Object Storage Metadata Search (OSMS)

OpenStack Summit Design Session, Hong Kong

Lincoln.Thomas@hp.com, IRC lincolnt

v4, 7-Nov-2013

Design Session Objectives

Frame the discussion around a new metadata search capability for Object Storage

Requirements

- Rich API feature set for complex searches
- Capture metadata changes with minimal performance cost
- Timely search results at scale (1 billion objects)

What's there now

- Swift API provides metadata retrieval and lists
- SoftLayer's "API Operations for Search Services"
- DIY external metadata databases

Proposed OSMS API

- A completed first pass
- Strawman for OpenStack community discussion
- HP-specific implementation on top of OpenStack started, but changes expected

Possible implementations

- Extend existing SQLite DB to include object metadata & more
- Add new metadata store in Swift
- Inline capture or scan later
- Vendor-specific (e.g. HP), on top of OpenStack

End Goal: Define and document the OSMS API, and develop a native reference implementation

Requirements / What's There Now

OpenStack Object Storage API v1 Reference

http://docs.openstack.org/api/openstack-object-storage/1.0/content/

Stub blueprint for metadata search

https://blueprints.launchpad.net/swift/+spec/searchable-metadata

SoftLayer API Operations for Search Services

http://sldn.softlayer.com/article/API-Operations-Search-Services

Nothing more to say in slides. Let's go on!

Proposed OSMS API

High Level Features, 1 of 9

HTTP REST API request defines:

1. Search API version
2. Base URI to search (the scope)
3. Search criteria
4. Metadata fields to return, for the items matching the criteria
   - Term “item” here = an account, container, or an object
5. Filter parameters
6. Output format parameters

Next slides describe each point…

Proposed OSMS API

High Level Features, 2 of 9

Complete HTTP Syntax

```
GET /<Object Storage API version>
[/<account>[/<container>[/<object>]]]                    Scope
?<OSMS API version>
[&attributes=<attr1>[,<attr2>][,…]]                      Attributes to return
[&query=[(]<query expr1>[ [AND|OR] <query expr2>][)][[AND|OR]…]]
[&limit=<max records>]                                   Filter params
[&all_results]
[&marker='<marker string>']
[&end_marker='<end marker string>']
[&offset=<start record>]
[&prefix='<prefix string>']
[&path='<object directory path>']
[&delimiter='<delimiter character>']
[&sorted[=<attr1>[,<attr2>,…]]]                          Output format params
[&format=[json|xml]] HTTP/1.1
```

Proposed OSMS API

High Level Features, 3 of 9

1. Search API Version

- Extends base Swift API
- Does not affect existing base request handling.
- Search API version independent of base API version, e.g.:

```
GET /<Object Storage API version>[/<account>[/<container>[/<object>]]]
?<OSMS API version>/...
```

2. Scope can identify:

- An object (unlikely to be used): `/a1/c2/o3`
- A container (and all of its objects): `/a1/c2`
- An account (and all of its containers and objects): `/a1`
- The entire object store (across all accounts): *`{no URL}`*

Proposed OSMS API

High Level Features, 4 of 9

3. Search criteria

`[&query=[(]<query expr1>[ [AND|OR] <query expr2>][)][ [AND|OR]…]]`

…where:

`query expr=<query attr><operator><query value>`

…and `<operator>` is one of:

`=, !=, <, <=, >, >=, ~` (regexp match, strings only), `!~`

- Note: Generating SQL query for complex Boolean expressions across multiple tables may be… complex! (If using SQL)
- Simpler: support only ANDs (multiple ANDs OK)
- Services request (see slide 16) shows whether simple (ANDs only) or complex supported

Proposed OSMS API

High Level Features, 5 of 9

Searching across accounts

1. The engine must check the access to each account and container to be searched, and will only return results for the accessible ones. This access checking can significantly impact the performance of the search.

"Authorized Searchers" feature

2. Maximize search speed by bypassing access check for a set of users defined by the storage administrator. *(How to manage user list? Keystone role? What if not using Keystone?)*
3. These users must authenticate (e.g. Keystone), just like any user.
4. Once authenticated, however, the authorized searcher can access any metadata for all accounts, containers, and objects via the API.
5. Optional feature. If not used, works as described in #1.
6. Services request (see slide 16) shows whether feature is supported or not.

Proposed OSMS API

High Level Features, 6 of 9

Search live and recently deleted items

- Allows clients to track recent change history, including deletes
- Useful for apps e.g. backup, analytics (e.g. HP Autonomy)
- When item deleted, keep only its delete time metadata (e.g. separate `DeleteTime` table)
- Clean up deleted items' metadata entries periodically

4. List of attributes to return

- Existing and many new system attributes, e.g. `account_last_activity_time`, `container_read_permissions`, `object_content_type`
- Custom attributes, e.g. `container_meta_<name>`
- Superset attributes for search request convenience, e.g.:

  `all_attrs`, `all_account_meta_attrs`, `all_object_system_attrs`

- See API spec for complete list and details

Proposed OSMS API

High Level Features, 7 of 9 (born Annika Hansen)

5. Filter parameters, 1 of 2

`[&limit=<max records>]`
If absent, =10,000 (matches base API)
`[&all_results]`
Allows >10,000 results, overrides base API's hard limit of 10,000.

`[&marker='<marker string>']`
`[&end_marker='<end marker string>']`
Same as base API

`[&offset=<start record>]`
Not in base API. Present in SoftLayer's API.
No need to parse last record of each page to find `marker` for next page.

Proposed OSMS API

High Level Features, 8 of 9

5. Filter parameters, 2 of 2

```
[&prefix='<prefix string>']
[&delimiter='<delimiter character>']
```

Same as base API

```
[&path='<object directory path>']
```

Same as base API, but seems redundant with prefix + delimiter.

Proposed OSMS API

High Level Features, 9 of 9

6. Output format parameters

`[&sorted[=<attr1> [ASC|DESC][,<attr2> [ASC|DESC],…]]]`

Sort by attr1 (ascending or descending order), then by attr2 for all results with the same attr2, etc.

`[&format=[json|xml|txt]]`

Output formatted JSON, XML, or plain text (default = text)

Proposed OSMS API

Example output

Plain text

```
/account1/container1
    container_last_modified_time:2013-07-23T13:17:55.435654031Z
/account1/container1/objectdir1/subdir1/photo.jpg
    object_last_changed_time:2012-12-02T00:53:29.849922518Z
    object_content_length:194532
```

Proposed OSMS API

Example output

JSON

```
[
  {
    "/account1/container1" :
    {
      "container_last_modified_time" : "2013-07-23T13:17:55.435654031Z"
    }
  },
  {
    "/account1/container1/objectdir1/subdir1/photo.jpg" :
    {
      "object_last_changed_time" : "2012-12-02T00:53:29.849922518Z",
      "object_content_length" : 194532
    }
  }
]
```

Proposed OSMS API

Example searches

Get all metadata for all accounts, containers, and objects

```
curl -g "http://99.226.50.92/v1?v1&attributes=all_attrs"
```

Get metadata for an account, and objects meeting a set of criteria

```
curl -g
"http://99.226.50.92/v1/acc1?v1&attributes=account_object_count,ob
ject_last_changed_time&query=container_create_time>2013-08-01 AND
object_manifest_type=1 and object_manifest~'segctr1/.*'"
```

Proposed OSMS API

Services request

Client asks: What does your implementation support?

```
GET /services HTTP/1.1
```

Example response (JSON), 1 of 3

```
[
  {
    "min_base_api_version" : "v1"
  },
  {
    "max_base_api_version" : "v1"
  },
  {
    "search_provider" : "HP"
  },
  {
    "search_enabled" : "true"
  },
```

(reference impl = “OpenStack”?)

(“false” = none of metadata search provided)

Proposed OSMS API

Services request

Example response (JSON), 2 of 3

```
{
  "min_search_api_version" : "v2.1"
},
{
  "max_search_api_version" : "v2.1"
},
{
  "freshness_complete" : "true"
},
{
  "freshness_partial" : ""
},
{
  "complex_boolean_expr" : "false"
},
```

Proposed OSMS API

Services request

Example response (JSON), 3 of 3

```
   {
     "attributes" :
     {
       {
         "attr_name" : "account_uri"
       },
       {
         "data_type" : "string"
       },
       {
         "sortable" : "true"
       }
     }, ...
   }
]
```

Not covered yet

- Aggregation parameters
  - E.g. SQL concepts such SUM, COUNT, …
- Python API library, other language bindings, or CLI
  - Prefer OpenStackClient
    - https://wiki.openstack.org/wiki/OpenStackClient
  - Now including Swift work
  - Keystone v3 depending on it
- Integration with existing apps that need to keep up with object store changes
  - Backup, e.g. HP Data Protector, Symantec NetBackup, …
  - Analytics, e.g. HP Autonomy, …

Possible Implementations

Vendor-specific add-on

HP's proof of concept so far

- Add python middleware to catch metadata changes in HTTP requests
- Store changes in metadata database outside Swift (asynchronously, eventually consistent)
- Treat Swift as a black box
- Convert searches into queries against database

- Pros
  - No changes to Swift

- Cons
  - No API standard
  - No OpenStack reference implementation
  - Vendor-specific implementation required

Possible Implementations

Search what's there

Search account/container DB and file system objects

- Convert account and container search parameters into DB queries
  - SQLite or substitute e.g. MySQL per http://docs.openstack.org/trunk/openstack-ops/content/example_architecture.html
- Convert object search parameters into "grep"-like searches of xattrs in objects on filesystem

- Pros
  - No additional storage for metadata
  - No additional development for ingest side

- Cons
  - Non-trivial conversion of API requests to queries and file searches
  - Performance drops dramatically as object counts rise
  - Many new attributes not captured, would be blank and non-searchable (could be added)

Possible Implementations

Extend SQLite database

Add new acct/ctr metadata and object metadata to DB schema

- Convert all search parameters into DB queries (SQLite or substitute)
- Update all metadata on HTTP requests. Inline (fresh) or asynchronously (eventually consistent)?

- Pros
  - No additional database for metadata
  - All searches are queries, no object file access

- Cons
  - Dramatically increases storage in DB tables
  - Slows existing DB accesses?
  - Is it scalable to millions / billions of objects?

Possible Implementations

Add new metadata store

Create new database or KV store, purely for metadata search

- Convert all search parameters into queries against metadata DB or KV store
- Update all metadata on HTTP requests. Inline (fresh) or asynchronously (eventually consistent)?

- Pros
  - Does not affect current container/object DB usage
  - Ref impl DB could be substituted like existing SQLite DB can be

- Cons
  - Still requires potentially large storage space
  - Is it scalable to millions / billions of objects?

Possible Implementations

Provide pluggable architecture to vendor-specific metadata stores

Hybrid of vendor-specific add-on and native OpenStack implementation

- Provide simple (simplistic?) reference implementation (SQLite, other…)
- Provide hooks/callbacks to vendor-specific implementations
  - For storing metadata changes
  - For processing search requests

- Pros
  - OpenStack-standard API
  - No vendor-specific implementation required but easily pluggable
- Cons
  - Defined API constrains vendors' creative implementations?
    - Even with "Services" request that defines vendor-specified variations?

Discussion / Q&A

Do we want to do this? Now?
Decent requirements? More specific?
API changes?
    Syntax, attribute list, parameter list
Possible implementations, what others?

Next Steps

Who starts what, when and how?
Incubation? Wiki? github? Mailing list?
Where to post these slides and spec?
How to gather feedback and update spec?

Thank you