Radiology in the Cloud Project Proposal

Vision and Goals Of The Project

Boston Children's Hospital(BCH) has developed system to house and query medical images, which they refer to as the Children's Research Integration System, or ChRIS for short. This system is composed of the following components:

- The web server that provides a RESTful interface for data collection, sharing, and process management
- The ability to query images from a standard resource
- A web front end GUI
- back-end that orchestrates the execution of jobs on remote resources
- A data controller that fetches/retrieves data from different services and across networks

BCH hopes to extend ChRIS to be able to offload its data processing onto a High Performance Cluster(HPC) in the MOC. In order to accomplish this, the BHC team has "mostly Dockerized" the above listed elements of ChRIS so they can be more easily be deployed in different environments. As implied by the phrase, "mostly dockerized," some of the components of ChRIS have not been completely adapted to work from within a container, and must be updated in order to function correctly. So, we will have to get the ChRIS containers to a phase of development where they can properly run and interact with each other, then to deploy a HPC in the MOC and run the containers dedicated to processing medical images on it.

Users/Personas Of The Project

ChRIS is and will be used by clinicians and researchers affiliated with the BCH. It targets non-technical users who will be interacting with its web-based GUI. Patients or the general public will not have access to ChRIS.

The work that we will be doing for the project will also be for system administrators at hospitals who plan on deploying their own instance of ChRIS. Currently, it is very difficult to set up ChRIS, however, enabling ChRIS to run in a dockerized cloud environment should make it much easier to deploy and manage.

Lastly, the changes we are making to ChRIS will make it easier for developers looking to test algorithms that process medical images.

As a result, our users and personas include clinicians, researchers, developers from both BCH and other hospitals including administrators, and people who want to deploy the program since this is a multi-sided service.

1. Scope and Features Of The Project

- Connect/tweak/adapt the existing search plugin in ChRIS to be able to query/retrieve images from the PACS Orthanc server
    - REST API's are currently implemented, but the back end needs to be modified to be able to use them
- complete the connectivity on the front end to an image server, which in our case will be an open source version of a PACS (Picture and Communications Service) called Orthanc
- complete/test the code that pushes/pulls data from the ChRIS host to the MOC, pushing the PACS data and pulling results back to the ChRIS file system
- Deploy an HPC on the MOC that is capable of running resource intensive processing algorithms on medical images. This HPC deployment should be able to:
    - Receive instructions from a ChRIS back end at the BCH;
    - Create a plugin between the remote HPC and ChRIS in its own docker container;
    - Receive images to be processed from the ChRIS back end;
    - Be able to send the processed images back ChRIS to be viewed on the web front end by a user.

2. Solution Concept

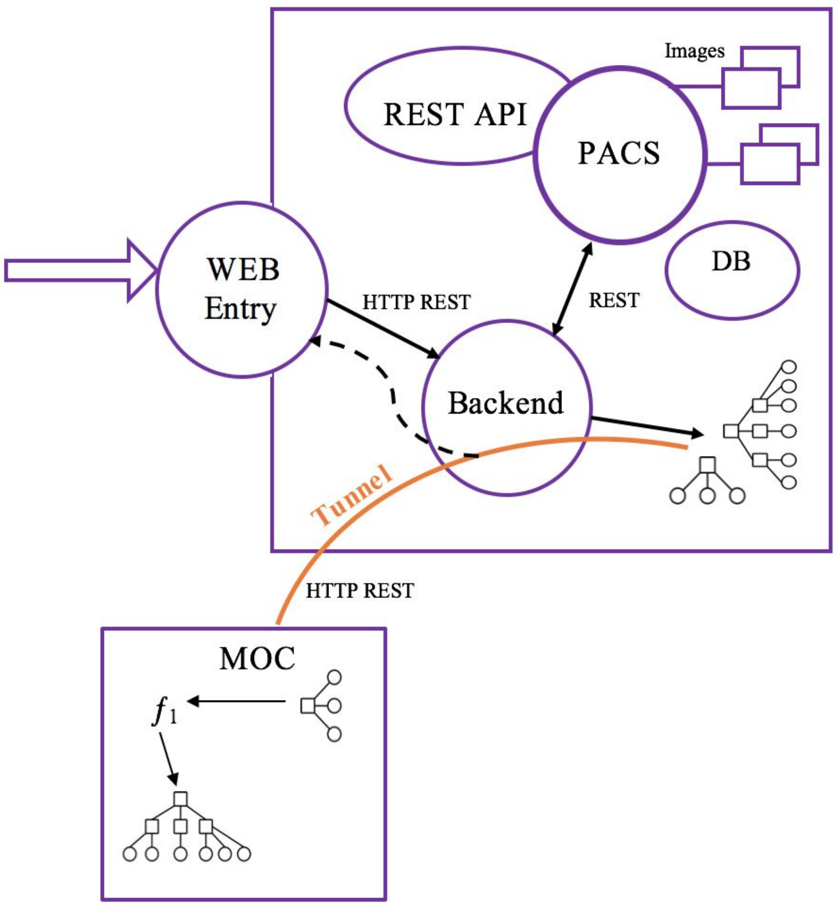
Pre-Existing Architecture

- PACS: a medical imaging technology which provides economical storage and convenient access to images from multiple modalities (source machine types).
- Web Client: the website, which is run on hospital servers and provides a UI for users to communicate with the PACS and retrieve their images. It has also been extended to support plugins that would allow the images stored in the PACS to be sent to a high-performance compute node where users could run advanced algorithms over them and then view the results in the web client. Currently, this system is implemented locally in the UI and backend only. The development version is not able to schedule any jobs on a remote resource, such as a High Performance Computer (HPC).
- REST APIs written to streamline communication between the front and backend of ChRIS.
- Various analysis jobs to process image data

Architecture To-Be-Implemented

- MOC hosted HPC
- Data plugin between the ChRIS backend and HPC
- Process management control of the ChRIS system

Architectural Diagram of the Project



Walkthrough

The diagram above shows the intended outcome of this project. The website, the back end, and the PACS server shown above are all hosted at the Boston Children's Hospital, which is why they are grouped together in that square. Then, the lower box represents the HPC that we will be setting up in the MOC. The orange arrow represents the data plugin between the BCH servers and the MOC that we will be setting up in order to allow the two clusters to communicate. We should be able to send secure medical images over this plugin and into the MOC for processing. Then we should be able to use it to send those images back to the BCH servers to be viewed by doctors.

Design Implications and Discussion

There are four main components to our project: (1) the transfer of data from the image database to the web server; (2) pushing and pulling data and results respectively to and from the remote resource; (3) using the front end to control job execution on the remote resource; and (4) dockerizing the remote job in its own container.

Getting data from Image database

PACS is an existing server that has a database of patient images. Each image has its own header containing patient information and other details. The PACS server uses this to query content and retrieve data. So, PACS would transmit data to a target host. For PACS to get the query request, another process needs to actively listen on the host for incoming data stream. The stream is parsed and the files are packed onto its file system. ChRIS uses python modules to transfer this image data from the web server file system to the MOC. After the MOC receives the image data, ChRIS starts an analysis on the data.

The existing PACS is in a dockerized instance of a package called Orthanc. For the purpose of the project, the data in PACS would be populated with anonymised data from the mentor. Data from PACS is viewable from the ChRIS front end. For our project, we will complete the PACS query/retrieve to ChRIS so that all queries and retrieve commands work with the current instance of Orthanc.

Pushing data to the remote resource and pulling results back

The data obtained from PACS will be pushed onto the Massachusetts Open Cloud (MOC) using ChRIS. On the MOC, there would be a high performance computing (HPC) infrastructure to run computationally intensive processes. The infrastructure would be dockerized and communicate with ChRIS over an SSH tunnel, ensuring security. The HPC would initiate the push of data onto a target file system and pull data from target to the host file system.

Currently a python module is performing the services. For our project, we will extend the existing structure to push and pull data to/from the MOC using both a python executable and a python module.

Controlling job execution on the remote resource from the front end

The ChRIS development environment contains a module called pman (process manager) that runs a job on a remote resource and keeps track of its status. For our project, we want to extend the ‘pman’ service to work with the MOC. It will be contained in its own docker, and communicate with ChRIS using a RESTful API. Consequently, external entities will also be able to communicate with the ‘pman’ service. These entities, like ChRIS, can use the REST calls to specify a job to run, and monitor the state of the remote job on the MOC.

Executing the remote job(s) in their own containers

When a job is called, each one has its own large set of internal dependencies and requirements. To best satisfy them, the appropriate set of jobs will be identified and dockerized to eliminate the dependency on the MOC and satisfy the requirements. Initially, all the jobs will be in a single docker image. If time permits, each job would ideally run in its own container to increase the granularity and portability of the job system.

Additional thoughts

One separate design discussion that was debated was the possibility of a cache system. To save computation time, the patient information that has already been processed could be stored in a separate database. So when the same function is called by the user interface ChRIS on the same patient, the back end could reference the cache library to quickly provide the output. This would cut out a lot of computation and data transfer time required to recompute the data. A way to implement this would be to create another database with the processed information. Hashing or a key-value pair system could help with the quick lookup in this database. However, this design addition may not be in the scope of this project.

Acceptance criteria

At minimum, the system we set up should be able to:

1. Perform a search on the PACS (Orthanc) container from ChRIS for an image set of interest;
2. "Pull" the image set from the Orthanc container to the web server file system;
3. Handle the back-end communication to transfer the image to some remote service, and on the remote service "run" the plugin selected;
4. Wait for job to finish (i.e. be able to query run status while job is executing);
5. When finished, transfer results back to web server file system and show the user the results.

Stretch goals are being able to create a container for each plugin running.

Release Planning

Detailed user stories and plans will be updated as the project progresses on the Trello board: https://trello.com/b/r5ekthAy/radiology-in-the-cloud

Sprint 1: February 23rd

Setup instances and HPC:

- Implementing docker instances within the OpenStack for both ChRIS and PACS
- Make both instances publically accessible
- Download the existing ChRIS on our instance

Sprint 2: March 16th

Getting data from Image database

- Understand how the Orthanc communicates with PACS
- Modify the communication between PACS and ChRIS to enable querying

Sprint 3: March 30th

Getting data from Image database

- Test querying and data-retrieval
- Show data on the ChRIS user interface on the front end

Pushing data to the remote resource and pulling results back

- Import functions into MOC to create a HPC instance

Sprint 4: April 13th

Pushing data to the remote resource and pulling results back

- Create python executable and python module to push and pull data

Controlling job execution on the remote resource from the front end

- Dockerize ‘pman’ service

Sprint 5: April 27th

Controlling job execution on the remote resource from the front end

- Use the RESTful API to enable communication between the front end and the remote resource to retrieve job status

Executing the remote job(s) in their own containers

- Dockerize the job in one container (or put every job in a separate container if we have enough time)

Sprint 6: May 2nd (Final Presentation)

- Debugging
- Polishing