Step-by-step guide to Shifter on Piz Daint

A note about SLURM commands

Piz Daint uses the SLURM Workload Manager to assign jobs to its compute nodes. In case you are not familiar with basic usage of SLURM, here we provide brief explanations to the commands used throughout this guide:

`salloc` is used to allocate resources for a job in real time. Typically this is used to allocate resources and spawn a shell. The shell is then used to execute `srun` commands to launch parallel tasks.

`srun` is used to submit a job for execution or initiate job steps in real time.

Both these commands support the following options:

`-C` indicates a list of contraints for the nodes where to make an allocation or run a job. In this document we will be using `-C gpu` to indicate we want to run on Piz Daint's hybrid partition, with nodes featuring Intel Haswell CPUs and NVIDIA Pascal GPUs.

`--reservation` allocates resources on a specific reservation (please note that if you are taking part to a hands-on session with a dedicated reservation, such reservation will have a limited time duration).

`-n` indicates the total number of tasks to run

`-N` indicates the number of compute nodes to use

Preparing the Shifter environment

```
module load shifter-ng
```

---

1. Query Shifter images

   You can list the Shifter images available to you on a system with the `shifter images` command. The images displayed here are located in an individual repository, and are not shared with other users.

   EXAMPLE:

```
$ shifter images

REPOSITORY            TAG          DIGEST         CREATED                SIZE
SERVER
ethcscs/cudasamples   8.0          7876f3019185   2018-06-12T12:40:27
978.69MB     index.docker.io
library/alpine        latest       9797e5e798a0   2018-06-12T12:39:06   1.94MB
index.docker.io
library/ubuntu        latest       7feff7652c69   2018-06-12T12:39:39
29.12MB      index.docker.io
```

2. Pull a new image from Docker Hub

   You can pull images from Docker Hub into the HPC system with the `shifter pull` command. We strongly recommend to run `shifter pull` on the compute nodes through SLURM, so that Shifter can take advantage of their large RAM filesystem, which will greatly reduce the pull process time and will allow to pull larger images.

   EXAMPLE:

```
$ salloc -N 1 -C gpu --reservation=<reservation name>
$ srun shifter pull debian

srun: job 760756 queued and waiting for resources
srun: job 760756 has been allocated resources
# image     : index.docker.io/library/debian/latest
# cacheDir  : /scratch/snx3000/<user>/.shifter/cache
# tmpDir    : /dev/shm
# imageDir  : /scratch/snx3000/<user>/.shifter/images
> save image layers ...
> pulling        :
sha256:cc1a78bfd46becbfc3abb8a74d9a70a0e0dc7a5809bbd12e814f9382db003707
> completed      :
sha256:cc1a78bfd46becbfc3abb8a74d9a70a0e0dc7a5809bbd12e814f9382db003707
> expand image layers ...
> extracting     :
/scratch/snx3000/<user>/.shifter/cache/sha256:cc1a78bfd46becbfc3abb8a74d9a70a0
e0dc7a5809bbd12e814f9382db003707.tar
> make squashfs ...
> create metadata ...
# created:
/scratch/snx3000/<user>/.shifter/images/index.docker.io/library/debian/latest.
squashfs
# created:
/scratch/snx3000/<user>/.shifter/images/index.docker.io/library/debian/latest.
meta

$ exit
```

3. Run a container with Shifter

   You can run containers using SLURM and the `shifter run` command, specifying the desired image as the first positional argument of the command. The arguments entered after the image's name will be interpreted as the command to be executed inside the container. If nothing is specified after the image, the container's `$SHELL` will be attempted. Finally, if `$SHELL` is not set, `/bin/sh` will be attempted.

   You can check that you are actually running in a container by inspecting `/etc/os-release`.

   EXAMPLE:

```
$ salloc -N 1 -C gpu --reservation=<reservation name>
$ srun shifter run debian cat /etc/os-release

PRETTY_NAME="Debian GNU/Linux 9 (stretch)"
NAME="Debian GNU/Linux"
VERSION_ID="9"
VERSION="9 (stretch)"
ID=debian
```

```
HOME_URL="https://www.debian.org/"
SUPPORT_URL="https://www.debian.org/support"
BUG_REPORT_URL="https://bugs.debian.org/"

$ srun cat /etc/os-release

NAME="SLES"
VERSION="12-SP2"
VERSION_ID="12.2"
PRETTY_NAME="SUSE Linux Enterprise Server 12 SP2"
ID="sles"
ANSI_COLOR="0;32"
CPE_NAME="cpe:/o:suse:sles:12:sp2"

$ exit
```

4. Run a container with an interactive shell

   As with Docker, you can access Shifter containers through an interactive shell. Contrary to Docker, no specific command line options are required by Shifter to start an interactive shell. The `--pty` flag to `srun` is optional, but it makes the experience much more user-friendly.

   EXAMPLE:

```
$ salloc -N 1 -C gpu --reservation=<reservation name>
$ srun --pty shifter run debian bash
$ cat /etc/os-release

PRETTY_NAME="Debian GNU/Linux 9 (stretch)"
NAME="Debian GNU/Linux"
VERSION_ID="9"
VERSION="9 (stretch)"
ID=debian
HOME_URL="https://www.debian.org/"
SUPPORT_URL="https://www.debian.org/support"
BUG_REPORT_URL="https://bugs.debian.org/"

$ exit  #from the container
$ exit
```

5. Access personal files from the container

   Your `$SCRATCH` directory is automatically mounted inside the container, so any personal data you intend to work with can be easily available within the container environment.

   EXAMPLE (exact path of the scratch filesystem can vary):

```
$ cd $SCRATCH
$ salloc -N 1 -C gpu --reservation=<reservation name>
$ srun shifter run debian pwd

/scratch/snx3000/<user name>

$ touch test_file
$ srun shifter run debian ls

test_file
```

```
$ exit
```

6. Detect GPUs available in the container

   Enabling native GPU support in Shifter on Piz Daint does not require any direct user action, besides using an image with the CUDA Toolkit installed.

   To list the GPU devices available in the container, you can run the CUDA sample `deviceQuery` which is provided with the CUDA Toolkit SDK. We have already built an image with compiled CUDA samples, and you can retrieve it from Docker Hub using the identifier `ethcscs/cudasamples:8.0`.

   EXAMPLE:

```
$ salloc -N 1 -C gpu --reservation=<reservation name>
$ srun shifter pull ethcscs/cudasamples:8.0

[ shifter pull output ]

$ srun shifter run ethcscs/cudasamples:8.0
/usr/local/cuda/samples/bin/x86_64/linux/release/deviceQuery

/usr/local/cuda/samples/bin/x86_64/linux/release/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
  CUDA Capability Major/Minor version number:    6.0
  Total amount of global memory:                 16276 MBytes (17066885120
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
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 2 copy engine(s)
  Run time limit on kernels:                     No
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Enabled
  Device supports Unified Addressing (UVA):      Yes
```

```
  Device PCI Domain ID / Bus ID / location ID:   0 / 2 / 0
  Compute Mode:
     < Exclusive Process (many threads in one process is able to use
::cudaSetDevice() with this device) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 8.0, CUDA Runtime
Version = 8.0, NumDevs = 1, Device0 = Tesla P100-PCIE-16GB
Result = PASS

$ exit
```

7. Run a GPU application in the container

   You can run the `nbody` CUDA sample which is provided with the CUDA Toolkit SDK.

   EXAMPLE:

```
$ salloc -N 1 -C gpu --reservation=<reservation name>
$ srun shifter run ethcscs/cudasamples:8.0
/usr/local/cuda/samples/bin/x86_64/linux/release/nbody -benchmark -fp64 -
numbodies=200000

Run "nbody -benchmark [-numbodies=<numBodies>]" to measure performance.
    -fullscreen       (run n-body simulation in fullscreen mode)
    -fp64             (use double precision floating point values for
simulation)
    -hostmem          (stores simulation data in host memory)
    -benchmark        (run benchmark to measure performance)
    -numbodies=<N>    (number of bodies (>= 1) to run in simulation)
    -device=<d>       (where d=0,1,2.... for the CUDA device to use)
    -numdevices=<i>   (where i=(number of CUDA devices > 0) to use for
simulation)
    -compare          (compares simulation results running once on the default
GPU and once on the CPU)
    -cpu              (run n-body simulation on the CPU)
    -tipsy=<file.bin> (load a tipsy model file for simulation)

NOTE: The CUDA Samples are not meant for performance measurements. Results may
vary when GPU Boost is enabled.

> Windowed mode
> Simulation data stored in video memory
> Double precision floating point simulation
> 1 Devices used for simulation
GPU Device 0: "Tesla P100-PCIE-16GB" with compute capability 6.0

> Compute 6.0 CUDA device: [Tesla P100-PCIE-16GB]
Warning: "number of bodies" specified 200000 is not a multiple of 256.
Rounding up to the nearest multiple: 200192.
200192 bodies, total time for 10 iterations: 4400.005 ms
= 91.084 billion interactions per second
= 2732.509 double-precision GFLOP/s at 30 flops per interaction

$ exit
```

   To see the effect of GPU acceleration, try to run the sample benchmark on the CPU using the `-cpu` option. We advise to greatly reduce the number of bodies specified with the `-numbodies` option to avoid waiting too long.

8. Run an MPI application in the container

   To enable native MPI support, supply the `--mpi` command line option to `shifter run`.

   As an example, you can run the MPI latency test which is part of the OSU micro-benchmarks. We have already built a container image with the OSU micro-benchmarks, and you can retrieve it from Docker Hub using the identifier `ethcscs/osu-mb:5.3.2-mpich3.1.4`.

   EXAMPLE:

```
$ salloc -C gpu -N 2 --reservation=<reservation name>
$ srun -n 1 shifter pull ethcscs/osu-mb:5.3.2-mpich3.1.4

[ shifter pull output ]

$ srun -n 2 shifter run --mpi ethcscs/osu-mb:5.3.2-mpich3.1.4
/usr/local/libexec/osu-micro-benchmarks/mpi/pt2pt/osu_latency

# OSU MPI Latency Test v5.3.2
# Size          Latency (us)
0                       1.14
1                       1.13
2                       1.11
4                       1.10
8                       1.10
16                      1.11
32                      1.08
64                      1.08
128                     1.10
256                     1.11
512                     1.14
1024                    1.38
2048                    1.65
4096                    2.25
8192                    4.36
16384                   5.20
32768                   6.86
65536                  10.21
131072                 16.84
262144                 30.15
524288                 56.87
1048576               110.00
2097152               216.69
4194304               433.59

$ exit
```