Deploying Containerized Applications on HPC Production Systems at LRZ

December 2021 | David Brayford

Motivation for using Containers in HPC

Transition workflows from the **laptop to supercomputer** with minimal effort

"It just works"

Workflow Portability in a HPC Container

The ability to transition workflows from **laptop to supercomputers** with minimal effort is increasingly important in the world of heterogeneous Exascale HPC systems

“Increase Productivity”

“Containers are NOT PORTABLE across all systems”

Key Challenges for HPC

Instruction Set Architecture & Hardware	CPU(s), GPU(s), Accelerators, Memory, Interconnects
System software	Filesystem, “drivers” (hardware), distributed processing (MPI)
Performance & stability	Use system optimized libraries and software (Licences)
Container size	Store large datasets outside of the container
Execution patterns	IO

Deployment with Charliecloud

Mechanism for deploying Container workflows:

1. Create a Docker image from your workflow recipe (Dockerfile)
   - Modify the Dockerfile for the HPC system
   - Create the Docker image & test
2. Convert to a HPC container (Charliecloud) & test
3. Copy the container to the HPC system & load the HPC container module
4. Copy the container to the HPC system & load the HPC container module
5. Execute via Slurm

TensorFlow Optimized HPC System Workflow SuperMUC-NG

Mounted the LRZ file system into the container & used the system version of Intel MPI at runtime

ch-run -b /dss/.:/dss/ -w container_name – python /location/in/container/training_script.py

Scaling Efficiency on SNG

Nodes	Training Time(S) per Epoch	Linear Time(S) per Epoch	Scaling Efficiency
4	907.26	907.26	-
8	479.52	453.63	94.6%
16	244.42	226.82	92.8%
32	124.22	113.41	91.3%
64	62.24	56.70	91.1%
128	31.22	28.35	90.8%
256	15.63	14.18	90.7%
512	7.84	7.09	90.4%
768	3.94	3.54	89.9%

Performance in FLOPs

Nodes	Measured Performance petaflops	Percentage of Theoretical Peak
4	0.01099	66.17%
8	0.02199	66.21%
16	0.04450	67.01%
32	0.08386	63.14%
64	0.17313	65.17%
128	0.31878	67.60%
256	0.70547	66.39%
512	1.39412	65.60%
768	2.08143	65.29%

Enable exAScale for EverYone (EASEY)

Issues:

1. Containerized application crashed due to MPI issues
   - Resolved by setting the libfabric parameters
2. Poor performance due to running MPI over TCP
   - Resolved by installing OmniPath software inside the container

Outcome:

All the issues with stability and performance were no longer observed and the containerized application was able to execute successfully on 100’s of nodes with 8000 MPI tasks

Software Fuzzing

Issues:

1. The applications IO pattern crashed the parallel file system
   - Switched to mounting a more performant directory of the parallel file server inside the container
   - Switched to mounting the host systems RAM disk inside the container for storing temporary files

Outcome:

Using the RAM disk of host system to store the millions of temporary files generated by the containerized application fixed the parallel file system crashes

QuantEX

Issues:

1. Julia installs packages into ~/julia by default. Charliecloud maps the host ~ directory inside the container
   - Resolved by changing the Julia package installation path and using the Docker environment instead of the host environment
2. Profiling the Julia application using LIKWID inside the container
   - Resolved by mounting the host module system inside the container

Outcome:

Able to execute and profile the containerized QuantEX software on the HPC systems at LRZ

Fujitsu Arm A64FX

Host execution time in Julia

Tot / % measured	Time	Allocations
Compile + exec	104s / 97.0%	3.79GiB / 98.4%
Exec only	414ms / 44.7%	11.7MiB / 79.2%

Container execution time in Julia

Tot / % measured	Time	Allocations
Compile + exec	101s / 96.9%	3.68GiB / 98.4%
Exec only	457ms / 50.0%	11.7MiB / 79.2%

BEAST System Simulation

Fujitsu Arm A64FX

likwid-perfctr -m -g L2CACHE -C 0 julia --project /QXContexts/bin/qxrun.jl -t -d /QXContexts/examples/ghz/ghz_5.qx -o /QXContexts/examples/ghz/out_THX_LK_container.jld2

Host LIKWID Profiling

Region Info	HWThread 0	
RDTSC Runtime [s]	0.433888	
call count	1	
Event	Counter	HWThread 0
FP_DP_FIXED_OPS_SPEC	PMC0	496
FP_DP_SCALE_OPS_SPEC	PMC1	0
L2D_CACHE_REFILL	PMC2	220698
L2D_CACHE_WB	PMC3	74772
L2D_SWAP_DM	PMC4	16940
L2D_CACHE_MIBMCH_PRF	PMC5	24380
Metric		HWThread 0
Runtime (RDTSC) [s]		0.4339
DP (FP) [MFLOP/s]		0.0011
DP (FP+SVE128) [MFLOP/s]		0.0011
DP (FP+SVE256) [MFLOP/s]		0.0011
DP (FP+SVE512) [MFLOP/s]		0.0011
Memory read bandwidth [MBytes/s]		105.8355
Memory read data volume [GBytes]		0.0459
Memory write bandwidth [MBytes/s]		44.1165
Memory write data volume [GBytes]		0.0191
Memory bandwidth [MBytes/s]		149.9521
Memory data volume [GBytes]		0.0651
Operational intensity (FP)		7.623451e-06
Operational intensity (FP+SVE128)		7.623451e-06
Operational intensity (FP+SVE256)		7.623451e-06
Operational intensity (FP+SVE512)		7.623451e-06

Container LIKWID Profiling

Region Info	HWThread 0	
RDTSC Runtime [s]	0.477314	
call count	1	
Event	Counter	HWThread 0
FP_DP_FIXED_OPS_SPEC	PMC0	496
FP_DP_SCALE_OPS_SPEC	PMC1	0
L2D_CACHE_REFILL	PMC2	883647
L2D_CACHE_WB	PMC3	373491
L2D_SWAP_DM	PMC4	48554
L2D_CACHE_MIBMCH_PRF	PMC5	40377
Metric		HWThread 0
Runtime (RDTSC) [s]		0.4773
DP (FP) [MFLOP/s]		0.0010
DP (FP+SVE128) [MFLOP/s]		0.0010
DP (FP+SVE256) [MFLOP/s]		0.0010
DP (FP+SVE512) [MFLOP/s]		0.0010
Memory read bandwidth [MBytes/s]		426.2337
Memory read data volume [GBytes]		0.2034
Memory write bandwidth [MBytes/s]		200.3161
Memory write data volume [GBytes]		0.0956
Memory bandwidth [MBytes/s]		626.5498
Memory data volume [GBytes]		0.2991
Operational intensity (FP)		1.658525e-06
Operational intensity (FP+SVE128)		1.658525e-06
Operational intensity (FP+SVE256)		1.658525e-06
Operational intensity (FP+SVE512)		1.658525e-06

Takeaways

- Start with a Dockerfile
- Test the containerized workflows (Docker & HPC container)
- Do as much work on your local system or development VM
- Use software already installed on the HPC system if possible
- Easy to deploy software on different HPC systems at LRZ
- Build the container image for each system
- Easy to use host installed software inside the container
- No big differences in performance

Documentation & Contacts

Web Page

https://doku.lrz.de/display/PUBLIC/Charliecloud+at+LRZ

Github repository

- https://github.com/JuliaQX
- https://juliaqx.github.io/QXTools.jl/dev

Publications

- Deploying AI Frameworks on Secure HPC Systems with Containers
- Deploying scientific al networks at petaflop scale on secure large scale HPC production systems with containers
- Deploying Containerized QuantEx Quantum Simulation Software on HPC Systems

Contact

- LinkedIn https://www.linkedin.com/in/david-brayford-5900a817
- Twitter @david_brayford