Introducing Cantaloupe

Seth Robbins – Manager, Scholarly Communications and Repository Services
Before we talk about cantaloupe we need to introduce IIIF: The International Image Interoperability Framework. IIIF maintains a set of APIs that provide standardized specifications for image delivery and presentation of complex digital objects.

**What is IIIF**

The Image API and Presentation API are important to Cantaloupe.

- **IIIF Image API 2.1**
- **IIIF Presentation API 2.1**
- **IIIF Authentication API 1.0**
- **IIIF Search API 1.0**
Image API

https://iiif.library.illinois.edu/dls/iiif/2/74815740-c455-0133-1d17-0050569601ca-9/full/!512,512/0/default.jpg

{scheme}://{server}/{prefix}/{identifier}/{region}/{size}/{rotation}/{quality}.{format}

The Image API provides a specification for providing image display parameters in the URL. This is how a client interacts with Cantaloupe and other IIIF image servers. It provides for server-side image transformations like cropping, rotation, resizing etc.
The Presentation API allows for detailed structural description of complex digital objects (e.g. digital books) via a JSON-LD manifest.

It’s used by image viewers like the Universal Viewer to make digital objects navigable and, in conjunction with the image API, explorable.
OK. Why does this exist?
Eliminate Redundant Work

Support Interoperability
Make Life Easier
Introducing Cantaloupe

Cantaloupe is an image server that implements the Image API. It was developed and is maintained by UIUC repository developer Alex Dolski. It is designed to be lightweight and easy to install and integrate into an exiting application eco-system. It can serve images from a file system, s3 storage, or a database.

https://medusa-project.github.io/cantaloupe/
The Illinois Digital Library was originally developed as an access portal for high-res digitized materials, but has grown to include born digital archival holdings as well.

Display is powered by the Universal Viewer, a javascript library that interoperates with the IIIF APIs to allow navigation and deep zooming of digital library objects.

The presentation API supports the paging functionality, using manifest generated based on digital library metadata, while the image API, served by Cantaloupe provides the detail view of individual pages.
Other Users

BAnQ (Bibliothèque et Archives Nationales Du Quebec)
http://numerique.banq.qc.ca/patrimoine/details/52327/1996201

Art Institute of Chicago
http://www.artic.edu/aic/collections/artwork/27992
Setting up a basic Cantaloupe dev server

Java 8 or higher should be installed. OpenJDK will work.
Setting up a basic Cantaloupe dev server

Get Started

This is a short guide to help you get going quickly with a typical standalone configuration. The user manual contains a more in-depth version of this guide that covers some additional features, like running in a web container and using the web-based "control panel."

Download the distribution archive.

Optional components, such as particular processors, may have additional requirements; see the user manual.

Configure

The distribution archive bundles a sample configuration file, named cantaloupe.properties.sample. Do not edit this file—instead, create a copy and edit that.

Open up your copy and set FilesystemResolver.BasicLookupStrategy.path_prefix to a folder path containing some images, and leave all the other options alone.
Setting up a basic Cantaloupe dev server

Once expanded, the distribution archive looks like this. Remove “sample” from the name of the configuration file.

Create a directory and save some image files into it.
Setting up a basic Cantaloupe dev server

Open the cantaloupe.properties file. Set this parameter to the path of the directory that you just created.

Setting up a basic Cantaloupe dev server

java -Dcantaloupe.config=~/.Downloads/Cantaloupe-3.4.2/cantaloupe.properties -Xmx2g -jar ~/.Downloads/Cantaloupe-3.4.2/Cantaloupe-3.4.2.war

The `cantaloupe.config` argument tells java where to find the config file.

Cantaloupe-x.x.x.war is the executable. This command uses `java -jar` to run it.

After running the command above, you can use the image API to display the files in a browser based on the filename.
Resolvers are components that can be configured to map identifiers from the URI to the resource you want to actually display.

By default, only the filesystem resolver is active allowing resources stored as files to be accessed based on their filesystem path, but resolvers also exist for S3 object storage as well as for images stored as BLOBs in a relational database.
Delegate Script

The delegate script allows scripted control over selected functionality including identifier resolution and authentication.

The delegate script is arranged as a series of overrideable Ruby methods.

```
module FilesystemResolver
  ##
  # @param identifier [String] Image identifier
  # @param context [Hash] Context for this request
  # @return [String,nil] Absolute pathname of the image corresponding to the given identifier, or nil if not found.
  #
  def self.get_pathname(identifier, context)
    end

  end
```
Delegate Script

```ruby
def self.get_pathname(identifier)
  uri = URI.parse( uri "#{MEDUSA_URL}/uuids/#{URI.escape( "arg identifier\").json"")
  attempts = 0
  found = false
  until found or attempts >= 3
    attempts += 1
    http = Net::HTTP.new( address uri.host, port uri.port)
    request = Net::HTTP::Get.new( path uri.request_uri)
    request.basic_auth(MEDUSA_USER, MEDUSA_SECRET)
    if uri.kind_of?(URI::HTTPS)
      http.use_ssl = true
      #http.verify_mode = OpenSSL::SSL::VERIFY_NONE
    end
    response = http.request(request)
    return nil if response.code.to_i >= 400
    break if response.code == 200
  end

  if response.header['location'] != nil
    new_uri = URI.parse( uri response.header['location']
    if new_uri.relative?
      new_uri = uri + response.header['location']
    end
    uri = new_uri
  else
    found = true
  end
end
```

For example, our configuration for resolving digital library object UUIDs to pathnames is shown here. The script constructs a URL to retrieve the file path information ...
Delegate Script

```ruby
def self.get_pathname(identifier)
  uri = URI.parse( uri: "#{MEDUSA_URL}/uuids/#{URI.escape( "arg identifier" )}.json")

  attempts = 0
  found = false
  until found or attempts >= 3
    attempts += 1
    http = Net::HTTP.new( address: uri.host, port: uri.port)
    request = Net::HTTP::Get.new( path: uri.request_uri)
    request.basic_auth(MEDUSA_USER, MEDUSA_SECRET)
    if uri.kind_of?(URI::HTTPS)
      http.use_ssl = true
      #http.verify_mode = OpenSSL::SSL::VERIFY_NONE
    end
    response = http.request(request)
    return nil if response.code.to_i >= 400
    break if response.code == 200

    if response.header['location'] != nil
      new_uri = URI.parse( uri: response.header['location'])
      if new_uri.relative?
        new_uri = uri + response.header['location']
      end
      uri = new_uri
    else
      found = true
    end
  end
  repos_root = REPOSITORY_ROOT
  JSON.parse(response.body)['relative_pathname']
end
```

...makes a request to an external API...
Delegate Script

```ruby
def self.get_pathname(identifier)
  uri = URI.parse( uri "#{MEDUSA_URL}/uuids/#{URI.escape( "arg identifier" )}.json")
  attempts = 0
  found = false
  until found or attempts >= 3
    attempts += 1
    http = Net::HTTP.new( address uri.host, port uri.port)
    request = Net::HTTP::Get.new( path uri.request_uri)
    request.basic_auth(MEDUSA_USER, MEDUSA_SECRET)
    if uri.kind_of?(URI::HTTPS)
      http.use_ssl = true
      #http.verify_mode = OpenSSL::SSL::VERIFY_NONE
    end
    response = http.request(request)
    return nil if response.code.to_i >= 400
    break if response.code == 200
    if response.header['location'] != nil
      new_uri = URI.parse( uri response.header['location'])
      if new_uri.relative?
        new_uri = uri + response.header['location']
      end
      new_uri = uri + response.header['location']
    end
    uri = new_uri
    if response.header['location']
      new_uri = URI.parse( uri response.header['location'])
      if new_uri.relative?
        new_uri = uri + response.header['location']
      end
      new_uri = uri + response.header['location']
    end
    Found = true
  end
  REPOSITORY_ROOT + JSON.parse(response.body)['relative_pathname']
end
```

...and the file path that is returned becomes the return value of the method.
Recommended Reading


Snydman, Sanderson and Cramer’s article gives a good introduction to both the image API and presentation API.
Thanks!

This an annotated version of the slides for a presentation at Open Repositories 2018 Conference, Bozeman, MT, 5 June 2018