MEDIator Data Replication Platform Documentation

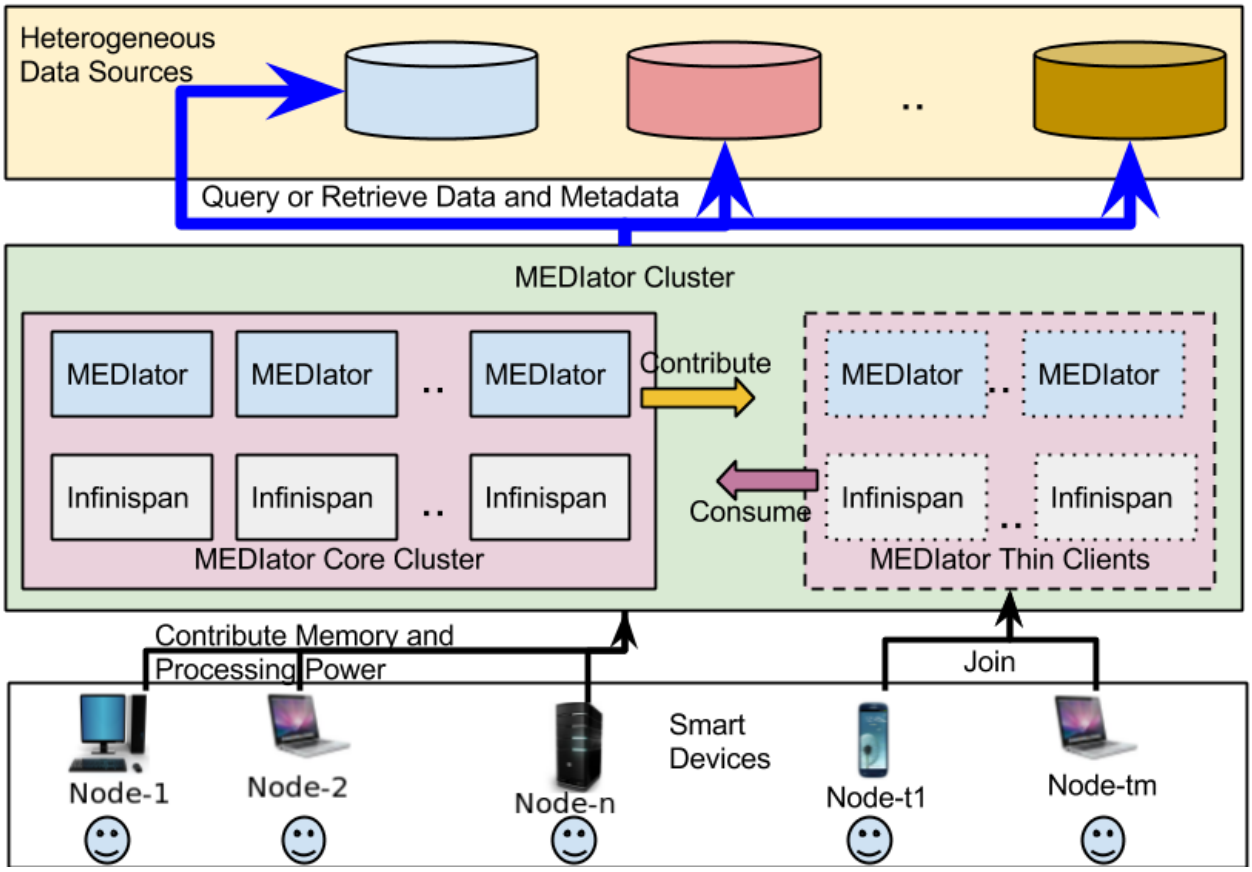
Release 1.0-SNAPSHOT (code named, SPREAD)

MEDIator Team

Nov 01, 2018

Contents

Welcome to the MEDIator Documentation. Here you will find information describing the features of the MEDIator platform, tips on how to use it, and details about its RESTful API.



With the growing adaptation of pervasive computing into medical domain and increasingly open access to data, metadata stored in medical image archives and legacy data stores is shared and synchronized across multiple devices of data consumers. While many medical image sources provide APIs for public access, an architecture that orchestrates an effective sharing and synchronization of metadata across multiple users, from different storage media and data sources, is still lacking.

MEDIator is a data sharing and synchronization middleware platform for heterogeneous medical image archives. MEDIator allows sharing pointers to medical data efficiently, while letting the consumers manipulate the pointers without modifying the raw medical data. MEDIator has been implemented for multiple data sources, including Amazon S3, The Cancer Imaging Archive (TCIA), caMicroscope, and metadata from CSV files for cancer images.

This documentation is intended to serve both the MEDIator developers/deployers as well as the MEDIator users. Please note that MEDIator version 1.0 has been code named SPREAD (System for Sharing and Publishing Research Data). You may find sentences referring to MEDIator by this code name.

CHAPTER 1

Getting Started With MEDIator

You may download and build MEDIator from its source code, which is readily avilable at

https://bitbucket.org/BMI/datareplicationsystem/

There is also a containerized version of MEDIator available at,

https://hub.docker.com/r/datacafe/mediator/

However, please note that the MEDIator container can often be an outdated version. So it is recommended to clone MEDIator source code and build it with Maven.

The source code of this documentation can be found at, https://github.com/pradeeban/mediator

This documentation is currently hosted at, http://mediator.readthedocs.io/

CHAPTER 2

MEDIator Research

2.1 Use case

Data Sharing is cited as a major impediment to progress in Precision Medicine. MEDIator has an objective of data sharing during research. It is similar to Dropbox/Box pattern for sharing file or folders. MEDIator does not use file name, as data is rarely organized as files and filenames. It Relies on metadata used during search. It provides an API that allows you to create and download 'links' using this metadata.

2.1.1 Replica Sets

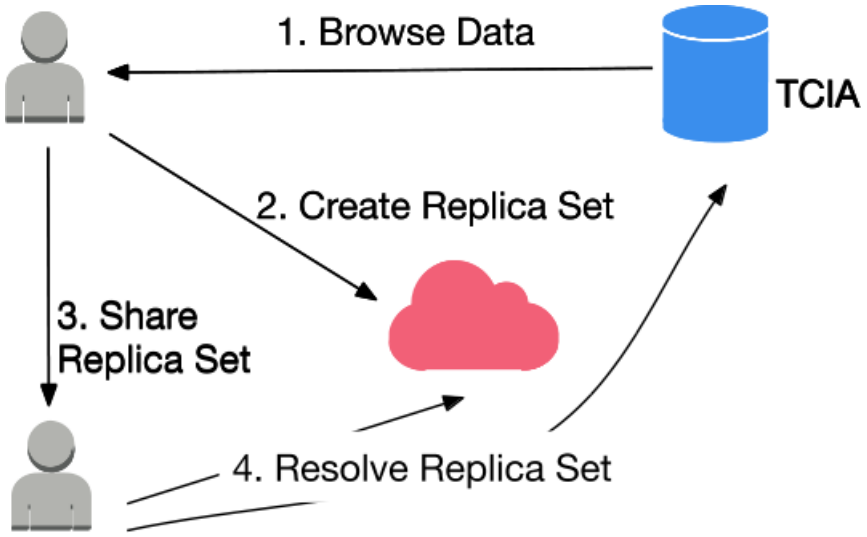
Replica sets is the key concept that drives MEDIator. Research data is addressed via metadata. Metadata is used to search, locate and download data. Replica Set is a combination of metadata that uniquely identifies the data of interest.

MEDIator offers CRUD APIs for Replica Sets and APIs to resolve Replica Sets and access the data.

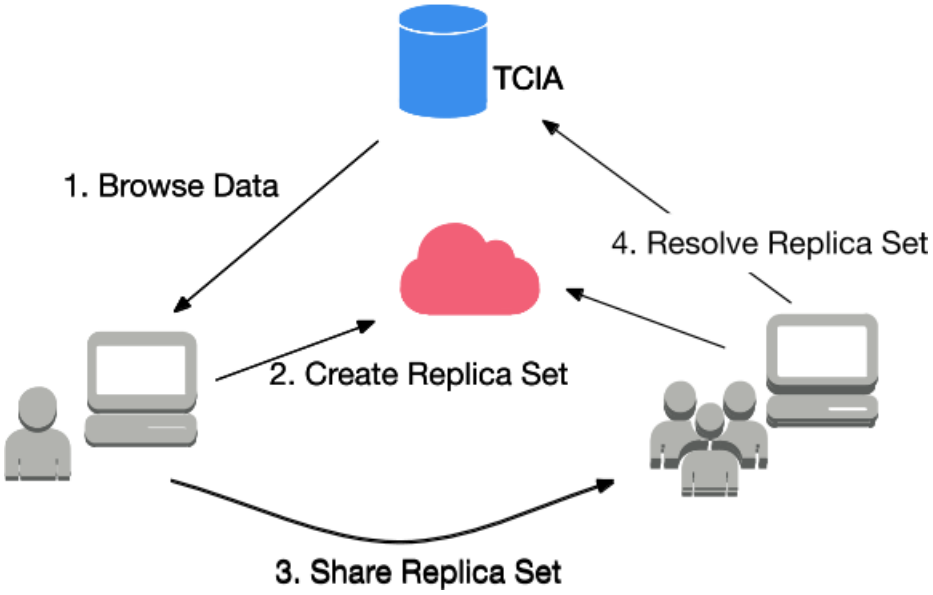
The Cancer Imaging Archive (TCIA) encourages and supports cancer-related open science communities by hosting and managing the data archive, and relevant resources to facilitate collaborative research. Hence, MEDIator has been primarily implemented with TCIA to begin with.

2.1.2 Research Use case

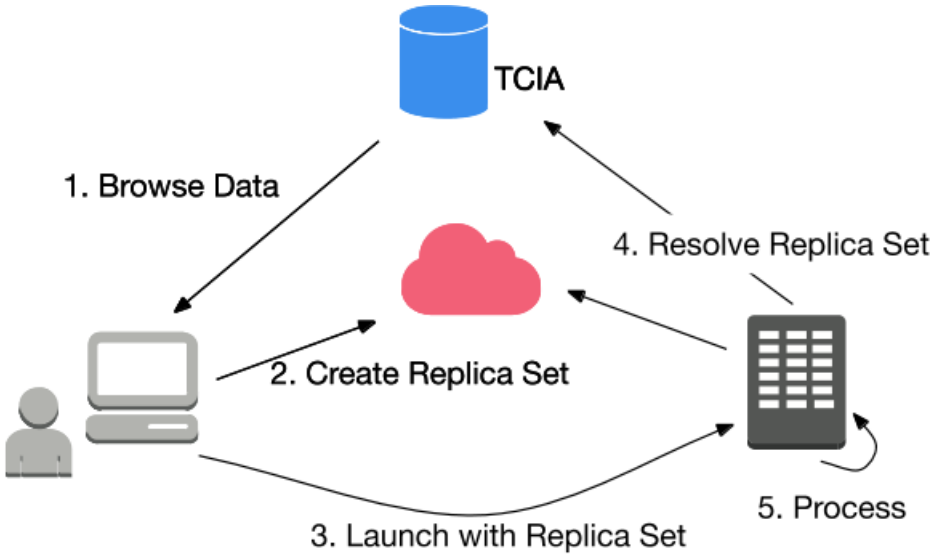
Researcher wants to share a subset of TCIA managed data with collaborator.



Orchestrate this data sharing from a research workstation.



Incorporate this in batch runs of data analysis pipelines.



Have feedback or corrections? Email us at pkathi2@emory.edu.

2.2 Introduction to the MEDIator Data Replication Platform

This page is intended to give an overall idea and the motivation behind the inception of MEDIator platform.

2.2.1 Introduction

Radiomics and Radiogenomics research requires large amounts of well curated, high quality, imaging and genomic data. The typical workflow involves downloading the data from a public repository, or the data warehouse of the data coordination site, and processing and analyzing the data on local compute clusters. As new data is added by the data providers, it is often left to the individual investigator to track and download new data. Since these research studies are often multi-investigator, research groups often designate a point- person as the local 'project-specific' data disseminators, who in turn set local databases/spreadsheets/ dropbox accounts to track the data downloads. This rather inefficient and error-prone workflow is necessary because there are no good systems to share data amongst investigators and collaborators. Finally, upon conclusion of study, investigators rarely have a good way to identify the specific subset of data (from the centralized repository/DCC) that was used in their project. In the absence of a good way to publish their data, it becomes difficult to reproduce scientific studies.

The purpose of this project was to develop a system that could provide data repositories and DCC with a seamless, and transparent, approach to give their users the ability to share and publish research data — a one-way Dropbox like environment for the community which gives researchers the ability to share data subsets, track and download updates, and publish data.

2.2.2 Methods

The primary design construct of MEDIator is a system for the creation, access and update of replica sets. Replica sets can be thought of as pointers that consist of metadata-tuples that can be used to uniquely identify and point to data. It points to a chosen subset of data, that is stored in one or more homogeneous/ heterogeneous data sources. When using replica sets, no raw data is duplicated, and only the pointers to the data are shared. Data consumers share the identifiers among the other interested data consumers. Replica sets and identifiers are stored in a distributed storage

offered by MEDIator. As the data and execution is distributed among multiple computing nodes, parallel execution of thousands of queries are enabled without a performance degrade.

Data providers are responsible for identifying metadata that is used to describe and access the raw data. Existing replica sets can be updated for each of the data sources, at the specified granularity of metadata.

While only the data consumers with the credentials (often those who created the replica set) can expose, update, or delete the existing replica sets, public replica sets can be duplicated by the other consumers. MEDIator data replication mechanism consists of 3 APIs: a) An Interface API provides a generic interface that can be used by a data provider, or a DCC, to integrate with their data management system. The implementation of this interface is used to query the metadata and retrieve the raw data. b) A PubCons API that can be used by data consumers, to help create, update, and delete replica sets, at varying granularity. c) An Integrator API that integrates data, and allows replica sets to be created from multiple data sources. Each replica set is represented as a hierarchical tree structure where each data source is mapped as a sub-tree. Leaves of the tree point to metadata at different granularity. Multiple homogeneous data sources can leverage a single implementation of the PubCons API, where heterogeneous data sources are integrated as sub trees of the replica set by invoking the relevant implementation of PubCons API for each of the data source.

We also provide an SDK to enable the creation of replica sets effectively through native Java and Python interfaces. This allows developers, of platforms such as 3DSlicer (a medical imaging research workstation) and Galaxy, to easily integrate the data sharing capability into their respective applications. Such native access allows their users to exchange data amongst collaborators without explicitly moving large amounts of data.

Finally, we provide users the ability to publish a replica set and associate it with a Digital Object Identifier. A published replica-set is immutable, and the act of publishing makes the replica-set read-only. Before a DOI can be generated, the publisher is also prompted to provide some metadata.

2.2.3 Evaluation

MEDIator was tested against TCIA, Amazon S3, and both TCIA and Amazon S3 together. It was evaluated in a cluster up to 6 identical nodes of servers with Intel® CoreTM i7-2600K CPU @ 3.40GHz and 12 GB memory. Data sources up to 300 GB from TCIA consisting of around 100000 image files and S3 were used in the evaluations.

Table 1: Evaluated TCIA Collections

Collection	Images	Patients	Size (GB)
TCGA-LUSC	35,678	36	14.5
TCGA-GBM	476,515	261	72.8
TCGA-BRCA	230,167	139	88.1
LIDC-IDRI	244,527	1,010	124

2.2.4 Results

The implementation demonstrates the extensibility and efficiency of MEDIator in integrating and sharing data from multiple heterogeneous data source among the data consumers. As a sample data sharing scenario, we deployed MEDIator against three heterogeneous biomedical image sources. The data sources were: a) Radiology data from The Cancer Imaging Archive; b) digital pathology images in caMicroscope; and c) sequence data that was stored in Amazon S3 buckets. While the genomic data was stored in S3 buckets, the corresponding clinical data (used as metadata) was stored as CSV files.

TCIA and caMicroscope provide robust data management services, including a set of RESTful APIs are used to query the images and the relevant metadata. The genomic data was stored as Amazon S3 blobs and linked to metadata (clinical data) that was stored as CSV files. The CSV files themselves were integrated and queried through CSVIntegrator,

an implementation of IntegratorAPI to read and parse the CSV meta files. The meta files were read and stored in the distributed in-memory grid of MEDIator for easy access to minimize the performance overhead caused by the frequent disk access.

The pilot project used ~100000 image files and nearly 300GB of data. The result of this pilot project illustrated the robustness of the MEDIator system in integrating multiple, diverse data sources (including a cloud based data source), and provide users the ability to share subsets of data without an explicit download of data.

2.2.5 Discussion

In this project we present an extensible, and transparent system, that gives data publishers the ability to allow their users to share data, without explicitly downloading and distributing data. Such a lightweight approach provides ubiquitous access to medical metadata from the radiomics/radiogenomics archives, while providing fault tolerance and load balancing, leveraging the distributed shared memory platforms. As new data is added by data providers, replica sets can be easily updated. This allows consumers to access and download only the data that was updated. Finally, we provide data consumers, the ability to 'publish' data used in their studies. In this model of publishing data, a consumer does not upload data. Instead they identify the data subset that was used in specific study. This model encourages reproducibility in science and easier access to the data used in projects.

2.2.6 Conclusion

The emerging research domains of radiomics and radiogenomics place a heavy emphasis on the need for large scale, well curated information repositories. Our work demonstrates a novel approach to sharing and accessing subsets of data stored in such repositories. Our motivating premise is that such novel mechanisms to share data, will encourage data reuse, as well as streamline access to data from algorithms.

MEDIator is a platform for sharing medical images across multiple users by merely sharing the metadata, from the heterogeneous image archives by leveraging the distributed shared memory platforms. The design can also be implemented for any other data sources having an index to query and structure them as replica sets.

2.2.7 Tentative Action Items

Public Release: Summer 2016 (TCIA)

Integrate with NIH Genomic Data Commons

Persist Replica Sets

Associate Digital Object Identifiers with Replica Sets

Have feedback or corrections? Email us at pkathi2@emory.edu.

CHAPTER 3

MEDIator for Users

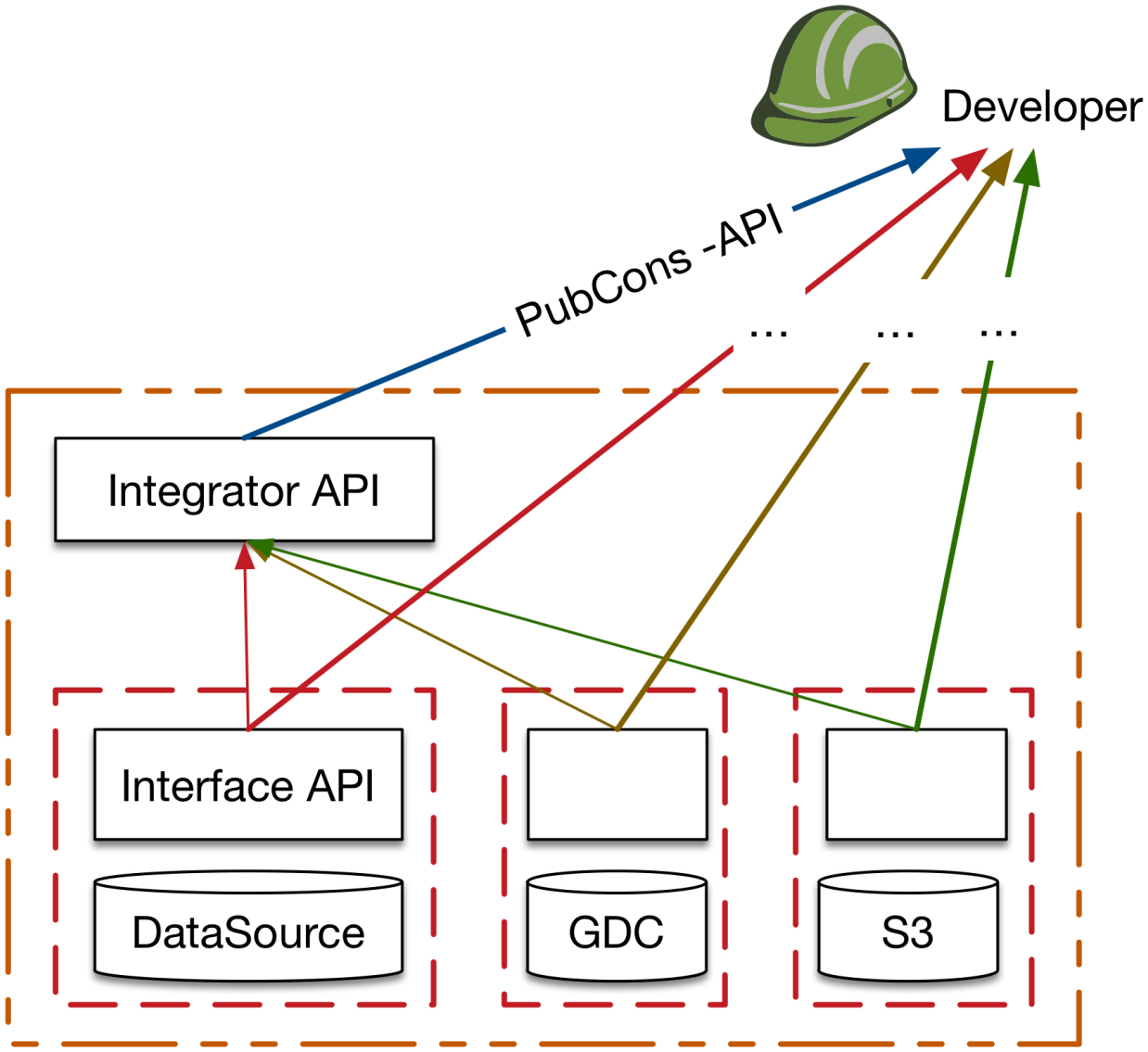
3.1 MEDIator Interfaces

MEDIator consists of 3 interfaces for data source management and replica set management. Data Source Management API (Formerly known as Interface API) is an interface for retrieving data and metadata from the original data sources. This is customized for each data source, though is often provided by the data source itself.

Replica Set Management API (Formerly known as PubCons API) is a public interface customized for each data source. It is the core of MEDIator, for creating and managing replica sets.

In addition to these two, there is also an Integrator API that allows the developers to create and manage replica sets that span across multiple heterogeneous data sources.

Infinispan in-memory data grid has been leveraged as the distributed storage and execution platform for MEDIator. It minimizes performance overheads caused by disk access. Furthermore, current work-in-progress ensures no duplicate downloads while ensuring duplicate detection across the locally downloaded data sets and the data source.

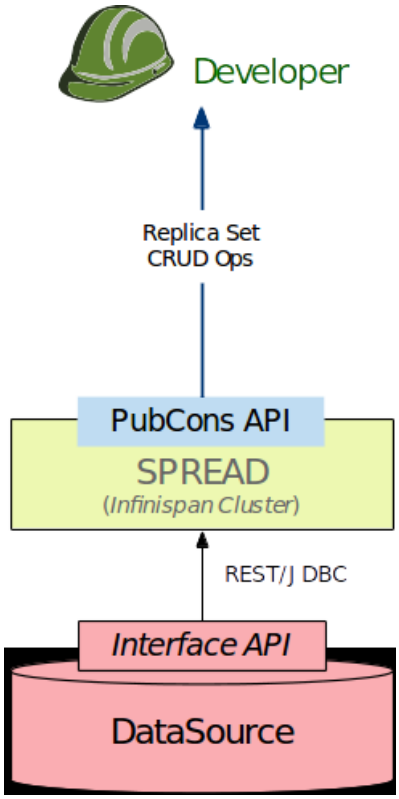


3.1.1 Core Interfaces of MEDIator

Data Source Management (Formerly known as Interface API)

Data Source Management for Users

Data source management API integrates with the data sources for data retrieval. As the original interface of the data sources, it was initially called as the Interface API. It utilizes access mechanisms provided by remote data sources, as a mean to query and retrieve the raw data.



Additional functionality of download tracking and near duplicate detection have been proposed and implemented by MEDIator in addition to the bare-bone data download offered by the data source providers. These include,

1. download changes since the previous downloads
2. resume or Update.

Data Source Management for Developers

Data source management refers to the management of the original data sources replicated by MEDIator. The implementation of the data source management is often offered by the data source providers themselves.

Relevant classes can be found in the package: ds_mgmt.

Data source management module manages the data sources themselves.

The relevant interfaces and implementations are often provided by the data sources or the data providers, and hence orthogonal to MEDIator. However, a TCIA data source management RESTful interface and implementation are included.

Have feedback or corrections? Email us at pkathi2@emory.edu.

Replica Set Management (Formerly known as PubCons API)

PubCons for Users

PubCons (Publish-Consume Replica Sets) is an interface specific to each data source. Customization via metadata mappings.

API parameters include

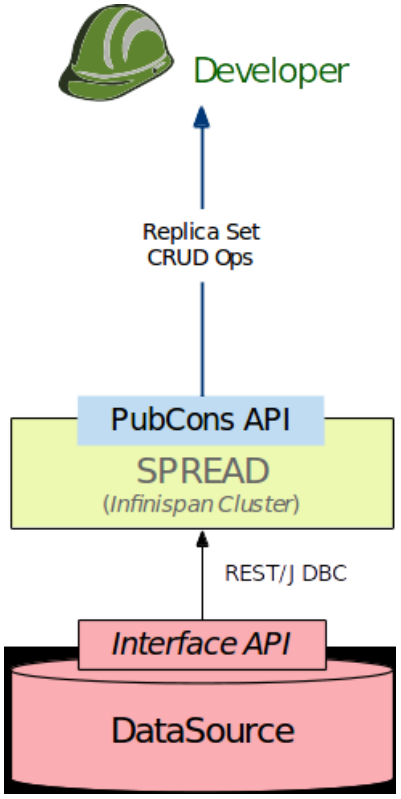
1. userID

ID of the user who created the replica set. This serves as the key of the map of replicas for the user

2. replicaSetID.

ID of the replica set.

PubCons consists of multi-hierarchical maps for efficient execution in a large-scale production environment. Boolean flags to indicate existence of entries at a certain granularity.



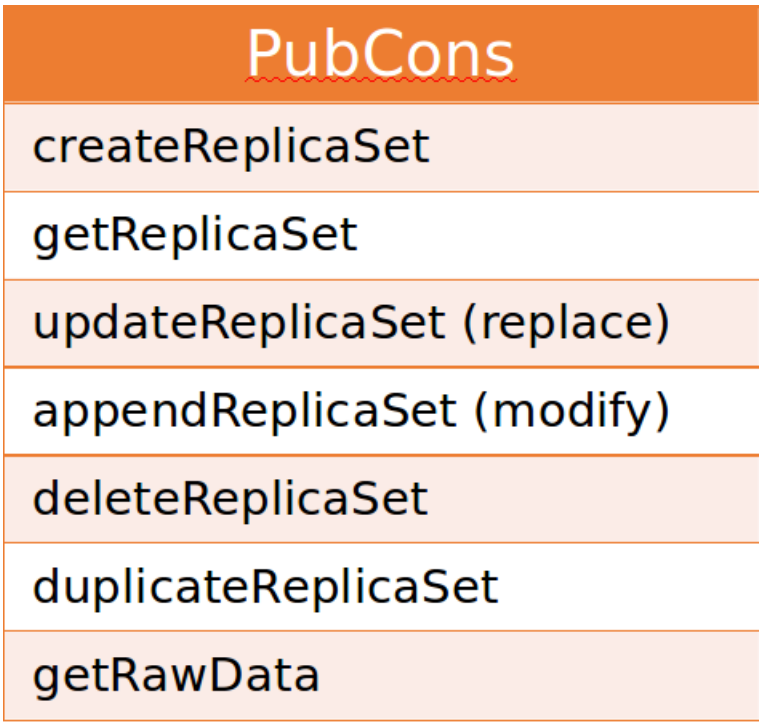
User facing CRUD REST API

C Create

R Retrieve

U Update

D Delete



PubCons for Developers

Relevant classes can be found in the package: rs_mgmt.

Replicaset management module manages the replica sets pointing to each of the data sources.

ReplicaSetHandlers are implemented for each of the data sources that are federated by MEDIator. The ReplicaSetHandlers offer an internal implementation of the replica sets management.

ReplicaSetManagers are the core REST APIs of MEDIator. They leverage the ReplicaSetHandlers to manage the replica sets of each of the data sources. There is a one-to-one mapping between the ReplicaSetManager api implementations and the data sources.

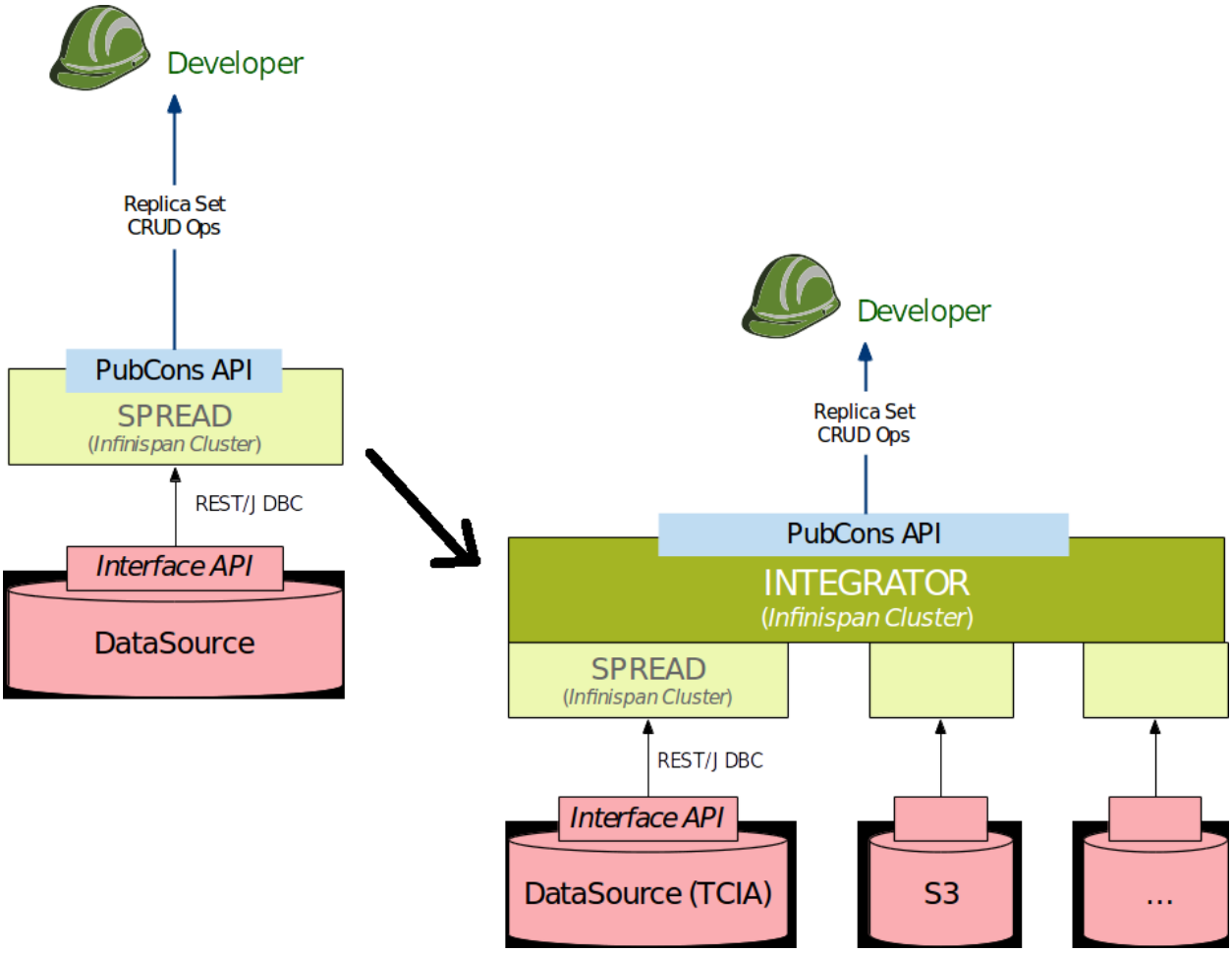
TciaReplicaSetManager provides the REST API for managing the TCIA replica sets. Relevant documentation can be found in the class as the method-level comments.

Have feedback or corrections? Email us at pkathi2@emory.edu.

Integrator

Integrator for Users

Integrator interacts with virtual data stores that integrate relevant public/private repositories. It integrates multiple heterogeneous data sources into MEDIator. It provides a meta map and maps for each of the sources contributing to the replica set. It allows the users to access relevant information for each of the entry at a coarse granularity.



Integrator for Developers

Relevant classes of the Integrator implementation can be found in the package: integrator. ReplicaSetsIntegrator is a singleton that manages integration of all the data sources for the replica set management. Hence, ReplicaSetsIntegrator is a single entity that has a one-to-many relationship with the data sources of MEDIator for the replicaset management.

Have feedback or corrections? Email us at pkathi2@emory.edu.

Have feedback or corrections? Email us at pkathi2@emory.edu.

3.2 MEDIator RESTful APIs

If you are hosting MEDIator for public access, you need to start it and expose its RESTful APIs. Execute the MEDIatorEngine class.

$ java -classpath lib/repl-server-1.0-SNAPSHOT.jar:lib/*:conf/ edu.emory.bmi.datarepl.core.MEDIatorEngine

You may get it running with nohup:

$ nohup java -classpath lib/repl-server-1.0-SNAPSHOT.jar:lib/*:conf/ edu.emory.bmi.datarepl.core.MEDIatorEngine &

To add more instances to the cluster, start the instances of Initiator class. $ java -classpath lib/repl-server-1.0-SNAPSHOT.jar:lib/*:conf/ edu.emory.bmi.datarepl.core.Initiator

The implementation of the RESTful invocations can be found at TciaReplicaSetManager.

You may access MEDIator's RESTful APIs directly or through your application that consumes the MEDIator APIs.

You may extend or leverage the exposed APIs. To begin with, you may consume the MEDIator RESTful APIs through a REST client such as the Postman plugin of the Chrome browser.

You will be required to run MEDIator server as a super-user if you choose to have the port to be below 1024.

MEDIator Client

A sample Python client developed with Unirest has been included in MEDIator root directory.

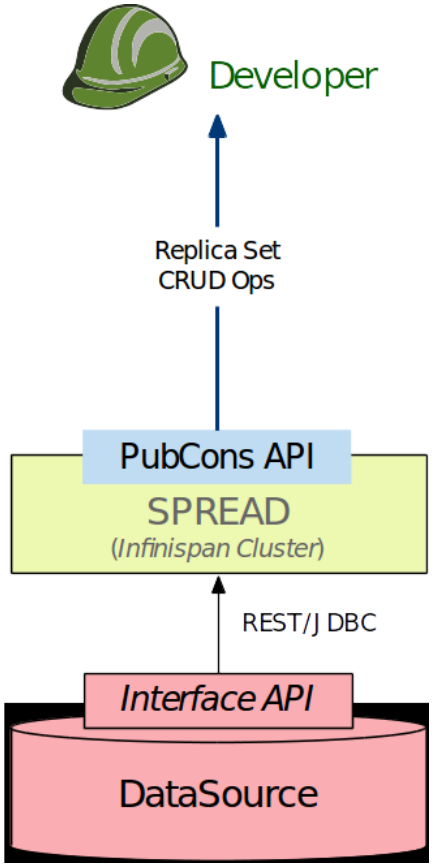
Set the REST_PORT variable in the MEDIator client to match the value as defined in the CommonConstants for the MEDIator server.

You may run the MEDIator client, after running the MEDIatorEngine, the core initialization class of the MEDIator server -

$ ./MEDIator_Client.py

You may have to change the execution mode of the client application prior to running it though.

$ chmod u+x MEDIator_Client.py



3.2.1 RESTful Invocations of MEDIator

Create Replica Set

Users may create replica sets to point to their interested sub sets of data at different granularity. They may do so, by creating a replica set from the scratch, or by duplicating an existing replica set. The created (or duplicated) replica sets can later be modified by the user.

For the ease of expression, port 9090 is used as an example in all the examples below.

Make sure to replace 9090 above with the correct value as defined in REST_PORT in MEDIator CommonConstants.

Create Replica Set from Scratch

Replica sets can be created using a POST call following the format of,

/POST

/replicasets

Sample POST request

http://localhost:9090/replicasets?iUserID=12&iCollection=TCGA-GBM&iPatientID=TCGA-06-6701%2CTCGA-08-0831&iStudyInstanceUID=1.3.6.1.4.1.14519.5.2.1.4591.4001.151679082681232740021018262895&iSeriesInstanceUID=1.3.6.1.4.1.14519.5.2.1.4591.4001.179004339156422100336233996679

This creates a replica set for the user with the user ID 12, with the given attributes of the images hosted in TCIA.

Sample POST response

The expected response of the REST call is the ID of the replica set that was created in the previous replica set creation POST request.

This is a random generated number.

-4764762120292626164

Duplicate Replica Set

Instead of creating a replica set from scratch, users may share their replica sets with their friends who may duplicate the replica set to have it as their own. The POST call for the duplicate replica set follows the format as below,

/POST

/replicaset

Sample POST request

http://localhost:9090/replicaset?userID=1234567&replicaSetID=-7196077834010820228

The inputs to this POST method are the ID of the current user who is duplicating the replica set, as well as the ID of the replica set that has been duplicated by the user.

Sample POST response

-5054196249282594410

Similar to the create replica set method above, the duplicate method too returns the ID of the newly created replica set. This is because the internals of the create and duplicate replica set methods function in a similar way, generating a replica set with a randomly generated replica set ID.

Have feedback or corrections? Email us at pkathi2@emory.edu.

Retrieve Replica Sets

A list of the replica sets of the given user, as well as the specifics of each of the replica sets can be retrieved through the relevant GET calls.

For the ease of expression, port 9090 is used as an example in all the examples below.

Make sure to replace 9090 above with the correct value as defined in REST_PORT in MEDIator CommonConstants.

Retrieve Replica Sets of the user

This follows the format of

/GET

/replicasets/:id

Sample GET request

http://localhost:9090/replicasets/12

Sample GET response

If the user has created replica sets, the IDs of the replica sets would be returned as an array, as below.

[-7466653342708752832, -7059417815353339196, -6908825180316283930, -6365519002970140943]

If he does not have any replica sets created, the lack of replica sets would be indicated.

> Replicasets not found for the user: 123

Retrieve A Replica Set

This follows the format of

/GET

/replicaset/:id

Sample GET request

http://localhost:9090/replicaset/-9176938584709039161

Sample GET response

If the replica set exists, it would be returned as below.

Collection Names: [TCGA-GBM]. Patient IDs: [TCGA-06-6701, TCGA-08-0831]. StudyInstanceUIDs: [1.3.6.1.4.1.14519.5.2.1.4591.4001.151679082681232740021018262895]. SeriesInstanceUIDs: [1.3.6.1.4.1.14519.5.2.1.4591.4001.179004339156422100336233996679]

If the replica set retrieval failed, it would cause an internal error as below.

> <html> <body> <h2>500 Internal Error</h2> </body> </html>

Have feedback or corrections? Email us at pkathi2@emory.edu.

Update Replica Sets

A replica set can either be replaced entirely with new contents, or be appended along with the existing content, by the user.

Replace Replica Set

A replica set can be replaced with pointers to new content. While the replica set ID remains the same, the entire of its content would be replaced with pointers to the updated data sets or images.

Replica sets can be replaced by a POST call of a following format.

/POST

/replicaset/:id

For the ease of expression, port 9090 is used as an example in all the examples below.

Make sure to replace 9090 above with the correct value as defined in REST_PORT in MEDIator CommonConstants.

Sample POST request

http://localhost:9090/replicaset/-5841894688098285105?iStudyInstanceUID=1.3.6.1.4.1.14519.5.2.1.4591.4001.151679082681232740021018262895&iSeriesInstanceUID=1.3.6.1.4.1.14519.5.2.1.4591.4001.179004339156422100336233996679

This replaces the replica set with the given ID with the newly given information.

Sample POST response

true

or

false

If the replica set was successfully replaced, 'true' will be returned. Otherwise, 'false' will be returned.

Append Replica Set

Practically, it is convenient to append an existing replica set than entirely replacing it. Replica sets can be appended by a PUT call of the following format.

/PUT

/replicaset/:id

Sample PUT request

http://localhost:9090/replicaset/-5841894688098285105?iCollection=TCGA-GBM

This appends the existing replica set of the given ID with pointers to the newly given subsets of data.

Sample PUT response

If the replica set was successfully updated, 'true' will be returned. Otherwise, 'false' will be returned.

Have feedback or corrections? Email us at pkathi2@emory.edu.

Delete Replica Set

An existing replica set can be deleted by invoking the relevant DELETE REST API, following the format of,

/DELETE

/replicaset/:id"

The replica set with the given ID would be deleted.

For the ease of expression, port 9090 is used as an example in all the examples below.

Make sure to replace 9090 above with the correct value as defined in REST_PORT in MEDIator CommonConstants.

Sample DELETE request

http://localhost:9090/replicaset/12?replicaSetID=-9176938584709039161

Sample DELETE response

true

or

false

True, if successfully deleted the replica set with the given ID, and false if the delete failed for some reason.

Have feedback or corrections? Email us at pkathi2@emory.edu.

Have feedback or corrections? Email us at pkathi2@emory.edu.

3.3 MEDIator Web Application

In order to execute the MEDIator web application, run the below command.

sh modules/repl-server/target/bin/webapp

To access the MEDIator web application, go to http://localhost:<EMBEDDED_TOMCAT_PORT>/mediator/ using your browser.

You may run the Initiator class in parallel, to create a MEDIator cluster in both cases.

You will be required to run MEDIator server as a super-user if you choose to have the port to be below 1024.

Have feedback or corrections? Email us at pkathi2@emory.edu.

CHAPTER 4

MEDIator for Developers

4.1 Developing MEDIator

If you would like to fork and extend MEDIator, have a look at its source code, which is publicly available at Bitbucket.

https://bitbucket.org/BMI/datareplicationsystem/

If you would like to contribute to this documentation, please send a pull request to,

https://github.com/pradeeban/mediator

Have feedback or corrections? Email us at pkathi2@emory.edu.

4.2 Building and Deploying MEDIator

4.2.1 Building from the source

Checkout the source code

git clone https://<user-name>@bitbucket.org/BMI/datareplicationsystem.git

Constants

edu.emory.bmi.datarepl.constants package contains the constants to be modified.

The class DataSourcesConstants contains the constants.

Configuring Sample Data

Download a set of medical source from The Cancer Genome Atlas (TCGA)

https://tcga-data.nci.nih.gov/tcga/

When you pick the data and click download, it will send the files in a tar, along with a file map, "FILE_SAMPLE_MAP.txt".

Download the tar, extract them, and upload to S3.

NOTE: The tar contains the associated "blob" that will be linked with the clinical data. The blob in itself does not have any metadata.

Meta data

Place all the meta data files in the root folder of the data replication system source code.

4.2.2 Configurations

Set the name of the CSV file that is used to load the CSV meta data into Infinispan.

public static final String META_CSV_FILE = "getAllDataAsCSV";

NOTE: This is the clinical data

Also set the name of the file sample map of TCGA that is uploaded to S3.

public static final String S3_META_CSV_FILE = "FILE_SAMPLE_MAP.txt";

NOTE: This is the mapping between the clinical data and the associated "blob" that was uploaded to S3

S3 constants

S3 base url contains https://s3.amazonaws.com/ and the bucket name, (dataReplServer was chosen for this prototype).

public static final String S3_BASE_URL = "https://s3.amazonaws.com/dataReplServer/";

Folder names.

The meta data comes into multiple folders, with level1 as the default. If you want to change the default folder, change the value of the constant S3_LEVEL1.

public static final String S3_LEVEL1 = "level1";

public static final String S3_LEVEL2 = "level2";

public static final String S3_LEVEL3 = "level3";

API Key

Provide your TCIA API Key in TCIAConstants for the below field.

public static String API_KEY = "";

NOTE: This is the TCIA API Key.

CA constants.

Check the caMicroscope base url and sample query url that is used in the prototype, in DataSourcesConstants class.

public static final String CA_MICROSCOPE_BASE_URL = "http://imaging.cci.emory.edu/camicroscope/camicroscope/";

public static final String CA_MICROSCOPE_QUERY_URL = "osdCamicroscope.php?tissueId=";

Build from the source code root

$ mvn package

To execute the DSIntegratorIntegrator:

$ java -classpath lib/repl-server-1.0-SNAPSHOT.jar:lib/*:conf/ edu.emory.bmi.datarepl.ds_impl.DSIntegratorInitiator

4.2.3 Dependencies

This project depends on the below major projects.

- Infinispan
- Apache Tomcat (Embedded)
- Apache HTTP Client
- Apache Velocity
- Apache Log4j2
- Mashape Unirest

4.2.4 Setting the Environment Variables

Set the API_KEY, MASHAPE_AUTHORIZATION, and BASE_UR as environment variables.

For example, in Linux.

```
export API_KEY=your-api-key
export MASHAPE_AUTHORIZATION=your-mashape-authorization
export BASE_URL=services.cancerimagingarchive.net/services/v4/TCIA/query
```

4.2.5 Building and Executing Using Apache Maven 3.x.x

mvn package

It is expected to have the TCIA API_KEY set in the environment variables to build with tests.

If you do not have a TCIA API_KEY yet, please build with skipping the tests.

mvn package -DskipTests

4.2.6 Logging

Make sure to include log4j2-test.xml into your class path to be able to configure and view the logs. Default log level is [WARN].

Have feedback or corrections? Email us at pkathi2@emory.edu.

4.3 MEDIator API Gateway and Portal

MEDIator can also be deployed with API gateways and portals such as those offered by Tyk.

4.3.1 Setting up Tyk Portal for MEDIator

There was this long running bug that is hitting us from setting up the Tyk portal. The issue was with Tyk not allowing custom domains from being set up as portal. While the settings as advised in Tyk mention the dashboard and portal port as 3000, ideally it should be set as 80 to get the portal configured seamlessly.

Setting up Portal Domain

Add to /etc/hosts file:

127.0.0.1 dashboard.tyk-local.com

127.0.0.1 portal.tyk-local.com

Most probably Apache2 or some other web server would already be listening on the port 80. Make sure to stop it to release the port.

Stop Apache2 Web Server

/etc/init.d/apache2 stop

Configure Tyk Gateway

sudo /opt/tyk-gateway/install/setup.sh –dashboard=http://dashboard.tyk-local.com –listenport=8080 –redishost=localhost –redisport=6379 –domain=””“

Configure Tyk Dashboard

sudo /opt/tyk-dashboard/install/setup.sh –listenport=80 –redishost=localhost –redisport=6379 –mongo=mongodb://127.0.0.1/tyk_analytics –tyk_api_hostname=127.0.0.1 –tyk_node_hostname=http://127.0.0.1 –tyk_node_port=8080 –portal_root=/portal –domain=”dashboard.tyk-local.com”

Configure Tyk Pump

sudo /opt/tyk-pump/install/setup.sh –redishost=localhost –redisport=6379 –mongo=mongodb://127.0.0.1/tyk_analytics

Initial start to set up the Tyk license

sudo service tyk-pump start

sudo service tyk-dashboard start

Access http://dashboard.tyk-local.com/ and add the license in the prompt.

Restart the Tyk Dashboard and start the Tyk Gateway

sudo service tyk-dashboard restart

sudo service tyk-gateway start

Fix the Tyk bootstrap script to listen at port 80

gedit /opt/tyk-dashboard/install/bootstrap.sh and remove :3000 from all the references, since we are going to use the default port 80, instead of port 3000 as the dashboard port.

#!/bin/bash

Usage ./bootstrap.sh DASHBOARD_HOSTNAME

LOCALIP=$1

RANDOM_USER=$(env LC_CTYPE=C tr -dc “a-z0-9” < /dev/urandom | head -c 10)

PASS=”test123”

echo “Creating Organisation”

ORGDATA=$(curl –silent –header “admin-auth: 12345” –header “Content-Type:application/json” –data ‘{“owner_name”: “Default Org.”,”owner_slug”: “default”, “cname_enabled”: true, “cname”: “”}’ http://$LOCALIP/admin/organisations 2>&1)

#echo $ORGDATA

ORGID=$(echo $ORGDATA | python -c ‘import json,sys;obj=json.load(sys.stdin);print obj[“Meta”]’)

echo “ORGID: $ORGID”

echo “Adding new user”

USER_DATA=$(curl –silent –header “admin-auth: 12345” –header “Content-Type:application/json” –data ‘{“first_name”: “John”,”last_name”: “Smith”,”email_address”: “’$RANDOM_USER’@default.com”,”password”:”’$PASS’”, “active”: true,”org_id”: “’$ORGID’”}’ http://$LOCALIP/admin/users 2>&1)

#echo $USER_DATA

USER_CODE=$(echo $USER_DATA | python -c ‘import json,sys;obj=json.load(sys.stdin);print obj[“Message”]’)

echo “USER AUTH: $USER_CODE”

USER_LIST=$(curl –silent –header “authorization: $USER_CODE” http://\protect\T1\textdollarLOCALIP/api/users 2>&1)

#echo $USER_LIST

USER_ID=$(echo $USER_LIST | python -c ‘import json,sys;obj=json.load(sys.stdin);print obj[“users”][0][“id”]’)

echo “NEW ID: $USER_ID”

echo “Setting password”

OK=$(curl –silent –header “authorization: $USER_CODE” –header “Content-Type:application/json” http://$LOCALIP/api/users/$USER_ID/actions/reset –data ‘{“new_password”:”’$PASS’”}’)

echo “”

echo “DONE”

echo “====”

echo “Login at http://\protect\T1\textdollarLOCALIP/”

echo “User: $RANDOM_USER@default.com”

echo “Pass: $PASS”

echo “”

Save and exit the bootstrap script and execute it.

sudo /opt/tyk-dashboard/install/bootstrap.sh dashboard.tyk-local.com

root@llovizna:/home/pradeeban# sudo /opt/tyk-dashboard/install/bootstrap.sh dashboard.tyk-local.com

Creating Organisation

ORGID: 579794145ee8571e46000001

Adding new user

USER AUTH: 6936a780a8e448fd73e3d7ef64eb8059

NEW ID: 5797941458149ac06a14e801

Setting password

DONE

Login at http://dashboard.tyk-local.com/

User: c9og13fnc8@default.com

Pass: test123

Tyk Identity Broker (TIB)

In order to enable TIB, you need to do some more steps.

In /opt/tyk-dashboard/tyk_analytics.conf

```
"identity_broker": {
    "enabled": true,
    "host": {
        "connection_string": "http://localhost:3010",
        "secret": "9348938451234912381923814 86djfhr87234827348"
    }
},
```

Make sure the shared secrets match in tyk-analytics.conf and tib.conf of TIB. https://tyk.io/docs/tyk-dashboard-v1-0/configuration/

```
{ "Secret": "test-secret",
    "HttpServerOptions": {
        "UseSSL": false,
        "CertFile": "./certs/server.pem",
        "KeyFile": "./certs/server.key"
    },
        "BackEnd": {
            "Name": "in_memory",
        "ProfileBackendSettings": {},
                "IdentityBackendSettings": {
        "Hosts" : {
            "localhost": "6379"
        },
        "Password": "",
        "Database": 0,
        "EnableCluster": false,
        "MaxIdle": 1000,
        "MaxActive": 2000
```

```
        }
        },
        "TykAPISettings": {
        "GatewayConfig": {
            "Endpoint": "http://localhost",
            "Port": "8080",
            "AdminSecret": "12345"
        },
        "DashboardConfig": {
            "Endpoint": "http://dashboard.tyk-local.com",
            "Port": "80",
            "AdminSecret": "12345"
        }
    }
}
```


Enable TIB in the Tyk Analytics Configurations

Modify the TIB configurations in /opt/tyk-dashboard/tyk_analytics.conf as below:


```
    "identity_broker": {
        "enabled": true,
        "host": {
            "connection_string": "http://127.0.0.1:3010",
            "secret": "test-secret"
        }
    },
```


Enable Email Notifications in the Tyk Analytics Configurations


```
    "email_backend": {
        "enable_email_notifications": true,
        "code": "",
        "settings": null,
        "default_from_email": "",
        "default_from_name": ""
    },
```


Execute TIB

You will have to restart everything after configuring the TIB.

sudo service tyk-pump restart

sudo service tyk-dashboard restart

sudo service tyk-gateway restart

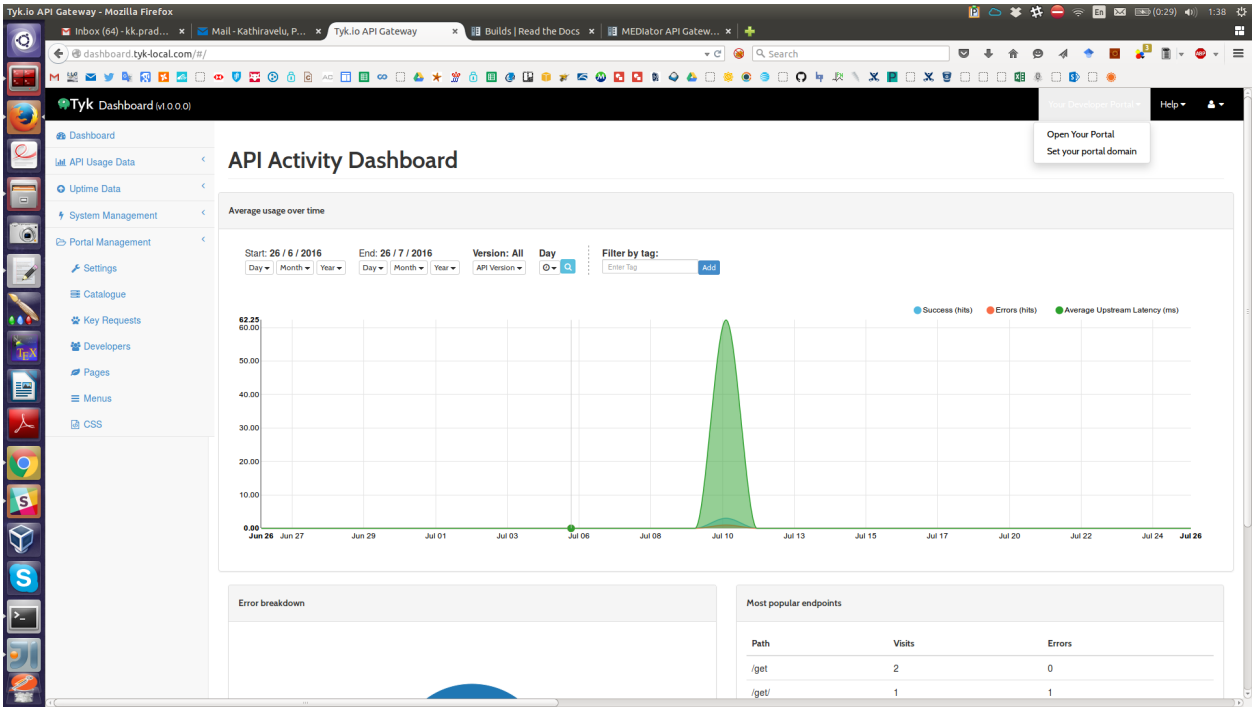
From the TIB home directory, /tib

Tyk Developer Portal

Now you are good to log in to the Tyk Dashboard with the given credentials.

Set Portal domain

portal.tyk-local.com

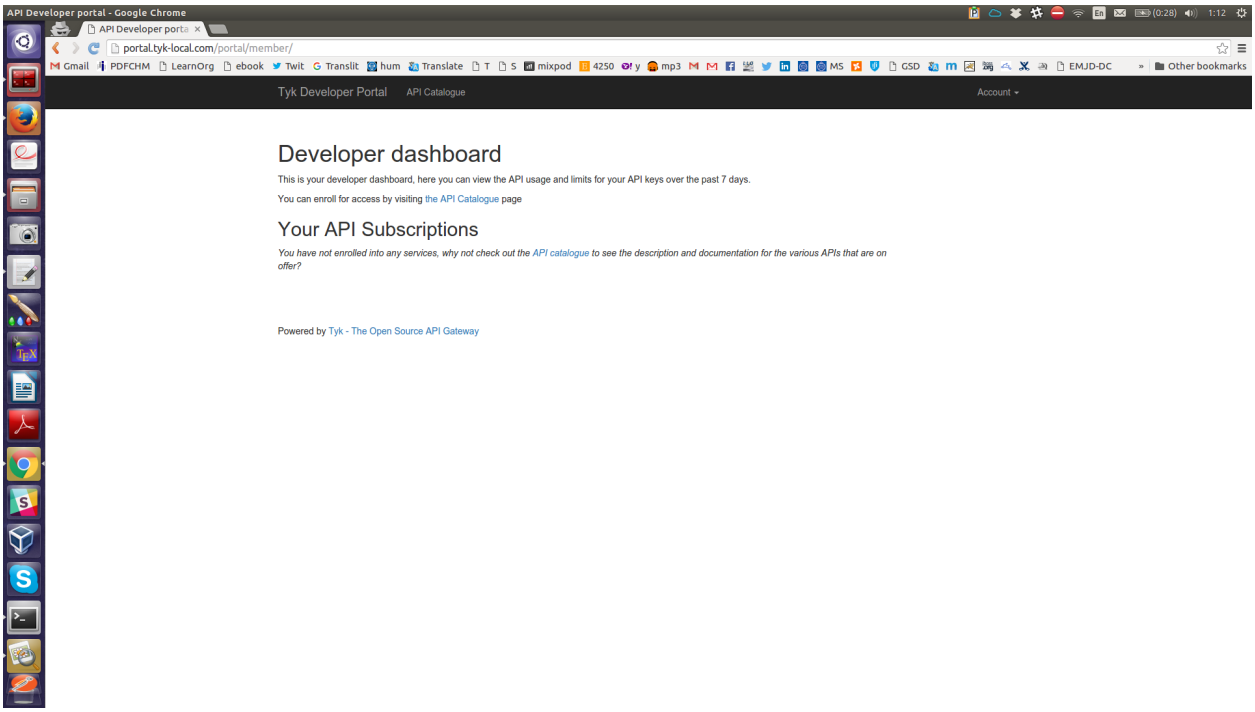


This would give a message, “CNAME updated”

Now if you access http://portal.tyk-local.com/portal/ it would say “Home page not found”. This is expected as you have not set up the portal and the welcome page yet.

Follow the below document to get this set up.

https://tyk.io/docs/tyk-api-gateway-v1-9/tutorials/set-up-your-portal/#step-6-set-your-portal-hostname:b4a940a28f68aca0b7fe0e28e62b2736



Once you have set this up, your portal is ready for the developers to register, log in, and consumes the APIs defined in the API catalogue.

Find the logs

sudo tail -f /var/log/upstart/tyk-dashboard.log

sudo tail -f /var/log/upstart/tyk-gateway.log

sudo tail -f /var/log/upstart/tyk-pump.log

Have feedback or corrections? Email us at pkathi2@emory.edu.

4.4 Installing MEDIator on CentOS

4.4.1 Installing the Core Dependencies

Install wget

$ yum install wget

Install Oracle JDK 8

cd /opt

sudo wget –no-cookies –no-check-certificate –header "Cookie: gpw_e24=http%3A%2F%2Fwww.oracle.com%2F; oraclelicense=accept-securebackup-cookie" "http://download.oracle.com/otn-pub/java/jdk/8u25-b17/jdk-8u25-linux-x64.tar.gz"

sudo tar xvf jdk-8u25-linux-x64.tar.gz

sudo chown -R root: jdk1.8.0_25

export JAVA_HOME=/opt/jdk1.8.0_25/

export PATH=$JAVA_HOME/bin:$PATH

Install Maven 3

wget http://mirror.cc.columbia.edu/pub/software/apache/maven/maven-3/3.0.5/binaries/apache-maven-3.0.5-bin.tar.gz

sudo tar xzf apache-maven-3.0.5-bin.tar.gz -C /usr/local

cd /usr/local

sudo ln -s apache-maven-3.0.5 maven

sudo vi /etc/profile.d/maven.sh

export M2_HOME=/usr/local/maven

export PATH=${M2_HOME}/bin:${PATH}

Install git

sudo yum install git

4.4.2 Configure MEDIator

git clone https://pradeeban@bitbucket.org/BMI/datareplicationsystem.git

cd datareplicationsystem

4.4.3 Execute the MEDIator web app

mvn package

sh modules/repl-server/target/bin/webapp

with nohup

nohup sh modules/repl-server/target/bin/webapp > nohup.out &

To view the logs

tail -f nohup.out

Go to http://localhost:<EMBEDDED_TOMCAT_PORT>/ using your browser.

4.4.4 Execute with multiple data sources

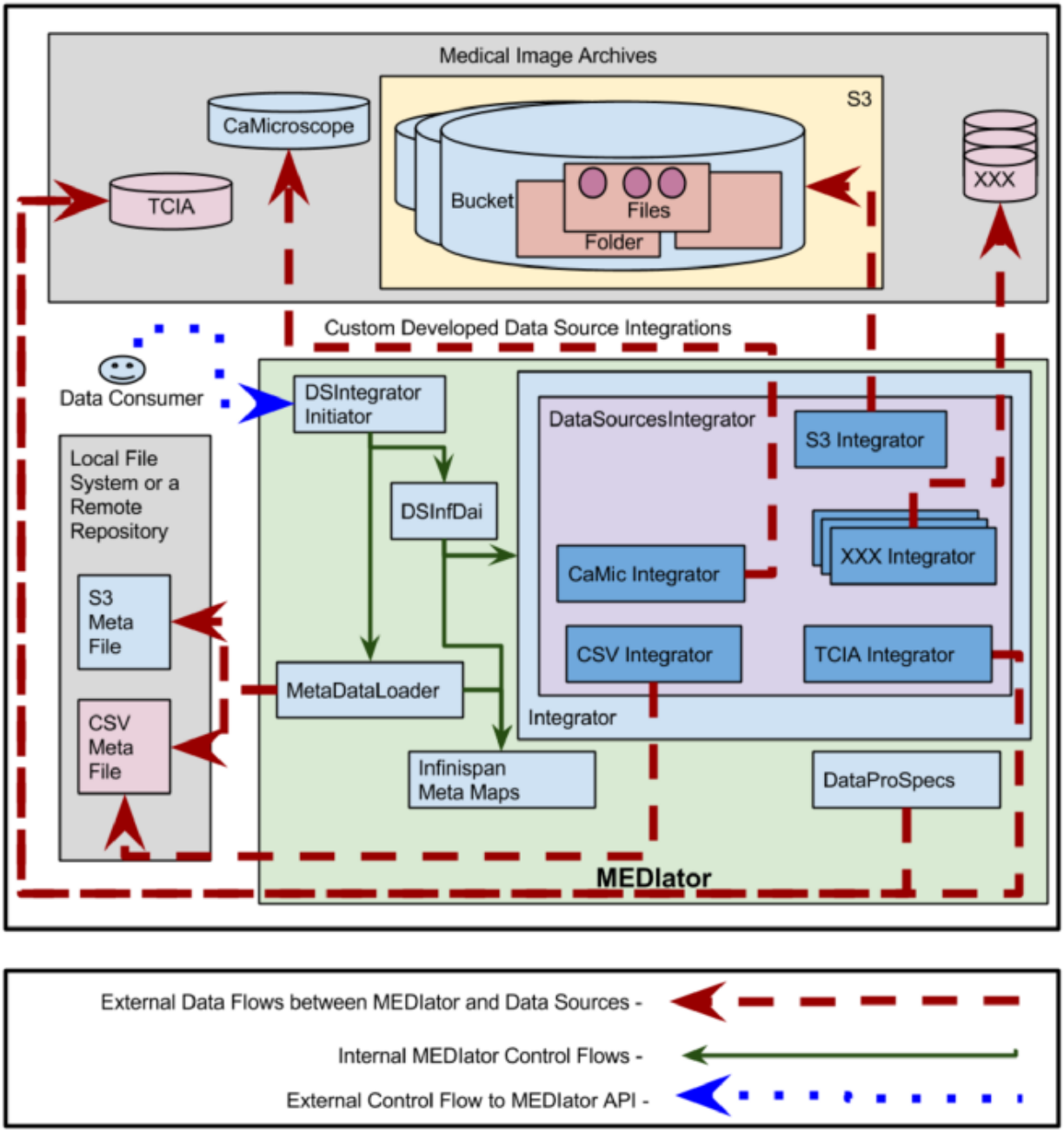
java -classpath lib/repl-server-1.0-SNAPSHOT.jar:lib/*:conf/ edu.emory.bmi.datarepl.ds_impl.DSIntegratorInitiator

---

Have feedback or corrections? Email us at pkathi2@emory.edu.

4.5 Data Sources

MEDIator version 1.0-SNAPSHOT has extensively been developed for The Cancer Imaging Archive (TCIA), while maintaining relevant interfaces for extension to the otehr data sources.



4.5.1 Integration with Medical Data Sources

Clinical data is deployed in multiple data sources such as TCIA, caMicroscope, and Amazon S3. The above figure depicts the deployment of the system with multiple medical data sources. A set of clinical data was uploaded to S3, where the metadata mapping of patientID ding{213} fileName was available as a CSV file. Similarly CSV file depicting clinical information is available, as multiple properties against the UUID, such as patient ID. These files are parsed and stored

> into metadata maps in Infinispan. The CSV files containing metadata or $resource ID ding{213} fileName$ mapping are stored locally in the file system or in a remote repository.

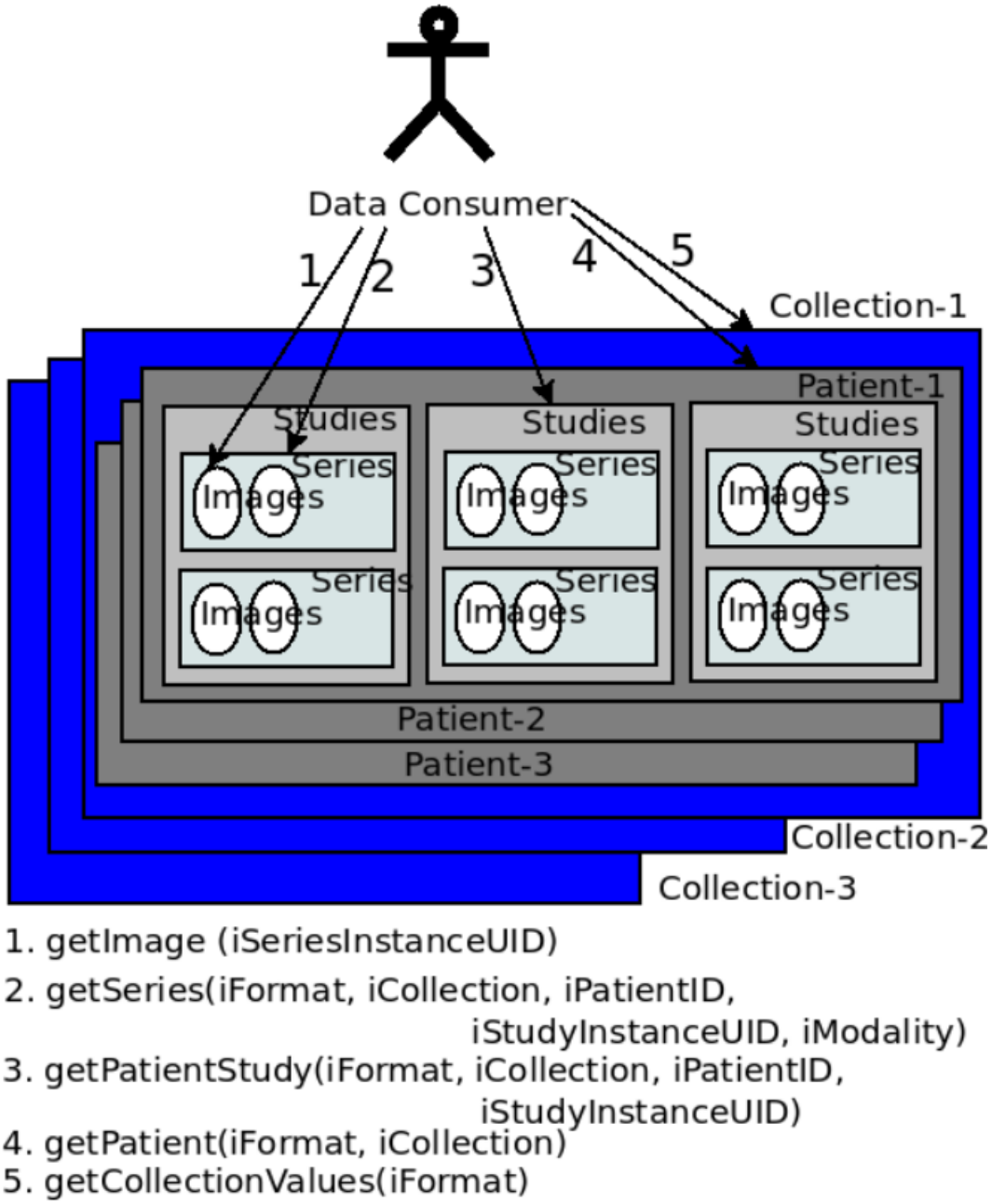
Each data source is connected to MEDIator by implementing a class that extends Integrator interface. DatasourcesInte-

grator, an abstract class provides the common features for the data sources integration. It is extended by CsvIntegrator, CaMicIntegrator, S3Integrator, and TciaIntegrator which respectively function as the integrators for CSV, caMicroscope, Amazon S3, and TCIA. MetaDataLoader loads the CSV files and stores them into Infinispan maps, against the ID, such as the patient ID.

A sub class of InfDataAccessIntegration, DSInfDai holds the instances of map to store all the metadata. DSIntegratorInitiator invokes the instances of DSInfDai and MetaDataLoader to parse the metadata, and store the instances into the respective maps.

4.5.2 Medical Image Archives

Medical images are stored in specific data sources such as TCIA that defines a schema for the metadata, or can be stored in a general-purpose data source such as Amazon S3, with user-enforced schema. TCIA public API provides methods to retrieve the images and metadata of different granularity. These methods are invoked by the public APIs such as the REST API. An initial search on TCIA may contain parameters such as modality, in addition to collection name, patient ID, study instance ID, and series instance UID. Each of the searches in TCIA returns the output in a finer granularity.



TCIA is made of multiple collections. Collections are queried and raw image data and metadata are downloaded by users. While some collections are open for public access, some are protected. Downloads create a considerable load in the servers, with unexpected spikes resulting from automated bulk downloads. The National Lung Screening Trial (NLST) 2 has a mean download rate of 0.5 TB/month, with a spike of 5.6 TB in June 2013, during the last 3 years.

4.5.3 Further Resources

Resources

TCIA REST API

[1] TCIA Programmatic Interface (REST API) Usage Guide - https://wiki.cancerimagingarchive.net/display/Public/

TCIA+Programmatic+Interface+%28REST+API%29+Usage+Guide

[2] TCIA on Mashape - https://market.mashape.com/tcia/the-cancer-imaging-archive

[3] TCIA REST API Client - https://github.com/nadirsaghar/TCIA-REST-API-Client

[4] TCIA on RapidAPI - https://rapidapi.com/tcia/api/The%20Cancer%20Imaging%20Archive

Infinispan - MongoDB Integration

[1] Using MongoDB as cache store - http://blog.infinispan.org/2013/06/using-mongodb-as-cache-store.html

[2] Infinispan MongoDB Cache Store - https://github.com/infinispan/infinispan-cachestore-mongodb

---

Have feedback or corrections? Email us at pkathi2@emory.edu.

Acknowledgements

- Google Summer of Code 2014
- NCI QIN: Resources for development and validation of Radiomic Analyses & Adaptive Therapy — PI: Prior, Sharma (UAMS, Emory)

Have feedback or corrections? Email us at pkathi2@emory.edu.

Have feedback or corrections? Email us at pkathi2@emory.edu.

CHAPTER 5

Citing MEDIator

If you have used MEDIator in your research, please cite the below papers:

[1] Kathiravelu, P. & Sharma, A. (2015). **MEDIator: A Data Sharing Synchronization Platform for Heterogeneous Medical Image Archives.** In *Workshop on Connected Health at Big Data Era (BigCHat'15), co-located with 21st ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD 2015)*. Aug. 2015. ACM. 6 pages. http://doi.org/10.13140/RG.2.1.3709.4248

[2] Kathiravelu, P. & Sharma, A. (2016). **SPREAD - System for Sharing and Publishing Research Data**. In *Society for Imaging Informatics in Medicine Annual Meeting (SIIM 2016)*. June 2016. http://c.ymcdn.com/sites/siim.org/resource/resmgr/siim2016abstracts/Research_Kathiravelu.pdf

Have feedback or corrections? Email us at pkathi2@emory.edu.