UTSouthwestern Medical Center
Lyda Hill Department of Bioinformatics

BioHPC

BioHPC Reproducibility Series
Containers for Scientific Software

Training will begin at 10:32AM

biohpc-help@utsouthwestern.edu

26 Apr 2023

BioHPC Reproducibility Series

- Part 1 – Containers from a user perspective – running, pushing, pulling
- Part 2 – Containers from a developer perspective – building/writing; more technical, version control
- Part 3 – Continuous Integration / Continuous Deployment – automating time-consuming tasks.

These sessions are not fully planned out, so if you would like to see additional content, please email BioHPC Help.

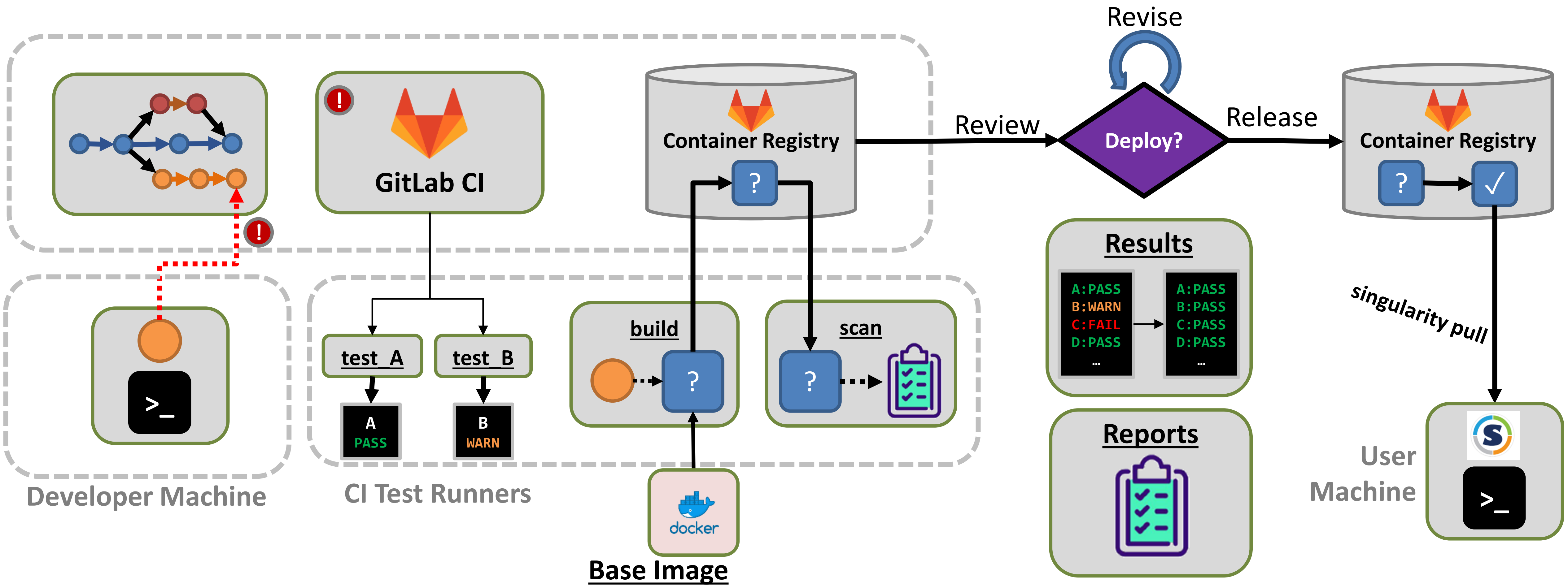
Code Repository
- Container Recipe
- .gitlab-ci.yml

→ **Test Runners**
- Run Tests
- Build/Tag images

→ **Image Registry**
- Login/Push to registry
- Container Scanning

→ **Review + Revise**
- CI Test Results
- Security Reports

→ **Release**



“Good to know” technical slides will be shown by this band – you can safely ignore these, but they are useful to improve your understanding.

TECHNICAL

In any code that follows,

- Lines beginning with $> are entered as commands in the terminal, or are individual lines in a script.

- A backslash \ at the end of a line is a line continuation, i.e.

```
$> cat \
~/my.file
```

is the same as

```
$> cat ~/my.file
```

BioHPC Reproducibility Series – Part 1: Containers

- What is a container?
  - Why should we use them for science?
  - What’s the difference between them and virtual machines?
  - Terminology
- Singularity and Apptainer – the HPC container technologies
  - pull – Downloading containers
  - running software inside of containers
  - push – Uploading containers
- BioHPC’s GitLab Container Registry
  - Access Tokens
- Walkthrough Repository
  - Code will be available some time after the training

Terminology

Container – A unit of encapsulated software (with dependencies) which is **running**

Image – The file which, when run, produces a running container.

- Often called a container image

Build – The process of creating an image from a recipe file.

- Details vary between different container technologies
- Usually requires root access or more modern virtualization technologies

Tag – The ‘name’ of an image. Can also include the ‘shipping address’ of an image.

- Assigned at build-time, or later
- Images can have multiple tags

Repository – Where your code goes.

Registry – Where your container images go.

Main points about containers

1. A container image is just a fancy directory tree containing different programs and libraries.
   - Different image formats → different ways to package this tree.
2. A running container is just a specially encapsulated process
   - Different container runtimes → Different ways to run a container.
3. Many programs that you might like to install are available as containers, and you can run them yourself.
   - Python, R, LAMMPS, bamtools, samtools, Tensorflow…
   - Biocontainers (https://github.com/BioContainers/containers)

- The following BioHPC modules are already running transparently as containers:

AtacWorks	danpos	magetbrain	Telseq
cellprofiler	deepvariant	Quarto	trinity
chimerax	DROMPAplus	R4.2	
Circos	guppy	Seurat	

Why do I care about containers?

- Containers are *isolated*
  - Running in their own environment, don’t affect each other → Combine software you normally couldn’t.
- Containers *include their dependencies* and *travel with them*
  - Same code running the same way everywhere.
- Containers are *lightweight*
  - Most containers run with with speed comparable to a regular program.

- Imagine a real-life shipping container that has an entire wet lab inside.

Why do I care about containers?

- Containers are *isolated*; *include their dependencies* and *travel with them*; and are *lightweight*

As a user of containers:

- This allows you to use more sophisticated workflows
- Easily run complex software without painful installations.

As a creator of containers:

- You can spend much more time adding core features
- Substantially easier to get others to use your code → Get your science out there.

What actually is a container? Why use containers at all when virtual machines (VMs) exist?

- Containers are a software technology which *contain* a computational environment.
  - Binaries and executables
  - Libraries
  - Dependencies in general – only what's needed for a single software.

- Containers are best for when you need…
  - Single-task applications with short lifetimes
  - An immutable artifact that you can use later.

- VMs are best for when you need…
  - Flexible systems with long lifetimes
  - Strong isolation
  - Administrator/root control (e.g. <u>BUILDING</u> containers)

Singularity and Apptainer, Docker and Podman

Singularity is a container technology which is designed for HPC usage.

- Containers run as the user who starts them
- User cannot elevate privileges
- Can use high-speed storage easily
- Can run most OCI containers
- Better with SLURM, MPI, etc.

Apptainer is effectively the same project, which split in a different direction around Dec 2021.

Docker is the original OCI-compliant container technology

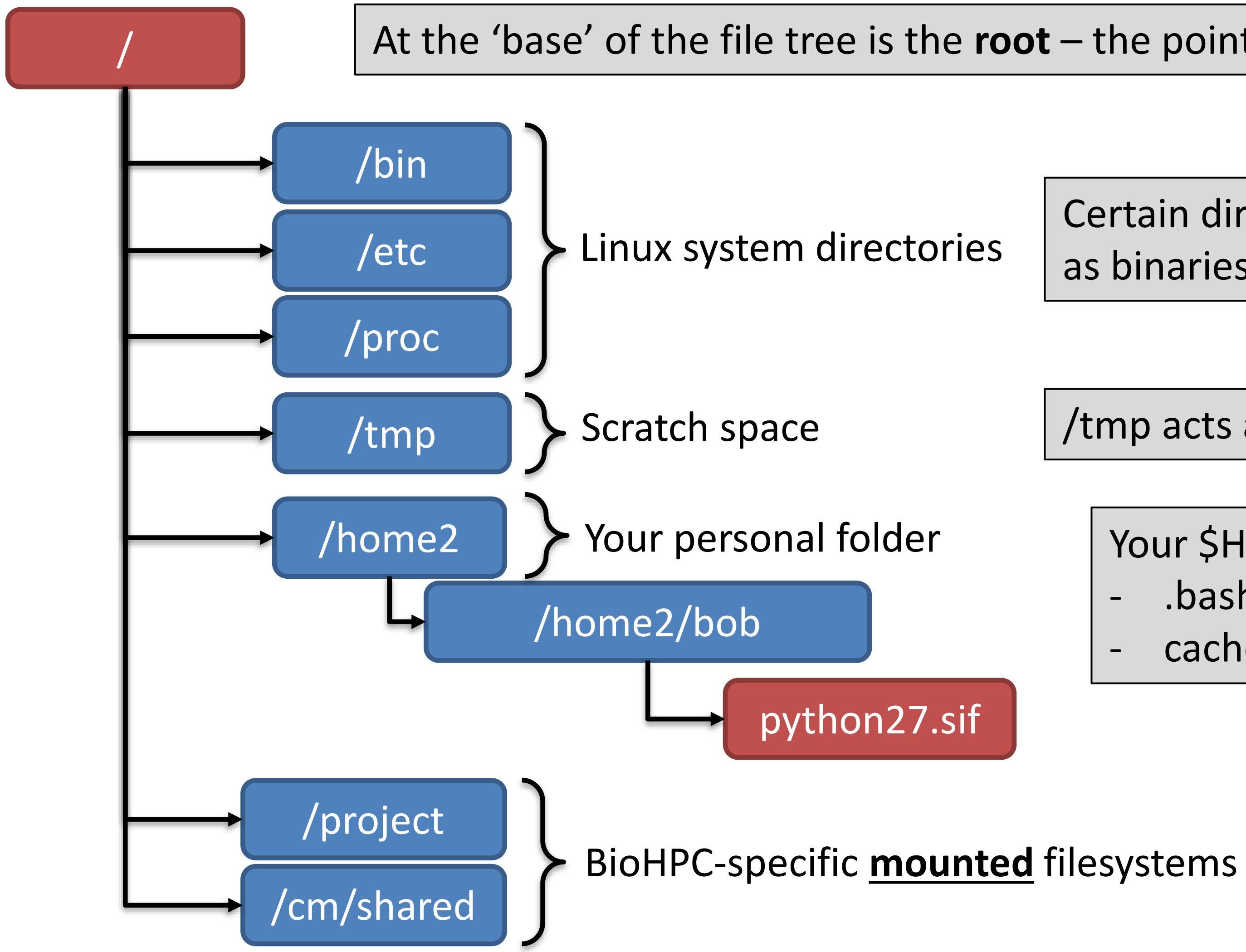
- Containers as services
- Containers are briefly run as a root user
- Entirely different mode of operation (client/server model)

Podman is a drop-in replacement for Docker with better security features.

- Works similarly to Singularity, with similar security.
- Can do rootless containers and rootless builds
- **Coming in the next couple of cluster upgrades**

We will focus on Singularity containers running on BioHPC

A basic filesystem



At the ‘base’ of the file tree is the **root** – the point where all absolute paths are referenced.

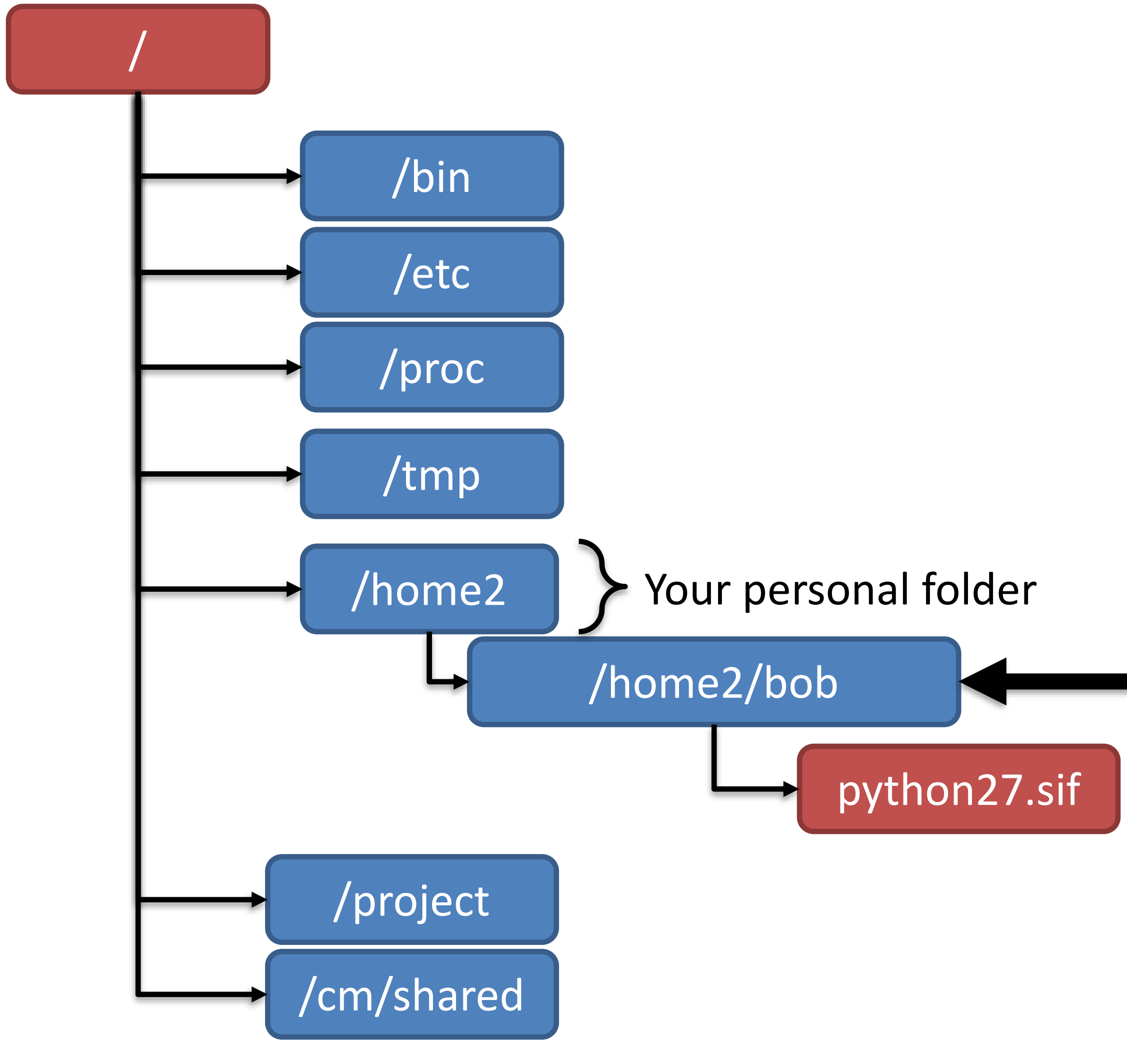
Certain directories come with Linux and are where such things as binaries/executables, system libraries, etc. are installed to.

/tmp acts as a temporary scratch space

Your $HOME folder (on our system, /home2) → personal config
- .bashrc, ssh keys, etc.
- caches, Python envs, R libraries…

On BioHPC, we provide access to additional storage through **mounts**. Provides access to your data, and in the case of /cm/shared, modules.

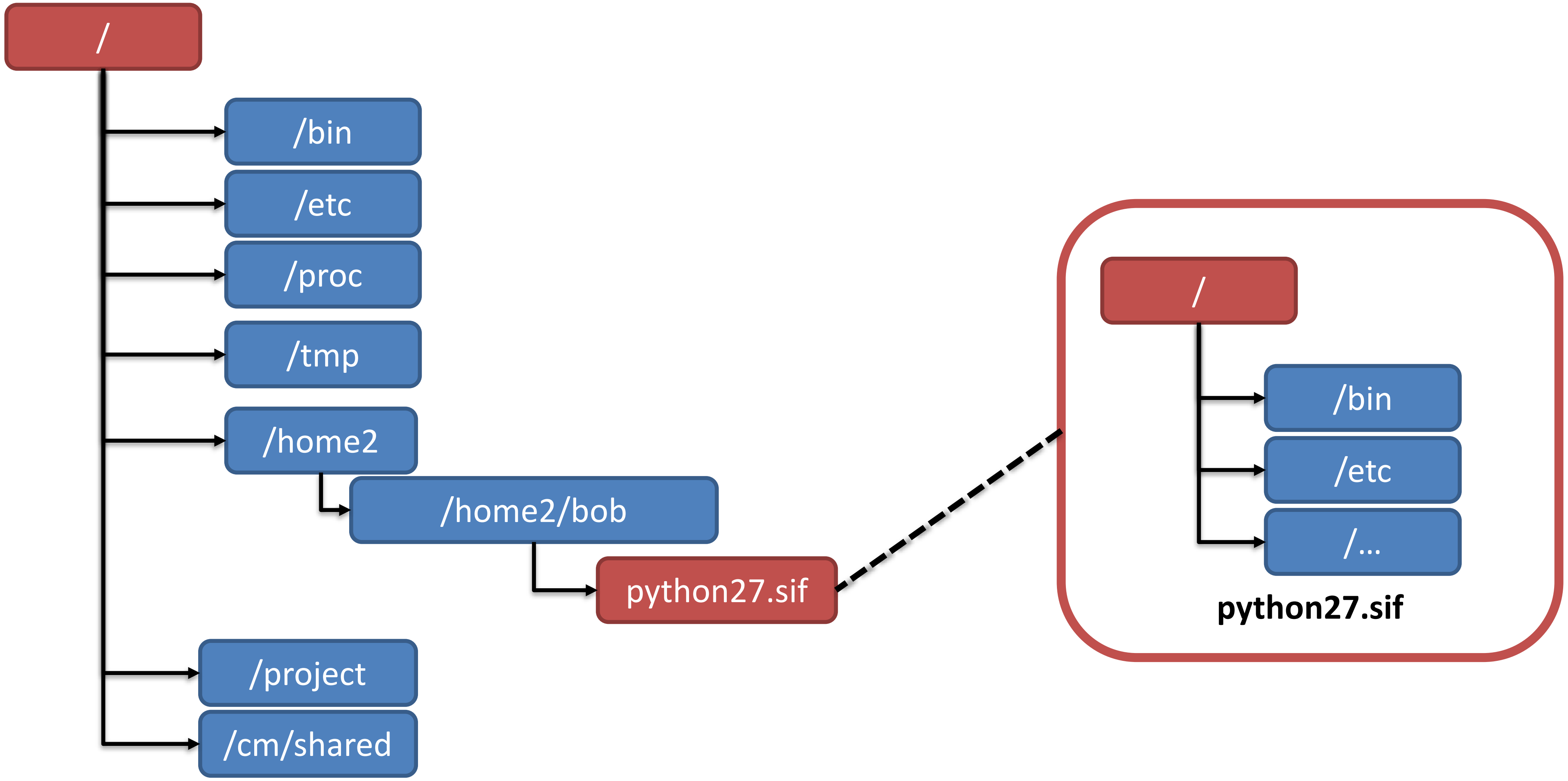
When you log in to BioHPC, you are dropped into a shell in your home folder.



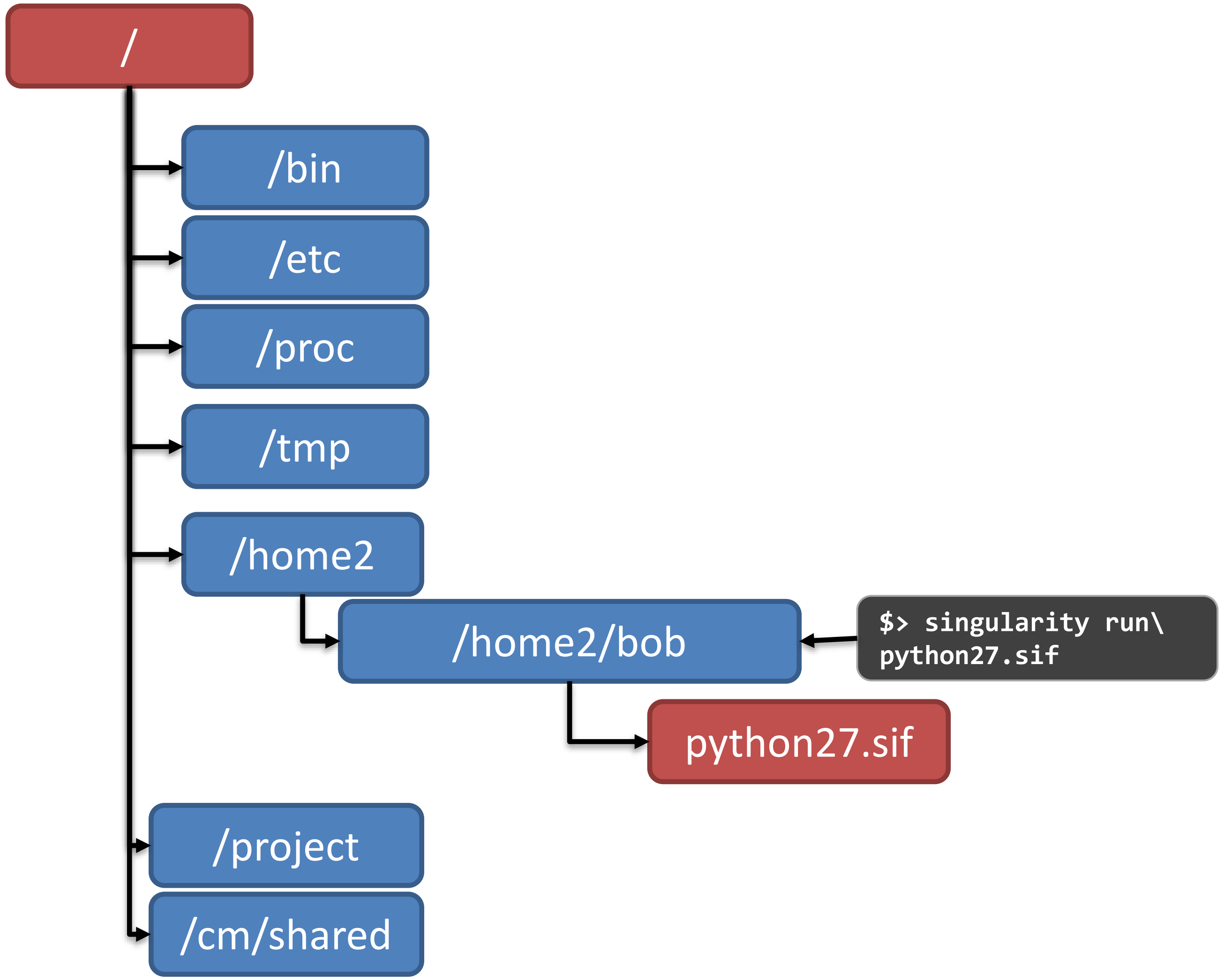
```
$> echo “${HOME}”
/home2/bob
```

UTSouthwestern Medical Center
Lyda Hill Department of Bioinformatics
BioHPC

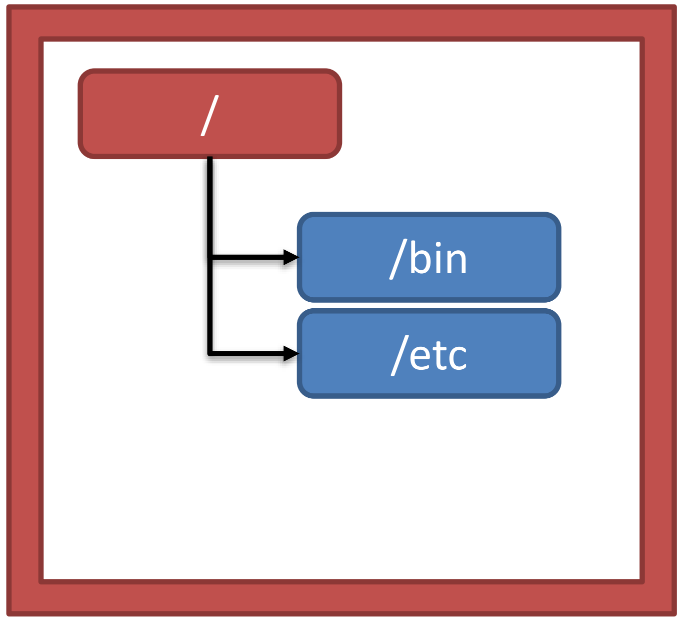
A container image is basically a directory tree of its own (like a zip archive)



Singularity creates a container from an image file…

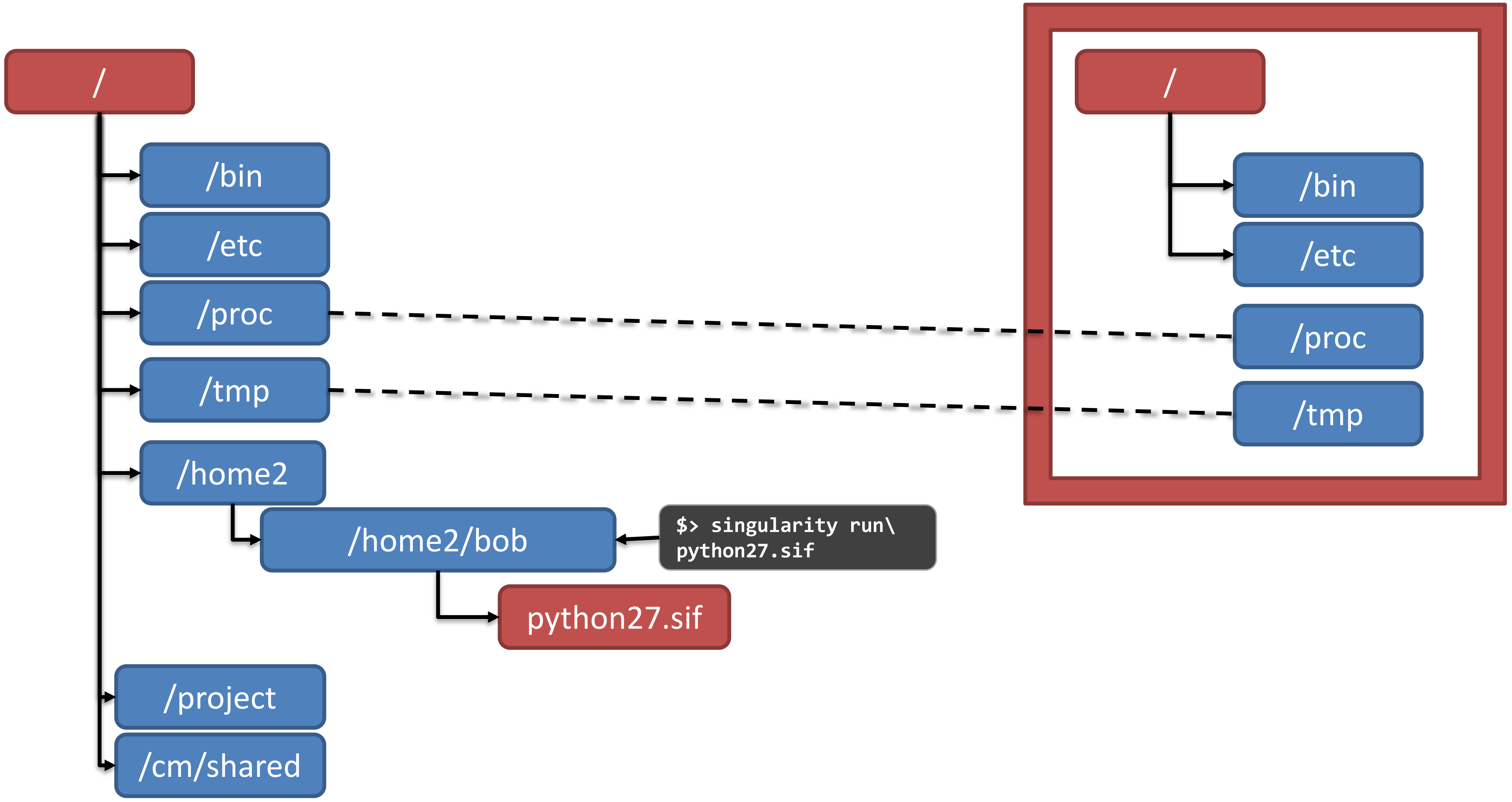


```
$> singularity run\
python27.sif
```



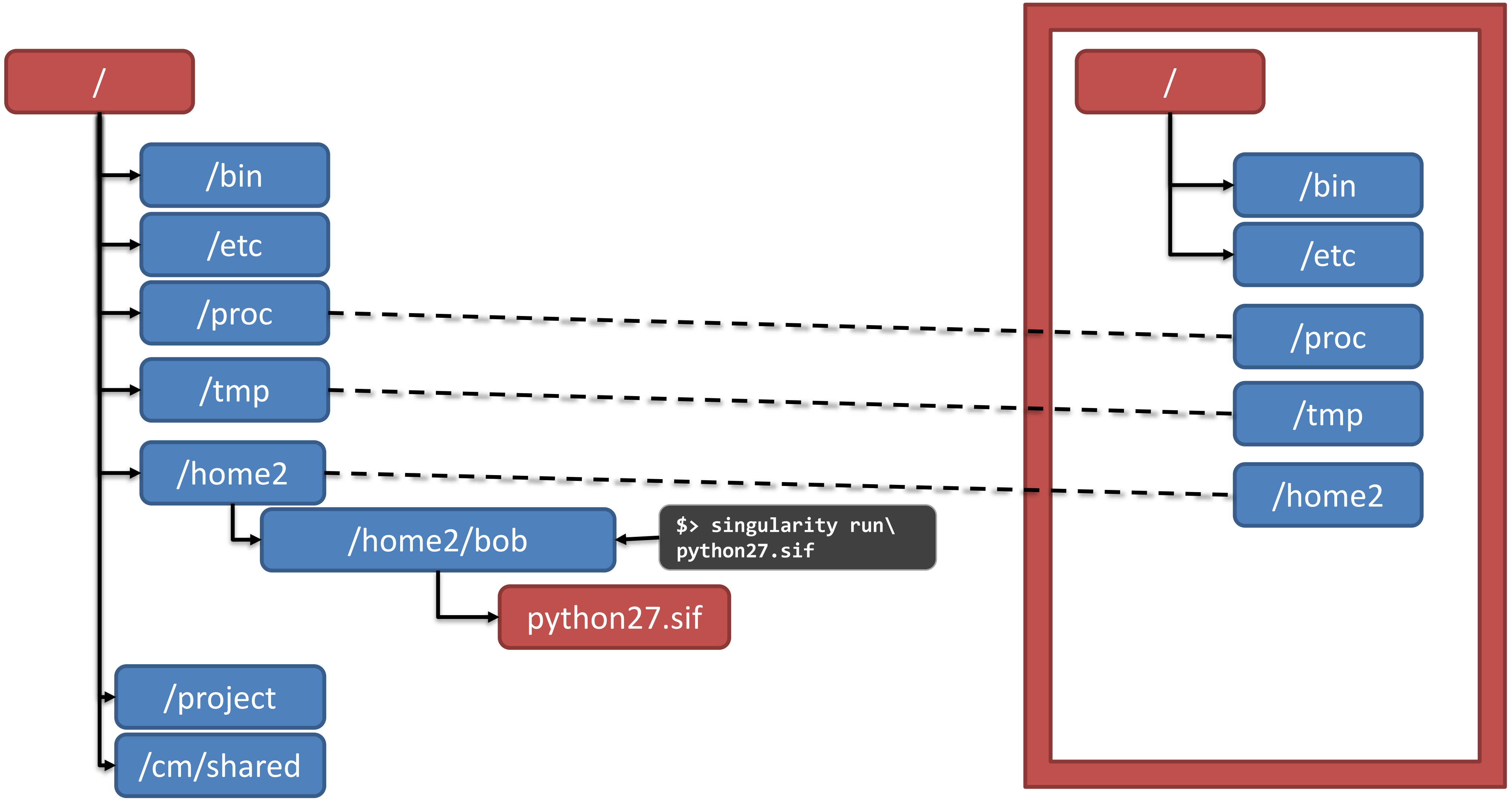
UTSouthwestern Medical Center
Lyda Hill Department of Bioinformatics
BioHPC

…borrows some critical folders from the host…



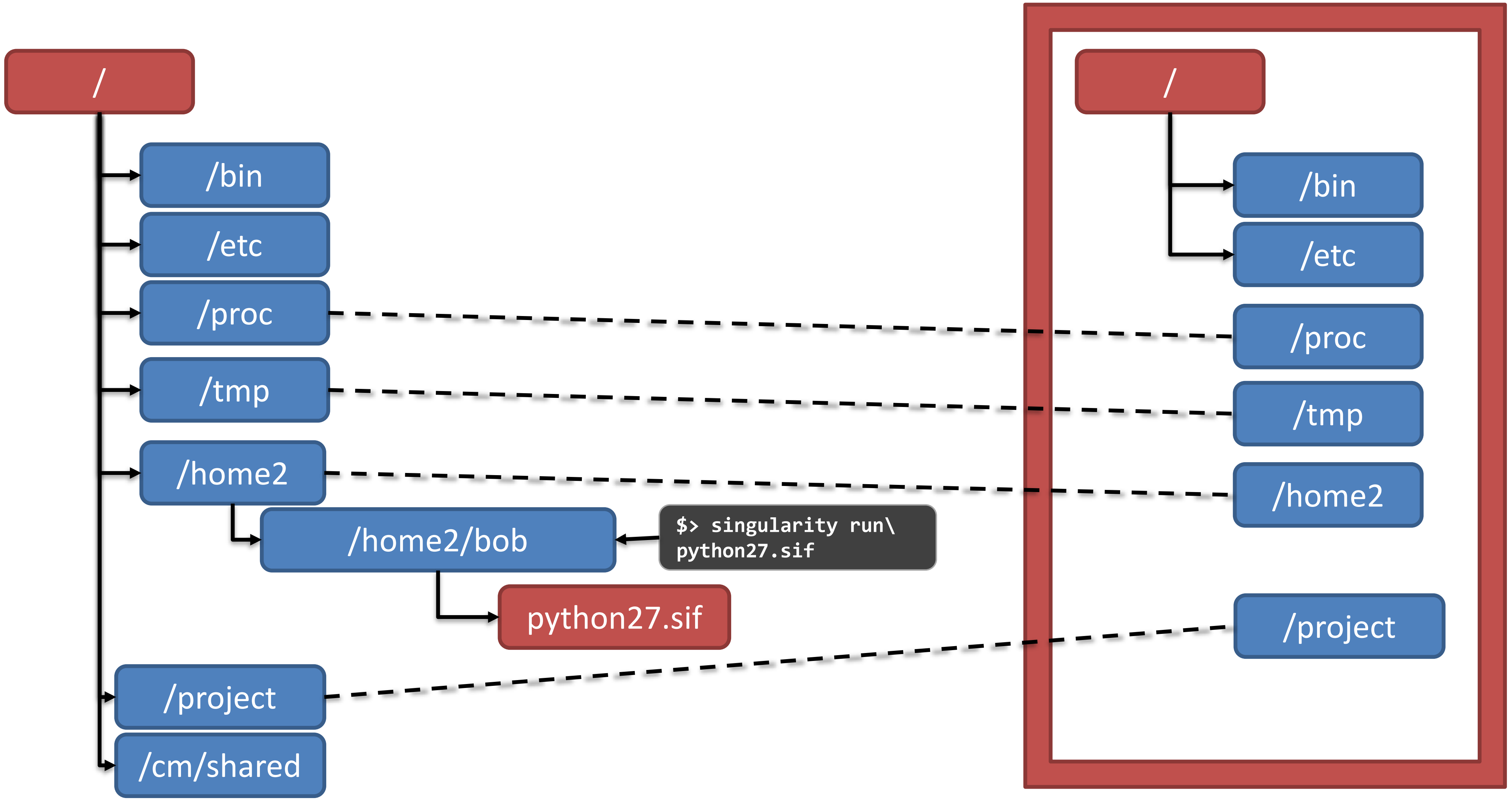
```
$> singularity run\
python27.sif
```

…maps your external user info inside the container…



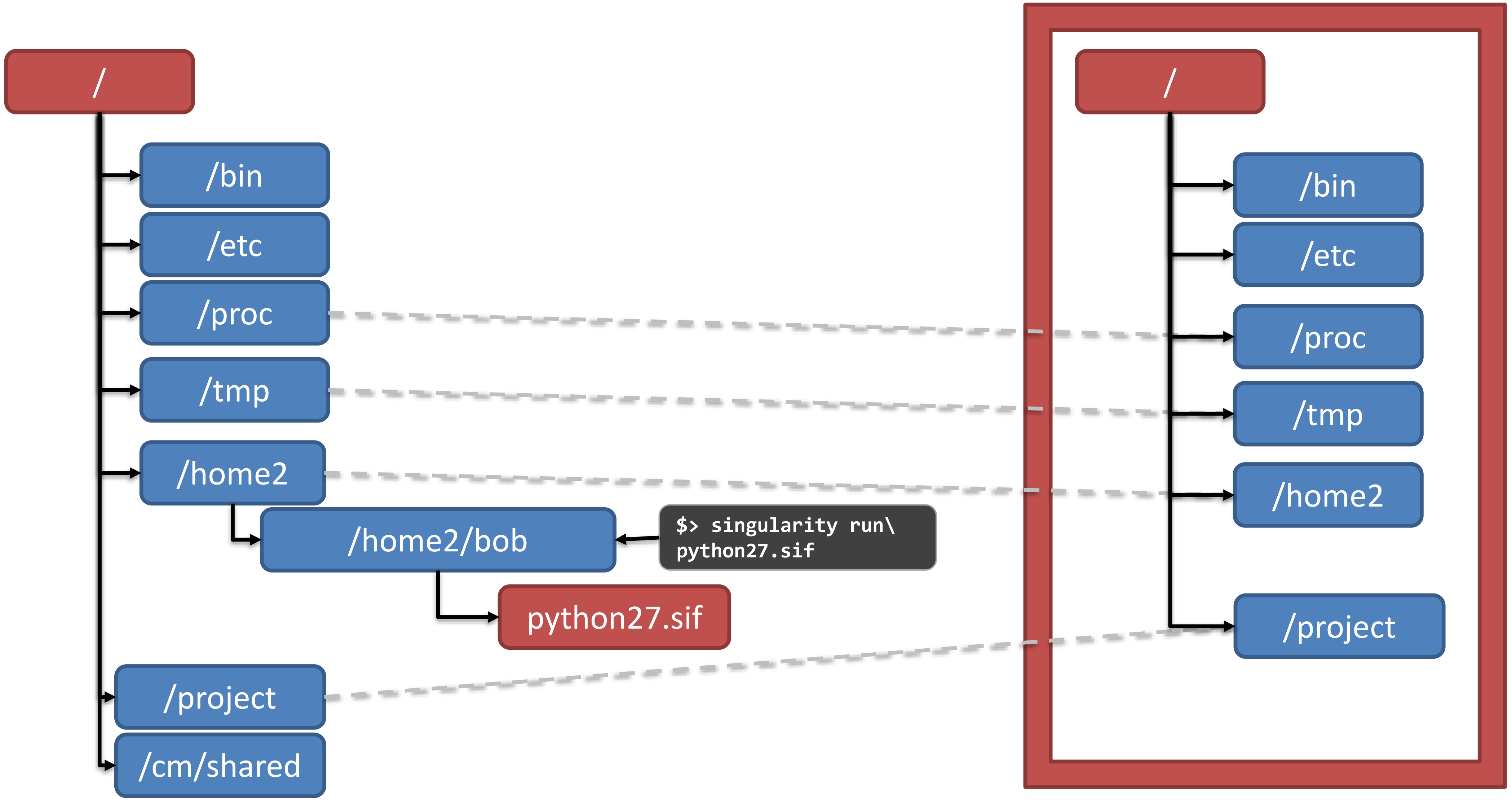
```
$> singularity run\
python27.sif
```

… optionally does additional config (e.g. mounting more folders, setting env vars)…



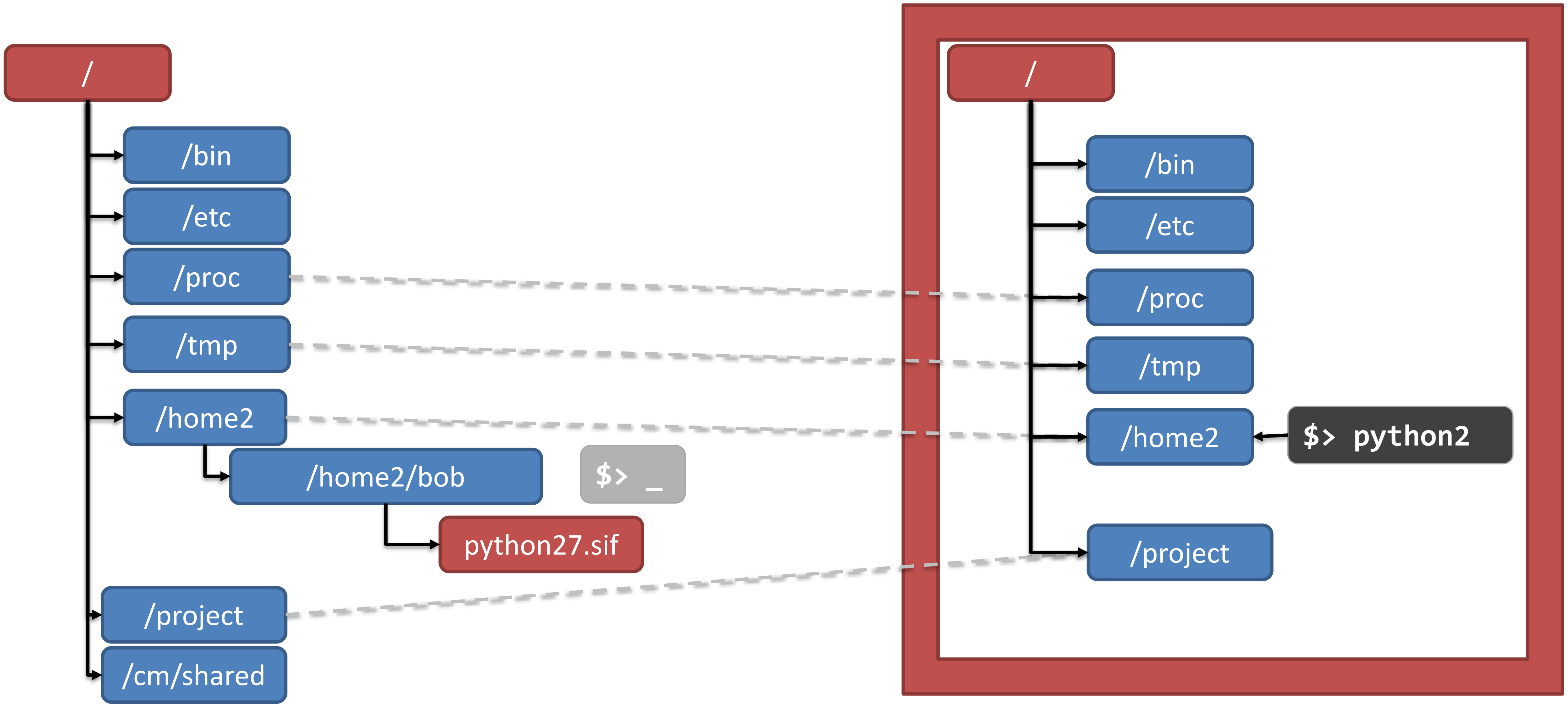
```
$> singularity run\
python27.sif
```

… links everything together inside the container …

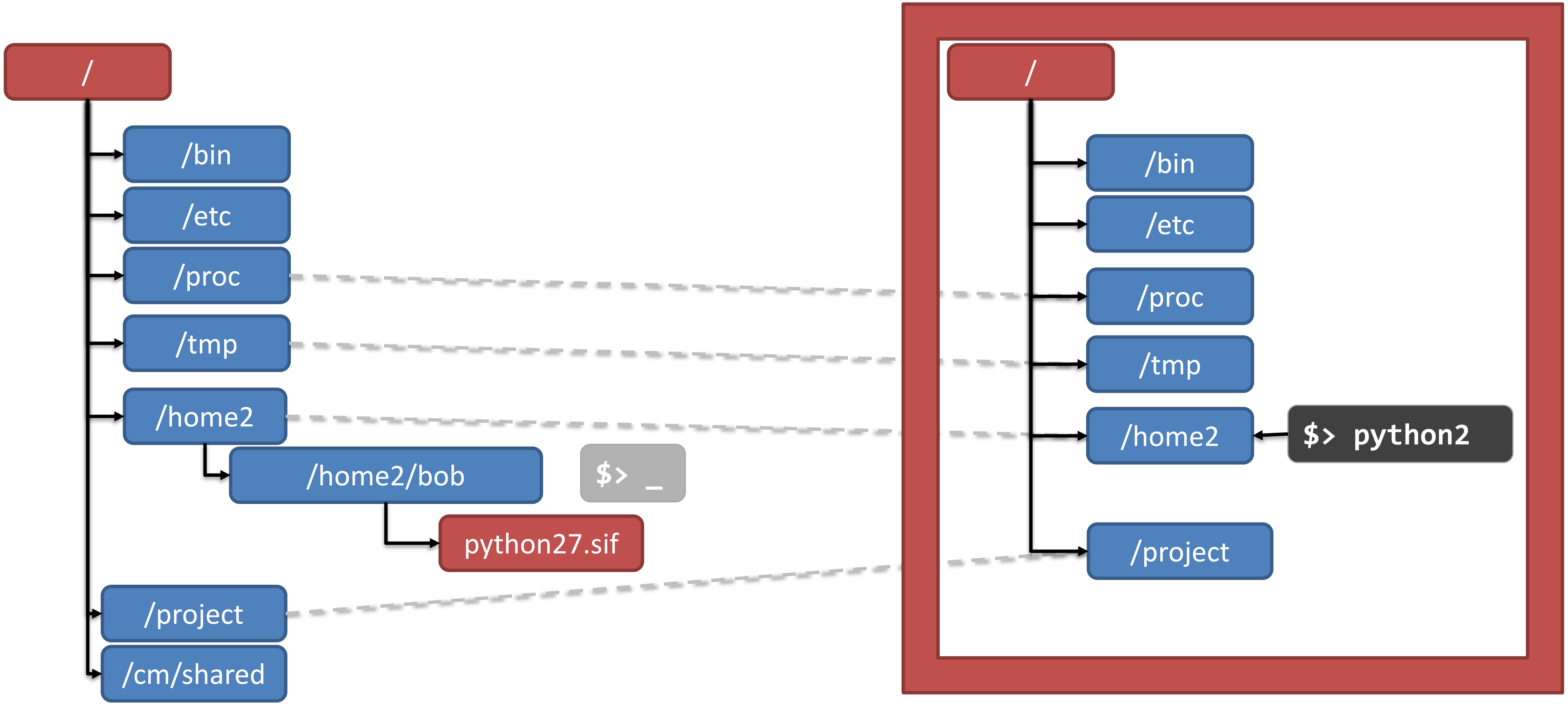


```
$> singularity run\
python27.sif
```

… and starts running a command inside the container.

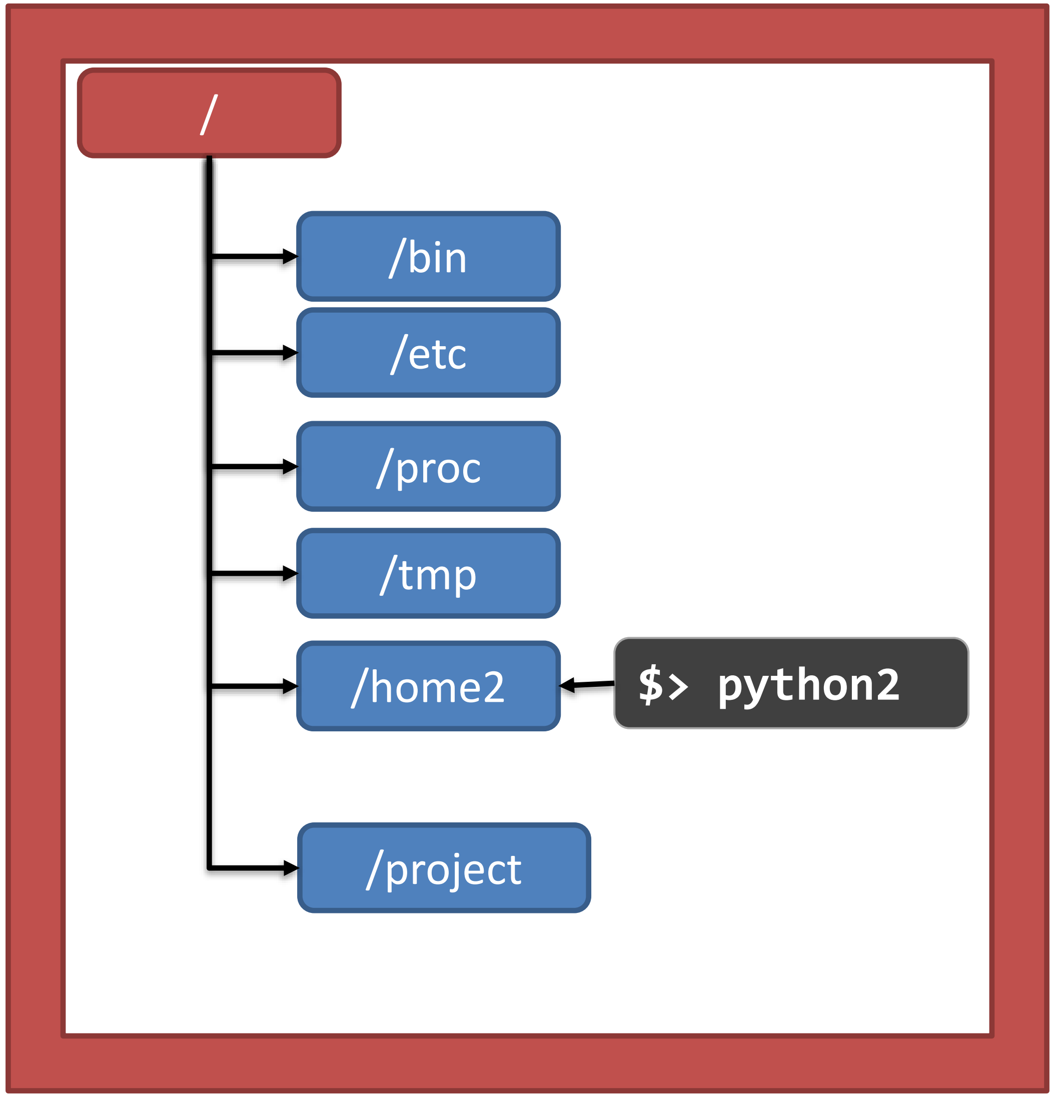


… and starts running a command inside the container.



UTSouthwestern Medical Center
Lyda Hill Department of Bioinformatics
BioHPC

… and starts running a command inside the container.



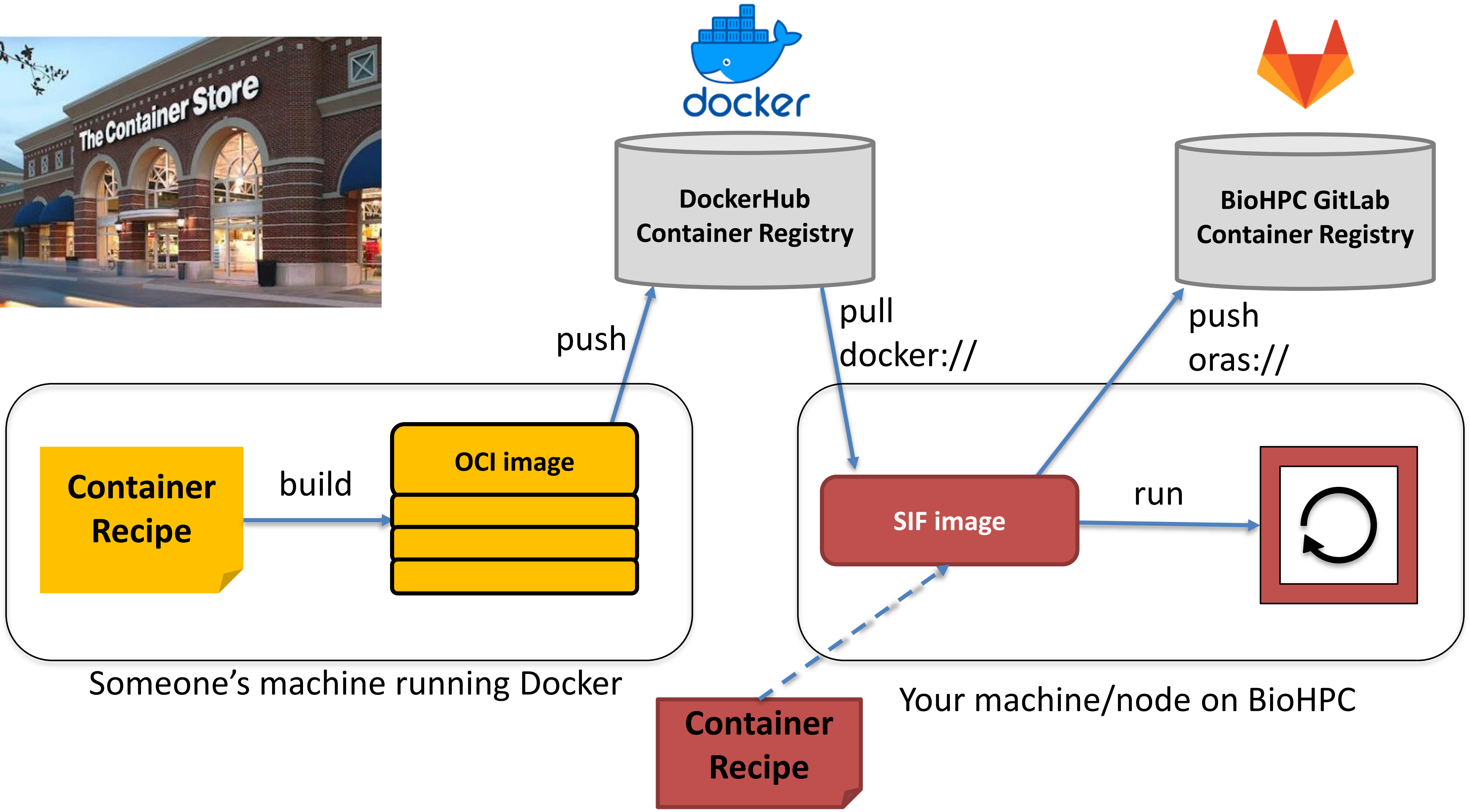
As far as the command inside the container is concerned, **it's running on a normal system**.

It doesn't matter if the host is Ubuntu 22.04, or Red Hat 7.3, or running on BioHPC, or running anywhere else –

- **the environment inside the container is the same.**

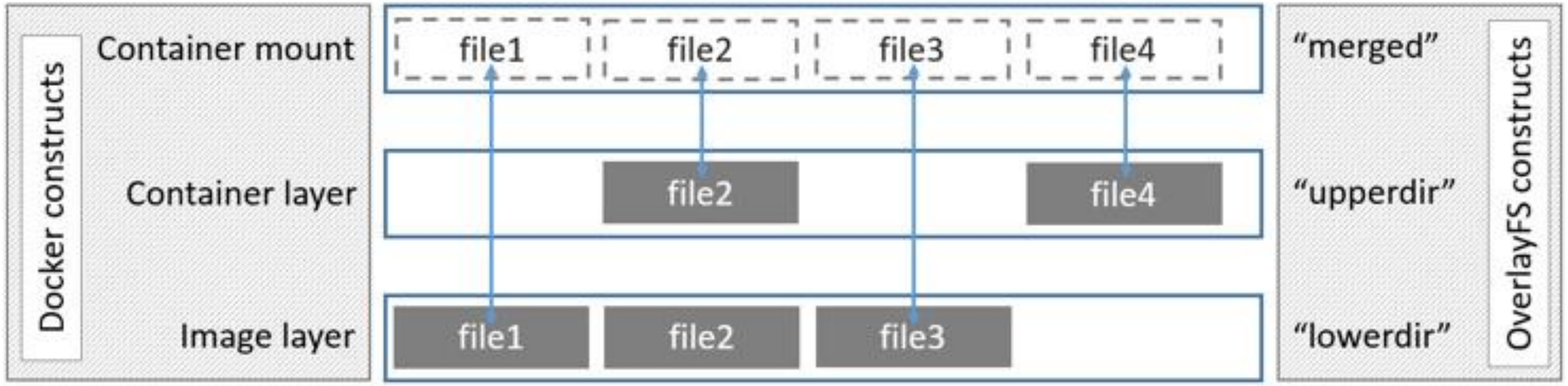
Rather than building workflows where everything has to install and work together, you can have each container perform one step at a time.

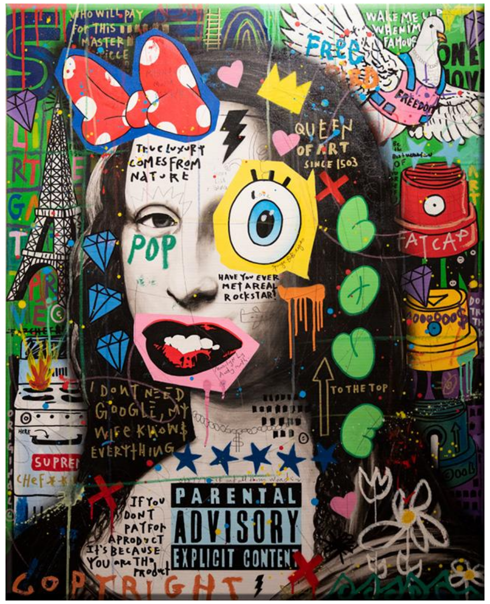
A Typical Container Lifecycle


docker
DockerHub Container Registry
BioHPC GitLab Container Registry
push
pull docker://
push oras://
Container Recipe
build
OCI image
SIF image
run
Someone's machine running Docker
Your machine/node on BioHPC
Container Recipe

Overlay Filesystems

- Like Photoshop layers, or merging config files, upper layer ‘overlays’ on a lower layer.
  - If a file is **present in the lowerdir** but **NOT the upperdir**, the **lowerdir** file will appear.
  - If a file is **NOT present in the lowerdir**, but **IS in the upperdir**, the **upperdir** file will be used.
  - If a file is **present in both**, the **upperdir** file will be used.
- Think Photoshop layers.





UTSouthwestern Medical Center
Lyda Hill Department of Bioinformatics
BioHPC

What’s the difference between OCI (Docker/Podman) and Singularity images?

TECHNICAL

OCI images are made of layers that are overlaid on top of one another to yield the final image.

- This is a good design if there are a lot of containers that are very similar – can reuse identical layers.

Singularity uses a single ‘monolithic’ image file

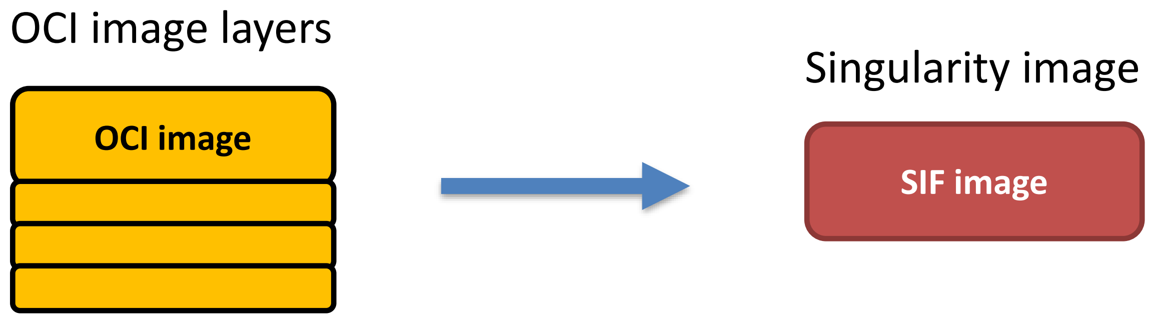
- This lets the entire container be moved around as a single file and is easier to archive.

More focused towards HPC workloads

- Easier to use with SLURM, MPI, etc.
- Better security design
- More integrated → Less configuration needed

Can convert OCI images to Singularity images

- ‘squashes’ the layers into a single file.



Using Singularity/Apptainer

- On BioHPC, you can **run** containers, and you can **pull/push** containers, but you may not **build** containers (currently)

- **build** usually requires additional permissions (e.g. **yum** or **apt-get** install – things for the root user)
- We are developing a Constructor interface to make it easier to build containers on BioHPC.

- We have a somewhat technical workflow for building containers (more on that later)
  - Let us know if you're interested in this!

<u>Resources:</u>

- https://docs.sylabs.io/guides/latest/user-guide/quick_start.html#overview-of-the-singularityce-interface
- https://apptainer.org/docs/user/main/quick_start.html#overview-of-the-apptainer-interface

What you need for what you're doing

If you want to use containers from a registry, you can use Singularity on any BioHPC system (workstation, cluster node)

- **module add singularity (Recommend 3.9.9 or 3.5.3)**
- Download Singularity images, or convert (most) OCI images to Singularity images
- Run Singularity image

If you want to build containers, you must build on a non-Nucleus system

- Virtual Machine (Running on personal computer, UTSW computers)
- Docker for Windows
- Vagrant Box on **BioHPC workstation** (not cluster node!)

If you would like guidance on building containers with Vagrant, contact BioHPC Help

singularity pull – Getting your first container!

- Singularity can pull from multiple locations:

```
From Sylabs cloud library
$ singularity pull alpine.sif library://alpine:latest

From Docker Hub – defaults to docker.io. These lines are equivalent.
$ singularity pull tensorflow.sif docker://tensorflow/tensorflow:latest
$ singularity pull tensorflow.sif docker://docker.io/tensorflow/tensorflow:latest

From supporting OCI registry
$ singularity pull image.sif oras://some.registry.endpoint/namespace/image:version_tag
```

Singularity images can be stored in OCI registries using the ORAS (OCI Registry As Storage) protocol

*shub:// endpoints are valid for now, but their future is uncertain.

singularity pull – Getting your first container!

- Singularity can pull from multiple locations:

```
From Sylabs cloud library
$ singularity pull alpine.sif library://alpine:latest

From Docker Hub – defaults to docker.io. These lines are equivalent.
$ singularity pull tensorflow.sif docker://tensorflow/tensorflow:latest
$ singularity pull tensorflow.sif docker://docker.io/tensorflow/tensorflow:latest

From supporting OCI registry
$ singularity pull image.sif oras://some.registry.endpoint/namespace/image:version_tag
```

Singularity images can be stored in OCI registries using the ORAS (OCI Registry As Storage) protocol

*shub:// endpoints are valid for now, but their future is uncertain.

UTSouthwestern Medical Center
Lyda Hill Department of Bioinformatics
BioHPC

singularity pull – Authenticating

- If you are pulling from a Docker/OCI-type registry, you may need to authenticate first.

--docker-login is one-time (the duration of the command)

```
# From Docker Hub – defaults to docker.io
$ singularity pull --docker-login tensorflow.sif docker://tensorflow/tensorflow:latest
$ singularity pull --docker-login tensorflow.sif docker://docker.io/tensorflow/tensorflow:latest

# From supporting OCI registry
$ singularity pull --docker-login image.sif oras://some.registry.endpoint/namespace/image:version_tag

# You can also export some environment variables containing your login info before trying to pull.
$ export SINGULARITY_DOCKER_USERNAME="Test_Tok"
$ export SINGULARITY_DOCKER_PASSWORD="zH_AQRxrnesN8UEgzNop"

# Singularity will automatically use the provided credentials to log in to the registry
$ singularity pull image.sif oras://some.registry.endpoint/namespace/image:version_tag
```

Singularity runs containers as your user account

- Singularity runs the containers as you - anything you do inside the container ‘looks like’ your username did it.
  - Mounts your /home2 directory by default.
- We have configured the Singularity module to mount the BioHPC filesystems into containers by default (/project, /work, /archive)

- Can prevent this behavior with --contain option.

- Can ‘bind’ additional directories within the container.
  - Using --bind src[:dest[:opts]]
  - So if you need a certain file to be in a particular location within the container, you can make that happen.

Singularity run

A container is built with a default command – in the case of the **python_2.7** image, this is the command **python2**

```
$ singularity pull python_2.7.sif docker://docker.io/python:2.7
$ singularity run python_2.7.sif
# Having run the previous command, you will be dropped into a Python instance.
```

```
$ singularity run python_2.7.sif
Python 2.7.18 (default, Apr 20 2020, 19:27:10)
[GCC 8.3.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> 2+2
4
>>>
```

You can also run without manually pulling.

```
$ singularity run docker://docker.io/python:2.7
# Singularity will pull the image, convert it to a SIF format, and then run it as before.
```

Singularity Exec

singularity exec <container> <command> runs the specified command inside of the container

```
$ singularity exec python_2.7.sif python -c 'print 2+2'
4
```

Any command available inside a container can be used

```
$ singularity exec python_2.7.sif ls /
archive  boot  endosome     etc   home2  lib64  mnt  proc     root  sbin         srv  tmp  var
bin      dev   environment  home  lib    media  opt  project  run   singularity  sys  usr  work
```

Commands might be available in the host, but not in the container.

```
$ nc --version
Ncat: Version 7.50 ( https://nmap.org/ncat )
$ singularity exec python_2.7.sif nc --version
/.singularity.d/actions/exec: 21: exec: nc: not found
```

Singularity shell

- **singularity shell** will drop you into an interactive command-line shell within the container.

```
$ singularity shell python_2.7.sif
        Singularity> cat /etc/os-release | head -n 3
        PRETTY_NAME="Debian GNU/Linux 10 (buster)"
        NAME="Debian GNU/Linux"
        VERSION_ID="10"
        Singularity> exit
        exit
$ cat /etc/os-release | head -n 3
NAME="Red Hat Enterprise Linux Server"
VERSION="7.7 (Maipo)"
ID="rhel"
```

This is a great way to play around with containers – **shell** is like an interactive **exec**

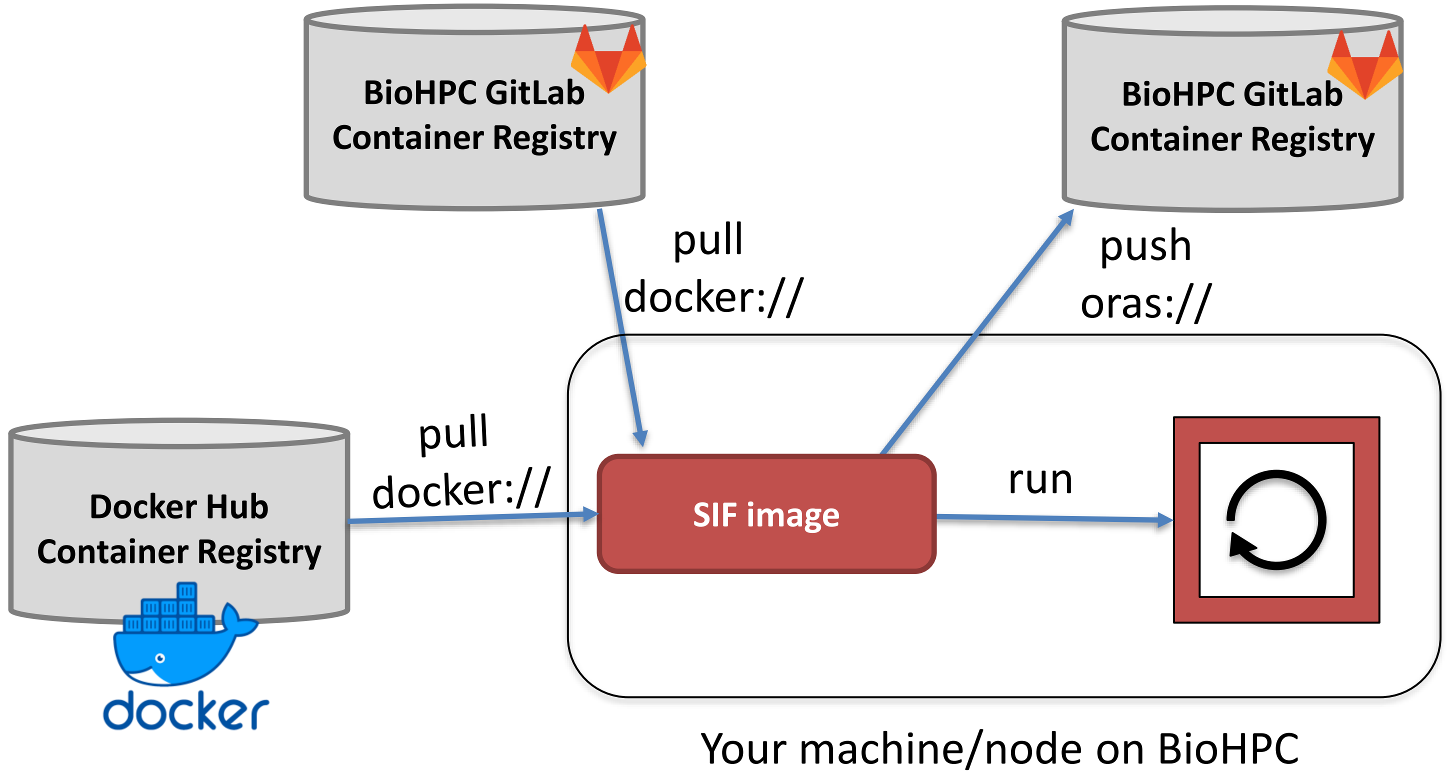
Other singularity commands of use

- singularity sif allows you to inspect the singularity image file itself, which can be useful for understanding its behavior.
  - This is an advanced debugging topic.
- singularity cache list shows the cache – this can be quite large, especially if you pull OCI images.
  - singularity cache clean will clear this (can save 10s of GB of storage)
- singularity instance will run a container in the background, like a daemon or service.

- singularity help
  - ☺

Singularity push

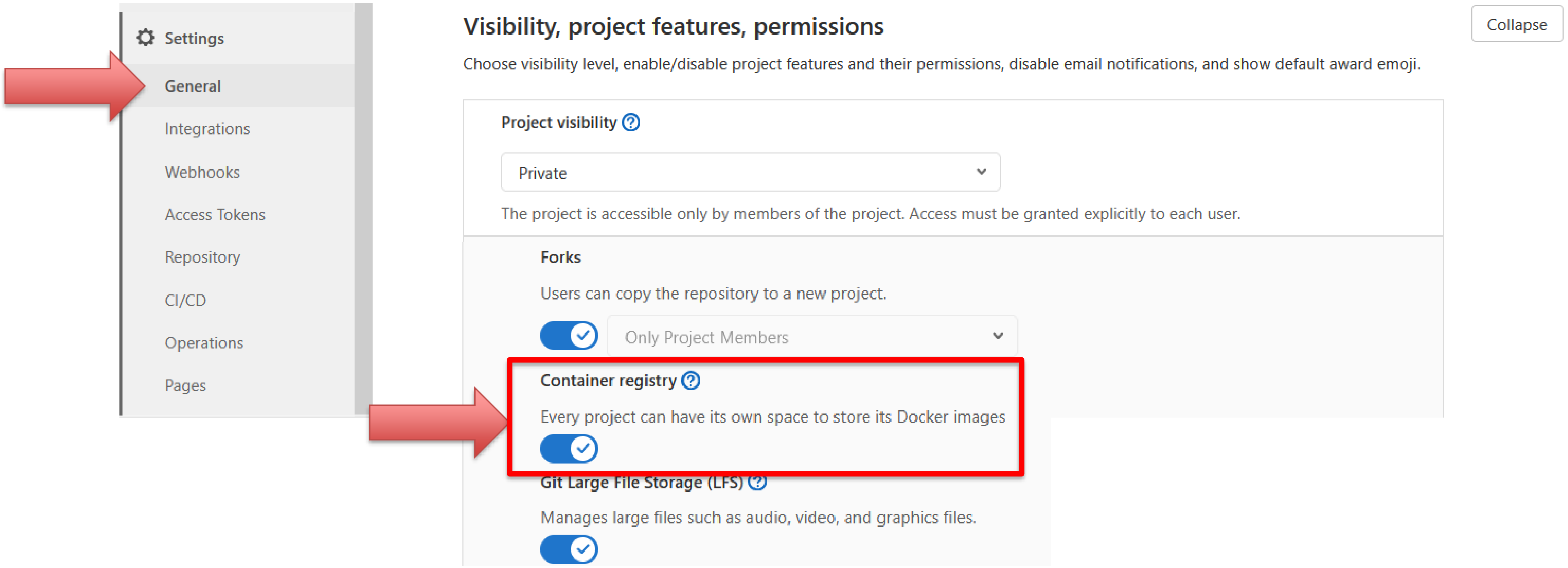
- Singularity can **push** to **oras://** and **library://** endpoints, but not **docker://**
  - Can’t ‘unsquash’ a SIF file into something Docker Hub understands.


BioHPC GitLab Container Registry
BioHPC GitLab Container Registry
pull docker://
push oras://
Docker Hub Container Registry
docker
pull docker://
SIF image
run
Your machine/node on BioHPC

GitLab Registry – Quick Overview

1) Setting up your GitLab Registry
2) Setting up access credentials
3) Logging in…
   a) With Docker (for building and pushing)
   b) With Singularity (for using/pulling)
4) Examples

First, create a repository and enable the Container Registry



Visibility, project features, permissions

Choose visibility level, enable/disable project features and their permissions, disable email notifications, and show default award emoji.

Project visibility

Private

The project is accessible only by members of the project. Access must be granted explicitly to each user.

Forks

Users can copy the repository to a new project.

✓ Only Project Members

Container registry

Every project can have its own space to store its Docker images

✓

Git Large File Storage (LFS)

Manages large files such as audio, video, and graphics files.

✓

Next, Access Credentials

- When logging into a private GitLab Container Registry, you CAN use your BioHPC credentials. This is more convenient for testing, but this is not a good practice.

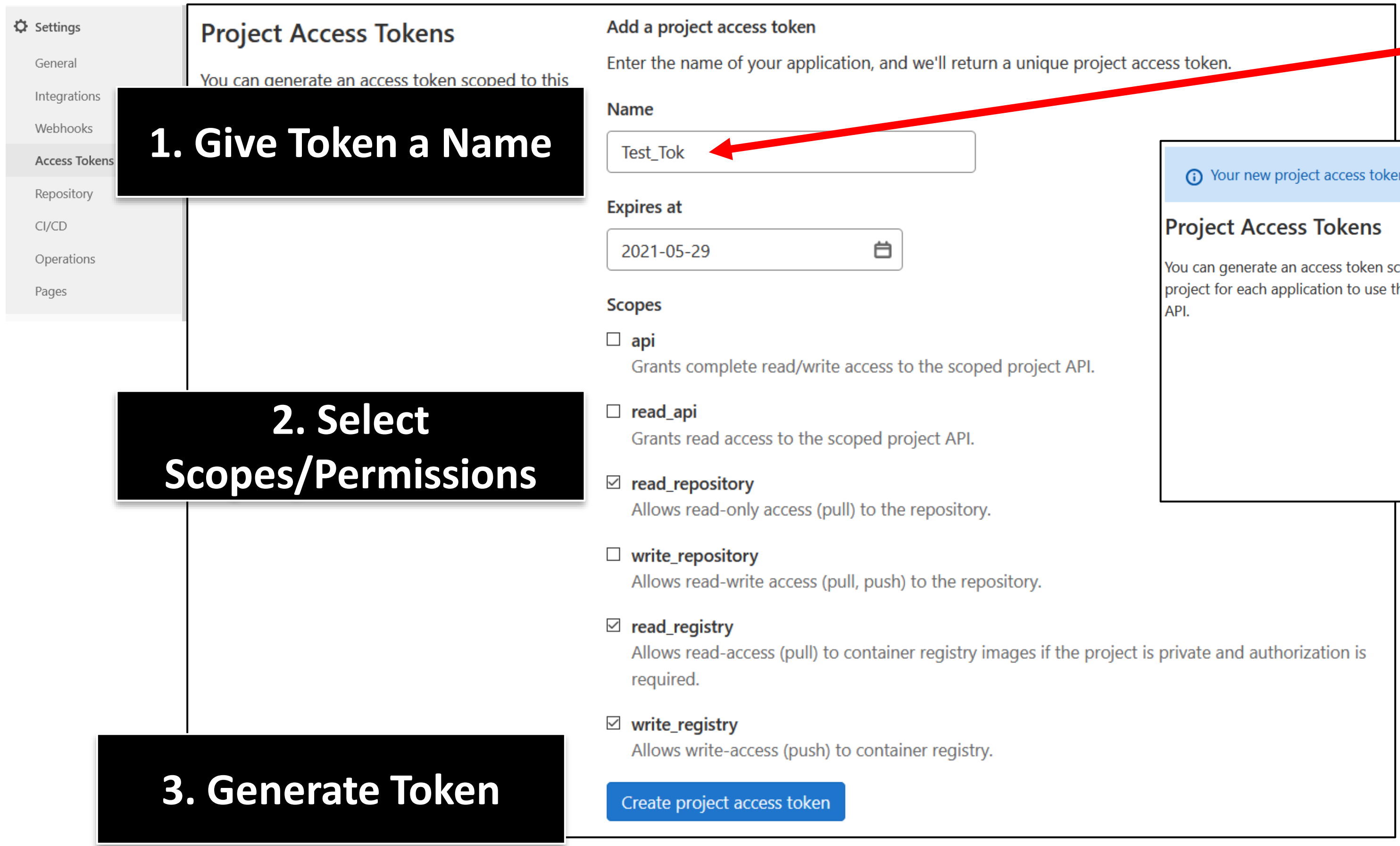
- For security purposes, it is best to generate Access Tokens.

- Project Access Tokens are generated in association with a single project, and can provide access to that project's repository and registry.

- Personal Access Tokens are generated in association with a user, and can provide access to the repositories and registries of any project that the user has access to.

- Both can have their permissions controlled (e.g. a Project Token that only allows users to pull images, but does not allow pushing)

Access Tokens - Project



Project Access Tokens

You can generate an access token scoped to this

Add a project access token

Enter the name of your application, and we'll return a unique project access token.

Name

Test_Tok

Expires at

2021-05-29

Scopes

- [ ] api
  Grants complete read/write access to the scoped project API.
- [ ] read_api
  Grants read access to the scoped project API.
- [x] read_repository
  Allows read-only access (pull) to the repository.
- [ ] write_repository
  Allows read-write access (pull, push) to the repository.
- [x] read_registry
  Allows read-access (pull) to container registry images if the project is private and authorization is required.
- [x] write_registry
  Allows write-access (push) to container registry.

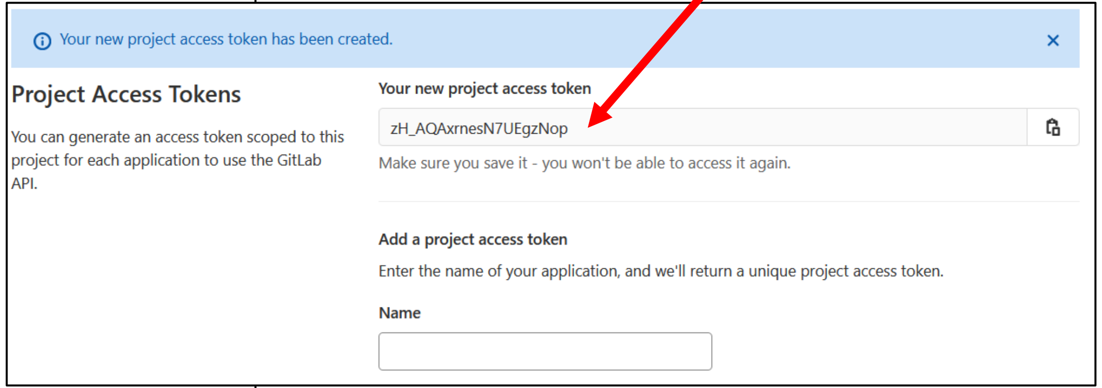
1. Give Token a Name

2. Select Scopes/Permissions

3. Generate Token

"Username"

"Password"



Your new project access token has been created.

Project Access Tokens

You can generate an access token scoped to this project for each application to use the GitLab API.

Your new project access token

zH_AQAxrnesN7UEgzNop

Make sure you save it - you won't be able to access it again.

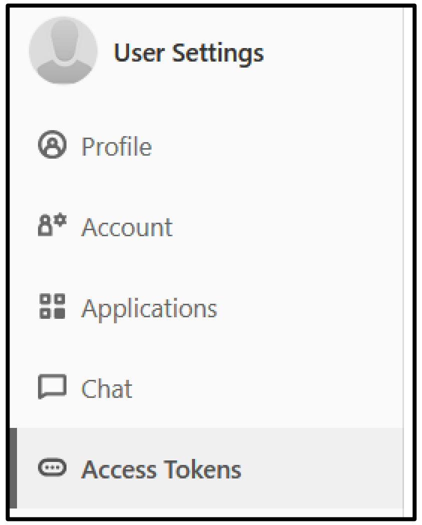
Add a project access token

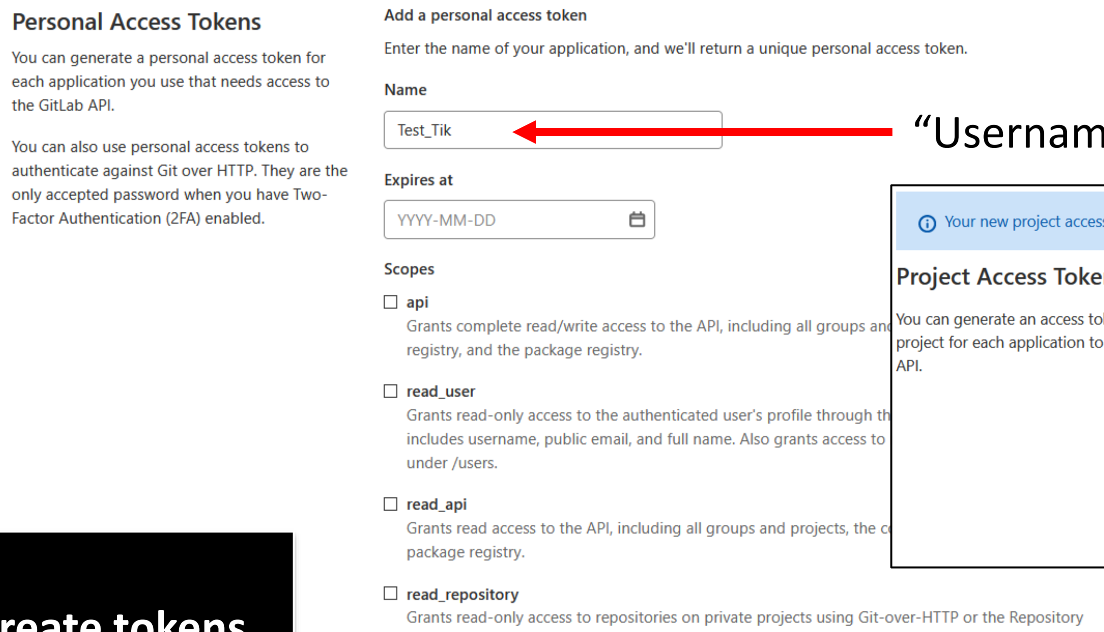
Enter the name of your application, and we'll return a unique project access token.

Name

4. Record the Name and Token – they are your Username and Password you will use to log in to GitLab from your command line or scripts.

Access Tokens - Personal




Personal Access Tokens

You can generate a personal access token for each application you use that needs access to the GitLab API.

You can also use personal access tokens to authenticate against Git over HTTP. They are the only accepted password when you have Two-Factor Authentication (2FA) enabled.

Add a personal access token

Enter the name of your application, and we'll return a unique personal access token.

Name: Test_Tik

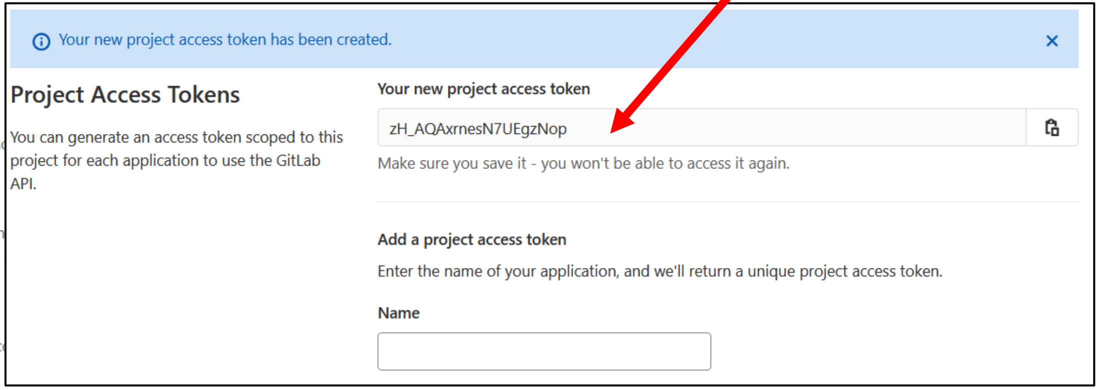
Expires at: YYYY-MM-DD

Scopes

- [ ] api
  Grants complete read/write access to the API, including all groups and registry, and the package registry.
- [ ] read_user
  Grants read-only access to the authenticated user's profile through th includes username, public email, and full name. Also grants access to under /users.
- [ ] read_api
  Grants read access to the API, including all groups and projects, the c package registry.
- [ ] read_repository
  Grants read-only access to repositories on private projects using Git-over-HTTP or the Repository

"Username"

"Password"


Your new project access token has been created.

Project Access Tokens

You can generate an access token scoped to this project for each application to use the GitLab API.

Your new project access token: zH_AQAxrnesN7UEgzNop

Make sure you save it - you won't be able to access it again.

Add a project access token

Enter the name of your application, and we'll return a unique project access token.

Name

Best practice is to create tokens with expiration dates, with as few scopes (permissions) as possible.

<u>For both Personal and Project Access Tokens:</u>

You may need to create a token with API scope in order to push/pull images
- **api + write_registry for pushing**
- **read_api + read_registry for pulling**

Examples

For all of the examples, we assume that we are dealing with:

– a user alice,

– who is part of a group example_group,

– working with a repository example_repository.

– Her password is my_password.

– She has created a Project Access Token in example_repository

  – with the username Test_Tok,

  – giving it api and read_registry scopes,

  – which created the token password `zH_AQRxrnesN8UEgzNop`

–She is working with the centos:centos8 image, originally located on docker.io

Logging in to the GitLab Registry - Docker

- BioHPC Username/Password – <u>AVOID DOING THIS.</u>

```
$ docker login -u "alice" -p "my_password" git.biohpc.swmed.edu:5050
```

- Project Access Token – No leaking of BioHPC credentials

```
$ docker login -u "Test_Tok" -p "zH_AQRxrnesN8UEgzNop" git.biohpc.swmed.edu:5050
```

Push to/Pull from GitLab Registry

- Assuming there's already a local Docker image my_local_image:1.5.123
- Need to re-tag the **local** image with a new tag **indicating the remote destination and name.**
- Before pushing, you must login to the git.biohpc.swmed.edu:5050 endpoint.
- Push the same tag, in full:

```
$ docker tag my_local_image:1.5.123 \
git.biohpc.swmed.edu:5050/example_group/example_repository/my_image:1.5.0
$ docker login -u "Test_Tok" -p "zH_AQRxrnesN8UEgzNop" git.biohpc.swmed.edu:5050
$ docker push git.biohpc.swmed.edu:5050/example_group/example_repository/my_image:1.5.0
```

Note that the local and remote image names (**my_local_image:1.5.123** vs **my_image:1.5.0**) do not need to match.

Further Resources

- GitLab Registry (Our current version of GitLab is 14.8)
  - General usage: https://docs.gitlab.com/14.8/ee/user/packages/container_registry/index.html
- Docker
  - Command line: https://docs.docker.com/engine/reference/commandline/cli/
- Singularity
  - Quick Start: https://sylabs.io/guides/3.9/user-guide/quick_start.html
  - Compatibility with Docker https://sylabs.io/guides/3.9/user-guide/singularity_and_docker.html
- BioHPC
  - Singularity on BioHPC: https://portal.biohpc.swmed.edu/content/guides/singularity-containers-biohpc/
  - Training Slides: https://portal.biohpc.swmed.edu/content/training

"I want to make sure that I have access to this image" → Using GitLab Container Registry as an archive for OCI images

- Working on a machine running Docker:
  - Pull Docker image

```
$ docker pull centos:centos8
```

Implicitly pulls from Docker Hub by default.

  - Tag Docker image

```
$ docker tag centos:centos8 \
git.biohpc.swmed.edu:5050/example_group/example_repository/centos:centos8
```

  - Push new tag

```
$ docker push \
git.biohpc.swmed.edu:5050/example_group/example_repository/centos:centos8
```

“I want to use an image from this public GitLab repository/registry, using Singularity”

- Working on any BioHPC system:
- (Choice 1) Run the image directly, implicitly pulling and caching it. Note the use of docker://

```
$ singularity run \
docker://git.biohpc.swmed.edu:5050/example_group/example_repository/centos:centos8
```

- (Choice 2) pull and run:

```
$ singularity pull local_centos8.sif \
docker://git.biohpc.swmed.edu:5050/example_group/example_repository/centos:centos8
$ singularity run local_centos8.sif
```

If a GitLab repository is public, **so is its container registry, if enabled.**

Pull image from DockerHub, run it, then store the image on the BioHPC GitLab.

- Working on any BioHPC system (with SINGULARITY_DOCKER credentials) :

```
$ singularity pull local_centos8.sif docker://docker.io/centos:centos8
$ singularity run local_centos8.sif
$ singularity push --docker-login local_centos8.sif \
oras://git.biohpc.swmed.edu:5050/example_group/example_repository/centos_sif:centos8
```

If a GitLab repository is public, **so is its container registry, if enabled.**

“I want to use an image from this <u>private</u> GitLab repository/registry, using Singularity”

```
$ singularity pull local_base.sif \
docker://git.biohpc.swmed.edu:5050/example_group/example_repository/centos:centos8
```

will fail with 403 (forbidden), because you haven’t logged in yet.

- Interactive :

```
$ singularity pull --docker-login local_base.sif \
docker://git.biohpc.swmed.edu:5050/example_group/example_repository/centos:centos8
```

- Programmatic :

```
$ export SINGULARITY_DOCKER_USERNAME=“Test_Tok”
$ export SINGULARITY_DOCKER_PASSWORD=“zH_AQRxrnesN8UEgzNop”
$ singularity pull \
docker://git.biohpc.swmed.edu:5050/example_group/example_repository/centos:centos8
```

Further Resources

- Singularity
  - Quick Start: https://sylabs.io/guides/3.0/user-guide/quick_start.html
  - Compatibility with Docker https://sylabs.io/guides/3.0/user-guide/singularity_and_docker.html
- BioHPC
  - Singularity on BioHPC: https://portal.biohpc.swmed.edu/content/guides/singularity-containers-biohpc/
  - Training Slides: https://portal.biohpc.swmed.edu/content/training/training-slides/
- GitLab Registry (Our current version of GitLab is 14.8)
  - General usage: https://docs.gitlab.com/14.8/ee/user/packages/container_registry/index.html
- BioHPC Astrocyte
  - Our workflow platform which allows you to build workflows, including using containers.