CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Opportunities for container environments on Cray XC30 with GPU devices

Cray User Group 2016, London

Sadaf Alam, Lucas Benedicic, T. Schulthess, Miguel Gila

May 12, 2016

Agenda

- Motivation
- Container technologies, Docker & Shifter
- Use case: High Energy Physics with containers on XC
- Use case: GPUs with containers on XC
- Conclusion

Motivation

Why do we at CSCS want to use containers on HPC?

- Containerizing applications provides an easy and portable way for packaging complex application setups
  - i.e. libraries, OS dependencies, etc.

- This allows us to support on our Cray systems a wider range of scientific applications and workflows
  - Reach communities beyond the common HPC use cases

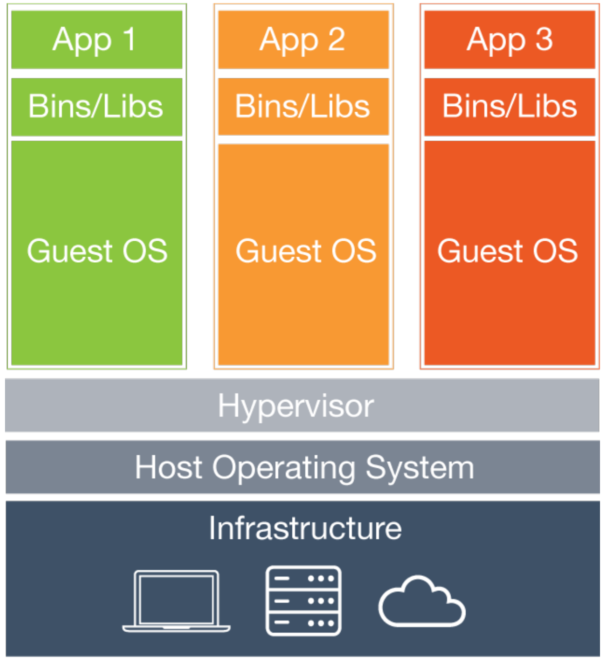
- This can also help consolidating our users and customers on fewer but elastic resources

- But, containers are not the solution for everything. Existing HPC workloads don't need to be containerized

Container technologies: Docker

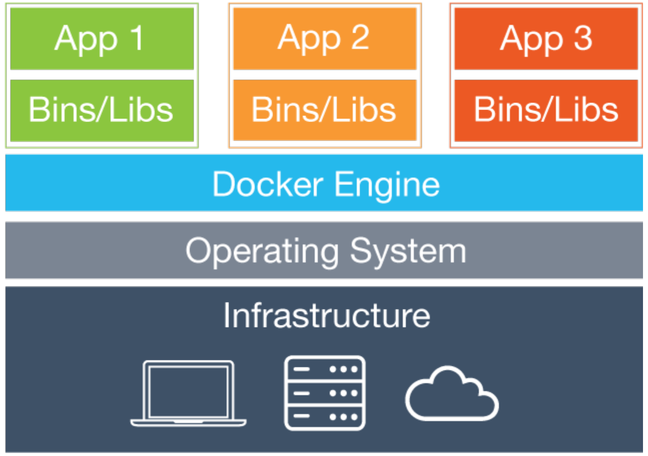
What is Docker and how does it work?

- "Docker containers wrap up a piece of software in a complete filesystem that contains everything it needs to run: code, runtime, system tools, system libraries – anything you can install on a server. This guarantees that it will always run the same, regardless of the environment it is running in." [*]



[*]

Virtual Machines



[*]

Containers

[*] https://www.docker.com/what-docker

What is Docker and how does it work?

- A Docker image is a file that contains a bunch of files on it. Usually libraries and application binaries
- Docker leverages the namespaces feature of the kernel to isolate processes

- On its simplest form, Docker basically
  1. Pulls an image to the local system
  2. Creates some sort of chroot environment with the image (=container)
  3. Runs our application in the container ('isolated' from the host thanks to kernel namespaces)

- However, it can also do *other* things:
  - Isolate network by creating NAT or bridge devices
  - *Can* use a *nice GUI*

Docker in HPC environments

- Docker is a nice tool, but it's not built for HPC environments, because:
  - Does not integrate well with workload managers
  - <u>Does not isolate users on shared filesystems</u>
  - Requires running a daemon on all nodes
  - Not designed to run on diskless clients
  - Network is by default NAT
  - Building Docker is done **within** a Docker container. It can be done outside, but is a complex task (Go language, seriously??)

Chicken or Egg?

- But after all, a sysadmin can make *anything* to work on a cluster, right?
  - We can create (and hopefully maintain) **monstrous wrappers** to run Docker containers…

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Container technologies: Shifter

What is Shifter and how does it work?

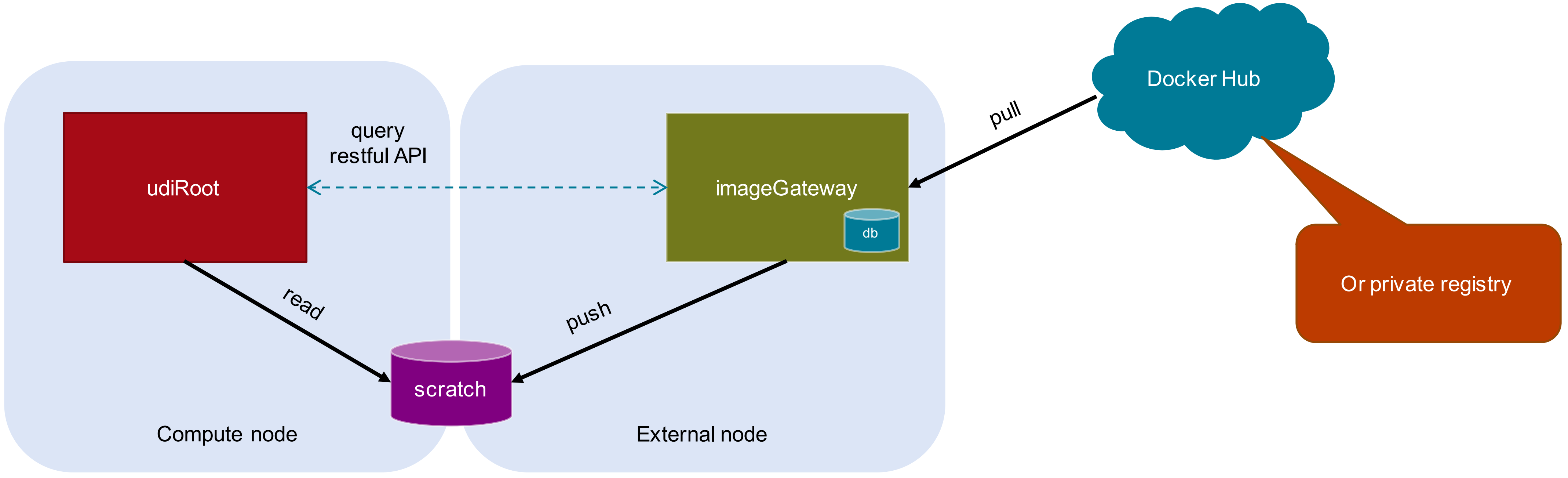
- Shifter is a container-based solution thought from the ground up for HPC environments and, in particular, Cray systems
- Open source tool created by NERSC. Available on Github
- It leverages the current Docker environment by using Docker images to create containers
- Shifter uses loop devices and chroot mechanisms as well as namespaces to provide the container environment

Shifter in HPC environments

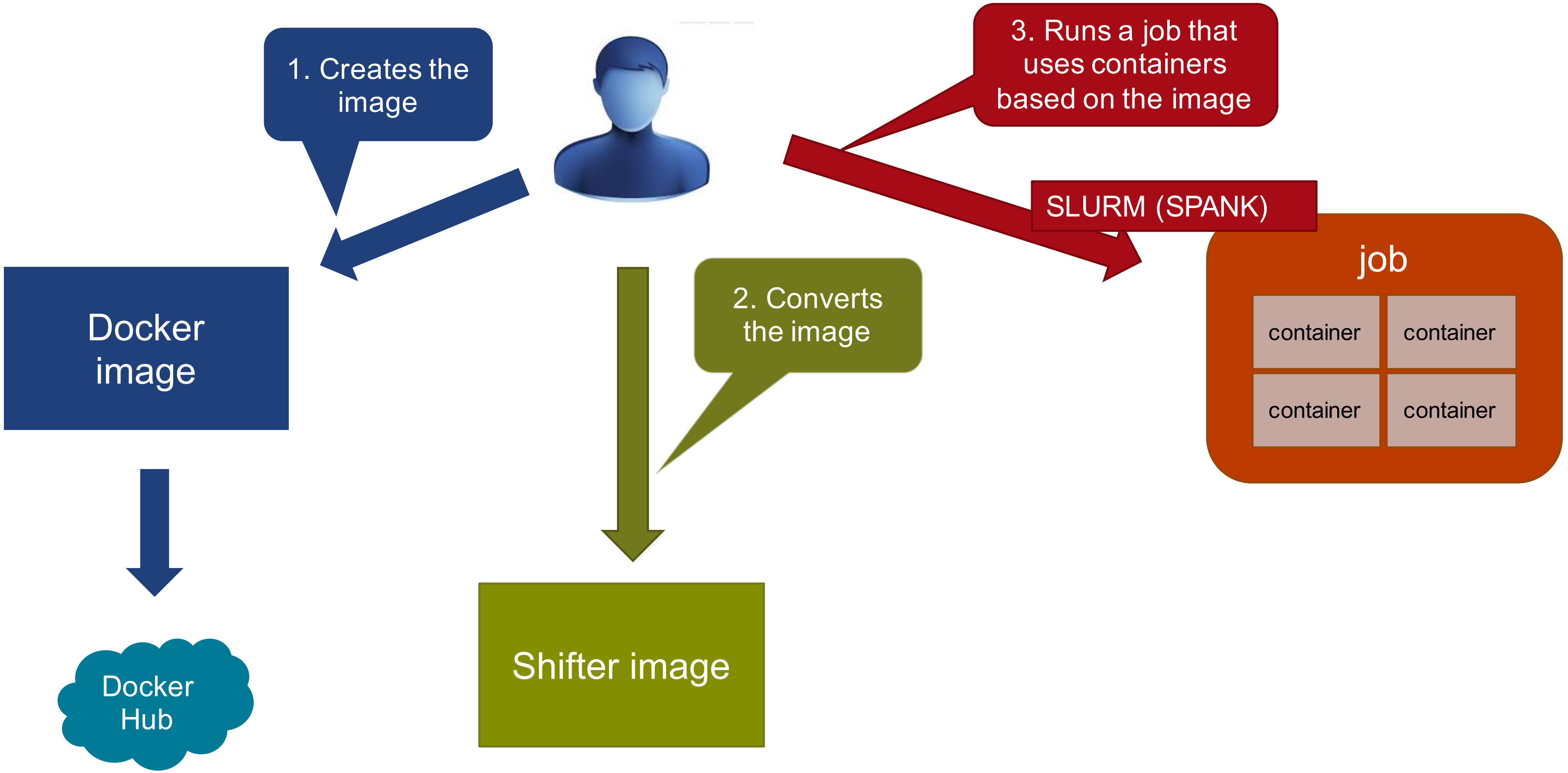
- Shifter is a great tool built for HPC!
- Integrates very well with workload managers (Slurm!)
- All user code is run in userland ☺
- No need for local storage ☺
- Can run off any mountpoint (/dsl/opt or /apps)
- Network and anything under /dev is exposed to containers
- Building Shifter is very easy
- Per-node caching (sparse xfs filesystems are awesome)

Architecture

- Shifter consists on two components:
  - imageGateway runs on an external server and converts any Docker image to a Shifter image. Built in Python, requires Redis & MongoDB.
  - udiRoot/Runtime runs on any compute node and chroots our application to run within the image constraints. Good old C.



Using shifter



Use case: High Energy Physics with containers on XC

WLCG Swiss Tier-2

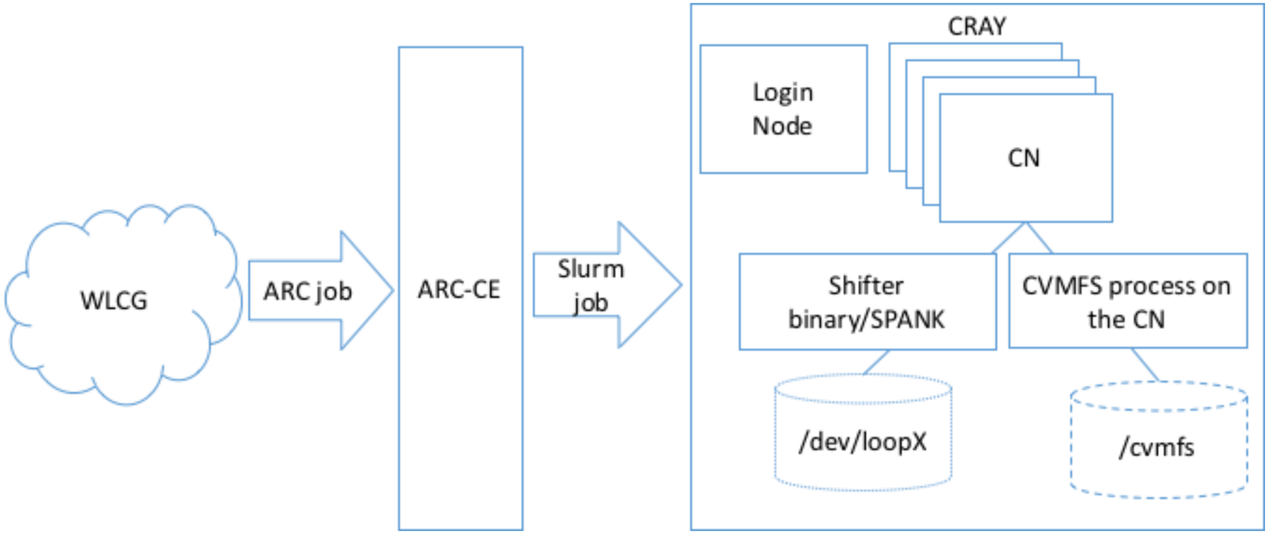
- CSCS operates the cluster Phoenix on behalf of CHIPP, the Swiss Institute of Particle Physics
- Phoenix runs Tier-2 jobs for ATLAS, CMS and LHCb, 3 experiments of the LHC at CERN and part of WLCG (Worldwide LHC Computing Grid)
- WLCG jobs need and expect RHEL-compatible OS. All software is precompiled and exposed in a cvmfs[*] filesystem
- But Cray XC compute nodes run CLE, a modified version of SLES 11 SP3

- So, how do we get these jobs to run on a Cray?

[*] https://cernvm.cern.ch/portal/filesystem

How do we get WLCG jobs to run on a Cray?

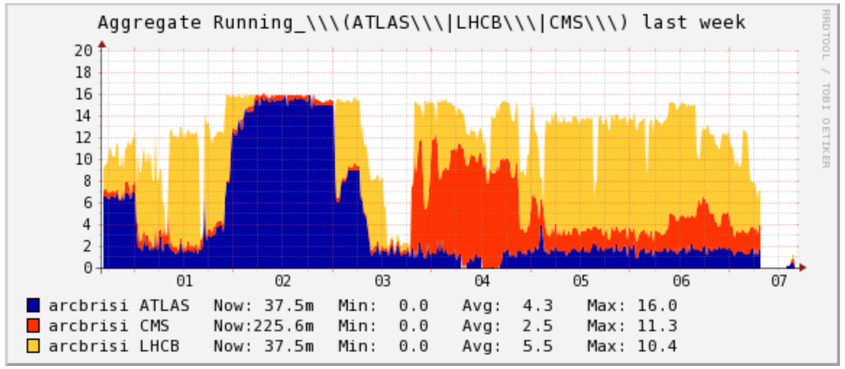
- Mount cvmfs natively on the compute nodes using a preloaded cache
  - Cvmfs process runs in the compute nodes, within dsl environment
  - Cvmfs cache is located on scratch and contains all the cache that CERN exposes
  - Compute nodes see /var/cvmfs/{atlas,cms,lhcb}.cern.ch and have 100% success hits
- Use shifter to contain the environment of a job to a RHEL-compatible image with a bunch of Grid packages (globus* and few others) installed
- Connect all this to WLCG with ARC

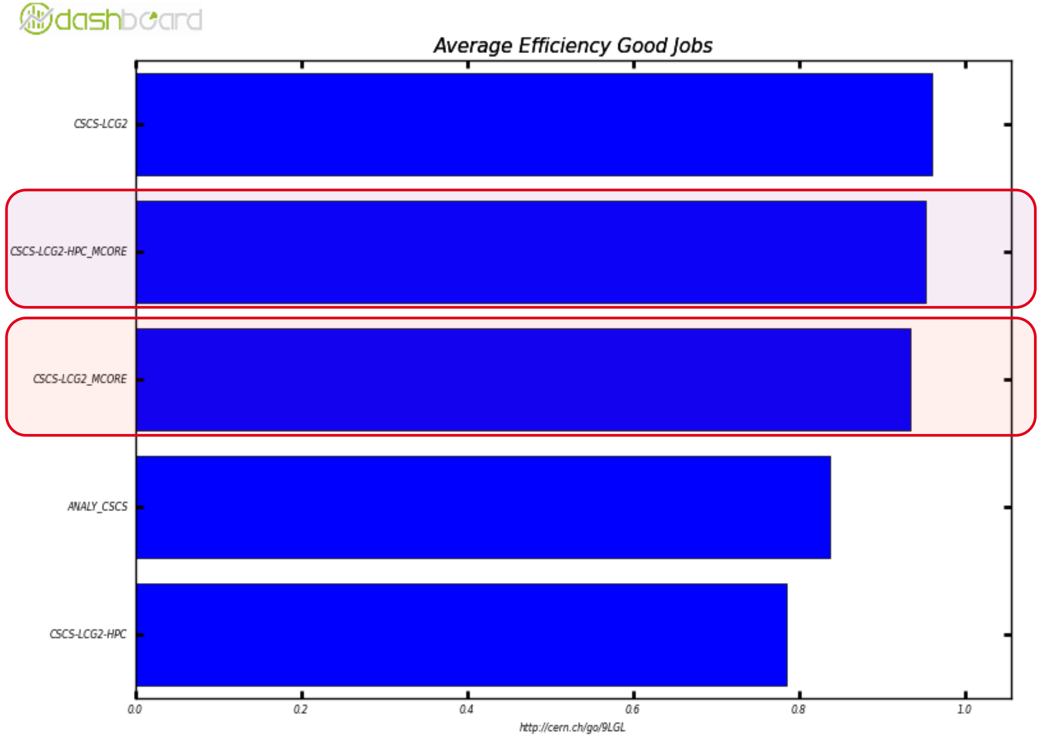


[*] https://cernvm.cern.ch/portal/filesystem

That's it!

- Using Shifter, we are able to run unmodified ATLAS, CMS and LHCb production jobs on a Cray XC TDS
- Jobs see standard CentOS 6 containers
- Nodes are shared: multiple single-core and multi-core jobs, from different experiments, can run on the same compute node
- Job efficiency is comparable in both systems





JOBID	USER	ACCOUNT	NAME	NODELIST	ST	REASON	START_TIME	END_TIME	TIME_LEFT	NODES	CPU
82471	atlasprd	atlas	a53eb5f8_34f0_	nid00043	R	None	15:03:33	Thu 15:03	1-23:54:18	1	8
82476	cms04	cms	gridjob	nid00043	R	None	15:08:39	Tomorr 03:08	11:59:24	1	2
82451	lhcbplt	lhcb	gridjob	nid00043	R	None	15:00:10	Tomorr 03:00	11:50:55	1	2
82447	lhcbplt	lhcb	gridjob	nid00043	R	None	14:59:31	Tomorr 02:59	11:50:16	1	2
82448	lhcbplt	lhcb	gridjob	nid00043	R	None	14:59:31	Tomorr 02:59	11:50:16	1	2
82449	lhcbplt	lhcb	gridjob	nid00043	R	None	14:59:31	Tomorr 02:59	11:50:16	1	2
82450	lhcbplt	lhcb	gridjob	nid00043	R	None	14:59:31	Tomorr 02:59	11:50:16	1	2
82446	lhcbplt	lhcb	gridjob	nid00043	R	None	14:49:01	Tomorr 02:49	11:39:46	1	2
82444	lhcbplt	lhcb	gridjob	nid00043	R	None	14:48:01	Tomorr 02:48	11:38:46	1	2
82445	lhcbplt	lhcb	gridjob	nid00043	R	None	14:48:01	Tomorr 02:48	11:38:46	1	2

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETHzürich

Use case: GPUs with containers on XC

Containerizing GPU applications

- Containerizing GPU applications provides and easy and portable way for packaging complex application setups

- Take advantage of several benefits:
  - Facilitate collaboration
  - Reproducible builds
  - Isolation of individual GPU devices
  - Running across heterogeneous CUDA toolkit environments
  - Requires only the NVIDIA driver installed on the host

The Docker catch

- Docker containers are both hardware-agnostic and platform-agnostic *by design*.

- This is not the case when using GPUs since:
  - it is using specialized hardware (that shows on your system as special character device), and
  - it requires the installation of the NVIDIA kernel driver

Collaboration with NVidia

- Mutual engineering and testing efforts to find a solution for Docker and Shifter

- Early prototype solution: to fully install the kernel driver inside the container, but:
  - the version of the host driver had to exactly match driver version installed in the container
  - container images had to be built locally on each machine, i.e., could not be shared
  - one needs to adhere to intellectual property regulations, i.e., not embedding proprietary code on a potentially sharable image without proper consent

Collaboration with NVidia: solution

- Shifter already provides the required character devices (/dev/nvidiaX) to the container 👍
- The driver files are mounted when starting the container on the target machine using the pre-mount hooks made available by Shifter
- Then we alter the runtime library search configuration to make the container aware of the new libraries available
- This makes images agnostic to the NVIDIA driver and capable of running on our environment without embedding any driver on the image

No overhead

- Testing done so far shows no overhead in terms of GPU performance when running within Shifter containers

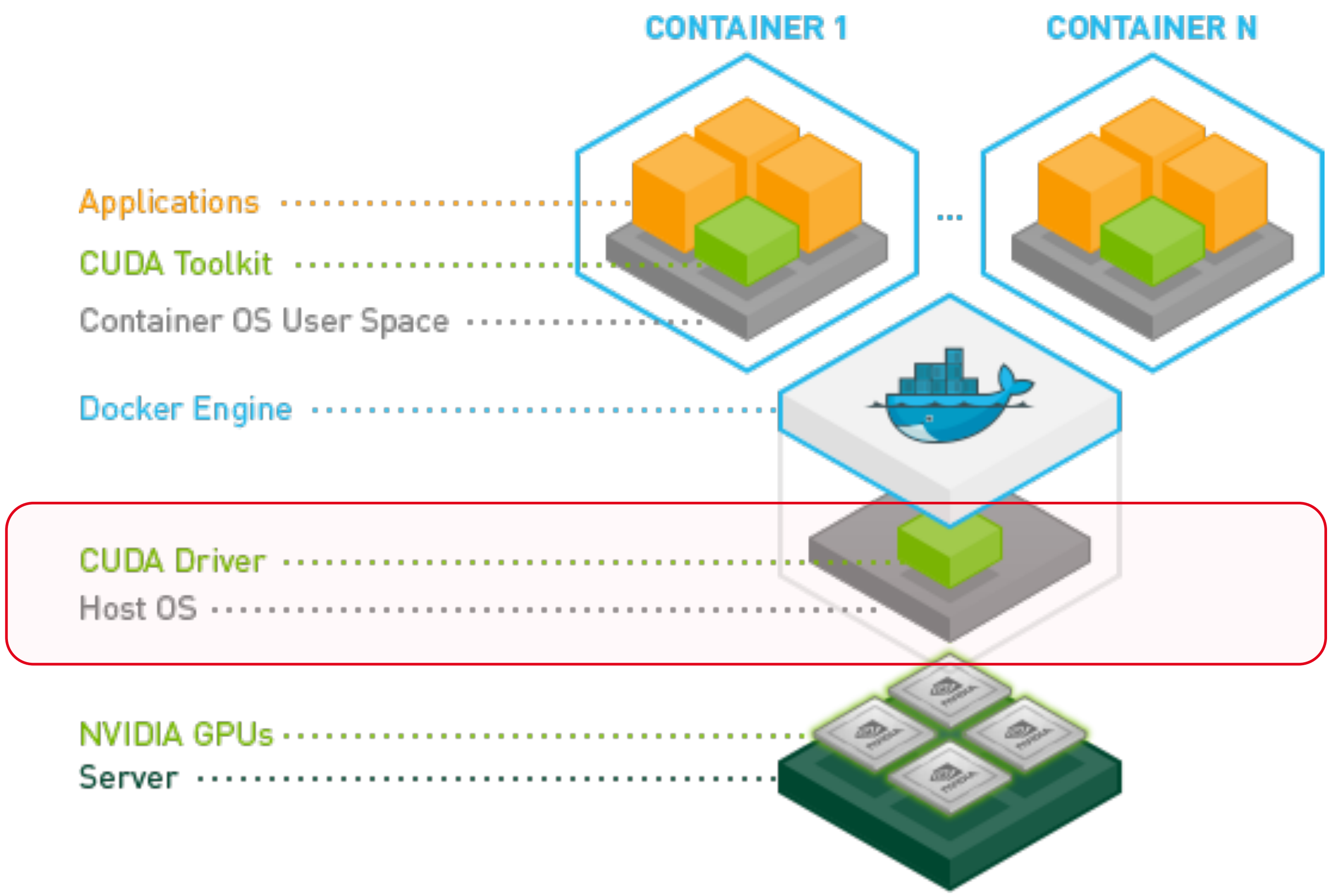
Stream benchmark within Shifter

```
lucasbe@santis01 ~/shifter-gpu> sbatch ./nvidia-docker/samples/cuda-
stream/benchmark.sbatch
Submitted batch job 496

lucasbe@santis01 /scratch/santis/lucasbe/jobs> cat shifter-gpu.out.log
Launching GPU stream benchmark on nid00012 ...
STREAM Benchmark implementation in CUDA
Array size (double precision) = 1073.74 MB
using 192 threads per block, 699051 blocks
Function      Rate (GB/s)   Avg time(s)   Min time(s)   Max time(s)
Copy:         184.3169      0.01167758    0.01165104    0.01170397
Scale:        183.1849      0.01175387    0.01172304    0.01178598
Add:          180.3075      0.01790012    0.01786518    0.01792288
Triad:        180.1056      0.01790700    0.01788521    0.01794291
```

A common approach

- NVIDIA DGX-1 uses the engineered solution for the management of its software stack



https://github.com/NVIDIA/nvidia-docker

Conclusion

Conclusion

- We like Shifter!
- It allows us to run workloads that traditionally have been difficult to port on Cray systems
- It helps our users to package complex applications and be able to reproduce results over time
- It's easy to use ☺

- But this is not all or nothing: things that already run on our Cray systems don't need to change

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Questions?

Thank you for your attention.

Extra slides