OSG

Containers and GPUs

Thursday, July 28 2022

Mats Rynge

This work was supported by NSF grants MPS-1148698, OAC-1836650, and OAC-2030508

Outline

What are containers?

Why containers on the OSPool?

Finding existing containers

Building containers (Remote builder, On your own machine)

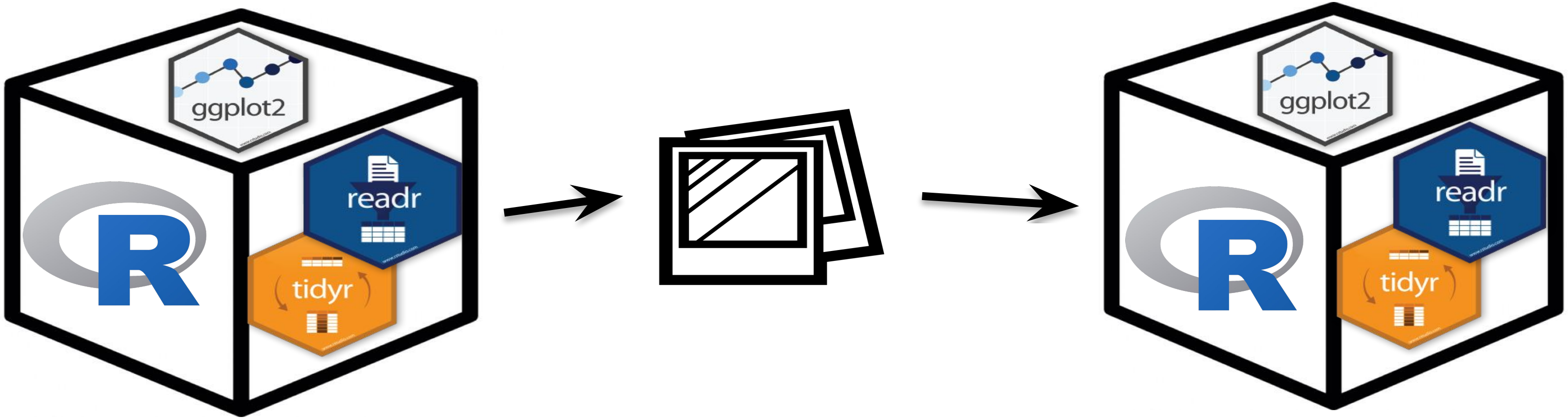
Test container on access points

Use containers in your jobs

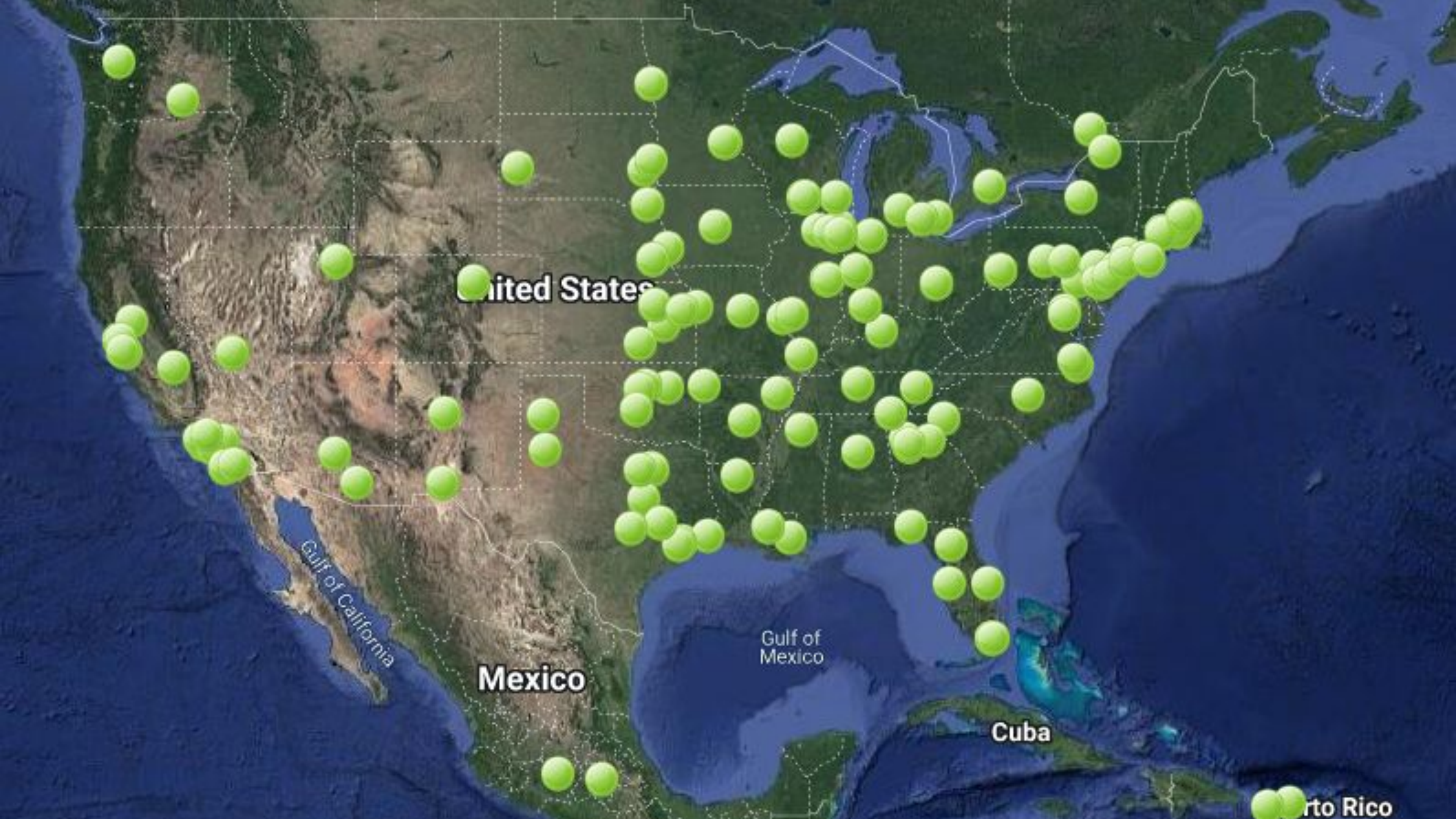
Hey, how are GPUs related to all this?

Containers

Containers are a tool for capturing an entire job “environment” (software, libraries, operating system) into an “image” that can be used again.



polaroid photos by Nick Bluth from the Noun Project


United States
Gulf of California
Mexico
Gulf of Mexico
Cuba
rto Rico

What problems do containers solve for HTC jobs?

Consistent Environment

your job has the same environment where ever it runs.

Complete Environment

everything you need is included

Reproducible Environment

the environment is “software defined” - easy to reproduce if you want to make small changes

Food Analogy!

Running software on your own computer is like cooking in your own kitchen.

Photo by jschantz on flickr, CC-BY

On Your Computer

- You know what you already have.
  - All the software you need is already installed.

- You know where everything is (mostly).

- You have full control.
  - You can add new programs when and where you want.

The Problem

?

Running on a shared computer is like cooking in someone else’s kitchen.

Photo by F Deventhal on Wikimedia, CC-BY

On Someone Else’s Computer

- What’s already there?
  - Is R installed? Or Python? What about the packages you need?
- Do you know where anything is?
- Are you allowed to change whatever you want?

The Solution

- Think like a backpacker.
- Take your software with you
  - Install anywhere
  - Run anywhere
- This is called making software *portable*

Photo by Derrick Mercer on Flickr, CC-BY-SA

Returning to Our Analogy…

Using a container is kind of like bringing along a whole kitchen…

Photo by PunkToad on Flickr, CC-BY

Container Motivations

Consistent environment (default images) - If a user does not specify a specific image, a default one is used by the job. The image contains a decent base line of software, and because the same image is used across all the sites, the user sees a more consistent environment than if the job landed in the environments provided by the individual sites.

Custom software environment (user defined images) - Users can create and use their custom images, which is useful when having very specific software requirements or software stacks which can be tricky to bring with a job. For example: Python or R modules with dependencies, TensorFlow

Enables special environment such as GPUs - Special software environments to go hand in hand with the special hardware.

- -

Process isolation - Sandboxes the job environment so that a job can not peek at other jobs.

File isolation - Sandboxes the job file system, so that a job can not peek at other jobs’ data.

Container Lifecycle (Hint: ephemeral)

Each and every job is encapsulated in a separate container instance

Container instance dies when the job finishes

An incredible amount of container image reuse, as workloads generally use one or a small number of images for a large number of jobs

Outline

~~What are containers?~~

~~Why containers on the OSPool?~~

Finding existing containers

Building containers (Remote builder, On your own machine)

Test container on access points

Use containers in your jobs

Hey, how are GPUs related to all this?

You can use existing containers!

- OSG provided: https://support.opensciencegrid.org/support/solutions/articles/12000073449-view-existing-ospool-supported-containers

- OSG collaboration/user provided (just a list, no descriptions): https://github.com/opensciencegrid/cvmfs-singularity-sync/blob/master/docker_images.txt

- Docker Hub: https://hub.docker.com/ (and there are other hubs!)

Outline

~~What are containers?~~

~~Why containers on the OSPool?~~

~~Finding existing containers~~

Building containers (Remote builder, On your own machine)

Test container on access points

Use containers in your jobs

Hey, how are GPUs related to all this?

Container Types

- Two common container systems:

Docker

https://www.docker.com/

docker

Singularity / Apptainer

https://sylabs.io/

Docker, Singularity (Apptainer)

Containers are defined using Docker or Singularity

- Public Docker Hub
- .sif files

… and executed with Singularity

- No direct access to the Singularity command line - that is controlled by the infrastructure

Docker - Extracted Images

OSG stores Docker container images on CVMFS in extracted form. That is, we take the Docker image layers or the Singularity img/simg files and export them onto CVMFS. For example, ls on one of the containers looks similar to ls / on any Linux machine:

```
$ ls /cvmfs/singularity.opensciencegrid.org/opensciencegrid/osgvo-el7:latest/
cvmfs  host-libs  proc  sys  anaconda-post.log     lib64
dev    media      root  tmp  bin                   sbin
etc    mnt        run   usr  image-build-info.txt  singularity
home   opt        srv   var  lib
```

Result: Most container instances only use **a small part** of the container image **(50-150 MB)** and that part is **cached** in CVMFS!

Singularity has a great security model, but building…

- All containers in the OSPool are invoked with Singularity
  - Source images can be Singularity SIF files, or Docker images

- Singularity containers are run as the user invoking them
  - Non-privileged user, not root

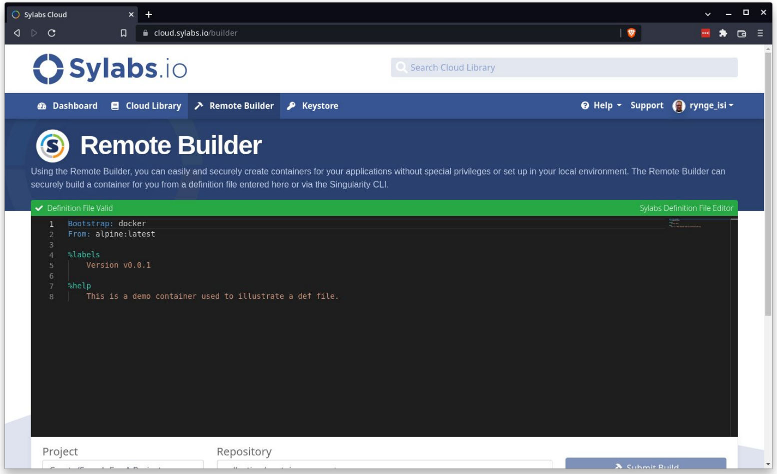
- However, building requires root
  - Use a remote builder hosted by Sylabs (no “private” files / bindmounts)
  - Install it on your own Linux system so you can use sudo

Sylabs Cloud

Create an account on (Google auth is easy):

https://cloud.sylabs.io/

Click on Remote Builder



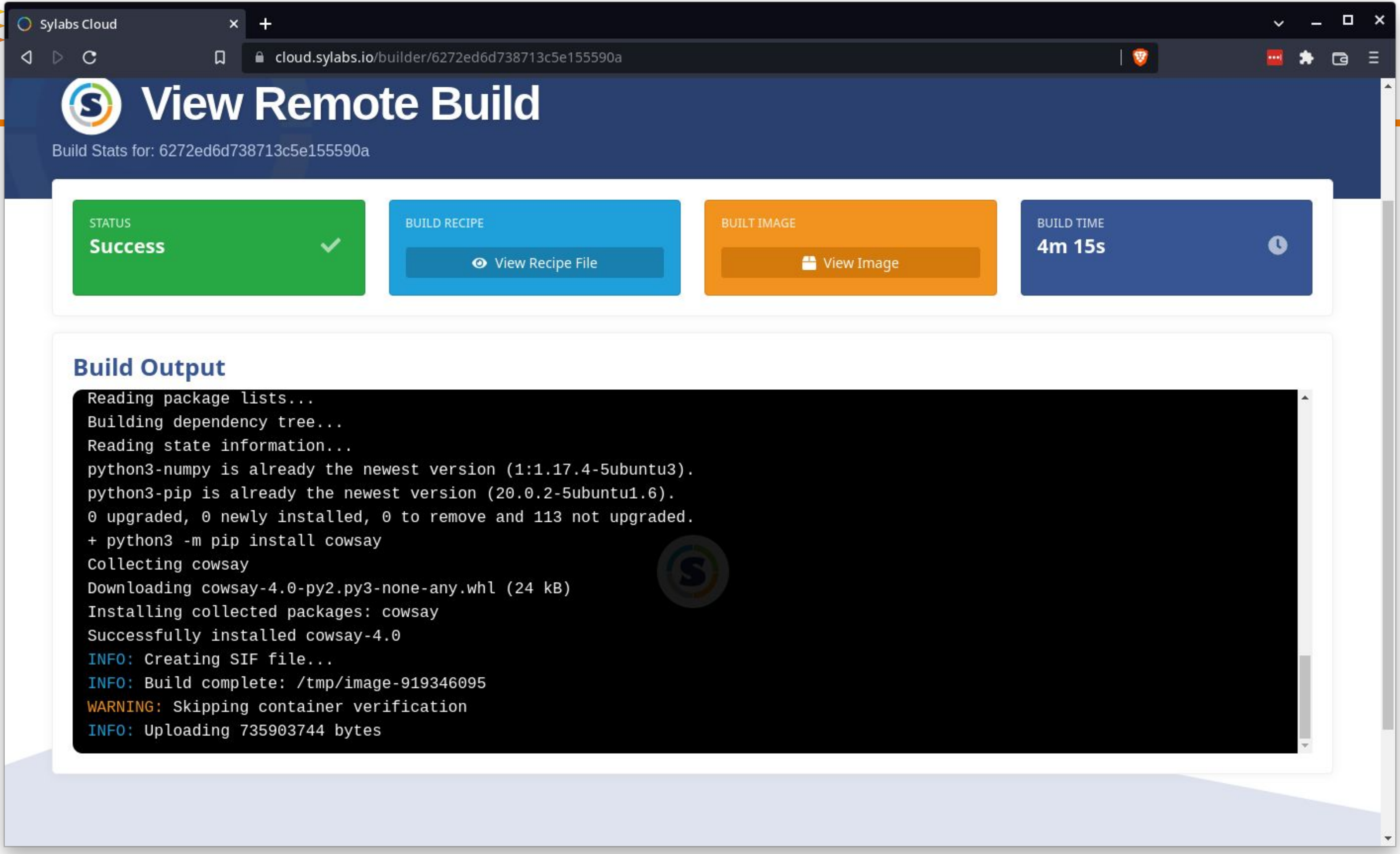
Remote Builder

Using the Remote Builder, you can easily and securely create containers for your applications without special privileges or set up in your local environment. The Remote Builder can securely build a container for you from a definition file entered here or via the Singularity CLI.

```
Bootstrap: docker
From: alpine:latest

%labels
    Version v0.0.1

%help
    This is a demo container used to illustrate a def file.
```

https://sylabs.io/guides/3.8/user-guide/build_a_container.html#building-containers-from-singularityce-definition-files

```
Bootstrap: docker
From: opensciencegrid/osgvo-ubuntu-20.04:latest

%post
    apt-get update -y
    apt-get install -y \
            python3-pip \
            python3-numpy
    python3 -m pip install cowsay
```



View Remote Build

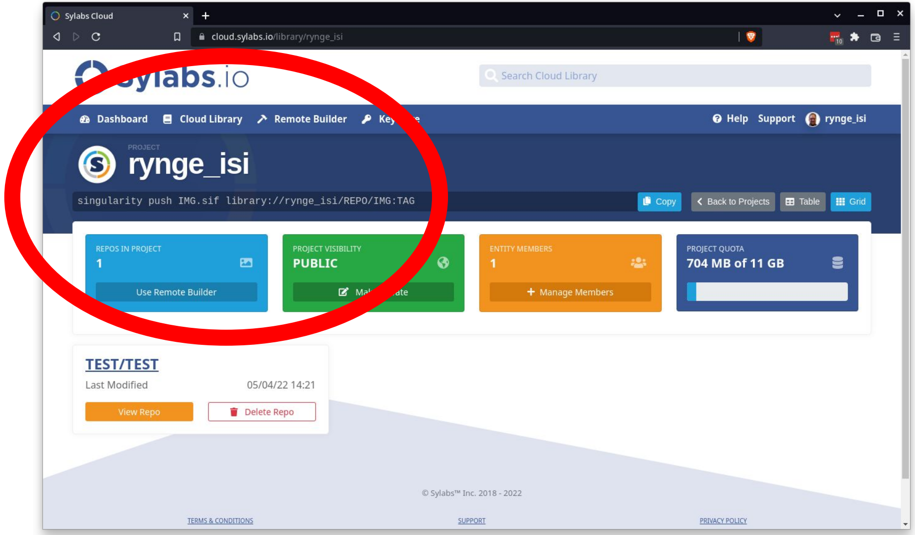
Build Stats for: 6272ed6d738713c5e155590a

Build Output

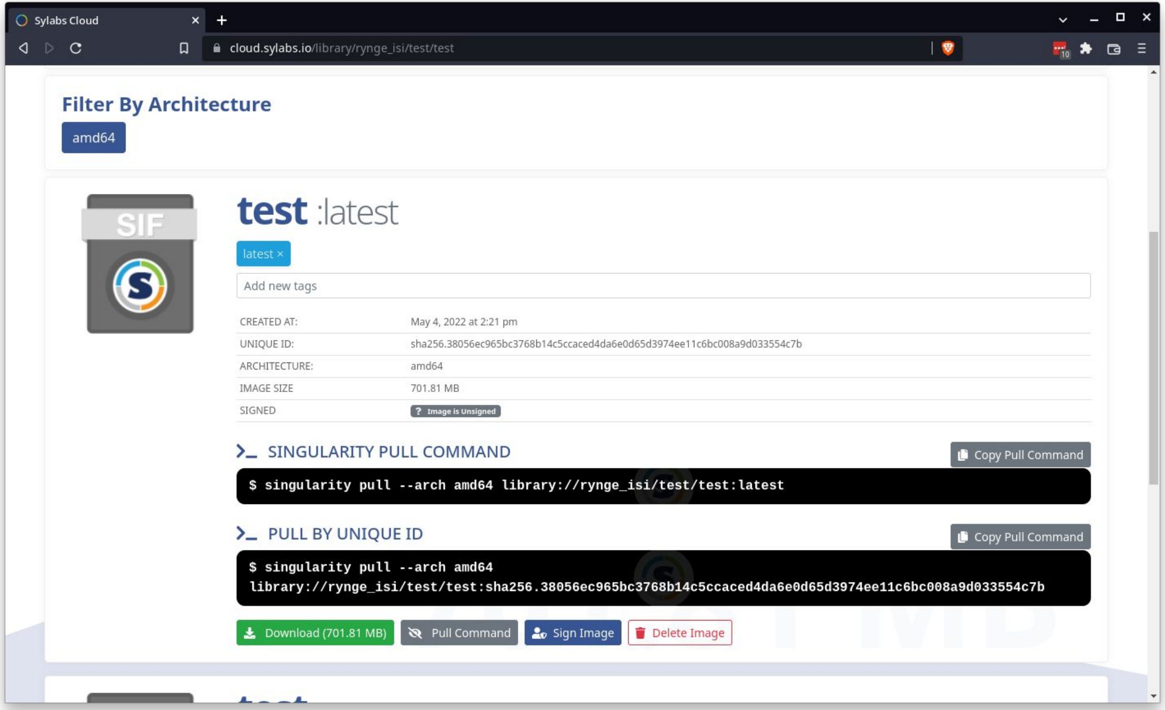
```
Reading package lists...
Building dependency tree...
Reading state information...
python3-numpy is already the newest version (1:1.17.4-5ubuntu3).
python3-pip is already the newest version (20.0.2-5ubuntu1.6).
0 upgraded, 0 newly installed, 0 to remove and 113 not upgraded.
+ python3 -m pip install cowsay
Collecting cowsay
Downloading cowsay-4.0-py2.py3-none-any.whl (24 kB)
Installing collected packages: cowsay
Successfully installed cowsay-4.0
INFO: Creating SIF file...
INFO: Build complete: /tmp/image-919346095
WARNING: Skipping container verification
INFO: Uploading 735903744 bytes
```

Download Image

To download the image, you have to either setup the access tokens, or make the project public:



```
singularity push IMG.sif library://rynge_isi/REPO/IMG:TAG
```



Filter By Architecture

test :latest

CREATED AT:	May 4, 2022 at 2:21 pm
UNIQUE ID:	sha256.38056ec965bc3768b14c5ccaced4da6e0d65d3974ee11c6bc008a9d033554c7b
ARCHITECTURE:	amd64
IMAGE SIZE	701.81 MB
SIGNED	? Image is Unsigned

SINGULARITY PULL COMMAND

```
$ singularity pull --arch amd64 library://rynge_isi/test/test:latest
```

PULL BY UNIQUE ID

```
$ singularity pull --arch amd64
library://rynge_isi/test/test:sha256.38056ec965bc3768b14c5ccaced4da6e0d65d3974ee11c6bc008a9d033554c7b
```

You can use the “pull command” on the OSGConnect access nodes

Download

```
[login05:~] $ singularity pull --arch amd64 \
                library://rynge_isi/test/test:latest


[login05:~] $ ls -lh test_latest.sif
-rwxr-xr-x 1 rynge osg 702M May 16 18:41 test_latest.sif
```

```
[~] $ cat my-container.def
Bootstrap: docker
From: opensciencegrid/osgvo-ubuntu-20.04:latest

%post
   apt-get update -y
   apt-get install -y \
           python3-pip \
           python3-numpy
   python3 -m pip install cowsay


[~] $ sudo singularity build /tmp/my-container.sif my-container.def
```

Only do this on your own machine - it will not work on OSGConnect servers

Outline

~~What are containers?~~

~~Why containers on the OSPool?~~

~~Finding existing containers~~

~~Building containers (Remote builder, On your own machine)~~

Test container on access points

Use containers in your jobs

Hey, how are GPUs related to all this?

Executing containers on access points

Containers can be tried out with the “singularity shell…” command

Your $HOME directory is automatically mounted, which give you an easy way to use your codes/data or compile inside the container, but access the results outside.

As always, do not run long or compute heavy jobs on the access points

```
[login05:~] $ singularity pull --arch amd64 library://rynge_isi/test/test:latest
INFO:    Using cached image

[login05:~] $ singularity shell test_latest.sif
Singularity> python3
Python 3.8.10 (default, Sep 28 2021, 16:10:42)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import cowsay
>>> cowsay.cow('Hello World')
 ___________
| Hello World |
 ===========
          \
           \
             ^__^
             (oo)\_______
             (__)\       )\/\
                 ||----w |
                 ||     ||
```

$HOME is automatically mounted

```
[login05:~] $ singularity shell test_latest.sif
Singularity> pwd
/home/rynge
Singularity> echo "Hello World" >mydata.txt
Singularity> exit
exit
```

Data (or for example executables) generated inside the container instance …

```
[login05:~] $ cat mydata.txt
Hello World
```

… is available “outside” because $HOME was automatically mounted

Outline

~~What are containers?~~

~~Why containers on the OSPool?~~

~~Finding existing containers~~

~~Building containers (Remote builder, On your own machine)~~

~~Test container on access points~~

Use containers in your jobs

Hey, how are GPUs related to all this?

Should you stash?

Option 1: put your .sif image anywhere under $HOME and just refer to it in the +SingularityImage attribute:

```
    +SingularityImage = “./test_latest.sif”
```

Downside is that every job has to transfer the full image. Only do this for small sets of jobs.

Option 2: put your .sif image under /public/$USERNAME/ and prepend stash:///osgconnect/. Example:

```
    +SingularityImage = “stash:///osgconnect/public/u/test_latest.sif”
```

Image is automatically cached.

Outline

~~What are containers?~~

~~Why containers on the OSPool?~~

~~Finding existing containers~~

~~Building containers (Remote builder, On your own machine)~~

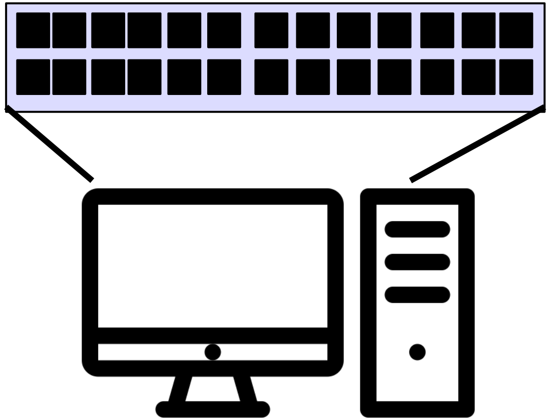
~~Test container on access points~~

~~Use containers in your jobs~~

Hey, how are GPUs related to all this?

What is a GPU?

- GPU = Graphical Processing Unit
- Has hundreds to thousands of “cores” that can be used to parallelize work.



Created by Idealogo Studio
from Noun Project

GPU Use Cases

- Programs that map well to GPUs include:
  - Deep learning
  - Molecular dynamics
  - Anything with lots of number crunching (like matrix operations) and low(er) data load.

GPUs on the OSG

- Scale: 100s (vs 10,000s of CPUs)
- Variety of available GPU cards

- Same restrictions as always: not sure what you'll get, jobs can be interrupted

- May take longer to start

Making robust GPU jobs

- Use a software strategy that can run on different GPU types:
  - Container
  - Install inside the job

- OR use job requirements to request certain kind of GPU (more limiting)

Submit File options

- Request GPUs with “request_gpus”
- Can use custom requirements

```
request_gpus = 1

requirements = (CUDACapability >= 6.0)
```

End