A Brief History of High Level Applications at TRIUMF

C. Barquest

March 16, 2019

Abstract

In this document, I recollect–to the best of my knowledge–the history of so-called "High Level Applications" at TRIUMF. I have attempted to motivate the need for control room web applications, summarize our previous efforts at deploying application servers, outline decisions which lead us to where we are today, and sketch out considerations we have for future improvements. This documentation does not rise to the level necessary to maintain each implementation detailed here, but instead is meant as an informal introduction and description of our current, former, and hopefully future, operations.

Contents

Acknowledgements

There are many people I wish to recognize whose contributions to this project form the foundation which made this all possible from the start. I am grateful to each and every one of them for the roles they have played in taking High Level Applications from an ambitious concept to an achievable reality. We have seen this project through many iterations, and navigated the interactions between the many independent groups that have come together to realize our ultimate goal.

I want to begin by thanking Thomas Planche and Paul Jung. Thomas' vision has been a driving force since the very beginning of this project, and his dedication and spirit truly inspired me. He has been my biggest support, and challenged me when needed. This has helped me grow and I want to thank him for being truly a great teammate. In every sense he is the reason the project is where it is today. I am also grateful for Paul's contributions in driving forward the project in each new step it takes. His focus and attention to detail have created a better organization and overall structure for the project and its applications. I will always cherish the times when Thomas, Paul, and myself would hold whiteboard sessions to brainstorm. I am so grateful for our teamwork.

I would also like to acknowledge Evgeniy Tikhomolv who dedicated much effort into this project. Although it evolved beyond the setup he originally built, his work was crucial in helping us establish requirements and bringing the project to maturity. My work would not have been possible without his initial contributions. I'd also like to thank the leadership and supervision of Friedhelm Ames and Oliver Kester for supporting this endeavor while giving us the freedom to branch out in new and exciting ways. Their open-mindedness and trust in our vision has let us pursue cutting-edge technology and aim this project towards truly the best version we could imagine.

In this pursuit, I owe a great debt to Spencer Kiy and Olivier Shelbaya. They both have helped drive this project from the beginning of the HLA Task Force and have carried on its leadership since my departure. Spencer has quickly adapted to the demands of system administration on the back-end, and has methodically organized requirements and coordinated work. Olivier has readily taken on the responsibilities of leading meetings and interfacing with front-end application needs. Together, their enthusiasm, teamwork, and leadership excite me for what's to come with this team's bright future. I am forever grateful for their work and dedication.

As part of this team, I'd like to thank Stephanie Rädel, who contributed to the eLINAC portion of the `acc` database, as well as designed our logos and made them look as polished and professional as they do. I will always be thankful for her enthusiasm and energy. I'd also like to thank Brad Schultz for his vision in bringing HLAs to the CANREB project at TRIUMF. I would like to sincerely thank the controls team, and especially Keiko Ezawa, Kyle LeBlanc, and Juan Pon for their work. Keiko always had time for me when I had questions; Kyle helped teach me the fundamentals of `PyEpics`; and Juan has helped give insight on the legacy cyclotron controls and how we can connect with them in the future.

I would especially like to thank Kel Raywood and Dan Thompson for their immense support as computing experts. Kel is the reason we were able to adapt our servers into the virtual machine framework in the first place, and his time and patience with me is probably the single most significant contribution to this project's success that I can name. I am forever grateful for his help. In addition, Dan has become an integral part to this project's success, in setting up Kubernetes and also with his help in instilling best practices. His advice and help have been exceptional, and it is because of his unique skills that this project's deployment is advancing to where we envisioned it from the beginning. I cannot thank the computing group enough for their support which has been an integral part to our success, and led to many tools being deployed for use site-wide across TRIUMF. I am so grateful that they have enabled this for all to benefit.

Next I'd like to thank the operator members of the HLA Task Force: David Prevost, Eric Chapman, and Kevin Lucow. Kevin especially has taken taken the opportunity to develop applications and I appreciate his willingness to test out new software frameworks and give feedback. David and Eric also provide valuable feedback during meetings, and I appreciate their time dedicated to this project. In addition, there are many operators who have helped give feedback on applications, and I'd especially like to thank Tiffany Angus for her contributions in this regard. It's only when we have expert feedback that our applications can be developed into their best form.

In addition, although they were not involved directly in the HLA project, I would like to thank many members of the Beam Physics group who have supported me throughout my post-doc at TRIUMF. I would like to thank Shane Koscielniak for his helpful advice on leading meetings, and refering me to Robert's Rules of Order which helped me in taking down meeting minutes. I'd like to thank Fred Jones for conversations on coding, hardware, and the evolution of technology at TRIUMF and for inspiring me to pursue a career where I could become a techincal expert as well. I'd also like to thank Yuri Bylinski, Dobrin Kaltchev, Yi-Nong Rao, Suresh Saminathan, and Marco Marchetto, who have provided valuable feedback and inspiration as well.

I also want to acknowledge the co-op students whose work on high-level applications with the Beam Physics group helped drive our concept forward: Samantha Marcano, David Tattan, Dan Sehayek, Vishvam Mazumdar, Margaret Corwin and Megan Stewart. Their work has been invaluable, and I cherish the time I have been able to spend as a mentor and supervisor. I appreciate the opportunity to be able to grow in my career at the same time helping these students experience the possibility of pursuing a career in science and technology. Their engagement has meant a lot to me, and I am thankful for all the interactions that I was able to have with them.

Finally I would like to thank my supervisor, Rick Baartman. His leadership of the Beam Physics group as well as his expert knowledge and kind nature have truly inspired me throughout my years at TRIUMF. He supported and encouraged our efforts in developing solutions without enforcing any one specific approach, but rather by providing guidance grounded in fundamental physics concepts and his years of expertise. I am thankful for having the opportunity to be a part of his team, where each member's hard work and dedication is inspired and thrives in part because of him. I will always cherish this time as a member of the Beam Physics group and I hope that each person who gets a chance to interact with this group realizes how fortunate they are.

I myself have personally gained much from this experience. I will always be beholden to my team at TRIUMF who, for me, embody Aristotle's saying that "the whole is greater than the sum of its parts." I hope to be so lucky as to find this synergistic energy elsewhere as I take on other pursuits throughout my career. In all, I hope this summary serves as useful documention of the work achieved on the HLA project at TRIUMF for the time I was involved. I eagerly anticipate the successes of this project to come.

1 Motivation: What We Needed

Many different types of applications are needed in an accelerator control room, from diagnostics tools to beamline models. Specialized applications are often developed as MATLAB GUIs because of the low barrier to entry for creating such programs. However, proprietary-software-based applications can be suboptimal for the simple reason that every computer running the application needs access to a license.

This constraint presents a few issues beyond the obvious upfront cost of maintaining these license subscriptions. For the sake of argument, let's assume it is cost-effective to have a site-wide license available. After all, tools like MATLAB are used in scientific labs for much more than just building control room applications.

Even if every computer in the control room has a license, there is still the issue of keeping each application running with the most up-to-date version of the software. This perpetual cycle of "upgrade-break-update, repeat" can lead to a reluctance to upgrade, or worse yet: multiple version dependencies within the same control room, an IT nightmare.

And even if every application is perfectly maintained to continue working with each software update, these applications are by definition stand-alone applications, each heavily influenced by the developer who has their own personal aesthetic and feeling for how the user-interface should be designed. They also do not benefit from a central database of functions available to them from previous applications developers.

One other drawback to this model of application development is that there are awesome applications developed for specific beamlines and the other beamlines in the lab cannot benefit from this development. If instead the application was initialized from an accelerator database, it could easily be ported over to deploy for multiple beamlines through the lab.

So all of these considerations lead the Beam Physics group to search for a different paradigm that could break the cycle and also provide additional benefits beyond keeping applications up-to-date without needing costly license subscriptions. Here the idea of an application server complete with central accelerator database was devised.

Although this endeavor had its origins in the Beam Physics group, we knew it could not be done alone. For these applications to be a success, it would involve coordinating effort across many groups: Beam Physics, Beam Delivery, Operations, Controls, and Computing. To this end, a High Level Applications Task Force was established with members from each group. The following section is a retrospective on the Beam Physics group's servers that were deployed before the creation of the High Level Applications Task Force. This history proved the concept and layed the groundwork for the central effort that followed.

2 Server History: Where We've Been

This section highlights the evolution of the hardware and software through each implementation, as well as the maintainer(s) of each setup. We started with a single desktop computer maintained as a side project by a Beam Physics group member. When this was deemed unsustainable, resources were invested into three new servers to be maintained by a dedicated Beam Physics group member. Although this was a step in the right direction, there were shortcomings to this approach as well. So finally, a solution was realized when we approached the Computing group and determined that they would be able to provide us with virtual machines and logical volumes running on their existing server cluster, a maintainable and expandable solution suitable for future growth.

2.1 The Original `beamphys` server

Our first web server incarnation, `beamphys`, runs on a salvaged desktop located in Room 105 (See Fig. 1). The operating system is some flavor of linux (CentOS?) running Apache for its web server. It provided a central location for Beam Physics group members to ssh into where they could share files from their home folders, controlling access through standard `.htaccess` files.

Figure 1: The original `beamphys` server: a Dell desktop computer located in Room 105.

Because this server was deployed by a group member with an already full-time work schedule to maintain, it was conceptualized as a bare-bones web-server with relatively meager security measures. Although logs were checked semi-regularly, it is accessible from off-site, does not require ssh-keys for login, and allows logins from all members of the Beam Physics group. In summary: it was an excellant platform to begin playing around with, but was probably never intended as a long-term solution. In order to determine requirements though, sometimes the best approach is to dive right in, which is where this server really shined.

I believe it was on this computer that the first study[1] by the Beam Physics group into control room web applications at TRIUMF began in 2015 with Tune Display and Beam Envelopes, "implementations of tools already being used" (See TRI-BN-15-13). This study proved that web applications could be a viable approach for control room applications, but that it would need more dedicated resources. For example, a developer testing out updates would have to take down the actively running application because there was only one deployed instance.

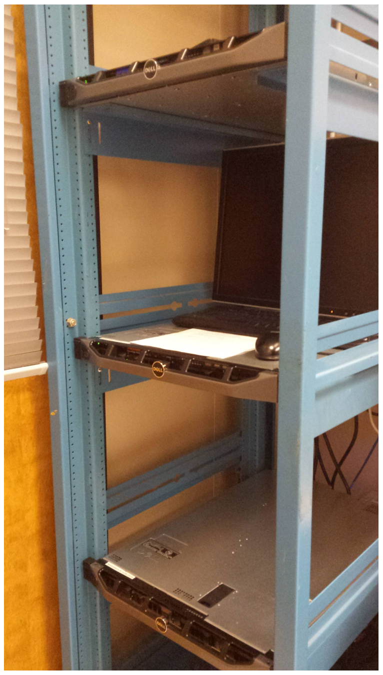
So while this setup worked well for sharing team members' static files, it lacked a standard approach for deploying interactive applications and further yet, a method for communicating with EPICS. It also goes without saying that better solutions exist than to keep critical functions running on hardware in an office space where a cord can accidentally be kicked or the power button accidentally bumped (yes, this has happened multiple times!).

At this point, the functionality of external publication, internal development, and control room production were separated out into the `hlaweb`, `hladevel`, and `hlaprod` servers respectively.

[1] I have since learned that there was actually a previously deployed `devel-xal` server (now shut down) on a machine managed by Controls which supported `XAL` control room applications. This experience taught a valuable lesson: we need to be able to properly articulate our vision, and have something under our own control.

2.2 The `hlaweb`, `hladevel`, and `hlaprod` Servers

The next generation of servers consists of the trio of `hlaweb`, `hladevel`, and `hlaprod` servers.[2] The hardware for these servers can be seen in Fig. 2. An obvious improvement in this setup is that the hardware has better performance specs than a single desktop computer, and also is no longer stored in an office setting. I don't know if these servers have an automatic external backup–this is something to look into–but a system of backing up files was enabled locally, as seen by the external usb drives located on top in Fig. 2b.


(a) Server rack showing all three stand-alone servers.

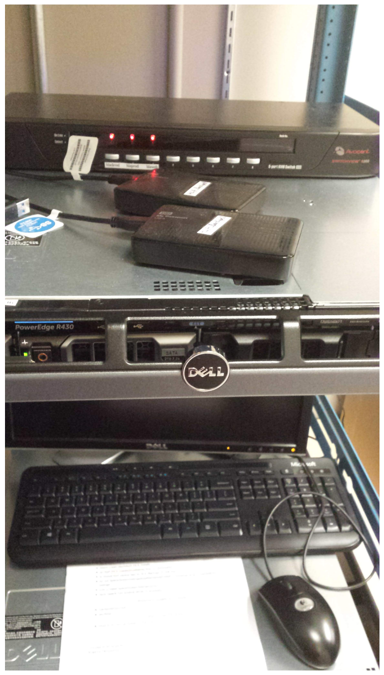

(b) On top, a monitor switch controller.

Figure 2: The `hladevel`, `hlaweb`, and `hlaprod` servers, located outside the cyclotron control room.

Similarly to `beamphys`, these servers are also CentOS-based, and also run Apache. One of the more striking differences with these servers in particular is their directory structure follows a somewhat non-standard approach. I believe these choices were made in order to more easily assign and divide up disk space, however there could be other underlying motivations that I'm not aware of as well. One of the most important locations on these servers were their `projects` directories. But instead of these `projects` directory being located at the top level, for instance: `/projects/`, this location was softlinked to its actual harddrive location within the `space` directory, `/space/usr1/projects/`. Although this allocation may have been adventageous from a system administration perspective, it did tend to lead to some confusion when navigating the file system. Many times while navigating in a terminal, you would find that you could not easily change directories up and down without getting lost because of a softlink.

A great advantage of this setup, however, was the concept of having three separate servers, which greatly increased its functionality over the original `beamphys` server setup. Each of the three servers were meant to be copies of each other–for easy system maintenance–with only small differences between them. For instance, I think there was a difference in the EPICS installation across the servers(?), but in general the directory structure was mirrored across each of the servers.

[2]From now on, I will denote these as `hla*` if I need to refer to all three of these servers as a group.

The `hladevel` server acted as a web server, code repository, and multi-user login/workspace. The `hlaweb` server is the public-facing duplicate server, along with a deployment of JupyterHub. The `hlaprod` server is the private or internal duplicate server, meant for production use inside the control rooms. Applications were developed and deployed on these servers, exclusively within what is called the `projects` directory.

The `projects` directory, as mentioned earlier, is a special directory on the `hla*` servers, being the one location where applications were allowed to be contained and deployed to the web server. I expect that this setup was meant to simplify the web server hosting, and meant also as a security measure as well. Inside the `projects` directory were application directories, one directory per application. This approach worked very well for stand-alone applications, but began to pose problems once our applications began growing, becoming connected, and requiring more functionality. This will become evident in the following discussion of deploying applications on these servers.

2.2.1 Deploying Applications

Developers would work on an application inside the `projects` directory on the `hladevel` server. These applications were originally static HTML (utilizing either PERL or Python libraries?). For more flexibility in reusing code, such as html templates, and additional functionality such as dynamic URLs, we began moving towards python-based Flask applications. For these apps, a static `wsgi` file is located at the top level of the project's directory folder.

This project folder was not version-controlled with `git`, although a `GitList` viewer was deployed for a separate `repo` location that contained the project's code as well. This was not extensively used as I recall, as code had to be manually updated in this location.[3] A major debate took place over whether or not the `projects` directory should actually be the code repository. Since developers are able to edit files directly within the `projects` directory, this would often happen and then differences would occur between the code repository and the `projects` directory, and need to be resolved (see Fig. 3)[4].

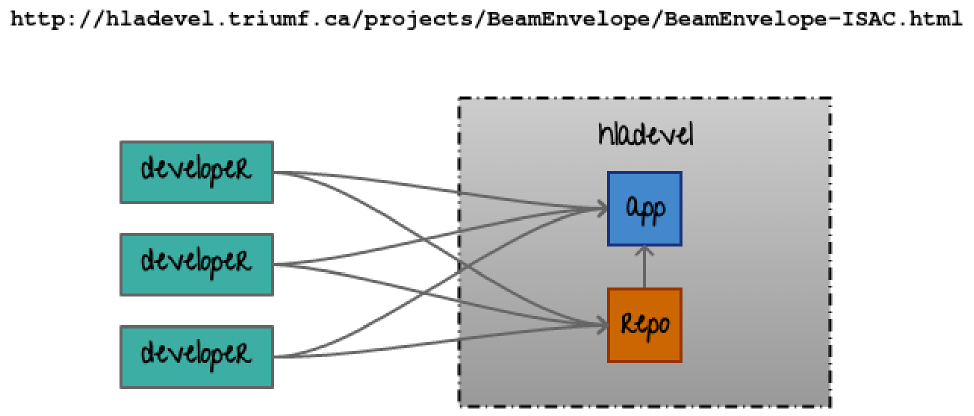

http://hladevel.triumf.ca/projects/BeamEnvelope/BeamEnvelope-ISAC.html
developer
developer
developer
hladevel
app
Repo

Figure 3: Manual application deploy to `hladevel`, with an example URL. Note that development work could take place directly in the app's project directory–from which it deployed–while its repository was stored in a different location. This lead to often out-of-sync versions.

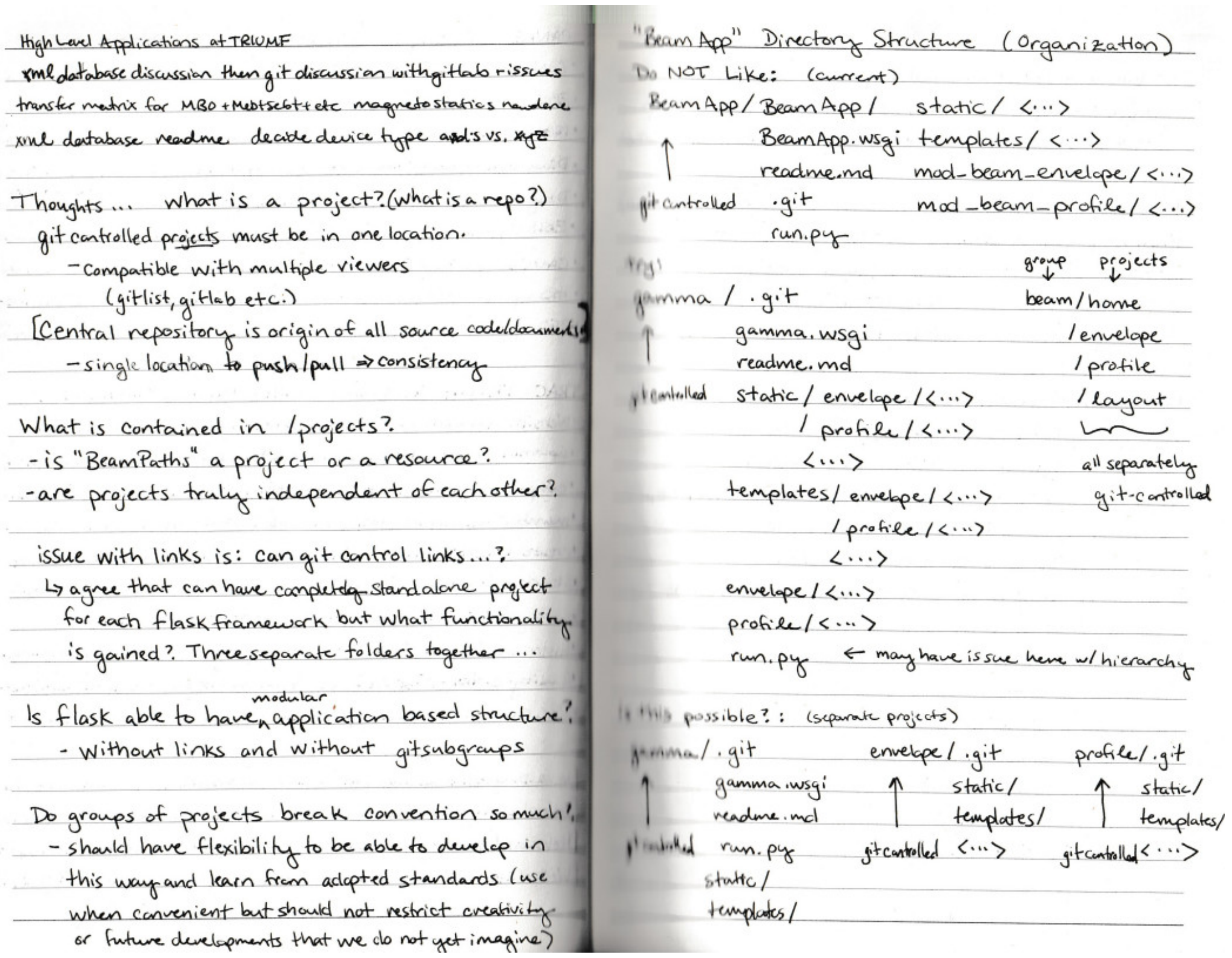
[3]A more advanced git-based repository was desired, such as GitLab, however this was never able to be successfully deployed on any of the `hla*` servers (tools such as GitLab are designed to be deployed independently in any case).

[4]In flowchart diagrams throughout this paper, I will denote web servers with a dot-dash outline.

It was actually quite useful to be able to directly edit files within the `projects` directory, because this is how quick debugging could take place. When the app's `wsgi` file is touched, it restarts the application, so the developer can view changes live on the `hladevel` server.[5]

When an application was ready for use in the control room, it could be copied over to the `projects` directory on the `hlaprod` server. The `hlaweb` server was used primarily for group members to publish externally-facing documentation or files.

One major issue encountered was with the strict structure of the `projects` directory which was required for application deployment (I admit I cannot remember why this restriction was in place–I believe it wsa something required from the web server implementation itself, but I'm not sure). This restriction meant that every application wound up having two folders with its project name: the project folder itself (which is the top-level, git-controlled directory) and a subfolder of the same name, which actually held the contents of the application beyond the `app.wsgi` and `run.py` files. See the "BeamApp" folders in Fig. 4 for an example of what I'm talking about here.



High Level Applications at TRIUMF
xml database discussion then git discussion with gitlab + issues
transfer matrix for MBO + Mebtsebt + etc magnetostatics nauderer
xml database readme decide device type and's vs. ~~xyz~~

Thoughts ... what is a project? (what is a repo?)
git controlled projects must be in one location.
- Compatible with multiple viewers (gitlist, gitlab etc.)

[Central repository is origin of all source code/documents]
- single location to push/pull ⇒ consistency

What is contained in /projects?
- is "BeamPaths" a project or a resource?
- are projects truly independent of each other?

issue with links is: can git control links ...?
↳ agree that can have completely standalone project for each flask framework but what functionality is gained? Three separate folders together ...

Is flask able to have modular application based structure?
- without links and without gitsubgroups

Do groups of projects break convention so much?
- should have flexibility to be able to develop in this way and learn from adopted standards (use when convenient but should not restrict creativity or future developments that we do not yet imagine)

"Beam App" Directory Structure (Organization)

```
Do NOT Like: (current)
 BeamApp/ BeamApp/        static/ <...>
          BeamApp.wsgi    templates/ <...>
    ↑     readme.md       mod-beam-envelope/ <...>
git controlled .git       mod_beam_profile/ <...>
          run.py

try!                                   group   projects
                                         ↓       ↓
gamma / .git                           beam/home
   ↑     gamma.wsgi                           /envelope
   ↑     readme.md                            /profile
git controlled static/ envelope/<...>         /layout
                     / profile/<...>          └────
                     <...>                    all separately
         templates/ envelope/<...>             git-controlled
                    / profile/<...>
                    <...>
         envelope/<...>
         profile/<...>
         run.py   ← may have issue here w/ hierarchy

is this possible?: (separate projects)
gamma/ .git            envelope/ .git         profile/ .git
  ↑    gamma.wsgi        ↑     static/          ↑    static/
  ↑    readme.md         ↑     templates/       ↑    templates/
git controlled run.py  gitcontrolled <...>    gitcontrolled <...>
       static/
       templates/
```

Figure 4: Some notes from brainstorming sessions thinking about how we'd like to restructure the application folders in order to have better version control capabilities.

[5]The astute reader would at this point realize that multiple developers working on the same application simultaneously would lead to conflicts. This did indeed happen, and to try to avoid this, the idea of working in `sandboxes` on the `hladevel` sever was instituted. Each developer could have their own version of an application running and play around with it in a separate dedicated area. However, this concept was never fully realized as there were some ongoing issues with its implementation.

One difficulty with this structure is that any applications that want to share resources, such as the same templates, or the same beam path descriptions, end up having to be inside the same project folder. This resulted in different applications, such as Beam Envelope and Beam Profile, being version-controlled together (again, see Fig. 4 where `mod_beam_envelope` and `mod_beam_profile` are within the same git project). It was desired for each of these projects to be version-controlled separately, but with some way of linking them, such as putting them in a group, so that it was clear that these projects use the same resources, but at the end of the day are distinct, individual projects. This flexibility was not possible with the `hla*` servers, which was one of the driving motivators for looking towards a new system.

2.2.2 Communicating with EPICS

I'm not sure exactly how communication with EPICS works on the `hla*` servers. From what I gather, a gateway was set up for the `hladevel` and `hlaprod` servers to be able to access EPICS values on at least one of the controls subnets, ISAC. I think that `hlaweb` was meant for public-facing projects and documentation, so did not have access to the EPICS values. This section will definitely need to be fleshed out, as this was the first time a gateway established by the controls group was accessible by an externally controlled server at TRIUMF, and is a significant milestone in achieving fully interactive web applications for the control rooms.

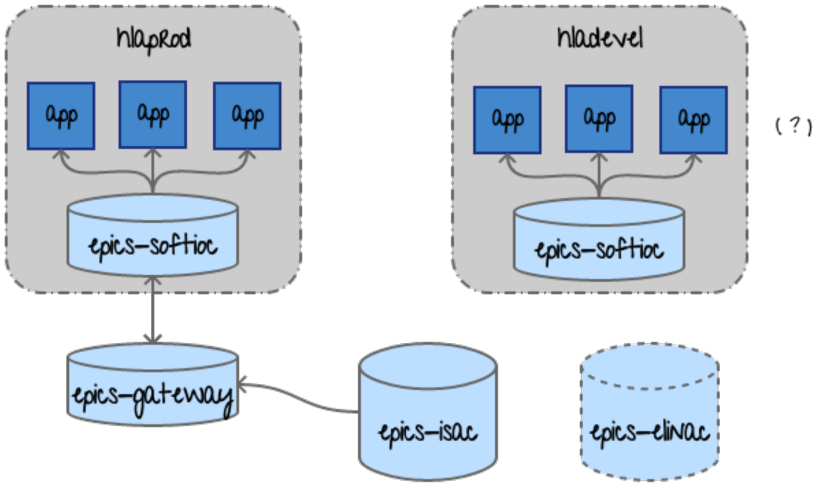


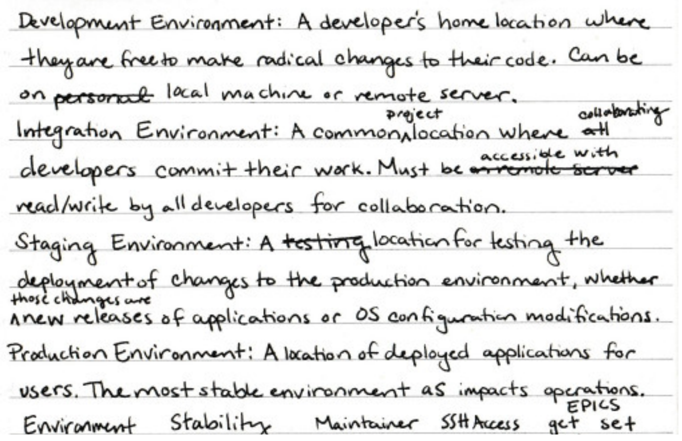
Figure 5: Communication with epics on `hla*`, fuzzy recollection.

I've at least attempted to reconstruct a schematic in Fig. 5. A soft-ioc was running on the server itself, allowing for direct commands such as `caget`, `caput` etc. which require EPICS libraries. A gateway was deployed by the controls group, external to the `hla*` servers. This gateway interfaced with the soft-ioc which mirrored live values. Someway these channels were monitored, in order to enact changes to the real EPICS subnet, but I don't know the details of how this works.

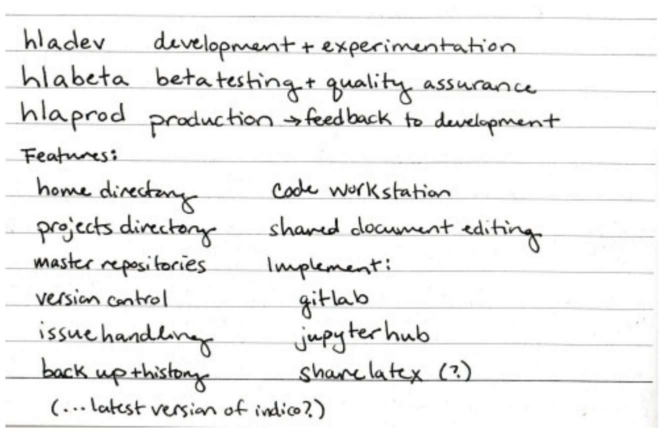
Suffice it to say, although this is a very sophisticated setup, and has been done professionally from a controls point-of-view, it still lacks one key element: the ability to get or set PVs from multiple EPICS subnets through a single application. This single aim, to be able to monitor ISAC and eLINAC values simultaneously for instance, would be the motivating factor behind the design and framework of the next generation of HLA-EPICS communication.

3 Current Servers: Where We Are

Figure 6a shows some notes from a brainstorming session where we were bouncing around ideas about what different environments we needed in order to properly develop and deploy control room applications. Although the `hla*` servers got us nearly there, there was still room for improvement. For example, by the definitions put forth in the brainstorming session, `hladevel` was serving the purpose of a Development, Integration, and Staging Environment. When a developer wants to make radical changes on `hladevel`, they may feel constrained because this is the same location where other collaborators commit their work–an "Integration Environment". It is also the only location where testing is performed for how the app deploys to the server–a "Staging Environment".



Development Environment: A developer's home location where they are free to make radical changes to their code. Can be on ~~personal~~ local machine or remote server.
Integration Environment: A common project location where ~~all~~ collaborating developers commit their work. Must be ~~a remote server~~ accessible with read/write by all developers for collaboration.
Staging Environment: A ~~testing~~ location for testing the deployment of changes to the production environment, whether those changes are new releases of applications or OS configuration modifications.
Production Environment: A location of deployed applications for users. The most stable environment as impacts operations.
Environment Stability Maintainer SSH Access EPICS get set



hladev	development + experimentation
hlabeta	beta testing + quality assurance
hlaprod	production → feedback to development

Features:

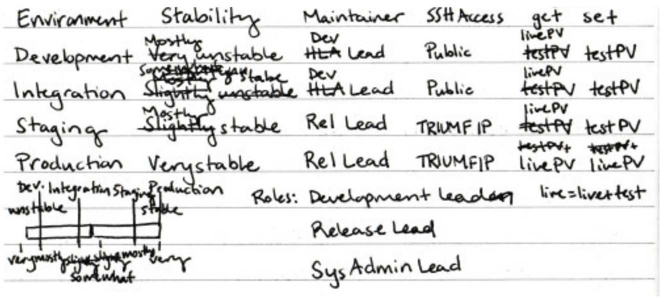
home directory	code workstation
projects directory	shared document editing
master repositories	Implement:
version control	gitlab
issue handling	jupyterhub
back up + history	sharelatex (?)

(... latest version of indico?)

(a) Notes regarding different environment needs. (b) Notes thinking about requirements and tools.

Figure 6: Some notes from brainstorming sessions.

Figure 6b shows a further brainstorming session which refined the environments into three categories: `hladev`, `hlabeta`, and `hlaprod`. The `hladev` server is defined as "development and experimentation", the `hlabeta` server is defined as "beta testing and quality assurance", and the `hlaprod` server is defined as "production with feedback to development". It's important to note here that while we were still considering public facing vs TRIUMF IP restricted SSH access, as seen in Fig. 7, there has been a shift in what we consider the functionality of the separation of servers. Now instead of having one development server and one production server, there's a step in between, showing a growth in maturity of the deployment procedure. Also significantly, there is no longer an `hlaweb` exclusive server for public-facing projects–every project is considered public-facing, and internal restrictions can be accomplished on the server-side as needed (discussed later).



Environment	Stability	Maintainer	SSH Access	get	set
Development	Mostly ~~Very~~ unstable	Dev ~~HLA~~ Lead	Public	livePV ~~testPV~~	testPV
Integration	Somewhat ~~Mostly~~ ~~Slightly~~ stable ~~unstable~~	Dev ~~HLA~~ Lead	Public	livePV ~~testPV~~	testPV
Staging	Mostly ~~Slightly~~ stable	Rel Lead	TRIUMF IP	livePV ~~testPV~~	test PV
Production	Very stable	Rel Lead	TRIUMF IP	~~testPV+~~ livePV	~~testPV+~~ livePV

Dev. Integration Staging Production
unstable stable
very mostly slightly slightly mostly very
somewhat

Roles: Development Leader
Release Lead
SysAdmin Lead

live=live+test

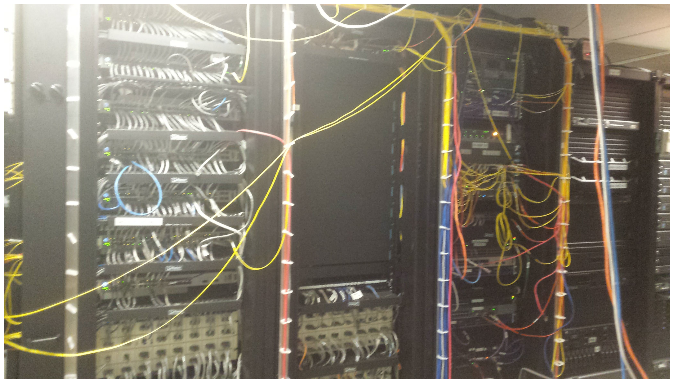
Figure 7: Continuation of notes from brainstorming sessions on different environments.

As also seen in the notes from Fig. 6b, there were many features that we were looking to improve upon, not only for deployment of control room applications, but also for general group tasks such as document editing. The lines on "home directory", "project directory", and "master repositories" speak to frustrations experienced with establishing these central working locations as I've touched upon earlier. Version control was implemented later on in the `hla*` servers, and this line here was meant to indicate that it needs to be at the very center of our work, the use of which drives our development, not a feature that is added on a bit as an after-thought.[6] Alongside version control, a greatly desired feature was that of "issue handling", which can be achieved with repository tools such as GitHub or GitLab. The reason issue handling is so important is that it ties in the feedback from users directly to the application developers, and streamlines communication between developers when bugs are found or new features are requested. Having issues or tickets gives a place to focus discussions, document work, and archive completed tasks.[7]

We can't have multiple developers working on the same directory structure within a server. Although it was a step up from `beamphys` implementation, this still was occuring on the `hladevel` server. Giving the developers write access to the project directory turned out to be not such a great idea. A quick fix would be implemented just to make it work for the moment, but that change would not be captured in the `git` history because it was just "temporary". This lead to different versions and a reluctance to commit changes. Forcing all changes to flow through version control, and deploy automatically with these commits, was the solution to this problem, and is seen in the latest incarnation of the `hla` server.

3.1 hla, devel.hla, staging.hla, beta.hla

The new `hla` server still runs CentOS, but is distinct from the `beamphys` and `hla*` servers in two major ways: (1) it is in fact a virtual server running on a server cluster, and (2) it uses `nginx` + `uWSGI` as its webserver instead of Apache. Pictures of the computer server room can be seen in Fig. 8, with closeups of the blade servers shown in Fig. 9.


(a) Left-hand side of computing server room.

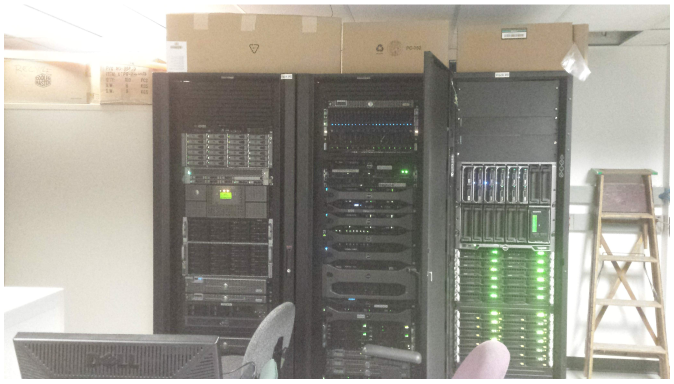

(b) Right-hand side view of computing server room.

Figure 8: Computing server room, located outside the ISAC control room.

[6]Versioning of applications was implemented manually, by copying each new application version to a differently named folder within the application's folder, for instance `R1.2`, and then pointing a link for the production application towards that folder. This works in principle, but was not being used as it was cumbersome to implement.

[7]A further step we could take would be to do Merge Requests, which we started somewhat, but is a more advanced development team process. It may be a good habit to adopt in the future, though, as the team matures.

(a) Close-up of pizza-box servers.

(b) Close-up of blade servers.

Figure 9: Virtual servers run on cluster. It's awesome.

Running a virtual server on a cluster/farm(?) has several advantages over stand-alone hardware. The biggest one from the Beam Physics Group's standpoint is arguably that it is maintained by the TRIUMF's Core Computing and Networking (CCN) group, and so frees up resources from our side and puts the system administration into the hands of the experts. A benefit of working alongside the Computing Group is that tools that were previously beyond our reach, such as GitLab, can be deployed site-wide at TRIUMF, and result in a benefit for the entire lab and not just the Beam Physics Group. Even tools that were able to be running on `hladevel`, such as JupyterHub, are greatly beneficial to the rest of the lab when they are deployed site-wide. And furthermore, now instead of having to maintain our own public-facing webserver and handle all the security details that are involved therein, we can deploy our applications securely on the network (see Fig. 10 for a discussion on the different VLANs at TRIUMF).

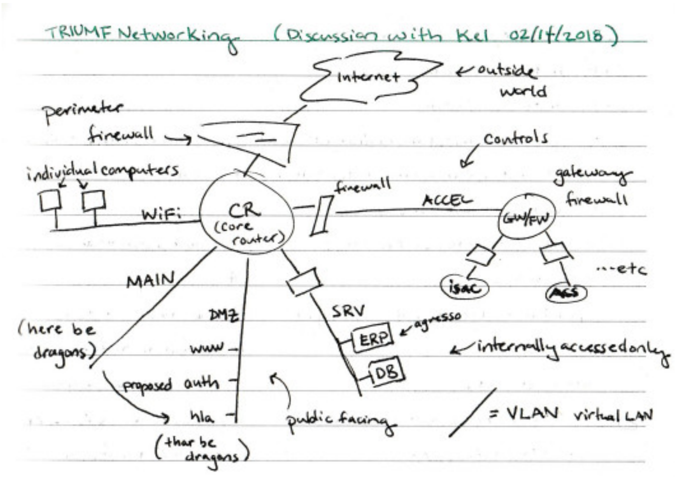


Figure 10: Some notes from a TRIUMF networking discussion.

Beyond the Computing Group being such a valuable resource for us, running with virtual servers has some distinct advantages as well. Space can be allocated dynamically, and when more is needed, a simple `lvextend` command extends the logical volume of the server without having to restart anything. This way we only use the disk space that we need, and can allocate more as our server grows. In terms of server hardware maintenance, this fancy system is even equipped with a "Dell-Phone-Home" feature which will recognize hardware malfunctions, redistribute that server's memory to the surrounding blades, and call up Dell to order a replacement part.

3.1.1 Deploying Applications

One of the most important factors in developing application for this new server was to make sure that application's code repositories (their version history) were in sync with the application's deployment version. We needed to be able to point directly to which version was currently running, and be able to revert changes easily if necessary. It's important to know what changes have been made, who authored those changes, etc.

This was difficult to keep on top of with our previous framework, because the code repository was not a mandatory step in the application deployment process. This meant that when quick changes had to be made (which often happens when things are broken and the application is needed immediately in the control room, for instance) they would be performed directly in the projects directory. Later on we'd often find that it was difficult to establish which version to keep.

An easy solution for this predicament is to tie the deployment of applications to the version control of the applications themselves, making it an essential component of the deployment procedure. With this setup, it just becomes a developer's habit to push changes to the repository, and the versions are immediately in-sync. We accomplish this by configuring `gitlab-runners` on the `hla` server, making it deploy applications automatically from commits to the master branch. This automatically keeps the latest repository version up-to-date with what is currently being deployed to the development server. A schematic of this process is shown in Fig. 11.

`https://devel.hla.triumf.ca/beam/envelope/isac/`

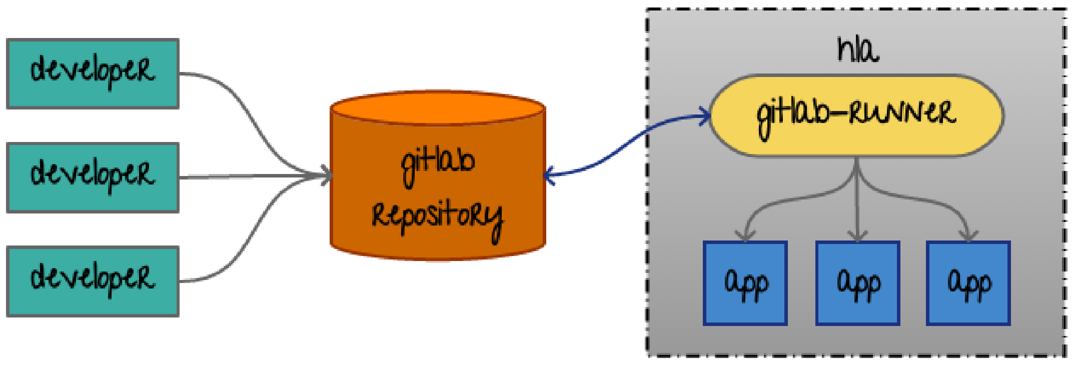
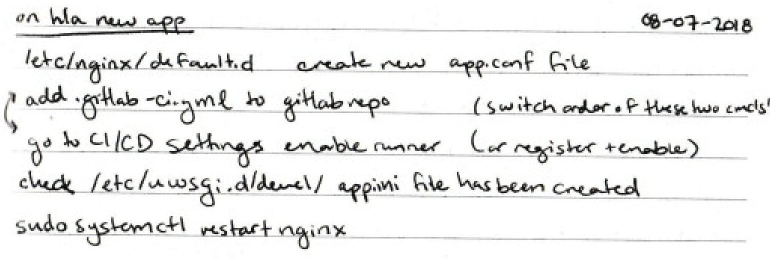


Figure 11: `GitLab` deploy to `hla`. Note that `gitlab-runner` is installed on the `hla` server, allowing it to pull code from the repository through firewalls etc. upon code commits. This enables us to deploy secure servers without having to expose any public access.

To deploy a new application, there are several steps that need to be done first on the `hla` server, outlined in Fig. 12. First, a new `app.conf` file is manually created within the `/etc/nginx/default.d` directory. This lets `nginx` know that a new application is to be hosted. This file maps static file locations, and configures the `uWSGI` mount, passing parameters such as the authorization tokens and configuring the socket for the application running through `uWSGI`.

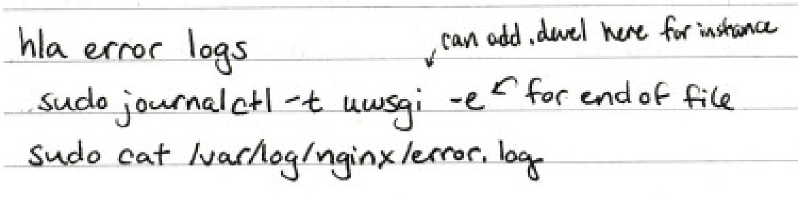


on hla new app 08-07-2018
/etc/nginx/default.d create new app.conf file
add .gitlab-ci.yml to gitlab repo (switch order of these two cmds!)
go to CI/CD settings enable runner (or register + enable)
check /etc/uwsgi.d/devel/ app.ini file has been created
sudo systemctl restart nginx

Figure 12: How to deploy a brand new app on hla.

Next, in the GitLab Continuous Integration setting, a runner is enabled. This `gitlab-runner` can be shared between projects, or a private runner can be registered and enabled if desired. Once a runner is enabled, then a `.gitlab-ci.yml` configuration file can be added to the application's GitLab repo, which tells the runner what to do. The sequence of commands given to this runner, typically a bash-runner, will include a line to create an `app.ini` file within the `/etc/uwsgi.d/devel/` directory, when pushed to the master branch. Check that this file has indeed been created.

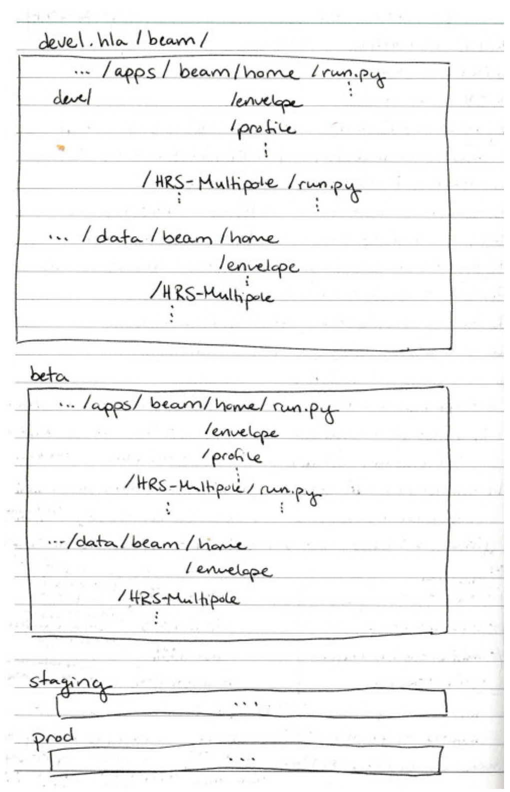
Finally, running the command `sudo systemctl restart nginx` will restart the web server and the application should be deployed to the corresponding web url. Web server error logs can be viewed when logged in with `sudo` privileges on `hla` with the following commands: `sudo journalctl -t uwsgi.devel -e` or `sudo cat /var/log/nginx/error.log`. The `uWSGI` errors may occur when the emperors cannot deploy vassal (for instance, if the application does not run but returns an error message, this is where you will find the output logged). The `nginx` errors might occur because of routing/configuration errors; typically in the past these types of errors have required help from the computing experts to resolve.



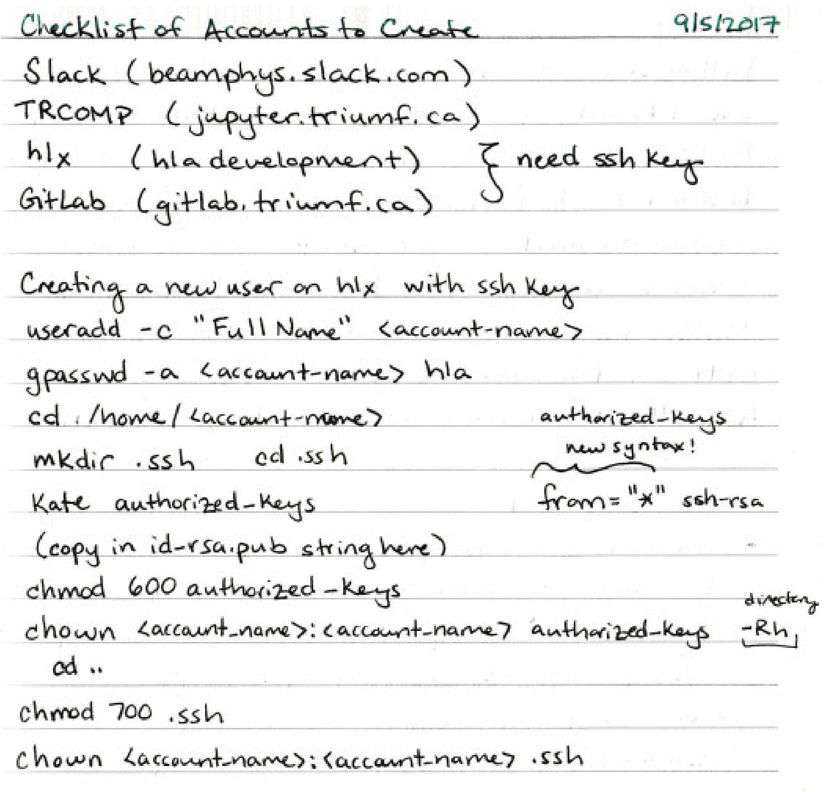
hla error logs
can add .devel here for instance
sudo journalctl -t uwsgi -e for end of file
sudo cat /var/log/nginx/error.log

Figure 13: How to access hla error logs.

The `.gitlab-ci.yml` configuration file determines where the `gitlab-runner` deploys the application files to the server. Each application is deployed within the top-level `/srv` directory, and has an `/srv/apps/` location and a `/srv/data` location for each of the `devel`, `staging`, `beta` and `prod` branches. The original schematic for this is shown in Fig. 14a, where each server was envisioned as a separate virtual machine. By combining them into one machine with separate directories within the `apps` and `data` directories, this saved much overhead, but gives one distinct disadvantage that any time `nginx` is restarted, all of the subdomains are restarted, which is unacceptable for a production deployment. This just means that the production server should be deployed on a separate virtual machine, to decouple the `nginx` web servers completely.



(a) Schematic for thedifferent modes devel/beta/staging/prod on `hla`: note separation of `apps` and `data` directories.



Checklist of Accounts to Create 9/5/2017

Slack (beamphys.slack.com)
TRCOMP (jupyter.triumf.ca)
hlx (hla development) } need ssh Key
GitLab (gitlab.triumf.ca) }

Creating a new user on hlx with ssh Key

```
useradd -c "Full Name" <account-name>
gpasswd -a <account-name> hla
cd /home/<account-name>
mkdir .ssh     cd .ssh
Kate authorized-keys
(copy in id-rsa.pub string here)
chmod 600 authorized-keys
chown <account_name>:<account-name> authorized-keys
  cd ..
chmod 700 .ssh
chown <account_name>:<account_name> .ssh
```

authorized-keys
new syntax!
from="x" ssh-rsa

directory
-Rh

(b) Accounts to be created for HLA developers as of late 2017– note that GitLab's Mattermost has replaced the chat functionality of Slack now, so this account is not longer needed.

Figure 14: Some notes from the `hla` server: directory structure and administrative tasks.

3.1.2 Communicating with EPICS

EPICS communication has a bit more of a complicated set up on the back end of this server, but this is to facilitate easy access to applications, and provide them an API from which they can gain values. It has gone through several iterations, which I will attempt to summarize here.

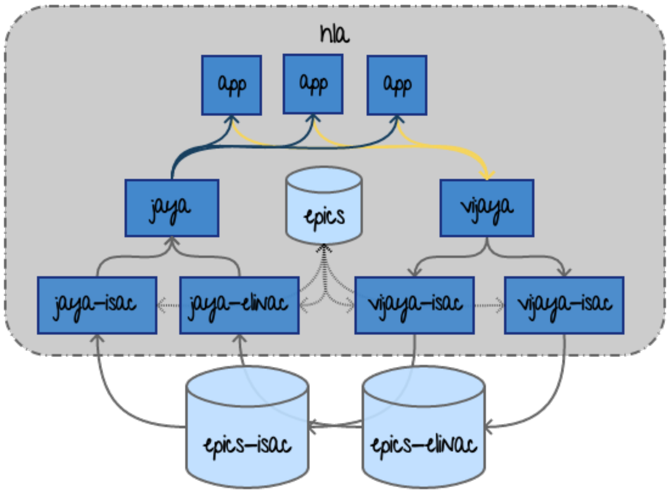


Figure 15: Pyepics on `hla`.

We realized early on that we needed a simple way of getting PV values from EPICS, but did not want to burden the controls network with having each application make individual requests to PVs. See Fig. 16 for a visual representation of this relationship.

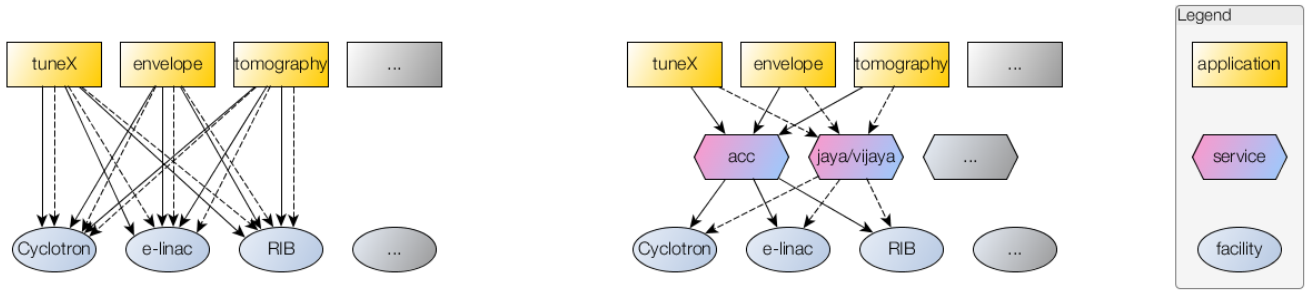


Figure 16: Establishing a service middle layer reduced direct connections between applications and EPICS controls subnets.

In the discussion from December 2017, we brainstormed some names for a pair of gateway services, as seen at the top of Fig 17. Our first instict was the create two gateway services, one for getting values, `Jaya`, and one for setting values, `Vijaya`. As we learned more about the PyEpics interface, these roles evolved into one service for getting monitored values, `Jaya`, and one for getting/setting individual PVs, `Vijaya`. This subtle difference is actually clearly delineated by the PyEpics functions of `camonitor` vs. `caget` and `caput`.

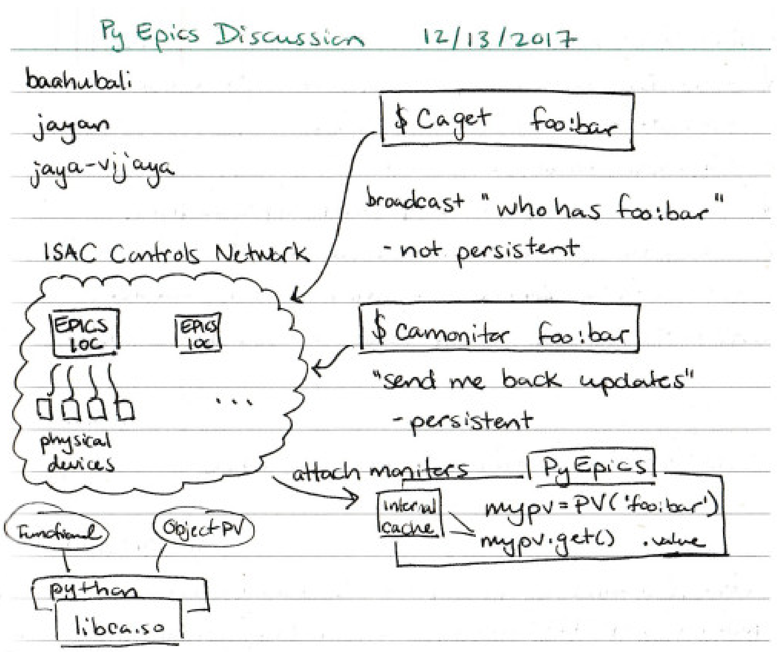


Figure 17: Some notes from our PyEPICS discussion.

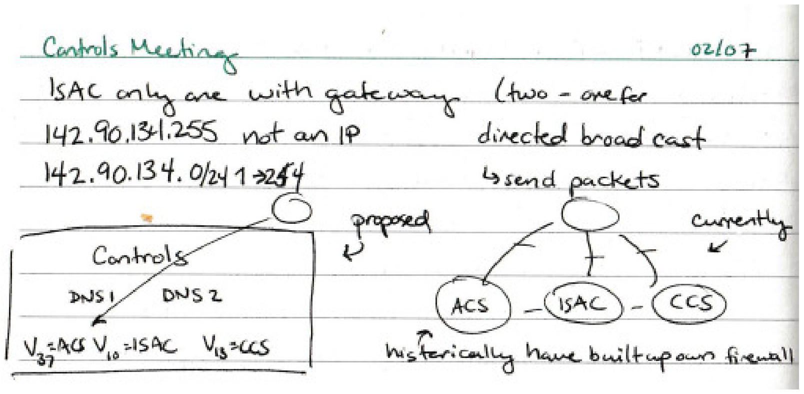
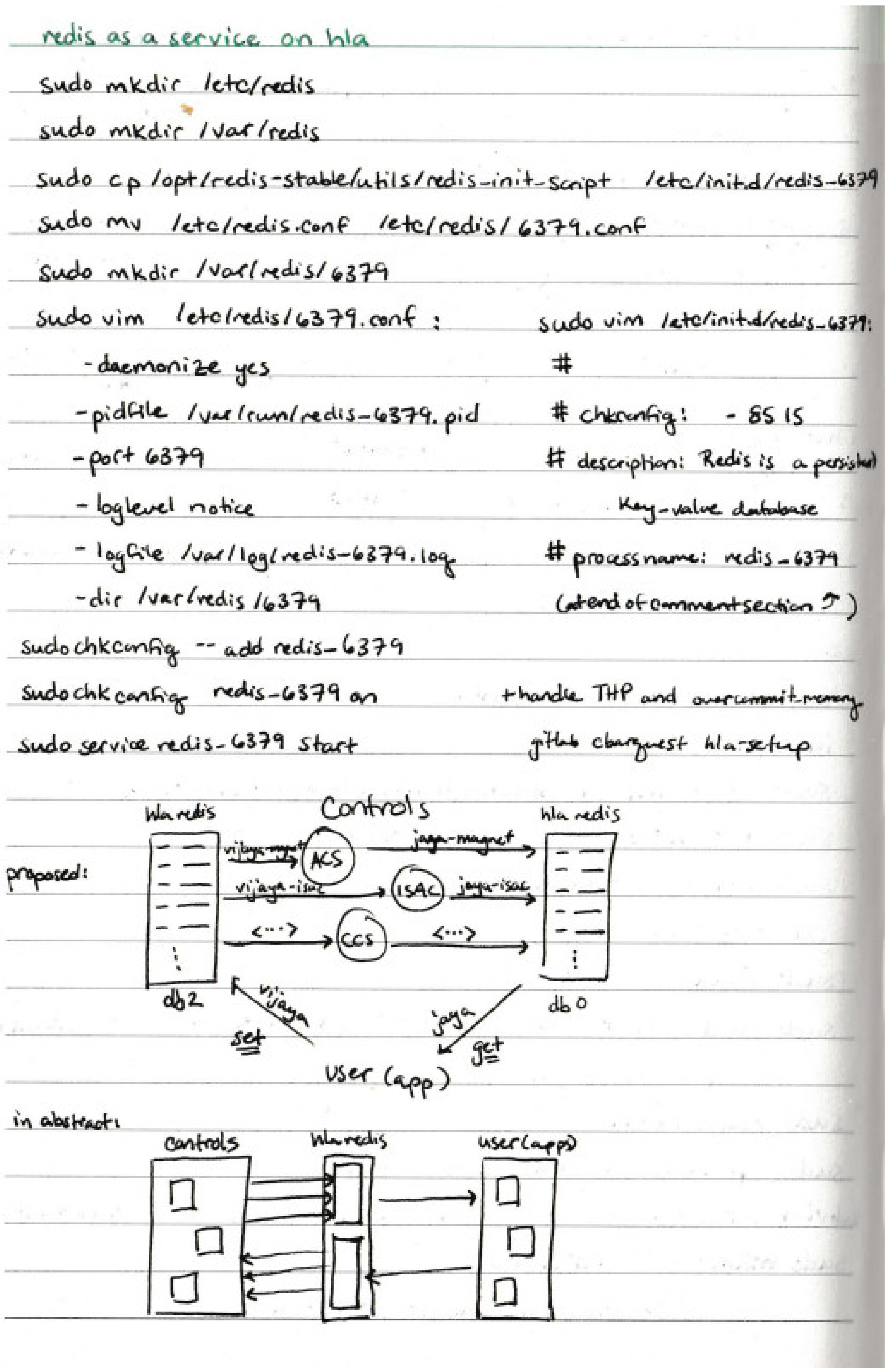

Controls Meeting 02/07
ISAC only one with gateway (two - one for
142.90.134.255 not an IP directed broad cast
142.90.134.0/24 1 → 254
↳ send packets
proposed
currently
Controls
DNS 1 DNS 2
V37 = ACS V10 = ISAC V13 = CCS
ACS
ISAC
CCS
historically have built up own firewall

Figure 18: Some notes from a controls meeting.



redis as a service on hla

```
sudo mkdir /etc/redis
sudo mkdir /var/redis
sudo cp /opt/redis-stable/utils/redis-init-script  /etc/init.d/redis-6379
sudo mv  /etc/redis.conf  /etc/redis/6379.conf
sudo mkdir /var/redis/6379
sudo vim  /etc/redis/6379.conf :
    -daemonize yes
    -pidfile /var/run/redis-6379.pid
    -port 6379
    -loglevel notice
    -logfile /var/log/redis-6379.log
    -dir /var/redis/6379
sudo chkconfig --add redis-6379
sudo chkconfig  redis-6379 on
sudo service redis-6379 start
```

```
sudo vim /etc/init.d/redis-6379:
#
# chkconfig:   - 85 15
# description: Redis is a persisted
      Key-value database
# processname: redis-6379
(at end of comment section ↑)
```

+handle THP and overcommit_memory

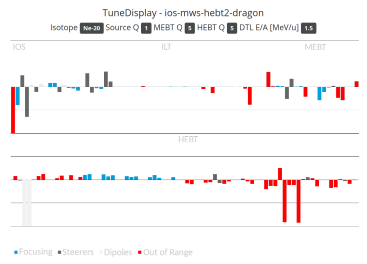
gitlab cbarquest hla-setup

proposed:

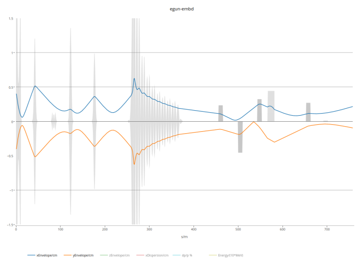
in abstract:

Figure 19: Some notes on redis as a service on hla.

3.2 Application Highlights

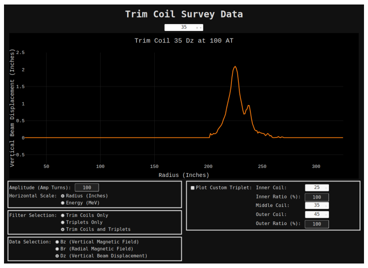

TuneDisplay - ios-mws-hebt2-dragon
Isotope Ne-20 Source Q 1 MEBT Q 5 HEBT Q 5 DTL E/A [MeV/u] 1.5
IOS
ILT
MEBT
HEBT
Focusing
Steerers
Dipoles
Out of Range

(a) TuneX application screenshot.

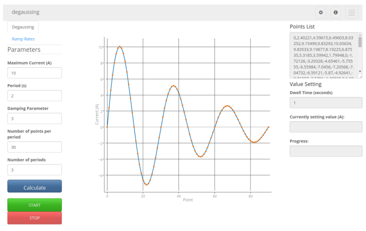


(b) Envelope application screenshot.

Figure 20: Two core HLA applications, TuneX and Envelope.

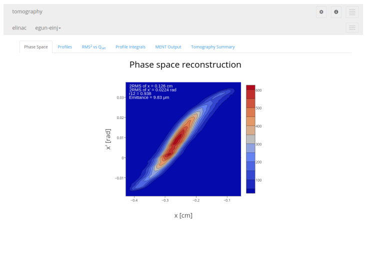

Trim Coil Survey Data
Trim Coil 35 Dz at 100 AT
Vertical Beam Displacement (Inches)
Radius (Inches)
Amplitude (Amp Turns): 100
Horizontal Scale: Radius (Inches) Energy (MeV)
Filter Selection: Trim Coils Only Triplets Only Trim Coils and Triplets
Data Selection: Bz (Vertical Magnetic Field) Br (Radial Magnetic Field) Dz (Vertical Beam Displacement)
Plot Custom Triplet: Inner Coil: Inner Ratio (%): Middle Coil: Outer Coil: Outer Ratio (%):

(a) Trimcoil Binder application screenshot.

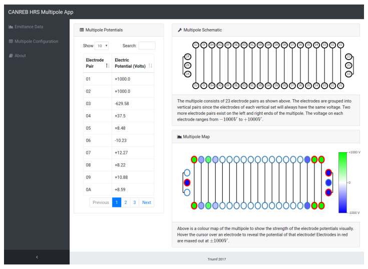


(b) Magnet Degaussing application screenshot.

Figure 21: Examples of specialty HLA applications, Trimcoil Binder and Magnet Degaussing.



(a) Tomography application screenshot.



(b) ATOM application screenshot.

Figure 22: Some examples of student-driven HLA applications, Tomography and ATOM.

3.2.1 beam app

The "beam" app is actually a grouping of applications that share a common core: the `acc` database. We also required HLA apps to have a common "look and feel" which promted us to try to use the same templates across applications, which can be accomplished for instance through using Jinja templates with Flask Blueprints. See Fig. 23 for a schematic of how this application was structured.

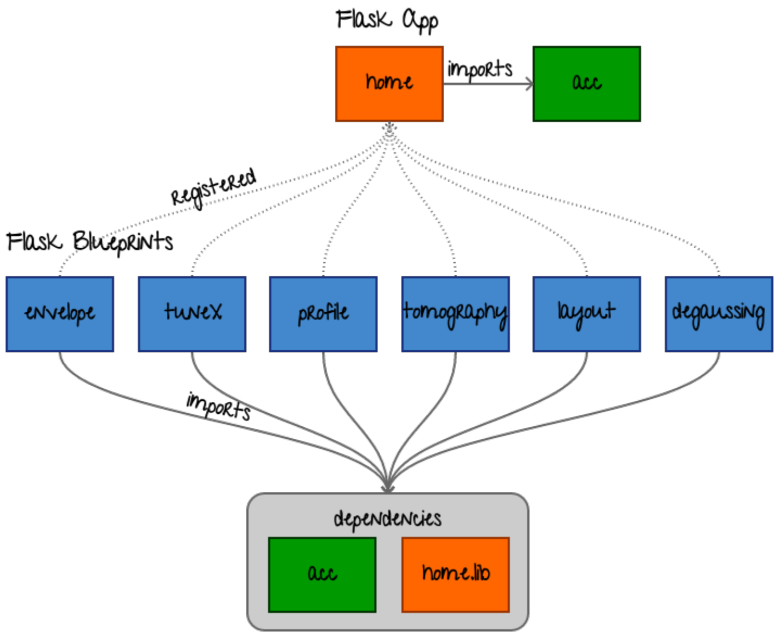


Figure 23: Beam app schematic. Templates are defined in the `home` Flask app.

3.2.2 acc library

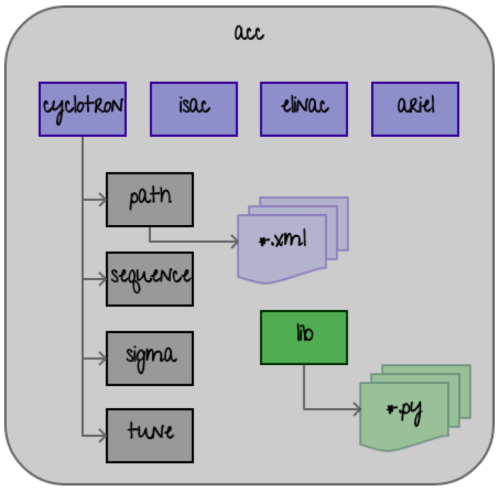


Figure 24: The `acc` database schematic. Database files are in `xml` format, while Python libraries for accessing and manipulating database entries are stored in the `lib` directory.

4 Future Servers: Where We're Going

We need to be able to tie exact versions of dependencies to each other – this can be accomplished with Docker. By tagging released versions, you can pull explicit commits into your code so when for instance the `acc` database gets updated, it will not break

4.1 beam app inverted

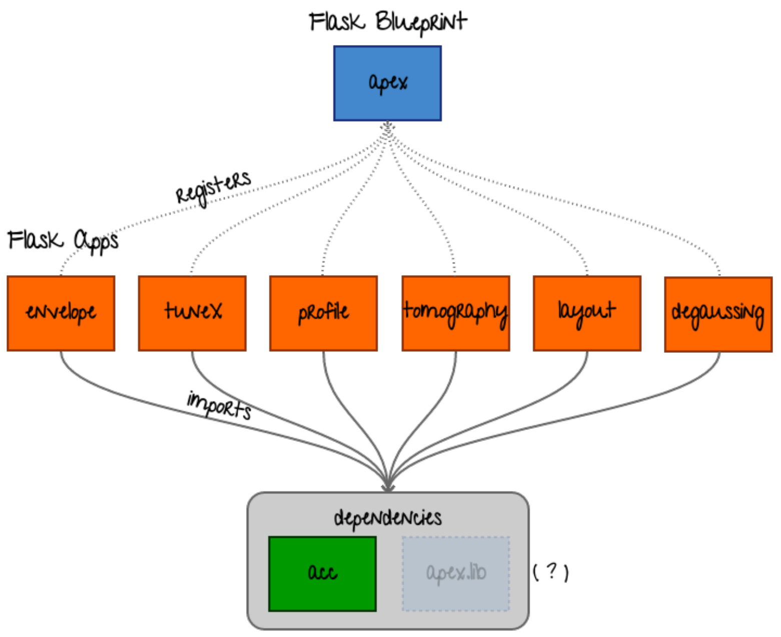


Figure 25: Beam app "inverted" schematic. Templates are now held in the blueprint project, `apex`.

4.2 completely separated servers

4.3 Docker Framework

`https://dev.envelope.triumf.ca/`

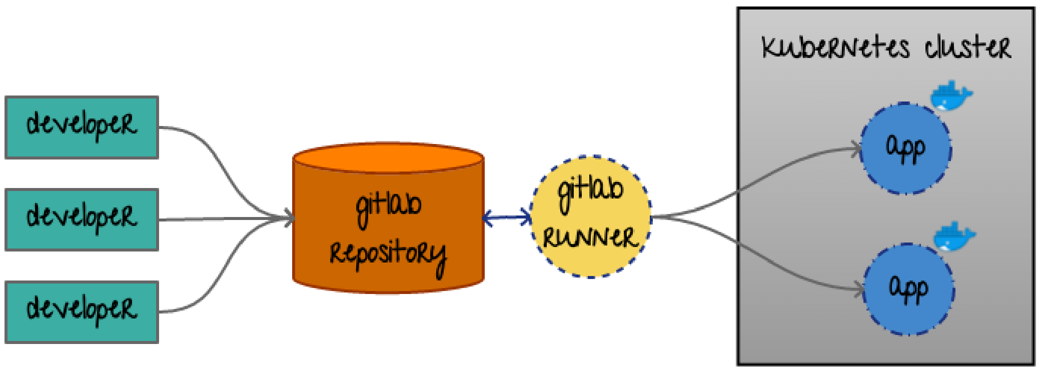


Figure 26: `Kubernetes` deploy to ?.

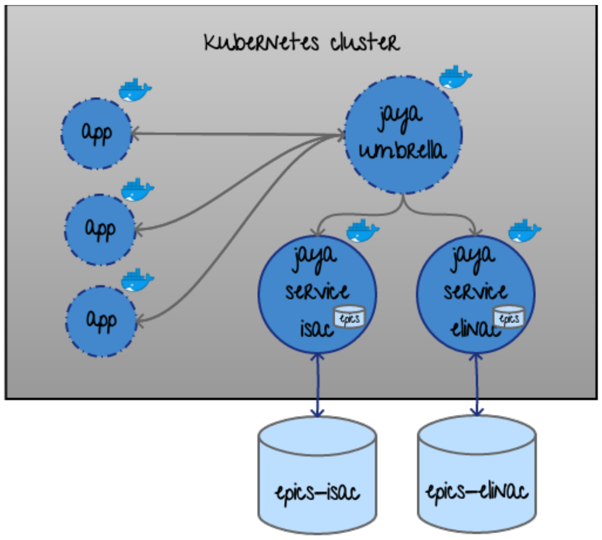


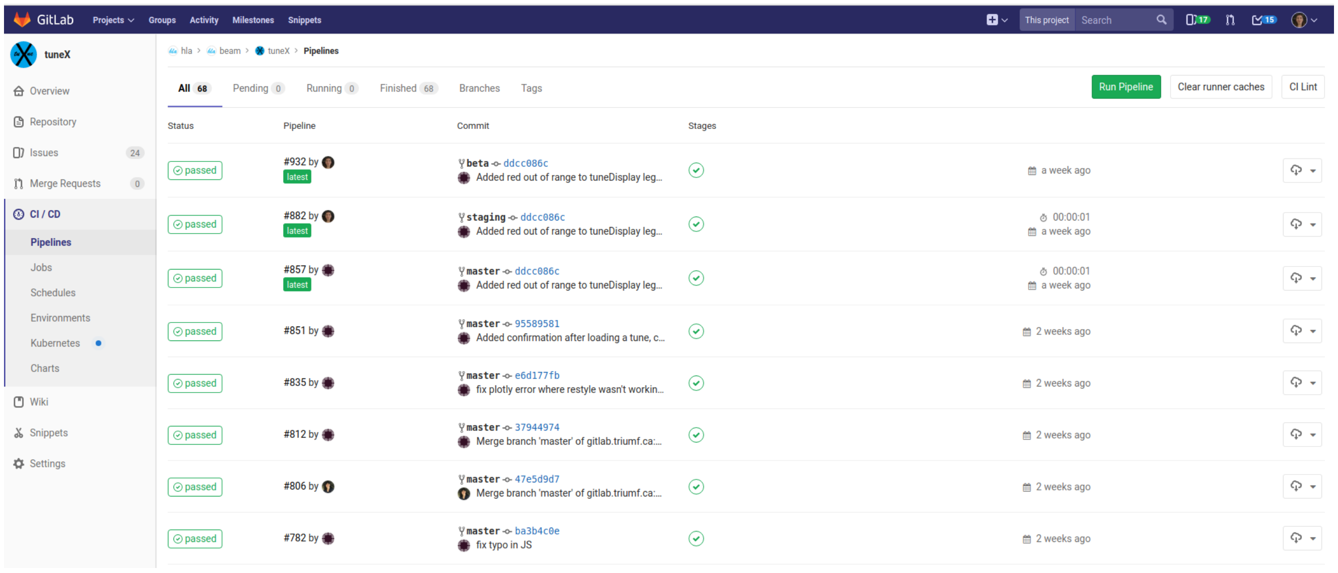
Figure 27: Pyepics in docker containers.

5 References: Tools We Use

5.1 GitLab

5.2 JupyterHub

5.3 Flask Framework


GitLab Projects Groups Activity Milestones Snippets

tuneX

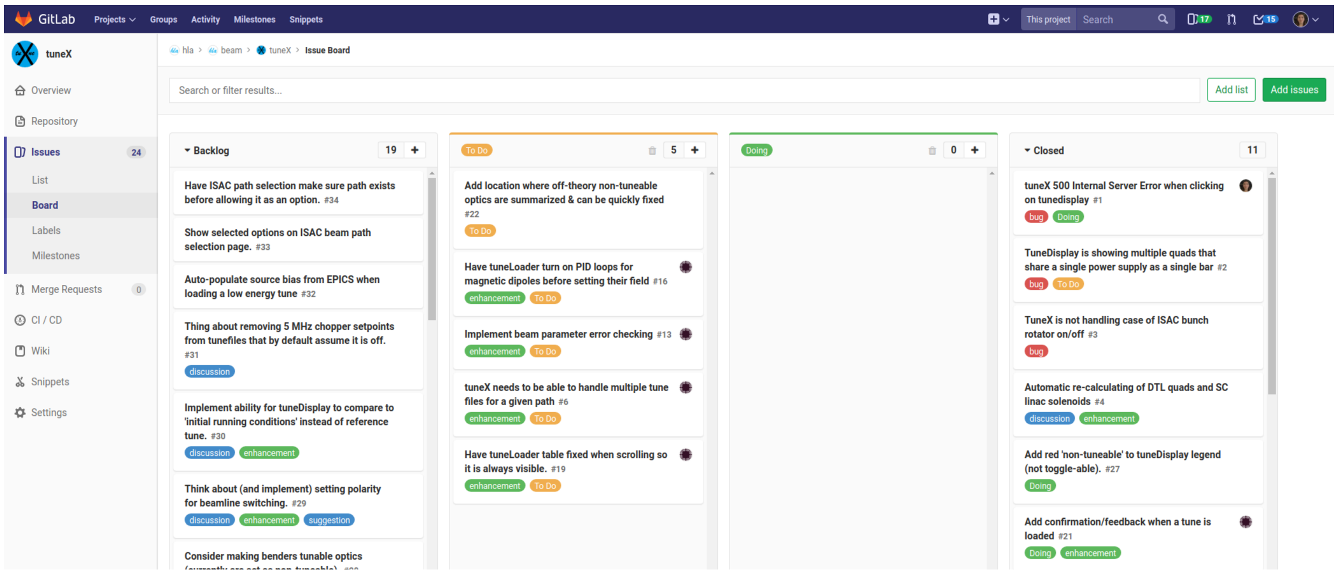
hla › beam › tuneX › Pipelines

All 68 | Pending 0 | Running 0 | Finished 68 | Branches | Tags

Run Pipeline | Clear runner caches | CI Lint

Status	Pipeline	Commit	Stages	
passed	#932 by latest	beta ddcc086c Added red out of range to tuneDisplay leg...	✓	a week ago
passed	#882 by latest	staging ddcc086c Added red out of range to tuneDisplay leg...	✓	00:00:01 a week ago
passed	#857 by latest	master ddcc086c Added red out of range to tuneDisplay leg...	✓	00:00:01 a week ago
passed	#851 by	master 95589581 Added confirmation after loading a tune, c...	✓	2 weeks ago
passed	#835 by	master e6d177fb fix plotly error where restyle wasn't workin...	✓	2 weeks ago
passed	#812 by	master 37944974 Merge branch 'master' of gitlab.triumf.ca:...	✓	2 weeks ago
passed	#806 by	master 47e5d9d7 Merge branch 'master' of gitlab.triumf.ca:...	✓	2 weeks ago
passed	#782 by	master ba3b4c0e fix typo in JS	✓	2 weeks ago

(a) `GitLab Continuous Integration Board` screenshot.


GitLab Projects Groups Activity Milestones Snippets

tuneX

hla › beam › tuneX › Issue Board

Search or filter results... | Add list | Add issues

Backlog 19

- Have ISAC path selection make sure path exists before allowing it as an option. #34
- Show selected options on ISAC beam path selection page. #33
- Auto-populate source bias from EPICS when loading a low energy tune #32
- Thing about removing 5 MHz chopper setpoints from tunefiles that by default assume it is off. #31 — discussion
- Implement ability for tuneDisplay to compare to 'initial running conditions' instead of reference tune. #30 — discussion, enhancement
- Think about (and implement) setting polarity for beamline switching. #29 — discussion, enhancement, suggestion
- Consider making benders tunable optics

To Do 5

- Add location where off-theory non-tuneable optics are summarized & can be quickly fixed #22 — To Do
- Have tuneLoader turn on PID loops for magnetic dipoles before setting their field #16 — enhancement, To Do
- Implement beam parameter error checking #13 — enhancement, To Do
- tuneX needs to be able to handle multiple tune files for a given path #6 — enhancement, To Do
- Have tuneLoader table fixed when scrolling so it is always visible. #19 — enhancement, To Do

Doing 0

Closed 11

- tuneX 500 Internal Server Error when clicking on tunedisplay #1 — bug, Doing
- TuneDisplay is showing multiple quads that share a single power supply as a single bar #2 — bug, To Do
- TuneX is not handling case of ISAC bunch rotator on/off #3 — bug
- Automatic re-calculating of DTL quads and SC linac solenoids #4 — discussion, enhancement
- Add red 'non-tuneable' to tuneDisplay legend (not toggle-able). #27 — Doing
- Add confirmation/feedback when a tune is loaded #21 — Doing, enhancement

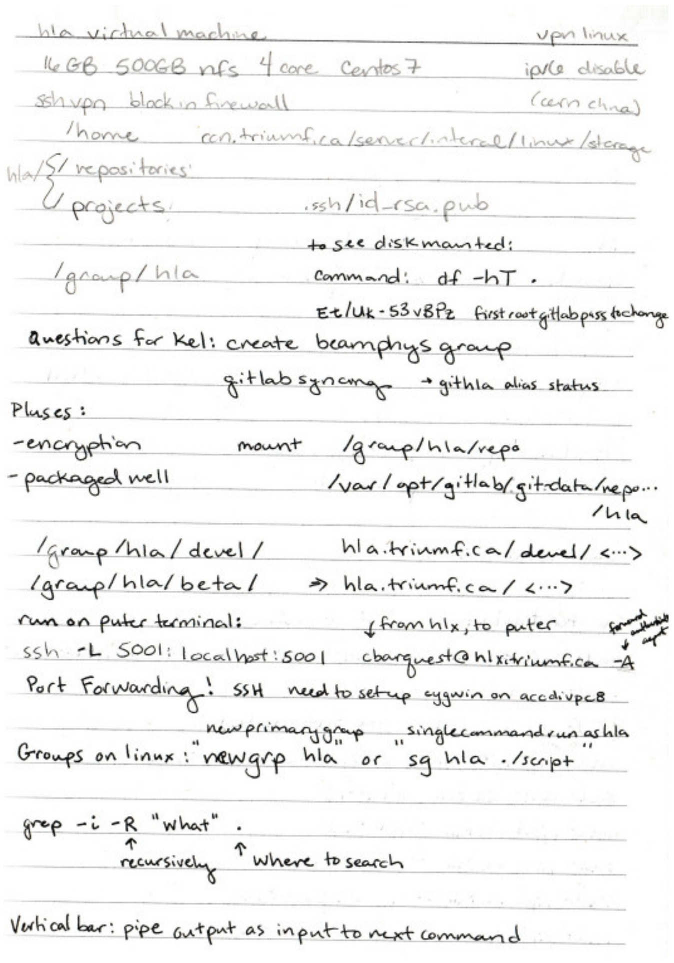
(b) `GitLab Issue Board` screenshot.

Figure 28: Some relevant GitLab screenshots.

6 Notes and Thoughts

In this section here, I'm just talking about each of the figures. Where each paragraph belongs within the document is to be determined.

Fig. 29 shows notes from a critical time period in and around May 2017.

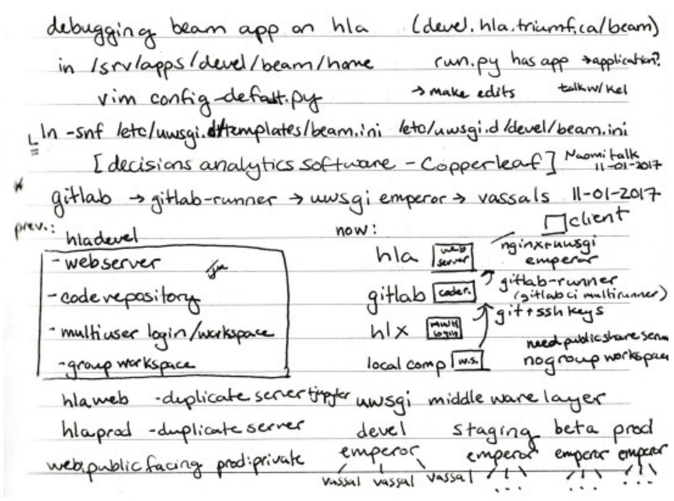


hla virtual machine vpn linux
16GB 500GB nfs 4 core Centos 7 ipv6 disable
ssh vpn block in firewall (cern china)
/home ccn.triumf.ca/server/interal/linux/storage
hla/ { / repositories'
/ projects' .ssh/id_rsa.pub
to see disk mounted:
/group/hla command: df -hT .
Et/Uk-53vBPz first root gitlab pass to change
Questions for Kel: create beamphys group
gitlab syncing → githla alias status
Pluses:
-encryption mount /group/hla/repo
- packaged well /var/opt/gitlab/git-data/repo...
/hla
/group/hla/devel/ hla.triumf.ca/devel/ <...>
/group/hla/beta/ ⇒ hla.triumf.ca/ <...>
run on puter terminal: ↓ from hlx, to puter forward authentication agent ↓
ssh -L 5001: localhost:5001 cbarquest@hlx.triumf.ca -A
Port Forwarding! SSH need to setup cygwin on accdivpc8
new primary group single command run as hla
Groups on linux: "newgrp hla" or "sg hla ./script"

grep -i -R "what" .
↑ recursively ↑ where to search

Vertical bar: pipe output as input to next command

Figure 29: Some notes on the hla setup in May 2017.



debugging beam app on hla (devel.hla.triumf.ca/beam)
in /srv/apps/devel/beam/home run.py has app →application?
vim config-default.py → make edits talk w/ Kel
ln -snf /etc/uwsgi.d/templates/beam.ini /etc/uwsgi.d/devel/beam.ini
[decisions analytics software – Copperleaf] Naomi talk 11-01-2017
gitlab → gitlab-runner → uwsgi emperor → vassals 11-01-2017

prev.:
hladevel
- webserver
- code repository
- multiuser login/workspace
- group workspace

now:
client
hla web server nginx+uwsgi emperor
gitlab coder. gitlab-runner (gitlab ci multirunner)
hlx multi login git+ssh keys
local comp w.s. need public share server, no group workspace

hlaweb -duplicate server jupyter
hlaprod -duplicate server
web: public facing prod: private

uwsgi middle ware layer
devel staging beta prod
emperor emperor emperor emperor
vassal vassal vassal ... ...

Figure 30: Some notes from brainstorming sessions thinking about hla servers where we were to where we are now.

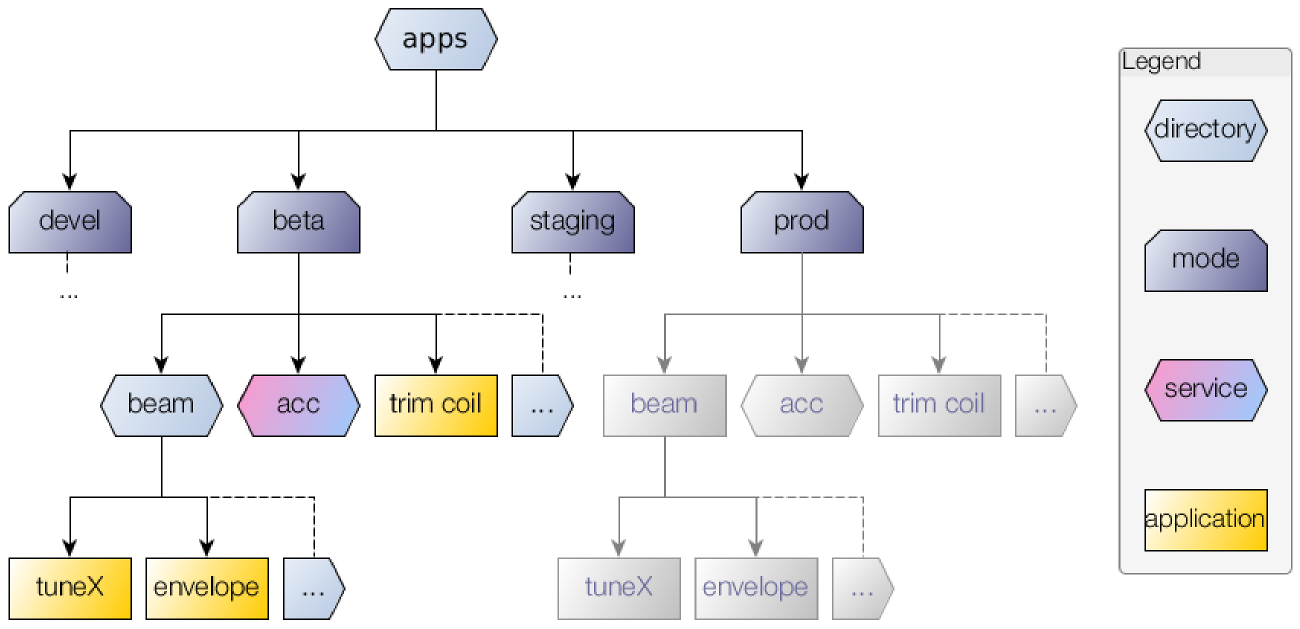
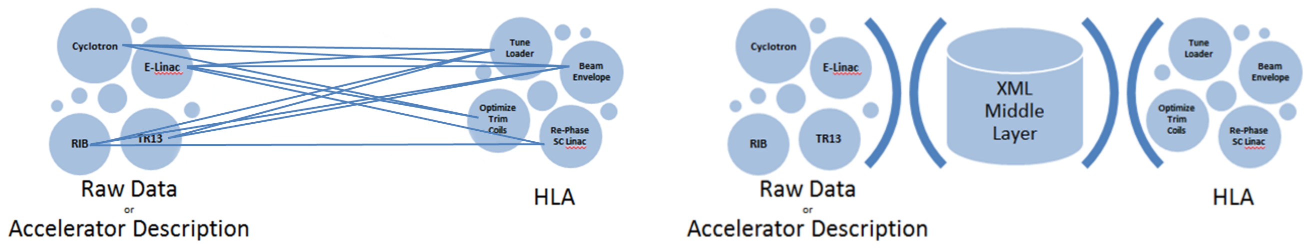
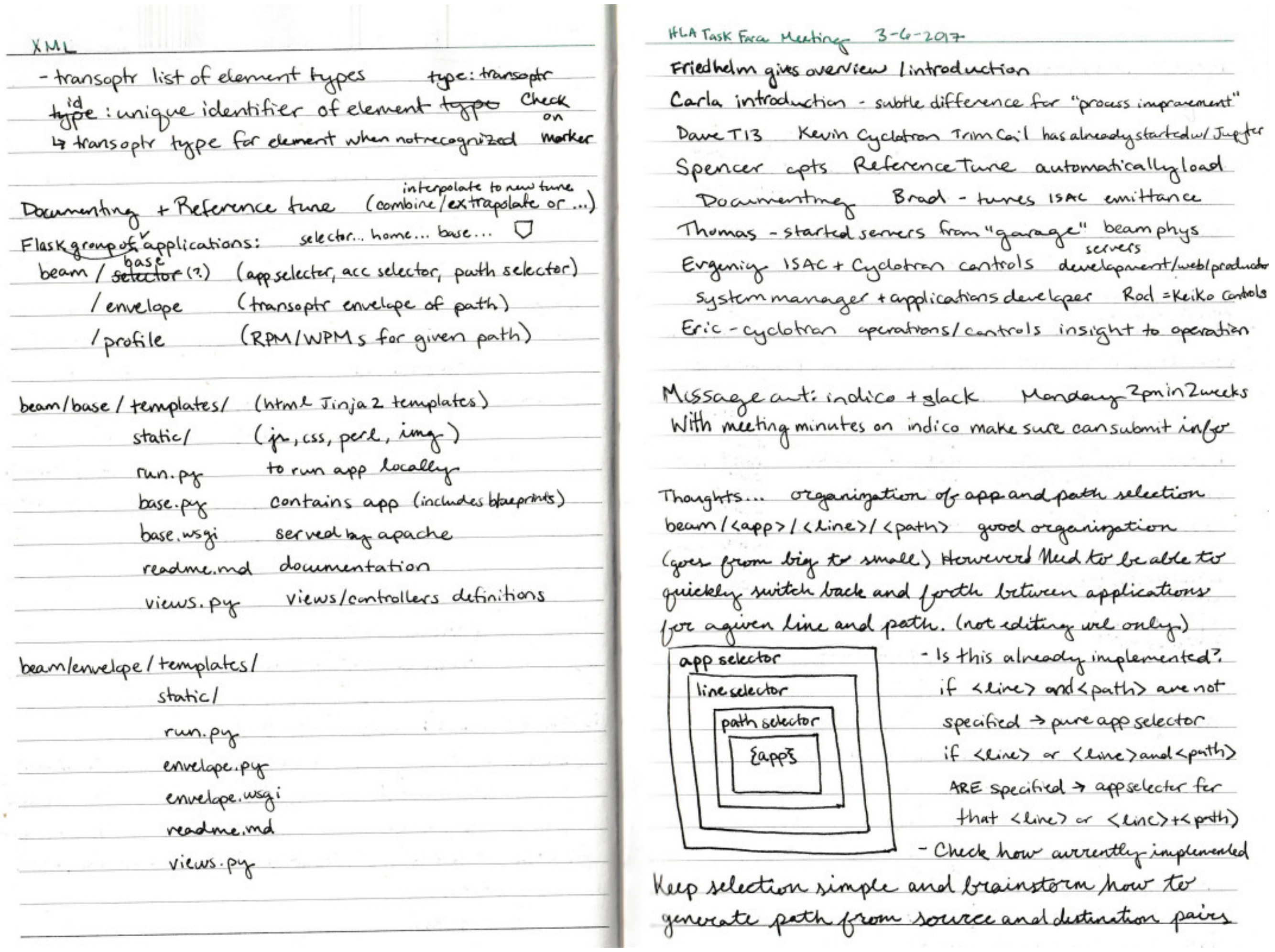


Figure 31: A caption.



(a) No middle layer–many connections. (b) With middle layer–reduce connections.

Figure 32: Schematic of XML middle layer benefit for reducing connections to maintain.



XML

- transoptr list of element types   type: transoptr
~~type~~ id : unique identifier of element ~~type~~   Check on marker
↳ transoptr type for element when not recognized

Documenting + Reference tune (combine/extrapolate or ...) interpolate to new tune

Flask ~~group of~~ applications: selector... home... base... ▽

```
beam / ~~selector~~ base (?)   (app selecter, acc selector, path selector)
       /envelope               (transoptr envelope of path)
       /profile                (RPM/WPMs for given path)

beam/base/ templates/   (html Jinja2 templates)
           static/      (js, css, perl, img)
           run.py       to run app locally
           base.py      contains app (includes blueprints)
           base.wsgi    served by apache
           readme.md    documentation
           views.py     views/controllers definitions

beam/envelope/ templates/
               static/
               run.py
               envelope.py
               envelope.wsgi
               readme.md
               views.py
```

HLA Task Force Meeting 3-6-2017

Friedhelm gives overview / introduction
Carla introduction - subtle difference for "process improvement"
Dave T13   Kevin Cyclotron Trim Coil has already started w/ Jupyter
Spencer cpts Reference Tune automatically load
Documenting   Brad - tunes ISAC emittance
Thomas - started servers from "garage" beam phys servers
Evgeniy ISAC + Cyclotron controls development/web/production
system manager + applications developer   Rod = Keiko controls
Eric - cyclotron operations/controls insight to operation

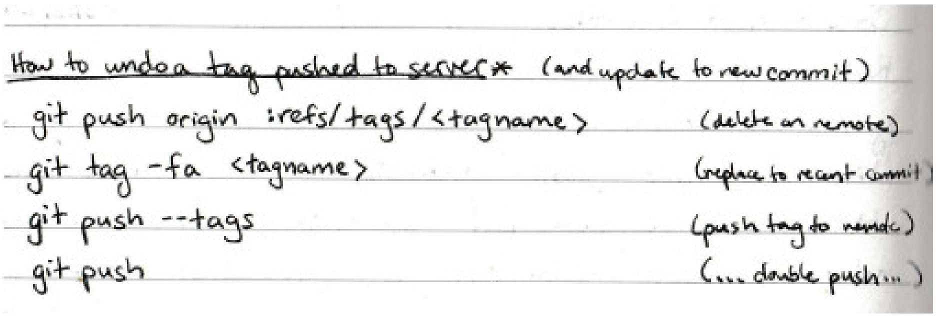
Message out: indico + slack   Monday 2pm in 2 weeks
With meeting minutes on indico make sure can submit info

Thoughts... organization of app and path selection
beam/<app>/<line>/<path> good organization
(goes from big to small) However! Need to be able to
quickly switch back and forth between applications
for a given line and path. (not editing url only)

- Is this already implemented?
  if <line> and <path> are not
  specified → pure app selector
  if <line> or <line> and <path>
  ARE specified → app selecter for
  that <line> or <line> + <path>
- Check how currently implemented

Keep selection simple and brainstorm how to
generate path from source and destination pairs

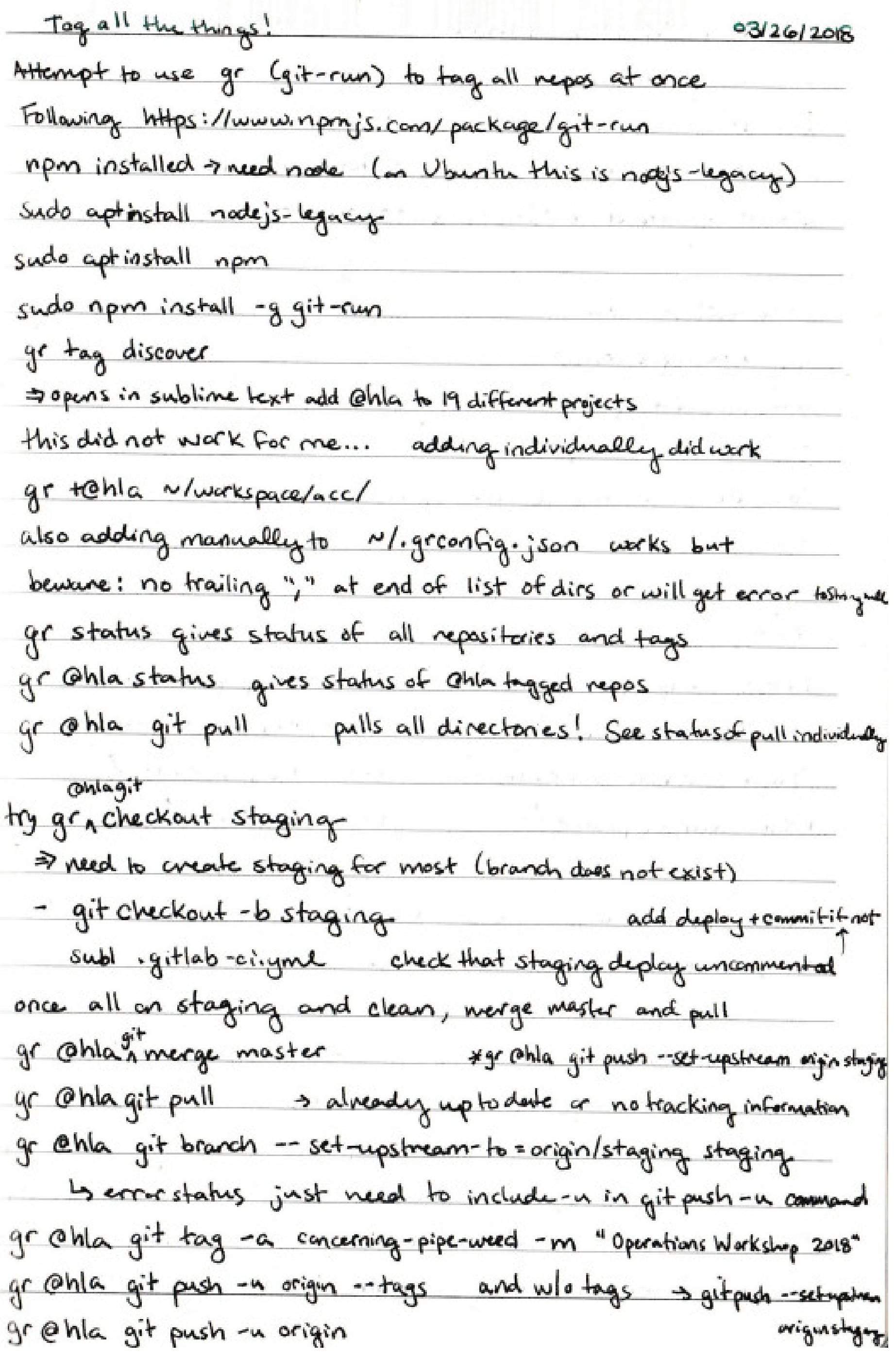
Figure 33: First HLA Task Force Meeting, plus thoughts on structure.



How to undo a tag pushed to server* (and update to new commit)

```
git push origin :refs/tags/<tagname>     (delete on remote)
git tag -fa <tagname>                    (replace to recent commit)
git push --tags                          (push tag to remote)
git push                                 (... double push...)
```

Figure 34: How to undo a pushed tag.



Tag all the things! 03/26/2018

Attempt to use gr (git-run) to tag all repos at once

Following https://www.npmjs.com/package/git-run

npm installed → need node (on Ubuntu this is nodejs-legacy)

sudo apt install nodejs-legacy

sudo apt install npm

sudo npm install -g git-run

gr tag discover

⇒ opens in sublime text add @hla to 19 different projects

this did not work for me... adding individually did work

gr +@hla ~/workspace/acc/

also adding manually to ~/.grconfig.json works but

beware: no trailing "," at end of list of dirs or will get error toString[illegible]

gr status gives status of all repositories and tags

gr @hla status gives status of @hla tagged repos

gr @hla git pull pulls all directories! See status of pull individually

try gr $\overset{\text{@hla git}}{\wedge}$ checkout staging

⇒ need to create staging for most (branch does not exist)

- git checkout -b staging add deploy + commit if not ↑

  subl .gitlab-ci.yml check that staging deploy uncommented

once all on staging and clean, merge master and pull

gr @hla $\overset{\text{git}}{\wedge}$ merge master *gr @hla git push --set-upstream origin staging

gr @hla git pull → already up to date or no tracking information

gr @hla git branch --set-upstream-to=origin/staging staging

↳ error status just need to include -u in git push -u command

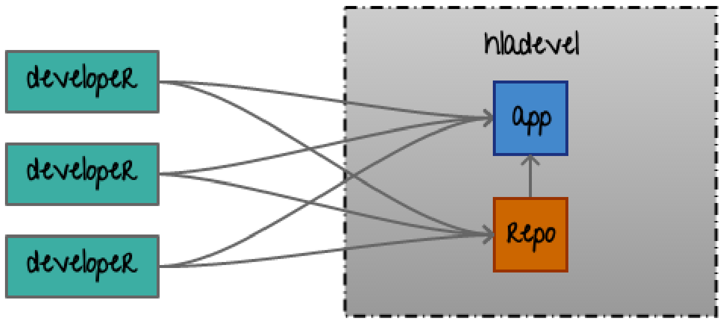
gr @hla git tag -a concerning-pipe-weed -m "Operations Workshop 2018"

gr @hla git push -u origin --tags and w/o tags → git push --set-upstream origin staging

gr @hla git push -u origin

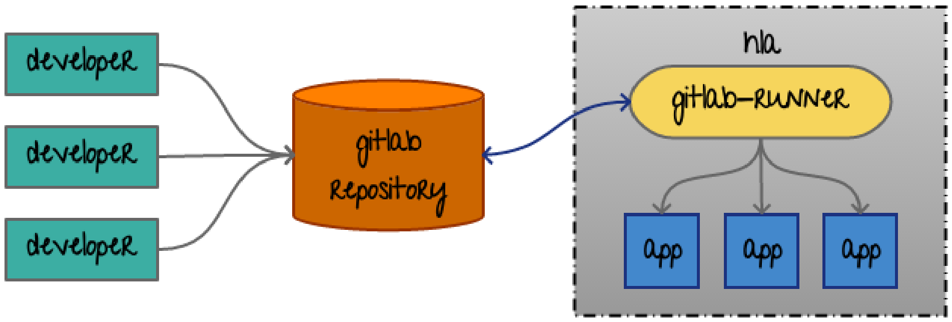
Figure 35: How to tag all the things.

`http://hladevel.triumf.ca/projects/BeamEnvelope/BeamEnvelope-ISAC.html`



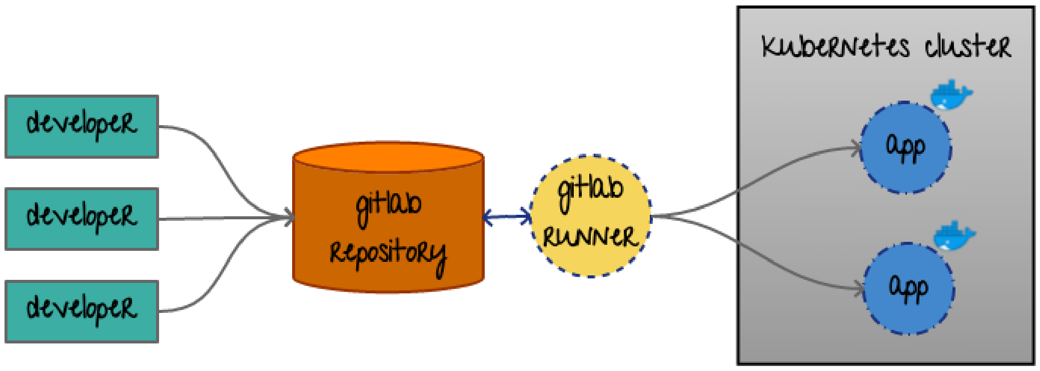
(a) Manual application deploy to `hladevel`.

`https://devel.hla.triumf.ca/beam/envelope/isac/`



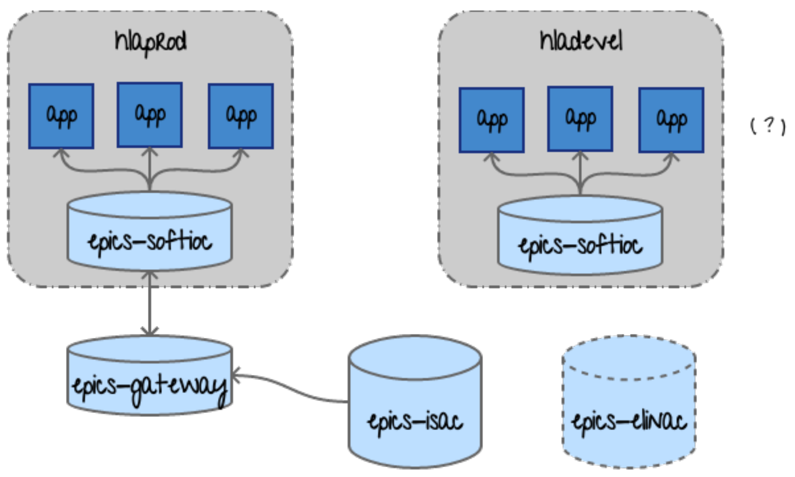
(b) `GitLab` deploy to `hla`.

`https://dev.envelope.triumf.ca/`

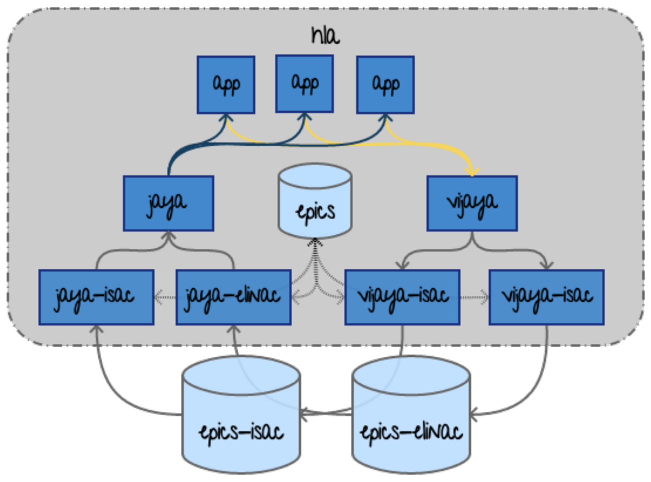


(c) `Kubernetes` deploy to `?`.

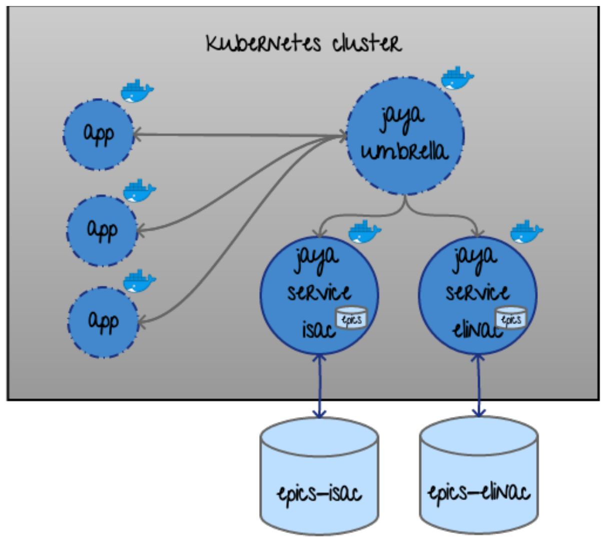
Figure 36: Evolution of application deployment.



(a) Gateway with `hla*`.

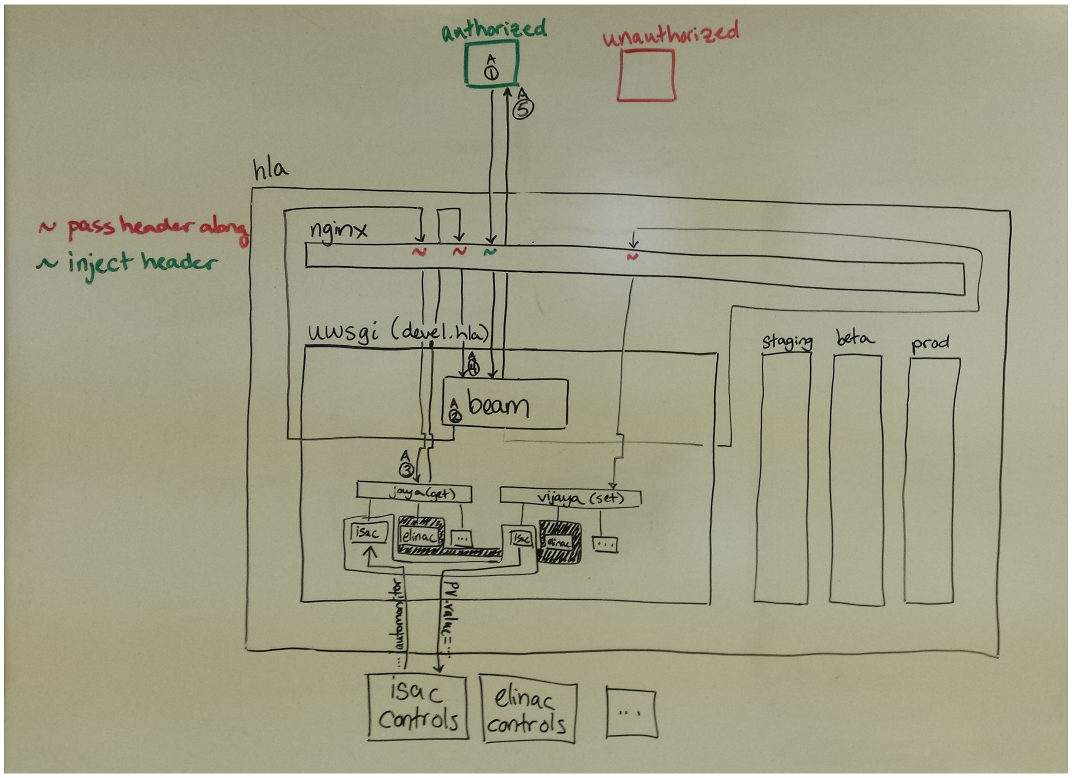


(b) Pyepics on `hla`.

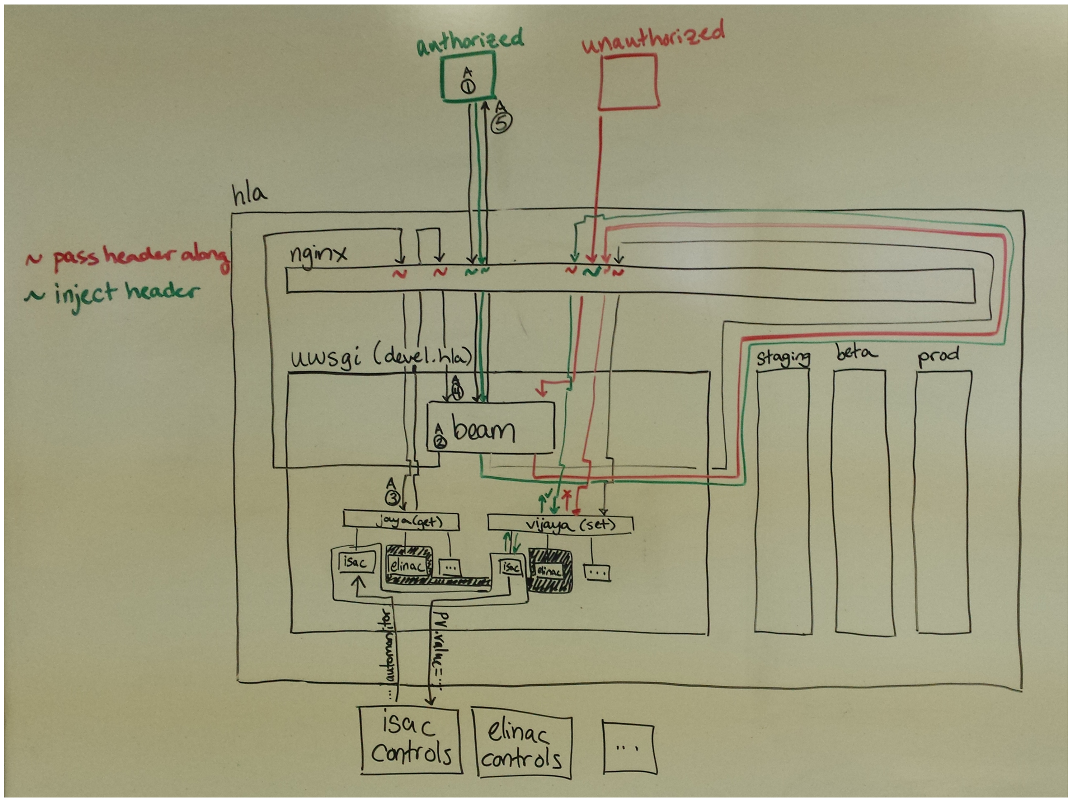


(c) Pyepics in docker containers.

Figure 37: Evolution of epics I/O.



(a) Authorized user only.



(b) Including unauthorized user.

Figure 38: Whiteboard diagrams from March 13, 2018 depicting injected headers for authorization scheme, including authorized users in green and unauthorized users in red.

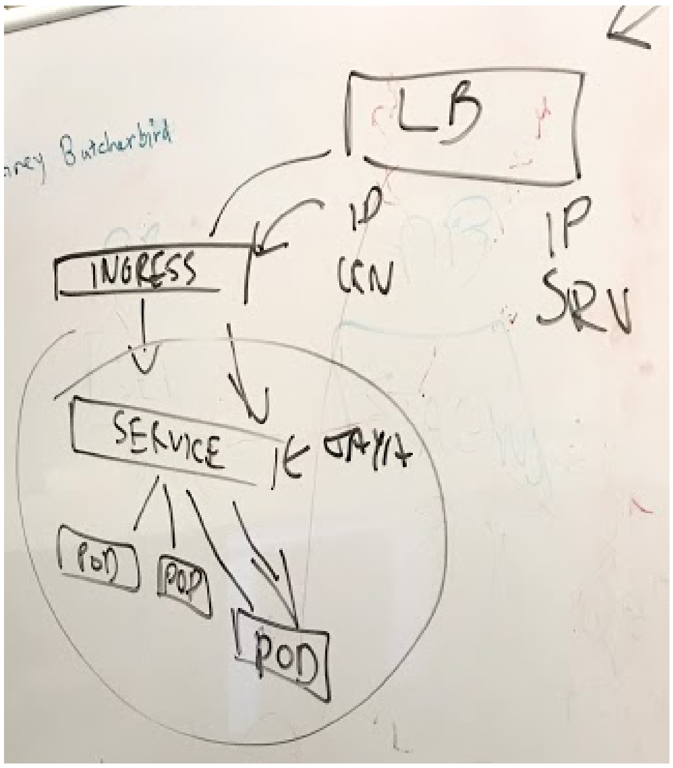


Figure 39: Whiteboard diagram of kubernetes structure.