Use of Singularity containers on IT4I systems from the perspective of normal user

Jakub Výmola & Dominik Legut

VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

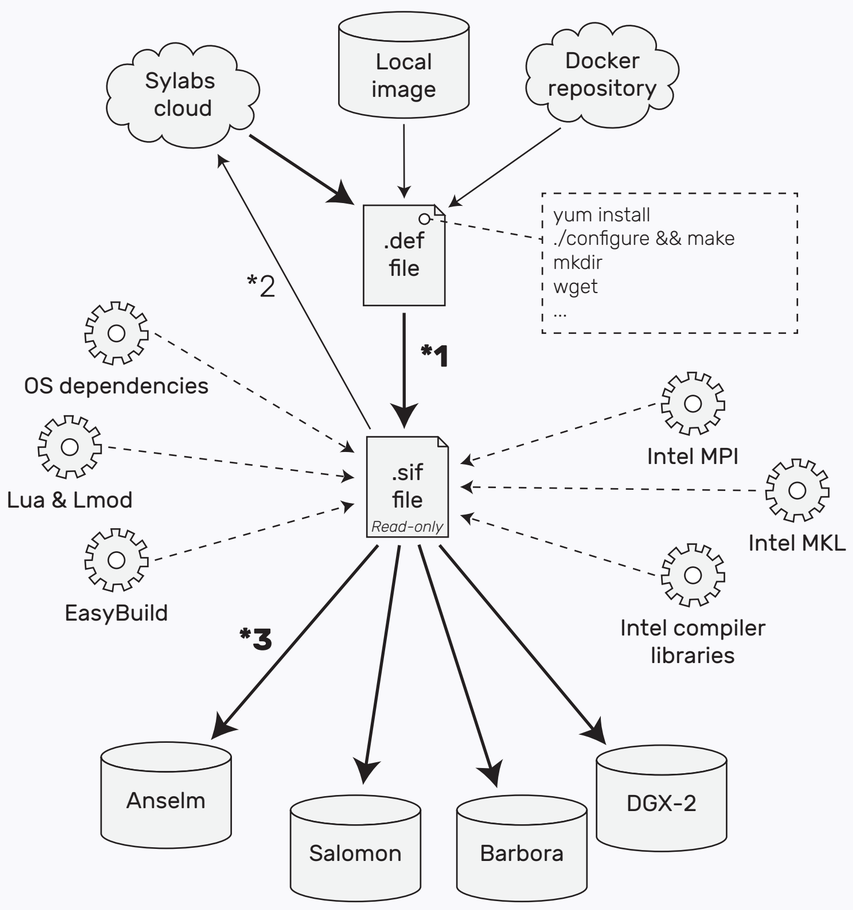
IT4Innovations national supercomputing Centre,
VŠB-Technical University of Ostrava
17. listopadu 15/2172, CZ 708 33, Czech Republic

Containerize software compiled with any toolchain that works anywhere.
Don't lose any more time and performance.

www.md-esg.eu

ABSTRACT

Containers are a great way to encapsulate software and environment for its reliable and quick deployment. They contain all the libraries and binaries needed to run specific software, however they share kernel with the host OS, which makes them more lightweight and faster to run relative to virtual machines. They can be transferred and executed anywhere, as long as there is the container runtime installed on the target machine and it has compatible architecture. This means that the same container can be seamlessly run on every type of node at IT4I, even on the accelerated ones, e.g. CUDA GPUs (as long as the installed software itself supports GPU acceleration). One can simply create the container with the compiled software either on a personal machine or on any cluster, then transfer the container to any other cluster and expect it to work right away, without having to configure or install anything. On IT4I HPC systems the container system of the choice is called Singularity. It works well on multi-user systems and was developed with HPC needs in mind. It is an open source project that began its development in 2015 and is supported by thorough documentation [1]. We focus on the use of Singularity from the point of view of a normal user. It is shown how one can create containers, compile software into them and run them at various IT4I clusters, including accelerated systems, like GPU nodes on Barbora or DGX-2. The major obstruction of using containers is the absence of licensed software on local machine and root privileges on the clusters. Here, we demonstrate the optimized workflow and some tested tips and tricks to get around those problems. The workflow includes the use of Sylabs cloud to create the containers, persistent overlay to compile additional software, as well as the use of modules tool, EasyBuild and mounted directories, to compile software using licensed tools, without ever having to leave the cluster.



LOCAL MACHINE

The image creation starts at your machine. That's because you have root permissions there.

Your actions will be the following:

- **Create a definition file** [1].
- In the file **specify the source image**. This is taken from your disk or a cloud repository.
- Also, **write the script** to install necessary and handy software.
  - You need to install **Intel MKL and Intel MPI** to run parallel software compiled with Intel compilers. [2]
  - You can optionally install EasyBuild and Lmod. EasyBuild is for easy compilation. Lmod is for easy work with toolchain and library modules on the cluster. It is also required by EasyBuild.
  - As a backup, you can install BLAS, LAPACK, etc.
  - Do not mix different MPI versions in one container.
- **Build the container image** with *sudo singularity build* (*1) command.
- Use one of *shell*, *exec* or *run* commands to work with the container.
- For easy access, you can push this image to the Sylabs cloud with *singularity push* (*2). You first need to *singularity verify* the image.
- You can now **distribute the image across the clusters** with *scp* or *singularity pull* (*3). Remember, it's just a file.
- Alternative way to build container is to use Singularity *remote builder* at cluster.

SINGULARITY relative speed with VASP

1 node, 1 MPI proc

Native	1
Singularity	1,02

4 nodes, 4 MPI procs

Native	1
Singularity	0,988

* Each number is the average of 4 identical jobs ran on Salomon. Using PtN2 pyrite and fluorite 2x2x2 supercell. Total of 96 atoms.

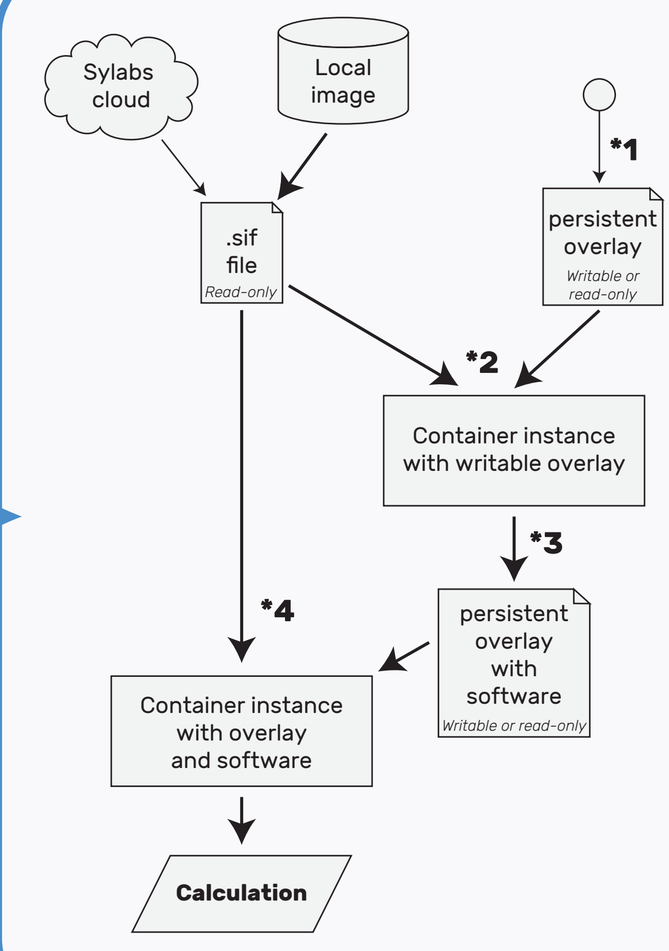
* Compiling VASP into overlay separately on on each cluster. Using *xHost* flag with *ifort* yields the best performance. Such build is not transferable between incompatible processor architectures.

ON THE CLUSTER

Notice that you haven't installed your target software yet. That is because we want to use a proprietary toolchain.

However, the container image is now read-only, so you cannot compile it directly into it. You could use the *sandbox* image format. But there is a much better way. It's called ***persistent overlay***.

- Use *mkfs.ext3* (*1) to **make ext3 image** - that's your overlay.
- Use the container with ***-o <overlay-name> -B /apps:/apps*** (*2) flags to both mount the overlay and gain access to all the modules in /apps. Now Lmod makes it easy to load those modules.
- **Get your source files into the container and compile them**. *eb, make* (*3) or anything else that does not require *sudo* works.
- Now the overlay has software in it. You can shrink it to save space. And you can distribute it to other machines. Just pay attention to incompatible processor architectures.
- **Run your container on computational nodes**. To run with MPI simply type *mpirun singularity {exec|run} -o <overlay-name> <container-name> [executable]* (*4). Notice, that ideally you don't need to bind any external directories anymore.



More information is written at a blog at http://www.md-esg.eu/category/containers/

References

[1] Singularity documentation at https://sylabs.io/docs. Learn how to create a container image and how to interact with it, how to work with overlay and much more.

[2] Intel documentation at https://software.intel.com. Guidelines on installing Intel performance libraries (Intel MPI and Intel MKL).

Acknowledgement

This work was supported by the European Regional Development Fund in the IT4Innovations national supercomputing center - path to exascale project, project number CZ.02.1.01/-0.0/-0.0/-16_013/0001791 within the Operational Programme Research, Development and Education and by the Czech Science Foundation grant No. 20-18392S.