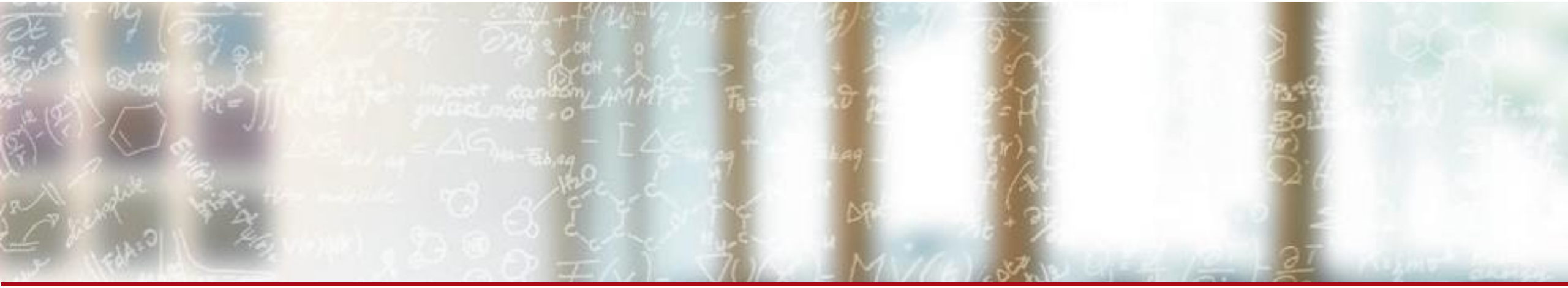
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



HPC Containers at CSCS – New features & enhancements

CSCS User Lab Day – Meet the Swiss National Supercomputing Center

Manitaras Theofilos-Ioannis, CSCS

August 31, 2020

Outline

- Container Basics
- Introduction to Docker
- Using Sarus at CSCS
- Using Singularity at CSCS
- Conclusions

Container Basics

Containers in a nutshell

What are Containers?

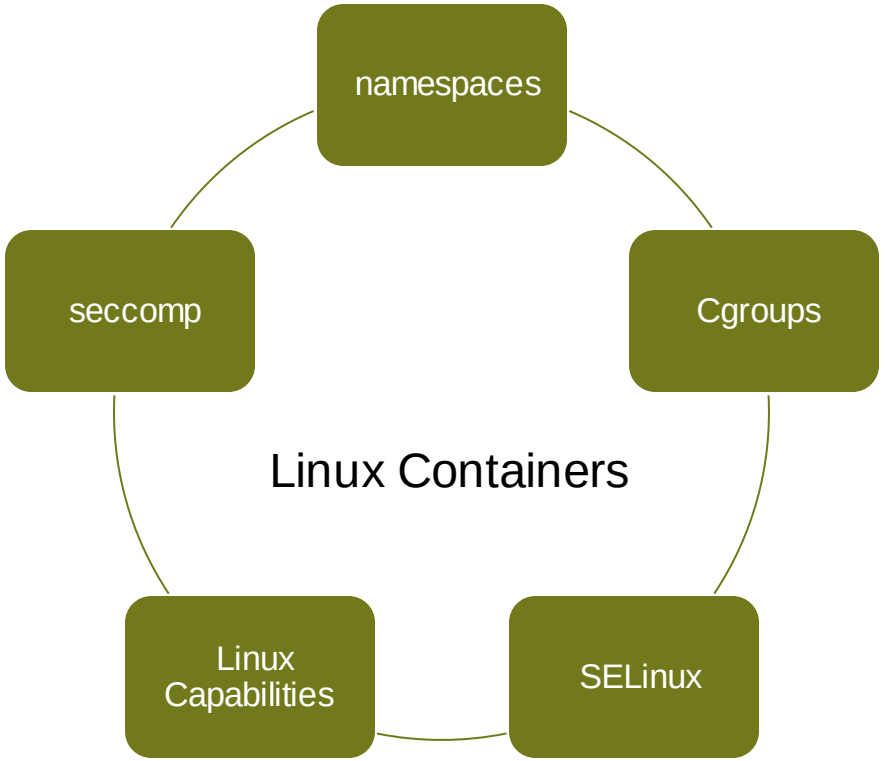
- Wikipedia (general container): *A container is any receptacle or enclosure for holding a product used in **storage**, **packaging**, and **shipping**. Things kept inside of a container are **protected** by being inside of its structure*
- Docker: *A container is a standard unit of software that **packages up code and all its dependencies,** so the application runs quickly and reliably from one computing environment to another*
- Google Cloud: *Containers offer a **logical packaging mechanism** in which applications can be abstracted from the environment in which they actually run*
- AWS: *Containers are a method of **operating system virtualization** that allow you to run an application and its dependencies in **resource-isolated processes***

Linux Containers under the hood

- Linux namespaces: Control what the process (container) can "see"
- cgroups: limit and monitor the resources that the process (container) can use

Security

- SELinux: Security-Enhanced Linux
- Linux capabilities: restrict allowed syscalls
- seccomp: Secure Computing



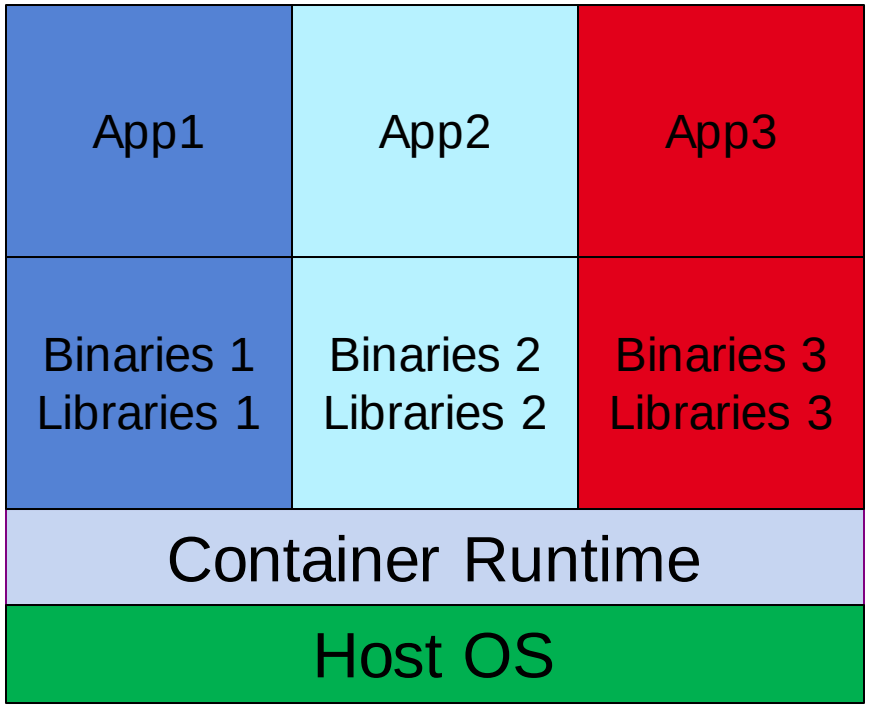
Linux Namespaces

The Linux namespaces used in containers are:

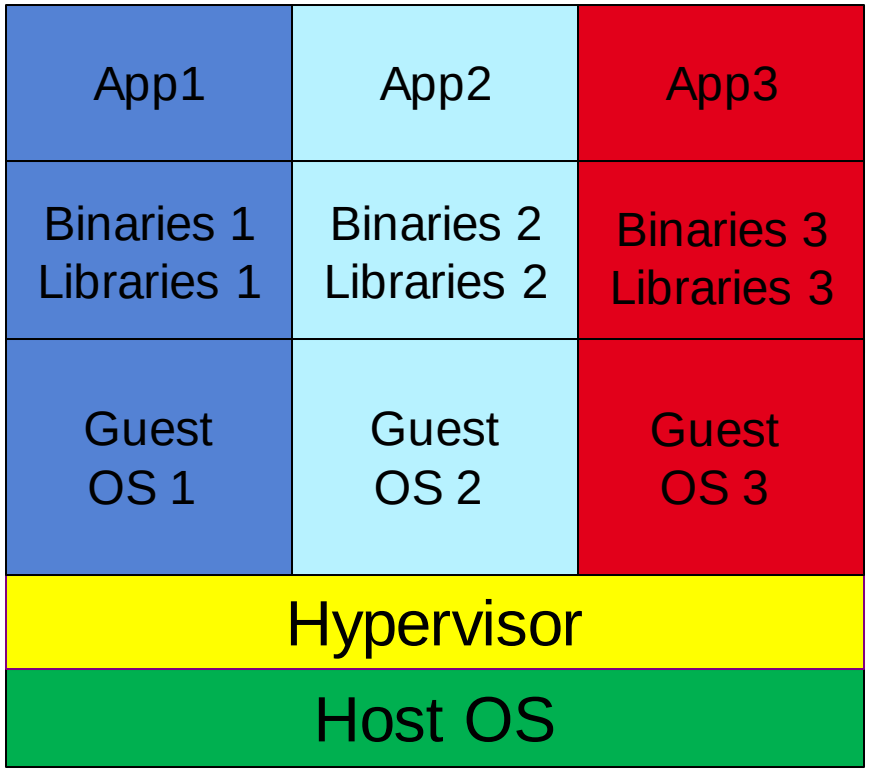
- Mount namespace
- UTS namespace
- IPC namespace
- Network namespace
- Pid namespace
- User namespace

- There are additional namespaces and new ones might be introduced

Containers vs Virtual Machines



Containers



Virtual Machines

Image registries, repositories, tags

- An **image registry** is a service where container images are stored, (e.g DockerHub, Nvidia Container Registry, Quay)
- Image **repositories** are collections of different images sharing the same name
- Image repositories can be grouped under **organizations**
- Image **tags** are used to differentiate between the different image versions
- A specific image, is identified using the following convention:

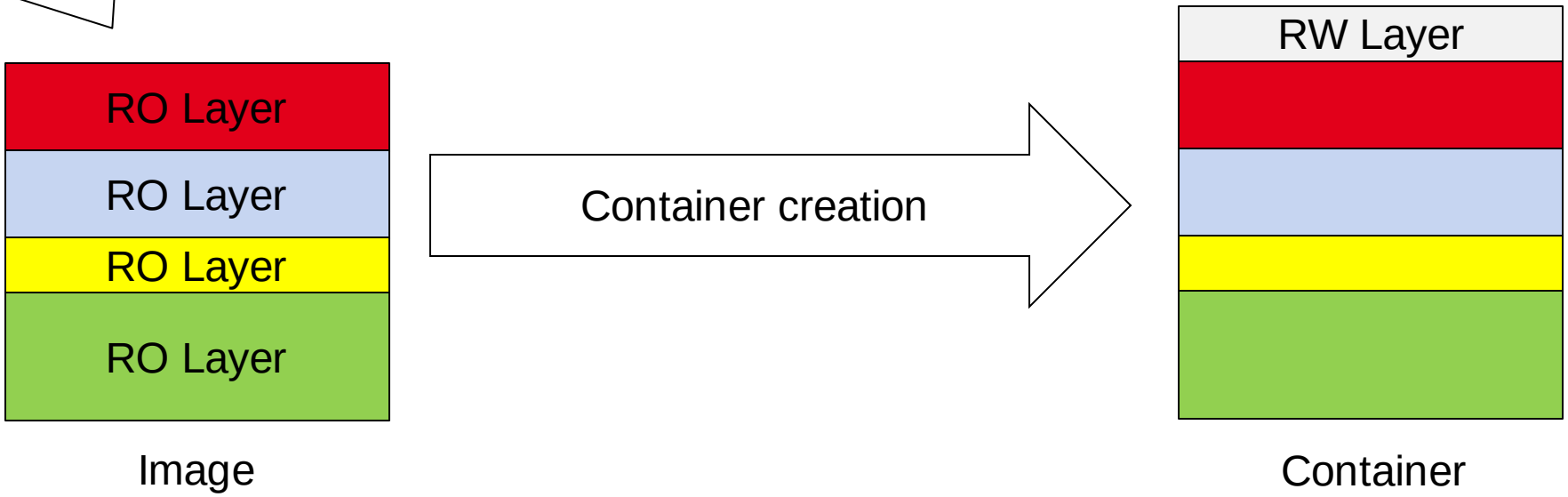
  [registry url]/[organization]/<repository>:[tag] (fields inside square brackets are optional)
  e.g from Nvidia Container Registry (nvcr.io/nvidia/tensorflow:20.03-tf2-py3)

- If no registry is specified, Docker uses DockerHub
- If no tag is given, Docker assumes the “latest” tag

Container images vs running containers

A container **image** consists of a series of read-only layers, each of them corresponding to a step during the image build.

When a new **container** is created, a new writable layer (container layer) is added on top of the underlying layers.



CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

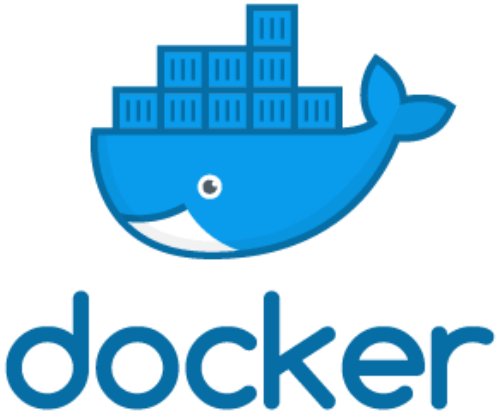
ETH zürich

Introduction to Docker

What is Docker?

Docker is a container "ecosystem" including various software components:

- **Docker Engine** for building-running-shipping containers
- **DockerHub** the main container registry
- **Docker-compose** for multi-container scenarios
- **Docker Swarm** for container orchestration



Furthermore, it defines:

- A container image format
- A container registry api
- The Docker Engine API

Docker Hello-World

```
 $ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
0e03bdcc26d7: Pull complete
Digest: sha256:8e3114318a995a1ee497790535e7b88365222a21771ae7e53687ad76563e8e76
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

All running containers use the host kernel

Here we use: docker run --name <container_name> <image> <command>

```
 $ docker run --name container1 ubuntu:latest uname -rv
4.15.0-96-generic #97~16.04.1-Ubuntu SMP Wed Apr 1 03:03:31 UTC 2020
 $ docker run --name container2 ubuntu:latest uname -rv
4.15.0-96-generic #97~16.04.1-Ubuntu SMP Wed Apr 1 03:03:31 UTC 2020
 $
 $ docker run --name container3 fedora:latest uname -rv
4.15.0-96-generic #97~16.04.1-Ubuntu SMP Wed Apr 1 03:03:31 UTC 2020
 $ docker run --name container4 fedora:latest uname -rv
4.15.0-96-generic #97~16.04.1-Ubuntu SMP Wed Apr 1 03:03:31 UTC 2020
 $
 $ docker run --name container5 opensuse/tumbleweed uname -rv
4.15.0-96-generic #97~16.04.1-Ubuntu SMP Wed Apr 1 03:03:31 UTC 2020
 $ docker run --name container6 opensuse/tumbleweed uname -rv
4.15.0-96-generic #97~16.04.1-Ubuntu SMP Wed Apr 1 03:03:31 UTC 2020
 $
 $ docker run --name container7 alpine uname -rv
4.15.0-96-generic #97~16.04.1-Ubuntu SMP Wed Apr 1 03:03:31 UTC 2020
 $ docker run --name container8 alpine uname -rv
4.15.0-96-generic #97~16.04.1-Ubuntu SMP Wed Apr 1 03:03:31 UTC 2020
```

Running containers interactively

In order to run a container interactively, use: docker run -it <image>

```
$ docker run -it ubuntu
root@2d2311497e00:/# cat /etc/os-release
NAME="Ubuntu"
VERSION="20.04 LTS (Focal Fossa)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 20.04 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
root@2d2311497e00:/# echo 'Hello from Ubuntu'
Hello from Ubuntu
root@2d2311497e00:/# whoami
root
root@2d2311497e00:/# exit
exit
$
```

Running **ubuntu** container interactively

```
$ docker run -it python
Python 3.8.2 (default, Apr 21 2020, 14:18:20)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> print('Hello from Python')
Hello from Python
>>>
>>> import socket, platform
>>>
>>> socket.gethostname()
'694e3ccd5a74'
>>>
>>> platform.platform()
'Linux-4.15.0-96-generic-x86_64-with-glibc2.2.5'
>>>
>>> exit()
$
```

Running **python** container interactively

Useful Docker cli commands

- List running containers: `docker ps`
- List all containers: `docker ps -a`
- List all the images: `docker images`
- Remove an image: `docker rmi <image_name>`
- Pull an image from DockerHub (default to tag "latest"):

  `docker pull <image_name>`
- Run a container with a specified name:

  `docker run --name <container_name> <image_name>`
- Save an image as a tar archive:

  `docker save <image_name> -o <image_arhive.tar>`

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

HPC Containers with Sarus

Introduction to Sarus

- Sarus is an OCI-compatible container engine for HPC
- It is developed at CSCS and is driven by the specific requirements of HPC systems
- It is extensible via OCI hooks to take advantage of custom hardware and achieve native performance
- Compatible with the workload managers used in HPC systems
- Allows pulling container images from registries adopting the OCI Distribution Specification or the Docker Registry HTTP API V2 protocol
- Can import images from image archives (e.g those created via **docker save**)
- Supports creation of container filesystems tailored for diskless nodes and parallel filesystems

Pulling images from container registries

Sarus can pull container images directly from registries using the **sarus pull** command. If no registry is specified, sarus pulls from DockerHub:

```
sarus_user@daint> module load daint-gpu
sarus_user@daint> module load sarus
sarus_user@daint> srun -C gpu -u sarus pull ubuntu:latest
srun: job 25318865 queued and waiting for resources
srun: job 25318865 has been allocated resources
# image            : index.docker.io/library/ubuntu:latest
# cache directory  :
# temp directory   :
# images directory :
> save image layers ...
> found in cache : sha256:f7bfea53ad120b47cea5488f0b8331e737a97b33003517b0bd05e83925b578f0
> found in cache : sha256:b66c17bbf772fa072c280b10fe87bc999420042b5fce5b111db38b4fe7c40b49
> found in cache : sha256:46d371e02073acecf750a166495a63358517af793de739a51b680c973fae8fb9
> found in cache : sha256:54ee1f796a1e650627269605cb8e6a596b77b324e6f0a1e4443dc41def0e58a6
> expanding image layers ...
> extracting     :
> extracting     :
> extracting     :
> extracting     :
> make squashfs image:
```

Running a container using Sarus

In order to run a container based on an image that is already pulled, the command **sarus run** is used.

The full syntax of the command is: **sarus run <image_name> <command>**

```
sarus_user@daint> srun -C gpu sarus run ubuntu:latest cat /etc/os-release
srun: job 25318876 queued and waiting for resources
srun: job 25318876 has been allocated resources
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
```

Loading images from Docker archives

Sarus can load container images from docker tar archives using **sarus load**:

```
sarus_user@daint> module load daint-gpu
sarus_user@daint> module load sarus
sarus_user@daint> srun -C gpu -u sarus load cuda_device_query.tar cuda_device_query
srun: job 25319088 queued and waiting for resources
srun: job 25319088 has been allocated resources
> expanding image layers ...
> extracting     : "/tmp/expansion-directory-wftusgtyuqtudwyb/c009908e6d9966fa9df4e8e78e20f55fca6dd604da84e7d67
"
> extracting     : "/tmp/expansion-directory-wftusgtyuqtudwyb/733dc421fb16a1a9d32c80eded33338fbc8431f9844e4ff5f
"
> extracting     : "/tmp/expansion-directory-wftusgtyuqtudwyb/1a72e91c46c04ed28fc0af33a68e4debc535dbc6fa68cfb95
"
> extracting     : "/tmp/expansion-directory-wftusgtyuqtudwyb/59ec8c26693b0e42d7883ab013d5ad085f4685b03ec2a2011
"
> extracting     : "/tmp/expansion-directory-wftusgtyuqtudwyb/91ea232374a91a969e60bb0cf763434bad17b30381856da3e
"
```

```
sarus_user@daint> module load daint-gpu
sarus_user@daint> module load sarus
sarus_user@daint> srun -C gpu -u sarus load osu_mpich_pt2pt.tar osu_mpich_pt2pt
srun: job 25320179 queued and waiting for resources
srun: job 25320179 has been allocated resources
> expanding image layers ...
> extracting     : "/tmp/expansion-directory-vbyasrdqobjzxnje/aac62d938efd4e2fab42fb57a7d06afa78c826ed379453f
"
> extracting     : "/tmp/expansion-directory-vbyasrdqobjzxnje/6f9033ea492262c8ab5bfe4823535249aecce3d2977f59c
"
> extracting     : "/tmp/expansion-directory-vbyasrdqobjzxnje/a6c722e0c379204c7d87b1234546538365348bbb6dd1969
"
> extracting     : "/tmp/expansion-directory-vbyasrdqobjzxnje/f06fca19fcccb65a15e75f65b5e0c9505a1bc677e7d8228
"
> make squashfs image: "[redacted].sarus/images/load/library/osu_mpich_pt2pt/latest.squashfs"
```

Sample Dockerfile (GPU)

```
FROM nvidia/cuda:10.1-devel

RUN apt-get update  && \
    apt-get install -y git -q && \
    git clone https://github.com/NVIDIA/cuda-samples.git  /usr/local/cuda_samples  && \
    cd /usr/local/cuda_samples  && \
    git fetch origin --tags && \
    git checkout 10.1.2 && \
    make

CMD /usr/local/cuda_samples/Samples/deviceQuery/deviceQuery
```

Running a GPU container

Running a gpu-enabled container is straightforward, since sarus mounts the required drivers inside the container:

```
sarus_user@daint> srun -C gpu sarus run load/library/cuda device query:latest
srun: job 25319098 queued and waiting for resources
srun: job 25319098 has been allocated resources
/usr/local/cuda_samples/Samples/deviceQuery/deviceQuery Starting...

 CUDA Device Query (Runtime API) version (CUDART static linking)

Detected 1 CUDA Capable device(s)

Device 0: "Tesla P100-PCIE-16GB"
  CUDA Driver Version / Runtime Version          10.1 / 10.1
  CUDA Capability Major/Minor version number:    6.0
  Total amount of global memory:                 16281 MBytes (17071734784 bytes)
  (56) Multiprocessors, ( 64) CUDA Cores/MP:     3584 CUDA Cores
  GPU Max Clock rate:                            1329 MHz (1.33 GHz)
  Memory Clock rate:                             715 Mhz
  Memory Bus Width:                              4096-bit
  L2 Cache Size:                                 4194304 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(131072), 2D=(131072, 65536), 3D=(16384, 16384, 16384)
  Maximum Layered 1D Texture Size, (num) layers  1D=(32768), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(32768, 32768), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
```

Sample Dockerfile (MPI)

```
FROM debian:jessie

RUN apt-get update && \
    apt-get install -y ca-certificates file g++ gcc gfortran make gdb strace realpath wget --no-install-recommends

RUN wget -q http://www.mpich.org/static/downloads/3.1.4/mpich-3.1.4.tar.gz && \
    tar xf mpich-3.1.4.tar.gz && \
    cd mpich-3.1.4 && ./configure --disable-fortran --enable-fast=all,O3 --prefix=/usr && \
    make -j$(nproc) && make install && ldconfig

RUN wget -q http://mvapich.cse.ohio-state.edu/download/mvapich/osu-micro-benchmarks-5.3.2.tar.gz && \
    tar xf osu-micro-benchmarks-5.3.2.tar.gz && \
    cd osu-micro-benchmarks-5.3.2 && \
    ./configure --prefix=/usr/local CC=$(which mpicc) CFLAGS=-O3 && \
    make && make install && cd .. && \
    rm -rf osu-micro-benchmarks-5.3.2 && \
    rm osu-micro-benchmarks-5.3.2.tar.gz

CMD /usr/local/libexec/osu-micro-benchmarks/mpi/pt2pt/osu_bw
```

Running an mpi-based container

In order to run an mpi-based container, the **--mpi** command line option should be used. Sarus is going to replace the mpi dynamic library of the image with the host one:

```
sarus_user@daint> srun -C gpu -u -N2 sarus run --mpi load/library/osu_mpich_pt2pt:latest
srun: job 25320272 queued and waiting for resources
srun: job 25320272 has been allocated resources
Detected glibc 2.19 (< 2.26) in the container. Replacing it with glibc 2.26 from the host. Please consi
 to a distribution with glibc >= 2.26.
Detected glibc 2.19 (< 2.26) in the container. Replacing it with glibc 2.26 from the host. Please consi
 to a distribution with glibc >= 2.26.
# OSU MPI Bandwidth Test v5.3.2
# Size      Bandwidth (MB/s)
1                       1.64
2                       3.26
4                       6.59
8                      13.30
16                     26.81
32                     53.47
64                    107.51
128                   218.80
256                   425.35
512                   835.04
1024                 1262.01
2048                 1931.20
4096                 2672.74
8192                 6173.91
16384                8680.64
32768                9265.38
65536                9601.15
131072               9759.88
262144               9859.09
524288               9905.06
1048576              9927.35
2097152              9931.24
4194304              9848.71
```

Additional Sarus features

- Sarus supports pulling container images from container registries requiring authentication via the **--login** option of **sarus pull**. The user is then required to enter the credentials for the specific registry.
- It is straightforward to mount host directories inside a running container using the **--mount** command line option of **sarus run:**

  (e.g **sarus run --mount=type=bind,src=<src_dir>,target=<target_dir>**)
- To list the images currently downloaded, use: **sarus images**
- To remove an image use: **sarus rmi <image_name>**
- For more information on Sarus, refer to the official documentation and the Sarus Cookbook which contains representative HPC use cases

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

HPC Containers with Singularity

Introduction to Singularity

- Singularity is a container platform created to run applications on HPC clusters in a simple, portable and reproducible way
- Singularity is open source and it's official repository is available on GitHub
- It is developed with security in mind, "allowing untrusted users to run untrusted containers in a trusted way"
- It uses the Singularity Image Format(SIF) making container images easy to transport and share
- It allows you to build container images using Singularity Definition Files (not supported on Piz Daint)
- It supports pulling OCI-based images and converts them to the SIF format

Pulling container images from image registries

Singularity can pull container images directly from registries using various forms of the **singularity pull** command, e.g :

```
singularity_user@daint> module load daint-gpu
singularity_user@daint> module load singularity/3.5.3
singularity_user@daint> srun -C gpu -u singularity pull docker://python:latest
srun: job 25318713 queued and waiting for resources
srun: job 25318713 has been allocated resources
[redacted]
INFO:    Converting OCI blobs to SIF format
INFO:    Starting build...
[redacted]
Getting image source signatures
```

```
singularity_user@daint> srun -C gpu -u singularity pull library://ubuntu:latest
srun: job 25318743 queued and waiting for resources
srun: job 25318743 has been allocated resources
INFO:    Downloading library image
 26.81 MiB / 26.81 MiB  100.00% 2.39 MiB/s 11s01ss9s
WARNING: unable to verify container: ubuntu_latest.sif
WARNING: Skipping container verification
```

Building container images using Singularity definition files(1/2)

Singularity allows building container images based on **Singularity definition files**. This in general requires elevated privileges and is not supported on Piz Daint:

```
Bootstrap: docker
From: nvidia/cuda:10.1-devel

%post
    apt-get update -q
    apt-get install -y git -q
    git clone https://github.com/NVIDIA/cuda-samples.git /usr/local/cuda_samples
    cd /usr/local/cuda_samples
    git fetch origin --tags
    git checkout 10.1.2
    make

%runscript
    /usr/local/cuda_samples/Samples/deviceQuery/deviceQuery
```

Building container images using Singularity definition files(2/2)

To build an image based on a Singularity definition file the **singularity build** command is used (requiring elevated privileges):

```
singularity_user@my_computer> sudo singularity  build device_query.sif device_query.def
INFO:    Starting build...
Getting image source signatures
Copying blob 7ddbc47eeb70 skipped: already exists
Copying blob c1bbdc448b72 skipped: already exists
Copying blob 8c3b70e39044 skipped: already exists
Copying blob 45d437916d57 skipped: already exists
Copying blob d8f1569ddae6 skipped: already exists
Copying blob 85386706b020 skipped: already exists
Copying blob ee9b457b77d0 skipped: already exists
Copying blob be4f3343ecd3 skipped: already exists
Copying blob 30b4effda4fd [--------------------------------------] 0.0b / 0.0b
Copying config cf9ecebf7b done
Writing manifest to image destination
Storing signatures
2020/08/29 22:15:55  info unpack layer: sha256:7ddbc47eeb70dc7f08e410a6667948b87ff3883024
2020/08/29 22:15:56  info unpack layer: sha256:c1bbdc448b7263673926b8fe2e88491e5083a8b4b0
2020/08/29 22:15:56  info unpack layer: sha256:8c3b70e3904492c753652606df4726430426f42ea5
2020/08/29 22:15:56  info unpack layer: sha256:45d437916d5781043432f2d72608049dcf74ddbd27
2020/08/29 22:15:56  info unpack layer: sha256:d8f1569ddae616589c5a2dabf668fadd250ee9d892
```

Running Gpu-enabled containers

To run an Cuda-enabled container the **--nv** command line option of **singularity run** has to be used:

```
singularity_user@daint> module load daint-gpu
singularity_user@daint> module load singularity
singularity_user@daint> srun -C gpu singularity run --nv device_query.sif
srun: job 25337409 queued and waiting for resources
srun: job 25337409 has been allocated resources
/usr/local/cuda_samples/Samples/deviceQuery/deviceQuery Starting...

 CUDA Device Query (Runtime API) version (CUDART static linking)

Detected 1 CUDA Capable device(s)

Device 0: "Tesla P100-PCIE-16GB"
  CUDA Driver Version / Runtime Version          10.1 / 10.1
  CUDA Capability Major/Minor version number:    6.0
  Total amount of global memory:                 16281 MBytes (17071734784 bytes)
  (56) Multiprocessors, ( 64) CUDA Cores/MP:     3584 CUDA Cores
  GPU Max Clock rate:                            1329 MHz (1.33 GHz)
  Memory Clock rate:                             715 Mhz
  Memory Bus Width:                              4096-bit
  L2 Cache Size:                                 4194304 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(131072), 2D=(131072, 65536), 3D=(16384, 16384, 16384)
  Maximum Layered 1D Texture Size, (num) layers  1D=(32768), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(32768, 32768), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
```

Running mpi-based containers

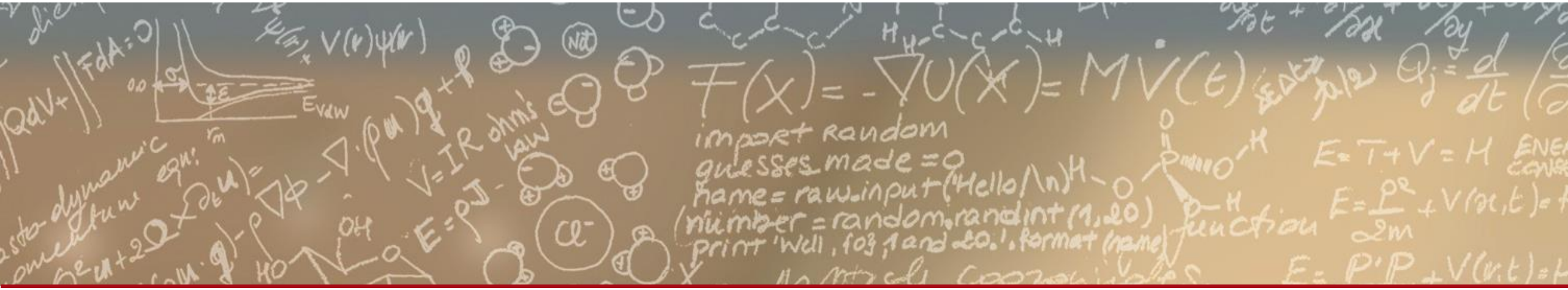
CSCS offers the module **singularity/3.5.3-daint** which defines the bind mounts and the environment variables to mount the host mpi in the container:

```
singularity_user@daint> module load daint-gpu
singularity_user@daint> module load singularity/3.5.3-daint
singularity_user@daint> srun -C gpu -N2 singularity run osu_mpich_pt2pt.sif
srun: job 25337426 queued and waiting for resources
srun: job 25337426 has been allocated resources
# OSU MPI Bandwidth Test v5.3.2
# Size      Bandwidth (MB/s)
1                    1.71
2                    3.41
4                    6.90
8                   13.77
16                  26.74
32                  55.16
64                 110.11
128                219.33
256                427.20
512                830.85
1024              1252.58
2048              1901.48
4096              2597.54
8192              6494.27
16384             8796.09
32768             9240.56
65536             9577.11
131072            9712.87
262144            9835.66
524288            9883.35
1048576           9901.42
2097152           9910.85
4194304           9836.91
```

Conclusions

Conclusions

- Containers bundle software and dependencies in a single portable package
- Easier, predictable and automation friendly deployment
- Reproducible behavior in different computing environments
- Fast start-up (milliseconds-few seconds vs seconds-minutes for VMs)
- Sarus and Singularity address the specific requirements of HPC environments
- Using the above applications, you can develop your software on your local computer and run at scale on Piz Daint



Thank you for your attention.