18X024Q HCMI Software Architecture

Version History

Version	Implemented By	Revision Date	Approved By	Approval Date	Description of Change
1.0	Rahul Verma / Francois Gerthoffert	April 27th, 2018	Vincent Ferretti	-	Initial Version
1.1	Rahul Verma	May 9th, 2018	Vincent Ferretti	-	Updated installation instructions

Table of Contents

Introduction

Purpose

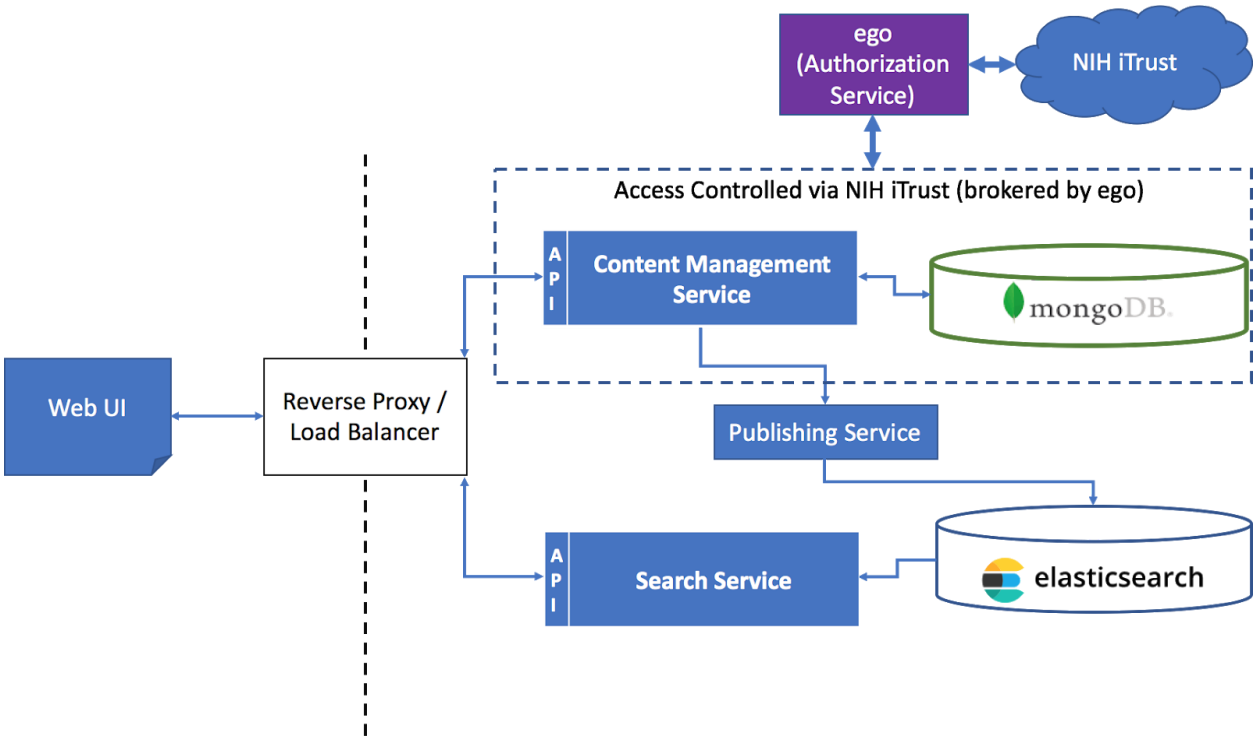
The purpose of the Human Cancer Model Initiative (HCMI) is to collate model data from several sources and provide a web-based query tool to search for models of interest. The data collated in the catalog will include key model variables deemed the most important features upon which investigators would make selection decisions. The data will include a range of tissue donor demographic and clinical diagnostic information, summaries of somatic genomic changes, and administrative information. The catalog will be searchable, and search results will be downloadable and include pointers out to primary data sources.

The intended audience of the 18X024Q HCMI Software Architecture document is to provide stakeholders and technical teams with an overview of all components implemented and deployed for the project.

Architecture Overview

The system is composed of two major software components:

- A public facing catalog UI providing users with the ability to search and view models
- An access controlled content management system to manage data available in the catalog



System architecture diagram

Architecture Overview

The HCMI Catalog will be composed of four loosely-coupled micro services. These services will be seamlessly integrated with Web UI. The objective is to keep the system modular and extensible. Another objective is to ensure that each component can be scaled out independently as scale and scope of the application changes. Each component of the system and their domain areas are described below:

Software Components

Web Frontend (UI)

The catalog frontend is a React application (https://reactjs.org/). The application will call the

content management and search services through their REST or GraphQL (https://graphql.org/) APIs.

Source code is available here: https://github.com/nci-hcmi-catalog/portal

Software code, stylesheets (CSS) and images are considered publicly available static assets and can be (safely) served from a dedicated server or from a CDN such as Netlify.

Search Service

The search service is a NodeJS (https://nodejs.org/en/) API supporting all the front-end search and browse capabilities. It uses Elasticsearch (https://www.elastic.co/products/elasticsearch) as its data store.

Ego

Ego (https://github.com/overture-stack/ego) is a stateless JWT based authorization service built in java and aimed at providing single sign on capabilities to microservices.

This service will be used to broker access between NIH iTrust and controlled features of HCMI Catalog, in particular for it's content management features.

Content Management Service

The content management service is a NodeJS Express (https://expressjs.com/) based REST API that uses MongoDB (https://www.mongodb.com/) as data store.

Authorized (through ego) content managers will be modifying content through the catalog frontend. The content management service will then interact with the publishing service to trigger the indexing of documents into Elasticsearch.

Publishing Service

Invoked by the content management service or by a batch process, the publishing service index documents from MongoDB into Elasticsearch.

Continuous Deployment

We will revise the deployment plan as the project progresses. Some of the candidates for continuous deployment are:

1. Jenkins
2. TravisCI
3. CircleCI

All above options have a Free Tier that is sufficient for the needs of HCMI project. However, if we decide to use Jenkins; we will have to setup our own VMs to execute Jenkins - that will incur the cost of running the VMs.

Deployment Summary

The HCMI catalog has been designed as a scalable and highly available resource, resources below can be adjusted but is considered by the engineering team as the optimal configuration for currently known use cases and features.

June deployment (search only)

Component	Resource requirements	Deployment Summary
Web UI	CDN	Use Netlify CDN for static assets (free, auto deployable and previews available for each new build)
Publishing/ Search Service	AWS EC2 m5.xlarge equivalent: vCPU: 4 Memory: 16 GB Storage: 20 GB	1. Install nodejs globally (nvm?) https://github.com/creationix/nvm 2. We use yarn too https://yarnpkg.com/lang/en/docs/install/ 3. Install docker
Reverse Proxy	AWS Reverse proxy/Load Balancer to direct all traffic to certain ports	Load balancer serving as proxy to all backend services
MongoDB	AWS EC2 m4.large equivalent: vCPU: 2 Memory: 8 GB Storage: 10 GB	Install mongodb: 1. Disable last access time setting for files on the server 2. Make sure mongo journaling is on 3. Setup a db path and make sure it is accessible by mongo process 4. Run mongodb pointing to dbpath
Elasticsearch	AWS EC2 m5.xlarge equivalent: vCPU: 4 Memory: 16 GB Storage: 20 GB	1. Install Oracle Java 8 on ES server. 2. Make sure the target Elasticsearch server is publicly accessible (Dev only) 3. Increase ulimit for open files on server to 64K. 4. Setup ElasticSearch 6.x a. Bind Elasticsearch to public IP of the server b. Leave default port to 9200 5. Setup Kibana a. Bind Kibana to public IP of the server b. Leave default port to 5601

September deployment (all features) (may be revised after June)

Component	Resource requirements	Deployment Summary

Web UI	CDN	Use Netlify CDN for static assets (free, auto deployable and previews available for each new build)
Content Management, Publishing & Search Service	A ***cluster of 2 VMs*** each of them equivalent to an AWS EC2 m5.xlarge: vCPU: 4 Memory: 16 GB Storage: 20 GB	1. Install nodejs globally (nvm?) https://github.com/creationix/nvm 2. We use yarn too https://yarnpkg.com/lang/en/docs/install/ 3. Install docker
Reverse Proxy	AWS Reverse proxy/Load Balancer to direct all traffic to certain ports	Load balancer serving as proxy to all backend services
MongoDB	***3 VMs:*** 2 of them of equivalent to an AWS EC2 m4.large: vCPU: 2 Memory: 8 GB Storage: 10 GB 1 more VM equivalent to an AWS EC2 t2.small: vCPU: 1 Memory: 2 GB Storage: 8 GB	Install mongodb: 1. Disable last access time setting for files on the server 2. Make sure mongo journaling is on 3. Setup a db path and make sure it is accessible by mongo process 4. Run mongodb pointing to dbpath
Ego	AWS EC2 m5.large equivalent: vCPU: 2 Memory: 8 GB Storage: 20 GB (Might be able to host it on same EC2 cluster as other services - but need more discussion with iTrust on that)	1. Setup Postgres Database 9.5.x 2. Setup application frameworks: a. Install Oracle Java 8 b. Install Maven 3.5.x c. Install docker
Elasticsearch	***3 VMs:*** Each VM equivalent to AWS EC2 m5.xlarge vCPU: 4	3. Install Oracle Java 8 on ES server. 4. Make sure the target Elasticsearch server is publicly accessible (Dev only) 5. Increase ulimit for open files on server to 64K.

	Memory: 16 GB Storage: 20 GB Storage: 20 GB	6. Setup ElasticSearch 6.x a. Bind Elasticsearch to public IP of the server b. Leave default port to 9200 7. Setup Kibana a. Bind Kibana to public IP of the server b. Leave default port to 5601

Logging and Monitoring

Each service will create log files in a directory (as suggested by NIH). NIH Infrastructure team will take care of aggregating logs from these log files in a central place.
We can also look into setting up Google analytics if we want to gather usage/user behavior logs.