# CernVM 5: a versatile container-based platform to run HEP applications

**Jakob Eberhardt**
Karlsruhe University of Applied Sciences, Moltkestraße 30, 76133 Karlsruhe, Germany
E-mail: `jakob.eberhardt@mail.de`

**Jakob Blomer**
CERN, 1211 Geneva 23, Switzerland
E-mail: `jblomer@cern.ch`

**Abstract.** Since its inception, the minimal Linux image CernVM provides a portable and reproducible runtime environment for developing and running scientific software. Its key ingredient is the tight coupling with the CernVM-FS client to provide access to the base platform (operating system and tools) as well as the experiment application software. Up to now, CernVM images are designed to use full virtualization. The goal of CernVM 5 is to deliver all the benefits of the CernVM appliance and to be equally practical as a container and as a full VM. To this end, the CernVM 5 container image consists of a "Just Enough Operating System (JeOS)", with its contents defined by the HEP_OSlibs meta-package commonly used as a base platform in HEP. CernVM 5 further aims at smooth integration of the CernVM-FS client in various container environments (such as Docker, Kubernetes, Podman, Apptainer). Lastly, CernVM 5 uses special build tools and post-build processing to ensure that experiment software stacks using their custom compilers and build chains can coexist with standard system application stacks. As a result, CernVM 5 aims at providing a single, minimal container image that can be used as a virtual appliance for mounting the CernVM-FS client and for running and developing HEP application software.

## 1. CernVM Project

The CernVM project [1] provides a scalable and easy-to-use platform for the distribution and management of scientific software stacks for HEP applications. The CernVM virtual appliance is designed to address the challenges of managing and distributing large and complex software stacks in scientific research environments. It allows scientists to easily access and use externally managed analysis applications and frameworks. Aiming to be a "Just Enough Operating System (JeOS)", CernVM images serve as a portable run time environment for the CernVM-File System (CernVM-FS) [2], a distributed file system that allows users to access scientific software without the need for local software installations. Up until version 4, CernVM images [3] were mostly designed for full virtualization. Consisting of a minimal Linux kernel and the µCernVM bootloader, CernVM images are capable of mounting the CernVM-File System upon boot to load system packages as well as scientific software on demand.

## 2. Problem Statement

Looking at previous versions, CernVM images are limited in terms of usability as a container. By design, the µCernVM bootloader approach requires full control of the kernel. However, users keep shifting their infrastructure towards unprivileged containerized deployments. In addition, the currently available CernVM container images lack standard features such as native derivability using common build tools as well as rootless mounting of CernVM-FS repositories.

As new versions of scientific software stacks are frequently released and can quickly grow to gigabytes of data, it is still not feasible to build individual images for each release and compiler / OS combination. Therefore, the new version of CernVM has to be designed as a minimal but extendable appliance, capable of running both system applications and scientific software installed on CernVM-FS in a single environment. In particular, this can lead to problems when setup scripts provided by the maintainers of scientific stacks use environment variables to overwrite local default values for resolving dependencies. In this case, local system applications and those installed on CernVM-FS break once linked with a mismatching version of a shared library required by the currently used scientific stack. To prevent this, methods have to be implemented to make system applications bypass the commonly used environment variables such as `LD_LIBRARY_PATH` to achieve stable usability of the image. Since the CernVM also targets users in interactive sessions, a graphical interface has to be integrated while keeping the image size as small as possible. To further enhance the user experience when working with software loaded from CernVM-FS, methods have to be investigated to enable special-purpose images with pre-loaded CernVM-FS caches. Lastly, some use cases, e. g. offline computing in data acquisition systems, still require full virtual machine images. To this end, in an additional step, the container images should be extended to full VM images to run on common hypervisors with their own kernel.

## 3. CernVM 5

The goal of the CernVM 5 project is to bring all the benefits and key features of previous CernVM versions to native container virtualization using standard container tools and runtimes. The goals as well as the conception and implementation are further described in this section.

### *3.1. Goals*

*Minimal* The CernVM 5 base layer image should be built around the applications it serves as a run time. This includes the CernVM-FS client and the HEP_OSlibs [4], a meta-package of common HEP libraries. To reduce the image size, additional user applications such as editors and graphical interfaces should be outsourced to a dedicated CernVM-FS repository.

*Versatility* The container image should run in all common container runtimes such as Docker, Podman, Apptainer and Kubernetes while supporting rootless mounting of CernVM-FS repositories. Extended to a full virtual machine image, CernVM 5 should run on standard hypervisors such as VirtualBox or KVM and on cloud infrastructure such as OpenStack.

*Maintainability and Derivability* Unlike the previous versions, CernVM 5 images should be built from standard EL9-compatible packages. This not only removes the effort of building and maintaining custom packages but also allows user to natively derive their own images from the CernVM base layer in, e. g. a Dockerfile using stock repositories. Any further customizations to the used packages should be implemented as a post-build step.

*Turnkey* CernVM 5 should reduce the effort of getting to a basic deployment with CernVM-FS access to a minimum by being distributed with a customizable standard configuration and in standard-compliant image formats.

## *3.2. Design*

The CernVM 5 system consists of the actual base layer image, the system application, and the externally managed scientific software stacks.

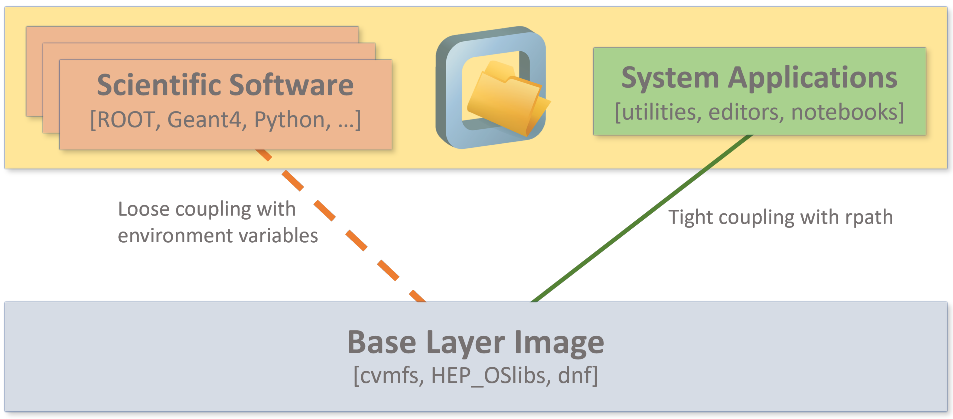
Scientific Software
[ROOT, Geant4, Python, ...]
System Applications
[utilities, editors, notebooks]
Loose coupling with environment variables
Tight coupling with rpath
Base Layer Image
[cvmfs, HEP_OSlibs, dnf]

**Figure 1.** The CernVM 5 system consists of the actual base layer image which is complemented on-demand by the separate system application area on CernVM-FS. Unlike the scientific stacks which use environment variables to set up the local environment, the system applications are closely coupled to the image by using rpath.

*Base Layer Image* The CernVM 5 base layer image represents a minimal platform to develop and run HEP analysis software. It consists of the CernVM-FS client and the most common base libraries included in the HEP_OSlibs. Further, the base layer is extendable using the standard package manager `dnf` and standard container tools.

*System Application Area* This CernVM-FS repository hosts interactive and user-focused applications such as editors, notebooks and command-line utilities. Its binaries are closely coupled to the base layer image using the hard-coded binary header `rpath`.

*Scientific Software* These externally managed stacks bundle applications and libraries into a compatible release distributed via CernVM-FS. The maintainers of these stacks provide setup scripts that use environment variables such as `LD_LIBRARY_PATH` or `PYTHONPATH` to allow users to use different stacks in their local environment.

## *3.3. Container Image*

The CernVM 5 container image build process is implemented in a muti-staged Dockerfile. In the first step, an intermediate build stage with all necessary build dependencies is started to construct the root file system of the final image. By creating the root file system from scratch rather than deriving from a pre-existing image, the final container image only includes a controlled set of actually needed packages defined in the `cernvm-system-default` meta-package, which bundles a minimal set of packages and configurations required to smoothly integrate the CernVM-FS client and to run scientific software. To prevent the breaking of dynamically linked system applications as described in the problem statement (see Section 2), an additional build step was implemented to add a `DT_RPATH` to all installed ELF executables using patchELF. This binary header is used to resolve the location of shared libraries which are by default located under the `lib` and `lib64` directories. Most linkers such as `ld` look up `DT_RPATH` before other environment variables such as `LD_LIBRARY_PATH`. This allows CernVM 5 system applications to run in local environments modified by scientific software stacks while still using standard packages. This method further enables users to re-run this post-build processing in their own derived images, e. g. by embedding it into their Dockerfile.

### *3.4. System Application Area on CernVM-FS*
Additional system applications are available on the dedicated CernVM-FS repository `cernvm-five.cern.ch`. It includes packages and applications for interactive use cases like editors, development tools and other expected utilities. Similar to how the container image is created, a new release is built by installing the `cernvm-system-cvmfs` into an empty scratch directory. The same script used during the container build automatically sets up a `DT_RPATH` for each ELF executable according to its future location on a CernVM-FS publisher node.

### *3.5. Web-Based Jupyter Notebooks*
Jupyter Notebooks were evaluated to be adequate as a graphical interface for CernVM 5 once for their popularity among scientists but also for technical reasons. Firstly, CernVM 5 containers aim to run rootless while still providing full access to all development features. By choosing a web-based application such as Jupyter Notebooks, the need for elevated privileges or other host prerequisites such as an open X server can be avoided. Additionally, many scientific stacks, e. g. the LCG stack [5], already provide their own custom Notebooks that can simply be used by forwarding the respective port 8888 of the container to the host.

### *3.6. Images with pre-loaded CernVM-File System Cache*
In particular when looking at interactive user sessions, but also when computing resources are only available for a short amount of time, the start latency of the CernVM-FS client installed in CernVM 5 images had to be addressed. As a first step towards this problem, methods to build and distribute CernVM 5 images with scenario-specific CernVM-FS caches were implemented. The FCC Starterkit tutorial [6] serves as a proof-of-concept for building such an image. Since the underlying key4hep [7] software stack of this tutorial is frequently updated, the cache delivered with the image has to be kept in sync with the latest version in an automated manner. This is implemented in an intermediate container build stage which mounts the latest version of the stack and runs the test suite of the tutorial, thereby populating the local CernVM-FS cache. After finishing the test run which performs the tutorial, the stopped intermediate container is exported and compressed to a standalone image. The resulting speedups can be seen in the following discussion in section 4.

### *3.7. Virtual Machine Extension*
The root file system of the container image is reused to build a full virtual machine image upon it. By deriving from the base layer or any other CernVM 5 image and installing the `cernvm-kernel-default` RPM, all configuration files and packages needed for full virtualization are added, such as a kernel and a bootloader.
Using a custom script, the extended root file system is copied to a bootable disk which can be run on various hypervisors such as VirtualBox or KVM. The CernVM 5 cloud templates provide seamless integration of `cloud-init` contextualization when creating or customizing resources on cloud platforms such as OpenStack or EC2.

## 4. Discussion

In this section, the results of the CernVM 5 implementation are discussed. As can be seen in table 1, a smaller image size was achieved by the extra steps taken during the build process which also includes the processing of system applications to use `rpath`. The CernVM 5 container image brings CernVM-FS to various tested container runtimes, some of which are listed in table 2. The increase in performance by pre-loading CernVM-FS caches for defined workflows can be seen in table 3.

**Table 1.** This table compares the result of the CernVM 5 custom build chain to a standard derived image created by installing the package on top of an EL9 base image. As the CernVM 5 base layer image is specifically built around the dependencies of HEP applications, it includes no extra packages commonly installed in pre-build base images. This results in a final image about 200MB smaller than the one built with a naive approach. Additionally, the multi-staged build enables a single-layer base layer which reduces its complexity and the time needed by the container runtime to construct it upon start. Lastly, the CernVM 5 system executables are post-build processed to use `DT_RPATH` to make them usable in the presence of more than one available source of shared libraries.

| | CernVM 5 Base Layer | Standard Derived Image |
|---|---|---|
| Volume (uncompressed, MB) | 805 | 1030 |
| Volume (compressed, MB) | 284 | 382 |
| Installed Packages | 457 | 502 |
| Image Layers | one | multiple |
| Standard Derivability | ✓ | ✓ |
| Use of rpath | ✓ | ✗ |

**Table 2.** As can be seen in this table, CernVM 5 images integrate the CernVM-FS client in various container engines using the FUSE interface without elevated privileges. However, if CernVM-FS is already available on the host system, it can still be bind-mounted to CernVM 5 containers, e. g., to benefit from shared caches. As a novelty, a CernVM 5 container can be used to forward the CernVM-FS namespace to a Linux host. Previously called Singularity, the Apptainer engine allows for pre-mounted file systems, e. g. the CernVM 5 system application area repository.

| CernVM-File System | Docker | Podman | Apptainer | Kubernetes |
|---|---|---|---|---|
| Mounted inside | ✓ | ✓ | ✓ | ✓ |
| Mounted from host | ✓ | ✓ | ✓ | ✓ |
| Mounted to Linux host | ✓ | ✓ | ✓ | ✓ |
| Pre-mounted | ✗ | ✗ | ✓ | ✗ |

**Table 3.** This table compares the execution time of commands needed during the FCC Starterkit tutorial. The commands were run in the standard CernVM 5 base image and a special CernVM 5 image distributed with an automatically pre-loaded CernVM-FS cache. It should be noted that this comparison was conducted from containers running in the CERN network which already considerably reduces the start latency of the CernVM-FS client due to the high bandwidth and local CernVM-FS proxy caches.

| | Base Image | FCC Scenario Image |
|---|---:|---:|
| KKMCee -h | 1.609s | 0.357s |
| BHLUMI -h | 1.569s | 0.055s |
| whizard Z_mumu.sin | 50.233s | 18.971s |
| babayaga -h | 1.635s | 0.487s |
| Import ROOT | 6.843s | 1.544s |
| fccanalysis run | 11.230s | 9.319s |

## 5. Conclusion

The CernVM 5 virtual appliance is a minimal yet complete platform to develop and run HEP analysis software. As a novelty, it is completely container-based, bringing access to CernVM-FS to various container runtimes such as Docker, Podman or Apptainer. Build in a custom multi-staged process, the CernVM 5 image can serve as a base layer using common container build tools. In an additional post-build step, the installed system executables are modified to use `DT_RPATH`. This not only ensures stable usability in the presence of multiple sources of shared libraries but also enables the use of standard packages from stock repositories. CernVM 5 integrates Jupyter Notebooks loaded from CernVM-FS as an easy-to-use and lightweight graphical interface for developing analysis code. To reduce the start-up latency of the CernVM-FS client when working on special workflows like tutorials, methods to build and distribute CernVM 5 images with pre-loaded caches in an automated manner were implemented as a proof-of-concept. A custom program can be used to extend the CernVM 5 container image or any given derivation of it to a full virtual machine image, compatible with all common hypervisors such as VirtualBox or KVM. All sources and links to build products can be found on the CernVM 5 project page [8].

## References

[1] Buncic P, Blomer J, Mato P, Sanchez C A, Franco L and Klemer S 2008 Cernvm - a virtual appliance for lhc applications *Proceedings of the XII International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT08)*

[2] Blomer J, Buncic P, Meusel R, Ganis G, Sfiligoi I and Thain D 2015 *Computing in Science and Engineering* **17** 61–71

[3] Blomer J, Berzano D, Buncic P, Charalampidis I, Ganis G, Lestaris G, Meusel R and Nicolaou V 2014 *Journal of Physics: Conference Series* **513**

[4] The HEP_OSlibs Meta-package `https://gitlab.cern.ch/linuxsupport/rpms/HEP_OSlibs` [Online; accessed 27-December-2022]

[5] Cervantes Villanueva, Javier, Ganis, Gerardo, Konstantinov, Dmitri, Latyshev, Grigorii, Mato Vila, Pere, Mendez Lorenzo, Patricia, Pacholek, Rafal and Razumov, Ivan 2019 *EPJ Web Conf.* **214** 05020 URL `https://doi.org/10.1051/epjconf/201921405020`

[6] The FCC Starterkit Tutorial `https://hep-fcc.github.io/fcc-tutorials/index.html` [Online; accessed 4-December-2022]

[7] Ganis G, Helsens C and Völkl V 2021 *Eur. Phys. J. Plus* **137** 149. 8 p (*Preprint* `2111.09874`) URL `https://cds.cern.ch/record/2790916`

[8] The CernVM 5 Project on GitHub `https://github.com/cernvm/cernvm-five` [Online; accessed 27-December-2022]