LAKEsuperior Recommendations

Stefano Cossu, the Art Institute of Chicago scossu@artic.edu Kevin Ford, the Art Institute of Chicago, kfordı@rtic.edu

Contents

General Remarks

Purpose

`lakesuperior` is the code name of the current internal Fedora repository that is part of LAKE, the institutional repository and DAMS for the Art Institute of Chicago Collections.

In order to overcome the current technical limitations of the current Modeshape-backed Fedora 4 implementation, we want to propose an alternative implementation of Fedora.

End goal of this project is a repository system primarily, but not exclusively, supported and used by AIC, fully compliant to the official Fedora API specifications at [http://fedora.info].

The full-scale project is divided into three levels, corresponding to three major milestones:

- Level 1: Proof of Concept(PoC): a product satisfying a minimum set of requirements for the sole purpose of evaluating feasibility and performance and to compare with existing alternative Fedora implementation efforts.
- Level 2: Minimum Viable Product (MVP): a production-ready Fedora implementation that satisfies the minimum requirements to interoperate with the current Hyrax version (at the time of completion), in addition to a small subset of features aimed at significant improvements in the Hydra/Fedora layer. The implementation of this level will be further subdivided into sub-projects: alpha, beta, release candidate and stable release. The main goal for this milestone is to provide a drop-in Fedora replacement that offers clear, quantifiable advantages over the Modeshape implementation and provides documented tools to migrate data from fedora 4 in order to gain support from other Hydra and Islandora adopters. Slight modification to the client may be tolerable depending on stakeholders' opinion.
- Level 3: Production-quality implementation.
- Level 4: Fedora-compliant implementation: a repository implementation that aligns with the official Fedora specification and passes all expected compatibility tests. This is pending the finalization of such specifications. The implementation of additional non-core specifications (which may be considered core by AIC or other stakeholders) is also to be determined based use cases and on the final status of such specification sets.

General requirements

Our Fedora implementation should fulfill the following high-level requirements:

- Scale vertically and horizontally
- Support a large volume of data and perform efficient graph traversals
- Implement a query language guarantee the maximum stability of content
- Consist of a thinlayer of custom code on top of systems with as many required features as possible provided out of the box
- Transparently store binary files (either in file system or in a dedicated store)

Scope of this document

This is in no way an exhaustive path to development and less than ever a spec sheet. It is mostly a collection of recommended patterns to resolve specific issues for building an alternative implementation of Fedora. Some of these recommendations may differ from the final implementation.

Level 1 Recommendations

Scope

Level 1 implementation is aimed at providing a proof-of-concept (PoC) repository system with a minimal set of features. The resulting system should be easily deployable and offer most commonly used features and features that present particular challenges in the current Fedora implementation.

The Level 1 development cycle should delivery a system that can be tested by a variety of stakeholders with the purpose of gathering feedback to drive further development and possible collaboration with other Fedora 4 alternative implementation efforts.

Compatibility

The Level 1 LAKEsuperior implementation supports a minimum set of features meeting the following requirements:

- Supports all the principal operations performed by Sufia 7.2 and, specifically, LAKEshore
- Supports all operations performed by Combine
- If a specific feature is particularly onerous to implement for Level 1 but is required by the above clients, it can be "mocked" for expediency's sake based on how critical its use is, and documented as incomplete

Technology Map

Graph Store

Blazegraph is the most convenient choice for a PoC.

Pros:

- familiarity of AIC with the software
- Ease of translation between Fedora API and back end
- Supports transactions (to be verified)
- Transparent import/export for migration (triples in, triples out)
- Easy translation of LDP data exchange format (RDF) and query language (SPARQL)

Cons:

- Does not store binaries
- No Python API (must use REST API)
- No new releases since August 2016 (although development is active)
- Need to purchase enterprise version for distributed setup

Alternatives can be considered beyond Level 1 if significant hurdles or limitations are encountered during this development phase. Therefore, the code should use standard triplestore features and be explicitly document the implementation-specific features necessary for LAKEsuperior to function.

Binary Store

Binaries are stored in the filesystem. See below for implementation details.

Application Layer

The application is written in Python 3 using the Python standard library and mature, well-respected 3rd party libraries: Flask, Requests, RDFLib, etc.

Application configuration is stored in separate files in an appropriate structured format (e.g. YAML).

Methodologies

Partitioning strategy

Partitioning within a quad-store can be done via named graphs. Each "resource" as intended in LDP is contained in a separate named graph. This allows to clearly identify such LDP "resources".

There are multiple approaches to this implementation, mostly differing in the handling of the graph name:

1. Graph name same as subject

```
[prefix definitions]
res:16fb6c41-b862-4adc-8656-5f9c356b56bb {
  res:16fb6c41-b862-4adc-8656-5f9c356b56bb a ldp:NonRdfSource ;
  ebucore:height "1024" ;
  ebucore:width "1920" ;
  ebucore:filename "xyz.png" ;
  premis:hasMessageDigest <urn:sha1:eaa3379d8415071369f3c8b3699fa91fcfc6888c> .
}
```

This is the most straightforward implementation and does not impose any additional construct on a straight SPARQL query. On the other hand, the repetition of the graph name in the subject is redundant and limits further strategic use of named graphs.

2. Named graphs as abstractions and proxies

[LEVEL 2?]

```
[prefix definitions]
main {
  res:a a ltype:Resource ;
        fcrepo:hasVersion res:b , res:c ;
   res:b a ltype:Snapshot .
   res:c a ltype:Snapshot .
}

res:a {
```

```
  lake:16fb6c41-b862-4adc-8656-5f9c356b56bb
    a ldp:Container , aictype:Work ;
    skos:prefLabel "Composition in Red" ;
}

res:b {
  lake:16fb6c41-b862-4adc-8656-5f9c356b56bb
    a ldp:Container , aictype:Work ;
    skos:prefLabel "Composition in Reed" ;
}

res:c {
  lake:16fb6c41-b862-4adc-8656-5f9c356b56bb
    a ldp:Container , aictype:Work ;
    skos:prefLabel "Composition in Read" ;
}
```

This other approach has a "main" graph holding metadata about the first-class resources: their provenance, relationships with other named graphs, etc. Multiple named graphs can be different representations of the same resource. This makes it easier to build provenance data, such as version snapshots. On the other hand it is a more indirect approach, in that LAKEsuperior has to find one or more named graphs that represent the resource that is being requested. In the example above, the repository application needs to get triples from the graph that has the `ltype:Resource` and skip the ones from the `ltype:Snapshot` graphs.

Graph store to LDP mapping

This is done by the application layer. This implementation supports all necessary building blocks for interacting with ActiveFedora: LDP-NR, LDPC, LDP-DC, LDP-IC, etc.

Resources are stored internally with a prefixed namespace that is replaced by the domain-specific URI prefix. No host-specific information is stored in the triplestore in order to guarantee portability of the data set.

LDP containers

LAKEsuperior supports all the LDP container types and their related behavior, especially with regard to direct and indirect containers.

Deletion strategy

Individual triples

For individually deleted triples (i.e. property values) versioning can be engaged.

Resources

Allowing the deletion of a resource altogether poses several challenges (see also "Referential integrity" below). Non-destructive deletion and optional admin-restricted purging is the recommended approach.

Two "soft-delete" approaches are possible:

1. A resource is marked as "deleted" by a special status, either internally managed or transparently communicated to the client.
2. A new version of the resource is created with an empty graph. This allow to enforce referential integrity and history tracking.

Tombstone

Deleted resources (with either method) should leave a tombstone. If the tombstone is deleted, the resource may be deleted permanently (barring referential integrity issues).

A tombstone, while available, may offer method to "resurrect" a deleted resource.

Hierarchies

Currently several clients rely on the mechanism by which, given `<main:a>` and `<main:a/b>`, if `<main:a>` is deleted, `<main:a/b>` is also deleted. A tombstone should be left on `<main:a>` and surface if either `<main:a>` or `<main:a/b>` is requested, as per current fcrepo4 behavior [VERIFY].

Explicit definition of LAKE resources

[DISCUSS utility, extent and pitfalls] Since any triples can be inserted in a triplestore, it may be useful to explicitly identify a LAKEsuperior "resource" by adding e.g. a `<http://definitions.artic.edu/lake/type#Resource>` RDF type to all the resources directly managed and exposed by LAKEsuperior. This RDF type is server managed and not exposed in the LDP API. The presence of this type for a resource may trigger further validation of server-managed properties and other constraints.

The designated LAKE resource class should be included in each named graph for each LAKEsuperior resource appearing as a subject.

While it is not ideal to enforce low-level structural rules in a repository, experience has proven that when systems become out of sync or external clients behave in an unexpected way, a repository without any structural integrity rules becomes more easily corrupted and structural issues become harder to identify and fix. Therefore it may be valuable to place some basic restrictions for resources in the repository so that errors surface early and are more likely to be caught and repaired.

Bitstream storage

Files are stored in the filesystem. The filesystem path for each LDP-NR is obtained by a root prefix defined in the application configuration and a balanced pairtree created from the file SHA1 checksum of the file content. This means that identical binaries can be represented by multiple LDP resources but are stored under the same file behind the scenes.

A server-managed triple contains the path to a file for each version.

Sample data set:

```
[prefix definitions]

area:main {
  res:16fb6c41-b862-4adc-8656-5f9c356b56bb
    a ldp:NonRdfSource , ltype:Resource ;
    ebucore:height "1024" ;
    ebucore:width "1920" ;
    ebucore:filename "xyz.png" ;
    premis:hasMessageDigest <urn:sha1:eaa3379d8415071369f3c8b3699fa91fcfc6888c> ;
    fcr:content "/eaa3/379d8/4150/eaa3379d8415071369f3c8b3699fa91fcfc6888c" ;
    aic:status res:58f00eca-c398-02f0-f9bb-6b2b6105c0ef .
}
```

Note that `fcr:content` is redundant since it can be inferred by the `premis:hasMessageDigest` URI.

The binary file content is stored in `<bitstream folder path>/eaa3/379d8/4150/eaa3379d8415071369f3c8b3699fa91f`

Sample LDP requests and responses:

```
GET http://lakesuperior.artic.edu/rest/main:16fb6c41-b862-4adc-8656-5f9c356b56bb

[binary data]

GET http://lakesuperior.artic.edu/rest/main:16fb6c41-b862-4adc-8656-5f9c356b56bb/fcr:metadata

[prefix definitions]

<http://lakesuperior.artic.edu/rest/main:16fb6c41-b862-4adc-8656-5f9c356b56bb>
  a ldp:NonRdfSource ;
  ebucore:height "1024" ;
  ebucore:width "1920" ;
  ebucore:filename "xyz.png" ;
  premis:hasMessageDigest <urn:sha1:eaa3379d8415071369f3c8b3699fa91fcfc6888c> ;
  iana:describedBy <http://lakesuperior.artic.edu/rest/main:16fb6c41-b862-4adc-8656-5f9c356b56bb
  aic:status res:58f00eca-c398-02f0-f9bb-6b2b6105c0ef .
}
```

Server-managed properties

Server-managed properties should be supported the same way the current Fedora 4.x implementation does, to the extent that allows a drop-in replacement as a Samvera client. This includes "magic" LDP predicates for direct and indirect containers.

Internal use only properties

Some server-managed properties are not exposed in the client-facing API. These properties have predicates within a dedicated namespace. If a client tries to insert a triple including an internal use only predicate, the application should return a `409 Conflict` response.

Some internal use only predicates are:

- content (reference to the filesystem path of the content of a binary resource)
- Versions
- […]

Level 2 Recommendations

Scope

The Level 2 implementation builds upon the Level 1 proof of concept and, ideally, possible feedback from a variety of testers.

The goal for Level 2 is a feature-complete, beta-quality product compatible with Hyrax and Islandora (pending the presence of Islandora stakeholders)

Structural restrictions

Types of restrictions

Structural restrictions can include:

- Cardinality of properties
- Uniqueness of property values across the repository
- Domain and range of properties

Enforcing these three types of restrictions may satisfy a very broad number of use cases.

Enforcing restrictions

Restrictions should be completely optional for implementers.

The subsystem responsible for enforcing restrictions should be close enough to the core repository to ensure that all interaction with the persistence layer passes through it; and isolated enough that it can always be configured and enabled or disabled separately.

Referential integrity

Referential integrity should be ideally enforced within resources which share the same domain, i.e. are managed by the same repository.

Referential integrity is hard to maintain if the repository allows deletion of versioned resources. Consider two resources, `<a>` and `<b>`:

1. Client creates a relationship: `<a> ns:rel <b>`
2. Client deletes the relationship. This is no longer present in the current version of `<a>`, but it is still present in historical versions.
3. `<b>` is deleted. The referential integrity is broken in previous versions of `<a>`.
4. If `<a>` is restored to its previous version, this breakage surfaces in the main resource.

There are three possible ways to address this:

1. Not supporting referential integrity at all;

2. Not supporting deletion of resources and replacing it with a "deleted" status (see above); 3. Only supporting deletion of a resource if this has no inbound links from any resource in any version (this can trigger a costly query in a large store).

Actual referential integrity is a better candidate for Level 2 but some early decisions should be made with this setup in mind.

Versioning

Previous versions of a resource are stored as sets of triples within a separate named graph:

```
PREFIX area: <http://definitions.artic.edu/lake/area#>
PREFIX res: <http://definitions.artic.edu/lake/resource#>
PREFIX ltype: <http://definitions.artic.edu/lake/type#>
PREFIX snap: <http://definitions.artic.edu/lake/snapshot#>
PREFIX aic: <http://definitions.artic.edu/ontology/1.0/>

area:main {
  res:16fb6c41-b862-4adc-8656-5f9c356b56bb
    a ldp:NonRdfSource , ltype:Resource ;
    ebucore:height "1024" ;
    ebucore:width "1920" ;
    ebucore:filename "xyz.png" ;
    premis:hasMessageDigest <urn:sha1:eaa3379d8415071369f3c8b3699fa91fcfc6888c> ;
    aic:status res:58f00eca-c398-02f0-f9bb-6b2b6105c0ef ;
}

area:historic {
  res:16fb6c41-b862-4adc-8656-5f9c356b56bb
    fedora:hasVersion
      snap:fa297320-4ae1-46c9-8d2b-9356e713489f ,
      snap:7c96c502-b101-410d-8119-3b690a38a46a .

  snap:fa297320-4ae1-46c9-8d2b-9356e713489f
    a ldp:NonRdfSource , ltype:Resource ;
    fedora:hasVersionLabel "Version 2" ;
    ebucore:height "1018" ;
    ebucore:width "1318" ;
    ebucore:filename "xyz_older.png" ;
    premis:hasMessageDigest <urn:sha1:dae1230ad61117e33f4e338a1648983e1af84377> ;
    fedora:created "2017-08-22T14:21:14.941Z"^^xsd:\#dateTime ;
    aic:status res:58f00eca-c398-02f0-f9bb-6b2b6105c0ef .

snap:7c96c502-b101-410d-8119-3b690a38a46a
   a ldp:NonRdfSource , ltype:Resource ;
   fedora:hasVersionLabel "Version 1" ;
   ebucore:height "768" ;
   ebucore:width "1024" ;
```

```
   ebucore:filename "xyz_oldest.jpg" ;
   premis:hasMessageDigest <urn:sha1:dae9d4b279aa27cde354e5a0a8e7da07c7560fbc> ;
   fedora:created "2017-08-22T14:51:46.036Z"^^xsd:\#dateTime ;
   aic:status res:58f00eca-c398-02f0-f9bb-6b2b6105c0ef .
}
```

Versions are retrieved by a mechanism similar to fcrepo4, with some differences:

```
GET http://lakesuperior.artic.edu/rest/historic:16fb6c41-b862-4adc-8656-5f9c356b56bb

[prefix definitions]

<http://lakesuperior.artic.edu/rest/historic:16fb6c41-b862-4adc-8656-5f9c356b56bb>
  fedora:hasVersion
    <http://lakesuperior.artic.edu/rest/fa297320-4ae1-46c9-8d2b-9356e713489f> ,
    <http://lakesuperior.artic.edu/rest/7c96c502-b101-410d-8119-3b690a38a46a> .

<http://lakesuperior.artic.edu/rest/historic:fa297320-4ae1-46c9-8d2b-9356e713489f>
  fedora:created "2017-08-22T14:21:14.941Z"^^xsd:\#dateTime ;
  fedora:hasVersionLabel "Version 2" .

<http://lakesuperior.artic.edu/rest/historic:7c96c502-b101-410d-8119-3b690a38a46a>
  fedora:created "2017-08-22T14:51:46.036Z"^^xsd:\#dateTime ;
  fedora:hasVersionLabel "Version 1" .
```

Most importantly note the lack of `fedora:hasVersions` property that points to the `./fcr:versions` resource. This is replaced by a resource with the same URI as the main resource, within the `historic` namespace. If a client requires the `fedora:hasVersions` property for whatever reason, it may be reasonable to add it and have it point to the resource in the `historic` area.

Transactions

Transactions are not supported by LAKEsuperior at the LDP API level, at least in Levels 1 and 2, and maybe not in early stable releases either. The pitfalls of implementing transactions are too many to tackle in early releases.

However, Level 1 should guarantee that a single client request is always atomic even if this entails multiple interactions with the underlying store. For this reason, any candidate backing triplestore must support transactions.

Fixity

[TODO] Fixity support should be implemented as per current fcrepo4 specs.

Level 3 Recommendations

TBD