Outline

1. Overview
2. Docker
3. Shifter
4. Workflows
5. Use cases
6. Conclusion
Overview
CSCS / Piz Daint

Location: Swiss National Supercomputing Center

Name: Piz Daint

Model: Cray XC50/XC40

Node: Intel Xeon E5-2690, Nvidia Tesla P100, Aries interconnect

TOP500: 8th in the world

Green500: 2nd in the world
Motivation

▪ Bring Docker containers to production on Piz Daint.
  ▪ Docker: flexible and self-contained execution environments.
  ▪ Tool that enable workflows for some users.
  ▪ Part of an ecosystem that provides value to users.

▪ SI group focus on enhancing Shifter’s container runtime.
  ▪ Usability.
  ▪ Robustness.
  ▪ High-performance.
Talk in a nutshell

- Production workflows with Docker and Shifter on Piz Daint

1. Build and test containers with **Docker** on a **Laptop**.
2. Run with high-performance with **Shifter** on **Piz Daint**.
About Docker

▪ Motto: “Build, Ship, and Run Any App, Anywhere”
  ▪ Portability and convenience first
  ▪ Target: web applications.

▪ Creation
  ▪ Creates semi-isolated “containers”.
  ▪ Packages all application requirements.

▪ Management
  ▪ Easy-to-use recipe file.
  ▪ Version-control driven image creation.

▪ Share
  ▪ Push and Pull images from a community-driven Hub (i.e., DockerHub)
About Shifter (1)

- A lean runtime that enables the deployment of Docker-like Linux containers in HPC systems.
  - From NERSC by D. Jacobsen and S. Canon

- Objectives:
  - HPC environments.
  - **Performance**.
  - Security.
About Shifter (2)

▪ **Flexibility for users**
  ▪ Enable complex software stacks using different Linux flavors
  ▪ Develop on your laptop and run it on an HPC system

▪ **Integration with HPC resources**
  ▪ Availability of shared resources (e.g., parallel filesystems, accelerator devices and network interfaces)

▪ **Distribution and reproducibility**
  ▪ Integration with public image repositories, e.g., DockerHub
  ▪ Improving result reproducibility
Workflow (From a Laptop to Piz Daint)

- **Docker**: Ease of use and convenience (1, 2, 3).
- **Shifter**: Performance and security (4, 5).

1) Build container
2) Test container
3) Push to registry
4) Pull container
5) Run on Piz Daint
About Docker and Shifter

- Containers are **hardware- and platform-agnostic** by design
  - How do we go about **accessing specialized hardware** like GPUs?

- 1) CSCS and NVIDIA co-designed a solution that provides:
  - **direct access to the GPU** device characters;
  - automatic discovery of the required libraries at runtime;
  - NVIDIA’s DGX-1 software stack is based on this solution

- 2) CSCS extended design to the MPI stack
  - Supports different versions MPICH-based implementations

- Let’s look at use cases to illustrate the workflows.
Use case: deviceQuery (GPU)
deviceQuery (1): building a simple image

- Start with an image: nvidia image + deviceQuery
  - ethcscs/dockerfiles:cudasamples8.0

FROM nvidia/cuda:8.0

RUN apt-get update && apt-get install -y --no-install-recommends \
    cuda-samples-$CUDA_PKG_VERSION && \
    rm -rf /var/lib/apt/lists/*

RUN (cd /usr/local/cuda/samples/1.Utilities/deviceQuery && make)
RUN (cd /usr/local/cuda/samples/5.Simulations/nbody && make)
deviceQuery (2): Testing on a Laptop

- Let’s start with Docker on the laptop

```bash
$ nvidia-docker build -t "ethcscs/dockerfiles:cudasamples8.0" .

$ nvidia-docker run ethcscs/dockerfiles:cudasamples8.0 ./deviceQuery

$ nvidia-docker push ethcscs/dockerfiles:cudasamples8.0
```

`./deviceQuery Starting...`

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

Detected **1** CUDA Capable device(s)

Device 0: "GeForce 940MX"

  CUDA Driver Version / Runtime Version 8.0 / 8.0

[...]

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 8.0, CUDA Runtime Version = 8.0,
NumDevs = **1**, Device0 = GeForce 940MX

Result = PASS
deviceQuery (3): Running on a Piz Daint

- Running the container on Piz Daint

$ shifterimg pull ethcscs/dockerfiles:cudasamples8.0

$ salloc -N 1 -C gpu
$ srun shifter --image=ethcscs/dockerfiles:cudasamples8.0 ./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 8.0 / 8.0

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 8.0, CUDA Runtime Version = 8.0, NumDevs = 1, Device0 = Tesla P100-PCIE-16GB
Result = PASS
Use case: Nbody (GPU)
Nbody: same image

- N-body double-precision calculation using 200k bodies, single node.
- GPU-accelerated runs using the official CUDA image from DockerHub.
- Relative GFLOP/s performance when comparing Laptop and Piz Daint.

<table>
<thead>
<tr>
<th></th>
<th>Laptop</th>
<th>Piz Daint (P100)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native</td>
<td>18.34 [GFLOP/s]</td>
<td>149.01x</td>
</tr>
<tr>
<td>Container*</td>
<td>18.34 [GFLOP/s]</td>
<td>149.04x</td>
</tr>
</tbody>
</table>

*Laptop run using nvidia-docker, Piz Daint uses Shifter*
Use case: TensorFlow (GPU / Third party container)
TensorFlow

- Software library capable of building and training neural networks uses CUDA.
- **Official** TensorFlow image from DockerHub (not modified).
- TensorFlow has a **rapid release cycle** (Once a week new build available!).
- **Ready to run** containers.
- Performance relative to the Laptop wall-clock time of image classification tests: MNIST.

<table>
<thead>
<tr>
<th>Test case</th>
<th>Laptop*</th>
<th>Piz Daint (P100)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST, TF tutorial</td>
<td>613 [seconds]</td>
<td>17.17x</td>
</tr>
</tbody>
</table>

*Laptop run using nvidia-docker*
Note on MPI-stack support

- Based on the **MPICH**, Application Binary Interface (ABI) compatibility
  - MPICH v3.1 (released February 2014)
  - IBM MPI v2.1 (released December 2014)
  - Intel MPI Library v5.0 (released June 2014)
  - CRAY MPT v7.0.0 (released June 2014)
  - MVAPICH2 v2.0 (released June 2014)

```bash
$ srun -n 2 shifter --mpi --image=osu-benchmarks-image ./osu_latency
```

- MPI library from the container is **swapped** by Shifter at run time.
- ABI-compatible MPI from the **host** system is checked.
  - Hardware acceleration is enabled.
Use case: OSU benchmark (MPI)
OSU Benchmark

$ srun -n 2 shifter --mpi --image=osu-benchmarks-image ./osu_latency

- **Host MPI:**
  - Cray MPT 7.5.0
  - Cray aries Interconnect

- **Container MPI:**
  - MPICH v3.1 (A)
  - MVAPICH2 2.1 (B)
  - Intel MPI Library (C)

- Native performance!

<table>
<thead>
<tr>
<th>Size</th>
<th>Native</th>
<th>Shifter MPI support</th>
<th>Shifter MPI support</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Enabled</td>
<td>Disabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>32</td>
<td>1.1</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>128</td>
<td>1.1</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>512</td>
<td>1.1</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>2K</td>
<td>1.6</td>
<td>1.06</td>
<td>1.00</td>
</tr>
<tr>
<td>8K</td>
<td>4.1</td>
<td>1.00</td>
<td>1.02</td>
</tr>
<tr>
<td>32K</td>
<td>6.5</td>
<td>1.03</td>
<td>1.03</td>
</tr>
<tr>
<td>128K</td>
<td>16.4</td>
<td>1.01</td>
<td>1.01</td>
</tr>
<tr>
<td>512K</td>
<td>56.1</td>
<td>1.00</td>
<td>1.01</td>
</tr>
<tr>
<td>2M</td>
<td>215.7</td>
<td>1.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Table 4: Results from OSU_latency on Piz Daint: Native runs use Cray MPT 7.5.0 over Cray Aries interconnect; relative performance against native is reported for containers with (A) MPICH 3.1.4, (B) MVAPICH2 2.2, and (C) Intel MPI library using Shifter with MPI support enabled and disabled.
Use case: PyFR (CUDA + MPI / Complex build)
PyFR

- **Python** based framework for solving advection-diffusion type problems on streaming architectures. 2016 **Gordon Bell** Prize finalist (Highly scalable).
- **GPU-** and **MPI-accelerated** runs using containers.
- Complex build (100 lines dockerfile) and test on Laptop.
- Production-like run on Piz Daint.
- Parallel efficiency for a 10-GB test case on different systems (4 node setup).

<table>
<thead>
<tr>
<th>Number of nodes</th>
<th>Piz Daint (P100)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.000</td>
</tr>
<tr>
<td>2</td>
<td>0.975</td>
</tr>
<tr>
<td>4</td>
<td>0.964</td>
</tr>
<tr>
<td>8</td>
<td>0.927</td>
</tr>
<tr>
<td>16</td>
<td>0.874</td>
</tr>
</tbody>
</table>
Use case: Portable compilation units / Linpack benchmark
Vanilla Linpack with specialized BLAS

- Some application performance depends on targeted optimization of libraries.
- Use container to pack application environment.
- Proof of concept: pack vanilla Linpack, compile specialized BLAS before run.
- Two stage: compile first (link against host libs), then run.
Conclusion
Conclusion

- The Docker-Shifter combo towards production on Piz Daint:
  - Portability
  - Scalability
  - High-performance.

- The showed use cases highlighted:
  - Pull and run containers;
  - high-performance containers;
  - access to hardware accelerators like GPUs;
  - use of high-speed interconnect through MPI;
  - portable compilation environment.
Thank you for your attention
GPU device access (3)

- On the GPU device numbering
  - Must-have for application portability
  - A containerized application will consistently access the GPUs starting from ID 0 (zero)

```
$ export CUDA_VISIBLE_DEVICES=0
$ srun shifter --image=ethcscs/dockerfiles:cudasamples8.0 ./deviceQuery
[...]
Detected 1 CUDA Capable device(s)
Device 0: "Tesla K40m"
[...]

$ export CUDA_VISIBLE_DEVICES=2
$ srun shifter --image=ethcscs/dockerfiles:cudasamples8.0 ./deviceQuery
[...]
Detected 1 CUDA Capable device(s)
Device 0: "Tesla K80"
[...]```
GPU device access (4)

- Same image on an multi-GPU system with Shifter

```
$ shifterimg pull ethcscs/dockerfiles:cudasamples8.0

$ export CUDA_VISIBLE_DEVICES=0,2
$ srun shifter --image=ethcscs/dockerfiles:cudasamples8.0 ./deviceQuery
```

```
/usr/local/cuda/samples/bin/x86_64/linux/release/deviceQuery Starting...

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

Detected 2 CUDA Capable device(s)

Device 0: "Tesla K40m"
    [...]  
Device 1: "Tesla K80"
    [...]  
deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 8.0, CUDA Runtime Version = 8.0,  
NumDevs = 2, Device0 = Tesla K40m, Device1 = Tesla K80  
Result = PASS
```
Use case: deploy at scale
Pynamic*

- Test startup time of workloads by simulating **DLL behaviour of Python** applications.
- Compare wall-clock time for **container vs native**.
- Over **+3000** MPI processes.

*Pynamic parameters: 495 shared object files; 1950 avg. functions per object; 215 math like library files; 1950 avg. math functions; function name length ~100 characters.