Evaluation of Containers for HPC

AUGUST 2018

AUTHOR:
Aleksander Wennersteen

CERN IT-CM-IS
Batch-HPC

SUPERVISORS(S):

Pablo Llopis
Carolina Lindqvist

Abstract

Some of the main challenges in scientific computing today deal with performance-preserving portability of software and reproducibility of the final results; likewise, with the advent of modern cloud computing, these, along with other issues like deployment, are also found in the sider software and computing world. Containers can help solve all of these issues by packing the software along with its dependencies together, in an easy-to-distribute and lightweight format. Herein we investigate the utility of Singularity, a HPC targeted container solution which overcomes the main issues with deploying more mainstream solutions such as Docker.

Singularity is found to be both suitable and easy to deploy with the current set-up at CERN. The performance costs are minimal in accordance with the previously reported figures for Singularity and does indeed behave well when submitted through Slurm. This report also considers the new possible extensions to the software-infrastructre enabled by containers that can run on several different systems without any additional compilation or configuration. Lastly, specific use-cases such as Fire Dynamics Simulator and Warp are containerised and deployed to the users.

Preface

I would like to thank my fellow summer students for the entire summer student experience. It would not have been the same without you. Moreover, I would like to thank the IT-CM group for an interesting summer, in particular, the wider batch group and, of course, the crown jewel — the HPC group.

Lastly, I would like to thank my supervisors Pablo and Carolina for a great working summer and, with help from Philippe, Håvard and Markus, many moments to look back upon fondly.

Contents

List of Figures

1. Introduction

Whilst processing data from the Large Hadron Colider (LHC) may be the most famous part of CERN computing, it is far from being alone. CERNs beam department constantly have to run simulations to see if the LHC is upgradable; the theorists have to run simulations to compare theory with the experiments as well as to guide the experimentalists' further search; In addition, simulations must be done for other experiments' design such as a future collider. Other departments such as engineering also need to perform simulation for constructions at CERN or fire simulations for verifying safety aspects in an ever changing environment such as at CERN.

Unlike most of the LHC data processing, which can be done on single machines without any special equipment given that the throughput per day is high enough, these all need access to High Performance Computing (HPC) equipment so that the calculations can communicate.

The current trend in scientific computing is that users are becoming less and less faithful to a single site, and that there are fewer and fewer applications that run. The end result of this is that we have very complicated softwares that need to run several places on potentially very different clusters. This is where the need for containers comes in from the user perspective. It would be very convenient if there was an easier way to move software around. Sure, on your personal machine it very often works to `./configure; make; make install;`. This, however, is often not the case with HPC clusters, and often the users will find limitations to what they can do themselves. Sys admins limited in time and often not responding immediately, this often results in unexplored options.

Such problems can be easily solved by using virtual machines (VMs) or containers such as Docker. However, in HPC, due to the communication needs this is ill advisable, and, until recently, impossible[1]. As we will explain in more detail in section 4, the standard in the software industry, Docker, is unsuitable for HPC loads due to missing networking capabilities and other performance impacting features. Whilst there are several other solutions out there, like CharlieCloud, Shifter. This project focused on Singularity, the arguably most mature solution.

Moreover, enabling containers could also expand the use-case for HPC clusters. Deep learning is becoming omnipresent, even in high energy physics. With work such as RDMA-Tensorflow and OSU-Caffe being done, i.e. taking industry standard deep learning libraries and optimising them for the HPC architecture, should also make the move to heterogenous HPC cluster an attractive choice for the training phase. Currently, it would be non-trivial to make such a move at the time-scales CERN operates at due to the need for consistency of the environments. However, a containerised environment could make such a move possible. This would allow for a more centralised, and thus more budget friendly, allocation of computing resources.

This report is organised as follows: chapter 2 introduces the basics of batch and high performance computing followed by a quick overview of the set-up at CERN in chapter 3. Afterwards follows chapter 4a discussion about containers, both HPC oriented and not. The final part outlines the work done to enable Singularity containers on the cluster as well as building containers in chapter 5 as well as appendix A which gives a full recipe for containerising the Warp application. This is completed with results, chapter 6; and discussion, conclusion in chapters 7 and 8.

[1] MVAPICH has recently released MVAPICH2-VIRT which enables MPI communications through virtualisation layers such as containers and VMs, albeit at a performance cost.

2. Batch and High Performance Computing

On modern computers users are used to clicking, or typing on the command line, to run a program. It launches instantaneously. This is in contrast to so-called batch computing. Here, you submit your program, typically called a *job*, which will run at some point in the future. This model is used in HPC so that the cluster is always utilised the most, and also so that users who need a large number of nodes can get their job executed at some point.

There are many types of batch jobs. For example cron jobs, which are tasks the system administrator has set to run on the machine every day, for example, are batch jobs. These can do things like empty the temporary files folder on the computer, or ensure that a part of the system is operational. Another example would be from High Throughput Computing (HTC) which is what most of CERNs compute needs can be classified as. HTC is characterised by the need to ensure a maximal amount of processed data over a long period of time.

Herein, however, we are only concerned with High Performance Computing (HPC) jobs. Here, the key thing is to provide an incredible amount of compute power, but over a short period of time. In order to obtain such compute power one must spread the computation out across multiple machines. This is more difficult, not just for the programmer, but also for the container, due to all the communication happening between the machines. In order to program effectively in this way programmers usually use the Message Passing Interface (MPI). Too much isolation, which is often what is wanted elsewhere, can result in a big loss of performance due to the networking. We will come back to this in chapter 4.

Usually a HPC cluster will have a distributed memory setup consisting of multiple nodes, a collection of CPUs, each node having a shared memory setup over several cores, see Fig. 2.1.

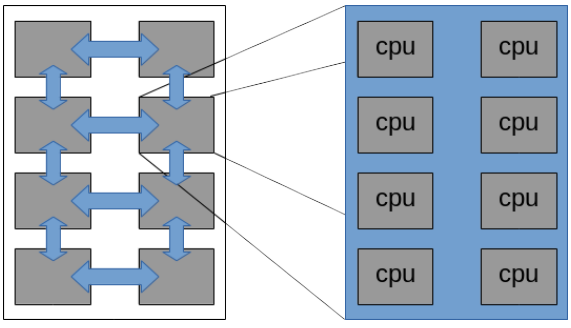

Figure 2.1: The memory layout of a typical HPC cluster. On the left is part of a cluster depicting the individual nodes with their interconnects giving a distributed memory unit. The right hand side shows an individual node composed of 8 cores with a shared memory.

Normally, in order to transfer data from one machine to another, one must go via the kernel, which will take care of a lot of underlying details. However, this introduces a lot of overhead. Thus, in HPC one usually uses Remote Direct Memory Access (RDMA). This requires some special hardware as well as software, but allows the communication to bypass the kernel and simply

allow one computer to directly access the memory of a remote machine in the same manner as it would access its own memory. Furthermore, HPC clusters often upgrade their networking to either low latency Ethernet cables or Infiniband interconnects.

The last part of a HPC cluster is the scheduler and resource manager. These days one often uses the same software for both, such as SLURM. Such software will take care of the job submission and allocation of computing resources on the cluster, and will try to give users access to resources in a way such that performance is optimised by e.g. giving the user machines which are physically close, or even with a network topology that suits the job. In the case of Slurm, Slurm also takes care of setting up MPI so that if the program is launched with Slurm then Slurm will find the correct version of MPI to run the program with rather than having to launch it with e.g. `mpirun`.

It is primarily the last two features, RDMA and the scheduler, that makes the introduction of any sort of containerisation difficult in HPC centres. Providing any sort of virtualisation is of course a contradiction towards performance, take RDMA which relies on direct hardware access. Afterall, the operating system is also a virtualisation level, and we always want to bypass it with, e.g., RDMA for performance reasons. Scheduling, on the other hand, might be less an obvious obstacle. The problem here lies more in the fact that common solutions such as Docker relies on daemons, thus making the process of scheduling more complicated.

3. Setup

The compute infrastructure at CERN batch consists of OpenStack managed virtual machines where each machine runs Puppet and Foreman for configuration management. This unified approach across the different machines makes it very easy to manage the system. Overall, this integration of cloud computing approaches allows CERN to take advantage of all the new technologies which are appearing in the open source market, often backed by large companies.

Puppet makes it very easy to modularise the set-up so that one can take advantage of inheritance to set-up more specialised machines. Moreover, the puppet manifests can access the global namespace such that the module setting up a specific software can be kept entirely separate to the manifest defining the machine it runs in. The machine manifest can then decide to redefine the variables used to set-up the software by referring to the global namespace in the environment that the machine lives in. For details see the Puppet documentation [1].

The HPC clusters at CERN are hyperconverged with CephFS shared file system and uses Slurm for job submission and cluster management. One cluster has low latency Ethernet whereas the other cluster has Infiniband interconnects, both support Remote Direct Memory Access (RDMA). There is also a third cluster used for lattice QCD by the theory group at CERN, this cluster is not included when this report refers to the current CERN HPC set-up. For an overview of the HPC setup see [2]. The further details of the setup not being relevant to this report we will not go into any details.

4. Containers

Containers have hit thes software industry hard the last few years. They allow us the escape ”dependency hell” and to let developers know that once they have got their program to run, it will run in the same way independently[1] of where it is being run. They are a form of operating system level virtualisation and thus provide some isolation from the host, although not as much as a hypervisor based virtual machine. This can be seen as a disadvantage, especially from the point of view of security, but also as an advantage. This is especially true in HPC where we can allow less isolation from the host machine as although we are concerned about security, we can trust our users and software more than the average cloud computing vendor.

The past couple of years containers have become omnipresent in computing due to the ever changing software stack which tends to rely on different versions of the same libraries or even programming languages. High Performance Computing centres have also started looking into this; both for the benefits of the system administrators, the users, and for reproducability for the wider community. Sadly, the standard solutions used in industry, such as Docker, have been found to be unsuitable for the needs of the HPC community.

Containers allow users to set up and try out software on their own machines, where they have root privileges and do not need to worry about affecting other people, pack it into a container and transfer the ready-made container to the cluster for running. Compared to the current solution of setting up the machines again on the cluster including potentially requesting updates from the system administrator is clearly much quicker and easier.

4.1 Docker

For a solid introduction to all aspects of Docker, as well as many issues regarding containers in general, see [3]. Docker is the industry standard and is therefore very well supported. Unfortunately, Docker is unsuitable for a variety of reasons, and, thus far, so are other solutions like runc and containerd (i.e. Open Container Initiative based solutions) as well as HPC specific applications like Shifter, also based on Docker. What is important to realise here is that is not the container iteself that is unsuitable, but the daemon. This is why Singularity, for instance, can run Docker images, giving the exact same benefits as using a regular Singularity image. Singularity, on the other hand, is simply a process, which makes it no different than other programs and will follow the rules already set on the cluster.

Summarised, the problems with docker are:

1. Privilege escalation
   (a) User is root inside the container
2. Too isolated for performance
   (a) Lack of direct access to host, must go via daemon
   (b) Unnecessary overhead induced by daemon

[1] No virtualisation is being performed by any of the current container technologies so one is still limited to running Linux on Linux and Microsoft on Microsoft, 64-bit on 64-bit, however, you can run Scientific Linux on CentOS, for example.

(c) Need for network bridges — unsuitable for MPI[2]

3. Daemon process does not integrate well with Slurm or other schedulers.

4.2 HPC Containers

Recall that we said that containers were a form of virtualisation, like virtual machines and the operating system for instance. Now that we talk about HPC specific containerisation, the reader should recall from chapter 2 that in HPC we like to bypass the kernel, i.e. a virtualisation layer. This is actually one of the key differences between HPC-specific and ”regular” containers. Most of the time, one would like slightly more separation and virtualisation in containers, to achieve similar levels of security as virtual machines and also be equally host independent. In HPC, however, most users would much prefer to run bare-metal if they could, with the same effort and portability. Indeed, many of the key features of Singularity is bypassing the virtualisation layer.

For a full overview of Singularity see the documentation [4] and the paper [5]. Note that there is both a users documentation and one for system administrators. Neither version is comprehensive by itself.

The following comparison table is presented, comparing some advantages and drawbacks of container technologies.

	Shifter	**CharlieCloud**	**Docker**	**Singularity**
Privilege Model	Chroot	UserNS	Root Daemon	UserNS
No trusted or privileged daemons	No	Yes	No	Yes
No process supervision	No	Yes	No	No
Sys Admin can limit capabilities	No	No	No	Yes
No additional network config	Yes	Yes	No	Yes
Access to host file system	Yes	Yes	Yes	Yes
Native GPU support	No	No	No	Yes
MPI and Infiniband support	Yes	Yes	No/No	Yes
Works with all major schedulers	No	Yes	No	Yes

The reader should, however, keep in mind that of these, thus far, Singularity is the most commonly supported and seemingly is the more mature technology and ecosystem. Nevertheless, sysadmins will have to expect to support multiple container technologies until one emerges as the winner.

[2] The latest MVAPICH2-VIRT overcomes this too some extent, but still more latency than what is generally acceptable.

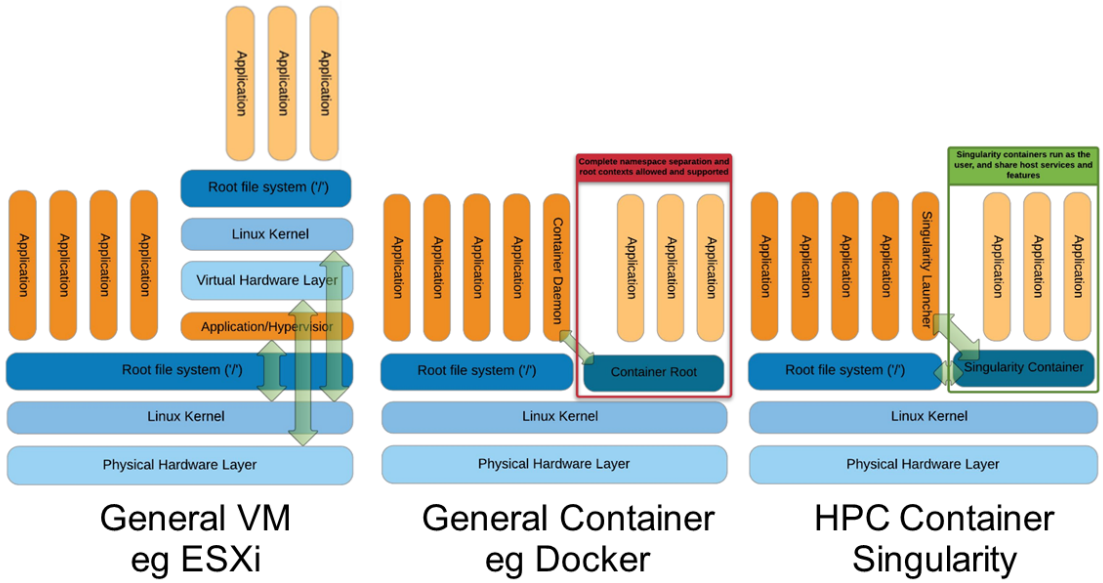

Figure 4.1: A comparison between virtual machines (VMs), standard containers (Docker) and HPC containers (Singularity). Note that the VM is completely isolated, Docker has very limited access to the physical hardware and runs on a daemon, Singularity has no daemon and is less isolated. Source: [6]

4.2.1 MPI in Singularity

MPI is perhaps the most ubiquitous member of the HPC software family, almost the definition of HPC codes one might even say. As such, it perhaps the main ingredients of a HPC container. Singularity has built in support for open MPI [5], and also work with the MPICH based MPI implementations.

The Open MPI/Singularity workflow works as follows:

1. `mpirun` is called by the resource manager or the user directly from a shell
2. Open MPI then calls the process management daemon (ORTED)
3. The ORTED process launches the Singularity container requested by the mpirun command
4. Singularity builds the container and namespace environment
5. Singularity then launches the MPI application within the container
6. The MPI application launches and loads the Open MPI libraries
7. The Open MPI libraries connect back to the ORTED process via the Process Management Interface (PMI)

This is also illustrated in Fig. 4.2. At this point the processes within the container run as they would normally directly on the host. This entire process happens behind the scenes, and from the user's perspective running via MPI. Moreover, Singularity parses command-line arguments in such a way that it feels natural, meaning that, if you forget all about what Singularity is, and just write the command like you would naively, it works.

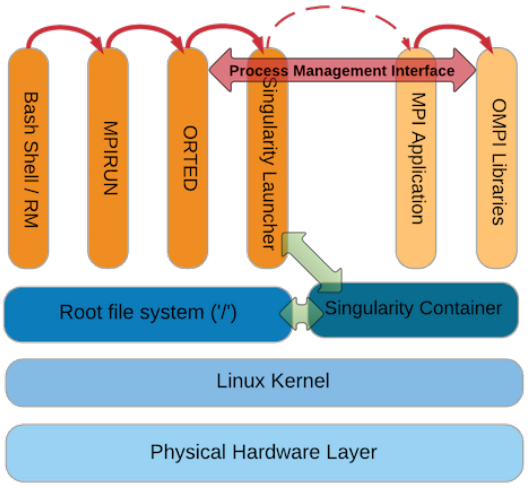

Figure 4.2: The ORTED daemon, or "MPI daemon" for all intents and purposes herein, communicates via the Process Management Interface (PMI) with the MPI libraries inside the container. This means that for MPI purposes the software inside the container behaves like normal software in user-space. Not that this refers to Open MPI, which is the best supported MPI distribution with Singularity. MPICH based MPI implementations are slightly different and slightly easier in a way. Similarly, it is slightly different when Slurm manages the MPI communication but the principles remain the same. Source: [6].

Fig. 4.3 places the container into the hardware. Note how Singularity sits right on top of the kernel like a normal process, but also at the same level as the root file system. Both whilst keeping the ability to communicate to the host file system. Compare this to Fig. 4.1, which also shows the situation for hypervisor based VMs and Docker-like containers.

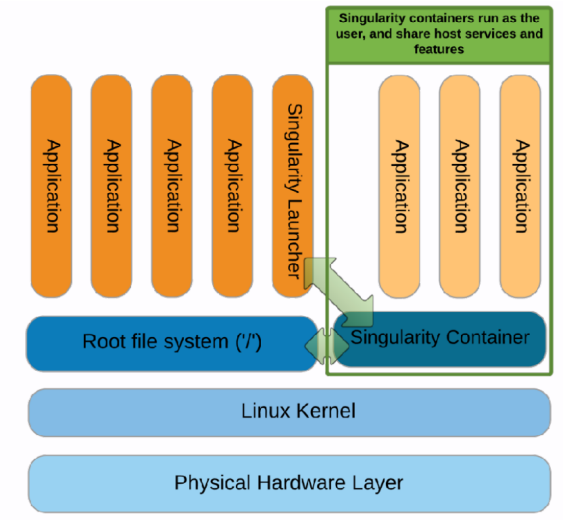

Figure 4.3: How Singularity fits into the computer. Notice that there is no ”Singularity daemon” running and that the container has direct access to the file system. Singularity runs as process in user space with all the associated benefits when it comes to accessing hardware. Source: [6].

4.3 Singularity images and deployment

One main point of discussion is whether to allow loop devices or not. These are the standard images that singularity produce. It consists of a single file, say `CentOS.img`, that contains everything needed for singularity to run it. The loop devices version has the advantage that the container is permanent unless the `--writable` option is passed. They do, however, have a known potential security issue [7, 8] that could allow an attacker to obtain root access. This immutability property means that the container, once created, is reproducible. One can expect that the result given by any software running inside the container will be the same. Moreover, if it ran once, it will run again.

The other possibility is to use so-called directory based images. These are created when the `--sandbox` option is passed during the build process. This results in a folder, say `CentOS.simg` that contains the same information as the previous image, `CentOS.img`. This does not require loop devices, but, the folder is just a regular folder on the computer and can thus be modified at any time, assuming a writable file system.

There are three sources of containers to be expected:

1. Provided by the HPC team for ease of setup
2. Provided by the users or externally as portable containers
3. Provided by CERN in general.

The first case is easy, the containers can be put somewhere on the file system for the users. In the second case the users will themselves transfer the container to their home directory. However, in the last case, also potentially in the first case, it would be useful to have some form of a central registry for containers. Indeed, this route has already been taken elsewhere. CERN has an excellent solution for this case, the CERN Virtual Machine File System (CVMFS). It is a read-only file system and repository where CERN, puts their software to be available from anywhere that mounts CVMFS, such as all the CERN compute machines! This approach to distributing images have already been implemented by the Open Science Grid and CERN experiments have ran tests.

This would make it easy to deploy and update containers for the users, and also have the added benefits of not having to allow loop devices, a potential security vulnerability. The read-only property also ensures the consistency of the container as it is used by multiple users.

4.3.1 A minimal Singularity Container

To show exactly how easy it is to produce a container the following example is provided. For more examples, see the appendices.

```
BootStrap: yum
OSVersion: 7
MirrorURL: http://mirror.centos.org/centos-\%{OSVERSION}/\%{OSVERSION}/os/$basearch/
Include: yum

%runscript
    echo "This is executed with the \"singularity run\" command only"

%post
    echo "The container is being built"

%environment
% sets up the environment at the end of the build process.
% works for both singularity run and singularity exec.

%files
% Imports files at build time that become a part of the image.
```

Listing 4.1: A minimal CentOS 7 container

This will for instance successfully execute `singularity exec centos.img echo "Hello World!"`, but will not contain most of the even most standard Linux utilities.

5. Implementing and Puppetising

The installation of singularity is very simple. In our case, it was already prepared and it was simply a matter of enabling it in puppet. That amounted to setting the variable `singularity = true` in the environment for the appropriate machine. For details on how to install Singularity we simply refer to the Singularity documentation [4].

5.1 Puppetising

As already mentioned, CERN machines are configured via Puppet, hence, any changes we make must be puppetised. We already mentioned the first change, namely enabling Singularity in the correct environment.

Next, experience suggested that we make the following change in the hostgroup found at `it-puppet-hostgroup-bi/data/hostgroup/bi/hpc/batch/workernode/test.yaml`[1]:

```
singularity::num\_loop\_devices: "\%{facts.processors.count}
mount\_home = no
```

The first is needed to allow loop devices, i.e. non-directory-based images and the second disables automatic mounting of the Linux home directory. The first change will be discussed several places elsewhere as it does present a potentially major security issue [7]. The latter prevents some subtle changes to the container environment, such as settings and variables polluting the Container, or more obvious ones such as refusing to run due to missing base-directory inside the container or having the container software overwritten by the home directory. This happens because the home directory on the host takes priority when mounted.

For a full overview of Singularity see the documentation [4]. Note that there is both a users documentation and one for system administrators. Neither version is comprehensive by itself.

Now, this `singularity::num_loop_devices` variable does not exist in puppet by default, obviously. To define it we create a file called `it-puppet-module-singularity/code/manifests/config.pp` along with some other files referenced therein. This combination allows CERN IT to have a central Singularity configuration which, for instance, does not allow loop devices by default. We can, however override them in our workernodes.

```
class singularity::config (
  $setuid             = $singularity::setuid,
  $allow_pid_ns       = $singularity::allow_pid_ns,
  $enable_overlay     = $singularity::enable_overlay,
  $config_passwd      = $singularity::config_passwd,
  $config_group       = $singularity::config_group,
  $config_resolv_conf = $singularity::config_resolv_conf,
  $mount_proc         = $singularity::mount_proc,
  $mount_sys          = $singularity::mount_sys,
  $mount_dev          = $singularity::mount_dev,
  $mount_home         = $singularity::mount_home,
  $mount_tmp          = $singularity::mount_tmp,
```

[1] This is a good example of the Puppet set-up, allowing for a lot of specialisation yet minimising code repetition. Reading from right to left we see that we are in the test environment of a hpc-batch workernode. In order to put this into all the machines, make the changes in workernode.yaml one level up.

```
  $mount_hostfs        = $singularity::mount_hostfs,
  $bind_path           = $singularity::bind_path,
  $user_bind_control   = $singularity::user_bind_control,
  $mount_slave         = $singularity::mount_slave,
  $container_dir       = $singularity::container_dir,
  $sessiondir_prefix   = $singularity::sessiondir_prefix,
  $num_loop_devices    = $singularity::num_loop_devices,
) inherits singularity {

  file{'/etc/singularity':
    ensure => directory,
  }

  file{'/etc/singularity/singularity.conf':
    ensure  => file,
    content => template('singularity/singularity.conf.erb'),
  }

  file{'/etc/singularity/init':
    ensure  => file,
    content => template('singularity/init.erb'),
  }

  file{'/etc/singularity/default-nsswitch.conf':
    ensure  => file,
    content => template('singularity/default-nsswitch.conf.erb'),
  }
}
```

These are all the variables that Singularity allows the administrator to set, as well as ensuring that certain files exists. The alfanumerical default value is then set in the `init` file specified.

Together, these changes enable Singularity and allows for full customisation down to the type of node. It also showcases the utility of Puppet.

5.2 Building Containers

Afterwards it became a matter of creating containers and finding out how they work. Example containers are provided as appendices at the end of the report and can be consulted along with [4]. Remember that the

There are 4 main sections in a Singularity recipe:

1. post – The post section is what is being executed during build time. This is where one installs software and generally sets up the container.
2. runscript – The runscript is what is executed with the singularity run command.
3. files – Files to be included from the local machine where the container is built.
4. environment – Setting up the environment, e.g. modifying the PATH.

The author found it very useful to use modules when building containers despite the fact that each container was created with one use case. This is probably often the case when a modulefile already exists, especially if it comes packed together with the application itself, such as the MPI versions from the CERN repository. Otherwise, use the `%environment` section of the recipe. That also ensures the reproducability of not just the container, but also the recipe.

Similarly, it is advisable to specify `--writable` when setting up the container initially as it will allow the builder to run the container as singularity shell `--writable` later. In this mode the container can be modified. This way the build process does not terminate because of a missing dependency.

6. Results

A main result was the confirmation of what is promised by the Singularity developers. Indeed, Singularity integrates well with MPI and Slurm, even on Infiniband networks. It can run all the MPI versions used at CERN currently and, to the extent we were able to check, different versions of MPI can be used inside/outside the container assuming that the versions are somewhat compatible. Note that this becomes even better when submitted through Slurm as the setup performed by Slurm at launch time sets up Singularity and MPI in the correct fashion. We also confirmed that Singularity left no extra network trace such as the network bridging often found with added virtualisation layers, in stead it is able to connect directly onto the existing infrastructure provided by the host.

Containers for Fire Dynamics Simulator and a container containing Python + MPI + Warp, a library needed for some of the physicists, were produced and shipped to the users. With these as templates, making containers in the future should be easier. See the appendices for examples.

We were also able to verify the possibility of running Singularity in the following modes:

1. MPI running outside the container;
2. MPI processes inside the container, including Slurm managed MPI;
3. A mixture of MPI processes inside and outside the container;
4. Hybrid MPI (outside) and OpenMP (inside) for node and thread level parallelism.

Nevertheless, running MPI processes inside the container is not recommended. Only a toy example ran nicely, and based upon external reports, such as the Singularity documentation [4] most uses will require a modification. See the discussion coming up in section 7.6.

As will be shown in the following benchmarks, the performance penalty for using containers is minimal. As can be seen from Fig. 6.1, the differences between the container and bare-metal are indeed so small that occasionally the container ran faster. This is also true for the example using Python + MPI in the form of mpi4py.

The data presented is the average difference over 3 runs on the low-latency Ethernet cluster. Therefore, Fig. 6.2 is cut off at 65kb message size as the run-time continues to scale linearly with the exponentially growing message sizes. These very large run-times, together with the low number of runs used, also explain the apparent outlier 1mb message size data point. The variance in individual MPI runs are on the same scale as the difference, hence without a larger sample size such results are not statistically unlikely. Overall the results agree with previous studies.

Running two containers on the same host (processor core) that need to communicate is actually slower than having the MPI processes run on different cores. This is nothing to be concerned about. Whereas in ”normal” container usage, say Docker for microservices on a server, you would often like to run several containers on the same processor for isolation. This is, however, not a HPC use-case. In HPC we only run a single software at a time, any thing else would result in unnecessary performance degradation.

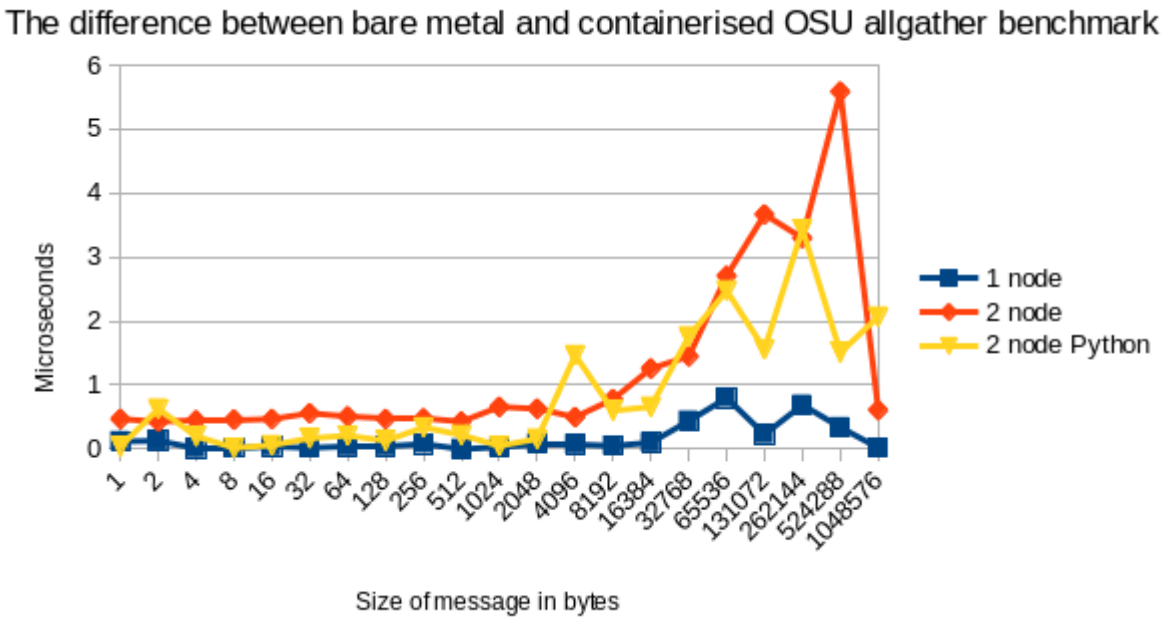

Figure 6.1: The difference in run-time of the OSU allgather benchmark between the containerised and bare-metal versions. The benchmark ran on a low-latency Ethernet cluster using Open MPI 3.0.0 and mpi4py for the python version. Note how, especially for message sizes above 65kb, the induced latencies vary a lot. Significantly more runs would probably have smoothed the graph, although here one sees how the variance is in fact comparable to the difference between the two versions.

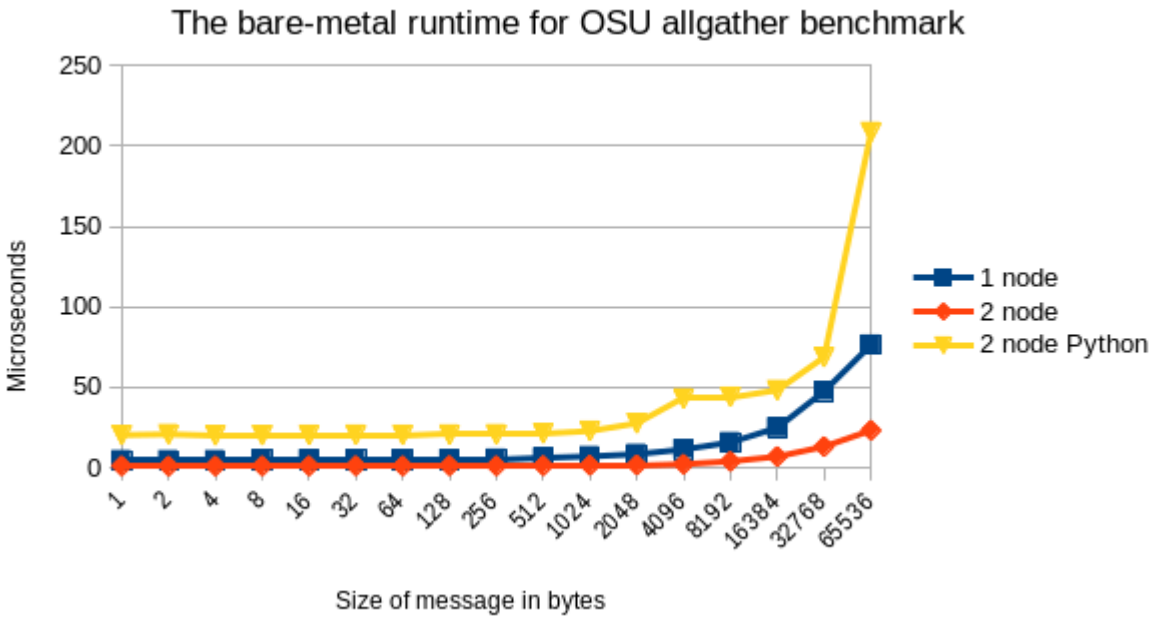

Figure 6.2: The bare-metal run-time of the OSU allgather benchmark that was ran on a low-latency Ethernet cluster using Open MPI 3.0.0 and mpi4py for the python version. Note how all datapoints behave nicely, with the python version having some extra latency induced.

7. Discussion

7.1 Performance

Consistent with other reports we saw no significant performance degradation using Singularity. This was true not only for the standard OSU benchmarks but also from a Python version using mpi4py. We also checked for any extra network bridges or similar, of which there were none.

For further benchmarking results comparing Singularity to bare-metal the reader should consult the extensive literature available. A starting point is the Singularity paper. These are pure MPI benchmarks that only does message passing. In other words, they measure the pure performance drop in the message passing. Furthermore, Singularity should not notably affect pure computing performance, see [9] which benchmarks container performance for IO, CPU, GPU, network latency and RAM.

This could enable more users to take advantage of the cluster in a performant way resulting in a higher utilisation of the resources and less waiting time for the users. In particular it could be possible that users could be allowed access to the clusters on a case by case basis where time is critical for them but the speedup required is insufficient to warrant regular access. Currently, it would likely take too long to set-up the software on the machine in addition to the processing of access granting. This would, of course, require some extra steps such as having a more homogeneous accounting and budgeting system. Nevertheless, the author does not see any challenges in implementing this provided that such a solution is deemed interesting.

7.2 Choice of container

Although no other containers were evaluated the author does feel that for now singularity is the best choice for production HPC. This is based upon reading comparison articles and looking for documentation online. Although other options are seemingly gaining traction, only Singularity seems to be widely supported around the HPC community for now. Singularity is also both open source and possess the capability of running Docker images. This means that not only will CERN have to possibility to stay in control of the software should it be necessary, but it is also capable of running the industry-standard images for ease of portability.

7.3 Issues with Singularity

Singularity has built in support for MPI, in particular, openMPI; nonetheless, no problems were found using MVAPICH2 or MPICH. The version which is the most supported is openMPI 2 and 3. For these, no problems with MPI versions inside/outside the container have been found or elsewhere reported. However, for other versions of MPI, the recommended steps would be to keep the same version both inside and outside the container. This is also true in other cases such as Infiniband libraries.

7.4 Implementation

As shown in section 5, enabling Singularity on the cluster was effectively trivial. Moreover, few problems were encountered whilst building and running the containers apart from general issues with installing software. One important takeaway was that initially, as an inexperienced container builder, it was tricky to determine which errors came from problems with Singularity, wrong host set-up, wrong container set-up, or wrong Slurm/MPI parameters. The problem was never with Singularity.

7.5 Integration with other submission systems at CERN

Singularity containers have already been successfully submitted through the normal batch service, HTCondor, and the BOINC serivce LHCHome. In a future utopia one can imagine a world where a user who does not really care about where her job is ran could submit a job with BOINC in mind that offloads to the regular batch or HPC service when those clusters are underutilised. This would allow for faster computations from the users perspective and would make it easier to justify buying larger clusters for CERN batch services.

7.6 Running nested MPI processes

Even though it was not experienced, the author would like to mention that certain applications has been reported to need an extra ssh wrapper inside the container which does extra forwarding to allow the container to ssh into a different container. This should not be needed whenever MPI/Slurm is used in the standard manner, but it should be checked for programs that manage this manually through ssh or similar methods. In this case it might happen that the application tries to look for the containerised software *outside* the container.

The need to for a ssh wrapper also arises if one uses the model of running MPI both inside and outside the container. The wrapper is used for having the *internal* (with respect to the container) MPI communicate with the *global* MPI processes, i.e. internode communication.Note that Slurm managed jobs, at least with Open MPI, seems to be able to handle this mixed mode.

For example, one can run `srun -N 8 Singularity exec CentOS.img mpiexec -np 8 foo` to run the program `foo` on eight cores inside a node, and on eight nodes. In this case only 8 singularity images are used. Referring back to Fig. 2.1; each of the eight nodes on the left hand side would run one Singularity container in the shared memory of the node which again would spawn 8 MPI processes, one per core. Note that this model is not particularly portable, but could, in theory, be slightly faster than running 8 singularity containers on the same node. The author advises benchmarking on the user side in case such an option is considered, as benchmarking this was outside the scope of this project.

8. Conclusions

Altogether, Singularity seems to be a sufficiently mature framework and is indeed supported by more and more HPC centres. The benefits are clear, both from the users' point of view with ease of portability and reproducibility, and from the system administrators' point of view for having a cluster which will be easier to manage. As promised, containers provide a level of separation but without the performance and space cost of a virtual machine along with modularisation and reproducibility. Singularity in particular packs this in a way that unlike, say, Docker, is acceptable to run on a HPC cluster from a security point of view, with respect to integrating with schedulers and in terms of performance preservation.

Furthermore, Singularity integrates well with Slurm; in fact, a Slurm managed job was much easier to set-up than a job launched with `mpirun`. It also does seemingly not put any restrictions on the user in terms of e.g. hybrid MPI + OpenMP jobs. Likewise, Singularity can make use of RDMA both over Ethernet and Infiniband and is ready for the future when it comes to native GPU support. This ease of compatibility with external, HPC-specific hardware hopefully also indicates that it will be easy to integrate with any future development in computer hardware.

In order to give the users maximal flexibility, it is recommended to allow loop device based images as well as facilitating for distribution of directory based images through CVMFS. Lastly, when building images, in HPC there is always a trade-off between portability and performance; this also applies to containers and is something to be considered when building containers.

Future work

Sticking strictly to the Singularity side more work is needed in making more containers of real applications and benchmarking them. This will benefit from the ongoing attempt to collect user application benchmarks at CERN HPC. Furthermore, the author regrettably did not realise that a further experiment combining MPI and OpenMP would have been useful due to the rather common model of using MPI for internode communication by OpenMP for intranode communication. Along with the nested MPI version presented above this should be benchmarked. Open MPI and Singularity has done work on both ends for better integration, testing should now be done to see how this affects the stability of jobs, and whether or not Slurm affects this. Testing of MVAPICH2-virt should also be done, and its capabilities being taken into account when considering changes in the CERN architecture.

The rest of the future work goes more out to the general evolution of the CERN computing framework, and several of the future enhancements that I have mentioned throughout are indeed being considered and even worked on. In particular there already is some support for unpacking container images on CVMFS and the plan for opportunistic allocation of HPC resources is also being pursued.

Lastly, as alluded in the introduction and throughout the report, exploring whether a containerised submission flow could help reduce cost and increase computer power should be considered. With Singularity one can centralise much of the computing resources used for offline processing, from plain CPUs to GPUs, much of it can be brought onto a common infrastructure.

Bibliography

[1] "Puppet documentation." https://www.puppet.com/docs. Accessed: 16-08-2018. 4

[2] Carolina Lindqvist, Pablo Llopis, Nils Høimyr, and Dan van der Ster, "Integrating hpc into an agile and cloud-focused environment at cern." https://indico.cern.ch/event/587955/contributions/2937098/attachments/1679299/2697270/CHEP_2018.pdf. Accessed: 16-08-2018. 4

[3] A. Mouat, *Using Docker*. O'Reilly, 2015. 5

[4] "Singularity 2.5.1 documentation." https://www.sylabs.io/docs. Accessed: 16-08-2018. 6, 11, 12, 13

[5] Kurtzer GM, Sochat V, and Bauer MW, "Singularity: Scientific containers for mobility of compute," *PLoS ONE*, vol. 5, no. 12, 2017. 6, 8

[6] Kurtzer GM, "Singularity - containers for science." http://www.hpcadvisorycouncil.com/events/2017/stanford-workshop/pdf/GMKurtzer_Singularity_Keynote_Tuesday_02072017.pdf. Accessed: 16-08-2018. 7, 8, 9

[7] "SVG advisory: image mounting via Singularity." https://wiki.egi.eu/wiki/SVG:Advisory-SVG-2018-13999. Accessed: 16-08-2018. 9, 11

[8] "Filesystem mounts in user namespaces." https://lwn.net/Articles/652468/. Accessed: 16-08-2018. 9

[9] C. Arango, R. Dernat, and J. Sanabria, "Performance evaluation of container-based virtualization for high performance computing environments." https://arxiv.org/pdf/1709.10140.pdf. Accessed: 16-08-2018. 16

A. Warp Container

This is the recipe for the Warp container that was produced, and is provided as an example of a full container recipe ready for the user. What is interesting here is also the installation of Open MPI 3 with mpi4py in the container as this has before been proven difficult. See https://github.com/singularityware/singularity/issues/1689.

```
BootStrap: yum
OSVersion: 7
MirrorURL: http://mirror.centos.org/centos-%{OSVERSION}/%{OSVERSION}/os/$basearch
Include: yum


%post
    mkdir -p /hpcscratch/
    yum -y install yum-utils
    yum-config-manager --add-repo http://linuxsoft.cern.ch/internal/repos/hpc7-stable/
        x86_64/os
    yum-config-manager --add-repo http://linuxsoft.cern.ch/internal/repos/batch7-qa/
        x86_64/os
    yum-config-manager --add-repo http://linuxsoft.cern.ch//epel/7/x86_64
    rpm --import http://linuxsoft.cern.ch/epel/RPM-GPG-KEY-EPEL-7
    yum-config-manager --add-repo http://linuxsoft.cern.ch//cern/centos/7/os/x86_64
    yum -y install openssh
    yum -y install gcc
    yum -y install gcc-c++
    yum -y install make
    yum -y install wget
    yum -y install which
    yum -y install environment-modules
    yum -y install munge
    yum -y install freeipmi
    yum -y install --nogpgcheck install slurm-libpmi-17.11.7-2.mvapich2_patched.el7
    yum -y --nogpgcheck install openmpi300
    yum -y install --nogpgcheck libibverbs
    rm /etc/modulefiles/mpi/.nodesettings || true
cat >> /etc/modulefiles/mpi/.nodesettings << EOF
#%Module
setenv MV2_USE_RDMA_CM 1
setenv MV2_USE_IWARP_MODE 1
EOF

    yum -y install git
    yum -y install python36
    yum -y install python36-devel
    wget https://bootstrap.pypa.io/get-pip.py
    python36 get-pip.py

    ln -s /usr/bin/python3.6 /usr/local/bin/python3

    . /usr/share/Modules/init/sh
    module load mpi/openmpi/3.0.0

    pip install mpi4py
    pip install numpy
```

```
    pip install scipy
    pip install h5py
    pip install python-dateutil

    yum -y install libX11-devel

    git clone https://bitbucket.org/dpgrote/pygist.git
    cd pygist
    python36 setup.py config
    python36 setup.py install

    pip install Forthon
    mkdir -p /warp
    cd /warp
    wget https://bitbucket.org/berkeleylab/warp/downloads/Warp_Release_4.5.tgz
    tar -xvf Warp_Release_4.5.tgz
    cd warp
    git pull
    cd pywarp90

cat >> /warp/warp/pywarp90/setup.local.py << EOF
if parallel:
    library_dirs += ['/usr/local/mpi/openmpi/3.0.0/lib64']
    libraries += ['mpi','mpi_mpifh']
EOF

cat >> /warp/warp/pywarp90/Makefile.local3.pympi << EOF
FCOMP = -F gfortran
FCOMPEXEC = \-\-fcompexec ftn
EOF


    make clean
    make pclean
    make install3
    make clean
    make pinstall3
    make pclean


    cd /warp/warp/warp_test/
    python3 runalltests.py
```

The Warp container will be used by physicists at CERN who run simulations for the Advanced WAKEfield Experiment (AWAKE).

B. Minimal CERN HPC container

Here we present a starting point for creating a Singularity container for CERN HPC.

```
BootStrap: yum
OSVersion: 7
MirrorURL: http://mirror.centos.org/centos-%{OSVERSION}/%{OSVERSION}/os/$basearch
Include: yum

%post
    mkdir -p /hpcscratch/

    %setting up CERN yum repos
    yum -y install yum-utils
    yum-config-manager --add-repo http://linuxsoft.cern.ch/internal/repos/hpc7-stable/
        x86_64/os
    yum-config-manager --add-repo http://linuxsoft.cern.ch/internal/repos/batch7-qa/
        x86_64/os
    yum-config-manager --add-repo http://linuxsoft.cern.ch//epel/7/x86_64
    rpm --import http://linuxsoft.cern.ch/epel/RPM-GPG-KEY-EPEL-7
    yum-config-manager --add-repo http://linuxsoft.cern.ch//cern/centos/7/os/x86_64

    % generally needed basic functions
    yum -y install make
    yum -y install wget
    yum -y install which
    yum -y install openssh
    yum -y install environment-modules
    yum -y install vim
    yum -y install git

    % GCC and C++ headers
    yum -y install gcc
    yum -y install gcc-c++

    % installing MPI
    yum -y install munge
    yum -y install freeipmi
    yum -y install --nogpgcheck install slurm-libpmi-17.11.7-2.mvapich2_patched.el7
    yum -y --nogpgcheck install openmpi300
    yum -y install --nogpgcheck libibverbs

    % required MPI modulefile settup due to current configuration
    rm /etc/modulefiles/mpi/.nodesettings || true
cat >> /etc/modulefiles/mpi/.nodesettings << EOF
#%Module
setenv MV2_USE_RDMA_CM 1
setenv MV2_USE_IWARP_MODE 1
EOF

    % Installing python3.6 as python3 and pip
    yum -y install python36
    yum -y install python36-devel
    wget https://bootstrap.pypa.io/get-pip.py
    python36 get-pip.py
    ln -s /usr/bin/python3.6 /usr/local/bin/python3
```

First, the container sets up the required CERN yum repos, although other repos can be used. The advantage of the CERN repos is that they are known to be configured correctly. For example, some of these pre-compiled MPI versions, the Ubuntu ones, do not support multithreading by default. Other times the main reason to go for a container is that the CERN repos do not contain the wanted software and/or versions, so other repos should be considered.

Then several standard things are installed, such as Vim, Git, GCC, OpenMPI and the modules package. The modules package is not strictly needed as the recipe can specify in `%environmnt` all the paths, but since the CERN MPI versions comes with a modulefile this seems easier. Just remember to source `/usr/share/Modules/init/sh`, or another shell such as bash, before attempting to `module load`.

Lastly Python 3.6 along with pip is installed. This was largely done because of the missing pip in the CERN repo Python version. Applications not needing Python can of course remove this. Similarly, Vim was included mostly for the benefit of the container developer. The author well knows the annoyance of having to rebuild the container upon realising that there was in fact no text editor in the container and not having passed the `--writable`.