ldmx-sw Tutorial Course: Advanced Methods

Tom Eichlersmith

University of Minnesota

eichl008@umn.edu

June 2, 2021

Outline

1. Pre-Requisites
2. Simulation Tools
3. Analysis Tools
4. Visual Tools
5. Containers
6. Outside the Container
7. Developing for ldmx-sw
   - git
   - Code Formatting
   - PR Workflow
8. Questions

Pre-Requisites

Pre-Requisites

- Familiarity with `ldmx-sw` already

Warning

If you are more inexperienced with `ldmx-sw`, much of this information may be out of your reach. That is okay. Seeing it early will help you learn it later if you need to use it.

Simulation Tools

Simulation Filters

- Implemented as `UserActions`
- Used mainly for event filtering and track filtering
  - Event Filtering: Abort the sim event early to save time if criteria aren't met
  - Track Filtering: Make sure sim saves a specific particle information to output file
- Check out `Biasing` module for examples
- **Very flexible** — Handles to different points in the simulation
  - At **end** of each step
  - Choose which tracks are processed first ("classification")
  - A handle to when a stack is emptied ("stage")
  - Start and end of processing each track
  - Start and end of processing each event
  - Start and end of processing the run
  - Any mixture of the above

Biasing Operators

SimCore/BiasOperators

- Usually paired with filters to make sure sample has interactions of interest
- Using Bias Operator means the event weights will differ from one $\Longrightarrow$ Include these event weights in histograms

C++

```
// weight for this entry comes after the value
my_hist.Fill(value, weight);
my_2d_hist.Fill(x,y, weight);
// pointers are also common
my_hist->Fill(value, weight);
my_2d_hist->Fill(x,y, weight);
```

Python

```
# weight comes after value
my_hist.Fill(value, weight)
my_2d_hist.Fill(x,y, weight)
```

Analysis Tools

HistogramPool

- Exemplified by `PhotoNuclearDQM` analyzer in `DQM` module
- `create`: Make a new histogram by name and type
- You can tell the HistogramPool what kinds of 1D histograms to create in your python configuration by using `build1DHistogram` member function of a processor
  - Then you don't need to `create` those histograms in the Analyzer code
- `get`: Get a histogram by name for extra configuration (if need be)
- `fill`: Fill the named histogram with the input value
- `setWeight`: Change weight for **all following histograms**
  - Helpful to set weight at the beginning of each event

NtupleManager

- Only needs to be done in Analyzer code
- `create`: Make a new tree to put variables in
- `addVar`: Put a new variable name into a tree (template)
- `setVar`: Set the variable value during processing

Upcoming Changes

Will soon be using the same buffer for the NtupleManager and the Event Bus, so the NtupleManager will soon be able to handle vectors as well as individual values.

Skimming Events

After using `setStorageHint` in your processor, you also need to tell the process that it should listen to your processor.

Hint?

Called a "hint" for the multiple-hint case — multiple different processors could provide competing hints and the underlying Framework will tally the votes.

In your python configuration file:

```
myAna = ldmxcfg.Analyzer( "myAna" , "ldmx::MyAnalzer" )
# p is Process object created earlier
p.skimDefaultIsDrop() # drop events unless told otherwise
p.skimConsider(myAna.instanceName) # tell Process to listen to myAna
```

Drop/Keep Event Objects

Maybe you know *for a fact* that you will *never* use the simulated hits later in your analysis chain and you want to save some space in your output file.

In your python configuration file:

```
# p is Process object created earlier
p.keep = [
    # don't put any collection with 'EcalSim' in the name into output file
    "drop EcalSim.*",
    ]
```

Drop/Keep Event Objects

Another use-case would be re-running the digitization and reconstruction after some developments. You don't want to keep the old stuff, so you can drop it.

In your python configuration file:

```
# p is Process object created earlier
p.keep = [
    # don't save Digi and Rec of ending with the 'old' pass
    "drop .*Digi.*old", "drop .*Rec.*old"
    ]
```

Visual Tools

Event Display

eve

```
cd build
ldmx cmake -DBUILD_EVE=ON ..
ldmx make install
ldmx eve file_to_view.root
```

Good Stuff

- Allows you to look at events in 3D space with detector drawn
- GUI hosted by ROOT to interact with 3D model and event objects

Bad Stuff

- Detector geometry hard-coded, so it may not align with detector GDML
- Not all event objects have a drawing method coded for them

Not tested on Windows

Simulated Geometry Display

g4-vis

Look at the ▸ SimCore README for more information

```
cd Detectors/data/ldmx-det-v12
ldmx g4-vis detector.gdml
```

Good Stuff

- Allows you to see the geometry *actually* imported into the simulation
- Compiled with ldmx-sw by default

Bad Stuff

- No mouse interaction (change perspective by typing out commands)
- No coloring

Not tested on Windows

Containers

Container Basics

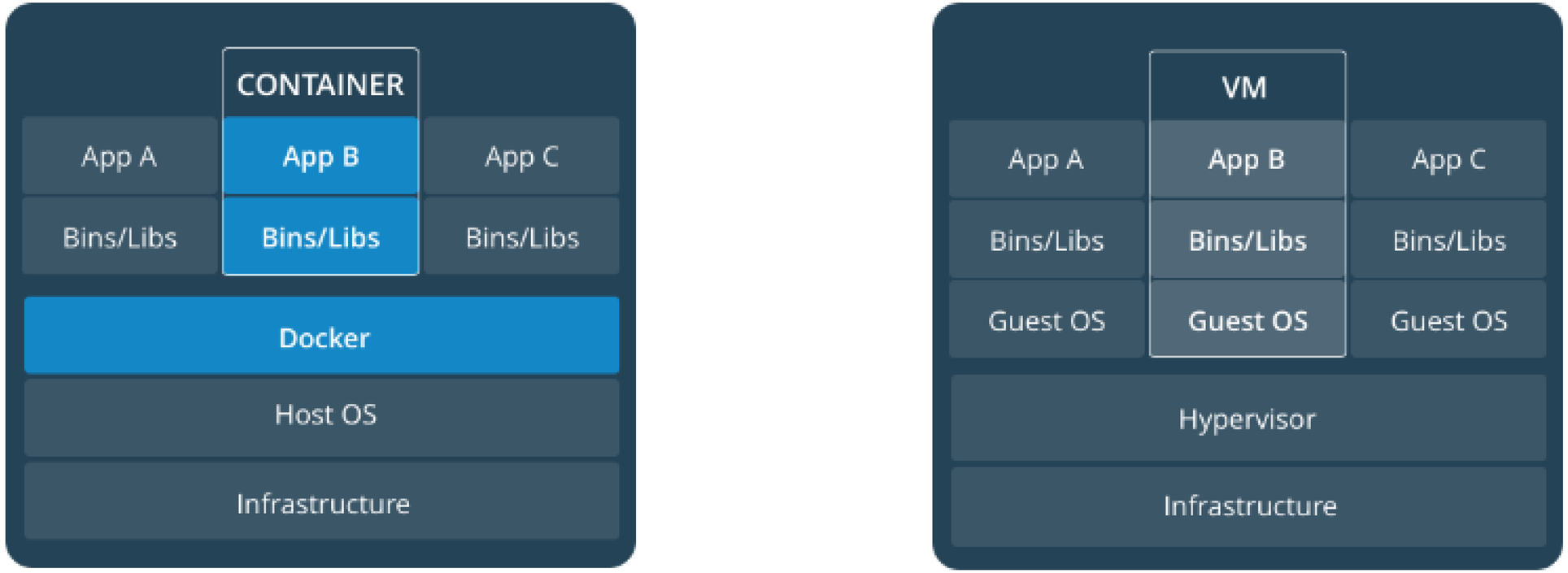

Figure: Taken from Docker Docs

Persistence

Changes made inside the container that are not mounted to the host system will not be saved.

Container Basics

Vocabulary

- A **container** is started by running an **image** which is **built** with the necessary software inside of it.
- You can **pull** and **push** images from **repos** on DockerHub.
- To access files on the host system, you need to **mount** the directory (or any of its parents) to the container when it is run[a]

Warning : We use "container" and "image" interchangeably in our discussions and guides. This is not techinically correct, but for most use cases does not lead to a misunderstanding.

[a] For the Unix enthusiasts, symlinks don't work. The actual file needs to be inside the mounted directory.

Persistence

To "persist" changes to the container, you need to build (or rebuild) another image.

Our Container Design

- `ldmx-sw` (and any other LDMX-related files) are in some directory — call this directory the **ldmx base directory** (the `everything-ldmx` directory from last time)
- Construct an image that has the necessary dependencies of ldmx-sw installed inside of it
- Mount the ldmx base directory to the container when running and then do all configuring/compiling/linking/running inside the container
- Write build/install files to another directory inside the ldmx base directory *so it is persisted after the container is removed*

`ldmx`

The `ldmx` command wraps this container running nonsense together with some other helpful bash-script functions to make a more user-friendly computing interface.

Our Image Design

Development Image

- Has all the dependencies necessary to compile and run ldmx-sw
- Is used to **develop** ldmx-sw
- DockerHub Repo: ldmx/dev

Production Image

- Built *on top* of a development image
- Has a specific version of ldmx-sw compiled and installed inside of it
- Is used to **produce** large samples using ldmx-sw (and occasionally used to reconstruct/analyze large samples too)
- DockerHub Repo: ldmx/pro

Make Changes to the Container Image

GitHub actions and workflows are used to build these images on GitHub's servers, so you don't have to use your own computer's resources.

New or Alter Dependency of ldmx-sw?

Need to build a new development image $\Longrightarrow$ ▸ LDMX-Software/docker

Alter Dependency: Investigate physics behavior of a different version of Geant4. ▸ Guide

New Dependency: A Common Tracking Software (ACTS) ▸ docker/dd4hep-acts branch

Updates or Fixes to ldmx-sw?

Need to build a new production image $\Longrightarrow$ ▸ How To Guide

Warning : We will be updating the method with which to build a custom production image.

Specific production image for ECal as Target analysis: ▸ tomeichlersmith/eat

Outside the Container

bash

bash is a moderately advanced "shell"[1]
It is the standard shell for the LDMX computing environment (i.e. the `ldmx-env.sh` script assumes you are running inside bash).

Why bash?

- Default shell on Ubuntu (common and user-friendly Linux distribution)
- Available on iOS (default shell on iOS got changed to tcsh a few versions back)
- Some other (more advanced) shells are based on it, so `ldmx-env.sh` will still work
- I (Tom Eichlersmith) like it and am used to it, so I chose it.

[1] A "shell" is a text interface to your computer. Usually a "terminal emulator" is used to connect you (a graphical user) to a shell. When connecting to a remote computer using `ssh`, a shell is started after you connect.

ROOT

Many LDMX users have ROOT installed outside of the container on their computers.

Why?

- Can use `rootbrowse` if the container is not connecting to your display
  - This is a common problem because the container–display connection is highly dependent on your container runner and how it interacts with your OS.
- Can run some scripts faster because there is no lag time caused by starting and stopping the container
- Can use ROOT to look at files not visible to the container
- Hold-over from the pre-container days of LDMX

My Suggestion : Wait to install ROOT outside the container until you need it.

Installing ROOT

Developing for ldmx-sw

git Primer

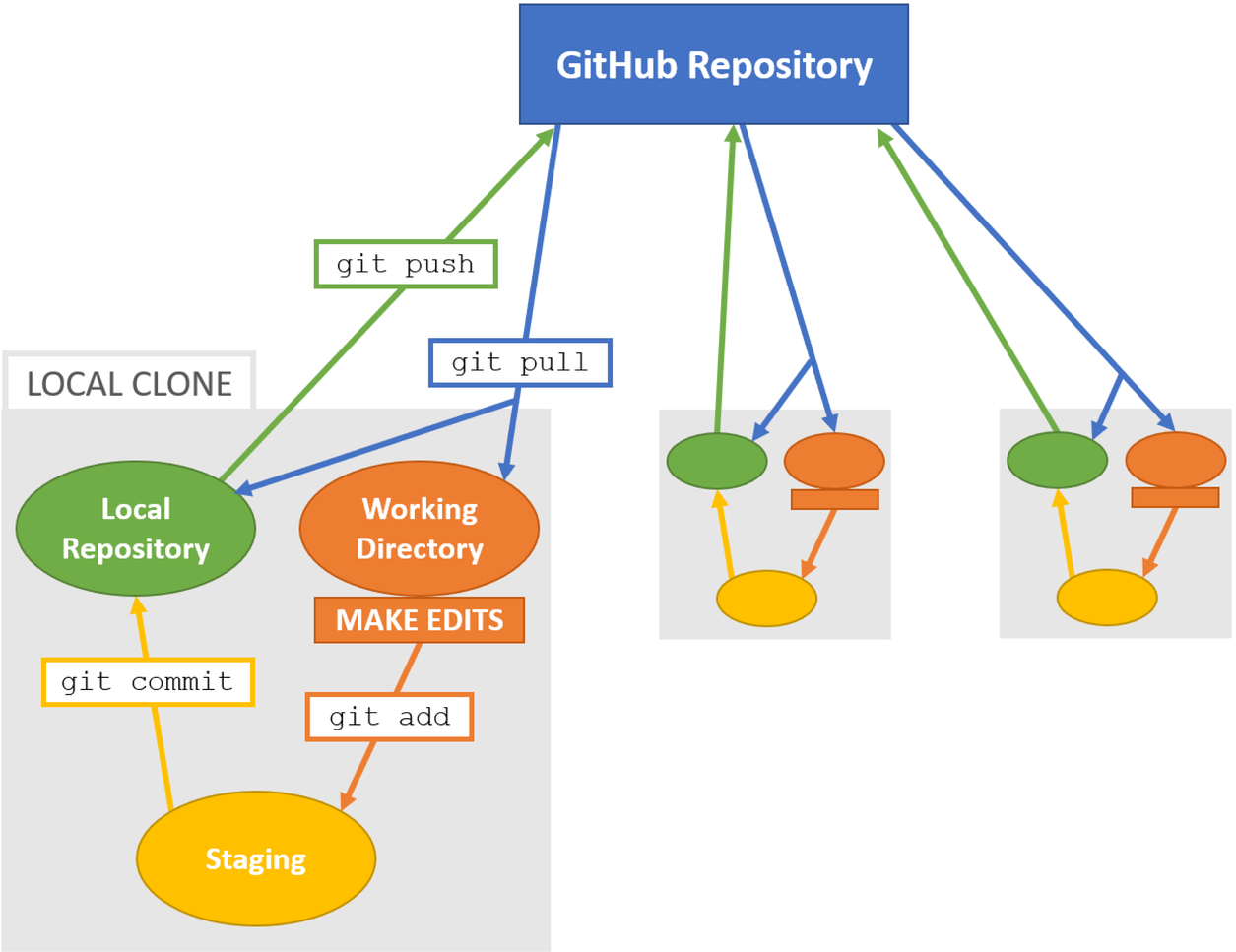

Figure: Look at ▸ Git Started for more surrounding this figure.

git branches

- When starting on a development project (no matter how small), it is good practice to open an "issue" in the relevant repository[2] and explain your plan and goals.
  - Garner insight from other developers of ldmx-sw
  - Alert other developers to your work so they don't do redundant work
  - Persist notes about the project incase you hand it off to someone else
- Each issue has a number (e.g. 420) so you can name your branch after your issue to connect the two (The `my-project` part is an optional phrase to remind everyone what this project is about.)

```
git checkout -b iss420-my-project
```

- Some people have a general dev branch just to store other code they want to store usually named `<username>-dev` (e.g. `tomeichlersmith-dev`[3])

[2] Ask and I can show you how to do this.

[3] Doesn't actually exist

git submodules

submodule Documentation

- Isolates different codes from each other ✓
- Requires more `git` nonsense (like `git submodule update`)
- Requires a branch in each submodule involved if changes are made across many submodules

Basically

Think of a submodule as a totally separate repo. ldmx-sw **only** stores which commit of that repo it should put into that submodule (i.e. ldmx-sw only "references" the submodule).

```
# checkout trunk and the submodules pointed-to by trunk
ldmx checkout trunk
# checkout trunk but have Ecal be in developments
ldmx checkout trunk Ecal:developments
```

Code Formatting

Using Google Style Guide and `clang-format`.

Uses the `find` command which comes with `bash`

```
# need clang-format installed
find <directory> -type f -name "*.h" -o -name "*.cxx" \
  -exec clang-format -i --style=Google {} \;
```

Pull Request Workflow

Template

`ldmx-sw` has a PR Template that walks you through the three big steps.

- Explain what you changed (and what it is supposed to fix and/or improve)
- Confirm that your changes don't break anything else (using the tests)
- Prove that your changes do what you say they do (depends on the changes)

Submodules

Since submodules are separate repositories, you will need separate PRs if you have changes in the submodule: One in the submodule for the changes and one in ldmx-sw to update the submodule reference.

Questions