








A Subset of the CERN Virtual Machine File System: Fast Delivering of Complex Software Stacks for Supercomputing Resources

Alexandre F. Boyer^{1,2}  , Christophe Haen¹ , Federico Stagni¹ ,
and David R. C. Hill² 

¹ European Organization for Nuclear Research, Meyrin, Switzerland

alexandre.franck.boyer@cern.ch

² Université Clermont Auvergne, Clermont Auvergne INP, CNRS, Mines Saint-Etienne, LIMOS, 63000 Clermont-Ferrand, France

Abstract. Delivering a reproducible environment along with complex and up-to-date software stacks on thousands of distributed and heterogeneous worker nodes is a critical task. The CernVM-File System (CVMFS) has been designed to help various communities to deploy software on worldwide distributed computing infrastructures by decoupling the software from the Operating System. However, the installation of this file system depends on a collaboration with system administrators of the remote resources and an HTTP connectivity to fetch dependencies from external sources. Supercomputers, which offer tremendous computing power, generally have more restrictive policies than grid sites and do not easily provide the mandatory conditions to exploit CVMFS. Different solutions have been developed to tackle the issue, but they are often specific to a scientific community and do not deal with the problem in its globality. In this paper, we provide a generic utility to assist any community in the installation of complex software dependencies on supercomputers with no external connectivity. The approach consists in capturing dependencies of applications of interests, building a subset of dependencies, testing it in a given environment, and deploying it to a remote computing resource. We experiment this proposal with a real use case by exporting Gauss - a Monte-Carlo simulation program from the LHCb experiment - on Mare Nostrum, one of the top supercomputers of the world. We provide steps to encapsulate the minimum required files and deliver a light and easy-to-update subset of CVMFS: 12.4 Gigabytes instead of 5.2 Terabytes for the whole LHCb repository.

Keywords: Supercomputer · Software distribution · Automation · CVMFS · Monte Carlo simulation

1 Introduction

To study the constituents of matter and better understand the fundamental structure of the universe, HEP collaborations rely on complex software stacks

© Springer Nature Switzerland AG 2022

A.-L. Varbanescu et al. (Eds.): ISC High Performance 2022, LNCS 13289, pp. 354–371, 2022.

https://doi.org/10.1007/978-3-031-07312-0_18

and a worldwide distributed system to process a growing amount of data: the World Wide LHC Computing Grid (WLCG) [23]. The infrastructure involves 170 computing centers, 1 million cores and 1 exabyte of storage spread around 42 countries.

Delivering a reproducible environment along with up-to-date software across thousands of heterogeneous computing resources is a major challenge: Buncic et al. designed CernVM and CVMFS (CernVM-File System) [16] to tackle it by decoupling the software from the Operating System.

CernVM [20] is a thin Virtual Software Appliance of about 150 Mb in its simplest form. It supports a variety of hypervisors and container technologies and aims to provide a complete and portable user environment for developing and running HEP applications on any end-user computer and Grid Sites, independently of the underlying Operating Systems used by the targeted platforms.

CVMFS [20] is a scalable and low-maintenance file system optimized for software distribution. CVMFS is implemented as a POSIX read-only file system in user space. Files and directories are hosted on standard web servers and mounted on the computing resources as a directory. The file system performs aggressive file-level caching: both files and file metadata are cached on local disks as well as on shared proxy servers, allowing the file system to scale to a large number of clients [16].

This approach has been mainly adopted by the HEP community and is now getting users from various communities according to Arsuaga-Ríos et al. [3]. In a few years, it has become the standard software distribution service on Grid Sites of WLCG. Nevertheless, computing infrastructure and funding models are changing, and national science programs are consolidating computing resources and encourage using cloud systems as well as supercomputers, as Barreiro et al. explain [5]. CVMFS developers have extended the features of the file system and have provided additional tools to support clouds [36, 46] and supercomputers [9].

Supercomputers are highly heterogeneous architectures that pose higher integration challenges than traditional Grid Sites. Many supercomputers do not allow a CVMFS client to be mounted on the worker nodes and/or do not provide external connectivity, which is critical to work with CVMFS. CVMFS tools designed to interact with High-Performance Computing sites are aimed at administrators of scientific communities that would like to integrate their workflows on such machines: they ease some steps of the process but may require additional efforts on behalf of the administrators.

In this study, we aim to automate the whole process and reduce these additional efforts by providing a utility able to extract, test and deploy parts of CVMFS on supercomputers not having outbound connectivity. Section 2 briefly introduces CVMFS and the ecosystem developed around it, in order to deal with supercomputers. Section 3 focuses on the design of the utility, the steps to extract software dependencies and to deploy them on a given supercomputer. Finally, Sect. 4 presents a use case and the obtained results in detail.

2 Context

2.1 CVMFS to Distribute Software on Grid Resources

At the beginning of 2021, CVMFS was managing about 1 billion files delivered to more than 100,000 computing nodes by (i) 10 public data mirror servers - called *Stratum1s* - located in Europe, Asia and the United States and (ii) 400 site-local cache servers [8].

To keep the file system consistent and scalable, developers conceived CVMFS as a read-only file system. Release managers - or continuous integration workers - aiming to publish a software release has to log in to a dedicated machine - named *Stratum0* - with an attached storage volume providing an authoritative and editable copy of a given repository [11]. Changes are written into a staging area until they are committed as a consistent changeset: new and modified files are transformed into a content-addressed object providing file-based deduplication and versioning. In 2019, Popescu et al. [43] introduced a gateway component, a web service in front of the authoritative storage, allowing release managers to perform concurrent operations on the same repository and make CVMFS more responsive (Fig. 1.1.b and 1.2.b).

The transfer of files is then done lazily via HTTP connections initiated by the CVMFS clients [43] (Fig. 1.3.b). Clients request updates based on their Time-to-Live (TTL) value, which is generally about a few minutes. Once the TTL value expires, clients download the latest version of a manifest - a text file located in the top-level directory of a given repository composed of the current root hash, metadata and the revision number of this repository - and make the updated content available. Dykstra et al. [27] provide additional details about data integrity and authenticity mechanisms of CVMFS to ensure that data received matches data initially sent by a trusted server. This pull-based approach has been proven to be robust and efficient, according to Popescu et al. [43], and has been widely used to distribute up-to-date software on grid sites for many years (Fig. 1.2.a). Figure 1 presents a simplified schema summarizing the software distribution process on grid sites via CVMFS.

Users may need to use various versions of software on heterogeneous computing resources implying different OS and architectures. To provide a convenient environment for the users, release managers generally provide software along with build files related to many architectures, OS and compilers. Framework for building and installing scientific software on heterogeneous systems can be used to supply CVMFS with build files. Easybuild [28], Spack [49], Nix [40] or Gentoo [33] are popular choices in this area [17, 56, 57].

2.2 Software Delivery on Supercomputers

Communities working around the Large Hadron Collider (LHC) [21] have extensively used WLCG and CVMFS to process a growing amount of data. This approach was reliable during LHC Run1 but has demonstrated its limit. According to the analysis of Stagni et al. [50] on the use of CPU cycles in 2016, all the LHC

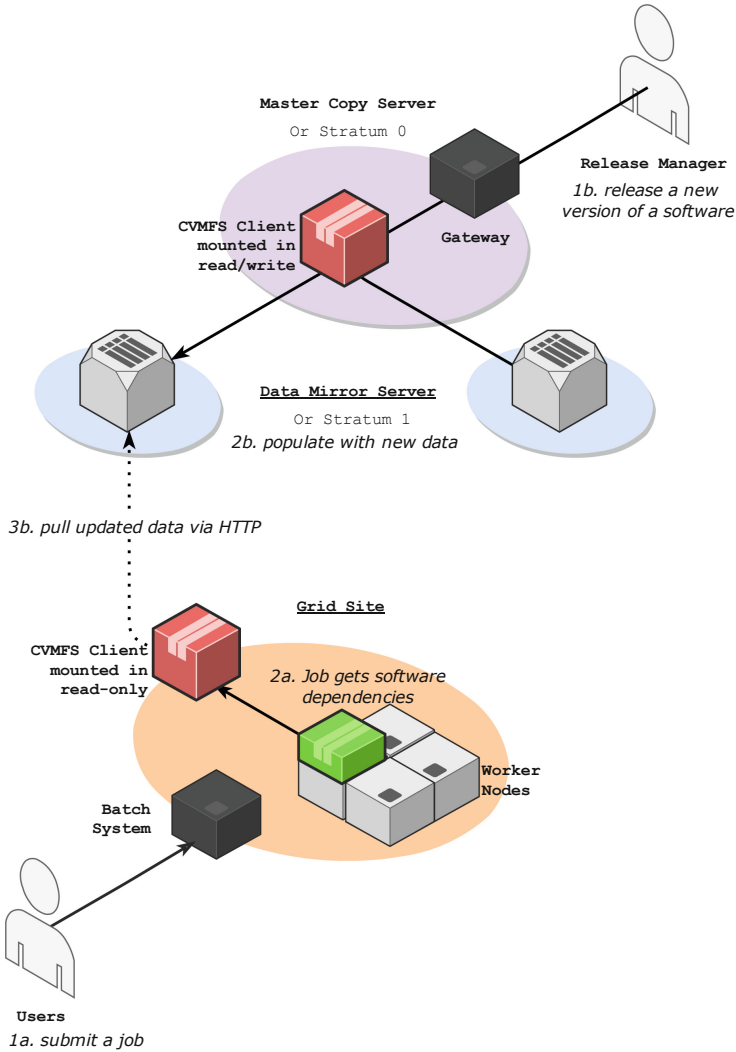


Fig. 1. Schema of the CVMFS workflow on Grid Sites: (a) the steps to get software dependencies from the job; (b) the steps to publish a release of a software in CVMFS.

experiments have consumed more CPU-hours than those officially pledged to them by the WLCG: they found ways to exploit opportunistic and not officially supported resources. Moreover, in the High-Luminosity Large Hadron Collider (HL-LHC) [2] era, experiments are expected to produce up to an order of magnitude more data compared to the current phase (LHC Run2). To keep up with the

computing needs, experiments have started to use supercomputers. They offer a significant amount of computing power and would potentially offer a more cost-effective data processing infrastructure compared to dedicated resources in the form of commodity clusters, as Sciacca emphasizes [45]. Nevertheless, supercomputers have more restrictive security policies than Grid Sites: they do not allow CVMFS to be mounted on the nodes by default and many of them have limited or even no external connectivity. The LHC communities have developed different solutions and strategies to cope with the lack of CVMFS, which is a critical component to run their workflows.

Stagni et al. [51] rely on a close collaboration with some supercomputer centers - Cineca in Italy and CSCS in Switzerland - to get CVMFS mounted on the worker nodes. Nevertheless, their strategy is limited to a few supercomputers and their approach would be difficult to reproduce on a large number of supercomputers: most of them do not allow such collaboration.

To deal with the lack of CVMFS on supercomputers with outbound connectivity, Filipčić et al. studied two solutions: *rsync* and *Parrot* [31]. The first solution consisted in copying the CVMFS software repository in the shared file system using *rsync*: a utility aiming to transfer and synchronize files and directories between two different systems. *rsync* added a significant load on the shared file system of the supercomputers and required changes in the repository absolute paths. The second solution was based on Parrot: a utility copied on the shared file system of the supercomputer, usable without any user privileges. Parrot is a wrapper using *ptrace* attached to a process that intercepts system calls that access the file system and can simulate the presence of arbitrary file system mounts, CVMFS in this case. Nevertheless, the solution was “unreliable in a multi-threaded environment” [31] because it was unable to handle race conditions. These methods did not constitute a production-level solution but contributed to further and future advanced solutions.

In recent years, developments in the Fuse user space libraries and the Linux kernel have lifted restrictions for mounting Fuse file systems such as CVMFS. Developers of CVMFS have integrated these changes and designed a package called *cvmfsexec* [26], which allows mounting the file system as an unprivileged user. The program needs a specific environment to work correctly: (i) external connectivity; (ii) the *fusermount* library or unprivileged namespace mount points or a setuid installation of *Singularity* (efficient High-Performance Computing container technology). Blomer et al. provide additional details about the package [10].

Communities exploiting supercomputers that do not provide outbound connectivity cannot directly benefit from *cvmfsexec*: the package still needs to pull updated data via HTTP, which is not available in such context. We can distinguish two cases: (i) supercomputers that grant outside network or specific service access to a limited number of nodes and (ii) supercomputers that do not provide nodes with any external connectivity at all.

Tovar et al. recently worked on the first case [54]. They managed to build a virtual private network (VPN) client and server to redirect network traffic from the workloads running on the worker nodes to external services such as CVMFS. In this configuration, the VPN client runs on a worker node along with the job, while the VPN server is hosted on one of the specific nodes of the supercomputer and can interact with external services. Communities working on supercomputers from the second case cannot leverage the solution developed by Tovar et al.

O'Brien et al., one of the first teams to work with supercomputers in the LHC context, address the lack of external network access by copying part of it to the shared Lustre file system accessible by the WNs [41]. The approach (i) worked because the environment of the supercomputer was similar to a grid site one, (ii) required changes in the CVMFS files and (iii) degraded the performance of the software as Angius et al. described [42]. To tackle the latter issue on the Titan supercomputer, Angius et al. moved the software to a read-only NFS server [42]: this eliminated the problem of metadata contention and improved metadata read performance.

Similarly, on the Chinese HPC CNGrid, Filipčić regularly packed a part of CVMFS in a tarball. Filipčić provided a deployment script to install the software and fix the path relocation on the shared file system to the local system administrators: they were then responsible for getting and updating the CVMFS tarball on the network when requested [30].

To help communities to unpack a CVMFS repository in a file system, a team of developers designed *uncvmfs* [37]. The utility deduplicates files of a software stack: it populates a given directory with the CVMFS files that are then hard-linked into it, if possible. The program was used, in combination with Shifter [34], a container technology providing a reproducible environment, in the context of the integration of the ALICE and CMS experiments workflows on the NERSC High-Performance Computing resources [29,38]. As a proof of concept, Gerhardt et al. used *uncvmfs* to deduplicate the ATLAS repository and copy it into an ext4 image - about 3.5 Tb of data containing 50 million files and directories -, compressed into a 300 Gb squashfs image; and Shifter to provide a software-compatible environment to run the jobs [34]. Despite encapsulating the files in a container reduced the startup time of the applications, the solution generated large images, long to update and deliver on time.

To cope with large images, Teuber and the CVMFS developers conceived *cvmfs_shrinkwrap* [52]. The tool supports *uncvmfs* features with certain optimizations and delivers additional features: *cvmfs_shrinkwrap* can extract specific files and directories based on specification files, deduplicate them, making them easy to export in various formats such as squashfs or tarball. In this way, the following operations remain on behalf of the user communities: (i) trace their applications - meaning, in this context, “capturing all their dependencies and their locations in the file system” -, (ii) call *cvmfs_shrinkwrap* to get a subset of CVMFS composed of the minimum required files, and (iii) export this subset in

a certain format and deploy it on sequestered computing resources to run their jobs.

Douglas et al. already described such a project in an article [7], but the work remains specific to the ATLAS experiment. They use *uncvmfs* to produce a large image that has to be filtered afterward. In this paper, we aim at assisting various user communities in this process by providing an open-source utility that would take applications of interest in input and would output - with the help of *cvmfs.shrinkwrap* - a subset of CVMFS with the minimum required files to run the given applications, in combination with a container image if needed. To our knowledge, no paper has already covered the subject.

3 Design of the CVMFS Subset Builder

3.1 Input and Output Data

The utility takes a directory as input that should contain: (i) a list of applications of interest (**apps**): a command along with its input data in a separate sub-directory for each application to trace; and/or (ii) a list of files composed of paths to include in the subset of CVMFS (**namelists**). Additionally, user communities can embed a (iii) container image compatible with Singularity to get a specific environment to trace and test the applications; (iv) and a configuration file to fine-tune the utility with variables related to the deployment process, or information about repositories. A schema of the inputs is available in Fig. 2.

The expected output can take different forms depending on the utility configuration:

- The subset of CVMFS, generated as a standalone. In this case, administrators representing their user communities need to provide the right environment by themselves, which might also involve discussions with the system administrators.
- The subset of CVMFS embedded within the given Singularity container image. The utility merges both elements and submits the resulting image, which can be long to generate and deploy but may limit manual operations on the remote location.

3.2 Features

We break down the process into four main steps, namely:

- *Trace*: consists in running applications contained in **apps** and trapping their system calls at runtime, using *Parrot*, to identify and extract the paths of their dependencies. Applications can run in a Singularity container when provided, which delivers further software dependencies and a reproducible environment. Dependencies are then saved in a specific file **namelist.txt**. In this context, *Parrot* is only used to capture system calls and, thus, is not impacted by the issues mentioned in Sect. 2.2. If the step detects an error during the execution

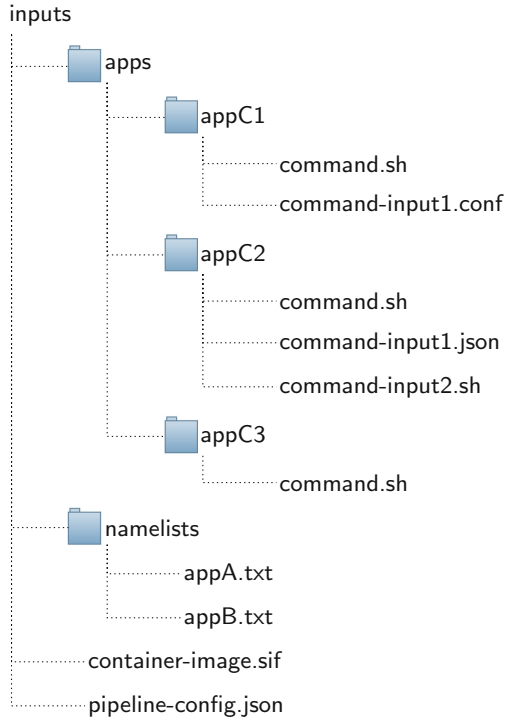


Fig. 2. Schema of the input structure given to the utility.

of an application, then the program is stopped. The step is particularly helpful for users of the utility having no technical knowledge of the applications of interest.

- *Build*: builds a subset of CVMFS based on the paths coming from *Trace* and the **namelists** directory. First, the step merges the namelist files to remove duplicated or non-existent path references, and then separates the paths in different specification files related to repositories. Finally, the step calls *cvmfs_shrinkwrap* to generate the subset of CVMFS. Figures 3 and 4.3 illustrate an example. The utility deduplicates the files, and hard-link data to populate a directory, ready to be exported in various formats as explained in Sect. 2.2 and shown in Fig. 4.3.
- *Test*: consists in testing certain applications - in the given Singularity container environment when provided - using the subset of CVMFS obtained during the *Build* step (see Fig. 4.4). By default, applications from **apps** are used but further tests can also be provided by modifying the utility configuration. All the applications have to complete their execution to go to the next step.


```

in namelist1.txt:
/cvmfs/repoA/path/to/file
/cvmfs/repoB/path/to/another/file
in namelist2.txt:
/cvmfs/repoA/path/to/file
/cvmfs/repoB/path/to/yet/another/file

in repoA.spec:
/path/to/file
in repoB.spec:
/path/to/another/file
/path/to/yet/another/file

```

Fig. 3. Transformation process occurring during the *Trace* step: CVMFS dependencies are extracted from `namelist.txt` and moved to specification files.

- *Deploy*: deploys the subset of CVMFS (Fig. 4.5) embedded or not within the container image depending on the configuration options. If such is the case, then the utility (i) generates a new container definition file that includes the files with the container image, (ii) executes it to produce a new read-only container image. The utility supports ssh deployment via *rsync*, provided the right credentials in the configuration.

3.3 Implementation

The utility is built as a 2-layer system. The first layer, *subcvmfs-builder* [12], is the core of the system and is self-contained. It takes the form of a Python package, which embeds the steps described in Sect. 3.2, and provides a command-line interface to call and execute steps independently from each other. The first layer is, and should remain, simple and generic to be easily managed by developers and used by various communities.

The second layer is the glue code: it consists of a workflow executing - all, or some of - the steps of the first layer. It contains the complexity required to generate and deliver a subset of dependencies according to the needs of its users. Unlike the first layer, the second one can take several forms and each community can tailor it for its software stack.

We propose a first, simple and generic layer-2 implementation calling each step one after the other: *subcvmfs-builder-pipeline* [13]. This layer-2 implementation is executed from a GitLab CI/CD [35], which provides a runner and a docker executor bound to a CVMFS client to execute the code (see Fig. 5) GitLab includes features such as log preservation to help debug the implementation and integrates a pipeline scheduling mechanism to regularly update a subset of dependencies. Even though this layer-2 solution is adapted for basic examples - implying a few commands to trace and test, having a small number of dependencies -, it might require further fine-tuning for more advanced use cases.

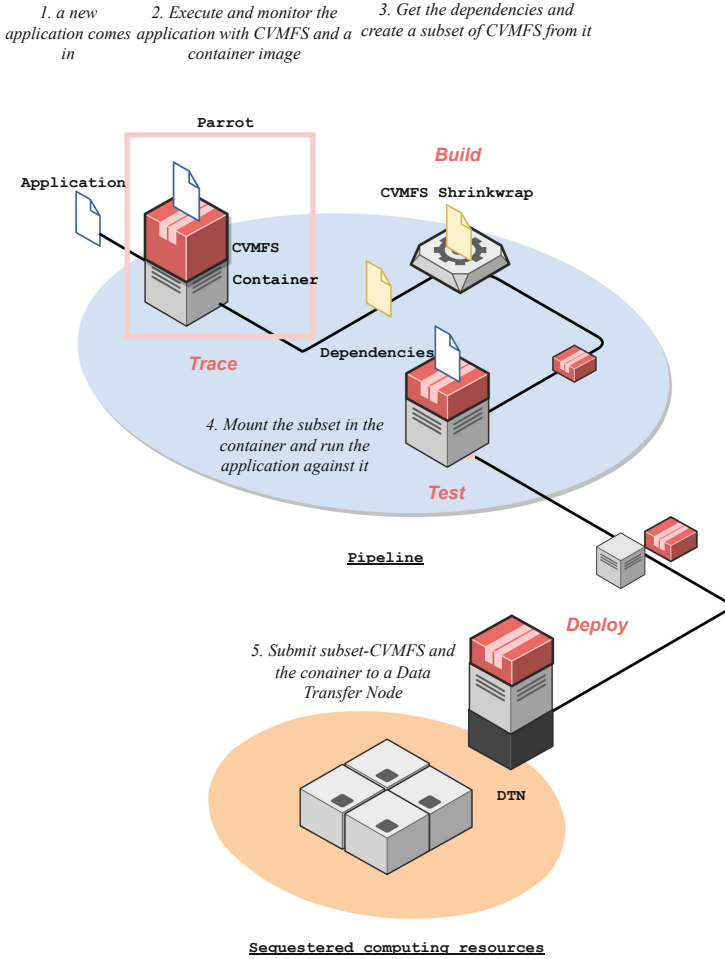


Fig. 4. Schema of the utility workflow: from getting an application to trace to a subset of CVMFS on the Data Transfer Node of a High-Performance Computing cluster.

Indeed, this generic layer-2 implementation is not scalable as it (i) is a single-threaded and single-process program, and (ii) requires manual operations to insert additional inputs in the process. This is not adapted to communities having to trace and test hundreds of various applications to generate large subsets of CVMFS. Two possibilities for such communities: building a new layer-2 implementation - able to automatically fetch applications and trace/test them in parallel - based on *subcvmfs-builder-pipeline* or creating one from scratch.

In the next section, we are going to study how the LHCb experiment [25] leverages *subcvmfs-builder* and *subcvmfs-builder-pipeline* to deliver Gauss [24], a Monte-Carlo simulation program, on the worker nodes of Mare Nostrum [55], a supercomputer with no external connectivity based in Barcelona, Spain.

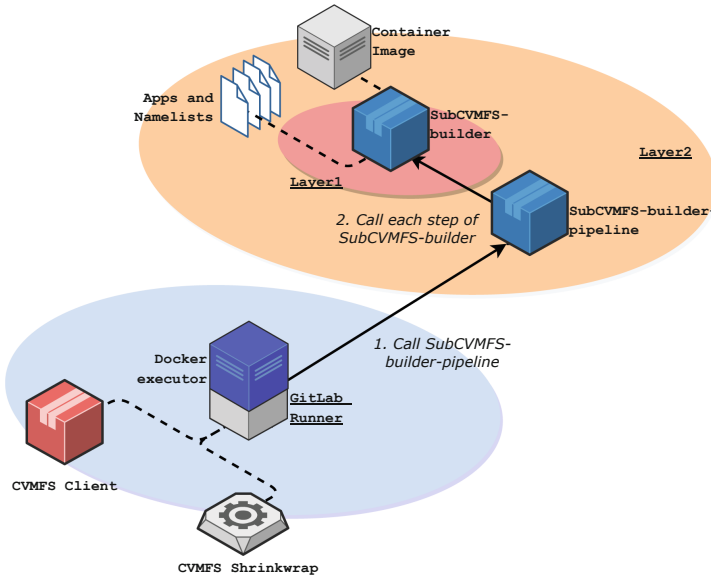


Fig. 5. Schema of a layer-2 implementation within GitLab CI.

4 A Practical Use Case

4.1 Gauss

To better understand experimental conditions and performances, the LHCb collaboration has developed Gauss, a Monte-Carlo simulation application - based on the Gaudi framework [4] - that reproduces events occurring in the LHCb detector. The application consists of two independent phases executed sequentially, namely the generation of the events [6] relying on Pythia [48] by default; the tracking of the particles through the simulated detector depending on Geant4 [1].

In 2021, Gauss represents about 70% of the distributed computing activities of the LHCb collaboration and 150 million events are simulated per day. The application has originally been tailored for WLCG grid sites: Gauss is a compute-intensive single-process (SP), single-threaded (ST) application, only supporting x86 architectures and CERN-CentOS-compatible environments [19]. Gauss and most of its dependencies are delivered via CVMFS.

Gauss takes a certain number of events to process as inputs, as well as a “run number” and an “event number”. The combination of both numbers forms a seed, which ensures repeatability during the generation and simulation phases. It mainly relies on packages such as Python, Boost and gcc to produce histograms and *ntuples* under the form of a ROOT [22] file.

Gauss is modular and highly configurable and constitutes a complex use-case: it can integrate extra packages such as various event generators and decay

tools. Depending on LHCb production needs and the computing environments available, different versions of Gauss and its attached packages can be used. A plethora of option files can also be passed as inputs to the extra packages. Figure 6 describes the inputs, outputs and dependencies of Gauss as well as its interactions with some extra packages and their options.

4.2 Mare Nostrum

To start integrating their workflows on High-Performance computing resources, LHC experiments can benefit from a collaboration with PRACE [44] and GÉANT [18, 32]. This collaboration gives them access to several European supercomputers such as Marconi in Italy and Mare Nostrum in Spain.

Managed by the Barcelona Supercomputing Center (BSC), MareNostrum is the most powerful and emblematic supercomputer in Spain [15]. MareNostrum was built in 2004 (MareNostrum 1), has been updated 3 times since then (Mare Nostrum 2, 3 and 4) and was ranked 63rd in the June 2021 Top500 list [53]. Each node composing the general-purpose block is equipped with two Intel Xeon Platinum 8160 24 cores at 2.1 GHz chips, and at least 2 GB of RAM: this configuration matches with Gauss requirements. Nevertheless, Mare Nostrum is more restrictive than a traditional Grid Site on WLCG: (i) no external connectivity at all; (ii) no service can be installed on the edge node; (iii) no CVMFS, and thus, no Gauss and its dependencies available.

4.3 Running Gauss on Mare Nostrum

Running embarrassingly parallel applications such as Gauss on a supercomputer can be seen as counterproductive. While it is true that the interconnect of the supercomputer partitions has not been designed for millions of small Monte-Carlo runs, it is better to use available, otherwise unused, cycles in agreement with the management of the supercomputer sites. In the meantime, developers are adapting software [39, 47], but it remains a long process, requiring deep and technical software inputs.

To deliver Gauss on Mare Nostrum, LHCb can rely on (i) *subcvmfs-builder* to produce a subset of CVMFS containing the required files; (ii) a CernVM Singularity container to provide a Gauss-compatible environment and to mount the subset of CVMFS as if it was a CVMFS client.

Nevertheless, as we explained in Sect. 4.1, a Gauss execution can involve different packages, extra packages, options, data and versions. Encapsulating its ecosystem requires a good understanding of the application and/or a large amount of storage to encapsulate the right dependencies. Therefore, different options are available:

- Include the whole LHCb CVMFS repository: would not require any specific knowledge about Gauss and would involve all the necessary files to run any Gauss instance. However, this option would imply a tremendous quantity of storage - the full LHCb repository needs 5.2 TB -, long periods to update the subset and many unnecessary files.

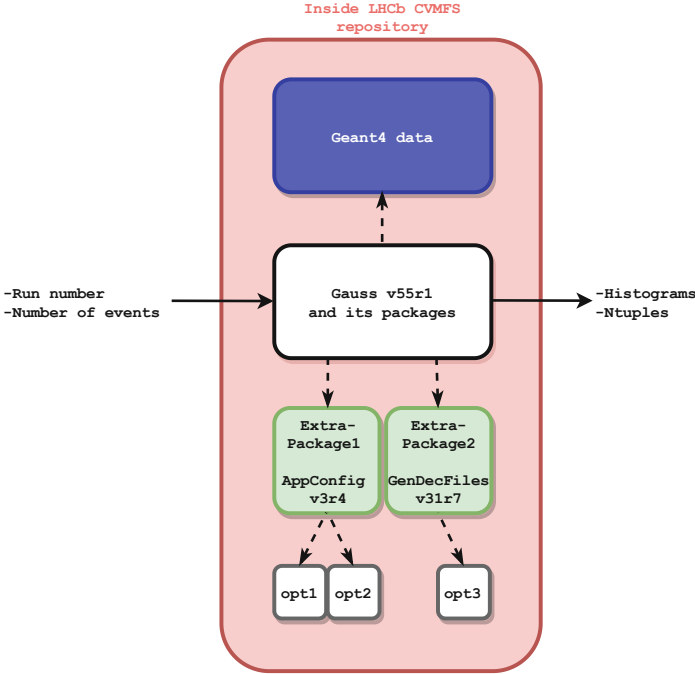


Fig. 6. Example of a Gauss instance, its dependencies and some interactions with extra packages and their options.

- Include the dependencies of various Gauss runs: as the first option, would not need any specific knowledge about Gauss and would include a few gigabytes of data. Nevertheless, such an option would not guarantee the presence of all needed files and would require a tremendous amount of computing resources to trace Gauss workloads continuously.
- Include all the known dependencies of Gauss: would require a deep understanding of Gauss and its dependencies to include all the required files in a subset of CVMFS. While this option would not involve many computing or storage resources, it would include human resources to update the content of the subset of CVMFS according to the releases of Gauss and its extra packages.

As the default storage quota on Mare Nostrum is smaller than the LHCb repository, we decided to reject the first option. LHCb has access to tremendous computing power: it interacts with hundreds of WLCG Sites to run Gauss workloads and could theoretically trace them and extract their requirements. In practice, tracing Gauss workloads in production could slow down the applications and their execution, which is not an option. Similarly, LHCb does not have human resources to update the subset of CVMFS according to the changes done. Thus, we chose to combine the second and the third options to propose a

light and easy to update and maintain solution. The process consists in getting insights into the structure of the Gauss dependencies by running and tracing a small set of Gauss workloads and analyzing the system calls before including the structure in *subcvmfs-builder-pipeline*.

After analyzing 500 commands calling Gauss from the LHCb production environment and tracing 3 Gauss applications using *subcvmfs-builder* [14], we noticed that:

- 97% of the workloads studied were running the same Gauss versions (v49r20) with the same extra packages and versions. The versions of Gauss and its extra packages seem related to the underlying architecture.
- 846 Mb of files were needed to run 3 Gauss (v49r20) workloads. About 95% of the size is related to the Gauss version and the underlying architecture, and is common to the Gauss workloads traced, while the 5% left is bound to the options and Geant4 data used that are specific to a given Gauss workload.
- Integrating all the options and Geant4 data related to Gauss v49r20 would correspond to 1.8 Gb of files.

Based on these assumptions, we created a `namelist` file containing (i) the files shared by the 3 Gauss applications that we traced and (ii) all the options and Geant4 data in order to generate a subset of CVMFS able to run any Gauss workload targeting the v49r20 version. We used *subcvmfs-builder-pipeline* to build the subset of CVMFS, to successfully test it with 5 Gauss workloads - different from the ones we used previously - and to deploy it to Mare Nostrum. We fine-tuned the utility to disable the *trace* step and to deploy the subset separately from the container. Indeed, CernVM - the container that we use to provide a reproducible environment to the workload - does not need regular updates and merging it with the subset of CVMFS is a time-consuming operation.

This resulted in a CernVM singularity container occupying 6.4 Gb on the General Parallel File System (GPFS) of Mare Nostrum combined with a subset of CVMFS covering 6 Gb: dependencies occupies 3.2 Gb of space while 2.8 Gb are required for the *cvmfs_shrinkwrap* metadata. Thus, 12.4 Gb of space on the GPFS of Mare Nostrum is currently sufficient to run 97% of the Gauss workloads analyzed: 0.24% of the LHCb repository.

Even though this approach provides a light, easy and fast-to-update solution, LHCb developers need to keep it up to date to integrate new versions or structure changes. One way to proceed would consist in automating and repeating the analysis work regularly. One could also integrate the *trace* command of *subcvmfs-builder* within the LHCb production test phase, which consists in running a few events of upcoming Gauss workloads on a given Grid Site. LHCb developers could trace some of them during the process and store the traces in a database. An LHCb-specific *subcvmfs-pipeline-builder* could then periodically fetch the content of the database to build, test and deploy a new subset of dependencies to Mare Nostrum.

5 Conclusion

This paper presents a dependency delivery system based on CVMFS to provide complex software stacks on sequestered computing resources such as worker nodes of supercomputers not having external connectivity.

After introducing CVMFS (Sect. 2.1), a critical tool - especially for LHC communities - to supply workloads with complex dependencies on Grid Sites, we have described the context of this study (Sect. 2.2): several virtual organizations are exporting their workflow from WLCG to supercomputers, which have more restrictive policies than grid sites and generally do not allow to mount CVMFS on the worker nodes.

We have highlighted several solutions aiming to overcome the issue such as collaborating with the system administrators and using tools such as *Parrot* and *cvmfsexec*. Nevertheless, these approaches do not work when worker nodes have no external connectivity. Then, we have emphasized different ways to export parts of CVMFS to supercomputers with no external connectivity: *uncvmfs* and *cvmfs_shrinkwrap*. These solutions require several manual steps and therefore we have proposed a utility to assist communities in this process.

We have explained the different steps of the utility in detail (Sect. 3.2). It traces - captures the system calls of - applications of interest, builds a subset with the required files, tests the subset and deploys it to a remote computing resource. We also described the structure of the solution (Sect. 3.3), which is composed of two layers: a first one, generic with simple components, and a second one more complex, adapted to communities needs that can be fine-tuned.

Finally, we have provided a use case based on Gauss, a Monte-Carlo simulation application reproducing events occurring in the LHCb detector (Sect. 4.1). Gauss is highly configurable and can be coupled with different packages, extra packages, options, data and versions. It represents a complex bundle of dependencies, which makes it ideal to test our utility. We have proposed a method to encapsulate Gauss and its dependencies in a subset, which represents 12.4 Gb of space on the GPFS of the Mare Nostrum supercomputer (Sect. 4.3). The solution produced represents 0.24% of the full LHCb repository and, thus, is easier to update. We have successfully tested the solution with different Gauss workloads. Future work could focus on encapsulating further applications from different domains using this utility, and analyzing its performances to deploy subsets on various supercomputers.

References

1. Agostinelli, S., et al.: GEANT 4-a simulation toolkit. Nuclear Instrum. Methods Phys. Res. Sect. A Acceler. Spectrom. Detect. Assoc. Equip. **506**(3), 250–303 (2003). [https://doi.org/10.1016/S0168-9002\(03\)01368-8](https://doi.org/10.1016/S0168-9002(03)01368-8), <http://www.sciencedirect.com/science/article/pii/S0168900203013688>
2. Apollinari, G., Béjar Alonso, I., Brüning, O., Lamont, M., Rossi, L.: High-Luminosity Large Hadron Collider (HL-LHC): Preliminary Design Report. CERN Yellow Reports: Monographs, CERN, Geneva (2015). <https://doi.org/10.5170/CERN-2015-005>, <http://cds.cern.ch/record/2116337>

3. Arsuaga-Ríos, M., Heikkilä, S.S., Duellmann, D., Meusel, R., Blomer, J., Couturier, B.: Using s3 cloud storage with ROOT and CvmFS. *J. Phys. Conf. Ser.* **664**(2), 022001 (2015). <https://doi.org/10.1088/1742-6596/664/2/022001>
4. Barrand, G., et al.: Gaudi—a software architecture and framework for building hep data processing applications. *Comput. Phys. Commun.* **140**(1), 45–55 (2001). [https://doi.org/10.1016/S0010-4655\(01\)00254-5](https://doi.org/10.1016/S0010-4655(01)00254-5), <https://www.sciencedirect.com/science/article/pii/S0010465501002545>, cHEP2000
5. Barreiro, F., et al.: The future of distributed computing systems in atlas: boldly venturing beyond grids. *EPJ Web Conf.* **214**, 03047 (2019). <https://doi.org/10.1051/epjconf/201921403047>
6. Belyaev, I., et al.: Handling of the generation of primary events in gauss, the LHCb simulation framework. *J. Phys. Conf. Ser.* **331**(3), 032047 (2011). <https://doi.org/10.1088/1742-6596/331/3/032047>
7. Douglas, B.: Building and using containers at HPC centres for the atlas experiment. *EPJ Web Conf.* **214**, 07005 (2019). <https://doi.org/10.1051/epjconf/201921407005>
8. Blomer, J.: CernVM-FS overview and roadmap (2021). https://easybuild.io/eum/002_eum21_cvmfs.pdf. Accessed 26 May 2021
9. Blomer, J., Ganis, G., Hardi, N., Popescu, R.: Delivering LHC software to HPC compute elements with CernVM-FS. In: Kunkel, J.M., Yokota, R., Taufer, M., Shalf, J. (eds.) *ISC High Performance 2017*. LNCS, vol. 10524, pp. 724–730. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-67630-2_52
10. Jakob, B., Dave, D., Gerardo, G., Simone, M., Jan, P.: A fully unprivileged CernVM-FS. *EPJ Web Conf.* **245**, 07012 (2020). <https://doi.org/10.1051/epjconf/202024507012>
11. Jakob, B., Gerardo, G., Simone, M., Radu, P.: Towards a serverless CernVM-FS. *EPJ Web Conf.* **214**, 09007 (2019). <https://doi.org/10.1051/epjconf/201921409007>
12. Boyer, A.F.: SubCVMFS-builder (2022). <https://doi.org/10.5281/zenodo.6335367>
13. Boyer, A.F.: SubCVMFS-builder-pipeline (2022). <https://doi.org/10.5281/zenodo.6335512>
14. Boyer, A.F.: SubCVMFS: gauss analysis (2022). <https://doi.org/10.5281/zenodo.6337297>
15. BSC: Marenostrum (2020). <https://www.bsc.es/marenostrum/>. Accessed 04 Oct 2021
16. Buncic, P., et al.: CernVM – a virtual software appliance for LHC applications. *J. Phys. Conf. Ser.* **219**(4), 042003 (2010). <https://doi.org/10.1088/1742-6596/219/4/042003>
17. Chris, B., Marco, C., Ben, C.: Software packaging and distribution for LHCb using nix. *EPJ Web Conf.* **214**, 05005 (2019). <https://doi.org/10.1051/epjconf/201921405005>
18. CERN: Cern, skao, gÉant and prace to collaborate on high-performance computing (2020). <https://home.cern/news/news/computing/cern-skao-geant-and-prace-collaborate-high-performance-computing>. Accessed 04 Oct 2021
19. CERN: Linux@cern (2020). <https://linux.web.cern.ch/>. Accessed 09 Feb 2021
20. CERN: CernVM-FS (2021). <https://cernvm.cern.ch/>. Accessed 19 May 2021
21. CERN: The large hadron collider (2021). <https://home.cern/science/accelerators/large-hadron-collider>. Accessed 27 May 2021
22. CERN: Root: analyzing petabytes of data, scientifically (2021). <https://root.cern.ch/>. Accessed 30 Sep 2021
23. CERN: Worldwide LHC computing grid (2021). <https://wlcg.web.cern.ch/>. Accessed 27 May 2021

24. Clemencic, M., et al.: The LHCb simulation application, gauss: design, evolution and experience. *J. Phys. Conf. Ser.* **331**(3), 032023 (2011). <https://doi.org/10.1088/1742-6596/331/3/032023>
25. Collaboration, T.L.: The LHCb detector at the LHC. *J. Instrum.* **3**(08), S08005–S08005 (2008). <https://doi.org/10.1088/1748-0221/3/08/s08005>
26. CVMFS: cvmfsexec (2021). <https://github.com/cvmfs/cvmfsexec>. Accessed 28 May 2021
27. Dykstra, D., Blomer, J.: Security in the CernVM file system and the frontier distributed database caching system. *J. Phys. Conf. Ser.* **513**, 042015 (2014). <https://doi.org/10.1088/1742-6596/513/4/042015>
28. EasyBuild: Easybuild: building software with ease (2021). <https://easybuild.io/>. Accessed 11 Dec 2021
29. Fasel, M.: Using nersc high-performance computing (HPC) systems for high-energy nuclear physics applications with alice. *J. Phys. Conf. Ser.* **762**, 012031 (2016). <https://doi.org/10.1088/1742-6596/762/1/012031>
30. Blomer, J., Ganis, G., Hardi, N., Popescu, R.: Delivering LHC software to HPC compute elements with CernVM-FS. In: Kunkel, J.M., Yokota, R., Taufer, M., Shalf, J. (eds.) *ISC High Performance 2017*. LNCS, vol. 10524, pp. 724–730. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-67630-2_52
31. Filipčić, A., Haug, S., Hostettler, M., Walker, R., Weber, M.: Atlas computing on CSCS HPC. *J. Phys. Conf. Ser.* **664**(9), 092011 (2015). <https://doi.org/10.1088/1742-6596/664/9/092011>
32. GÉANT: GÉant (2021). <https://www.geant.org/>. Accessed: 04 Oct 2021
33. Gentoo: Gentoo linux (2021). <https://www.gentoo.org/>. Accessed: 11 Dec 2021
34. Gerhardt, L., et al.: Shifter: containers for HPC. *J. Phys. Conf. Ser.* **898**, 082021 (2017). <https://doi.org/10.1088/1742-6596/898/8/082021>
35. GitLab: Gitlab ci/cd (2021). <https://docs.gitlab.com/ee/ci/>. Accessed 23 Sep 2021
36. Harutyunyan, A., et al.: CernVM co-pilot: an extensible framework for building scalable computing infrastructures on the cloud. *J. Phys. Conf. Ser.* **396**(3), 032054 (2012). <https://doi.org/10.1088/1742-6596/396/3/032054>
37. ic hep: uncvms (2018). <https://github.com/ic-hep/uncvms>. Accessed 30 May 2021
38. Hufnagel, D.: CMS use of allocation based HPC resources. *J. Phys. Conf. Ser.* **898**, 092050 (2017). <https://doi.org/10.1088/1742-6596/898/9/092050>
39. Mazurek, M., Corti, G., Muller, D.: New simulation software technologies at the LHCb Experiment at CERN (2021)
40. NixOS: Nixos (2021). <https://nixos.org/>. Accessed 11 Dec 2021
41. O’Brien, B., Walker, R., Washbrook, A.: Leveraging HPC resources for high energy physics. *J. Phys. Conf. Ser.* **513**(3), 032104 (2014). <https://doi.org/10.1088/1742-6596/513/3/032104>
42. Oleynik, D., et al.: High-throughput computing on high-performance platforms: a case study (2017)
43. Radu, P., Jakob, B., Gerardo, G.: Towards a responsive CernVM-FS architecture. *EPJ Web Conf.* **214**, 03036 (2019). <https://doi.org/10.1051/epjconf/201921403036>
44. PRACE: Partnership for advanced computing in Europe (2021). <https://prace-ri.eu/>. Accessed 04 Oct 2021
45. Sciacca, F.G.: Enabling atlas big data processing on piz daint at CSCS. *EPJ Web Conf.* **245**, 09005 (2020). <https://doi.org/10.1051/epjconf/202024509005>

46. Segal, B., et al.: Lhc cloud computing with CernVM. In: 13th International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT2010) vol. 093, issue 4, p. 042003 (2011). <https://doi.org/10.22323/1.093.0004>
47. Siddi, B.G., Müller, D.: Gaussino - a gaudi-based core simulation framework. In: 2019 IEEE Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC), pp. 1–4. IEEE, Manchester, United Kingdom (2019). <https://doi.org/10.1109/NSS/MIC42101.2019.9060074>
48. Sjöstrand, T., et al.: High-energy-physics event generation with pythia 6.1. *Comput. Phys. Commun.* **135**(2), 238–259 (2001). [https://doi.org/10.1016/s0010-4655\(00\)00236-8](https://doi.org/10.1016/s0010-4655(00)00236-8)
49. Spack: Spack (2021). <https://spack.readthedocs.io/en/latest/>. Accessed 11 Dec 2021
50. Stagni, F., McNab, A., Luzzi, C., Krzemien, W., Consortium, D.: Dirac universal pilots. *J. Phys: Conf. Ser.* **898**(9), 092024 (2017). <https://doi.org/10.1088/1742-6596/898/9/092024>
51. Stagni, F., Valassi, A., Romanovskiy, V.: Integrating LHCb workflows on HPC resources: status and strategies. *EPJ Web Conf.* **245**, 09002 (2020). <https://doi.org/10.1051/epjconf/202024509002>
52. Teuber, S.: Efficient unpacking of required software from CERNVM-FS (2019). <https://doi.org/10.5281/zenodo.2574462>
53. Top500: Top500 (2021). <https://www.top500.org/>. Accessed 04 Oct 2021
54. Benjamin, T., Brian, B., Michael, H., Kevin, L., Douglas, T.: Harnessing HPC resources for CMS jobs using a virtual private network. *EPJ Web Conf.* **251**, 02032 (2021). <https://doi.org/10.1051/epjconf/202125102032>
55. Vicente, D., Bartolome, J.: BSC-CNS research and supercomputing resources. In: Resch, M., Roller, S., Benkert, K., Galle, M., Bez, W., Kobayashi, H. (eds.) *High Performance Computing on Vector Systems 2009*, pp. 23–30. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-03913-3_2
56. l Valentin, V., et al. : Building hep software with spack: experiences from pilot builds for key4hep and outlook for LCG releases. *EPJ Web Conf.* **251**, 03056 (2021). <https://doi.org/10.1051/epjconf/202125103056>
57. Benda, X., Guilherme, A., Fabian, G., Michael, H.: Gentoo prefix as a physics software manager. *EPJ Web Conf.* **245**, 05036 (2020). <https://doi.org/10.1051/epjconf/202024505036>