Object ACLs Bucket Policy IAM Policy	Container ACLs (Read/Write only)	Metadata Property "cdmi_acl"	POSIX File/Dir Permissions	ACL: RFC3744
Versioning	Enabled per Bucket Each unique write assigned a revisionId	Enabled per Container X-Versions-Location header specifies target for previous versions	*CDMI Extension	N/A	DeltaV: RFC3253
Large Object Support	Multipart Upload	Dynamic Large Object Static Large Object	HTTP Range, X-CDMI-Partial Headers	N/A	N/A
Provisioning	Create Bucket Delete Bucket	Create Container Delete Container	Create Container Delete Container	Create Directory Delete Directory	MKCOL Ext. MKCOL: RFC5689

Software Architecture for Multi-Interface Support



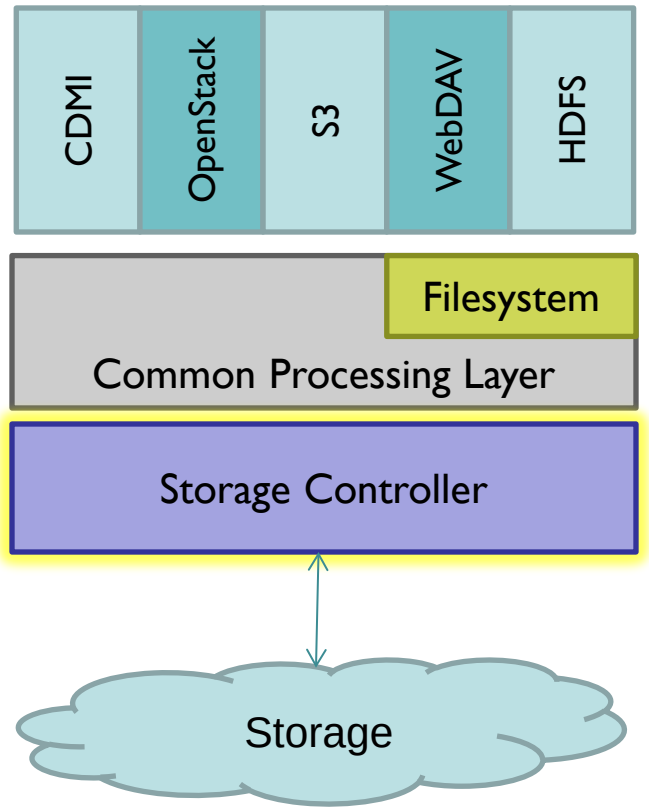
- Protocol-specific servlets implement interface application logic
  - Request parsing
  - Input validation
  - Response Serialization

Software Architecture for Multi-Interface Support



- Common Processing Layer
  - Bridge between interface specific processing and back-end storage interface
  - Performs common HTTP request processing
    - Conditional Request Evaluation
    - Ranged Read Processing
  - Agnostic to interface/protocol type used in request

Software Architecture for Multi-Interface Support



- Storage Controller
  - Interface to backing storage

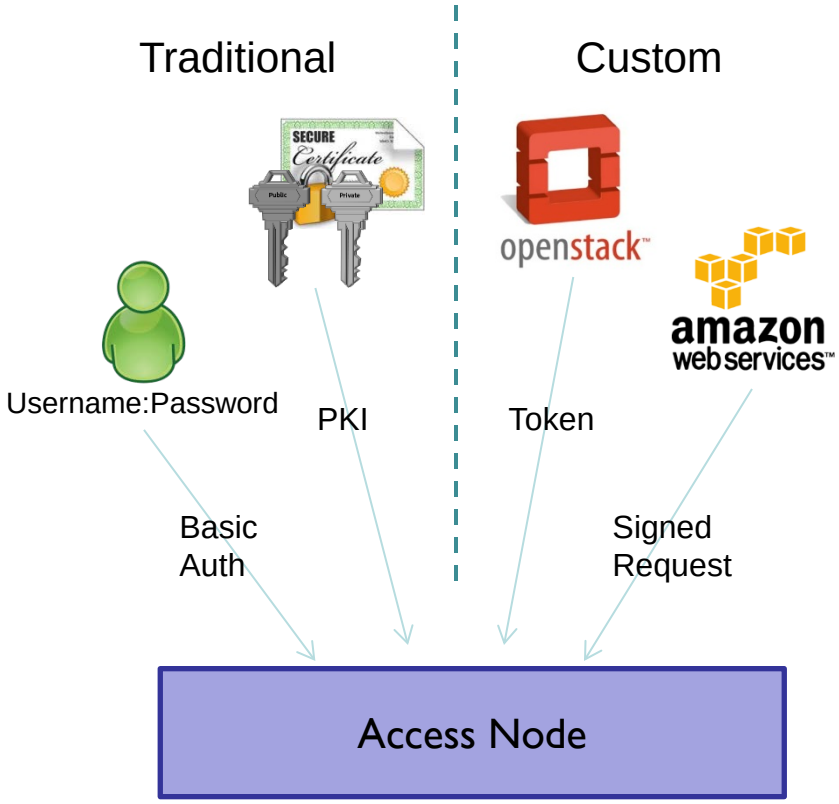
REST API endpoints

- Goal: Make protocol handling as transparent as possible
- Assign each protocol to a separate path spec
  - /cdmi/* -> CDMI endpoint
  - /s3/* -> S3 endpoint
  - /swift/* -> OpenStack Swift endpoint
  - /webdav/* -> WebDAV
  - /xyz/*
  - /* -> “Default” protocol

Compatibility Tip: Allow for a configurable “default” protocol to be assigned to the web service root

Authentication

- HTTP supports several authentication mechanisms
  - Basic Authentication, Kerberos, PKI
- Many Object Storage APIs define custom authentication methods
  - Amazon S3: Custom Keyed-HMAC based HTTP Authentication
  - OpenStack Swift: Token-based Authentication



Compatibility Tip: Allow users to authenticate via primary authentication mechanism or fall back to traditional Basic Auth / Kerberos / PKI

Identity Service

- Each user entity is identified by a unique ID
- One or more credentials can be associated with each user identity
- Identity service maintains and enforces a core set of permissions and access levels associated with each user identity
  - This is translated into a protocol-specific grant/permission/ACL when queried through the storage API

Identity Mapping

PKI

Certificate Chain	CN=…

openstack™

Token	65lkjca#a5hj5d5f…
Username	User1
Password	*********

Basic Authentication

Username	user@domain
Password	*********

amazon web services™

AWS Access Key ID	9a0aa9d0b8a19cea7f60
Secret Access Key	jcWAkKL8ztxEAUpqAAUC 9ImX5gp+FNvXztyAU8NC

Identity Service

Account Identifier	077ab320-151b-48d0-8d85-17205f1df5ae
Credentials	'{"aws": "9a0aa9d0b8a19cea7f60", "active_directory": "user@domain", "pki": "CN…", "openstack": "User1"}'

Exposing Metadata Across Interfaces

- Container/Object metadata comes in various types
  - Resource attributes
    - Content-Type, Size
  - Storage-assigned attributes
    - OID, Storage Location
  - User-defined metadata
    - Typically key-value pair, but can be more complex
  - Interface-specific information
    - Revision Identifier, Part Number, Delete Markers
  - ACLs

Example: Container Metadata

Openstack: GET Container Metadata

```
HTTP/1.1 204 No Content
Date: Wed, 04 Sep 2013 16:24:00 GMT
Server: Cleversafe
Accept-Ranges: bytes
X-Container-Object-Count: 0
X-Container-Bytes-Used: 0
X-Container-Meta-customer-id: 12345
X-Container-Meta-description: This is a container
```

S3: GET Bucket Tagging

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Tagging xmlns="http://s3.amazonaws.com/doc/2006-03-
01/">
    <TagSet>
        <Tag>
            <Key>customer-id</Key>
            <Value>12345</Value>
        </Tag>
        <Tag>
            <Key>description</Key>
            <Value>This is a container</Value>
        </Tag>
    </TagSet>
</Tagging>
```

WebDAV: Container PROPFIND

```
<?xml version="1.0" encoding="UTF-8"?>
<D:multistatus xmlns:D="DAV:">
  <D:response>
    <D:href>/webdav/v2/</D:href>
    <D:propstat>
      <D:prop>
        <D:supportedlock/>
        <D:lockdiscovery/>
        <D:getlastmodified>Wed, 04 Sep 2013 16:23:48
UTC</D:getlastmodified>
        <D:creationdate/>
        <D:displayname/>
        <customer-id>12345</customer-id>
        <description>This is a container</description>
        <D:resourcetype>
          <D:collection/>
        </D:resourcetype>
      </D:prop>
      <D:status>HTTP/1.1 200 OK</D:status>
    </D:propstat>
  </D:response>
</D:multistatus>
```

Metadata Broker

- Translation often required for some metadata types when accessing between interfaces
  - ACLs
  - Timestamps
- Raw data stored as resource metadata
- Metadata broker defines mapping/translation to be performed for specific interface

Metadata Broker Example: Timestamps

openstack™

Last-Modified	Fri, 12 Jun 2010 13:40:18 GMT

DAV

```
<D:propstat>
      <D:prop xmlns:R="http://ns.example.com/">
      <D:creationdate>2013-06-12T13:40:18+00:00</D:creationdate>
…
</D:propstat>
```

Metadata Broker

Timestamps	create_time	1378403269
	last_modified	1378415883
	last_changed	1378415883
	last_accessed	1378421143
ACLs	7b76af69-40d0-4749-9a28-3ed67bb43272	{WRITE_OBJECT, READ_OBJECT, READ_ATTRIBUTES, WRITE_ATTRIBUTES,…}
	34198946-1ba9-460a-834c-8b23537d2177	{READ_ACL, WRITE_ACL, WRITE_OWNER,…}

Metadata Broker Example: ACLs

SNIA

"cdmi_acl"	WRITE_OBJECT, READ_OBJECT, READ_ATTRIBUTES, WRITE_ATTRIBUTES

amazon web services™

Grantee	<Grantee> <ID>75aa57f09aa0c</ID> <DisplayName>user1@amazon.com</DisplayName> </Grantee>
Permission	FULL_CONTROL

Metadata Broker

Timestamps	create_time	1378403269
	last_modified	1378415883
	last_changed	1378415883
	last_accessed	1378421143
ACLs	7b76af69-40d0-4749-9a28-3ed67bb43272	{WRITE_OBJECT, READ_OBJECT, READ_ATTRIBUTES, WRITE_ATTRIBUTES,…}
	34198946-1ba9-460a-834c-8b23537d2177	{READ_ACL, WRITE_ACL, WRITE_OWNER,…}

Object Versioning

- Motivations
  - Versioning allows multiple revisions of an object to be preserved
  - Object overwrite causes new revision
- Compatibility Issues:
  - Naming convention and representation of revisions
  - Visibility of previous revisions in object listings

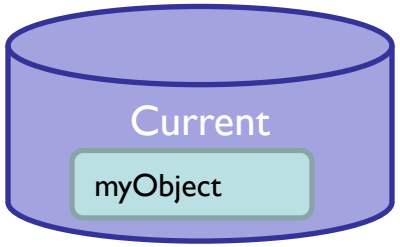
Object Versioning

amazon web services™

openstack™

Create Object:

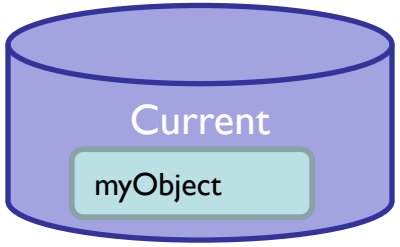
myObject
Rev: 001

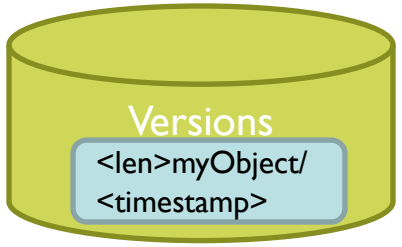


Overwrite Object:

myObject
Rev: 002

myObject
Rev: 001



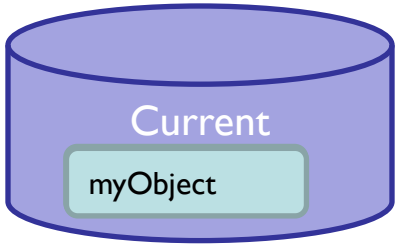


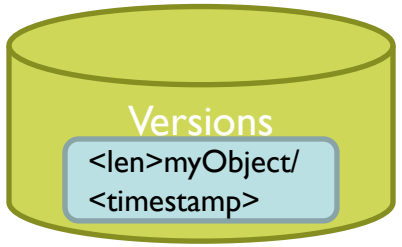
Delete Object:

myObject
Rev: 003

myObject
Rev: 002

myObject
Rev: 001



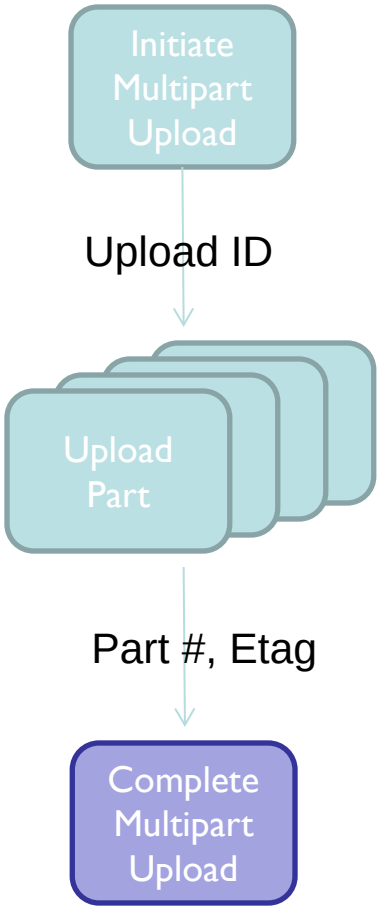


Large-Object Segmentation

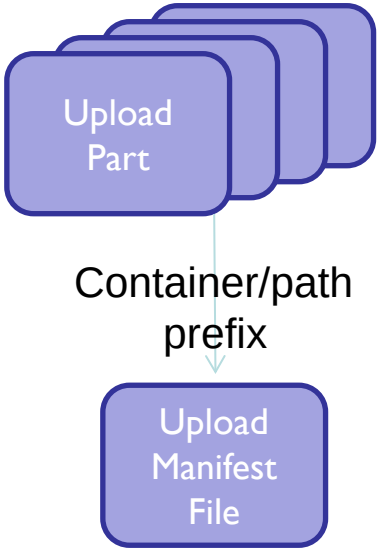
- Motivations
  - Support object sizes that exceed the maximum size limitations for individual objects
  - Parallelization of single object upload
  - Focused retry for failed parts
- Compatibility Issues:
  - Visibility of completed parts
  - Listing of in progress transactions
  - Dynamic inclusion of constituent parts

Large Object Segmentation

S3 Multipart Upload



Openstack Static/Dynamic Large Object Support

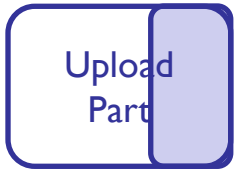


Static: Parts are fixed
Dynamic: Parts can be dynamically inserted/removed

CDMI Large Object Support

X-cdmi-partial: false

X-cdmi-partial: false



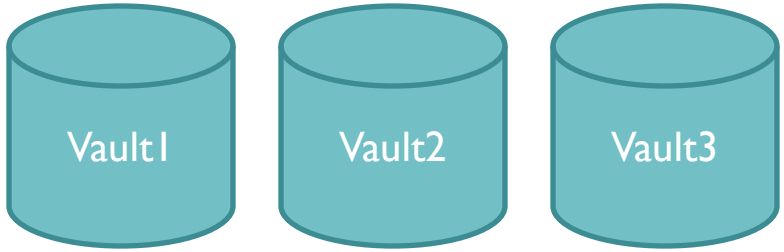
X-cdmi-partial: true

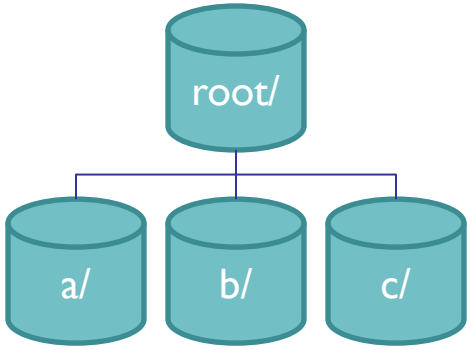
Container Provisioning

- Motivations
  - Allow creation and deletion of storage containers through object storage API
- Compatibility Issues:
  - Differing definitions of “container”
  - Enforcement of naming restrictions/uniqueness
  - Container configuration parameters
    - location constraint, reliability/availability parameters, etc.

Container Definition

- Container resides at account/service root
  - Example: Amazon S3, OpenStack Swift
- Containers/sub-containers organized in hierarchy
  - Similar to directories in file system
  - Often represented as object resource with specific naming convention (trailing "/")
  - Example: CDMI, WebDAV





Example: Container Provisioning

Openstack: Create Container

```
PUT /v1/<account>/ContainerName HTTP/1.1
Host: cloud.example.com
X-Auth-Token: eaaafd18-0fed-4b3a-81b4-663c99ec1cbb
X-Container-Meta-customer-id: 0ea39f84c41
```

S3: PUT Bucket

```
PUT / HTTP/1.1
Host: ContainerName.s3.amazonaws.com
Content-Length: length
Date: date

<CreateBucketConfiguration
xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
   <LocationConstraint>Region</LocationConstraint>
</CreateBucketConfiguration>
```

Containers reside at service/account root

CDMI: Create Container

```
PUT /webdisc/xfiles/ HTTP/1.1
Host: cloud.example.com
```

WebDAV: Make Collection (MKCOL)

```
MKCOL /webdisc/xfiles/ HTTP/1.1
Host: cloud.example.com
```

Allows nested, heirarchical sub-collections (similar to filesystem directory structure)

Conclusion

- Cross-protocol compatibility adds value for customers
  - Enables a broader set of use cases
  - Streamlines application workflows
- Important to design storage systems with compatibility in mind
  - Develop architecture that supports accessing resources across multiple interfaces
    - Identity Service, Metadata Broker
- Not all storage APIs are created equal
  - Divergent behavior across related features can hinder compatibility

Questions?