GitLab, Containers, and Continuous Integration

code.jlab.org

Christopher Dilks

CLAS Collaboration Meeting, November 2024

What’s a Container?

- Put all the software you need in a self-contained “box”
  - Terminology:
    - Image: the “box” which you can distribute to others
    - Container: a *running* instance of the image
  - Software options include Docker, Podman, Apptainer, Singularity

- No need to build your software or dependencies
  - Creating the image does the building or downloading of software
  - This is done by a build “recipe”, e.g., a Dockerfile

alamy
Image ID: 2HCCM6F
www.alamy.com

Why are we doing this?

- Portability: easy to share images
  - Your development environment on ifarm can be reproduced locally
  - Run your code in containers
  - Share images with others

- If everyone uses the same images, we expect:
  - The same results → reproducibility
  - The same bugs → facilitates maintenance

- Preservation of ***running software***
  - Containerize *your* analysis!

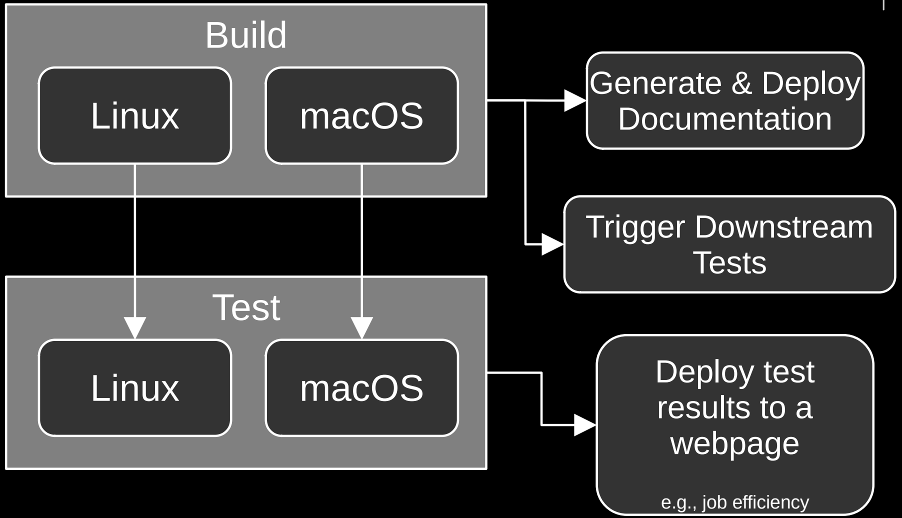
GitLab: a place to build images (and more)

- **GitLab provides a remote host for ‘git’ repositories**
  - ‘git’ is an open source distributed Version Control System
  - https://gitlab.com/ - the “main” GitLab website
    - GitLab can also be “self-hosted”: your own GitLab instance
    - By the way, GitHub also is a remote host at https://github.com/
- **JLab hosts a GitLab instance: https://code.jlab.org/**
  - HallB’s code: https://code.jlab.org/hallb
  - All the ‘git’ commands are the same (‘git commit’, ‘git push’, etc.)
    - Same concepts, e.g., branches, merges
    - A request to merge a branch, usually to the ‘main branch’, is called:
      - Pull Request (PR) in GitHub
      - Merge Request (MR) in GitLab
  - All the buttons you’re used to clicking on GitHub are (most likely) found on GitLab

Continuous Integration (CI)

- Both GitHub and GitLab offer “Continuous Integration” (CI)
  - Basic idea: run jobs, triggered by some ‘git’ action, usually by
    - Commits on a MR branch
    - Any commit on the Main branch
    - Other custom triggers (scheduled, manual, etc.)
  - What kind of jobs? Here are some examples:
    - See the diagram to the right →

- CI helps ensure software stability
  - MRs should not be merged unless the jobs pass
  - Previous jobs give a sense of “history” of the software project (which can also be useful for debugging)
  - Automation helps make sure we don’t “forget” something



Building Images with CI

- Creating the image does the building or downloading of software
  - We use GitLab's Continuous Integration to build images
  - Images for CLAS12 are deployed to GitLab's Container Registry:
    - https://code.jlab.org/hallb/clas12/clas12-containers/container_registry
    - These images are NOT "production ready" yet, but you're welcome to try, e.g.,
      - `apptainer pull docker://codecr.jlab.org/hallb/clas12/clas12-containers/clas12_analysis:latest`
  - GitLab has a ton of other features we may take advantage of

clas12-containers

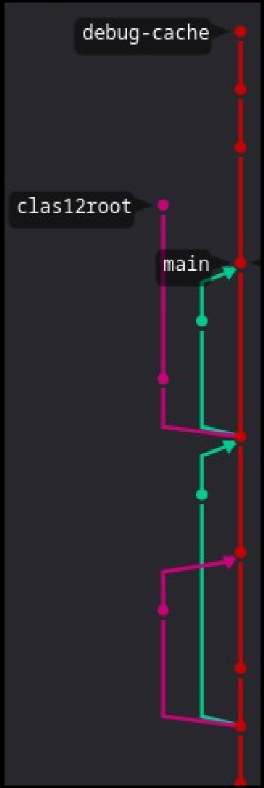
- GitLab repository for building and deploying images for CLAS12 (and related) software
  - **https://code.jlab.org/hallb/clas12/clas12-containers**
  - Uses CI to automatically build and test images
- Still in the early stages of development!
  - Contributions welcome, but should be discussed
  - Merge requests are *always* welcome (since they trigger image builds)
  - Requests for certain software to be included are also welcome, just ask!
- Some issues (see https://code.jlab.org/hallb/clas12/clas12-containers/-/issues)
  - Versioning → need to sync with Module Environment files (clas12-env)
  - Build cache usage → some things are rebuilding (viz. ROOT) when they *should* be using the cache build
  - Documentation → there isn't any yet
  - Missing license and contributing guidelines
  - Add more software

clas12-containers Strategy

- Main branch commits
  - Images get tagged “latest”

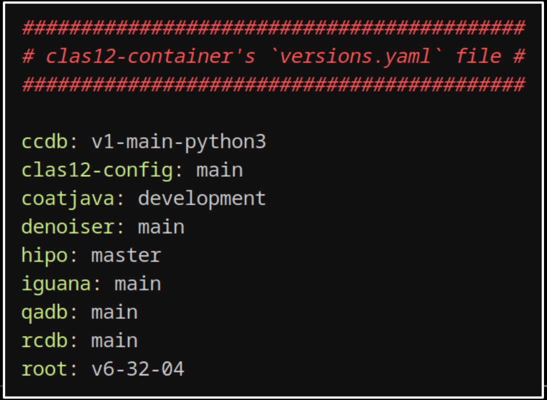
- Tagged versions of clas12-containers
  - Image get tagged as “v#####”, with the version number
  - Need to include a list of the version numbers of software

- Merge Requests (MR)
  - Images are tagged as “MR-….”, with the MR number
    - These images are eventually auto-deleted from the registry
  - Each commit on the MR branch re-triggers image builds
    - First one is a full rebuild
    - Subsequent ones use the cache

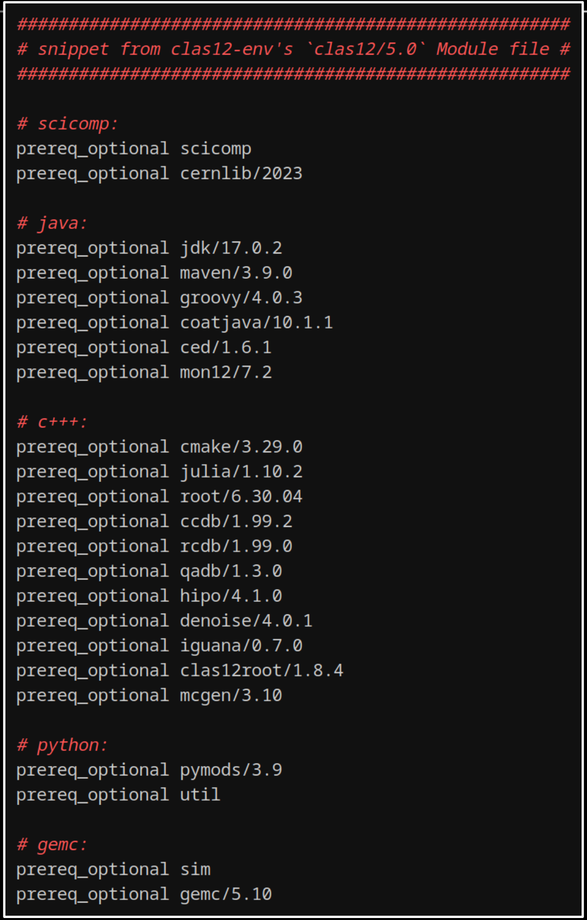


Syncing with our Module Environment

- Need to stay in sync with clas12-env
  - Currently just using everyone’s ‘main’ branch for now, to get us started
  - Need to switch to using specific versions
  - clas12-containers will only build the “latest” versions
    - Module Environment can handle the dependency combinatorics, whereas clas12-containers will not, instead having only one version of each package
    - Older versions will be in tagged images; we can try to “maintain” them, if needed (e.g., re-build against latest upstream, etc.)
    - The tag number should match the ‘clas12’ module number



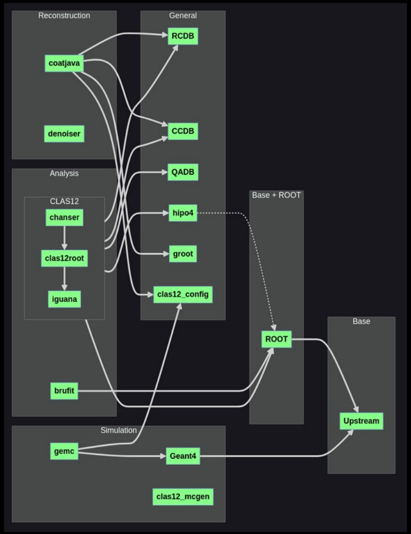
```
############################################
# clas12-container's `versions.yaml` file #
############################################

ccdb: v1-main-python3
clas12-config: main
coatjava: development
denoiser: main
hipo: master
iguana: main
qadb: main
rcdb: main
root: v6-32-04
```



```
#######################################################
# snippet from clas12-env's `clas12/5.0` Module file #
#######################################################

# scicomp:
prereq_optional scicomp
prereq_optional cernlib/2023

# java:
prereq_optional jdk/17.0.2
prereq_optional maven/3.9.0
prereq_optional groovy/4.0.3
prereq_optional coatjava/10.1.1
prereq_optional ced/1.6.1
prereq_optional mon12/7.2

# c++:
prereq_optional cmake/3.29.0
prereq_optional julia/1.10.2
prereq_optional root/6.30.04
prereq_optional ccdb/1.99.2
prereq_optional rcdb/1.99.0
prereq_optional qadb/1.3.0
prereq_optional hipo/4.1.0
prereq_optional denoise/4.0.1
prereq_optional iguana/0.7.0
prereq_optional clas12root/1.8.4
prereq_optional mcgen/3.10

# python:
prereq_optional pymods/3.9
prereq_optional util

# gemc:
prereq_optional sim
prereq_optional gemc/5.10
```

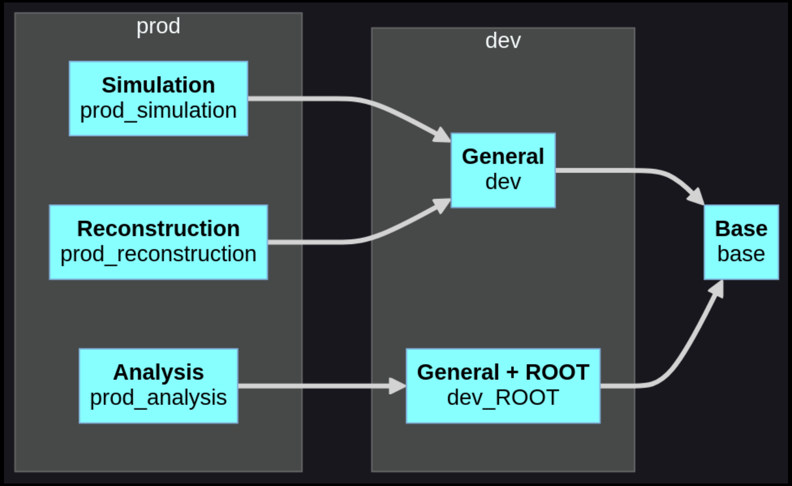
What images are we building?

Package dependency graph



inspires

Images



- "prod" images are for users
- "dev" images are for development, and serve as "bases" for the prod images
- "base" is the base Linux distribution + updates + common packages

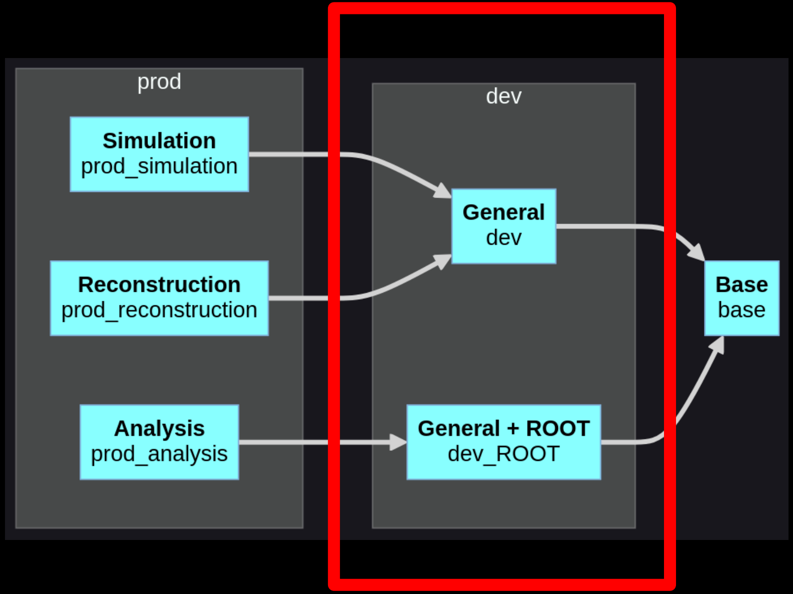
Base Image

- Currently based on Arch Linux
  - The latest version of everything: “the bleeding edge”
  - Minimal → smaller image sizes
  - Arch Linux repositories have a *lot* of software available
  - Still supports x86-64 v1 baseline (old OSG nodes)
  - Does *not* support ARM, e.g., newer Macs
    - Arch Linux ARM does, but doesn’t seem as well maintained
  - Need to think about security

- Alternatives
  - Alma9: as used on ifarm; baseline is x86-64 v2 (some OSG nodes are still v1); a lot of software packages are held back on old versions
  - Debian – EIC has been using this
  - openSUSE Tumbleweed – supports both x86 and ARM, and is also staying near the bleeding edge
  - **We’re open to other ideas**

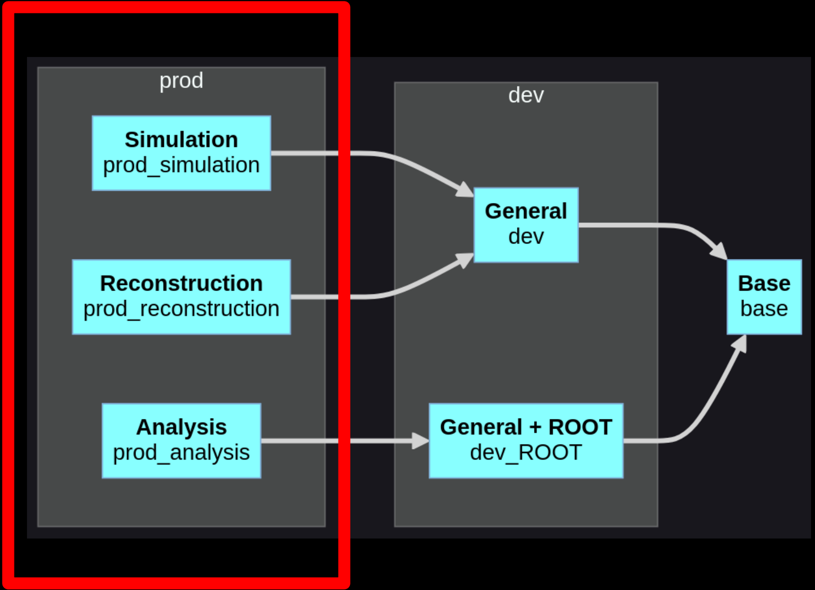
dev and dev_root images

- "Development" images: common CLAS12 software *dependencies*
  - RCDB
  - CCDB
  - QADB
  - ROOT (in dev_root, *not* in dev)
  - HIPO
  - clas12-config



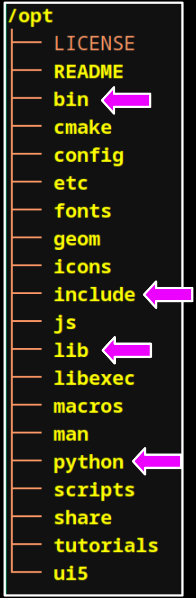
production images

- Reconstruction
  - coatjava
  - denoiser
- Analysis
  - iguana
  - clas12root
  - chanser
  - brufit
- Simulation
  - Synergy with OSG images (Maurizio)

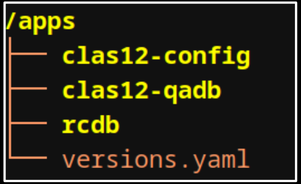


What's in the Image

- **/opt**
  - Common installation prefix
  - … for software that generates an installation tree
    - e.g., directories such as bin/, include/, lib/
  - Examples: clas12root, iguana, HIPO
  - Environment variables ($PATH, $LD_LIBRARY_PATH, etc.) include this
  - ROOT is also installed here, but may be moved elsewhere (/apps/ROOT?)
- **/apps**
  - Common location for all the rest of the software, that does not generate an installation tree
  - Examples: QADB, clas12-config



```
/opt
├── LICENSE
├── README
├── bin
├── cmake
├── config
├── etc
├── fonts
├── geom
├── icons
├── include
├── js
├── lib
├── libexec
├── macros
├── man
├── python
├── scripts
├── share
├── tutorials
└── ui5
```



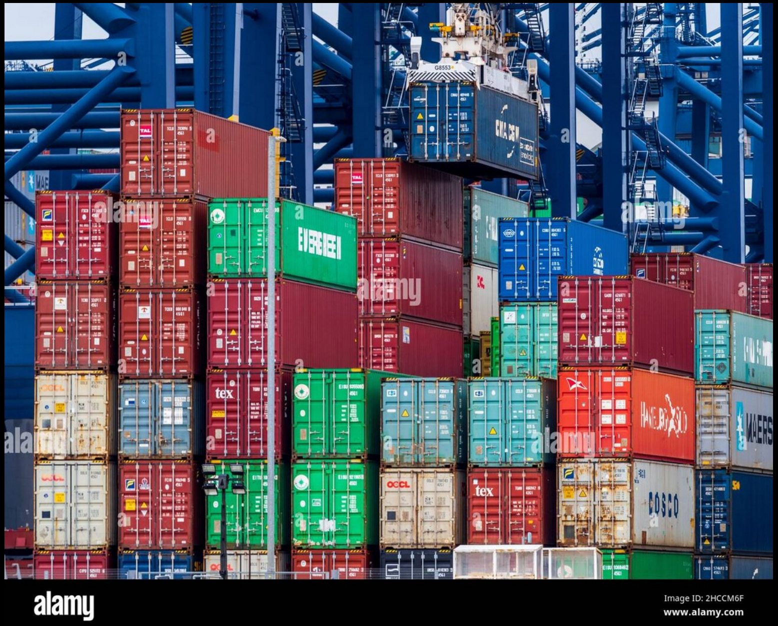
```
/apps
├── clas12-config
├── clas12-qadb
├── rcdb
└── versions.yaml
```

How do I build a new image?

- Open a merge request!
- Then you can use it
  - But Container Registry will delete it eventually, unless your MR is approved and merged

Containerize Your Analysis

- For the preservation of your analysis, consider adding a Dockerfile which builds your analysis code
- Complicated dependencies? Complicated setup? Containerize!
- Consider basing your image off of one of clas12-containers's images
  - https://code.jlab.org/hallb/clas12/clas12-containers
  - Send a merge request (MR), then you can use the CI to build your image



Summary

- **clas12-containers**
  - Build images with CLAS12 software and more
  - Using JLab’s GitLab Continuous Integration and Container Registry
- **Preservation of running code**
  - cf. Preservation of Data efforts
  - cf. Iguana → Preservation of data analysis algorithms
- **Portability**
  - Shared features (and shared problems)
  - Streamline local development
- **Containerize your analysis**
  - For all the above reasons

backup

Under the Hood

- The Runners
  - 32 CPUs
  - ~380 GB RAM
  - However, job constraints limit us to (which may change depending on load)
    - 12 CPUs
    - 4 GB RAM
    - Issue: not enough memory per core to take full advantage

- The Software
  - OpenShift + Kubernetes for the runners
  - Kaniko to build a Docker image within a running container
    - Issue: Kaniko is no longer maintained!!!
    - SciComp is working on Buildah support

Red Hat
OpenShift

kubernetes

Kaniko

buildah