APPLIED
LINKED DATA

Steven Anderson, Boston Public Library
Slides: https://goo.gl/BP2inf
Workshop Helpers: Corey Harper and David Lacy
Vagrant: https://github.com/scande3/hydra-vagrant

GETTING SETUP

Further Setup

Setup Vagrant:

> Download Vagrant from:
https://www.vagrantup.com/downloads.html
> sudo apt-get install VirtualBox

> git clone
https://github.com/scande3/hydra-vagrant.git
> cd hydra-vagrant
> vagrant up

Ensure VagrantFile is 4GB RAM:

```
> vim VagrantFile
> v.memory = 2048 -> v.memory = 4096
> esc, :w, :q
> vagrant halt
> vagrant up
```

Get an older LOC authorities file:

(Newest June version seems to lack narrower skos concept relationships...?)

```
> cd <vagrant_home>/downloads
> wget static.digitalcommonwealth.org/samples/authoritiessubjects.nt.skos.zip
> cd ..
```

Setting up Blazegraph:

```
> vagrant ssh
> cd ldata
> vim Gemfile
> gem 'ldfwrapper', github:
'boston-library/ldf-wrapper', branch: "master"
> esc, :w, :q
> bundle install
> rake ldfjetty:install
> rake ldfjetty:start
```

Server should be accessible at: http://localhost:8988/
(Go to Blazegraph -> Explore -> search for
“http://id.loc.gov/authorities/subjects/sh2002006395”)

Blazegraph Without Jetty:

Blazegraph download site: https://www.blazegraph.com/

For a tomcat 7 deploy:

- Rename to blazegraph.war
- Deploy to /var/lib/tomcat7/webapps (or equivalent webapps directory)
- Set appropriate permissions so tomcat can access the war.

Populate Blazegraph with LCSH:

```
> vagrant ssh
> cd /vagrant/downloads
> unzip authoritiessubjects.nt.skos.zip
> curl -H 'Content-Type: text/turtle' --upload-file
/vagrant/downloads/subjects-skos-20140306.nt -X POST
"http://localhost:8988/blazegraph/sparql?context-uri=au
thoritiessubjects.nt.skos.zip"
```

(Try Blazegraph -> Explore -> search for "http://id.loc.gov/authorities/subjects/sh2002006395" once again to see it is now populated with LOC)

FCREPO_WRAPPER:

```
> vagrant ssh
> cd ldata
> fcrepo_wrapper -p 8984 -i /vagrant/downloads

Service should be accessible at: http://localhost:8984
```

SOLR_WRAPPER:

```
> vagrant ssh
> cd ldata
> solr_wrapper -p 8983
```

Service should be accessible at: http://localhost:8983

Generate a Test Curation Concerns Model:

```
> vagrant ssh
> cd ldata
> rails generate curation_concerns:work myobj
> rails s -p 3000 -b 0.0.0.0
```

Service should be accessible at: http://localhost:3000 . Create an account using the “Log In” link in the upper right. See the fields of a new object in your browser… they are essentially built on literals.

Using a URI

Making subject use a uri

DCTerms Subject:

- http://dublincore.org/documents/dcmi-terms/#terms-subject -> “This term is intended to be used with non-literal values as defined in the DCMI Abstract Model (http://dublincore.org/documents/abstract-model/). As of December 2007, the DCMI Usage Board is seeking a way to express this intention with a formal range declaration.”

- In addition, seems there is a slight preference to not mix literal and uri values in the same predicate.

Turn subject into uris:

```
> vagrant ssh
> cd ldata
> vim Gemfile

gem 'rdf', '1.99.0'

gem 'rdf-blazegraph'

> esc, :w, :q

> bundle update
```

Turn subject into uris:

```
> vim app/models/myobj.rb

class Myobj < ActiveFedora::Base
  ...
  def self.indexer
    MyobjIndexer
  end
end
```

Turn subject into uris:

```
> mkdir app/indexers
> vim app/indexers/myobj_indexer.rb

class MyobjIndexer < CurationConcerns::WorkIndexer
  def self.repo
    @repo ||=
::RDF::Blazegraph::Repository.new('http://localhost:8988/blazegraph/sparql'
)
  end

  def generate_solr_document
    super.tap do |solr_doc|
      #your solr content goes here
    end
  end
end
```

Turn subject into uris:

See gist at the following for rest of solr code: https://gist.github.com/scande3/283867b05b9ce070dc4c685763115430

(Quick TLDR: Parse the URI for the best “English” version of the subject as the primary label. Put the rest in an alternative label field so we can still search on those terms. In the end, we have five fields in solr: the original Subject field with the uri, two “preferred label” subject fields of ssim and tesim, and two “alternative labels” subject fields of ssim and tesim).

Turn subject into uris:

Go and get a uri or two from: http://id.loc.gov/search/?q=&q=cs%3Ahttp%3A%2F%2Fid.loc.gov%2Fauthorities%2Fsubjects (Example: http://id.loc.gov/authorities/subjects/sh85063283)

Create a sample object or two in your application using those uris in the “Subject” field!

Turn subject into uris:

Our object creates. . . but we still have these ugly uri's?!?! I can't read those at a glance!

> vim app/models/solr_document.rb

Add before the last "end":

```
  def subject
    fetch('subject_primary_label_ssim', [])
  end
```

> esc, :w, :q

Refresh the screen for magic!

Turn subject into uris:

Huh? What's that? Facets, index view, and searching now? Gee, aren't you demanding!

```
> vim app/controllers/catalog_controller.rb
```

Make changes from gist:
https://gist.github.com/scande3/9250e282dd48e4b5d227ff2c6cc366df

Turn subject into uris:

Try it out some!

Take a moment to see that it works correctly. Note:

1. We haven't added much validation. The user could enter in bad uri's or even string literals.

2. We could further expand upon this. For example, when searching for "Sports", you could return "Baseball" tagged objects.
    a. The theory is the same as the alt labels for this case. You just assign it a really low search weight so it only comes up once all other "Sports" objects have been exhausted.

Linked Data Fragments

Or how can this work without Blazegraph? How can I share code with my buddy using Marmotta?

Get the application:

```
> vagrant ssh
> git clone
https://github.com/ActiveTriples/linked-data-fragments.git
> cd linked_data_fragments
> cp config/ldf.yml.sample_blazegraph config/ldf.yml
> vim ldf.yml

Edit all ":3000" references to ":3001".

> esc, :w:, :q
> bundle install
> rake
> rails s -p 3001 -b 0.0.0.0
```

Test the application:

Root response test:

http://localhost:3001?format=jsonld

Sample response in different formats:

- http://localhost:3001/http://id.loc.gov/authorities/subjects/sh2010112128?format=jsonld
- http://localhost:3001/http://id.loc.gov/authorities/subjects/sh2010112128.nt
- http://localhost:3001/http://id.loc.gov/authorities/subjects/sh2010112128.ttl

Modify existing code for new backend:

```
> vagrant ssh
> cd ldata
> vim app/indexers/myobj_indexer.rb
```

Remove “self.repo” method. Add the following line above the first usage of that (after full_alt_term_list = []):

```
repo = RDF::Graph.load("http://localhost:3001/#{subj}.ttl", format: :ttl)
```

Replace “MyobjIndexer.repo” with just “repo”.

Gist of these changes:
https://gist.github.com/scande3/2ade4f5cf2d9551efc8b1b5c078f64b3

Try the application again!:

Restart your application (ctrl+c the window with “rails c -p 3000 -b 0.0.0.0” and run that command again).

Verify that it all still works. Congrats! You know have backend agnostic code for your Hydra Head.

Metadata Enrichment Interface

An example of potentially sharing code

Add Mei to the project:

```
> vagrant ssh
> cd ldata
> vim Gemfile

Gem ‘mei’, github: ‘boston-library/mei’

> esc, :w, :q
> bundle install
> rails generate mei:install
```

Setup your form:

```
> mkdir app/views/curation_concerns/base
> vim
app/views/curation_concerns/base/_form_additional_infor
mation.html.erb
```

Add content from:
https://gist.github.com/scande3/7c287794b7e5b6e0c36cb241601dd855

```
> esc, :w, :q
```

Setup your controller:

```
> vim
app/controllers/curation_concerns/myobjs_controller.rb
```

Add content from:
https://gist.github.com/scande3/5f616b2d2fc36e3cd041bacd12c145df

```
> esc, :w, :q
```

Try it out!:

Your form should now have a “lookup” option and display the label along with the uri!

This framework is extensible to more than just subjects. Is there interest in taking this beyond an “alpha release” gem?

Oregon Controlled Vocabulary Manager

Hosting a vocabulary

Controlled Vocabulary Manager:

```
> vagrant ssh
> cd ldata
> git clone
https://github.com/OregonDigital/ControlledVocabularyManager.git
> cd ControlledVocabularyManager
> sudo apt-get install build-essential libmysqlclient-dev
> sudo apt-get install cmake pkg-config
> bundle install
> vim config/settings/development.yml

Set blazegraph url to:
  url: "http://localhost:8988/blazegraph/sparql"
```

Controlled Vocabulary Manager:

```
> rake db:create
> rake db:migrate
> rails s -p 3002 -b 0.0.0.0

Create an account at: http://localhost:3002/

> ctrl+c (end rails)
> rails c
> user = User.all.last
> user.role = “admin”
> user.save
> ctrl+d (exit console)
> rails s -p 3002 -b 0.0.0.0
```

Opaquenamespace:

You can now try out your own linked data service!

Access it at: http://localhost:3002/

What next?

Continuing this type of work

Applied Linked Data Working Group

Is there a further interest in this type of work? If so, when do people want to collaborate on it?

Unresolved TODOs:

- Linked Data Fragments as a mountable engine.
- Sidecar indexer to poll and update linked data in Solr so the labels don't go stale.
- Next generation harvesting (ie. potentially ResourceSync?).
- Advancement of better Metadata interfaces.

For past meeting notes and information, see: https://wiki.duraspace.org/display/hydra/Applied+Linked+Data+Working+Group

THANKS!

Any questions?

Steven Anderson
Twitter: @scande3
sanderson@bpl.org

Slides: https://goo.gl/BP2inf