ChRIS: An OpenSource Containerized Service for Pipelining Clinical and Research Computation

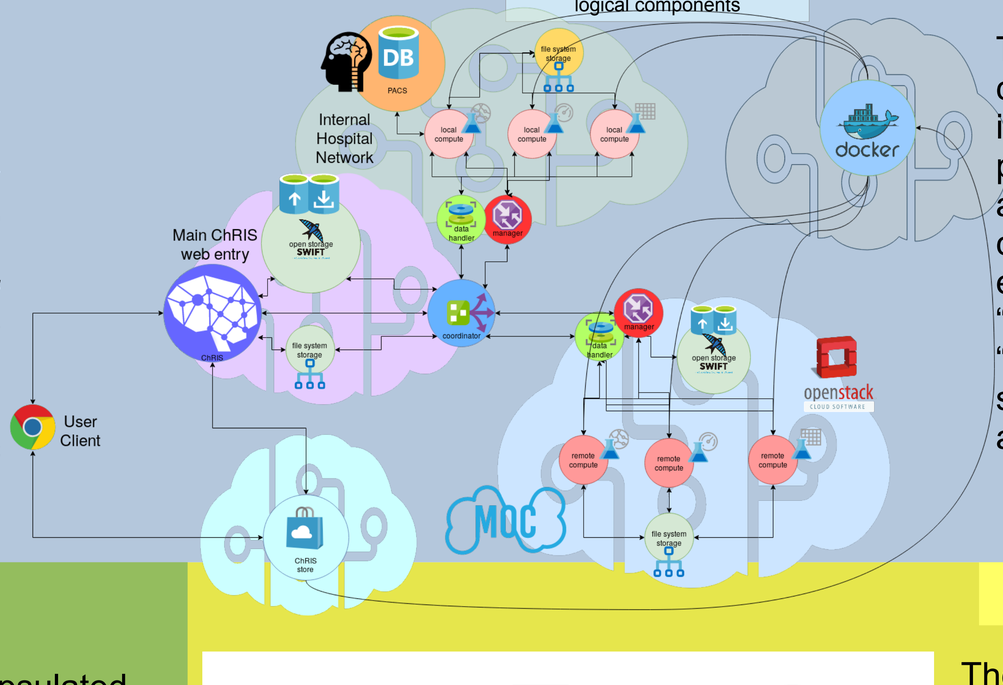
Rudolph Pienaar[1,2], Jorge Bernal-Ruisel[1], P Ellen Grant[1,2], Daniel McPherson[3], Ravisantosh Gudimetla[3], Ata Turk[4], Orran Krieger[5]

[1]Fetal-Neonatal Neuroimaging and Development Science Center, Childrens Hospital Boston
[2]Harvard Medical School, Boston, MA; [3]Red Hat, Inc, Boston, MA; [4]Mass Open Cloud, Boston, MA; [5]Boston University, Boston, MA

introduction

The recursively named **ChRIS**, or **Ch**RIS **R**esearch **I**ntegration **S**ystem, platform is a novel solution geared at addressing the pressing need in computational medical research for a powerful, effective, developer friendly, and completely opensource vehicle for disseminating data and research computation.ChRIS is built to bridge the gap between research and clinical words from a computation perspective and aims to create a community of algorithm-developing scientists, foster collaboration, and greatly accelerate the development and use of computational tools by the medical-scientific and clinical communities. No other platform offers the depth of potential to effect a lasting impact on scientific computation in the “cloud” age. **ChRIS** builds on much prior work in web-based visualization and remote processing [1-4].

Figure 1. **ChRIS** overview schematic showing the main logical components



The main ChRIS logical components are shown in Figure 1 and exist primarily to distribute and manage self-contained computational elements called “plugins”. The different “cloud” graphics denote separate networks and/or clouds.

compute

Within **ChRIS**, the term “plugin” is used to denote the actual encapsulated compute (or program) that is being scheduled, run, and disseminated by the platform. A “plugin” is the irreducible atomic building block of **ChRIS**. As shown in Figure 1, these “plugins” are the “remote compute” circles within the separate clouds.

A plugin is a standalone command line executable program that once initialized, runs with no user interaction. Plugins are in fact fully realized containerized compute that can be run independently of **ChRIS**, a significant characteristic which has important real world implications.

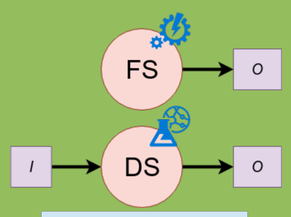


Figure 3. **ChRIS** plugin types

Plugins are of two types, *FS* (Feed-Synthesis) plugins that create output data based on non-filesystem input (e.g. a database query such as a Hospital PACS or similar resource), and *DS* (Data-Synthesis) plugins that create output based on parsing data contained explicitly in a single filesystem directory, as shown in Figure 2. Plugins also have a corresponding JSON representation, as shown in Figure 4.

All plugins are required to create all their output within a single directory, *O*, and for *DS* plugins, also read all data from an input directory, *I*. **ChRIS** guarantees to provide some directory with input and guarantees that the contents of the output directory will be registered by the system. Plugins can be grouped together into tree graphs – subcomponents of a tree can be encapsulated into *pipelines* as shown in Figure 5.

```
$> docker  run  --rm          \
      fnndsc/pl -simpledsapp \
      simpledsapp.py --json
```

Figure 4. **ChRIS** plugins have an JSON


```
{
  "type": "ds",
  "icon": "",
  "authors": "FNNDSC (dev@babyMRI.org)",
  "title": "Simple chris ds app",
  "category": "",
  "description": "A simple chris ds app demo",
  "selfpath": "/usr/src/simpledsapp",
  "selfexec": "simpledsapp.py",
  "execshell": "Python3",
  "max_number_of_workers": 1,
  "min_number_of_workers": 1,
  "max_memory_limit": "",
  "max_cpu_limit": "",
  ...
  "parameters": [
    {
      "name": "prefix",
      "type": "str",
      "optional": false,
      "flag": "--prefix",
      "action": "store",
      "help": "prefix for file names",
      "default": null
    },...]
}
```

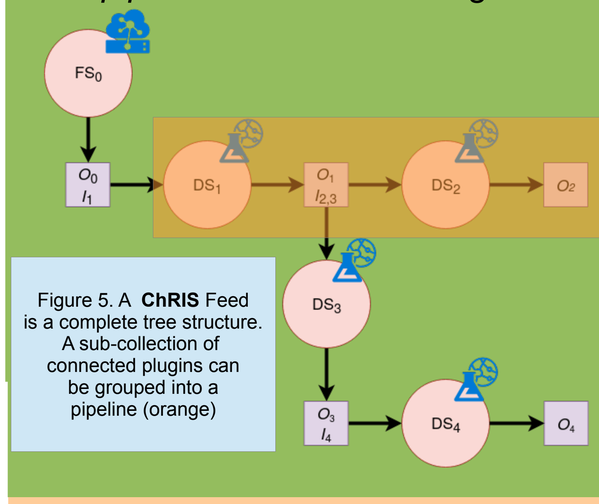



Figure 5. A **ChRIS** Feed is a complete tree structure. A sub-collection of connected plugins can be grouped into a pipeline (orange)

architecture

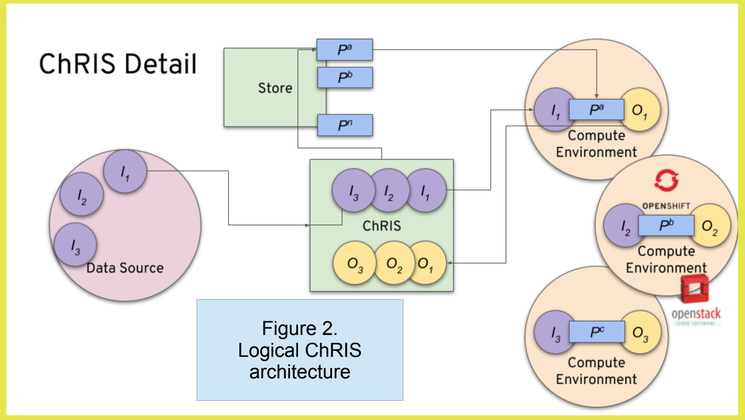


Figure 2. Logical ChRIS architecture

The core of the system consists of the main **ChRIS** backend – **CUBE**, or the **ChRIS U**nderlying **B**ack **E**nd – and the `coordinator`, running typically on the same system. On remote systems are two services: the `datahandler` for data transfers; and for job exec, the `computehandler`.

CUBE sends instructions to the `coordinator` on data and compute. In turn, the coordinator communicates with a remote `datahandler`, sending it the data to be processed, and asks the `computehandler` to schedule some defined analysis on this data. The `coordinator` then, when asked by **CUBE**, checks on the status of the remote analysis, and returns and registers results

Internally, on the MOC, plugins are contained within a Job context, initialized by an **Init Container** that presents a file system analog to the plugin. On completion, a **Publish Container** handles the details of storing results in Swift storage – see Figure 6.

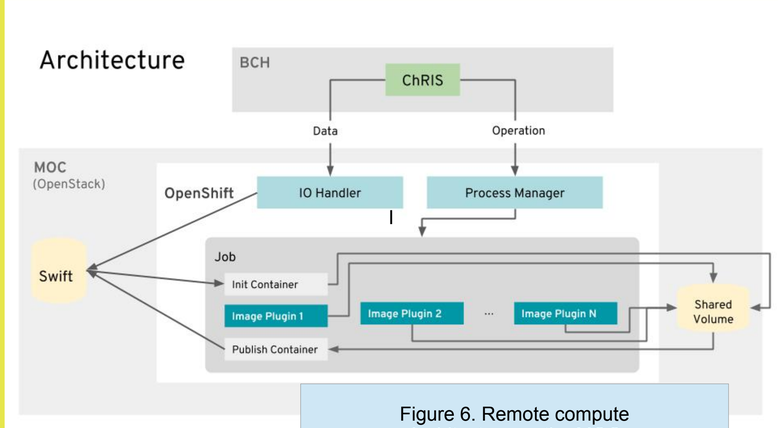


Figure 6. Remote compute Architecture on the MOC

u i / x

Visually,the ChRIS UI presents a tree-graph overview of processing nodes (each node is a plugin), and the edges connecting them represent data (information flows from the output directory of one plugin node to the input directory of the next connect downstream node) – see Figure 7.

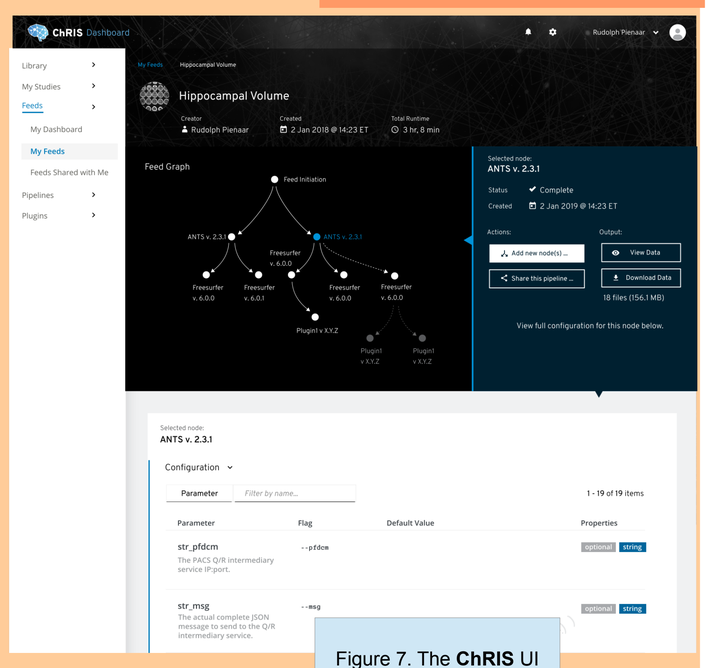


Figure 7. The **ChRIS** UI

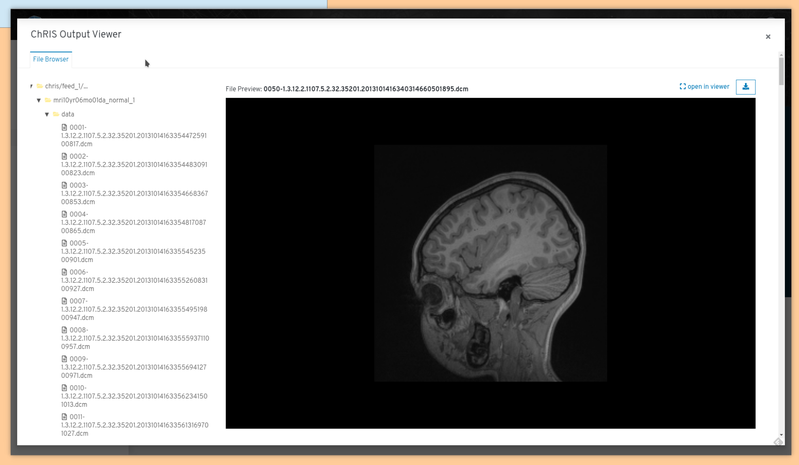


Figure 8. Visualizing pulled and processed data in the default and specialized viewers

Complex visualization per-node output is also handled by the UI. A baseline viewing experience is available and more complex / stand alone viewers can also be added to the system.

The UI dashboard provides an intuitive mechanism to add new nodes (plugins) to a processing flow, and also generate/define a subset of the tree as a re-usable pipeline. For a given node, the actual flags that are contextual to its behaviour are accessible and can be set by the user.

1) Jorge L. Bernal-Rusiel, Nicolas Rannou, Randy L. Gollub, Steve Pieper, Shawn Murphy, Richard Robertson, Patricia E. Grant, and Rudolph Pienaar. Reusable client-side javascript modules for immersive web-based real-time collaborative neuroimage visualization. Frontiers in Neuroinformatics, 11:32–40, 2017
2) Daniel Haehn, Nicolas Rannou, Banu Ahtam, Ellen Grant, and Rudolph Pienaar. Neuroimaging in the browser using the x toolkit. In Frontiers in Neuroinformatics Conference Abstract: 5th INCF Congress of Neuroinformatics (Munich), 2014.
3) Rudolph Pienaar, Jorge Bernal, Nicolas Rannou, P.E. Grant, Daniel Haehn, Ata Turk, and Orran Krieger. Architecting and Building the Future of Healthcare Informatics: Cloud, Containers, Big Data and CHIPS. In Proceedings of the Future Technologies Conference, November, Vancouver, Canada, 2017.
4) N. Rannou, J.L. Bernal-Ruisel, D. Haehn, P. E. Grant, and R. Pienaar. Medical Imaging in the Web Browser with the A* Medical Imaging (AMI) toolkit. In Proc. of European Society of Magnetic Resonance in Medicine and Biology, 2017.

conclusion

This work presents a cloud-based solution designed to address the needs of data accessibility, and portable computing. **ChRIS** allows for pervasive anonymization, sharing of data, and powerful remote analytics. The system is designed using a micro-services model with pervasive containerization and provides a strong platform for future informatics.

rudolph.pienaar@childrens.harvard.edu