# Innovation Lab (ILAB) Container Primer & Best Practices

Table of Contents

[1. Innovation Lab (ILAB) Container Primer & Best Practices 1](#_Toc70947781)

[1.1 Overview 3](#_Toc70947782)

[2.1 Container Lifecycle 3](#_Toc70947783)

[2.1.1 Container Architecture 4](#_Toc70947784)

[2.1.2 Container Naming Conventions 5](#_Toc70947785)

[2.1.3 Container Inventory 6](#_Toc70947786)

[3. Container Design & Testing Strategy [CB & Developer] 8](#_Toc70947787)

[4. Container Build & Deployment Strategy [CB] 9](#_Toc70947788)

[5. Container Configuration Management Strategy [CB] 10](#_Toc70947789)

[6. Frequently Asked Questions (FAQs) 13](#_Toc70947790)

[6.1 Container Builder FAQs 13](#_Toc70947791)

[6.2 Container User FAQs 14](#_Toc70947792)

[6.3 Container Developer FAQs 16](#_Toc70947793)

[Appendix A. Snapshot of Current ILAB Shared Repository 19](#_Toc70947794)

[Appendix B. Example Container CM Workflows for Developers 20](#_Toc70947795)

[Appendix C. Creating Tags 22](#_Toc70947796)

[Appendix D. ILAB Software Release & Container Build Example (Core) 25](#_Toc70947797)

[Appendix E. Application-Specific Software Release & Container Build (Modis-water) 27](#_Toc70947798)

[Appendix F. Application-Specific Notebook Example (EIS CFM) 28](#_Toc70947799)

[Appendix G. Notebook Publication Process (EIS IMERG) 30](#_Toc70947800)

|  | | | |
| --- | --- | --- | --- |
| **Version** | **Changed By:** | **Date**  **(MM/DD/YYYY)** | **Comments** |
| 1.0 | Glenn Tamkin | 03/11/2021 | Initial Draft |
| 2.0 | Glenn Tamkin | 04/19/2021 | Added Application-Specific Build Appendix |
| 3.0 | Caleb Spradling | 05/02/2021 | Added Notebook Publication Process Appendix |

## 1.1 Overview

This document describes the current practices and policies for containers as instituted by the Innovation Lab (ILAB). The goal is to enable the ILAB development staff to identify, design, and reuse them. As containers proliferate, it has become clear that we need a clear set of heuristics for managing and extending the ILAB functionality suite. Note that the “as is” state is represented here, and there are acknowledged differences from the proposed NCCS Container Primer. The resolution of such differences is TBD.

The containers that are described herein are hosted on the ADAPT platform. They are accessed via a suite of virtual machines (VMs) on ADAPT and are typically referred to by names such as dsg101 and dsg103. This specific collection of VMs is part of the ILAB container workflow, but this suite will continue to evolve moving forward. These 64-bit VMs consist of ten CPUs with Intel Core Processors (Skylake) and 118G RAM. Singularity is also installed and required to run these containers on the ADAPT VMs. This VM configuration is typical for ADAPT, but not required to execute the ILAB containers. Any platform capable of hosting Singularity can host ILAB containers.

## 2.1 Container Lifecycle

ILAB containers are built according to the strategy defined in this document. In short, Singularity[[1]](#footnote-1) is used to build a container from a ***definition file[[2]](#footnote-2)*** at discrete points in time. Builds do not occur automatically, they are event-driven and performed manually by a human. The event that triggers a container build is the Software Task Template (STT) submission. Once built, containers are read-only and never modified. If any change is required to the container, another definition file is created along with a subsequent image. Only definition files are placed under configuration management (CM), not the images themselves since these large objects are reliably reproducible from the definition file. Containers are named the same as the definition file, except for the suffix[[3]](#footnote-3), which is .***sif***.

[iluser@dsg101 containers]$ ls -alt ilab-core-3.0.0\*

lrwxrwxrwx 1 iluser ilab 116 May 5 2020 ilab-core-3.0.0.def

-rwxr-xr-x 1 iluser ilab 994127872 May 5 2020 ilab-core-3.0.0.sif

After being built, all containers are placed in a single directory for access by development staff and end users:

[iluser@dsg101 containers]$ cd **/att/nobackup/iluser/containers**

[iluser@dsg101 containers]$ ls -alt \*.sif

-rwxr-xr-x 1 iluser ilab 1375834112 Feb 23 11:34 ilab-nepac-2.0.0.sif

-rwxr-xr-x 1 iluser ilab 1366532096 Feb 22 17:24 ilab-nepac-1.0.0.sif

-rwxr-xr-x 1 iluser ilab 1364156416 Feb 22 10:55 ilab-core-7.0.0.sif

-rwxr-xr-x 1 iluser ilab 2278514688 Feb 1 09:36 ilab-stereo-pipeline-ubuntu-1.0.0.sif

-rwxr-xr-x 1 iluser ilab 1957322752 Jan 26 18:45 ilab-core-centos-2.0.0.sif

-rwxr-xr-x 1 iluser ilab 1395322880 Jan 21 16:54 cisto-centos-singularity-gdal-3.0.0.sif

-rwxr-xr-x 1 iluser ilab 1395318784 Jan 21 15:47 cisto-centos-singularity-gdal-2.0.0.sif

-rwxr-xr-x 1 iluser ilab 361639936 Jan 6 16:22 ilab-stereo-pipeline-3.0.0.sif

-rwxr-xr-x 1 iluser ilab 361492480 Jan 6 14:11 ilab-stereo-pipeline-2.0.0.sif

-rwxr-xr-x 1 iluser ilab 358543360 Jan 6 12:53 ilab-stereo-pipeline-1.0.0.sif

. . . . . .

When a new version of an existing container is created, the previous image is deprecated. However, it does not disappear. Instead, it is moved to the deprecated directory (/att/nobackup/iluser/containers/deprecated) to encourage users to begin accessing updated containers.

[iluser@dsg101 containers]$ cd /att/nobackup/iluser/containers

[iluser@dsg101 containers]$ ls -alt ilab-cb\*.simg

-rwxr-xr-x 1 iluser ilab 12020183040 Jun 8 2020 ilab-cb-6.0.0.simg

[iluser@dsg101 containers]$ ls -alt deprecated/ilab-cb\*.simg

-rwxr-xr-x 1 iluser ilab 12011393024 Jun 5 2020 bak/ilab-cb-5.0.0.simg

-rwxr-xr-x 1 iluser ilab 12009635840 Jun 4 2020 bak/ilab-cb-4.0.3.simg

-rwxr-xr-x 1 iluser ilab 12007878656 Jun 4 2020 bak/ilab-cb-4.0.2.simg

-rwxr-xr-x 1 iluser ilab 12006072320 Jun 3 2020 bak/ilab-cb-4.0.1.simg

-rwxr-xr-x 1 iluser ilab 12006068224 Jun 3 2020 bak/ilab-cb-4.0.0.simg

-rwxr-xr-x 1 iluser ilab 12006068224 Jun 2 2020 bak/ilab-cb-3.0.1.simg

-rwxr-xr-x 1 iluser ilab 12004745216 Jun 2 2020 bak/ilab-cb-2.0.0.simg

-rwxr-xr-x 1 iluser ilab 12004745216 May 28 2020 bak/ilab-cb-1.0.0.simg

-rwxr-xr-x 1 iluser ilab 12004749312 May 28 2020 bak/ilab-cb-1.1.0.simg

### 2.1.1 Container Architecture

An explicit ILAB goal is to create reusable containers that guarantee identical environments for R&D, development, testing, and deployment. Specifically, this approach involves designing a layered set of containers that extend one another to support targeted applications. ILAB applications are defined by three tiers as shown in Figure 1. Logical Container Hierarchy: 1). General purpose Python scientific stack, 2) ILAB shared resources, and 3). Application-specific resources. The functionality of each tier, which is inherited moving from left to right, ends with narrow resources that are typically only needed for the application being built.



### 2.1.2 Container Naming Conventions

Containers are named according to these prefixes as illustrated in Figure 2:

* cisto-data-science – contains general-purpose Python scientific stack (e.g., scikit, dask, numpy, etc.)
* ilab-core – extends (a) with ilab-specific tools (e.g., GDAL, Celery, Redis)
* ilab-<*application name*> - extends (a+b) with application-specific artifacts

For example, ilab-cb-1.0.0.simg is created specifically for use by the Chesapeake Bay Water Quality (CBWQ) application users. It contains CBWQ dependencies such as Seadas and OCSSW. However, since it extends the ilab-core-4.1.0.simg container, it also inherits Celery and GDAL. Similarly, since ilab-core-4.1.0.simg inherits from cisto-data-science-3.0.0.simg, the entire Python ecosystem is also accessible. Each container definition file[[4]](#footnote-4) begins by specifying its parent, which can be a 3rd party image or an existing ILAB container. In the example below, the container is inheriting from a local image:

Bootstrap: localimage

FROM: ./cisto-data-science-3.0.0.simg

Note that this inheritance hierarchy repeats with all of ILAB application-specific containers.

### 2.1.3 Container Inventory

When one parent container definition is updated (i.e., python -m pip install --upgrade scikit-learn==0.24.0) the remaining children must also be rebuilt in order to inherit the parent updates. As mentioned above, this is not an automatic[[5]](#footnote-5) build cycle, but is triggered via STT. This does not mean that the existing containers with older dependencies are broken, they are simply deprecated.

Regarding version numbers, application-specific containers are suffixed with the Git tag that was generated by the developer during the corresponding software release. General-purpose containers are suffixed according to this convention: <container-name>-<**X**>-<**Y**>-<**Z**>.sif. Initial versions start at <container-name>-1.0.0.sif. Subsequent containers are incremented as follows:

* **X** – major new feature
* **Y** – dependency change
* **Z** – bug fix

For convenience, a snapshot of the current ILAB container inventory is shown in Appendix A. Snapshot of Current ILAB Shared RepositoryAppendix. All container definition files are co-located in this single repository along with build details: <https://github.com/nasa-nccs-hpda/containers>.

*Based on feedback from the review of this primer, the ILAB is now applying changes to this CM process*. In summary, general-purpose container definition files will remain in the shared repository. However, new application-specific container definition files will now reside in the respective application-specific repositories. Existing container definition files will be migrated to the application-specific repositories when they are next touched in the ILAB software development process. Figure 4, which illustrates the new layout of the ILAB repositories and contents, summarizes the new rules for container management. Note that the concept of *development* containers now formalized (e.g., “-dev”). Previously, development containers were identified by sequential version numbers. Moving forward, they will contain the corresponding branch name in order to maintain specific mapping to the software repository.



Figure 4. Containers and Repositories

# 3. Container Design & Testing Strategy [CB & Developer]

1. Design, build, and test a defined science application (developer)
2. Identify application dependencies (developer)
3. Define API and application use (developer)
4. Create a benchmark that includes a run-time script, nominal parameters, and specific results (developer)
5. Determine if a container already exists for this new application (developer). See 6.3 Container Developer FAQs for dependency identification strategies.
6. Document items #1-4 in a Software Task Template (STT)
7. Place the above under CM in a git project (developer). See Appendix B. Example Container CM Workflows for Developers
8. Retrieve the STT from git (container-builder [CB])
9. Create a Singularity container definition file (CB) that extends the IL stack (data-science, core, apps, jupyter) on dsg101[[6]](#footnote-6)
10. Build the container from the definition file as sudo (CB) on dsg101
11. Test the container against the benchmark (CB)
12. Independently verify the application via container (developer) on dsg103
13. Independently verify the application via container (tester)
14. Revisit IL stack with lessons learned (i.e., import new dependencies into IL core containers) (CB) after each new container is verified.

# 4. Container Build & Deployment Strategy [CB]

1. Container builds are initiated via STT.
2. Singularity builds require sudo. As such, we’ve designated a shared user called iluser for this role to eliminate dependencies on any single user account. This access is granted on VM dsg101 only to specific users. Note that migration to ilab101-104 has begun. Right now, the CB logs into this VM and runs the build like below[[7]](#footnote-7):

gtamkin@GSSLA18081061:~$ ssh -XY adaptlogin.nccs.nasa.gov

gtamkin@adaptlogin102:~$ ssh -XY dsg101

gtamkin@dsg101:~$ sudo /bin/su – iluser

gtamkin@dsg101:~$ cd /att/nobackup/iluser/containers

[iluser@dsg101 containers]$ time /usr/bin/sudo -E SINGULARITY\_NOHTTPS=1 /bin/singularity build /att/nobackup/iluser/containers/ilab/ilab-cisto-data-science-3.0.0.sif /att/nobackup/iluser/containers/cisto-data-science-3.0.0.def

1. Containers are suffixed with .sif. **Note that some legacy containers are suffixed with .simg, but the .sif convention is now being used for alignment with Singularity best practices.**
2. All active containers are deployed here: /att/nobackup/iluser/containers/
3. Deprecated containers are stored here: /att/nobackup/iluser/containers/deprecated
   1. **Once built, a container is never changed or deleted.**  When an updated version is created, the older one becomes deprecated and is moved to the backup directory (/att/nobackup/iluser/containers/deprecated). Note that it can still be accessed and used as it always was; however, it is moved to encourage acceptance of the latest container version.

# 5. Container Configuration Management Strategy [CB]

1. Place the Singularity specification files in the existing git/innovation-lab repository:
   1. <https://github.com/nasa-nccs-hpda/containers>
2. When a dependency is required in more than one application, add it to the core when triggered via STT.
3. Create new versions of definition files as necessary (i.e., cisto-data-science-**1.2.0**)
4. Rebuild versioned container hierarchy when triggered via STT
5. Notionally, the ILAB hosts containers in two spots, 1) the ***active*** directory (/att/nobackup/iluser/containers), and 2) the ***deprecated*** (/att/nobackup/iluser/containers/deprecated) directory. The active directory contains the latest versions of the containers while the deprecated directory contains outdated container versions. The deprecated containers remain available for benchmark testing by users migrating across container versions.
6. ***EVERY*** container is generated from one and only one associated .def file. Once published (i.e., a container is built and deployed to the ***active*** directory), the related .def files are placed under configuration management (CM) in the repository noted in #1. These definition files are ***NEVER*** changed or removed. Instead, new versions with updated version numbers are created if necessary. It follows then that deprecated specification files always remain in the repository.
7. The union of the ***active*** and ***deprecated*** directory containers should always match the superset of .def files under CM.
8. ***NEW****:* Add a ‘%test’ section to each .def file moving forward. Recently, we did this for particular container for a quick reference to certain dependencies and should now adopt it for all future containers. This section should have contents that are helpful for the application itself, but must also contain basic information like versions for OS, Python, GDAL, Singularity, etc. An example looks like this (.def code followed by command line invocation):

%test

echo 'General System Information:'

uname -a

echo 'Linux Version:'

cat /etc/os-release

echo 'gdalinfo --formats | grep -i jpeg'

gdalinfo --formats | grep -i jpeg

echo 'gdalinfo --formats | grep -i hdf'

gdalinfo --formats | grep -i hdf

echo 'ogrinfo --formats | grep GDB'

ogrinfo --formats | grep GDB

echo 'Python version:'

python -V

echo 'GDAL version:'

python -c 'from osgeo import gdal; print(gdal.\_\_version\_\_)'

echo 'Java version:'

java --version

echo 'Singularity version:'

singularity --version

echo ‘Application-specific Information’

echo 'Verify ASP install'

dg\_mosaic -h

gtamkin@dsg103:/tmp/containers$ singularity test /att/nobackup/iluser/containers/ilab-stereo-pipeline-ubuntu-1.0.0.sif

System Information:

Linux dsg103 4.9.155 #1 SMP Thu Feb 7 09:58:05 EST 2019 x86\_64 x86\_64 x86\_64 GNU/Linux

Linux Version:

NAME="Ubuntu"

VERSION="20.04.1 LTS (Focal Fossa)"

ID=ubuntu

ID\_LIKE=debian

PRETTY\_NAME="Ubuntu 20.04.1 LTS"

VERSION\_ID="20.04"

HOME\_URL="https://www.ubuntu.com/"

SUPPORT\_URL="https://help.ubuntu.com/"

BUG\_REPORT\_URL="https://bugs.launchpad.net/ubuntu/"

PRIVACY\_POLICY\_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"

VERSION\_CODENAME=focal

UBUNTU\_CODENAME=focal

gdalinfo --formats | grep -i jpeg

JPEG -raster- (rwv): JPEG JFIF

JP2OpenJPEG -raster,vector- (rwv): JPEG-2000 driver based on OpenJPEG library

JPEGLS -raster- (rwv): JPEGLS

gdalinfo --formats | grep -i hdf

HDF4 -raster,multidimensional raster- (ros): Hierarchical Data Format Release 4

HDF4Image -raster- (rw+): HDF4 Dataset

HDF5 -raster,multidimensional raster- (rovs): Hierarchical Data Format Release 5

HDF5Image -raster- (rov): HDF5 Dataset

ogrinfo --formats | grep GDB

OpenFileGDB -vector- (rov): ESRI FileGDB

Python version:

Python 3.8.5

GDAL version:

3.3.0dev-28306791ede238f0076ae83c919d245a92907d6f

Java version:

openjdk 11.0.9.1 2020-11-04

OpenJDK Runtime Environment (build 11.0.9.1+1-Ubuntu-0ubuntu1.20.04)

OpenJDK 64-Bit Server VM (build 11.0.9.1+1-Ubuntu-0ubuntu1.20.04, mixed mode, sharing)

Singularity version:

singularity version 3.7.0

Verify ASP install

[ASP 2.7.0]

# 6. Frequently Asked Questions (FAQs)

## 6.1 Container Builder FAQs

1. Who builds the containers?
   1. The container builder [CB]. Right now, Glenn is primary with Savannah being trained as replacement.
2. Where do the containers live?
   1. Active containers are hosted on ADAPT in this directory: /att/nobackup/iluser/containers.
   2. Deprecated containers are located here: /att/nobackup/iluser/containers/deprecated.
3. How are containers built?
   1. See Container Build & Deployment Strategy [CB]
4. When are containers updated?
   1. Never. New container versions are created only when triggered via STT. Motivations for newer container versions are described here 2.1.2 Container Versions
5. What is the general container lifecycle?
   1. Containers never go away, they just migrate. See 4. Container Build & Deployment Strategy [CB] for details.

## 6.2 Container User FAQs

1. Where are the ILAB containers?
   1. Active containers are hosted on ADAPT in this directory: /att/nobackup/iluser/containers.
   2. Deprecated containers are located here: /att/nobackup/iluser/containers/deprecated.
2. How do I use them?
   1. After logging into dsg103, containers are typically used in two ways: I) run from the command line, or II) shelled into for manual interaction. Examples of both are provided here:
      1. Check GDAL version by running Python from the command line:

gtamkin@dsg103:/att/nobackup/iluser/containers$ singularity **run** ilab-nepac-2.0.0.sif python -c 'from osgeo import gdal; print(gdal.\_\_version\_\_)'

2.1.2

* + 1. Check GDAL version by running Python in the container shell:

gtamkin@dsg103:/att/nobackup/iluser/containers$ singularity **shell** ilab-nepac-2.0.0.sif

Singularity> python -V

Python 3.7.4

Singularity> python

Python 3.7.4 (default, May 12 2020, 15:01:34)

[GCC 6.3.0 20170516] on linux

Type "help", "copyright", "credits" or "license" for more information.

>>> from osgeo import gdal

>>> print(gdal.\_\_version\_\_)

2.1.2

* 1. See <https://sylabs.io/guides/3.0/user-guide/quick_start.html#interact-with-images> for full usage options.

1. How long do ILAB containers last?
   1. Essentially forever unless they are explicitly *removed* via STT. If they are simply upgraded, the new one is added and the old one is migrated to the deprecation directory (/att/nobackup/iluser/containers/deprecated).
2. Are containers patched for security reasons?
   1. No. Builds are triggered via STT.
3. What if my container disappeared?
   1. First check the deprecation directory (/att/nobackup/iluser/containers/deprecated), since a newer version may have been created. If it cannot be located, contact the CB.

## 6.3 Container Developer FAQs

1. Where do I start when I have a new task/application?
   1. Create a new repository to house the application-specific definition file and other optional artifacts.
   2. Shell into the latest ilab-core-x.x.x.sif[[8]](#footnote-8)container.
   3. Run your application and identify what, if anything, is missing.
   4. Apply any additional dependencies to the new application-specific definition file
   5. In the example below, the dependency package called netCDF4 is desired but not found in core. Prepare and submit STT to CB.

gtamkin@dsg101:/att/nobackup/iluser/containers$ singularity shell ilab-core-7.0.0.sif

Singularity> python

Python 3.7.4 (default, May 12 2020, 15:01:34)

[GCC 6.3.0 20170516] on linux

Type "help", "copyright", "credits" or "license" for more information.

>>> from netCDF4 import Dataset

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

ModuleNotFoundError: No module named 'netCDF4'

1. Which container, if any, has my dependency?
   1. Clone the ILAB repositories locally.
   2. Search the existing definition files for your dependency: $ grep <dependency> \*.def.
   3. If none have the dependency, prepare and submit a STT to CB.
2. What if another application-specific container has my dependency?
   1. Prepare and submit a STT to CB noting which container(s) have the dependency. **Best practices suggest that each application maintain its own lifecycle rather than sharing application-specific containers**. While the new container is scheduled, proceed testing the latest container that has this dependency until the new one is ready. Report additional discrepancies to the CB as soon as possible to avoid unnecessary build iterations. When the new one is ready, repeat the tests on it. In the example below, the netCDF4 dependency is located in the CBWQ application-specific container (ilab-cb-6.0.0.def).

gtamkin@dsg103:/tmp$ git clone https://github.com/nasa-nccs-hpda/containers.git

Cloning into 'containers'...

remote: Enumerating objects: 206, done.

remote: Counting objects: 100% (206/206), done.

remote: Compressing objects: 100% (154/154), done.

remote: Total 206 (delta 92), reused 160 (delta 50), pack-reused 0

Receiving objects: 100% (206/206), 1.32 MiB | 0 bytes/s, done.

Resolving deltas: 100% (92/92), done.

gtamkin@dsg103:/tmp$ cd containers/

gtamkin@dsg103:/tmp/containers$ ls -alt \*.def

-rw-r--r-- 1 gtamkin k3000 491 Feb 24 12:35 ilab-nepac-2.0.0.def

-rw-r--r-- 1 gtamkin k3000 6155 Feb 24 12:35 ilab-otb-gpu-1.0.0.def

. . . . . . . .

gtamkin@dsg103:/tmp/containers$ grep netCDF4 \*.def

ilab-cb-6.0.0.def: python -m pip install --upgrade netCDF4==1.5.3

gtamkin@dsg103:/tmp/containers$

1. What if no container has my dependency?
   1. Prepare and submit the STT to CB. The initial version of the application-specific container will be built.
2. What if a new dependency is discovered after the initial container is built during testing?
   1. Update and submit the STT to CB. A new version of the container will be created.
3. When multiple containers have my dependency, which one do I use?
   1. If ilab-core-x.x.x.sif has your dependency, use the latest version for testing.
   2. If multiple application-specific containers have your dependency, prepare and submit the STT to CB noting the other containers. In the meantime, we encourage testing with one of these containers. Compare the related .def files. Select the one with the latest version of the dependency. If all else is equal, select the container with the dependencies that are most likely aligned with your application moving forward.
4. What happens when it is discovered that multiple applications need the same new dependency?
   1. CB schedules creation of an updated ilab-core-x.x.x.sif version. Dependent *children* (i.e., application-specific containers) of *core* will only be rebuilt if a STT triggers it. This happens, for example, when it is known that a child container needs an updated version of a shared dependency in *core*.
5. What’s in any specific container?
   1. The most accurate way of determining the contents and configuration of a container is by simply reading the corresponding .def file. Specifically, the FROM command states which base container this one is derived from. The python pip install declarations list the package dependencies and versions.
   2. Moving forward we will begin implementing a Singularity test command that interrogates each container. See 5.8
6. Can I add a package for development testing without iterating with the CB?
   1. Yes, and this is highly encouraged. One way to do this is to modify your $PATH or $PYTHONPATH to point to something that exists outside of the container. You could also specify runtime parameters to expose the file system to the container. In the example below, the ‘-B’ parameter tells Singularity that it has access to the path that follows it on the file system. Of course, this path could point to an executable, configuration file, directory, etc. to temporarily expand your containers default capabilities.

gtamkin@dsg103:/tmp/containers$ singularity **run** ***-B /att/nobackup/iluser/containers/deprecated*** /att/nobackup/iluser/containers/ilab-stereo-pipeline-ubuntu-1.0.0.sif ls -alt /att/nobackup/iluser/containers/deprecated/\*core\*.si\*

-rwxrwxr-x 1 60021 ilab 1004311483 Apr 23 2020 /att/nobackup/iluser/containers/deprecated/remote-ilab-core-2.0.0.simg

-rwxr-xr-x 1 60021 ilab 1003909120 Apr 17 2020 /att/nobackup/iluser/containers/deprecated/ilab-core-2.0.0.simg

-rwxrwxr-x 1 60021 ilab 1003683840 Mar 28 2020 /att/nobackup/iluser/containers/deprecated/ilab-core-1.0.0.simg

* 1. This approach is very helpful for development, but there is no magic ☹. When the dust settles, we cannot deploy this ‘situation’ to a remote user. So, you will still have to document the specific environment modifications in the STT and go through the process. The ILAB ultimately needs to create a deployable Singularity container as per ILAB policy.

# Appendix A. Snapshot of Current ILAB Shared Repository

Application, table

Description automatically generated with medium confidenceTable

Description automatically generated with medium confidence

# Appendix B. Example Container CM Workflows for Developers





# Appendix C. Creating Tags

Create a tag when the current version of the software must be preserved. Sample session below:

*Moral of the story – branches must be tagged before a related container is built. That goes for master, develop, imagfile-readonly, etc. ALL OF THEM. Then, the container is suffixed by that tag name.*

*Notes: When you pull by branch you only get tags for that branch. When you pull by tag, you get all tags in the repo, but you are automatically switched to the tagged branch. So, CM should pull by tag for container builds.*

Pull by branch:

centos@ip-172-31-54-47:/tmp/git$ rm -rf \*

centos@ip-172-31-54-47:/tmp/git$ mkdir core

centos@ip-172-31-54-47:/tmp/git$ cd core

centos@ip-172-31-54-47:/tmp/git/core$ mkdir master

centos@ip-172-31-54-47:/tmp/git/core$ mkdir develop

centos@ip-172-31-54-47:/tmp/git/core$ mkdir image-readonly

centos@ip-172-31-54-47:/tmp/git/core$ **git clone --single-branch --branch master https://github.com/nasa-nccs-hpda/core.git ./master**

centos@ip-172-31-54-47:/tmp/git/core$ **git clone --single-branch --branch develop https://github.com/nasa-nccs-hpda/core.git ./develop**

centos@ip-172-31-54-47:/tmp/git/core$ **git clone --single-branch --branch imagefile-readonly https://github.com/nasa-nccs-hpda/core.git ./imagefile-readonly**

Create tag:

centos@ip-172-31-54-47:/tmp/git/core$ cd image-readonly

centos@ip-172-31-54-47:/tmp/git/core/imagefile-readonly$ **git tag core-image-readonly-1.0.0 -m "Initial tag after creating temporary branch for core"**

centos@ip-172-31-54-47:/tmp/git/core/imagefile-readonly$ **git tag -l -n3**

core-image-readonly-1.0.0 Initial tag after creating temporary branch for core

centos@ip-172-31-54-47:/tmp/git/core/imagefile-readonly$ **git push origin core-image-readonly-1.0.0**

Username for 'https://github.com': gtamkin

Password for 'https://gtamkin@github.com':

Counting objects: 1, done.

Writing objects: 100% (1/1), 193 bytes | 0 bytes/s, done.

Total 1 (delta 0), reused 0 (delta 0)

To https://github.com/nasa-nccs-hpda/core.git

\* [new tag] core-image-readonly-1.0.0 -> core-image-readonly-1.0.0

Pull by tag:

centos@ip-172-31-54-47:/tmp/git$ mkdir tag

centos@ip-172-31-54-47:/tmp/git$ cd tag

centos@ip-172-31-54-47:/tmp/git/tag$ mkdir image-readonly-1.0.0

centos@ip-172-31-54-47:/tmp/git/tag$ **git clone** [**https://github.com/nasa-nccs-hpda/core.git -b core-image-readonly-1.0.0 ./image-readonly-1.0**](https://github.com/nasa-nccs-hpda/core.git%20-b%20core-image-readonly-1.0.0%20./image-readonly-1.0)**.0**

centos@ip-172-31-54-47:/tmp/git/tag/image-readonly-1.0.0$ **git tag**

core-1.0.0

core-develop-1.0.0

core-image-readonly-1.0.0

ilab-core-2.0.0

Delete tag:

centos@ip-172-31-54-47:/tmp/git/tag/image-readonly-1.0.0$ **git tag**

core-1.0.0

core-develop-1.0.0

core-image-readonly-1.0.0

ilab-core-2.0.0

centos@ip-172-31-54-47:/tmp/git/tag/image-readonly-1.0.0$ **git tag -d ilab-core-2.0.0**

Deleted tag 'ilab-core-2.0.0' (was af4c436)

centos@ip-172-31-54-47:/tmp/git/tag/image-readonly-1.0.0$ **git tag**

core-1.0.0

core-develop-1.0.0

core-image-readonly-1.0.0

Build container by tag:

centos@ip-172-31-54-47:/tmp/git/core$ mkdir master-1.0.0

centos@ip-172-31-54-47:/tmp/git/core$ cd master-1.0.0/

centos@ip-172-31-54-47:/tmp/git/core$ **git clone https://github.com/nasa-nccs-hpda/core.git -b core-1.0.0 master-1.0.0/**

Cloning into 'master-1.0.0'...

remote: Enumerating objects: 188, done.

remote: Counting objects: 100% (188/188), done.

remote: Compressing objects: 100% (119/119), done.

remote: Total 188 (delta 93), reused 155 (delta 60), pack-reused 0

Receiving objects: 100% (188/188), 289.34 KiB | 0 bytes/s, done.

Resolving deltas: 100% (93/93), done.

Note: checking out '766fb93d250fc3b680f9163075adad357e3efea4'.

You are in 'detached HEAD' state. You can look around, make experimental

changes and commit them, and you can discard any commits you make in this

state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may

do so (now or later) by using -b with the checkout command again. Example:

git checkout -b new\_branch\_name

centos@ip-172-31-54-47:/tmp/git/core$ cd master-1.0.0/container/

centos@ip-172-31-54-47:/tmp/git/core/master-1.0.0/container$ ls

cisto-data-science.def cisto-gdal.def core.def

centos@ip-172-31-54-47:/tmp/git/core/master-1.0.0/container$ cd ~/containers

centos@ip-172-31-54-47:~/containers/refactor$ **time /usr/bin/sudo -E SINGULARITY\_NOHTTPS=1 /usr/local/bin/singularity build /home/centos/containers/core-1.1.0.sif /tmp/git/core/master-1.0.0/container/core.def**

INFO: Starting build...

. . . . .

WARNING: Help message already exists and force option is false, not overwriting

INFO: Adding labels

WARNING: Label: Author already exists and force option is false, not overwriting

WARNING: Label: Version already exists and force option is false, not overwriting

INFO: Adding environment to container

INFO: Creating SIF file...

INFO: Build complete: /home/centos/containers/core-1.1.0.sif

real 5m14.041s

user 14m4.983s

sys 0m51.798s

# Appendix D. ILAB Software Release & Container Build Example (Core)

Create a tag when the core of the software or notebook must be preserved. Sample session below, after checking out master branch:

centos@ip-172-31-54-47:~/projects/core$ **cd /home/centos/projects/core**

centos@ip-172-31-54-47:~/projects/core$ **git clone https://github.com/nasa-nccs-hpda/core.git -b master**

centos@ip-172-31-54-47:~/projects/core$ **git tag -l -n3**

core-1.0.0 Initial tag after restructuring master branch for application-specific projects

core-develop-1.0.0 Refactored to make ubuntu default OS in core thread

core-footprints-query-1.0.0 Initial tag for footprints

core-image-readonly-1.0.0 Initial tag after creating temporary branch for core

centos@ip-172-31-54-47:~/projects/core$ git tag

*core-1.0.0*

*core-develop-1.0.0*

*core-footprints-query-1.0.0*

*core-image-readonly-1.0.0*

# Create tag to match master branch, build container with version only to keep the peace.

centos@ip-172-31-54-47:~/projects/core$ **git tag core-master-2.0.0 -m "Release for imagefile-read-only"**

centos@ip-172-31-54-47:~/projects/core$ **git push origin core-master-2.0.0**

Username for 'https://github.com': gtamkin

Password for 'https://gtamkin@github.com':

Counting objects: 1, done.

Writing objects: 100% (1/1), 177 bytes | 0 bytes/s, done.

Total 1 (delta 0), reused 0 (delta 0)

To https://github.com/nasa-nccs-hpda/core.git

\* [new tag] core-master-2.0.0 -> core-master-2.0.0

centos@ip-172-31-54-47:~/projects/core$ **git tag -l -n3**

*core-1.0.0 Initial tag after restructuring master branch for application-specific projects*

*core-develop-1.0.0 Refactored to make ubuntu default OS in core thread*

*core-footprints-query-1.0.0 Initial tag for footprints*

*core-image-readonly-1.0.0 Initial tag after creating temporary branch for core*

***core-master-2.0.0 Release for imagefile-read-only***

# Build out hierarchy

centos@ip-172-31-54-47:~/containers$ **cd /home/centos/containers**

centos@ip-172-31-54-47:~/containers$ **ls -alt /home/centos/projects/core/container/\*.def**

-rw-r--r--. 1 centos centos 3144 Apr 22 19:45 /home/centos/projects/core/container/core-dev.def

-rw-r--r--. 1 centos centos 3150 Apr 22 19:21

-rw-r--r--. 1 centos centos 2550 Apr 22 19:21 /home/centos/projects/core/container/cisto-data-science-2.0.0.def

-rw-r--r--. 1 centos centos 1093 Apr 21 19:03 /home/centos/projects/core/container/cisto-gdal-2.0.0.def

centos@ip-172-31-54-47:~/containers$ **time /usr/bin/sudo -E SINGULARITY\_NOHTTPS=1 /usr/local/bin/singularity build /home/centos/containers/cisto-gdal-2.0.0.sif /home/centos/projects/core/container/cisto-gdal-2.0.0.def**

centos@ip-172-31-54-47:~/containers$ **time /usr/bin/sudo -E SINGULARITY\_NOHTTPS=1 /usr/local/bin/singularity build /home/centos/containers/cisto-data-science-2.0.0.sif** /home/centos/projects/core/container/cisto-data-science-2.0.0.def

centos@ip-172-31-54-47:~/containers$ **time /usr/bin/sudo -E SINGULARITY\_NOHTTPS=1 /usr/local/bin/singularity build /home/centos/containers/core-2.0.0.sif /home/centos/projects/core/container/core-2.0.0.def**

**/home/centos/containers**

**centos@ip-172-31-54-47:~/containers$ ls -alt \*2.0.0.sif**

*-rwxr-xr-x. 1 centos centos 2025730048 Apr 22 19:51 core-2.0.0.sif*

*-rwxr-xr-x. 1 centos centos 1637158912 Apr 22 19:44 cisto-data-science-2.0.0.sif*

*-rwxr-xr-x. 1 centos centos 975925248 Apr 22 19:26 cisto-gdal-2.0.0.sif*

-rwxr-xr-x. 1 centos centos 1567653888 Apr 14 18:33 ilab-cfm-2.0.0.sif

-rwxr-xr-x. 1 centos centos 1375834112 Feb 23 16:31 ilab-nepac-2.0.0.sif

-rwxr-xr-x. 1 centos centos 1957322752 Jan 26 23:31 ilab-core-centos-2.0.0.sif

-rwxr-xr-x. 1 centos centos 1395318784 Jan 21 19:40 cisto-centos-singularity-gdal-2.0.0.sif

-rwxr-xr-x. 1 centos centos 361492480 Jan 6 19:08 ilab-stereo-pipeline-2.0.0.sif

**~~# save versioned copy for sanity and then push to adapt as core.sif~~**

~~centos@ip-172-31-54-47:~/containers$~~ **~~cp core.sif versioned/core-master-2.0.0.sif~~**

# Appendix E. Application-Specific Software Release & Container Build (Modis-water)

# Implement, check in, and tag container

centos@ip-172-31-54-47:~/projects/modis\_water$ **git clone --single-branch --branch water\_detection https://github.com/nasa-nccs-hpda/modis\_water.git**

centos@ip-172-31-54-47:~/projects/modis\_water$ **more container/modis\_water-dev.def**

Bootstrap: localimage

FROM: ./core-2.0.0.sif

%help

===========================================================================

- modis\_water (extends -> core) – ILAB dependencies:

===========================================================================

%environment

# set PYTHONPATH for access to modis\_water code

export PYTHONPATH="$PYTHONPATH:/usr/local/water\_detection"

%post

# retrieve modis\_water source from git repository and open permissions

mkdir -p /usr/local/modis\_water

git clone --single-branch --branch water\_detection https://github.com/nasa-nccs-hpda/modis\_water.git /usr/local/water\_detection

chmod a+rwx -R /usr/local/water\_detection

centos@ip-172-31-54-47:~/projects/modis\_water$ **git tag**

centos@ip-172-31-54-47:~/projects/modis\_water$ **git tag modis\_water-water\_detection-1.0.0 -m "Release for water\_detection branch"**

centos@ip-172-31-54-47:~/projects/modis\_water$ **git push origin modis\_water-water\_detection-1.0.0**

Username for 'https://github.com': gtamkin

Password for 'https://gtamkin@github.com':

Counting objects: 1, done.

Writing objects: 100% (1/1), 185 bytes | 0 bytes/s, done.

Total 1 (delta 0), reused 0 (delta 0)

To https://github.com/nasa-nccs-hpda/modis\_water.git

\* [new tag] modis\_water-water\_detection-1.0.0 -> modis\_water-water\_detection-1.0.0

centos@ip-172-31-54-47:~/projects/modis\_water$ **git tag -l -n3**

**modis\_water-water\_detection-1.0.0 Release for water\_detection branch**

# Build container on SMCE and transfer to ADAPT

centos@ip-172-31-54-47:~/containers$ **cd /home/centos/containers**

centos@ip-172-31-54-47:~/containers$ **time /usr/bin/sudo -E SINGULARITY\_NOHTTPS=1 /usr/local/bin/singularity build /home/centos/containers/modis\_water-dev.sif /home/centos/projects/modis\_water/container/modis\_water-dev.def**

centos@ip-172-31-54-47:~/containers$ **scp modis\_water-dev.sif** [**gtamkin@troy.nccs.nasa.gov:~/test/xfer/containers**](mailto:gtamkin@troy.nccs.nasa.gov:~/test/xfer/containers)

gtamkin@adaptlogin102:/att/nobackup/gtamkin/containers$ **scp gtamkin@troy.nccs.nasa.gov:~/test/xfer/containers/\*water\* .**

[iluser@ilab101 containers]$ **cp /att/nobackup/gtamkin/containers/\*water\* .**

# Appendix F. Application-Specific Notebook Example (EIS CFM)

Create a tag when the current version of the software or notebook must be preserved. Sample session below:

Timeline

Description automatically generated with medium confidence

Create tag:

centos@ip-172-31-54-47:~$ cd ~/projects/eis

centos@ip-172-31-54-47:~/projects/eis**$ git tag eis-1.0.0 -m "Initial release supporting CFM Notebook delivery to EIS users"**

centos@ip-172-31-54-47:~/projects/eis$ **git tag -l**

eis-1.0.0

centos@ip-172-31-54-47:~/projects/eis$ **git push origin eis-1.0.0**

Username for 'https://github.com': gtamkin

Password for 'https://gtamkin@github.com':

Counting objects: 1, done.

Writing objects: 100% (1/1), 199 bytes | 0 bytes/s, done.

Total 1 (delta 0), reused 0 (delta 0)

To https://github.com/nasa-nccs-hpda/eis.git

\* [new tag] eis-1.0.0 -> eis-1.0.0

centos@ip-172-31-54-47:~/projects/eis$ git tag -l

eis-1.0.0

Pull by tag:

centos@ip-172-31-54-47://tmp/git/tag/eis

centos@ip-172-31-54-47:/tmp/git/tag/eis$ **git clone https://github.com/nasa-nccs-hpda/eis.git -b eis-1.0.0 ./eis-1.0.0**

Cloning into './eis-1.0.0'...

Username for 'https://github.com': gtamkin

Password for 'https://gtamkin@github.com':

remote: Enumerating objects: 282, done.

remote: Counting objects: 100% (282/282), done.

remote: Compressing objects: 100% (202/202), done.

remote: Total 282 (delta 62), reused 240 (delta 56), pack-reused 0

Receiving objects: 100% (282/282), 11.90 MiB | 0 bytes/s, done.

Note: checking out '04f2d030225843156b0e9002d94f394d9cb0380e'.

You are in 'detached HEAD' state. You can look around, make experimental

changes and commit them, and you can discard any commits you make in this

state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may

do so (now or later) by using -b with the checkout command again. Example:

git checkout -b new\_branch\_name

centos@ip-172-31-54-47:/tmp/git/tag/eis$

centos@ip-172-31-54-47:/tmp/git/tag/eis$ **ls -alt eis-1.0.0/sealevel/notebooks/ilab-cfm-notebook/**

total 52

drwxrwxr-x. 4 centos centos 149 Apr 19 10:30 .

drwxrwxr-x. 3 centos centos 49 Apr 19 10:30 ..

-rw-rw-r--. 1 centos centos 71 Apr 19 10:30 requirements.txt

-rw-rw-r--. 1 centos centos 1207 Apr 19 10:30 setup.py

drwxrwxr-x. 7 centos centos 165 Apr 19 10:30 docs

-rw-rw-r--. 1 centos centos 25014 Apr 19 10:30 changelog.md

drwxrwxr-x. 5 centos centos 4096 Apr 19 10:30 CFM\_main

-rw-rw-r--. 1 centos centos 25 Apr 19 10:30 \_config.yml

-rw-rw-r--. 1 centos centos 1070 Apr 19 10:30 LICENSE.txt

-rw-rw-r--. 1 centos centos 3588 Apr 19 10:30 README.md

# Appendix G. Notebook Publication Process (EIS IMERG)

1. Steps to notebook publication.
   1. Once notebook is finished being developed, we need to commit the NB into the correct ILAB GitHub repository (https://github.com/nasa-nccs-hpda).
      1. <https://github.com/nasa-nccs-hpda/eis>
   2. Follow the steps in Appendix F of the ILAB Container to commit a tag to the EIS repository with the new notebook.
      1. $ # this is the ILAB EIS repository
      2. $ cd projects/eis
      3. $ git tag eis-imergfwi-1.0.0 -m “Initial commit of the IMERG FWI \

notebook for EIS Fire users”

* + 1. $ git tag -l

eis-imergfwi-1.0.0

* + 1. $ git push origin eis-imergfwi-1.0.0
  1. See ilab-container-primer.docx Appendix F to see how to pull by the tag created above.
  2. Once approved by customer(?) and/or Mark, we can publish the notebook to the EIS-specific SMCE GitLab instance.
     1. Note: Publishing to the GitLab instance should be reserved for notebooks that are polished and customer ready.
  3. If this is a new notebook (not a part of an existing GitLab project), make a new project using the “New Project” button in the correct group (EIS Fire, Hydro, Sealevel). See screenshots below.Graphical user interface

     Description automatically generated



* 1. This will lead to a prompt of the type of project. Choose blank project.Graphical user interface, application

     Description automatically generated
  2. This will lead to another page which prompts you for more information. Be sure to put the correct name (the name of the tag you made without the version number), a brief description of the notebook, and visibility set to private (private always lets those in the group (Fire, Hydro, Sealevel) to view the notebook. Additionally, set the project to initialize with a README so we can immediately clone the repository.Graphical user interface, text, application, email

     Description automatically generated
  3. Once this is done, you should see a repository that looks similar to this:Graphical user interface, text, application, email

     Description automatically generated
  4. Ensure that the Data Science Group (that’s us) has the maintainer role. If it does not, we won’t be able to push our code to the repository.
  5. Then we can clone the repository, move the Notebook (with accompanying source code if necessary) into the repository.
  6. Then follow the exact steps in 1.b to tag the notebook and push the tag up to the SMCE GitLab project.

1. How a user becomes aware of the notebook.
   1. Look for the project in the group’s GitLab.
   2. Look for the most recent tag (version number)
   3. Follow ILAB container primer-appendix F to pull by tag.

1. See the following for Singularity and container motivations along with use cases: https://sylabs.io/guides/3.7/user-guide/ [↑](#footnote-ref-1)
2. A definition file is simply a text file ending in .def that specifies help info, installation dependencies, paths, labels, etc. [↑](#footnote-ref-2)
3. Some legacy containers are suffixed with .simg, but the .sif convention is now being used for alignment with Singularity best practices [↑](#footnote-ref-3)
4. See <https://sylabs.io/guides/3.0/user-guide/definition_files.html> for a full description of the Singularity definition file [↑](#footnote-ref-4)
5. As a practical matter, container builds are performed manually by a human in the Container Builder [CB] role. Glenn is currently the primary CB grooming Savannah. This process is subject to change when the new NCCS Container policies are enacted. [↑](#footnote-ref-5)
6. Containers are built on dsg101 by the CB (where sudo is required) and tested on dsg103 to replicate remote testing. [↑](#footnote-ref-6)
7. See the following for detailed build instructions: https://sylabs.io/guides/3.7/user-guide/ [↑](#footnote-ref-7)
8. The ilab-core-x.x.x.sif container is the parent for all ILAB application-specific containers because it houses shared ILAB resources. [↑](#footnote-ref-8)