Instant OpenHIE Technical Architecture

Links

Instant OpenHIE Docs Site
Preliminary one-pager for wiki
Slides from 2019 Community Meeting with Edits
Proposed components

The OpenHIE Discourse and monthly OpenHIE Dev-Ops Community calls are for regular communication and updates.

Project Team

Instant OpenHIE is an OpenHIE community project; all contributions are welcome. It is primarily meant as a volunteer effort by existing product owners. Initial support is provided by Digital Square with launch coordination by Jembi and IntraHealth.

Why Instant OpenHIE?

It is critical to exchange health information to reach UHC and the SDGs. However, exchanging health information is complex and expensive.

- Patient security and privacy are paramount.
- It is technically costly to launch and test an HIE.
- There is an ever-expanding set of use cases, scenarios, workflows, and components.
- Components are written in many programming languages and require diverse systems administration skills to demonstrate use cases, let alone move into enterprise-ready production environments.
- Data sovereignty and locality regulations and guidance dictate that HIE and related components are hosted in-country. This makes it difficult to demonstrate HIE use cases, train persons on advanced IT skills, and manage running systems.
- Many implementers need or start with HIEs for one project or a narrow national use case, then face the challenge of scaling up and out into new areas.

Objectives

The Instant OpenHIE project aims to make it possible for anyone to get started with HIE use cases to see how they can address the SDGs and UHC. It will provide for:

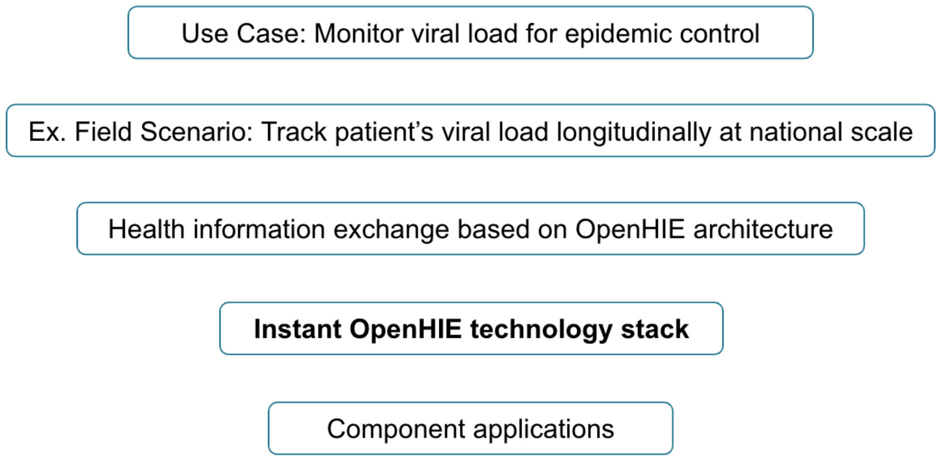
- Easy demonstrations of key UHC and SDG workflows using an HIE based on the OpenHIE architecture.
- Hands-on and practical training.
- Reduced costs and skills required for software developers to deploy an OpenHIE architecture for quicker initial solution testing.
- As a starting point for faster production implementation and customization.

Instant OpenHIE will be a simple way for technical persons to install and see a complex system working against a real-world use case. It will allow users to illustrate how interoperability can work to solve health challenges and show how UHC and SDG health use cases are addressed with open-source software and standards.

Where it fits

Instant OpenHIE is one layer of an answer to the question, ‘How do I get started with using health information exchanges to meet UHCs and SDGs?’ The Technical Architecture document is the entry point for understanding how to get started and make use of the stack, as well as contribute to it.

Example of where Instant OpenHIE fits


Use Case: Monitor viral load for epidemic control
Ex. Field Scenario: Track patient’s viral load longitudinally at national scale
Health information exchange based on OpenHIE architecture
Instant OpenHIE technology stack
Component applications

Roadmap

At maturity, Instant OpenHIE activities will provide portable, launchable versions of multiple OpenHIE components to facilitate:

1. Demonstrable reference products -- those that align with the OpenHIE Community's vision for low resource contexts

2. Rapid software development of mediators and point-of-service systems by making it possible to launch several applications easily so the developer can focus on their task
3. Reproducible, version-controlled infrastructure for user-contributed tests of the OpenHIE Architecture profiles, workflows, and use cases.
4. Production-ready containers and orchestrated components that are deployable in a range of contexts.

The first phase of activities addresses items 1-2 with a focus on a particular use case and set of packages, while production-ready deployments and infrastructure for testing will be built incrementally upon the innovations and lessons learnt from the efforts of the earlier phase.

The first phase focuses on:

- Consultations with the relevant communities on profiles, use cases, and workflows, including with the OpenHIE Architecture Community and DevOps Community.
- Capturing the community feedback into an Instant OpenHIE Technical Design document for sharing. The Technical Design Document is the entry point for understanding how to get started and make use of the stack as well as contribute to it and its future.
- The initial efforts at creating a **core** prototypical health information exchange using open standards and open-source software to help developers add interoperability to your own products.

User Stories

Instant OpenHIE will facilitate demonstrations, rapid software development, and reproducible tests as they become available. During the first phase, a core subset of components will be stood up that will focus on a limited but fundamental set of user stories. Consider the following set of core user stories:

- A user wants to **demonstrate** a data use scenario for HIV epidemic control. In this use case, the data managers need to provide a coherent, deduplicated list of facilities in their HMIS to support the rapid reporting of program indicators and thus inform health policy.
- A user wants to **develop** a software workflow for family planning and reproductive health. For example, MOH is correlating low-performance family planning outcomes in certain districts with the recency of health workforce training.
- A user wants to **confirm** that a recent version of a product passes workflow tests before recommending the implementation for deployment.

Each package that is added to Instant OpenHIE will expand the core infrastructure with additional functionality. Each package will define additional user stories that will be available if those packages are deployed.

Healthworkforce user stories

In the first phase only the Healthworkforce package will be available. It will sync facility, health worker and service data into a central FHIR store. This FHIR store will allow mobile care services discovery(mCSD) queries to be made to interrogate this interlinked data. The following are examples of the user stories that it will enact:

- A patient, June, wants to find out which facilities in their location that do HIV testing and ART treatment. She is able to use a public app on their phone to look up the contact details and locations for these facilities so they may go there for treatment. (Service -> Facility)
- A doctor, Joseph, at a rural clinic wants to refer a patient, Mousa, to an Oncologist because of a lump that they suspect may be cancerous. They are able to look up a list of specialists that offer that service in their EMR system. They choose a particular specialist and find out the facilities in which they work. A referral can now be produced by the EMR for the patient and they can be sent to that facility for their visit. (PractitionerRole -> Practitioner -> Facility)

It will be easy for additional packages to be created and plugged-in to add more user stories to the Instant OpenHIE portfolio.

How it Works

Instant OpenHIE will provide multiple sets of scripts to configure and setup HIE components for particular use cases and workflows. Each of these are organised into self-contained packages and each of these packages may depend on other packages. This allows highly complex infrastructure to be setup in time once more packages are created.

In this first phase, a **core package** will be created that will provide some of the most fundamental components that all other packages will use. In addition, a **health workforce package** that builds off core will also be created to showcase more concrete workflows and a particular reference use case. It will also demonstrate how additional packages can be created to extend Instant OpenHIE in the future.

In particular, the use cases that the health workforce package will add are described below:
- A patient wants to find out which facilities, in their location, do HIV testing and ART treatment. They are able to use a public app on their phone to lookup the contact details and locations for these facilities so they may go there for treatment. (Service -> Facility)
- A GP at a rural clinic wants to refer a patient to an Oncologist because of a health issue that they suspect may be cancer. They are able to look up a list of specialists that offer that service in their EMR system. They choose a particular specialist and find out the

facilities in which they work. A referral can now be produced by the EMR for the patient and they can be sent to that facility for their visit. (Service -> Practitioner -> Facility)

Each of these packages will contain scripts that will setup containerised applications, configure them and ensure necessary metadata is loaded into them. Docker will be used to containerise each of the necessary applications and to enable them to be easily deployed. Two modes of deployment will be supported: local and cloud. Docker Compose will be used for local deployment as it has the lightest footprint and will give the highest performance for local development. Kubernetes will be used for doing cloud deployment as it is the most popular orchestration platform available at present. An ability to do a local Kubernetes deployment using Minikube (kubernetes bundled into a virtual machine) will also be supported to allow users to explore a local Kubernetes deployment as well.

User Experience

Usage depends a great deal on someone's use case and their comfort with the command line.

- **Experienced users** can clone with GitHub repository, decide on their stack (core + whatever additional apps they wish for their use case), and then run a container workflow for their use case. Experienced users would have to be familiar with git and be able to know or learn Docker/Kubernetes, which is available for Mac, Windows, or Linux.
- The initial contributors are exploring ways for **users unfamiliar with the command line** to be able to run Instant. This is done by creating a downloadable application that does the git and Docker/Kubernetes commands for the user, who only has to use a simple web interface. The preliminary design goals for the native application are that:
  - Easy-to-use: The user doesn't need to know the command line, git, Docker, Kubernetes.
  - There are minimal dependencies: Only Docker and Docker Compose have to be installed.
  - Native: Anyone on any platform can use it, so there are binaries for Windows, macOS, Linux.
  - Simple and opinionated: Narrow scope and limited options. It just works.

Not in Scope

Deploying and managing private health information on patients and providers is among the most sensitive of any data. It is critical to ensure security and privacy, backups and data recovery, authentication, authorization, and other enterprise standards. At maturity, Instant OpenHIE would provide production-ready containers and some orchestration assets, such as Kubernetes manifests, but these would be borrowed from Instant OpenHIE and still managed by implementers, who are responsible for databases, upgrades, privacy, security, backups and recovery, authentication, authorization, and other production-ready concerns. Instant OpenHIE would be a way for implementers to develop their tooling around the OpenHIE Architecture and the versions of it, rather than as a substitute for enterprise HIS implementation, support, and management.

The Instant OpenHIE project may provide tests for the OpenHIE architecture, to the extent that the use cases dictate the need for them. However, tests are not comprehensive, not for conformance, and should embrace a long term strategy to align with IHE or other relevant bodies.

Partnerships, especially with regard to security, privacy, and standards-testing are critical, including future coordination with IHE and other entities to ensure alignment and to follow best practices of the leading institutions and prevent duplication.

Deploying to the cloud will be supported via managed Kubernetes, however, there is only limited budget for cloud hosting throughout the project so hosting services may be spun up to support demonstrations or testing. Continuous cloud hosting of a demonstration environment can only be supported if budget allows and can only be supported until the project comes to a close.

Architecture

This section will outline the logical architecture of the components of Instant OpenHIE for this first phase. It will detail the following:

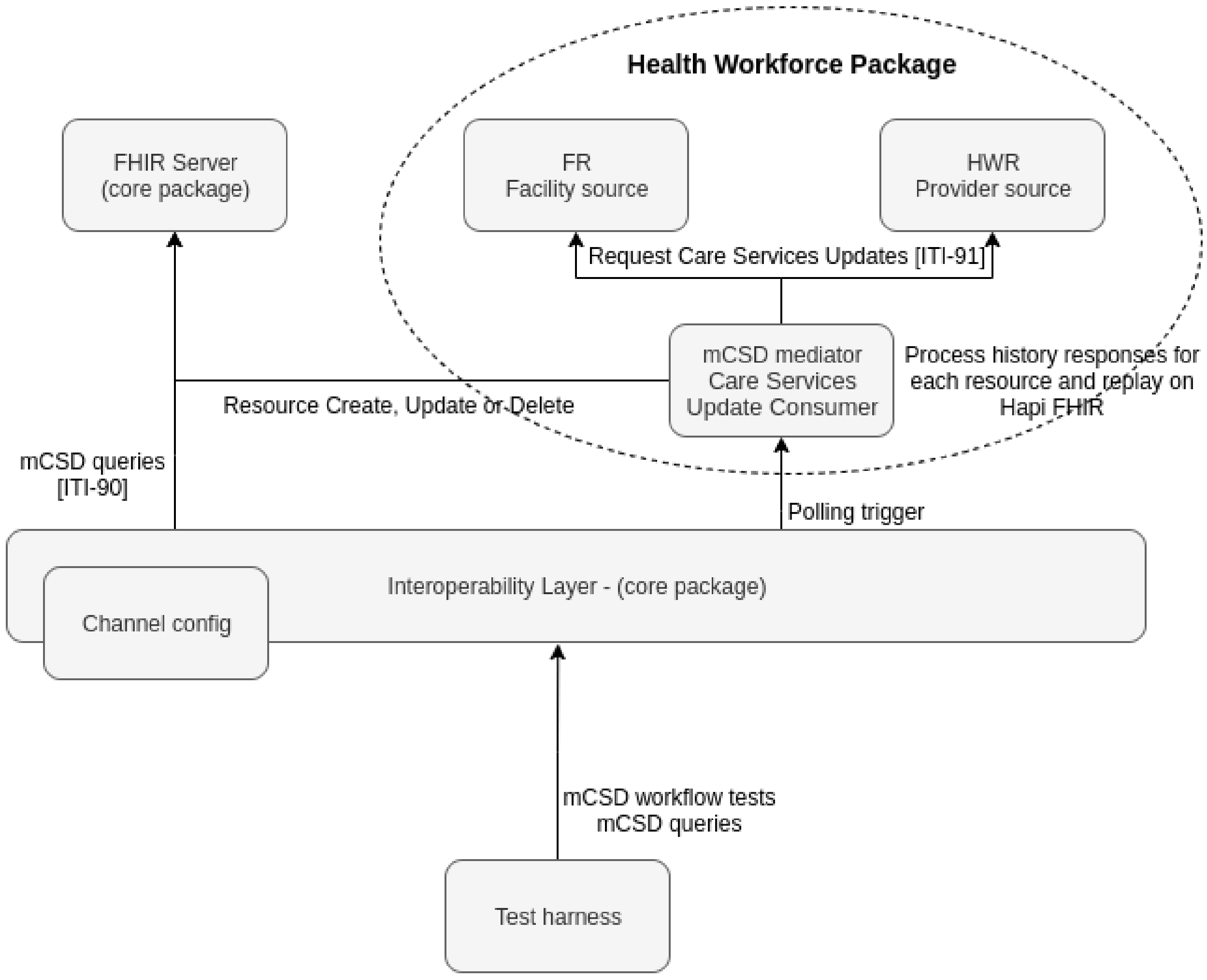
- Components that will be involved in the **core package** and **health workforce package**
- Profile and data flows that Instant OpenHIE will support
- Infrastructure and deployment strategy

These approaches will be modified and updated as the project progresses and more is learnt. The current architecture shown describes the planned approach that appears to be the most logical at the outset.

Architecture for Core and Health Workforce packages

Two packages will be produced in the first phase, the core package and the health workforce package that extends from the core and adds health workforce-related functions and metadata.

The following diagram shows a generic logical architecture showing the involved OpenHIE components. Each of these components' roles could be played by any application that supports the necessary OpenHIE workflows and component requirements. It would be possible to swap out applications for others as long as they conform to these specifications. In the future, Instant OpenHIE could support multiple application options for each role.

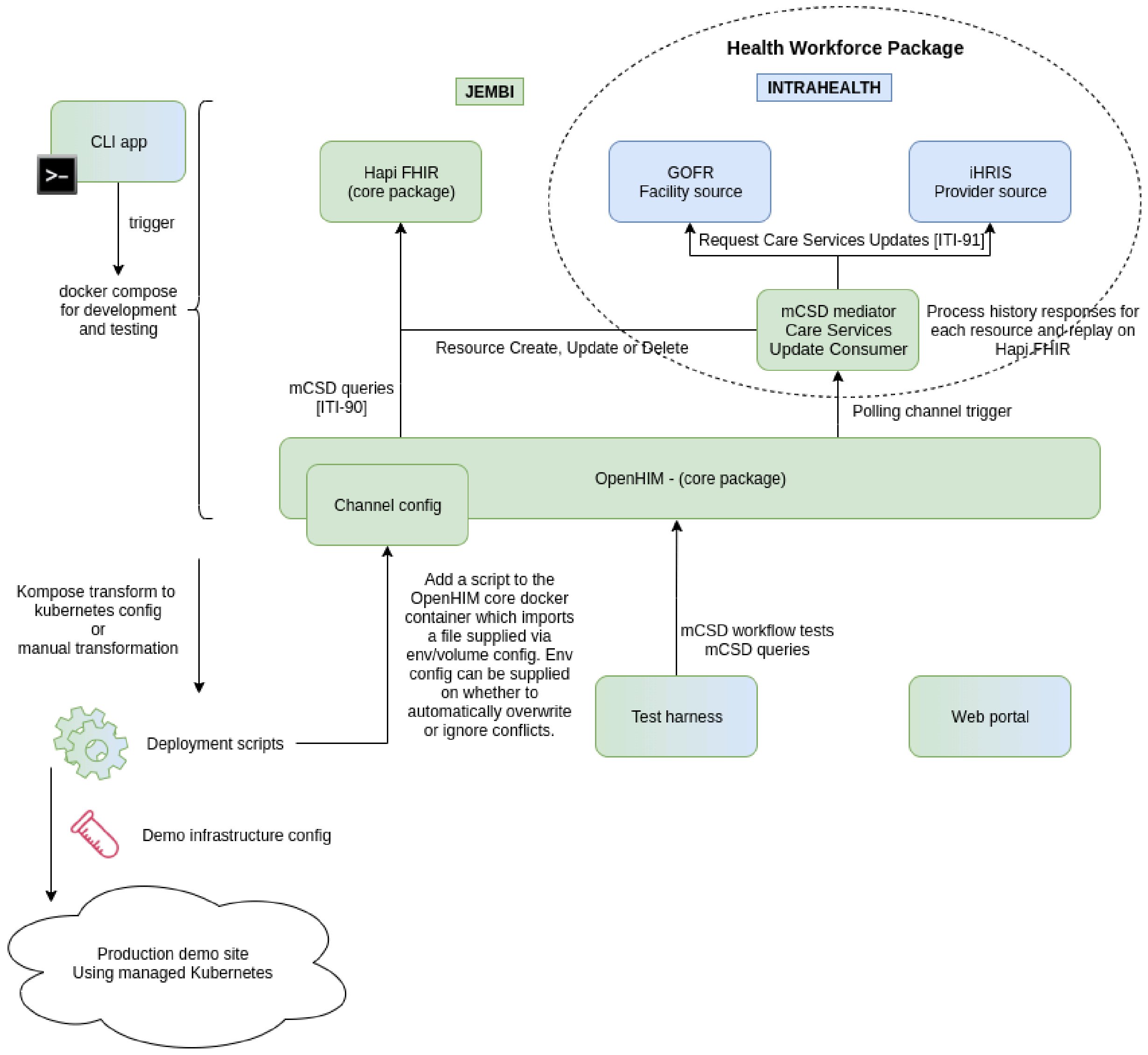

Health Workforce Package
FHIR Server (core package)
FR Facility source
HWR Provider source
Request Care Services Updates [ITI-91]
mCSD mediator Care Services Update Consumer
Process history responses for each resource and replay on Hapi FHIR
Resource Create, Update or Delete
mCSD queries [ITI-90]
Polling trigger
Interoperability Layer - (core package)
Channel config
mCSD workflow tests mCSD queries
Test harness

[diagram source]

For phase 1 particular applications have been chosen to support the required roles. The diagram below shows the complete architecture, with the selected components applications, for the work that will be completed in phase 1. This includes:

- The selected applications that will be included to play each role
- The separation of packages
- The project teams that will be working on each component of the architecture
- A depiction of the deployment strategy for development, testing and production

These components and their actions are further described in the sections that follow.


Health Workforce Package
JEMBI
INTRAHEALTH
CLI app
>_
trigger
docker compose for development and testing
Hapi FHIR (core package)
GOFR Facility source
iHRIS Provider source
Request Care Services Updates [ITI-91]
mCSD mediator Care Services Update Consumer
Process history responses for each resource and replay on Hapi FHIR
Resource Create, Update or Delete
mCSD queries [ITI-90]
Polling channel trigger
OpenHIM - (core package)
Channel config
Kompose transform to kubernetes config or manual transformation
Add a script to the OpenHIM core docker container which imports a file supplied via env/volume config. Env config can be supplied on whether to automatically overwrite or ignore conflicts.
mCSD workflow tests mCSD queries
Test harness
Web portal
Deployment scripts
Demo infrastructure config
Production demo site Using managed Kubernetes

[diagram source]

mCSD mediator and FHIR metadata store

An mCSD mediator will be developed for the OpenHIM that can sync data from the Care Services Update Suppliers and then forward each received record to the FHIR store which can then in turn act as a Care Services Selective Supplier. It will use the Request Care Service Updates [ITI-91] transaction from the mCSD profile to pull this information from each source system (in this phase those source systems will be GOFR and iHRIS). It will process each resource returned in the history bundle in order of original execution and apply the effect (create, update or delete) to the FHIR store. Thus the FHIR store becomes the central metadata store for all the source systems.

The FHIR store will then act as a Care Services Selective Supplier to clients of Instant OpenHIE by supporting the Find Matching Care Services [ITI-90] transaction. This enables clients to query for a variety of interlinked metadata.

The mediator will be triggered by an OpenHIM polling channel which gives us a configurable scheduled trigger. The mediator will request updates from each source system each time it is triggered and it will remember when the last time it sync'd was so that only new updates are queried.

Generic Architecture

Above we discuss the architecture for phase 1 of the project where we will be implementing an mCSD use case. However, in the future Instant OpenHIE is envisioned to cover a number of user cases with additional packages being implemented and included in the default installation.

Due to this, it makes sense to extract the generic architecture (which is package agnostic) of Instant OpenHIE and to make it available as part of the user facing documentation to drive adoption and understanding. This also allows it to evolve with the project independent of this document. This generic description of the architecture can now be found here (link broken until we do a docs deploy) instead of in this document.

Contributing

Instant OpenHIE should be extensible so that others may base their architectures off of the base that will be created. By splitting the functionality into particular packages others are able to choose the specific packages of functionality that they require and add new ones for implementation-specific functions.

There are two broad areas of contributing:

- New apps and packages added to the Instant OpenHIE stack
- New use cases, workflows, and tests to validate and make use of it.

Instant OpenHIE uses containerized applications and should rely on existing container images provided by product owners, not build one-off solutions. App owners should also provide a docker-compose file to demonstrate how to launch a running stack for their product.

App owner responsibilities	**Description**

Tagged releases	Releases should be tagged in git or other version control system and in a public repository.
Environment variables	Configurations must be stored in or be able to be overridden by environment variables. See the Twelve Factor App: https://12factor.net/config
Dockerfile	Create a publicly available Dockerfile used to build the image and a link to it.
Container image	Make available a link to a public image of the application. A tagged release image should be available.
Docker Compose	Provide a link to a versioned Docker Compose script. A Docker Compose file should exist for running the application stack, including databases or web servers or other needs. Where possible use existing containers for things like databases or web servers. Slim images (e.g. Alpine) are recommended as many images will be run concurrently.
Automated configuration	Provide detailed information or scripts that can run in a non-GUI environment for automated configuration.

Ideally, component owners should become Instant OpenHIE maintainers. Component and use case/workflow owners and Instant OpenHIE Maintainers are often going to be the same set of persons because component (app) owners generally also contribute to the definitions of the architecture.

Workflow owner responsibilities	**Description**
Test data	Generate fake but realistic data for E2E tests and general functionality.
Package test data	Make test data available or reproducible online.
Tests	Write tests for the expected workflows supported using the containers, configuration, and fake data.
Dockerfile	Tests can be written in any language. Provide a Docker container for tests so that they can be run easily (with environment variables) against any stack (not just Instant OpenHIE)
Container image	Make available a link to a public image of the tests. A tagged release image should be available.

Risks

There are several challenges that may arise in this set of activities

- Fake data for data workflows is fragmented, inconsistent, incomplete, and fragile. This means that, for example, to demonstrate an HIV use case the data is not available that would be used to move from component to component. This could slow the process of demonstrating Instant OpenHIE considerably.
- It is important to define and manage scope creep. The tools developed for Instant should not be considered production-ready. It is important to ensure all stakeholders understand the limits of the project.
- Additional steps will need to be addressed before Instant OpenHIE can be used in production setting:
    - Security configuration of databases
    - Password storage and rotation
    - Backup scripts

License

The documentation, fake data, and code developed should be made available under a permissive open-source license.