Welcome

Thank you for taking an interest in the Open Client Registry (OpenCR)! This is a community project and meant for others to adopt to their use cases as they wish.

> **Tip**
>
> Regardless if you're just curious, an implementer, or a developer, please read the this Overview tab first and then the User Manual. We've kept them short.

What is OpenCR?

OpenCR is an open source and standards-based client registry. A client registry facilitates the exchange of patient information between disparate systems. A client registry holds patient identifers and may include patient demographic information. It is a necessary tool for public health to help manage patients, monitor outcomes, and conduct case-based surveillance.

A client registry sits within a health information exchange (HIE). An HIE is used to safely and effectively exchange information. A critical component of an HIE are registries, such as those to manage a shared, canonical facility list, practitioners, and patients.

What does OpenCR do?

OpenCR is offers the ability to:

- Assign and look-up unique identifiers,
- Allow connections from diverse point of service (POS) systems, such as lab systems and electronic medical record (EMR) systems, that can submit messages in FHIR,
- Configure decision rules around patient matching.

🟧 **Caution**

This implementation does not allow point-of-service systems to get patient demographic information stored in the Client Registry. This is also not a Shared Health Record, nor does it contain patient clinical data.

Use Cases

The Client Registry is one component in a more complex HIS architecture needed to accomplish important use cases, such as:

- **Deduplicating patients**: Sometimes patients have multiple diagnostic results stored within a POS. The Client Registry will link patients based on configurable decision rules so multiple test results for the same patient can be found.
- **Tracking patients lost to clinical care**: EMRs are often not interoperable with one another, resulting in difficulty tracking patients as they move between facilities to seek care. A Client Registry will help data managers to track patients, decreasing instances of duplicate and incomplete records, patients LTFU, and sub-optimal care.

🟧 **Caution**

The Client Registry is not deduplicating or even touching patient clinical and demographic records within point-of-service systems. Instead, it provides a way to enable use cases like deduplication - which must be an external process.

About

OpenCR was developed by IntraHealth International with support from PEPFAR through the USAID MEASURE Evaluation Project. Technical direction was provided by CDC.

Last update: April 13, 2020

Matching Process

This is an overview of the matching process.

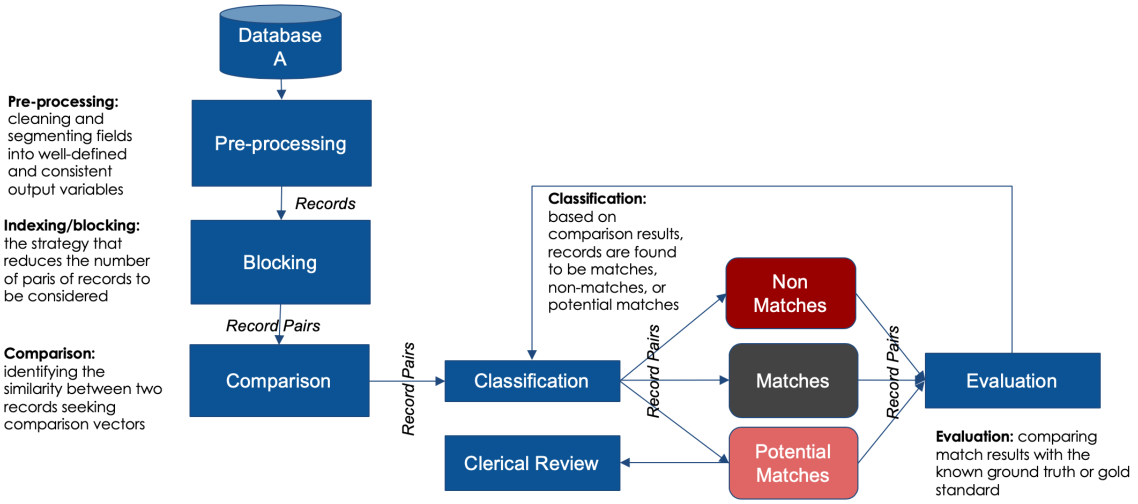
Generic Matching Process

It is helpful to look at a generic matching process first, and then move to to see where OpenCR fits.

This diagram is reproduced from Christen, Peter, 2012, "Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection"

The diagram is an example of a deduplication process with only one data source.

- **Database**: The flow begins on the top left at 'Database'.
- **Preprocessing**: Data from the database source is preprocessed. This means cleaning the data before submission of errors in date formats, data entry mistakes, biologically implausible values, and similar.
- **Blocking**: This means using filters to be more efficient in queries. For example, filtering on the birth year of 1960 reduces the amount of searching that has to be done because only 1960 is used.
- **Comparison**: Algorithms compare pairs of records.
- **Classification**: Records are classified as matches, non-matches, or potential matches.
- **Clerical review**: For records that are potential matches, they may be reviewed individually.
- **Evaluation**: This process is a way to understand the matching performance against a known baseline. It is not necessarily built into the client registry but may be conducted using other tools.



Pre-processing: cleaning and segmenting fields into well-defined and consistent output variables

Indexing/blocking: the strategy that reduces the number of paris of records to be considered

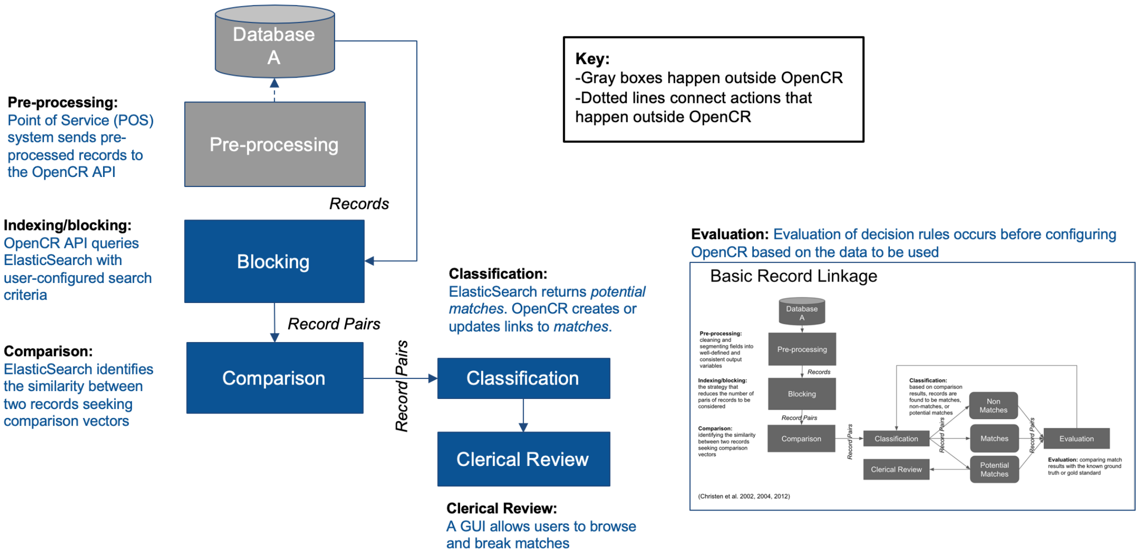
Comparison: identifying the similarity between two records seeking comparison vectors

Classification: based on comparison results, records are found to be matches, non-matches, or potential matches

Evaluation: comparing match results with the known ground truth or gold standard

OpenCR performs much of the functionality in the matching process.

- **Database and preprocessing**: The database and cleaning of records is done outside of OpenCR.
- **Comparison and classification**: In production, ElasticSearch is used for these processes. ElasticSearch is a part of OpenCR.
- **Clerical review**: There is a UI for viewing and breaking matches.
- **Evaluation**: This process is conducted externally with other tools, it is not provided as a feature set in OpenCR.



Key:
-Gray boxes happen outside OpenCR
-Dotted lines connect actions that happen outside OpenCR

Pre-processing: Point of Service (POS) system sends pre-processed records to the OpenCR API

Indexing/blocking: OpenCR API queries ElasticSearch with user-configured search criteria

Comparison: ElasticSearch identifies the similarity between two records seeking comparison vectors

Classification: ElasticSearch returns *potential matches*. OpenCR creates or updates links to *matches*.

Clerical Review: A GUI allows users to browse and break matches

Evaluation: Evaluation of decision rules occurs before configuring OpenCR based on the data to be used

Last update: April 13, 2020

Use Cases

At its core, OpenCR provides a unique identifier (UID) that also links to all other already matched records from submitting systems. This means that OpenCR stores an identifier from submitting systems so that it can uniquely identify according to however the submitting systems store their records, but it also produces a UID for the entire domain using the service.

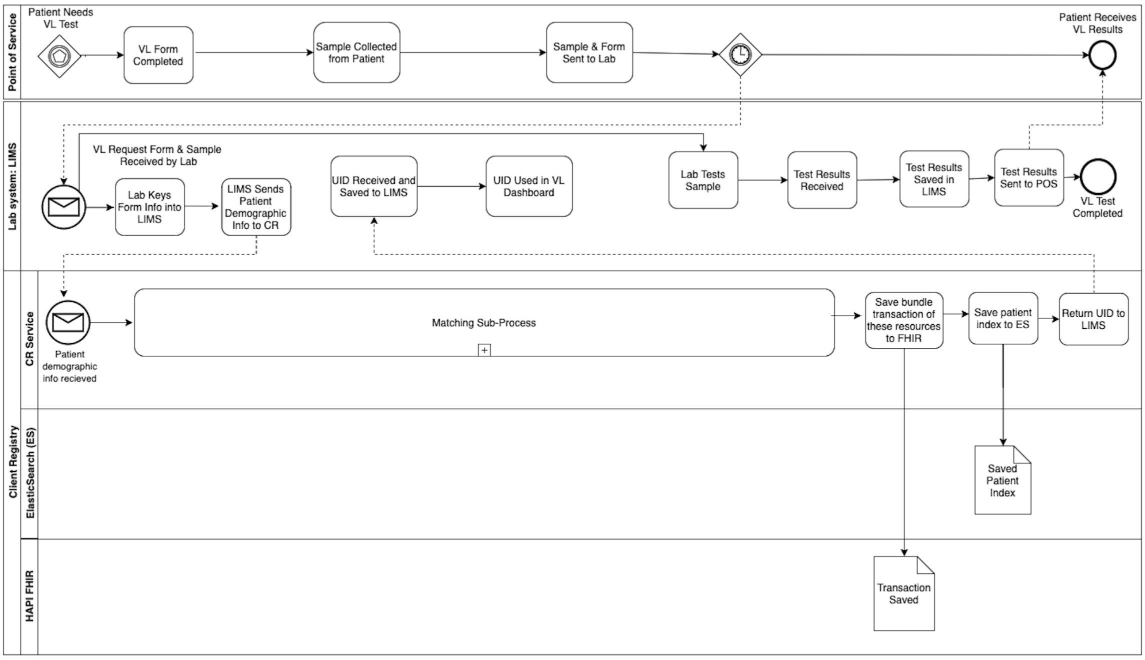
Several workflows are supported out-of-the-box depending on the POS-OpenCR use case. For example:

- A specimen is received by a laboratory. Demographic data and requesting location data is entered into the LMIS. The LMIS queries OpenCR for a UID. OpenCR provides the UID if one did not exist and stores limited patient demographic information but **does not** store test results. A use that this enables (but OpenCR **does not** provide) is the ability to track persons lab results over time.
- A patient is registered at a clinic. The clinician recommends a viral load test. The specimen is sent for processing to the laboratory. OpenCR receives the UID and specimen and returns a diagnostic result that is then stored in the EMR.
- A patient is registered at a clinic and has been assigned a UID. In the course of their clinical encounter, a sentinel event occurs, triggering the EMR to send limited clinical information to the Health Information Exchange (HIE). The HIE sends the data to a data analysis warehouse for population analysis and case-based surveillance.

Warning

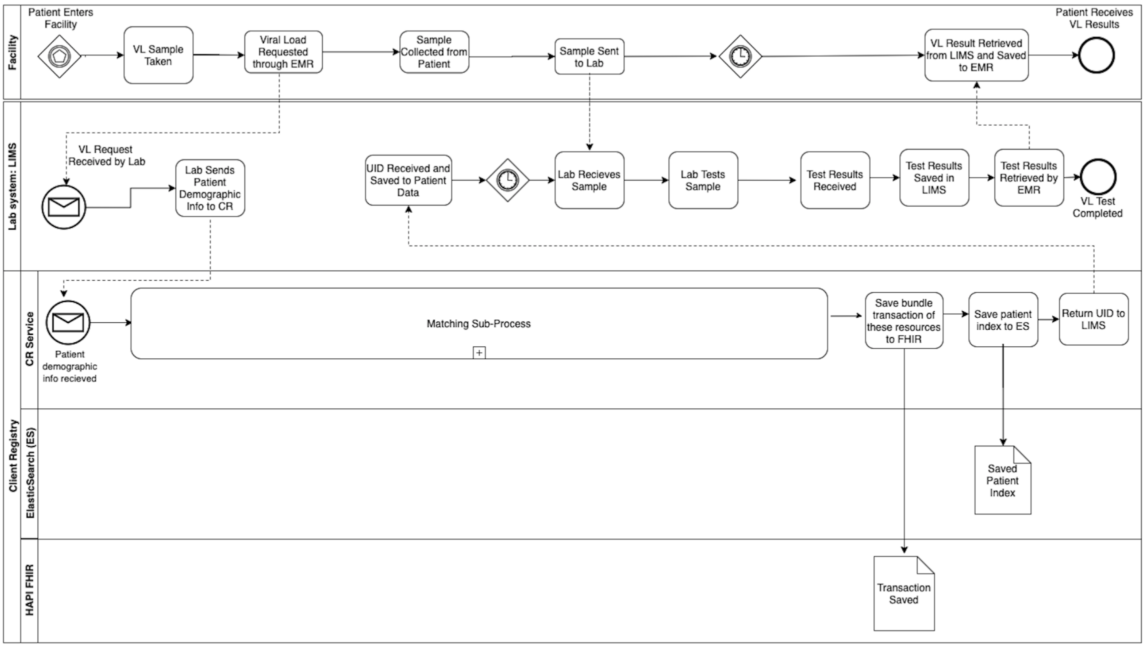
It is important to note that in the above workflows OpenCR does not store or provide clinical data. Such processes are external to OpenCR and must be separately created, governed, and enabled.

Viral Load Test Request (Paper)


Point of Service
Patient Needs VL Test
VL Form Completed
Sample Collected from Patient
Sample & Form Sent to Lab
Patient Receives VL Results
Lab system: LIMS
VL Request Form & Sample Received by Lab
Lab Keys Form Info into LIMS
LIMS Sends Patient Demographic Info to CR
UID Received and Saved to LIMS
UID Used in VL Dashboard
Lab Tests Sample
Test Results Received
Test Results Saved in LIMS
Test Results Sent to POS
VL Test Completed
Client Registry
CR Service
Patient demographic info recieved
Matching Sub-Process
Save bundle transaction of these resources to FHIR
Save patient index to ES
Return UID to LIMS
ElasticSearch (ES)
Saved Patient Index
HAPI FHIR
Transaction Saved

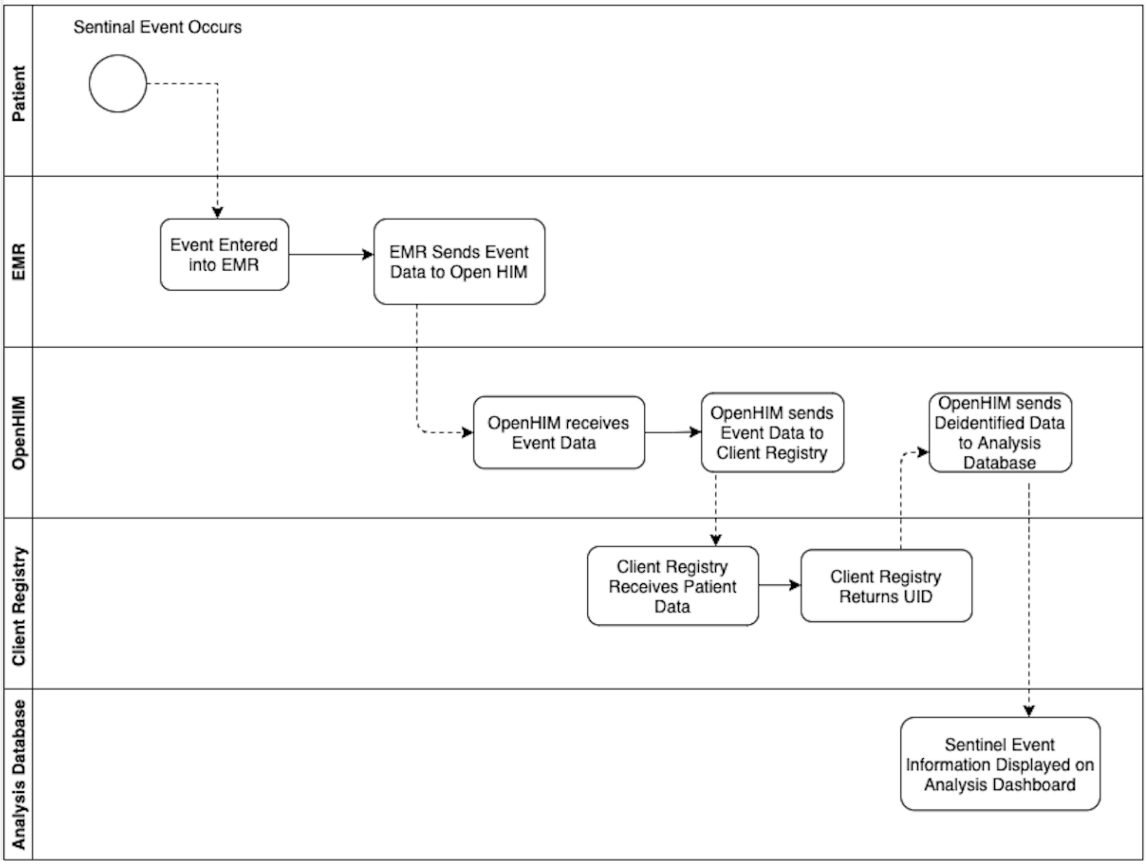
A plasma specimen is received by a laboratory for HIV viral load testing. Demographic data and requesting location data is entered into the LMIS. The LMIS queries OpenCR for a UID. OpenCR provides the UID if one did not exist and stores limited patient demographic information but **does not** store test results. A use that this enables (but OpenCR **does not** provide) is the ability to track persons lab results over time.

Viral Load Test Request (EMR)


Facility
Patient Enters Facility
VL Sample Taken
Viral Load Requested through EMR
Sample Collected from Patient
Sample Sent to Lab
VL Result Retrieved from LIMS and Saved to EMR
Patient Receives VL Results
Lab system: LIMS
VL Request Received by Lab
Lab Sends Patient Demographic Info to CR
UID Received and Saved to Patient Data
Lab Recieves Sample
Lab Tests Sample
Test Results Received
Test Results Saved in LIMS
Test Results Retrieved by EMR
VL Test Completed
Client Registry
CR Service
Patient demographic info recieved
Matching Sub-Process
Save bundle transaction of these resources to FHIR
Save patient index to ES
Return UID to LIMS
ElasticSearch (ES)
Saved Patient Index
HAPI FHIR
Transaction Saved

A patient is registered at a clinic. The clinician recommends an HIV viral load test. The plasma specimen is sent for processing to the laboratory. OpenCR receives the UID and specimen and returns a diagnostic result that is then stored in the EMR.

Case-Based Surveillance


Patient
EMR
OpenHIM
Client Registry
Analysis Database
Sentinal Event Occurs
Event Entered into EMR
EMR Sends Event Data to Open HIM
OpenHIM receives Event Data
OpenHIM sends Event Data to Client Registry
OpenHIM sends Deidentified Data to Analysis Database
Client Registry Receives Patient Data
Client Registry Returns UID
Sentinel Event Information Displayed on Analysis Dashboard

A patient is registered at a clinic and has been assigned a UID. In the course of their clinical encounter, a sentinel event occurs, triggering the EMR to send limited clinical information to the Health Information Exchange (HIE). The HIE sends the data to a data analysis warehouse for population analysis and case-based surveillance.

Last update: April 13, 2020

OpenMRS MPI Client

Demo

This demo shows how a patient is registered in OpenMRS with the MPI module (see below) and how that patient shows up in OpenCR where matches can be viewed and broken.

Your browser does not support the video tag.

Installing the Module

OpenCR can be setup to work with OpenMRS with the MPI module. There is a reference implementation for 2.x and legacy OpenMRS to connect to the CR. The branch must be chosen correctly.

- 2.x support
- Legacy support

Last update: April 13, 2020

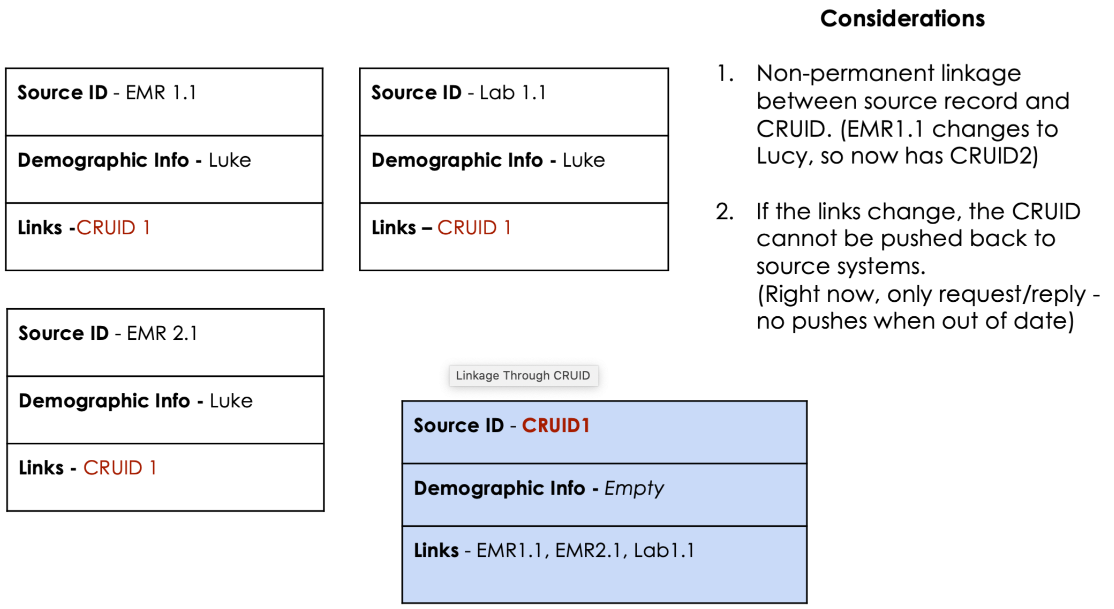
Unique Identifiers

OpenCR creates unique idenfiers (CRUIDs) which provide the key for record linkage.

Example

In this example, there are three records for one person, Luke, and a CRUID has been assigned.

- Luke has records one EMR, another EMR, and a lab system.
- Each of his records has some demographic data.
- OpenCR has also created a CRUID - "CRUID-1", for Luke.
- There is a record for CRUID-1. It does not have demographic data, but it does have links to all of Luke's records.
- Once CRUID-1 has been assigned to Luke's records, his CRUID-1 is also linked to the original records.



Considerations

1. Non-permanent linkage between source record and CRUID. (EMR1.1 changes to Lucy, so now has CRUID2)
2. If the links change, the CRUID cannot be pushed back to source systems.
(Right now, only request/reply - no pushes when out of date)

Source ID - EMR 1.1

Demographic Info - Luke

Links -CRUID 1

Source ID - Lab 1.1

Demographic Info - Luke

Links – CRUID 1

Source ID - EMR 2.1

Demographic Info - Luke

Links - CRUID 1

Source ID - **CRUID1**

Demographic Info - *Empty*

Links - EMR1.1, EMR2.1, Lab1.1

Last update: April 13, 2020

Intro to Algorithms

Algorithms can be deterministic, fuzzy, probabilistic, or a combination of approaches; most include some element of automatic matching plus human review. No perfect algorithm currently exists [1]. Typically, the creator(s) of the client registry will agree on the appropriate threshold to determine a "match" and this may need to be recalibrated over time.

Approaches

Types of matching approaches:

- **Deterministic**. This can be thought of enforcing exact matches. This type of matching performs poorly when the quality of the data is low and/or the discriminatory power of the variable used for matching is low (e.g. sex is a poor discriminatory variable). If exact matching is enforced, the data scientist should verify that other fields also match using a combination of approaches, as needed.
- **Fuzzy**. The fuzzy match method relies on converting items to their core components (e.g. phonetics) to determine matching rather than relying on probabilities. This method's strength is addressing typing errors but remains imperfect in cases where words are similar, but are not the same (e.g. there, their). Its efficacy is unproven in most languages other than English.
- **Probabilistic**. This is a method that assigns a match probability to a pair of records. The most common method (Fellegi-Sunter) checks a number of fields and sums up the probabilities that each field is a match. Higher scores indicate a match. Recent research has been directed at improving the FS method [2].

Matching algorithms are evaluated based on their sensitivity and specificity in detecting matches. One caveat to remember is to check whether the sensitivity and specificity reported are with or without human review. Generally, it is better to minimize the chances of a false positive; however, as false positives go towards zero false negatives increase and the balance must be maintained.

Types of Deterministic Matching

- **Field-based (exact) matching**. Typically fields are compared to each other across data sets and a result is a match or non-match, thus deterministic.

Types of Fuzzy Matching

Methods that assist in matching based on phonetic transformations to remedy spelling errors:

- **Soundex** indexes names by sound as pronounced in English. The Soundex encoding includes the first letter of a surname followed by 3 digits. This method relies on correctly recording the first letter of a name.
- **Metaphone** indexes words by their English pronunciation.
- **Double Metaphone** implements Metaphone but allows for two encodings to be returned for a word rather than constraining it to one choice. It works well with names of various origins, but the lettering must be English.
- **Metaphone 3** is the next generation of Double Metaphone and is a commercial product.

Methods that compare string similarity:

- **Levenshtein edit distance** is a measure of the number of edits (insertions and deletions) that would be required to change one string to another.
- **Jaro-Winkler** comparator/distance is similar to Levenshtein, but it gives more weight to strings that match on the beginning of the string. 0 is an exact match. Lower scores are better than higher scores. If the beginning of the string is misspelled, this will not work as well.
- **Longest common subsequence** matches subsequences within strings. Subsequences do not need to match positions.

Types of Probabilistic Matching

The most widely used probabilistic matching method is the **Fellegi-Sunter method** [3]. In this method, each field is weighted by its discriminatory power and data quality; could be manually or via model training [4]. The algorithm checks each field and decides whether the field matches between the pairs or not. If the fields match, the weight is added to the final score. If the fields do not match, the weight is subtracted from the final score. If the sum of the weights is above a threshold, they are considered a match. Possible matches can also be produced and these would require human review.

One drawback is that this method assumes that no fields are correlated with each other or that they are not conditionally dependent. The assumption of conditional independence does not always hold and can lead to suboptimal results which is the biggest limitation of this method [5]. For example, there are more women named Barbara than men named Barbara. First name and sex are not independent.

Extensions to the F-S method include:

- The **approximate comparator extension** (ACE) [6] which includes string matching strategies prior to performing the weighting and scoring.

Alternatives to F-S include (see video):

- **GHC Scaling Algorithm** [7] is an order of magnitude faster than F-S. Calculates underlying weights and doesn't rely on independence assumption.

Speed

Methods to reduce the number of comparisons and speed up the process:

- **Blocking**. This is like sorting socks by color before trying to match them. It speeds up the process by reducing the number of pairs that need to be checked.

1. McFarlane, T. D., Dixon, B. E., & Grannis, S. J. (2016). Client registries: identifying and linking patients. In Health Information Exchange (pp. 163-182). Academic Press. ↩

2. Li, X., Xu, H., Shen, C., & Grannis, S. (2018). Automated linkage of patient records from disparate sources. Statistical methods in medical research, 27(1), 172-184. DuVall, Scott L., Richard A. Kerber, and Alun Thomas. "Extending the Fellegi–Sunter probabilistic record linkage method for approximate field comparators." Journal of biomedical informatics 43.1 (2010): 24-30. ■
3. McFarlane, T. D., Dixon, B. E., & Grannis, S. J. (2016). Client registries: identifying and linking patients. In Health Information Exchange (pp. 163-182). Academic Press. ■
4. DuVall, Scott L., Richard A. Kerber, and Alun Thomas. "Extending the Fellegi–Sunter probabilistic record linkage method for approximate field comparators." Journal of biomedical informatics 43.1 (2010): 24-30. ■
5. Li, X., Xu, H., Shen, C., & Grannis, S. (2018). Automated linkage of patient records from disparate sources. Statistical methods in medical research, 27(1), 172-184. ■
6. DuVall, Scott L., Richard A. Kerber, and Alun Thomas. "Extending the Fellegi–Sunter probabilistic record linkage method for approximate field comparators." Journal of biomedical informatics 43.1 (2010): 24-30. ■
7. Goldstein, H., Harron, K., & Cortina-Borja, M. (2017). A scaling approach to record linkage. Statistics in medicine, 36(16), 2514-2521. ■

Last update: April 13, 2020

Supported Algorithms

A number of algorithms are supported using ElasticSearch with the analysis-phonetic plugin and the OpenCR Service (alone).

Algorithm	OpenCR Service	ElasticSearch
Exact	Yes	Yes
Metaphone	Yes	Yes
Double-metaphone	Yes	Yes
Levenshtein	Yes	Yes
Damerau-Levenshtein	Yes	Yes
Jaro-Winkler	Yes	No
Soundex	Yes	Yes

For more advanced string similarity matching, the similarity-scoring plugin for ElasticSearch
can provide more features, and is based on the https://github.com/tdebatty/java-string-similarity library. The library is open source.

For more information, see the similarity-scoring repository:

Matcher Parameter for Query	Algorithm	Type	Normalized?
cosine-similarity	Cosine	similarity	yes
dice-similarity	Sorensen-Dice	similarity	yes
jaccard-similarity	Jaccard	similarity	yes
jaro-winkler-similarity	Jaro-Winkler	similarity	yes
normalized-lcs-similarity	Normalized Longest Common Subsequence	similarity	yes
normalized-levenshtein-similarity	Normalized Levenshtein	similarity	yes
cosine-distance	Cosine	distance	yes
damerau-levenshtein	Damerau-Levenshtein	distance	no
dice-distance	Sorensen-Dice	distance	yes
jaccard-distance	Jaccard	distance	yes
jaro-winkler-distance	Jaro-Winkler	distance	yes
levenshtein	Levenshtein	distance	no
longest-common-subsequence	Longest Common Subsequence	distance	no
metric-lcs	Metric Longest Common Subsequence	distance	yes
ngram	N-Gram	distance	yes
normalized-lcs-distance	Normalized Longest Common Subsequence	distance	yes
normalized-levenshtein-distance	Normalized Levenshtein	distance	yes
optimal-string-alignment	Optimal String Alignment	distance	no
qgram	Q-Gram	distance	no

Last update: April 23, 2020

Architecture

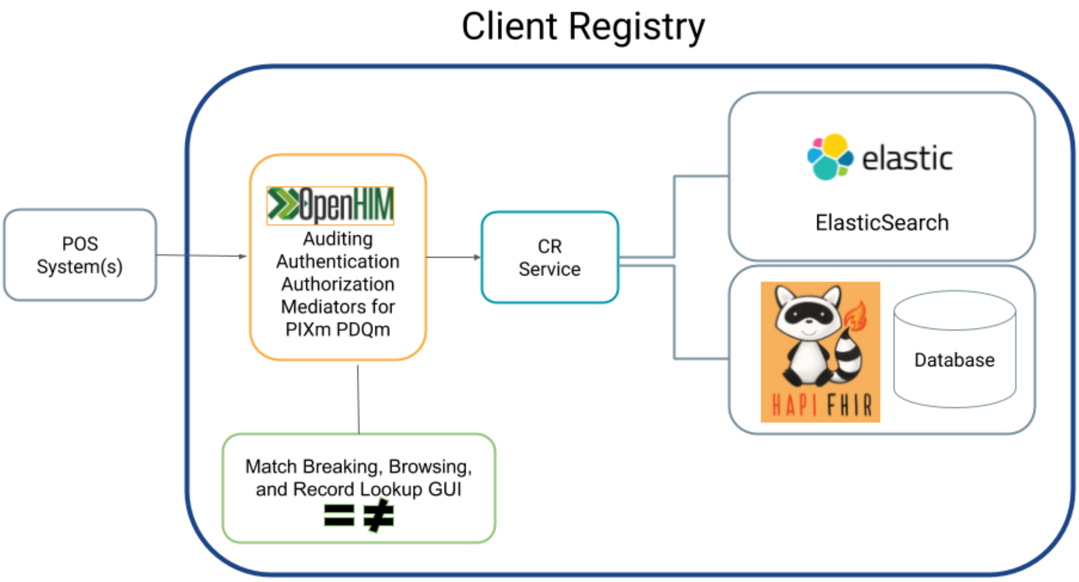
OpenCR is not one application, instead it's a set of applications that work together in the Open Health Information Exchange (OpenHIE) architecture to serve point-of-service systems, like EMRs, insurance mechanisms, and labs.

> **Note**
>
> This is not an OpenHIE product. The OpenHIE community of practice does not produce software products. Rather OpenHIE produces an architecture specification and is composed of a large, global community of practice around standards-based health information exchanges, particularly in low resource settings. Please join us!

The OpenCR architecture includes:

- The **OpenCR Service**: The API for managing queries, routing traffic to the components, and overall entrypoint. It is written in Node JS.
- The **HAPI FHIR Server**: HAPI is the reference FHIR server in Java and scalable into production environments.
- The **ElasticSearch**: Elasticsearch is a powerful search engine that is highly performant.
- An **optional UI** to view and break matches between records, and view matching histories (audit events).
- The **Open Health Information Mediator (OpenHIM)** (Optional): The OpenHIM is the entrypoint for POS systems, and includes authentication (are you who you say you are?), authorization (what roles do you have permission to fulfill?), and auditing of all transactions. OpenHIM is optional but the administrator must manage users and node access in some manner if not with OpenHIM.

Client Registry Production Architecture


Client Registry
POS System(s)
OpenHIM
Auditing Authentication Authorization Mediators for PIXm PDQm
CR Service
elastic
ElasticSearch
HAPI FHIR
Database
Match Breaking, Browsing, and Record Lookup GUI
=≠

Last update: April 13, 2020

Introduction

OpenCR is an open source and standards-based client registry. A client registry facilitates the exchange of patient information between disparate systems. A client registry holds patient identifers and may include patient demographic information. It is a necessary tool for public health to help manage patients, monitor outcomes, and conduct case-based surveillance.

A client registry sits within a health information exchange (HIE). An HIE is used to safely and effectively exchange information. A critical component of an HIE are registries, such as those to manage a shared, canonical facility list, practitioners, and patients.

What does OpenCR do?

OpenCR is offers the ability to:

- Assign and look-up unique identifiers,
- Allow connections from diverse point of service (POS) systems, such as electronic medical record (EMR) systems, that can submit messages in FHIR, and
- Configure decision rules around patient matching.

> **Caution**
>
> This implementation does not allow point-of-service systems to get patient demographic information stored in the Client Registry. This is also not a Shared Health Record, nor does it contain patient clinical data.

The process for a point-of-service system like an EMR to get a unique ID from the Client Registry is straightforward though it looks complicated at first.

1. A POS provides some demographic information to the Client Registry.

2. The Client Registry looks for an existing record matching that patient.
3. If there is an existing record, the Client Registry provides the unique ID back to the POS.
4. If there is not an existing record, the Client Registry makes a new one and provides a unique ID back to the POS.

As noted in the introduction, the Client Registry provides a unique identifier that also links to all other already matched records from submitting systems. This means that the Client Registry stores an identifier from submitting systems so that it can uniquely identify according to however the submitting systems store their records, but it also produces a UID for the entire domain using the service.

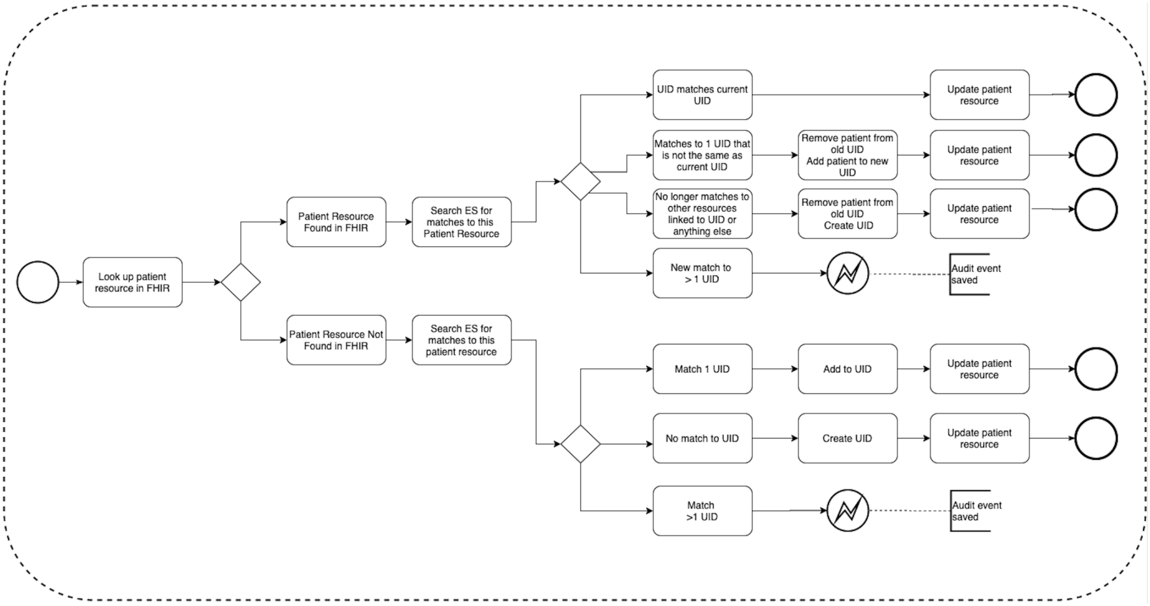
Warning

The below workflows the Client Registry does not store or provide clinical data. Such processes are external to the Client Registry and must be separately created, governed, and enabled.

Last update: April 13, 2020

Record Linkage Process

The below diagram shows how OpenCR performs record linkage after the matching process. The diagram begins with a source system submitting a request with patient demographic data in a FHIR message, as indicated by the circle on the left.



Look up patient resource in FHIR
Patient Resource Found in FHIR
Search ES for matches to this Patient Resource
UID matches current UID
Update patient resource
Matches to 1 UID that is not the same as current UID
Remove patient from old UID Add patient to new UID
Update patient resource
No longer matches to other resources linked to UID or anything else
Remove patient from old UID Create UID
Update patient resource
New match to > 1 UID
Audit event saved
Patient Resource Not Found in FHIR
Search ES for matches to this patient resource
Match 1 UID
Add to UID
Update patient resource
No match to UID
Create UID
Update patient resource
Match >1 UID
Audit event saved

After requests are submitted with demographic data, OpenCR reads the submitting system's ID of that patient. The Client Registry searches for that source system's ID in its records. This happens regardless if it is a new patient or update of existing patient.

When the submitting system's ID matches an existing record, the Client Registry updates the patient demographic information of that record with changes submitted. Once the update is complete, the existing record linkages may affected. This is because algorithms may not continue to link records as before because details have changed. Therefore, the Client Registry will pool all patients that were previously matched and break all the matches. The Client Registry will rerun matching algorithms again to see what matches are currently true matches of the patient. Then the Client Registry will be updated with the true matches given the changes in demographic data.

Another scenario is when the Client Registry searches and doesn't find anyone already with same submitting system's ID. If there is not existing match, the Client Registry runs the matching algorithms for existing patients who matches that patient and will provide record linkages with other records.

Requirements

In order for this process to work as expected, there are some requirements:

- Requests sent to the Client Registry must be made of FHIR messages. FHIR is a popular specification for accessing an API for providing data in health systems. Messages must support FHIR R4.
- Requests can only be received from trusted systems. See the security page in the Developers Manual for mode detail.

Last update: April 13, 2020

User Interface (CRUX)

The OpenCR User Interface (CRUX) is a key way to monitor the operation of OpenCR. With CRUX, you can:

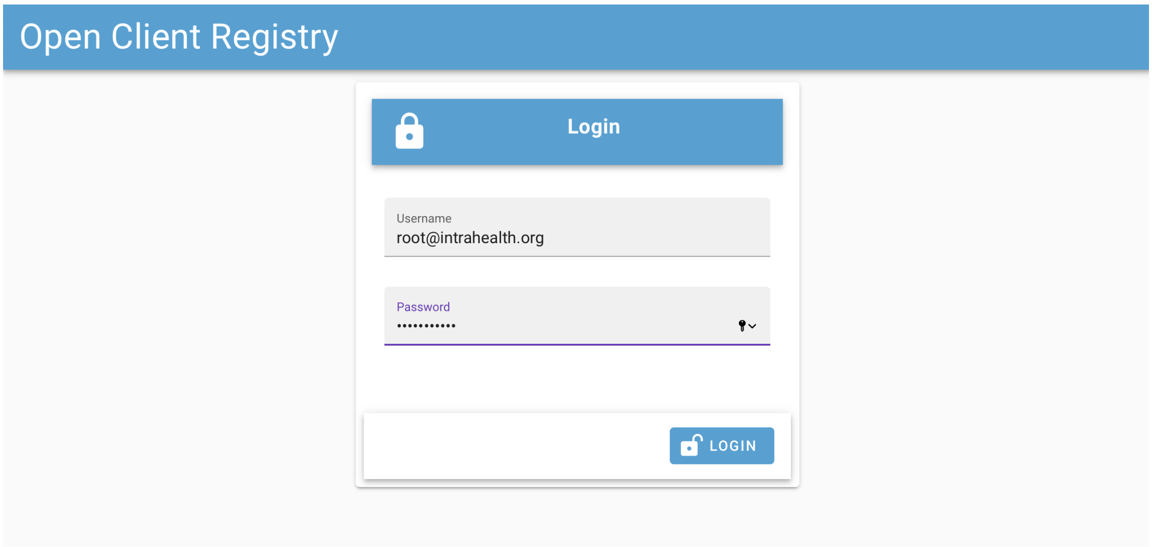
- View matches, break matches, revert broken matches
- Verifying FHIR messages are being processed correctly from submitting systems
- Validating matching is working as expected
- Perform deep inspection before putting it into production

Caution

You may have access to the CRUX of a system, and that's a good thing. The CRUX allows users to be able to view any break any match, which includes viewing demographic data from submitting systems. It should be secure and only authorized users would be able to access it.

Login

User must be added to the CRUX to be able to login.

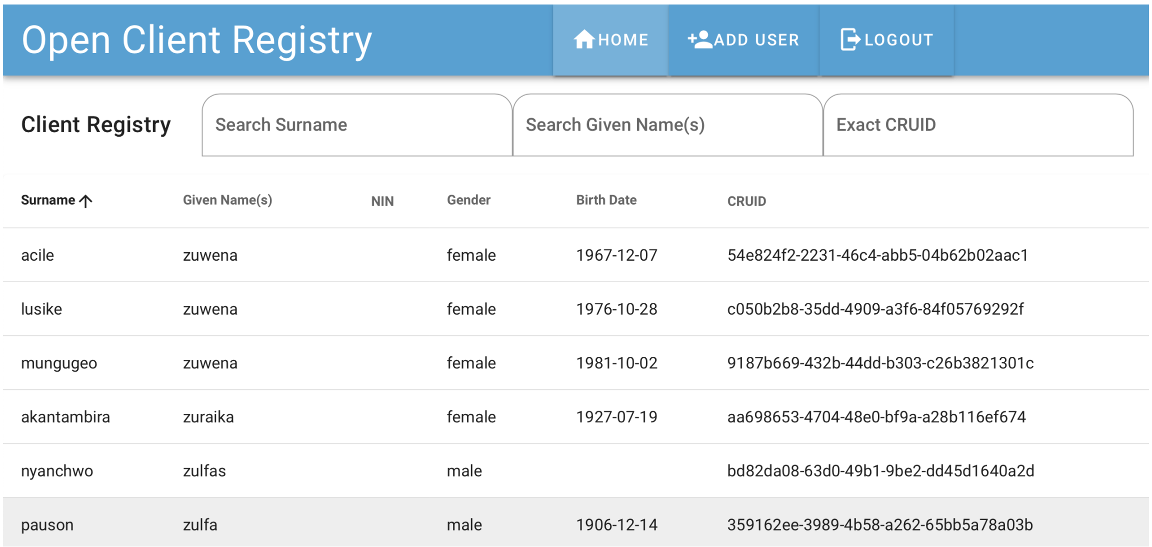


Username
root@intrahealth.org

Password
•••••••••••

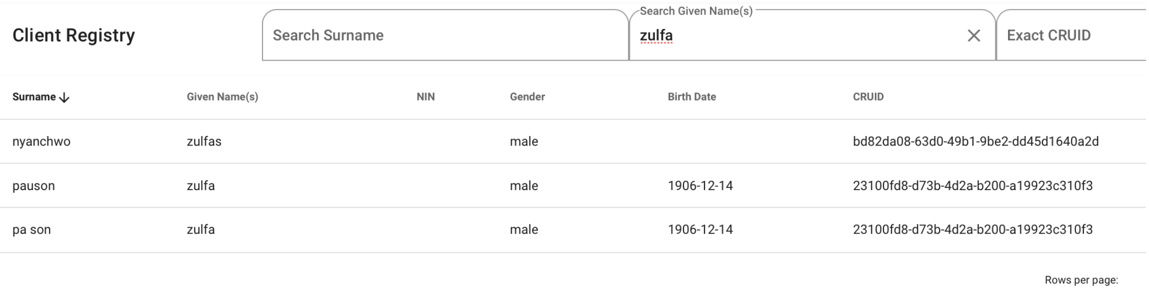
Landing Page

On landing inside CRUX, it display the records submitted in a row. These are individual records for POS that submit them.



Surname ↑	Given Name(s)	NIN	Gender	Birth Date	CRUID
acile	zuwena		female	1967-12-07	54e824f2-2231-46c4-abb5-04b62b02aac1
lusike	zuwena		female	1976-10-28	c050b2b8-35dd-4909-a3f6-84f05769292f
mungugeo	zuwena		female	1981-10-02	9187b669-432b-44dd-b303-c26b3821301c
akantambira	zuraika		female	1927-07-19	aa698653-4704-48e0-bf9a-a28b116ef674
nyanchwo	zulfas		male		bd82da08-63d0-49b1-9be2-dd45d1640a2d
pauson	zulfa		male	1906-12-14	359162ee-3989-4b58-a262-65bb5a78a03b

It is easy to search for records on fields. In the below example, there are two records submitted that share the same CRUID.



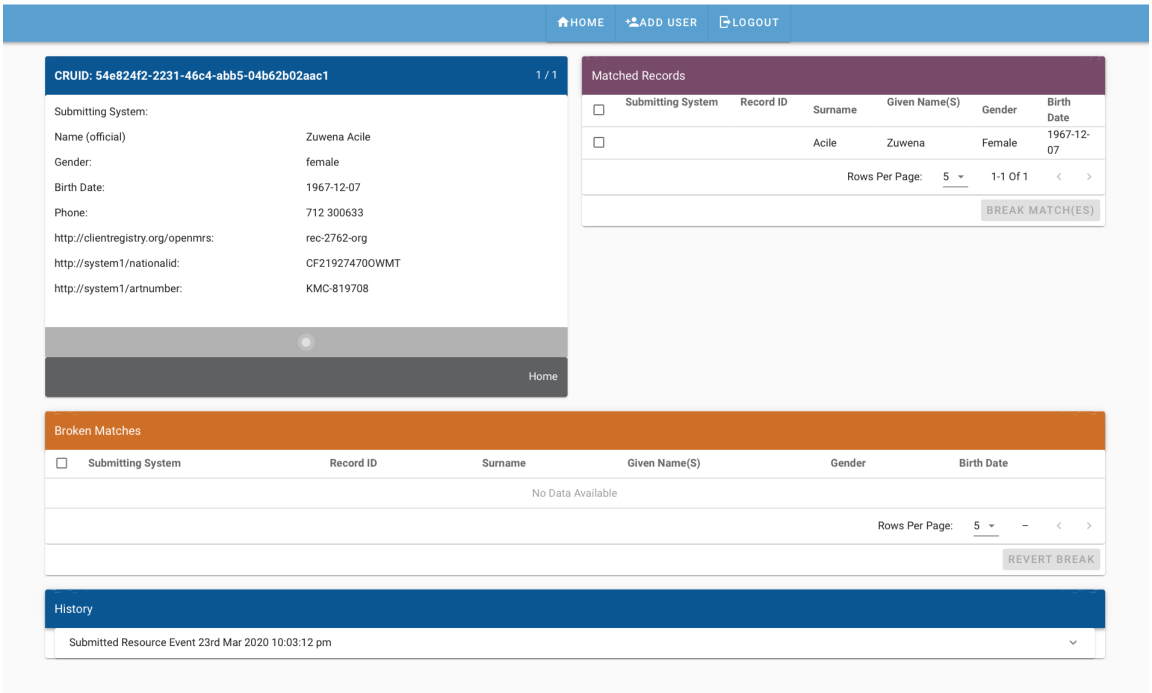
Search Given Name(s): zulfa

Surname ↓	Given Name(s)	NIN	Gender	Birth Date	CRUID
nyanchwo	zulfas		male		bd82da08-63d0-49b1-9be2-dd45d1640a2d
pauson	zulfa		male	1906-12-14	23100fd8-d73b-4d2a-b200-a19923c310f3
pa son	zulfa		male	1906-12-14	23100fd8-d73b-4d2a-b200-a19923c310f3

Record

The record page has a great deal of information, including:

- On the top right: All of the fields stored in the system
- On the top left: Matched records to the current one being viewed.
- Middle of the page: Broken matches, if they exist for the record.
- Bottom of the page: A history of all events affected the record, including creation, modification, and the decision rules used to make the matches.



CRUID: 54e824f2-2231-46c4-abb5-04b62b02aac1 1 / 1

Submitting System:	
Name (official)	Zuwena Acile
Gender:	female
Birth Date:	1967-12-07
Phone:	712 300633
http://clientregistry.org/openmrs:	rec-2762-org
http://system1/nationalid:	CF219274700WMT
http://system1/artnumber:	KMC-819708

Matched Records

□	Submitting System	Record ID	Surname	Given Name(S)	Gender	Birth Date
□			Acile	Zuwena	Female	1967-12-07

Rows Per Page: 5 1-1 Of 1

Broken Matches

□	Submitting System	Record ID	Surname	Given Name(S)	Gender	Birth Date

No Data Available

Rows Per Page: 5 –

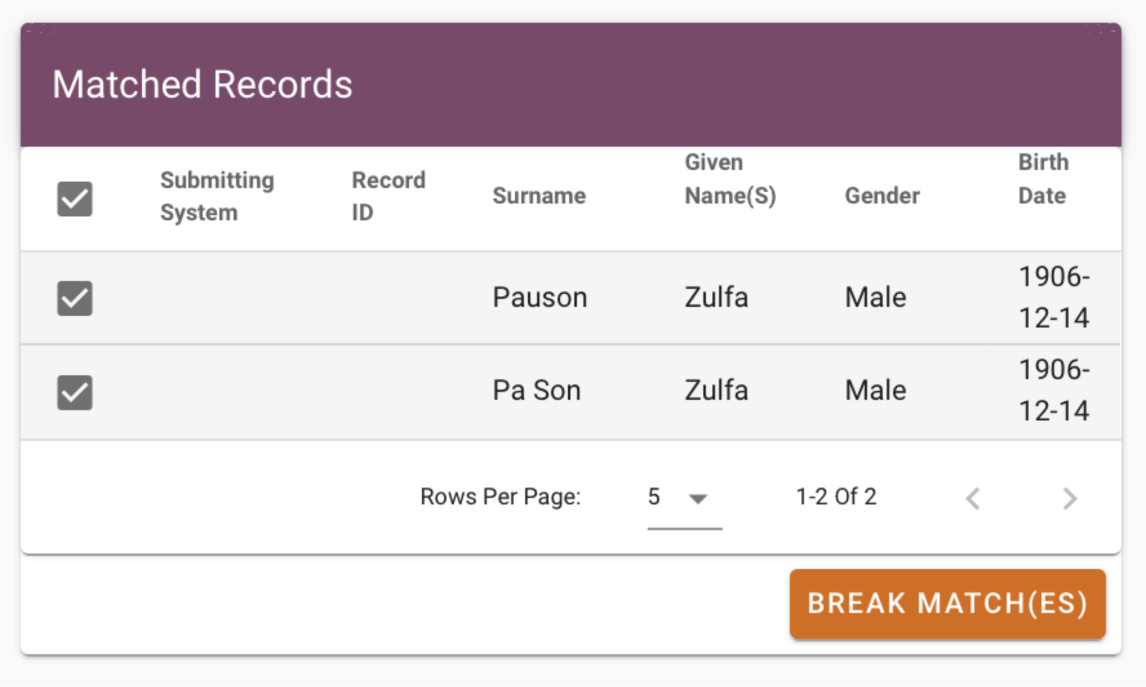
History

Submitted Resource Event 23rd Mar 2020 10:03:12 pm

Matched Records and Break Match

Matched records are listed in a compact table with links to other record.

There is also an option to break one or all matches.



Matched Records

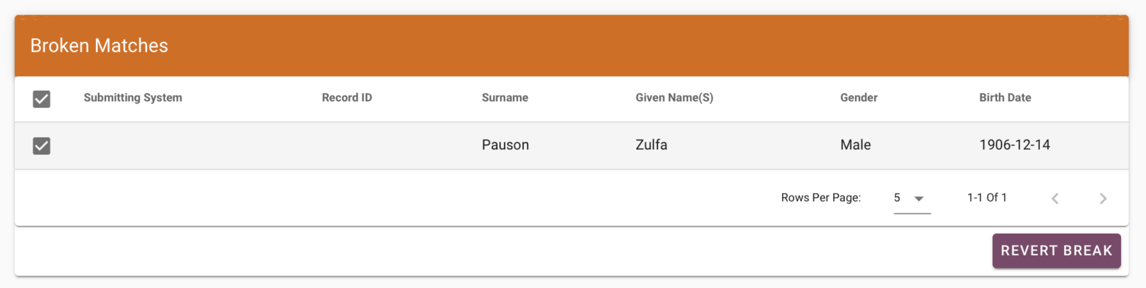
☑	Submitting System	Record ID	Surname	Given Name(S)	Gender	Birth Date
☑			Pauson	Zulfa	Male	1906-12-14
☑			Pa Son	Zulfa	Male	1906-12-14

Rows Per Page: 5 1-2 Of 2

Break and Revert Matches

A match can be broken. When a match is broken, the patient record is no longer linked to it, therefore its CRUID changes.

Once a match is broken, it may be reverted, meaning that the match can be reinstated.

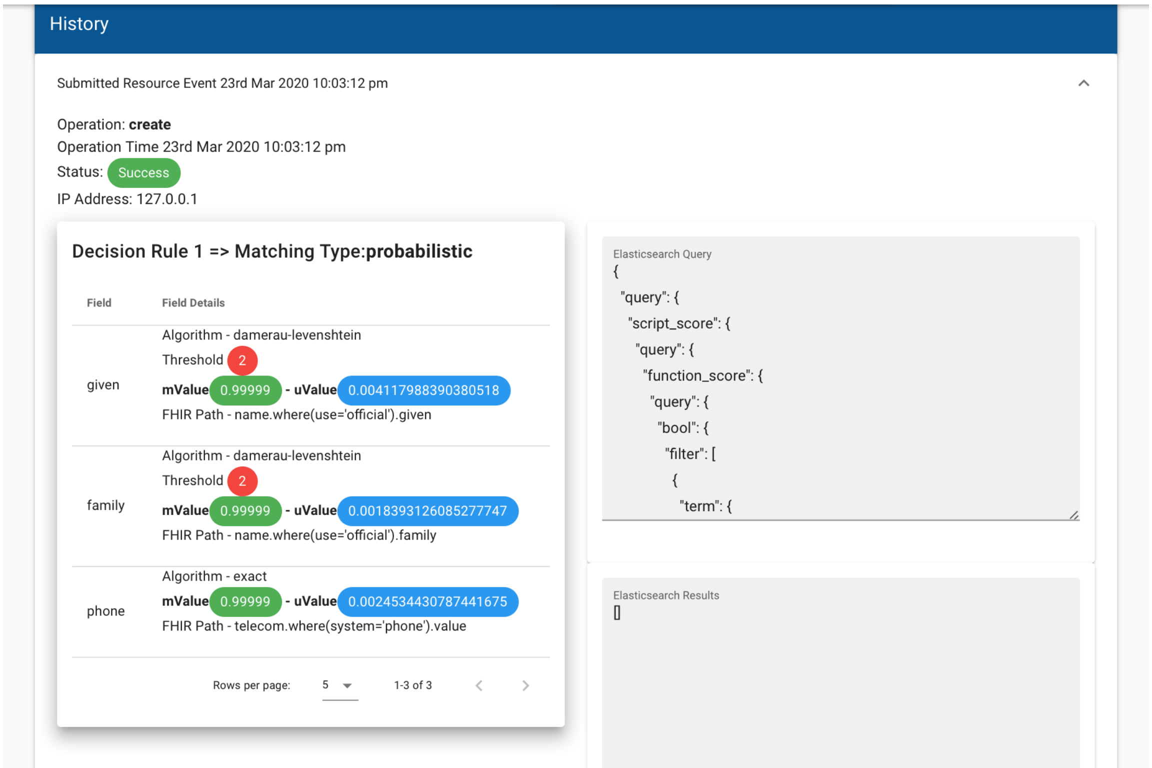


Broken Matches

☑	Submitting System	Record ID	Surname	Given Name(S)	Gender	Birth Date
☑			Pauson	Zulfa	Male	1906-12-14

History

The history card shows the set of decision rules and overall submission information about each history event. All events include any decision rules that were used to make those matches and the specific ElasticSearch query.



History

Submitted Resource Event 23rd Mar 2020 10:03:12 pm

Operation: **create**
Operation Time 23rd Mar 2020 10:03:12 pm
Status: Success
IP Address: 127.0.0.1

Decision Rule 1 => Matching Type:probabilistic

Field	Field Details
given	Algorithm - damerau-levenshtein Threshold 2 **mValue** 0.99999 - **uValue** 0.004117988390380518 FHIR Path - name.where(use='official').given
family	Algorithm - damerau-levenshtein Threshold 2 **mValue** 0.99999 - **uValue** 0.0018393126085277747 FHIR Path - name.where(use='official').family
phone	Algorithm - exact **mValue** 0.99999 - **uValue** 0.0024534430787441675 FHIR Path - telecom.where(system='phone').value

Elasticsearch Query

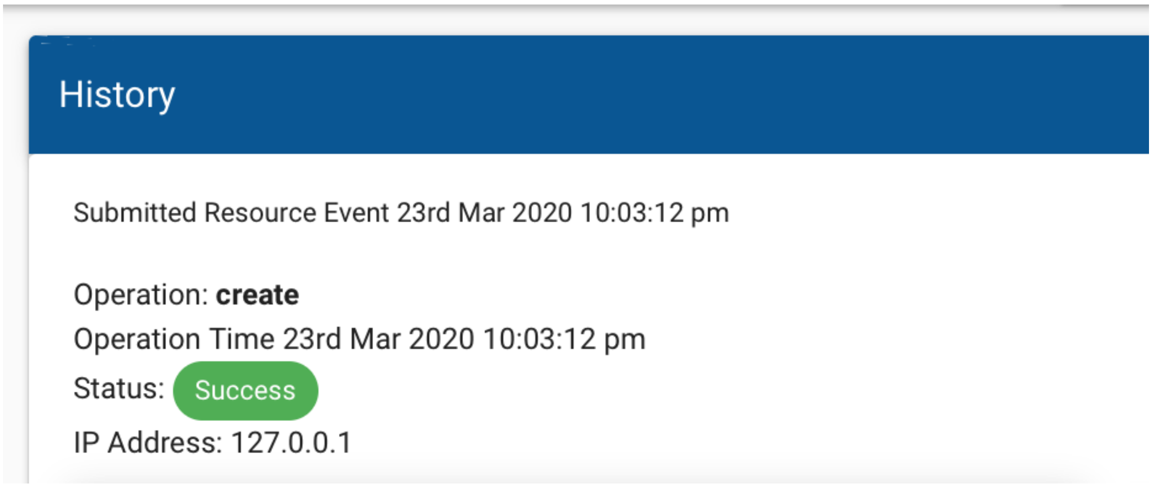
```
{
 "query": {
  "script_score": {
   "query": {
    "function_score": {
     "query": {
      "bool": {
       "filter": [
        {
         "term": {
```

Elasticsearch Results

```
[]
```

Submission

The submission information includes when the event occurred, the status, and the IP address of the submitting system. (The address 127.0.0.1 in the example means within the same computer, not from the network, and is for example purposes only.)



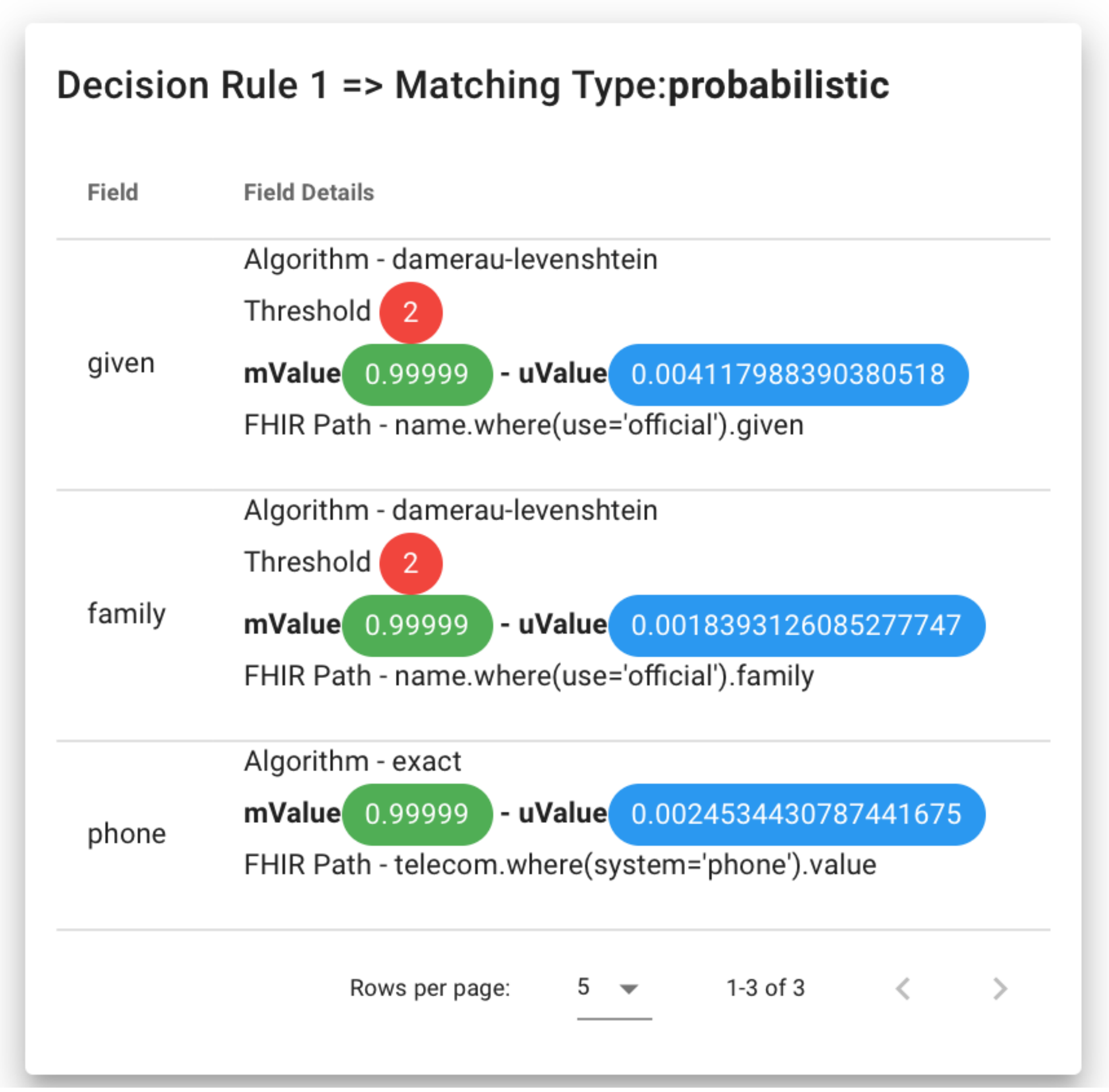
History

Submitted Resource Event 23rd Mar 2020 10:03:12 pm

Operation: **create**
Operation Time 23rd Mar 2020 10:03:12 pm
Status: Success
IP Address: 127.0.0.1

Decision Rules

Decision rules include the overall rule to evaluate the chain of decision rules, which is either probabilistic or deterministic. Then the card shows each decision rule and its configuration. This card helps understand how a decision was made and is critical for evaluation purposes.



Decision Rule 1 => Matching Type:probabilistic

Field	Field Details
given	Algorithm - damerau-levenshtein Threshold 2 **mValue** 0.99999 **- uValue** 0.004117988390380518 FHIR Path - name.where(use='official').given
family	Algorithm - damerau-levenshtein Threshold 2 **mValue** 0.99999 **- uValue** 0.0018393126085277747 FHIR Path - name.where(use='official').family
phone	Algorithm - exact **mValue** 0.99999 **- uValue** 0.0024534430787441675 FHIR Path - telecom.where(system='phone').value

Last update: April 13, 2020

OpenHIE Implementation Guide

Implementation processes are complex. The OpenHIE Client Registry Subcommunity is a valuable resource for understanding the policies and practices of client registries, particularly in low resource settings. Visit their site.

The community provides a comprehensive resource, The Client Registry Implementation Guide, may be consulted for the overall processes required to implement, including how to:

- Analyze the current environment
- Establish leadership and governance
- Document specifications and requirements
- Implement specifications
- Create a support plan
- Conduct a post-production evaluation

Last update: April 13, 2020

Roles and Responsibilities

Responsibilities

There are a great deal of responsibilities that must be addresses for a successful implementation beyond the OpenHIE Implementation Guide.

- **Which systems will connect to the CR and how will support for querying the CR be implemented on the POS side?** There is emerging FHIR support for POS systems but features will need to be added to submit and process the queries. The Developer Guide includes a link to a reference implementation for OpenMRS MPI Client.
- **Which form fields of demographic data will be submitted?** Every use case and every form is different. There may be many different sets of demographic data that are stored, and this affects how matching is done.
- **Which algorithms and decision rules make sense for the use case?** There is a scientific literature on which algorithms perform efficiently for matching. There is a need to test algorithms -- which can be done outside of the CR such as in R and Python – and the need to evaluate against what the CR implementation is doing to ensure consistency.
- **How will the matching algorithms be implemented, deployed, and backed-out for incorrect matches?** The CR includes an Admin UI for matching. The UI is meant to be highly restricted; it makes available demographic records.
- **Who is responsible for providing preprocessed data to the CR?** The CR accepts formatted FHIR messages. It does not impose its own algorithms for cleaning. Connecting POS systems must provide data in the correct format and preprocess the data before sending. Updates can be made to incorrect demographic data later and those will be added to existing records.
- **How will systems launch and scaling up be managed?** What network and compute resources are available for deployment. Advanced Linux systems administration skills are required to launch and maintain the CR in production.

Roles

In actual practice, there are specific roles.

- **Point-of-service systems users**: POS systems use the Client Registry to obtain a CRUID. This process is mostly invisible. Users of EMRs and other systems submitting queries may only see that there is a CRUID for a patient. The Client Registry is invisible to them.
- **POS developers**: Client Registry integration must be added into POS systems for them to be able to query for a CRUID. Software developers of POS systems should review the Developer Manual and understand how to implement the proper FHIR query for obtaining a CRUID.
- **Matching administrators** There may be situations in which the Client Registry implementation uses the UI for viewing and breaking matches. This is a privileged role that should be restricted to few individuals.
- **Client Registry systems administrator**: People managing the network, servers, backups and other aspects of the Client Registry. They should be very familiar with the Developer Guide, particularly security of the system, how to perform upgrades, and recovery procedures
- **Management team**: Governance of the system should be handled by a management team familiar with the implications of decisions, strategy, roll-out, and other aspects.

Last update: April 13, 2020

Additional Resources

Last update: April 13, 2020

Proficiencies

Linux Systems Administration

OpenCR is not one application, it's several and is expected to be run on Linux. Persons installing and managing OpenCR require advanced expertise with Linux. Here are the major topics where knowledge is required:

- **Linux users and groups**: This includes understanding and restricting `sudo` access.
- **Networking**: Limiting the public and LAN exposure of services. For instance, HAPI FHIR Server and ElasticSearch should only be exposed to localhost while the UI may be exposed to a LAN subnet, if at all.
- **Databases**: HAPI FHIR Server requires a database backend. For demos, it can be used with an existing temporary datastore, Derby, but this is not appropriate for maintaining data in production. In production, databases should be backed up and those backups tested as suitable artifacts for recovery.
- **Process management**: systemd and the systemctl series of commands are recommended for managing the process lifecycle, including restarting services and logging their status.
- **Logging**: Suitable logging practice requires safely logging the minimum data required to understand performance and uptime.
- **Auditing**: The OpenCR software stack should be regularly audited. The security section discusses the range of issues to address.

Software

For installing and managing an existing OpenCR installation, there are a handful of commands that can be learned.

- **Java**: HAPI FHIR Server and ElasticSearch are written in Java. Java applications are generally built with frameworks for common design patterns and often built using Gradle or Maven.
- **JavaScript and Node**: The OpenCR Service is written in Node, a popular JavaScript framework for building RESTful applications. Node applications are packaged around the Node Package Manager.
- **Postgres/MySQL**: Either Postgres or MySQL are recommended to be used with HAPI FHIR Server. Administrators should become familiar with backup and recovery procedures.

Last update: April 13, 2020

Internals

ElasticSearch

ES is a web service around the Apache Software Foundation-supported Lucene search engine. ES provides a JSON-based REST API, cluster management, and other value-add on top of Lucene. Many of the features discussed below are actually in Lucene, and are not specific to ES, although these docs refer to ES as including Lucene features. This can be confusing. ES means ES which includes Lucene.

Mapping

Data fields to be indexed in ES require that they first be mapped. The CR mediator takes a mapping config file and generates the mapping based on it. Then, records are submitted to and indexed by ES. Indexes are created separated into segments. Segments can be on one system or across server nodes in cluster. When searches happen, the segments are searched in parallel, and then the results merged.

For more information, see: Elasticsearch from the bottom up (EuroPython 2014 - Start at 18:28 unless you want to get deep into Lucene.): https://www.youtube.com/watch?v=PpX7J-G2PEo

ES Filters versus Queries

An important distinction is between filters and queries in ES. For CR purposes, filters can be used for blocking. Queries result in a score that is assigned to the result based on how well it matches the query. For more, eee: https://logz.io/blog/elasticsearch-queries/ Note that filters are cached, and thus faster. Also, query clauses can be used as either a filter or a query. Since filters are boolean (true/false), they are not scored.

There are queries of diverse types, including Boolean, compound (chaining queries together), fuzzy matching, and other types. Queries result in a score that is assigned to the result based on how well it matches the query. With regard to risk for the Uganda use case, we have included blocking (filters) to meet the use requirements and the required query types.

For more, see ElasticSearch Queries And the ES Query Domain Specific Language (DSL)

Last update: April 13, 2020

Local Installation using Docker

Time to complete

10 Minutes

Warning

This guide is for demonstrations or tests only, not for servers or production environments.

The easiest way to get started with OpenCR is to use Docker to launch ElasticSearch and HAPI FHIR Server and run the OpenCR Service directly. By running the OpenCR Service directly, it is easy to revise and reload decision rules.

These instructions have been tested on Linux and macOS.

Note

This installation method requires some familiarity with the command line.

Prerequisites

- Any modern PC capable of running Docker for Desktop.
  - macOS: 2010 and newer Macs. macOS 10.13 or later (Sierra, Mojava, Catalina).
  - Windows 10 64-bit (Education, Pro, or Enterprise). Note that you must have Hyper-V and Containers Windows enabled and these require administrator privileges.
- 8GB RAM on the computer is recommended. ElasticSearch and HAPI FHIR Server will use up to 1GB of RAM. OpenCR Service will use less than 200MB RAM.

- Docker for Desktop
- Node 10 which includes npm.
- git

Instructions

- Clone the repository and change directory into the root folder.

```
https://github.com/intrahealth/client-registry.git
cd client-registry
```

- Ensure that Docker is installed and running.

```
docker --version
```

- Start ElasticSearch and HAPI FHIR Server using Docker.

Warning

You cannot use the existing hosted ElasticSearch image because OpenCR requires a plugin to be installed. The docker-compose file provided uses the Dockerfile-es which builds an ES image with the plugin.

```
docker-compose up fhir es
```

- Switch to a new terminal window. Install the requirements for the OpenCR Service.

```
cd server
npm install
```

- Copy a configuration for Docker for the OpenCR Service to use.

```
cp config/config_docker_template.json config/config_docker.json
```

- Run the server using the docker config for NODE_ENV.

```
# from client-registry/server
sudo NODE_ENV=docker node lib/app.js
```

- Visit the UI at: https://localhost:3000/crux
    - **Default username**: root@intrahealth.org
    - **Default password**: intrahealth

OpenCR may require access to /var/log for logging. This requirement may be changed in the future.

Last update: April 13, 2020

Example API Query (cURL)

Certificates (Mandatory)

A system querying the Client Registry needs a server-issued certificate or it will not be authorized to use the service.

The way that this works is that a server creates a certificate for a client. The certificate is signed by the server issuing it. The querying system then uses that certificate that has been issued to them in their requests. The server's public key is used by the querying system to verify that the certificate being sent is how the server verifies that the certificate was created by them.

There is a set of generated certificates for testing and demonstrations. They are not appropriate for production.

A Simple CLI Query

Inside /client-registry/server directory, a cURL query using the provided example JSON file would be:

```
curl --cert sampleclientcertificates/openmrs.p12 --cert-type p12 --
cacert certificates/server_cert.pem -d @/Users/richard/src/
github.com/intrahealth/client-registry/DemoData/
patient1_openmrs.json -H "Content-Type: application/json" -XPOST
https://localhost:3000/Patient
```

Should result in a successful result in stdout:

```
info: Received a request to add new patient
{"timestamp":"2020-01-28 14:29:20"}
info: Searching to check if the patient exists
{"timestamp":"2020-01-28 14:29:20"}
info: Getting http://localhost:8080/baseR4/Patient?
identifier=431287 from server {"timestamp":"2020-01-28 14:29:20"}
info: Patient [{"system":"http://clientregistry.org/
openmrs","value":"431287"},{"system":"http://
system1.org","value":"12349","period":
{"start":"2001-05-06"},"assigner":{"display":"test Org"}}] doesnt
exist, adding to the database {"timestamp":"2020-01-28 14:29:20"}
```

Last update: April 21, 2020

Basic query in python

Duplicate a simple cURL query in Python.

This example is available in the Jupyter notebook at: github.com/intrahealth/client-registry-docs/notebooks/simple_query_in_python.ipynb

```
curl --cert sampleclientcertificates/openmrs.p12 --cert-type p12 --cacert certificates/server_cert.pem -d @/Users/richard/src/github.com/openhie/client-registry/DemoData/patient1_openmrs.json -H "Content-Type: application/json" -XPOST https://localhost:3000/Patient
```

```python
#!/usr/bin/env python3
from pathlib import Path
# import requests
from requests_pkcs12 import get, post

# path to your git clone of github.com/intrahealth/client-registry
crhome = Path.home() / 'src' / 'github.com' / 'intrahealth' / 'client-registry'
clientcert = crhome / 'server' / 'sampleclientcertificates' / 'openmrs.p12'
servercert = crhome / 'server' / 'certificates' / 'server_cert.pem'
csv_file = crhome / 'tests' / 'uganda_data_v21_20201501.csv'
payload_bytes = crhome / 'DemoData' / 'patient1_openmrs.json'
payload = open(payload_bytes)

server = 'https://localhost:3000/Patient'
headers = {'Content-Type': 'application/json'}
response = post(server, headers=headers, data=payload,
                pkcs12_filename=clientcert,
                pkcs12_password='',
                verify=servercert)

print(response.headers['location'])
```

```
Patient/07d346c7-b787-4a57-a280-7ad4d0cc2086
```

Last update: April 21, 2020

Decision Rules

Overview

Demographic data from submitting systems is stored in HAPI FHIR. It is also recommended that the demographic data that is primarily stored in HAPI FHIR be indexed into Elasticsearch.

For match processing, there are two options. One is run in mediator-only mode, which is highly flexible and supports a handful of algorithms that can be chained together. Additional algorithms can be added as needed.

The second is to use ES. ES is very fast and supports compound queries but currently only supports Levenshtein distance. When using ES, every request to the FHIR Server is cached in ES.

(One additional caveat for Levenshtein distance is that the mediator-only matching can support edit distances exceeding two, while ES edit distance cannot exceed two.)

Every client wishing to use the Client Registry must be authenticated and authorized. See the configuration page for more information.

How to Set Decision Rules

Decision rules determine how matches are made among records, for example, by using a certain algorithm on one field and a different algorithm on another.

Let's use the below example:

`rules.givenName` is used as one rule on the field givenName.

`rules.givenName.algorithm` defines an algorithm, in this instance Jaro-Winkler, and an threshold for that algorithm unique to it.

`rules.givenName.path` is a required FHIRpath for the fields, a standard way to define how to traverse a FHIR resource. In future, a GUI may be used for defining the FHIRpath.

By default, all of the rules are chained together in a logical AND statement. In ES the search queries are assembled into compound queries.

Link to file

Contents of `server/config/decision_rules.json`

```
{
  "__comments": {
    "path": "Its a fhir path, for syntax refer to https://
www.hl7.org/fhir/fhirpath.html",
    "type": "String, Date, Number or Boolean",
    "threshold": {
      "levenshtein": "Lower the number, the closer the match, 0
being exact match",
      "jaro-winkler": "number between 0 and 1, where 0 for no
match and 1 for exact match"
    }
  },
  "rules": {
    "givenName": {
      "algorithm": "jaro-winkler",
      "threshold": 0.89,
      "path": "name.where(use='official').last().given",
      "type": "string",
      "systems": ["system1", "system2", "system3"]
    },
    "familyName": {
      "algorithm": "damerau-levenshtein",
      "threshold": 3,
      "path": "name.where(use='official').last().family",
      "type": "String"
    },
    "gender": {
      "algorithm": "exact",
      "path": "gender",
      "type": "String"
    }
  }
}
```

Last update: April 13, 2020

System Requirements

IT Resource Planning

Benchmarking will be completed in future phases to make recommendations for medium to heavy workloads. The below resource suggestions should be revised based on benchmarking for the particular context into which OpenCR is being deployed.

Servers

For an MVP in a production environment where potential data loss is acceptable, a single large server can be used. ES has high memory requirements.

CPU

- 2-8 cores available for the OpenCR platform apps.

Memory

Memory usage depends on the number of records and the performance required. At minimum: 32GB with 24GB free for OpenCR is recommended for light loads if using one VM.

- 16GB minimum for ElasticSearch with 32GB preferred or 64GB for high volume: Follow the guidelines provided by the maintainers here.
- 8GB for OpenHIM, mediator, Postgres, and HAPI FHIR Server.

Disk Space

This depends heavily on the workload. Expect 200GB at minimum per node.

Last update: April 13, 2020

Choosing a Method

Select an installation method based on your requirements.

> **Hint**
>
> If you've used Docker before, it's the fastest way to evaluate OpenCR.

Local installation (Docker):

- *Skills required*: git, Docker, some command line familiarity
- *OS*: macOS, Windows, Linux.
- *Usage*: Demos, training, evaluation, research.

Local installation (Direct):

- *Skills required*: git, Docker, some command line familiarity
- *OS*: macOS, Windows, Linux.
- *Usage*: Software development, demos, training, evaluation, research.

Server installation:

- *Skills required*: Command line and Linux expertise.
- *OS*: Linux.
- *Usage*: Production, research.

Last update: April 13, 2020

Configuration

Often there are many records of the same person but in many people in different systems. The purpose of the Client Registry is to link patients in different systems, but not to transfer any data, neither clinical records nor demographic data.

Caution

The Client Registry does not store clinical information. Having the Client Registry enables the ability to create a Shared Health Record in the future.

The Client Registry stores the patient demographic data submitted to it in queries. The Client Registry stores demographic data at least in the HAPI FHIR Server, which can have any database backend an implementer chooses to use.

ElasticSearch (ES) is an optional search engine, and requires configuration. ES can also store patient data fields selectably.

JSON files are used to configure the system. Later iterations will support environment variables and a graphical interface. See https://github.com/openhie/client-registry/tree/master/server/config for example configuration files discussed here.

Deciding Between a Standalone or Mediator Configuration

A central application is the Client Registry Service, as distinct from the larger Client Registry platform. There are two options for running the application, as an OpenHIM mediator or as standalone application.

Choose running the app standalone when:

- For testing, demonstration, or development environments.

- There are few clients that will connect to managing client authentication and roles will not be a burden.
- There is no need for an additional layer of auditing.

Choose running the app as a mediator when:

- For production. The central application should be run as a mediator registered in OpenHIM.
- There are many clients that will need to connect.
- There is a need to audit transactions.
- There is an existing health information exchange layer or OpenHIM.
- One advantage of using the OpenHIM interface is the ability to change settings like the FHIR server.

Security and Privacy

Many configuration options relate to privacy and security. These steps are critical to address. See the security page

Whether in standalone or as a mediator, the Client Registry must interact only with known, trusted clients with TLS certificates. Clients must be registered and certificates assigned to them.

In standalone mode, the server runs TLS by default, and requires signed certificates. Client certificate needs can be turned off in OpenHIM when running as a mediator and this feature must be regularly audited to ensure security.

Connecting Services

The default ports are as follows:

- **3000**: Client Registry Service
- **9200**: ElasticSearch (closed to external)
- **8080**: HAPI FHIR Server (closed to external)

In `server/config/config_development_template.json` there is a template for configuration.

Link to file

Contents of `server/config/config_development_template.json`

```
{
  "app": {
    "port": 3000,
    "installed": false
  },
  "mediator": {
    "api": {
      "username": "root@openhim.org",
      "password": "openhim-password",
      "apiURL": "https://localhost:8080",
      "trustSelfSigned": true,
      "urn": ""
    },
    "register": false
  },
  "fhirServer": {
    "baseURL": "http://localhost:8080/clientregistry/fhir",
    "username": "hapi",
    "password": "hapi"
  },
  "elastic": {
    "server": "http://localhost:9200",
    "username": "",
    "password": "",
    "max_compilations_rate": "10000/1m",
    "index": "patients"
  },
  "structureDefinition": {
    "reportRelationship": "patientreport"
  },
  "matching": {
    "tool": "mediator"
  },
  "systems": {
    "openmrs": {
      "uri": "http://clientregistry.org/openmrs"
    },
    "dhis2": {
      "uri": "http://clientregistry.org/dhis2"
    },
    "lims": {
      "uri": "http://clientregistry.org/lims"
    },
    "brokenMatch": {
      "uri": "http://ihris.org/CR/brokenMatch"
    }
  },
  "sync": {
    "lastFHIR2ESSync": "1970-01-01T00:00:06"
  },
  "__comments": {
    "matching.tool": "this tells if the app should use mediator
algorithms or elasticsearch algorithms for matching, two options
mediator and elasticsearch"
  }
}
```

General App Configuration

`app.port` is the port the application will run on.

`app.installed` can be left to True. This tells the Client Registry Service to load structure definitions into HAPI FHIR Server, otherwise it will not.

Mediator App Configuration

`mediator.register` to true if the application will run as a mediator. Or, to false if the app will run as standalone.

`mediator.api.xx` settings are only if running as a mediator.

`mediator.api.username | password` must be different. The existing settings are defaults and must be changed when configuring the OpenHIM.

`mediator.api.trustSelfSigned` should be set to false in production or any sensitive environment. True is only for demonstrations.

FHIR Server

The currently supported FHIR version is R4.

`fhirServer.baseURL` is the default. Note that it may change depending on the way HAPI is installed. It may, for example, default to a baseURL of http://localhost:8080/baseR4/.

`fhirServer.username | password` must be changed from defaults in HAPI.

ElasticSearch Configuration

For ES, the relationship between patient resources in FHIR and what fields are synchronized in ES must be explicitly defined. This is termed the Report Relationship mapping. One must define what resource to be used (patient) and what fields need to be available in ES. After this, the Client Registry reads these fields, and populates ES with the information.

In `resources/Relationships/PatientRelationship.json` there is a template for configuration.

Link to file

Contents of `resources/Relationships/PatientRelationship.json`

```
{
  "resourceType": "Basic",
  "id": "patientreport",
  "meta": {
    "versionId": "1",
    "lastUpdated": "2019-07-30T07:34:24.098+00:00",
    "profile": [
      "http://ihris.org/fhir/StructureDefinition/iHRISRelationship"
    ]
  },
  "extension": [{
    "url": "http://ihris.org/fhir/StructureDefinition/
iHRISReportDetails",
    "extension": [{
      "url": "label",
      "valueString": "Patient Report"
    }, {
      "url": "name",
      "valueString": "patients"
    }, {
      "url": "http://ihris.org/fhir/StructureDefinition/
iHRISReportElement",
      "extension": [{
        "url": "label",
        "valueString": "gender"
      }, {
        "url": "name",
        "valueString": "gender"
      }]
    }, {
      "url": "http://ihris.org/fhir/StructureDefinition/
iHRISReportElement",
      "extension": [{
        "url": "label",
        "valueString": "birthDate"
      }, {
        "url": "name",
        "valueString": "birthDate"
      }]
    }, {
      "url": "http://ihris.org/fhir/StructureDefinition/
iHRISReportElement",
      "extension": [{
        "url": "label",
        "valueString": "given"
      }, {
        "url": "name",
        "valueString": "name.where(use='official').last().given"
      }]
    }, {
      "url": "http://ihris.org/fhir/StructureDefinition/
iHRISReportElement",
      "extension": [{
        "url": "label",
        "valueString": "family"
      }, {
        "url": "name",
        "valueString": "name.where(use='official').last().family"
      }]
    }, {
      "url": "http://ihris.org/fhir/StructureDefinition/
iHRISReportElement",
      "extension": [{
        "url": "label",
        "valueString": "fullname"
      }, {
        "url": "name",
        "valueString": "name.where(use='official').last().text"
      }]
    }, {
      "url": "http://ihris.org/fhir/StructureDefinition/
```

```
iHRISReportElement",
      "extension": [{
        "url": "label",
        "valueString": "phone"
      }, {
        "url": "name",
        "valueString": "telecom.where(system='phone').value"
      }]
    }]
  }],
  "code": {
    "coding": [{
      "system": "http://ihris.org/fhir/ValueSet/ihris-resource",
      "code": "iHRISRelationship"
    }],
    "text": "iHRISRelationship"
  },
  "subject": {
    "reference": "StructureDefinition/Patient"
  }
}
```

OpenHIM Mediator JSON Configuration

If using OpenHIM, it must be configured for proper clients and roles to accept and forward requests from the Client Registry. An example export of a working JSON configuration that can be imported for development purposes is available.

Link to file

Contents of `server/config/mediator.json`

```
    {
      "urn": "urn:uuid:4bc42b2f-b5a8-473d-8207-5dd5c61f0c4a",
      "version": "0.0.1",
      "name": "Client Registry",
      "description": "Uganda Client Registry",
      "config": {
        "fhirServer": {
          "username": "hapi",
          "password": "hapi",
          "baseURL": "http://localhost:8080/hapi/fhir"
        },
        "elastic": {
          "server": "http://localhost:9200",
          "username": "",
          "password": "",
          "max_compilations_rate": "10000/1m",
          "index": "patients"
        },
        "matching": {
          "tool": "elasticsearch"
        }
      },
      "configDefs": [{
        "param": "fhirServer",
        "displayName": "FHIR Server",
        "description": "FHIR Server Configuration Details",
        "type": "struct",
        "template": [{
            "type": "string",
            "description": "The base URL (e.g. http://localhost:
8080/hapi/fhir)",
            "displayName": "Base URL",
            "param": "baseURL"
          },
          {
            "type": "string",
            "description": "Username required to access FHIR
server",
            "displayName": "Username",
            "param": "username"
          },
          {
            "type": "password",
            "description": "Password required to access FHIR
server",
            "displayName": "Password",
            "param": "password"
          }
        ],
        "values": []
      }, {
        "param": "elastic",
        "displayName": "Elasticsearch Server",
        "description": "Elasticsearch Server Configuration
Details",
        "type": "struct",
        "template": [{
            "type": "string",
            "description": "The base URL (e.g. http://localhost:
9200)",
            "displayName": "Base URL",
            "param": "server"
          },
          {
            "type": "string",
            "description": "Username required to access
elasticsearch server",
            "displayName": "Username",
            "param": "username"
          },
          {
            "type": "password",
            "description": "Password required to access
elasticsearch server",
            "displayName": "Password",
            "param": "password"
          }, {
            "type": "string",
            "description": "Number of requests to compile per
minute",
            "displayName": "Maximum Compilations Rate",
            "param": "max_compilations_rate"
          }, {
            "type": "string",
            "description": "index to use for data storage",
            "displayName": "Index Name",
            "param": "index"
          }
        ],
        "values": []
      }, {
        "param": "matching",
        "displayName": "FHIR Server",
        "description": "FHIR Server Configuration Details",
        "type": "struct",
        "template": [{
          "type": "option",
          "values": ["mediator", "elasticsearch"],
          "description": "Tool to Use for Matching",
          "displayName": "Tool to Use for Matching",
          "param": "tool"
        }],
        "values": []
      }],
      "defaultChannelConfig": [{
        "requestBody": true,
        "responseBody": true,
        "name": "Add Patients",
        "description": "Post a new patient into the client
registry",
        "urlPattern": "/addPatient",
      "matchContentRegex": null,
      "matchContentXpath": null,
      "matchContentValue": null,
      "matchContentJson": null,
      "pollingSchedule": null,
      "tcpHost": null,
      "tcpPort": null,
      "autoRetryPeriodMinutes": 60,
      "autoRetryEnabled": false,
      "rewriteUrlsConfig": [],
      "addAutoRewriteRules": true,
      "rewriteUrls": false,
      "status": "enabled",
      "alerts": [],
      "txRerunAcl": [],
      "txViewFullAcl": [],
      "txViewAcl": [],
      "properties": [],
      "matchContentTypes": [],
      "routes": [{
        "name": "Add Patient",
        "secured": false,
        "host": "localhost",
        "port": 3000,
        "path": "/addPatient",
        "pathTransform": "",
        "primary": true,
        "username": "",
        "password": "",
        "forwardAuthHeader": false,
        "status": "enabled",
        "type": "http"
      }],
      "authType": "public",
      "whitelist": [],
      "allow": [],
      "type": "http",
      "methods": [
        "POST"
      ]
    }],
    "endpoints": [{
      "name": "Activate Client Registry",
      "host": "localhost",
      "path": "/addPatient",
      "port": 3000,
      "primary": true,
      "forwardAuthHeader": false,
      "status": "enabled",
      "type": "http"
    }],
    "_uptime": 2201.945,
    "_lastHeartbeat": "2017-12-15T03:47:03.365Z",
    "_configModifiedTS": "2017-12-15T02:52:49.054Z"
  }
```

Last update: April 13, 2020

Local Installation

🟦 **Time to complete**

60 Minutes

🟧 **Warning**

This guide is for demonstrations or tests only, not for production environments.

🟦 **Note**

This installation method requires familiarity with the command line.

Prerequisites

- CPU/RAM: Modern CPUs with 8GB RAM.
- Java version 8 (1.8). Oracle-licensed Java (requires sign-in) and AdoptOpenJDK (not sign-in required) have been tested.
- Node 10 which includes npm.
- git

HAPI FHIR Server CLI

For non-production environments, the HAPI maintainers provide a simple CLI-based tool to run it.

The only required dependency is Java >= 8 (1.8).

See HAPI FHIR CLI for instructions for the OS of choice.

The Client Registry requires FHIR version R4 and HAPI must be started for this version. To run HAPI:

```
hapi-fhir-cli run-server -v r4
```

The HAPI Web Testing UI is available at http://localhost:8080/ The Web Testing UI should be disabled for production. It allows the viewing of any resource on the server.

The FHIR Base URL is at http://localhost:8080/baseR4/

Visit http://localhost:8080/ to ensure HAPI is up and running or

```
curl -X GET "localhost:8080/baseR4/Patient?"
```

ElasticSearch

Install and start ES for the intended OS. See the ES install instructions

The required version is >=7.6.

The phonetic analysis package must be installed. For example:

```
/usr/share/elasticsearch/bin/elasticsearch-plugin install analysis-phonetic
```

The string similarity plugin must be installed. See: https://github.com/intrahealth/similarity-scoring

Once installed and started, ensure that ES is up and running:

```
curl -X GET "localhost:9200/_cat/health?v&pretty"
```

Status should be yellow for a single-node cluster.

OpenCR Service and UI

Clone the repository into a directory of choice.

```
git clone https://github.com/intrahealth/client-registry.git
```

Enter the server directory, install node packages.

```
cd client-registry/server
npm install
```

Copy and edit the configuration file to your liking.

```
cp config/config_development_template.json config/
config_development.json
# edit the servers...
```

The minimum changes to start a running standalone system are:

- Change `fhirServer.baseURL` to "http://localhost:8080/baseR4/"

Run the server from inside client-registry/server:

```
# from client-registry/server
sudo NODE_ENV=development node lib/app.js
```

- Visit the UI at: https://localhost:3000/crux
    - **Default username**: root@intrahealth.org
    - **Default password**: intrahealth

OpenCR may require access to /var/log for logging. This requirement may be changed in the future.

Congratulations! Now it's time to run a query.

Last update: April 24, 2020

Server Installation

🟧 **Caution**

Installing and maintaining a production installation is not trivial. This installation method requires strong familiarity with the command line and expertise administering Linux environments.

The core production stack consists of four components:

- **OpenCR Service**: This includes primary API for fielding requests, and the record viewing and matching breaking UI.
- **HAPI FHIR Server + Database**: HAPI FHIR Server is the reference implementation of FHIR in Java. It requries a database backend (e.g. Postgres or MySQL).
- **ElasticSearch**: Version >=7.5 supported and the analysis-phonetic plugin is required.

Optional components:

- **OpenHIM core and OpenHIM admin console**. Requires MongoDB. OpenHIM is an authentication, authorization, and auditing layer. While OpenHIM is optional, nodes and users must be managed in some application if not OpenHIM. Nodes must have certificates issued to query OpenCR and they must be rotated out over time. The OpenCR Service can manage simply installations but using an enterprise secrets management tool is recommended.

Prerequisites

Linux is the expected operating system for production.

It is critical that systems administrators note the version compatibilities outlined below. This guide does not cover most aspects of enterprise systems administration, rather it attempts to cover the OpenCR platform. If there are key areas missing, please open an issue on GitHub.

- If entities outside of your LAN are connecting to OpenCR, you will need a public-facing domain name. A domain is necessary for a certificate which is required for any queries.
- See Security

HAPI FHIR Server and Postgres

HAPI FHIR must use a database backend in production. HAPI FHIR stores the patient demographic data from queries. If the data is lost, then OpenCR data is unrecoverable.

- Follow the JPA Server information and instructions for how to customize the hapi.properties file and build the server using maven.
- The ES integration is separate from HAPI FHIR Server, so there is not need to use it as an indexer. ES only works with an old version of ES.
- Install and configure the preferred database. Postgres has been tested by the maintainers but any database should work that HAPI supports. Change default passwords on the database.
- Database replication should be encrypted.
- Confirm that HAPI accepts requests.
- The web interface for HAPI should be disabled for privacy reasons.

> **Caution**
>
> In production, Postgres should run on multiple nodes with replication. This is to ensure high availability and backups of the data.

ElasticSearch

- Follow the instructions for installation

- Systemd is the preferred system and service manager. There are commands to initiate systemd and journalctl.
- The phonetic analysis package must be installed.

```
/usr/share/elasticsearch/bin/elasticsearch-plugin install analysis-
phonetic
```

- The string similarity plugin must be installed. See: https://github.com/intrahealth/similarity-scoring

Caution

ES is not production-ready when run as one single node. It is recommended to run ES on several nodes. Those nodes can also run followers of Postgres.

OpenCR Service and UI

Clone the repository into a directory of choice.

```
git clone https://github.com/intrahealth/client-registry.git
```

Enter the server directory, install node packages.

```
cd client-registry/server
npm install
```

Copy and edit the configuration file to your liking.

```
cp config/config_development_template.json config/
config_development.json
# edit the servers...
```

The minimum changes to start a running standalone system are:

- Change `fhirServer.baseURL` to "http://localhost:8080/baseR4/"

Run the server from inside client-registry/server:

```
node lib/app.js
```

OpenHIM (Optional)

OpenHIM supports the last 2 versions of NodeJS LTS and requires MongoDB.

- Follow the instructions to install OpenHIM core and admin console. The maintainers use the NPM PPA installation method.
- Note the important step to obtain a certificate immediately after installation. The configuration should be that any client must have a certificate and the server has a certificate (mutual TLS).
- Follow the instructions including console configuration.
- Note the important step to change the console password. It is also recommended that the console only be accessible on a local subnet and not to the WAN.
- The config mediator.register must be set to true for the OpenCR Service to use OpenHIM.

Last update: April 24, 2020

Ansible

This documents how to use Ansible playbooks to set up a production-like server installation. It differs from a production installation in that certificates must not be self-signed in a production environment.

> These steps are for installing on a server OS directly and require experience with remote configuration and Linux administration.

Preparation

You must have a local VM or remote server. See `/packaging/vagrant/centos` for a Vagrant VM (CentOS 7) script for working example of creating a local VM.

Clone the main repository. The Ansible playbooks and templates are in that folder.

```
git clone https://github.com/intrahealth/client-registry.git
cd client-registry/packaging/ansible
```

SSH

Create a VM. Make sure to include a public ssh key for the user who will install prerequisites. Your SSH public key should be in `.ssh/authorized_keys` on the remote host, ie:

```
cat ~/.ssh/id_rsa.pub | ssh user@remotehost 'cat >> .ssh/
authorized_keys'
```

Specify hosts

Hosts can be specified in inventory files or on the command line. To use Ansible with an inventory file, you must create a file or edit the one in the repository. There are yaml and ini formats supported.

A `hosts` file that has an entry for one server would be:

```
[servers]
172.16.174.137
```

> Note that `[servers]` is not necessary, it is way to tag groups of servers. The file may simply contain an IP address or domain.

To use the hosts file:

```
ansible-playbook -i hosts someplaybook.yaml
```

Alternately, hosts may be specified on the command line (the comma is necessary even if there is only one host):

```
ansible-playbook -i 172.16.168.158, someplaybook.yaml
```

opencr user (optional)

A example playbook is provided to show how to create a `opencr` user with sudo permissions using Ansible to be used with the host.

Create the `opencr` user and gives it sudo access:

```
ansible-playbook -i hosts user.yaml
```

Installation

```
ansible-playbook -i hosts prep_centos.yaml -e user=opencr
ansible-playbook -i hosts elasticsearch.yaml -e user=opencr
ansible-playbook -i hosts tomcat.yaml -e user=opencr
```

```
ansible-playbook -i hosts postgres.yaml -e user=opencr -e
pgpass=hapi
ansible-playbook -i hosts hapi.yaml -e user=opencr
ansible-playbook -i hosts opencr.yaml -e user=opencr
```

OpenCR is now running. It will only allow requests from localhost (from the same server it is installed on).

Visit: https://ipaddress:3000/crux

HTTPS must be used.

Warning

If not running localhost, follow the next steps to create self-signed server and client certs, and copy them onto the server using an Ansible script below.

Certificates (Required if not using localhost)

Two certificate pairs are required, one pair for the server and one for the client generated from the server's. The existing self-signed server certs use localhost as the CN. This can be seen with the following for any cert:

```
$ openssl x509 -in ../../server/certificates/server_cert.pem -text
...
        Subject: CN = localhost, O = Client Registry
...
```

This means that new server and client certificates need to be generated with the IP address or domain for clients to access the client registry if it is not running on localhost.

Caution

Self-signed certificates must only be created for testing and demonstrations and in non-production settings.

Make a note of your IP or domain for which you need to create a server cert. Run the following to create a new server cert/key pair. It will ask for a pass phrase (which is required in production) but -nodes option squashes that. This will create two files, server_key.pem and server_cert.pem. We can inspect the certificate to verify it has the IP address in the subject.

```
# confirm ip being used
cat hosts
openssl req -x509 -newkey rsa:4096 -keyout server_key.pem -out
server_cert.pem -days 365 -subj "/CN=172.16.168.172" -nodes
# confirm new CN
openssl x509 -in server_cert.pem -text
```

Now it is necessary to create new a new client cert based on the server cert. A key is first created, then the certificate, and they are packaged together in a p12 file.

```
openssl req -newkey rsa:4096 -keyout ansible_key.pem -out
ansible_csr.pem -nodes -days 365 -subj "/CN=ansible"
openssl x509 -req -in ansible_csr.pem -CA server_cert.pem -CAkey
server_key.pem -out ansible_cert.pem -set_serial 01 -days 36500
# requires specifying an export key
openssl pkcs12 -export -in ansible_cert.pem -inkey ansible_key.pem
-out ansible.p12
```

The client certs can be placed in the existing folder for client certs for convenience.

```
# add client certs
cp ansible_key.pem ../../server/sampleclientcertificates/
cp ansible_csr.pem ../../server/sampleclientcertificates/
cp ansible_cert.pem ../../server/sampleclientcertificates/
cp ansible.p12 ../../server/sampleclientcertificates/
```

> Server certs may also be copied into ../../certificates/ for convenience but this will overwrite the copies and break localhost if the repo is used to upload code to GitHub.

Warning

To complete the process, server certs need to be placed into the server. This will be done using an Ansible script below.

Copy the server certs to the server and restart opencr service to use them.

```
ansible-playbook -i hosts servercerts.yaml -e user=opencr
```

Test (for servers not localhost):

```
# this assumes the server cert and client cert are in this (/packaging/ansible) directory
# replace the path to your copy of the repo
# replace the ip address of the server
curl --cert ansible.p12 --cert-type p12 --cacert server_cert.pem -d @/Users/richard/src/github.com/intrahealth/client-registry/DemoData/patient1_openmrs.json -H "Content-Type: application/json" -XPOST https://172.16.168.172:3000/Patient
```

Add additional user public keys

As necessary, add additional ssh keys to the user `opencr`. (Ensure that the user's public key is available on github, ie. https://github.com/citizenrich.keys):

```
ansible-playbook -i hosts keys.yaml
```

Last update: April 21, 2020

Load Demo Data (JavaScript)

Demonstration data is provided in the /tests directory in the code repository.

Install the Node Packages

```
cd tests
npm install
```

Configure the Script

The /tests/uploadCSV.js script is used to upload the CSV data in the /tests directory.

If using OpenHIM, then change the auth options and the IP address/hostname in /tests/uploadCSV

```
# make a copy to modify
cp uploadCSV.js uploadCSV_mychanges.js
```

The defaults are:

```
const options = {
url: 'http://localhost:5001/Patient',
auth,
json: entry.resource,
};
```

Edit uploadCSV_mychanges.js. If not running OpenHIM then change: * remove `auth` and change it to `agentOptions` * Change the IP address/hostname as required, for example for Docker: 'https://localhost:3000/Patient'.

After the edits, the code block looks like this:

```
const options = {
url: 'https://localhost:3000/Patient',
agentOptions,
json: entry.resource,
};
```

Notice the https as without OpenHIM the OpenCR Service encrypts the connections using TLS instead of OpenHIM doing so.

Running the Script

While in the /tests directory, ensure that OpenCR is running and run the script, with the required argument of the CSV:

```
sudo node uploadCSV_mychanges.js uganda_data_v21_20201501.csv
```

Caution

The script may take several hours to process all of the records.

Last update: April 13, 2020

Load bulk data in Python

This example is available in the Jupyter notebook at: github.com/intrahealth/client-registry-docs/docs/notebooks/load_bulk_data_in_python.ipynb

```
#!/usr/bin/env python3
from pathlib import Path
from requests_pkcs12 import get, post
import pandas as pd
import numpy as np

import recordlinkage

import fhirclient.models.patient as p
import fhirclient.models.humanname as hn
import fhirclient.models.contactpoint as cp
import fhirclient.models.fhirdate as fd
import fhirclient.models.identifier as ident
from fhirclient import client

import json
import time
import itertools

# suppress warning: "Certificate for localhost has no `subjectAltName`, falling back to check for a `commonName` for
import urllib3
urllib3.disable_warnings(urllib3.exceptions.SubjectAltNameWarning)

# versions
print("Pandas version: {0}".format(pd.__version__),'\n')
print("Python Record Linkage version: {0}".format(recordlinkage._version.get_versions()['version']),'\n')
print("Numpy version: {0}".format(np.__version__),'\n')
print("FHIR client version: {0}".format(client.__version__),'\n')
```

```
  Pandas version: 1.0.3

  Python Record Linkage version: 0.14

  Numpy version: 1.18.2

  FHIR client version: 3.2.0
```

```
# path to your git clone of github.com/intrahealth/client-registry
crhome = Path.home() / 'src' / 'github.com' / 'intrahealth' / 'client-registry'
clientcert = crhome / 'server' / 'sampleclientcertificates' / 'openmrs.p12'
servercert = crhome / 'server' / 'certificates' / 'server_cert.pem'
csv_file = crhome / 'tests' / 'uganda_data_v21_20201501.csv'

df_a = pd.read_csv(csv_file)
# df_a = df_a.set_index('rec_id')
print('Number of records :', len(df_a))
print(df_a.head())
```

```
  Number of records : 5000
           rec_id sex date_of_birth given_name       surname phone_number   \
0  rec-2762-org   f      19671207     zuwena         acile   712 300633
1  rec-2009-org   f      19761028     zuwena        lusike   772 614594
2  rec-3269-org   f      19811002     zuwena      mungugeo   772 162632
3  rec-1609-org   f      19270719    zuraika   akantambira   772 837692
4  rec-2802-org   m                   zulfas      nyanchwo   782 855101

       uganda_nin   art_number
0  CF21927470OWMT   KMC-819708
1  CF68167355NUZY   KUB-176148
2  CF50136842UQFQ   MBA-746695
3  CF68008770HZML   KMC-270901
4  CM25736526XWGC   KSG-830566
```

```
# some cleaning
df_a['rec_id'] = df_a['rec_id'].str.strip()
df_a['sex'] = df_a['sex'].str.strip()
df_a['given_name'] = df_a['given_name'].str.strip()
df_a['surname'] = df_a['surname'].str.strip()
df_a['date_of_birth'] = df_a['date_of_birth'].str.strip()
df_a['phone_number'] = df_a['phone_number'].str.strip()
df_a['uganda_nin'] = df_a['uganda_nin'].str.strip()
df_a['art_number'] = df_a['art_number'].str.strip()

df_a['sex']= df_a['sex'].replace('f', 'female')
df_a['sex']= df_a['sex'].replace('m', 'male')
print(df_a['sex'].value_counts())

# fhirclient validates and some birthdate fields are empty/improperly formatted
# remove non-digits
df_a['date_of_birth'] = df_a['date_of_birth'].str.extract('(\d+)', expand=False)
# force into datetime (coerce has benefit that it removes anything outside of 8 digits)
df_a['date_of_birth'] =  pd.to_datetime(df_a['date_of_birth'], errors='coerce')
# now back into str or fhirdate will complain
df_a['date_of_birth'] = df_a['date_of_birth'].apply(lambda x: x.strftime('%Y-%m-%d')if not pd.isnull(x) else '')

print(df_a.head())
```

```
female    3224
           963
male       809
r            1
d            1
k            1
q            1
Name: sex, dtype: int64
         rec_id     sex date_of_birth given_name      surname phone_number  \
0  rec-2762-org  female    1967-12-07     zuwena        acile   712 300633
1  rec-2009-org  female    1976-10-28     zuwena       lusike   772 614594
2  rec-3269-org  female    1981-10-02     zuwena     mungugeo   772 162632
3  rec-1609-org  female    1927-07-19    zuraika  akantambira   772 837692
4  rec-2802-org    male                   zulfas     nyanchwo   782 855101

       uganda_nin  art_number
0  CF21927470OWMT  KMC-819708
1  CF68167355NUZY  KUB-176148
2  CF50136842UQFQ  MBA-746695
3  CF68008770HZML  KMC-270901
4  CM25736526XWGC  KSG-830566
```

```
# default server/path
server = "https://localhost:3000/Patient"
# 3 records, modify if more are required
limit = 3
for index, row in itertools.islice(df_a.iterrows(), limit):
# for index, row in df_a.iterrows():
    patient = p.Patient() # not using rec_id as pandas id, leaving empty
    patient.gender = row['sex']
    name = hn.HumanName()
    name.given = [row['given_name']]
    name.family = row['surname']
    name.use = 'official'
    patient.name = [name]
    phone = cp.ContactPoint()
    phone.system = 'phone'
    phone.value = row['phone_number']
    patient.telecom = [phone]
    patient.birthDate = fd.FHIRDate(row['date_of_birth'])
    emr = ident.Identifier()
    emr.system = 'http://clientregistry.org/openmrs'
    emr.value = row['rec_id']
    art = ident.Identifier()
    art.system = 'http://system1/artnumber'
    art.value = row['art_number']
    nin = ident.Identifier()
    nin.system = 'http://system1/nationalid'
    nin.value = row['uganda_nin']
    patient.identifier = [emr, art, nin]
    # print(json.dumps(patient.as_json()))

    headers = {'Content-Type': 'application/json'}
    start = time.time()
    response = post(server, headers=headers, data=json.dumps(patient.as_json()),
                    pkcs12_filename=clientcert, pkcs12_password='', verify=servercert)
    end = time.time()
    print(index, response.headers['location'], " | ", round((end - start), 1), "ms") # response.headers['Date']
    # print(response.headers)
```

```
0 Patient/53eddeb0-2fb1-4494-8b51-d2e667b962f2  |  1.3 ms
1 Patient/16e4cb00-847f-49af-b624-4b0357d6c897  |  1.5 ms
2 Patient/dd4766a5-cfed-4914-b650-6b49a99ccc06  |  1.1 ms
```

Last update: April 21, 2020

Security

🟧 **Warning**

It is difficult (and irresponsible) to try to explain all of the best practices in computer security. This page focuses on how security is addressed in the Open Client Registry. The information may be incomplete. Where there is more clarity or information needed, please provide feedback in an issue on GitHub so that it can be added.

This page addresses several security areas, including hardening, user authentication, node authentication, auditing, and non-production (demos, tests) configurations.

For links to information on server resource planning see the requirements page.

Hardening

🟧 **Warning**

Server and network hardening and production best practices are out of scope. This document only attempts to capture aspects relevant to the Client Registry.

General Server, Network, and Service Hardening

Hardening and production best practices include:

- Removing unnecessary services, software, network protocols
- Backup and recovery
- Patches
- Vulnerability scanning
- Limiting remote administration
- Managing open internal and external ports

- Auditing, logging software

See, for example, the Guide to General Server Security: Recommendations of the National Institute of Standards and Technology by Karen Scarfone, Wayne Jansen, Miles Tracy, July 2008, (NIST Special Publication 800-123).

OpenCR Platform Hardening

In addition to the above general hardening practices that should be followed, some additional areas are important for the Client Registry.

- The **OpenCR UI** should only be available on a local subnet, and further restricted with user and node authentication so that it is not exposed on the WAN.
- **Close external ports**: Ports for Client Registry services should be locked down to localhost. The only external port required is for TLS with point-of-service systems that make queries to the Client Registry Service. This includes ES and HAPI FHIR. Those services can serve localhost only and effectively.
- **TLS for cluster (interservice) communication**: Where Postgres and ES must communicate with other nodes TLS can be configured for ES for internode communication and Postgres for replication.
- **Disable HAPI Web Testing**: The HAPI FHIR Server Web Testing UI should be disabled. This tool allows the viewing of demographic records on the server. It is not a tool suitable for production, or if it is used, then it is restricted to a local subnet and further restricted with user and node authentication.
- **Double-check for default passwords**: Ensure no default passwords are in use for OpenHIM core and console, HAPI FHIR Server, ES, Postgres, and other services.

Authentication, Authorization, and Auditing

User Authentication

- Point-of-service systems should possess an appropriate identity provider solution built-in or externally. This also means following best practices for user authentication.
- The Client Registry Service and admin interface are the only direct access to demographic data systems that should require user authentication. As this Client Registry is meant for further customization during deployment, JWT and other user authentication solutions are not provided out-of-the-box but can be added.
- It is intended to support user-requested user authentication solutions as use cases are further identified.

Node Authentication

- OpenCR follows best practices outlined in the ITI-19 standard for node authentication.
- Only secure nodes – one with the ability to authenticate itself to other nodes and transmit data securely – should be allowed to communicate with the Client Registry.
- Systems administrators should ensure that all clients must be registered and certificates assigned to them. In production, OpenCR may act as an OpenHIM mediator which provides an extra layer of security. Clients may be registered in OpenHIM.

Audit Events

- All transactions (including queries) are stored in audit events. These events are stored in the HAPI FHIR Server and viewable in the OpenCR UI.
- Being able to view audit events helps administrators run the system and tune queries,

- But, audit events must also be secured as they contain all information about queries. This means locking down the OpenCR by subnet, node, and user authentication.

ATNA Logging

OpenHIM supports the Audit Trail and Node Authentication (ATNA) Integration Profile, which establishes a standard for responsibly storing audit events. It is highly recommended that the OpenHIM be configured as such but only after it is ensured that all default passwords have been changed and the OpenHIM is operating on a local subnet and thus not exposed externally. See the above hardening notes.

See the OpenHIM user guide for information on ATNA configuration.

Non-Production

In non-production settings only may self-signed certificates be created for testing and demonstrations. An example is as follows:

```
openssl req -newkey rsa:4096 -keyout dhis2_key.pem -out
dhis2_csr.pem -nodes -days 365 -subj "/CN=dhis2"
openssl x509 -req -in dhis2_csr.pem -CA ../certificates/
server_cert.pem -CAkey ../certificates/server_key.pem -out
dhis2_cert.pem -set_serial 01 -days 36500
openssl pkcs12 -export -in dhis2_cert.pem -inkey dhis2_key.pem -
out dhis2.p12
```

Last update: April 13, 2020

Backup and Recovery

Backup

The primary datastore is the database of HAPI FHIR Server. This means that while an ES cluster should be backed-up, the ES index can be rebuilt from HAPI.

- Either Postgres or MySQL are recommended to be used with HAPI FHIR Server. In production, database should be cloned or replicas created and cloned and those backups tested.
- Depending on the database, there are separate processes to backup data itself in a database and information about users, groups and other metadata.
- It is recommended to create a backup and recovery policy (data and metadata, timeframes, full-versus-incremental, from replicas or not).

Recovery Policy

- Backups should be tested in a non-production system for their ability to be used for recovery. There are existing online resources on how to test backups.
- A backup policy should include scheduled recovery tests to ensure that backups are suitable.

Last update: April 13, 2020

Troubleshooting

Last update: April 13, 2020

Add Algorithms

25 types and variations of the most common algorithms are supported by OpenCR, including those through the ElasticSearch plugins: analysis-phonetic and string-similarity

If more are required or revisions needed, the recommended approach is to:

- Write and test the primary algorithm code as an ElasticSearch plugin. This ensures performance, usage in the broader ElasticSearch community, and a platform to test for accuracy.
- Add hooks into the OpenCR Service to support the algorithm.
- Add a decision rules template to show how to use the plugin for end users.
- Determine if the plugin is required. If so, see the code in this config

Last update: April 23, 2020

Building Documentation

This documentation is built using MkDocs and the Material for MkDocs theme.

PDF export is done using mkdocs-pdf-export-plugin. All configuration information is in mkdocs.yaml in the repository. Note that at some future time the docs may be migrated into the main client registry repository.

Edits to docs are made in the master branch of the client registry repository docs repo.

After docs are edited, they are pushed to origin master, and then the `mkdocs gh-deploy` is run on the command line. This pushes into the gh-pages branch on GitHub. Only master is ever edited. The gh-pages is only modified by the CLI.

Last update: April 13, 2020

License

License

CR Service

The CR Service is under a permissive license.

HAPI FHIR Server

HAPI is licensed under the Apache 2.0 license. See: https://hapifhir.io/hapi-fhir/license.html.

ElasticSearch

ElasticSearch is Apache 2.0 and primarily supported by Elastic.co who offer open and open plus proprietary releases and a stack of apps based on ElasticSearch (like the Kibana dashboarding platform).

This CR implementation only uses core, open features, not enterprise features. Netflix and Amazon Web Services also have a distribution of ElasticSearch that is 100% open source and that they have added open source enterprise features to like authentication. Developers who want to test other distributions and features of ElasticSearch are welcome to do so and encouraged to share their experiences with the maintainers.

Open Distro for ElasticSearch

Last update: April 13, 2020

Contributing

There may be many areas of potential contribution as OpenCR is not one application, it's several and can be more than that in your use case.

It's recommended that you identify the specific feature or use case that needs support and

For a quick question, reach out on the iHRIS Slack team. Sign up here

For a bug or feature, reach out to the relevant repository to share the information. See the developer page for links to the different applications.

For a broader discussion with others interested and with a background in Client Registry implementation science, please join the OpenHIE Client Registry Community calls and get involved.

Last update: April 13, 2020