Radiology in the Cloud Project Proposal

Vision and Goals Of The Project:

Radiology in the Cloud will be a web-based distributed system for the Boston Children's Hospital (BCH). The system is a medical image data and information management software platform called ChRIS ([Boston] Children's Research Integration System). ChRIS allows retrieval of medical image data from a system called PACS. The medical images are pipelined through a High Performance Compute Cluster (HPC) to analyze and process images that the user has requested.

ChRIS was recently dockerized, allowing it to be easily deployed in containers, and hopefully in the cloud. It is expensive and difficult to run a supercomputer on proprietary servers, making it far more practical for BCH to run it in the cloud. Therefore, our goal is to set up a HPC in the MOC that can communicate with ChRIS and run complex algorithms on medical images. The existing ChRIS backend should be able to retrieve images from the Orthanc server, send these images to the remote HPC we set up in the MOC, then be able to retrieve the processed images from the HPC and view them on the web front-end.

Users/Personas Of The Project

ChRIS is and will be used by doctors 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 admins 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 doctors, researchers, developers from both BCH and other hospitals including admins, 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 backend needs to be modified to be able to use them

- 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 backend at the BCH
    - Create a pipeline between the remote HPC and ChRIS in its own docker container
    - Receive images to be processed from the ChRIS backend
    - Be able to send the processed images back ChRIS to be viewed on the web front end by a user

2. Solution Concept

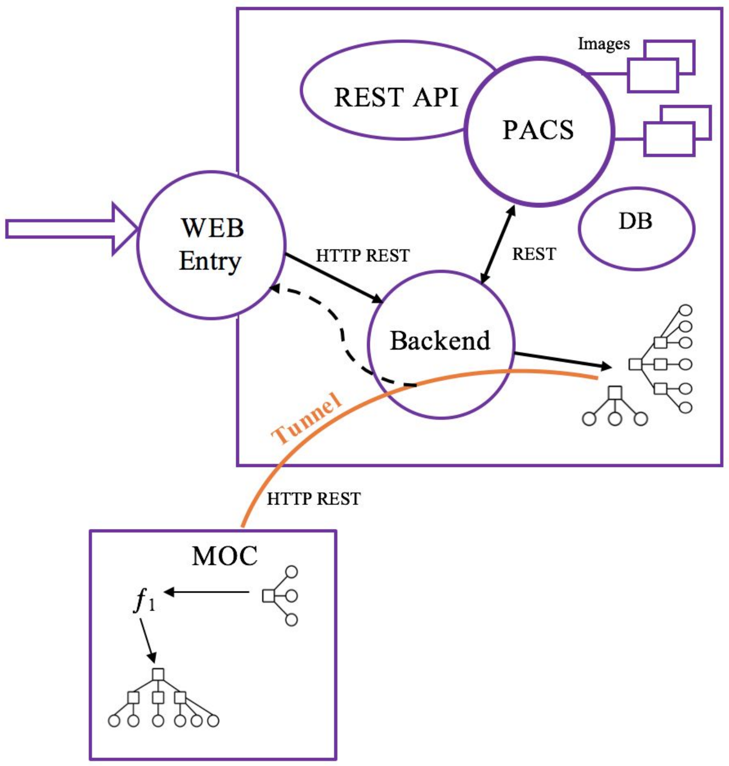
Pre-Existing Architecture

- PACS: a simple, and redundant, unix-based file server that stores medical images in either a DAS, NAS, or SAN configuration
- Web Client: the website is run on hospital servers and provides a UI for doctors 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, and is unable to connect to any actual High Performance Computer (HPC).
    - A backend that runs the algorithms in the HPC has already been written as well
- Rest API's have been written to streamline communication between the PACS and ChRIS

Architecture To-Be-Implemented

- MOC hosted HPC
- Data Pipeline between the ChRIS backend and HPC

Architectural Diagram Of the Project



Walkthrough:

The diagram above shows the intended outcome of this project. The web site, the backend, 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 pipeline 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 pipeline 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 three main components to our project: MOC cluster of functions and database, the backend that sends jobs to and from the cluster, and the user interface.

The MOC will be used as a high performance computing infrastructure to run computationally intensive processes. The service should be modular such that if another program wants to communicate with it, the same RESTful service could be used with the MOC isolated. The dockerization of the service makes this much simpler for even other hospitals to use as a shared service directory that can span multiple regions/providers. The cluster of functions should be able to receive multiple files and send the output to the correct application. Hence, multiple users will be implemented with their own credentials. The cluster will talk to the backend using the RESTful API and a HTTP tunnel. Since this is a highly secured system with classified patient information being passed to and from, HTTP tunnelling will allow the network protocols to be encapsulated using the HTTP protocol within a restricted connectivity system. This could be done behind NATs, firewalls, or proxy servers.

The database of patient information is located in an existing server called PACS. The PACS server uses python modules to transfer data from the MOC to a remote HPC. The new design includes moving the PACS server from a local server to the MOC server that is accessible anywhere with the correct credentials. This will make the database more scalable as well. The user interface and PACS will communicate with SSH and SFTP. The unique private key required to decrypt and access the database will add security to PACS while making it globally accessible.

The design of the user interface called ChRIS ([Boston] Children's Research Integration System) follows the same principles of modularity within a docker system as the cluster of functions. The user interface has been completed and is available online. What needs to modified is a python module that creates data transfer pipeline to the MOC and the backend. The backend of ChRIS uses a search plugin to query the PACS server and obtain relevant patient data. So, information will be channeled from the PACS server to the MOC functions, and then back to ChRIS with the output information. A router could be set up between ChRIS and PACS instances for SSH tunneling.

An important aspect of this project is security. Despite there being HTTP tunneling and SSH tunneling using keys, our design must adapt to prevent unauthorized users from simply replicating the python module and accessing the data from the PACS server. Patient confidentiality is of high priority to the hospitals. So security will be prioritized above the modularity with dockerized components.

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 backend 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 backend 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 pipeline 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 cluster for both ChRIS and PACS
- Make both instances publically accessible
- Import functions into MOC to create a HPC instance

Sprint 2: March 16th

Communication between HPC on the MOC and ChRIS:

- Modify existing python modules to create a standard protocol to communicate with functions in MOC
- Test data pipeline with pre-existing ChRIS on the BCH server

Sprint 3: March 30th

Communication between ChRIS and PACS:

- Modify existing backend for ChRIS to query PACS to obtain the data

Sprint 4: April 13th

Testing and integration:

- Ensure communication is working between the dockerized components
- Co-ordinate with BCH to implement new changes in the user interface ChRIS

Querying run status:

- Pass on the job status and process information to ChRIS to see real time updates on the processing

Sprint 5: April 27th

Full workflow communication:

- Modify python modules in ChRIS to transfer data from the MOC to a remote HPC

Sprint 6: May 2nd (Final Presentation)

Final Touches:

- Debugging
- Polishing