



Karlsruhe University of Applied Sciences

Department of Computer Science and Business Information Systems  
Business Information Systems

## BACHELOR'S THESIS

Development of a container-based  
virtual appliance for developing and running  
HEP data analyses at CERN

Author	Mr. Jakob Eberhardt
Matriculation number	64983
Workplace	CERN, Genf
First Advisor	Prof. Dr. Andreas Schmidt
Second Advisor	Prof. Dr. Ingo Stengel
Closing Date	01 August, 2022

01 August, 2022

Chairman of the Examination Board



## Abstract

The High Energy Physics (HEP) research community studies fundamental physics by analyzing petabytes of data produced in particle colliders and taken by particle detectors every year. To process this amount of data, the analysis code developed by scientists is executed on a global network of computing resources. Scientific software maintainers (so-called “software librarians”) bundle a compatible set of needed scientific software into a versioned release. These large and frequently changing stacks are commonly distributed to scientists and worker nodes using the CernVM-File System (CernVM-FS). This distributed file system enables on-demand loading of binaries installed on centrally managed software repositories. 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 scientific software. To this day, CernVM images are designed to use full virtualization. The goal of this project is to develop a new version of CernVM, CernVM 5, that should deliver all the benefits of the CernVM appliance and should be equally practical as a Linux container and as a full virtual machine. 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 which is commonly used as a base platform in HEP. CernVM 5 further aims to smoothly integrate the CernVM-FS client in various container runtimes. Lastly, CernVM 5 uses special build tools and post-build processing to ensure that scientific software stacks using their custom compilers and build chains can coexist with standard system applications. As a result, CernVM 5 aims to provide 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.

Many thanks to my supervisor Jakob Blomer for his involvement in my project and this thesis. I would also like to thank my professor Andreas Schmidt, my co-supervisor Radu Popescu, and my other colleagues Andrea Valenzuela Ramírez, Graeme Stewart, and Valentin Volkl, who each supported me during the project or while writing this thesis.

### Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die hier vorgelegte Bachelor-Thesis selbstständig und ausschließlich unter Verwendung der angegebenen Literatur und sonstigen Hilfsmittel verfasst habe. Die Arbeit wurde in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde zur Erlangung eines akademischen Grades vorgelegt.

Genf, 29. Juli 2022  
Ort, Datum

Eberhardt  
Unterschrift

# Contents

<b>1</b>	<b>Introduction</b>	<b>7</b>
1.1	Project . . . . .	8
<b>2</b>	<b>Basics</b>	<b>9</b>
2.1	Virtualization . . . . .	9
2.1.1	Virtual Machines . . . . .	9
2.1.2	Containers . . . . .	10
2.2	CernVM-File System . . . . .	11
2.3	Virtual Software Appliances . . . . .	13
2.3.1	CernVM Virtual Appliance . . . . .	13
2.4	Dynamic Linking . . . . .	13
<b>3</b>	<b>Conception</b>	<b>15</b>
3.1	Use Cases . . . . .	15
3.2	Requirements . . . . .	15
3.2.1	Minimal . . . . .	15
3.2.2	Derivability . . . . .	15
3.2.3	Maintainability . . . . .	15
3.2.4	Reproducibility . . . . .	15
3.2.5	Turnkey . . . . .	16
3.2.6	Separated System Application Area on CernVM-FS . . . . .	16
3.2.7	Versatility . . . . .	16
3.2.8	Automated Build and Test Infrastructure . . . . .	16
3.3	Design . . . . .	17
3.3.1	Base Layer . . . . .	17
3.3.2	System Applications Area on CernVM-FS . . . . .	17
3.3.3	Scientific Software . . . . .	17
3.4	Challenges . . . . .	18
3.4.1	Dependency Linking . . . . .	18
3.4.2	Graphical Interface and User Integration . . . . .	18
3.4.3	CernVM-FS Cache Management . . . . .	18
<b>4</b>	<b>Implementation of the Container Image</b>	<b>19</b>
4.1	Source Code Repository Structure . . . . .	19
4.2	Building RPMs . . . . .	19
4.2.1	System RPM . . . . .	19
4.2.2	Configuration RPM . . . . .	19
4.3	Multi-Staged Build . . . . .	20
4.4	Building the CernVM 5 Root File System . . . . .	20
4.5	Adding runtime search paths to ELF Executables . . . . .	20
4.6	Final Stage . . . . .	21
4.7	Resulting CernVM 5 Base Layer Image . . . . .	21
4.8	<code>cernvm_config</code> Command . . . . .	22
4.9	Integration of a Graphical Interface . . . . .	23
4.9.1	Web-Based Jupyter Notebooks . . . . .	23
4.10	Derived Images with pre-loaded CernVM-File System Cache . . . . .	26
4.11	Build Infrastructure . . . . .	26
4.12	Implementation of Tests . . . . .	26
4.12.1	CI Integration . . . . .	27

4.13	Distribution . . . . .	27
4.14	Container Deployment . . . . .	28
<b>5</b>	<b>System Application Area on CernVM-File System</b>	<b>29</b>
5.1	Experimental Installation using Spack . . . . .	29
5.2	System Application Area RPM . . . . .	29
5.3	Stock RPMs with rpath . . . . .	29
5.4	Resulting CernVM 5 System Application Area . . . . .	30
<b>6</b>	<b>Virtual Machine Images</b>	<b>31</b>
6.1	Virtual Machine RPM . . . . .	31
6.2	Contextualization of Cloud Instances . . . . .	31
6.3	Creating the Root File System . . . . .	32
6.4	Creating and Mounting a Bootable Partition . . . . .	33
6.5	Installing the Root File System and Bootloader . . . . .	33
6.6	Master Boot Record . . . . .	34
6.7	Resulting CernVM 5 Virtual Machine Image . . . . .	34
<b>7</b>	<b>Conclusion</b>	<b>36</b>

## List of Figures

1.1	The Worldwide LHC Computing Grid . . . . .	7
2.1	Hardware Virtualization Model . . . . .	9
2.2	Container Virtualization Model . . . . .	10
2.3	CernVM-FS Architecture . . . . .	12
2.4	ld Look Up Chain . . . . .	14
3.1	The CernVM 5 System Design . . . . .	17
4.1	Staged Root File System Build . . . . .	20
4.2	Jupyter Landing Page . . . . .	24
4.3	Example ROOT Jupyter Notebook Workflow . . . . .	25
4.4	Example Plot created with ROOT . . . . .	25
4.5	CernVM 5 Nightly Builds and Test Results . . . . .	27
6.1	CernVM 5 running in VirtualBox . . . . .	34
6.2	CernVM 5 running in QEMU/KVM . . . . .	35

## Listings

1	Segment of the cernvm-system-default <b>spec</b> file . . . . .	19
2	Copying Root File System to scratch . . . . .	21
3	The cernvm_config Command . . . . .	23
4	Running CernVM 5 Container using Docker . . . . .	28
5	Mounting CernVM-FS inside a CernVM 5 Container . . . . .	28
6	Sourcing the LCG Software Stack Environment . . . . .	28
7	Segment of the System Application Area RPM <b>spec</b> File . . . . .	29
8	Environment Variables Breaking Dynamic Executables and Libraries . . . . .	30
9	rpath of an Executable installed on CernVM-FS . . . . .	30
10	Snapshots of the CernVM 5 System Application Area . . . . .	30
11	Virtual Machine RPM . . . . .	31
12	cloud-init Configuration . . . . .	31

13	CernVM 5 Kernel Dockerfile . . . . .	32
14	The syslinux.cfg Configuration File . . . . .	32
15	Exporting the prepared File System . . . . .	33
16	Creating a bootable partition . . . . .	33
17	Mounted Partition and Workspace . . . . .	33
18	Setting up the Root-FS . . . . .	34
19	Copying the MBR . . . . .	34
20	Creating a CernVM Instance on OpenStack . . . . .	35
21	Attaching a Virtual Volume to CernVM 5 Instance . . . . .	35
22	Mutli-staged CernVM 5 Dockerfile . . . . .	37

## List of Tables

1	Resulting CernVM 5 Base Layer Comparison . . . . .	22
2	Container Runtime Versatility . . . . .	22
3	CernVM 5 Scenario Image Performance . . . . .	26

# 1 Introduction

The European Organization for Particle Research (CERN) is the world’s largest particle physics laboratory. CERN operates high-energy particle colliders, in particular the Large Hadron Collider (LHC) where the four major experiments ATLAS, CMS, ALICE and LHCb are located. In the quest to gain further understanding of fundamental physics, scientists employ many software frameworks and tools that run over the petabytes of data collected by these particle detectors.

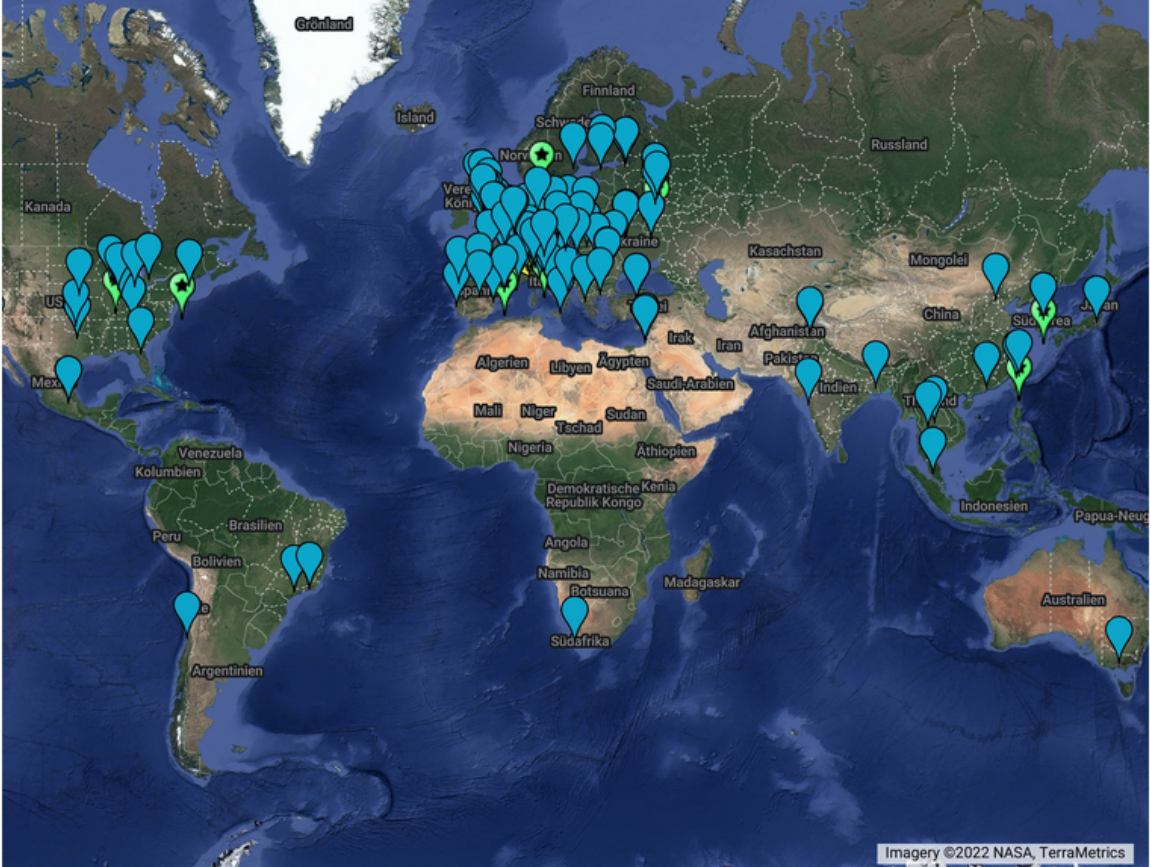


Figure 1.1: The Worldwide LHC Computing Grid [1] is a collaborative project of more than 170 computing centers located in over 40 countries.

Since the billions of collisions or “events” produced by the collider are statistically independent, the resulting data can be analyzed in data-parallel jobs. This enables CERN to take advantage of High-Throughput Computing (HTC) [2]. HTC provides large amounts of computing resources distributed on multiple systems for arbitrary periods. In contrast, in High-Performance Computing (HPC), large computational power is available on a single supercomputer to process inherently interconnected and dependent problems. To handle the computational demand, the High Energy Physics (HEP) community operates on the Worldwide LHC Computing Grid (WLCG) [3] with its sites illustrated in Figure 1.1 above. Distributing large and frequently updated experiment software to this distributed and federated HTC computing network, as well as to scientists developing analysis code, is a central and difficult task. For this large landscape of users, often running a variety of Compiler-OS combinations, up-to-date software stacks have to be distributed and retained in a performant and reproducible manner.

## 1.1 Project

The goal of the project is to develop the next generation of the minimal Linux image CernVM. With operators shifting their infrastructure further towards container virtualization, CernVM 5 will focus on providing a containerized environment for HEP applications. CernVM 5 will run in common container runtimes, such as Docker, Kubernetes, Podman or Apptainer, while still working equally well as a full virtual machine on desktop hypervisors or cloud infrastructure, such as OpenStack. Designed as a “Just Enough Operating System (JeOS)”, CernVM 5 will serve as a minimal virtual runtime environment for CernVM-File System (CernVM-FS) and the externally managed scientific software stacks distributed via this globally scalable read-only file system. Capable of serving multiple scientific software stacks, CernVM 5 will be a portable platform for developing and running scientific analysis code. To keep the base image “Just Enough”, additional applications, e.g., for interactive use cases, will be outsourced to a separate System Application Area on CernVM-FS. This includes common utilities like editors, but also a web-based graphical interface for developing analysis code. To enable such a distributed installation, suitable build methods for both the image and the packages on CernVM-FS have to be implemented. All images and artifacts will be built and tested in an automated pipeline and the project will be complemented with adequate documentation.



## 2 Basics

The following chapter summarizes basic concepts of virtualization techniques and software distribution in High Energy Physics.

### 2.1 Virtualization

“Virtualization is a way to abstract applications and their underlying components away from the hardware supporting them and present a logical or virtual view of these resources. This logical view may be strikingly different from the physical view. The goal of virtualization is usually one of the following: higher levels of performance, scalability, reliability/availability, agility, or to create a unified security and management domain.” [4]

#### 2.1.1 Virtual Machines

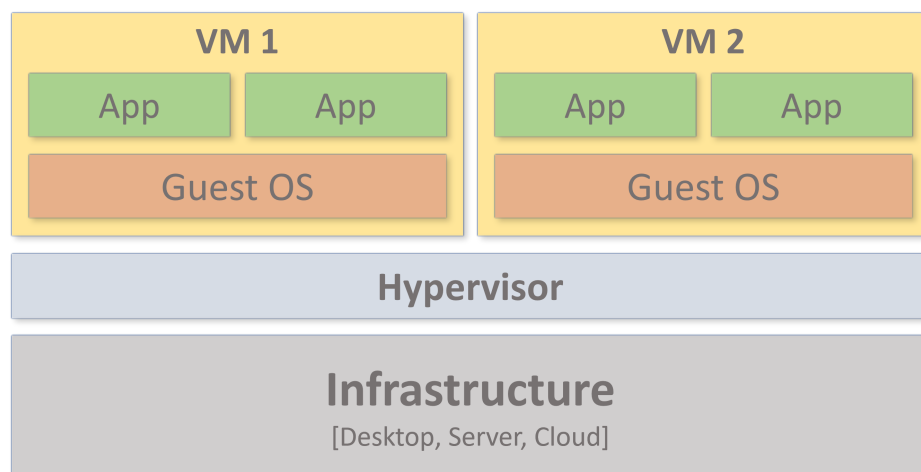


Figure 2.1: A hypervisor is used to allocate the available resources of a host system to multiple guest systems. The guests are referred to as virtual machines since each runs an entire operating system, including a kernel.

A virtual machine describes a fully virtualized and self-contained computer system. Virtual machines resemble real physical machines. Therefore, a VM image includes an operating system with a kernel and the applications it is supposed to run. This creates an isolated environment, e. g., for developing, testing or deploying software and services. As shown in Figure 2.1, multiple virtual machines running arbitrary operating systems can be installed on a single host using a hypervisor. This layer, in between the host and the guest system, is used to manage and partition the available resources of the host system to make flexible use of them. Hardware-level virtualization leads to high isolation between host and guest, which is beneficial in terms of security, but a disadvantage considering performance and efficient usage of resources. This is due to the rather large overhead of the hypervisor compared to, e. g., container engines. In addition, a given host system usually only supports one active hypervisor at a time. Nonetheless, in many use cases, full virtual machines are still required, e. g., in security-sensitive environments or when it comes to emulating hardware. Modern hypervisors support processor emulation for running software built for architectures deviating from the host system, e. g., running x86 binaries on ARM, which is especially useful when it comes to long-term data preservation.

## Hypervisors

A variety of hypervisor products are available for different purposes of virtualization. VirtualBox [5] is a popular tool for desktop virtualization. It is available for the three major platforms Linux, Windows and macOS and comes with a graphical interface to manage guests and resources. VirtualBox provides a wide set of features when it comes to configuring guests, networking, emulating hardware as well as snapshotting virtual machines and volumes. Since it is focused on desktop users, it also provides graphical host integration services, e.g., a shared clipboard between host and guest. The virtualization module KVM [6] enables the Linux kernel [7] to serve as a hypervisor. KVM is part of the mainline Linux kernel and can be used for virtualizing a large variety of supported guest operating systems (Linux, Windows, Solaris, macOS, etc. [8]) on Linux hosts. KVM itself provides low-level virtualization (CPU, memory, I/O). For higher-level device emulation, virtual machine and resource management as well as graphical interfaces, KVM relies on third-party software such as QEMU [9]. In the CERN data center, as an example, the cloud computing platform OpenStack [10] is used as an infrastructure-as-a-service (IaaS) layer to provide KVM-virtualized machines to users.

### 2.1.2 Containers

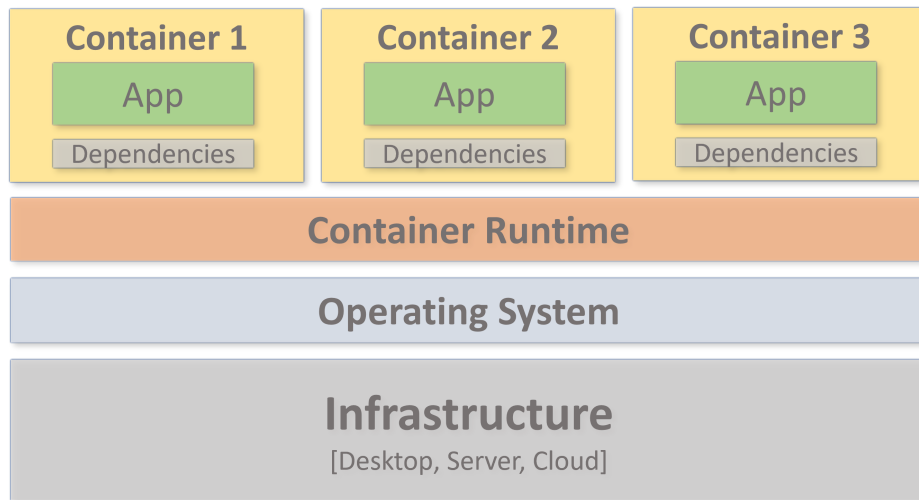


Figure 2.2: Container runtimes are used for virtualization on the operating system level. A container is started from an image containing an operating system’s root file system. Containers running on a single host are isolated from each other but use the same kernel provided by the host.

Containers provide a virtualized environment on the operating system level, as pictured in Figure 2.2. Containers are started from images that bundle a layered root file system of an OS with needed applications, dependencies and configurations. Images are built by extending and customizing an existing base image available in standardized formats on image registries. Based on the respective file system layers of such an image, container runtimes construct a consolidated root file system using OverlayFS [11] and start a process tree in a separate namespace [12] that resembles a container. Through namespaces, containers are unaware of other containers running on the same host. While each container appears to have exclusive access to the kernel, the available resources and capabilities are limited by control groups (cgroups) [13]. Although the shared kernel model loosens the isolation between host and guests, it comes with several advantages. Container images are easier to customize

and distribute since they do not contain a kernel and are therefore more lightweight than VM images. Compared to the roughly 15% performance overhead [14] caused by virtual machine hypervisors, container runtimes cause a negligible overhead, making containerized deployments more resource-efficient. Containers can be started using different container runtimes simultaneously on a single host. Since no kernel or emulated hardware has to be bootstrapped, containers can be started virtually instantaneously.

### Container Runtimes

The Docker [15] platform provides a variety of container tools. Images can be pulled from the DockerHub registry [16] and customized using the later described `build` command. Besides the actual container runtime, Docker provides further tooling for running services, managing persistent data volumes and cluster deployments. Podman [17] offers a similar set of features without the need to run a system daemon. This is particularly useful when operating on a host system without elevated privileges. Apptainer [18] (previously called Singularity) focuses on scientific High-Performance Computing and is therefore tuned for running in supercomputer environments.

### Images and Build Tools

Container images consist of layered file system trees. Base layer images are built from scratch and contain the root file system of an OS. Based on these general images available on public registries, application-specific child images can be built by adding a further file system tree to the underlying read-only parent image. A Dockerfile [19] can describe various aspects of an image build process. It contains instructions to further customize a parent image, e. g., by installing packages, pre-configuring an application or to set up an entry point. Each of these instructions adds a reusable layer to the final image. Although Dockerfiles originated from Docker, other build tools such as the daemonless Buildah [20] tool support them as well. Final images can be distributed via public or private registries.

## 2.2 CernVM-File System

“The CernVM-File System (CernVM-FS) provides a scalable, reliable and low-maintenance software distribution service. It was developed to assist High Energy Physics (HEP) collaborations to deploy software on the worldwide-distributed computing infrastructure used to run data processing applications. CernVM-FS is implemented as a POSIX read-only file system in user space (a FUSE module).” [21] The functional principle and architecture of CernVM-FS are described and pictured on the next page.

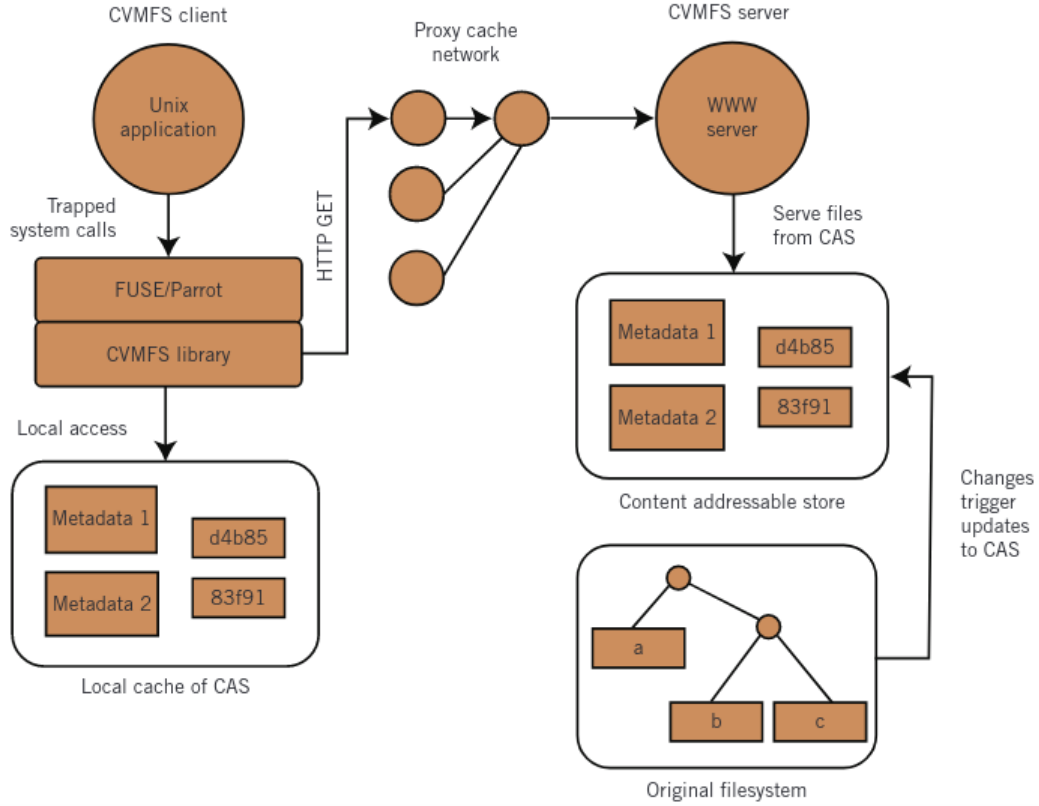


Figure 2.3: The software installed on CernVM-File System servers can be accessed by mounting the CernVM-FS client in user space. The needed software is then loaded on-demand and cached locally. [22]

The client-server architecture of CernVM-FS is pictured in Figure 2.3 above. Experiment software stack maintainers, so-called software librarians, bundle needed scientific software and their dependencies into compatible and versioned stacks, such as the LCG [23] stack. Librarians install and retain multiple versions of these stacks on a repository-specific CernVM-FS server, which is also called a publisher node. CernVM-FS is also used for container image distribution. An image, e.g., containing a scientific stack, can be provided by unpacking its root file system on a CernVM-FS publisher node.

Users and worker nodes can access the contents of the repositories by mounting the CernVM-FS client under the `cvmfs` namespace. CernVM-FS removes the need to install software locally by loading and caching the needed files and binaries installed on a remote repository on-demand. Distributing software and container images with CernVM-FS is highly efficient since the binaries and files needed during runtime usually only comprise a small subset of the available ones. The client can be mounted in userspace using Parrot [24] or the FUSE [25] framework. Although it works completely in user space, the Parrot implementation tends to be brittle and causes a considerable amount of performance overhead. Since unprivileged FUSE mounts become easier, it makes sense to use the FUSE implementation, especially in a Linux environment where the needed interface is usually already present. The FUSE module allows a filesystem to run in userspace while interfacing with the kernel through a remote procedure call layer.

## 2.3 Virtual Software Appliances

A Virtual Software Appliance describes a complete software package containing an application embedded into a dedicated “Just Enough Operating System” that serves as a runtime environment. Depending on the specific requirements and circumstances, Virtual Appliances can come in different variants. “All in one” or “fat images” are applicable and performant in rather stable deployments where operators use a specific stack for long periods. In addition, they can be used in offline computing scenarios. For large software stacks, e. g. in HEP, where the combined volume of all active stack versions can quickly reach a 100GB, building and distributing such large images is impractical. “Special purpose” images comprise a subset of packages needed for a variant of the embedded software. In HEP, this could be an image with a single version of a given scientific stack. Although a special-purpose image is more lightweight, in many scenarios this approach leads to a large number of individual appliances. A “minimal and extendable” image not only creates an environment for the embedded software but also provides a mechanism to extend the image during run time, e. g. by mounting a globally shared file system such as CernVM-File System. Therefore, the actual image can be as small and lightweight as the minimal dependencies of the extension mechanism allow while being capable of bootstrapping all supported software stacks built and managed by other teams.

### 2.3.1 CernVM Virtual Appliance

The minimal Linux image CernVM [26] is a Virtual Appliance for a CernVM-Filesystem client. Capable of mounting CernVM-FS repositories, the image can serve as a complete and portable platform for HEP applications. Previous versions of CernVM ( $\leq 4$ ) consist of a minimal Linux kernel, the custom `pCernVM` bootloader [27] and a CernVM-FS client, which is mounted during the boot process and used to load the remaining parts of an operating system on-demand. Although this method enables tiny image sizes of  $\sim 20\text{MB}$ , it requires full control of a kernel which binds CernVM to full virtualization.

## 2.4 Dynamic Linking

Dynamic linking refers to resolving and providing needed libraries and objects to an executable during runtime. A common file format for dynamically linked programs and shared libraries is the Executable and Linkable Format (ELF) [28]. Dynamic linkers find and load shared libraries needed by ELF files. Dynamically linked executables and libraries installed on a system often depend on common libraries such as the `glibc` [29]. This saves disk space and memory compared to statically linked executables, which include all their needed library routines. In contrast, updating a shared library can break dynamically linked executables that depend on an older version. Besides others, such as `ld` [30] or `ld` [31], the standard linker on most Linux systems is `ld` [32]. By default, `ld` tries to resolve matching dependencies in the root directories `lib` and `lib64` on 64-bit systems. This behavior can be overwritten with several hierarchically arranged variables, as shown in Figure 2.4 on the next page.

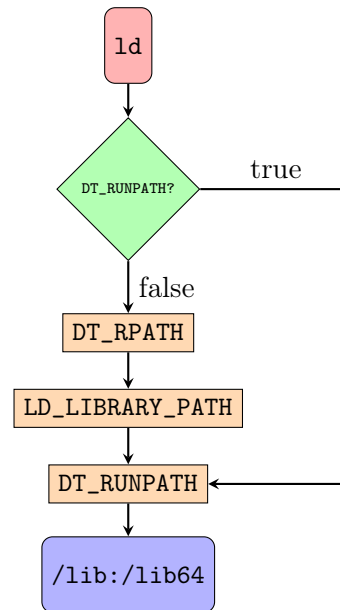


Figure 2.4: The linker `ld` tries to look up dependencies in every directory listed in each present path variable and stops on the first match. The binary header `DT_RPATH` can be added to the `.dynamic` section of an ELF file during build time and is read first if no `DT_RUNPATH` exists. `LD_LIBRARY_PATH` is an environment variable and can be exported during run time. The binary header `DT_RUNPATH` is read last if no `DT_RPATH` exists and first if it does. If none of these paths are present, `ld` falls back to the default value `/lib:/lib64`. Note that this behavior and prioritization might vary when using other linkers.

## 3 Conception

The CernVM Virtual Appliance addresses use cases in both desktop environments and large-scale batch computing. This leads to the various requirements and technical design decisions described in this chapter.

### 3.1 Use Cases

From an end user’s perspective, e. g., for developing analysis code, the image should feature a set of user applications like editors or a graphical interface. Special-purpose images should be built for interactive user sessions, e. g., tutorials, or to avoid start-up latency, e. g., when running on opportunistic resources only available for a short amount of time. CernVM 5 should serve as a containerized runtime environment for large-scale job execution on cloud infrastructure, capable of running in different container engines like Docker, Podman, Apptainer and Kubernetes. Extended with a kernel and bootloader, the image should be available as a full VM for desktop hypervisors such as VirtualBox, on cloud services such as OpenStack or EC2 [33] and for isolation-sensitive scenarios, e. g., online computing in High Level Trigger Farms [34] at LHC experiments.

### 3.2 Requirements

The CernVM 5 project aims to meet several requirements listed in the following sections.

#### 3.2.1 Minimal

By reducing its contents to a near minimum, the image is easy to distribute, less complex and can be built faster. The contents of the image are derived from the dependencies of the integrated CernVM-FS client and HEP\_OSlibs [35]. This meta-package defines a set of dependencies that creates a widely accepted base platform for running software distributed via CernVM-FS.

#### 3.2.2 Derivability

Since deriving from a base layer image is one of the most popular and useful features of containers, the new version should be derivable in a native manner using standard tools like Dockerfiles. Consequently, for both build and runtime, CernVM 5 images cannot rely on any further custom logic or non-native features of common build tools.

#### 3.2.3 Maintainability

To keep the components of the project easily maintainable and compatible, CernVM 5 should rely on stock packages from common repositories. Further customizations should be automated and integrated into the build process.

#### 3.2.4 Reproducibility

Especially in a scientific environment, the reproducibility of results is a necessity. To be able to later re-run previous calculations and analysis, the used packages and versions should be documented and available for any given component.

### 3.2.5 Turnkey

Users should be able to run both the container image and the full VM without the need for additional configurations or efforts to get to a basic deployment. All components should be distributed with a customizable default configuration.

### 3.2.6 Separated System Application Area on CernVM-FS

Similar to the previous version, but to a lesser extent, CernVM 5 should take advantage of proven mechanisms to outsource system applications to CernVM-FS. This not only allows for a smaller image but also creates a common set of applications.

### 3.2.7 Versatility

The container image should be supported by standard runtimes such as Docker, Podman, Apptainer and Kubernetes. The extended full virtual machine image should run in common desktop hypervisors such as VirtualBox and should be available on cloud infrastructure such as OpenStack.

### 3.2.8 Automated Build and Test Infrastructure

An automated build pipeline for all artifacts should allow future developers to make modifications to the code without having to deal with the corresponding build process. This also includes the integration of tests that ensure core functionality.



### 3.3 Design

The CernVM 5 system consists of three components shown in Figure 3.1 below and further described in the following sections.

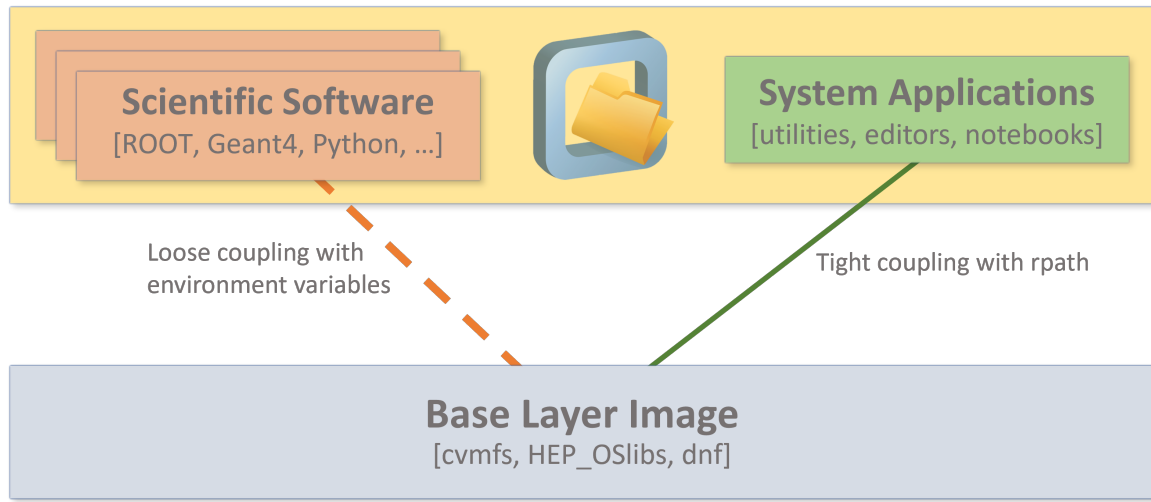


Figure 3.1: The CernVM 5 system consists of the base layer image, serving as an appliance for a CernVM-FS client, the CernVM 5 system applications, and the scientific software. While both the system applications and scientific software are distributed via CernVM-FS, the latter is managed by external groups who use environment variables rather than hard-coded runtime search paths to couple their installed software with the base layer.

#### 3.3.1 Base Layer

The base layer serves as an appliance for a CernVM-FS client. Capable of mounting software repositories and complemented with the meta-package `HEP_OSlibs`, the base layer creates a compact and common platform for developing and running analysis code. Build from stock packages, it can be easily extended using standard container build tools and package managers.

#### 3.3.2 System Applications Area on CernVM-FS

The system applications area repository complements the base layer with user-focused applications loaded on demand. This includes common tools for interactive user sessions, e. g., editors and command-line utilities.

#### 3.3.3 Scientific Software

External maintainers, so-called Librarians, e. g., from experiment collaborations, bundle scientific software into complete and consistent stacks distributed via CernVM-FS. Setting up the environment for the remotely installed software can be achieved in many ways, but maintainers mostly rely on providing a setup script that sets up environment variables such as `PATH`, `LD_LIBRARY_PATH` or `PYTHONPATH` to point to the corresponding installation. This loose coupling allows both users and maintainers to be highly flexible without any further tooling. Therefore, the CernVM system cannot interfere with this established mechanism.

### 3.4 Challenges

Due to the given requirements, the following problems have to be addressed.

#### 3.4.1 Dependency Linking

The previously described method to set up scientific software stacks may be convenient and effective, but also leads to problems when it comes to linking dependencies of binaries from different installation roots, namely the CernVM 5 application area and the image itself. When dealing with distributed installations, the usage of search path environment variables is often error-prone, because most packages rely on standardized structures to resolve dependencies properly. For instance, the dynamic link loader `ld` resolves needed shared libraries and objects by default in the root directories `lib` and `lib64`. The corresponding environment variable for overwriting this default value to set up a scientific software stack is `LD_LIBRARY_PATH`. This higher prioritized path points to the stack-specific directories on CernVM-FS. Fatal errors occur when an application installed in the image or on the system applications area resolves a common shared library or object in a stack-specific directory, but with a mismatching version. Thus, the library dependency paths of CernVM 5 system applications must be fixed to ensure the usability of the image, regardless of the environment variables.

#### 3.4.2 Graphical Interface and User Integration

CernVM also targets users in a desktop environment. Therefore, the container image requires an adequate graphical interface for developing analysis code. However, installing fully-fledged graphical applications into the image would certainly inflate the image size to an impractical volume. Outsourcing the applications to the system applications area would solve this issue, but would still require many users to find an individual configuration for a graphics-enabled container deployment for a limited set of GUI applications. Therefore a suitable and lightweight solution, especially for tutorial scenarios, has to be investigated.

#### 3.4.3 CernVM-FS Cache Management

The software installed on CernVM-FS is loaded on-demand. This means rather slow internet connections can lead to some start latency. For interactive use cases such as tutorials, a more fluent and responsive user experience could be achieved by distributing special-purpose images with pre-populated caches. Avoiding the initial need to load large amounts of data also helps to take full advantage of ephemeral computing resources, e.g. in the case external computing resources are temporarily available. Proper build techniques for such images had to be investigated and implemented to keep the caches consistent and up to date. Since CernVM-FS caches can quickly grow to ten gigabytes and more, users operating multiple instances of CernVM 5, e.g., in a Kubernetes cluster, should be encouraged to use shared caches.

## 4 Implementation of the Container Image

This chapter describes the implementation of the CernVM 5 container image and its build system. A specialized container image build process was implemented, which includes post-build processing of executables installed in the image. The chapter further comprises the integration of the CernVM-FS client and a web-based graphical interface.

### 4.1 Source Code Repository Structure

All sources of CernVM 5 are available on `github.com/cernvm/cernvm-five`. The `docker` directory contains all Dockerfiles needed to build the base layer image or derivatives of it. Scripts used during the container image build are located under the `system` directory. The `config` directory resembles the `etc` directory of a Linux operating system and contains configuration files. Sources for Red Hat Package Manager packages (RPM) [36] are stored in the `rpm` directory. Files in the `vm` directory are used to extend container images to full virtual machines. The `build` directory contains CI scripts and Dockerfiles for building all CernVM 5 artifacts in containerized environments. The `test` directory contains the test suite and the project documentation can be found under the `doc` directory.

### 4.2 Building RPMs

In this project, RPM packages are used to bundle consistent and versioned sets of configuration and system contents. RPMs are built from `spec` files [37].

#### 4.2.1 System RPM

Unlike most RPMs which contain a set of files, binaries or source code (`.src.rpm`), the `cernvm-system-default` meta-package only bundles the dependencies of the CernVM 5 base image, which can be seen in Listing 1 below. Besides the `bash` [38] shell and the `dnf` [39] package manager, this includes the later described CernVM 5 configuration package, the HEP\_OSlibs meta-package and the CernVM-FS client.

```
Requires: bash
Requires: dnf
Requires: cernvm-config-default
Requires: wlcg-repo
Requires: HEP_OSlibs
Requires: cvmfs-release
Requires: cvmfs
```

Listing 1: The listing shows parts of the `spec` file of the `cernvm-system-default` package. The keyword `Requires` is used to specify a dependency of an RPM package.

#### 4.2.2 Configuration RPM

The goal of the `cernvm-config-default` RPM is to make the image easily usable by distributing it pre-configured. This includes a default configuration for the CernVM-FS client. The `90-cernvm.conf` file contains parameters for needed repositories, e.g., the project-specific `cernvm-five.cern.ch` development repository (see Chapter 5). Further configuration for this repository is included in `cernvm-five.cern.ch.conf` along with its public key for verifying the digital signature of CernVM-FS file catalogs. The later described CernVM 5 helper script `cernvm_config` (see Section 4.8) and the source file `cernvm_env.sh` is installed into a new directory `cernvm` located under `etc`.

### 4.3 Multi-Staged Build

The main Dockerfile (see Listing 22 in Appendix) is the central artifact throughout the project. It is used to orchestrate the needed build steps and commands which result in the CernVM 5 base layer. Commonly, container images are built by extending and modifying an existing base layer image. Although this approach is sufficient in most cases, building the image in a staged manner [40] comes with several advantages. The general idea is to separate packages only needed during the build process from the desired final image. In practice, the `FROM`-instruction is used to choose a container image, in this case, the latest version of Rocky Linux 8 provided by the official DockerHub registry. This image is defined as an intermediate `builder` container using the `AS`-instruction. The `RUN`-instruction is used to further customize and extend the `builder` by running commands during build time. This includes creating the target directory `build`, installing standard packages needed during the build, such as `gcc` or `make`, and building the later described `patchelf` [41] utility. Docker or other container build tools such as Buildah cache the `builder` container to speed up repeated builds.

### 4.4 Building the CernVM 5 Root File System

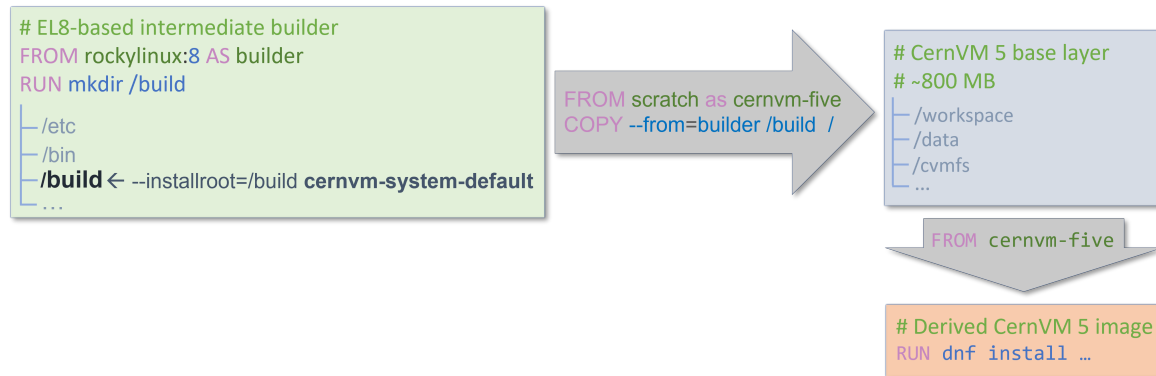


Figure 4.1: The previously prepared build container installs all packages and their dependencies, which define the CernVM 5 base layer, into an empty directory. This creates the root file system of the final, single-layered CernVM 5 container image, which is copied to the empty image `scratch`.

Started during the build, the `builder` container is used to construct the root file system of CernVM. The `dnf` package manager is used to `rootinstall` the `cernvm-config-default` and `cernvm-system-default` RPM into the `build` directory, as shown in Figure 4.1. By resolving the dependencies defined in these RPMs, the root file system of the final container image is created, which contains only necessary files and no unwanted caches, log files or additional packages. This leads to a smaller image size compared to an image providing the same desired functionalities but directly derived from Rocky Linux 8. See Section 4.7 for a comparison of the different approaches.

### 4.5 Adding runtime search paths to ELF Executables

Regarding the stability of dynamically linked executables, the use of environment variables to set up scientific software stacks would lead to the problems described in Section 3.4.1. Findings of an experimental installation (see Section 5.1) led to the conclusion that both ELF executables installed in the image and on the CernVM-FS System Application Area have to be manually linked with their needed shared libraries by adding a runtime search

path. Since this is usually done during build time, suitable tooling for adding a `DT_RPATH` to the header of a stock ELF file had to be investigated and implemented. The command-line utility `patchelf` can be used to add and modify such a `DT_RPATH` to already installed executables. The custom script `patch.sh` was implemented to automate this process by evaluating each ELF executable installed in the image and adding a `DT_RPATH` pointing to its default directories for shared libraries. The script is embedded into the main Dockerfile but can be used universally, e.g., to patch the System Application Area after new packages have been installed.

Adding a `DT_RPATH` to the ELF executables prevents previously occurring problems when shared libraries of CernVM 5 components mismatch with those needed by scientific stacks.

## 4.6 Final Stage

As seen in Listing 2 below, the root file system located in the `build` directory of the build container is copied to the empty image `scratch`. Unlike most common images, which are created by adding layers to an already existing one, this build method results in a single-layered image, which aligns with the concept of base layer images. The image is distributed as the smallest possible unit, making it less complex.

```
FROM scratch AS cernvm-five
COPY --from=builder /build /
```

Listing 2: The `COPY`-instruction can be used to copy files or directories of a previous build stage to the current one.

Meta-data labels are added, e.g., with information about the maintainers of the image. The `EXPOSE`-instruction is a purely symbolic suggestion for users to forward a specific port of a container. In this case, the default port of the later-described Jupyter web interface (see Section 4.9.1) is recommended. The stop signal `SIGTERM` allows for a graceful shutdown and is therefore the default value. To provide an overwritable default entry point, the `CMD`-instruction is used to start a bash on startup when nothing else is specified.

## 4.7 Resulting CernVM 5 Base Layer Image

The previously described steps create a minimal base layer for CernVM 5. Tailored around the dependencies of CernVM-FS and scientific software, CernVM 5 containers are capable of mounting CernVM-FS software repositories. This not only allows users to run scientific software inside the container but also to make it available on the host system by bind mounting the container's `cvmfs` directory to the host. With an uncompressed size of ~800MB, the CernVM 5 image is smaller compared with the 1.03GB image providing the same desired packages, but built the common way by directly deriving from a base layer. See Table 1 for a comparison between the two approaches. Unlike previous versions of CernVM, the single-layered CernVM 5 image can be natively used as a parent image to derive from using standard build tools. By adding `DT_RPATH`, the executables of the image can be used regardless of the given environment variables. With a suitable method implemented to do so post-build, the image can be built from stock packages. Furthermore, this enables users to patch their own derivatives of CernVM 5. Distributed with a default configuration, the image can be deployed out of the box using a variety of runtimes, such as Docker, Podman, Apptainer and Kubernetes.

Table 1: This table compares the CernVM 5 base layer image (v5.1.1), built in a custom build chain to keep it close to the dependencies defined by the CernVM-FS and HEP\_OSlibs, to a standard derived image. Although the standard image (created by simply installing the `cernvm-system-default` package) provides the same desired packages and applications, it contains unwanted packages and other artifacts that combine to over 200MB. CernVM 5 also conforms to the concept of a base layer by consisting of only one file system layer representing the smallest possible unit which still brings all the benefits of CernVM. Additionally, the ELF executables installed in the CernVM 5 base image are post-build processed to use `DT_RPATH`, making them consistently usable in the presence of environment variables setup by scientific stacks.

	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	✓	✓
		(But not as a true base layer)
Use of rpath	✓	✗

Table 2: CernVM 5 containers can make CernVM-FS available using various container engines by mounting it inside using the FUSE interface. Furthermore, if CernVM-FS is already present on the host, it can be bind mounted to CernVM 5 containers, which also comes with the benefit of a shared CernVM-FS cache among the containers. Vice versa, a CernVM 5 container can be used to forward CernVM-FS to a Linux host. The possibility of pre-mounting CernVM-FS to each started CernVM 5 container by default is unique to Apptainer.

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

#### 4.8 `cernvm_config` Command

The `cernvm_config` command was implemented to automate different actions when working with CernVM 5. The command wraps recurrent workflows, e. g., mounting CernVM-FS repositories. This is especially useful for common CernVM-FS repositories such as `sft.cern.ch` or `cvms-config.cern.ch`, which are basically considered to be available in CernVM 5 by default. Since the CernVM 5 System Application Area (see Chapter 5) is also mounted in almost every session, it can be mounted with a specific option (`mount -s`), as can be seen in Listing 3. The sub-command `setup` automatically determines a compatible (architecture, CernVM 5 image version) System Application Area snapshot on CernVM-FS and sets up the `PATH` variable to use it.

```
$ cernvm_config --help
Usage: cernvm_config <command> [options]
Command      Description
mount        Used for mounting CernVM-FS repositories
Options:  -a  Mounts every default CernVM-FS repository
              specified in 90-cernvm.conf
           -s  Mounts cernvm-five.cern.ch for system applications
           -r  Mounts CernVM-FS repository [-r <domain>]
setup        Sets up PATH for compatible system applications
version      Prints versions of CernVM 5 components
```

Listing 3: The `cernvm_config` is a useful addition to CernVM 5. It can be used to substitute recurrent and repetitive commands, e. g., in automation scripts or as an entry point to mount specified CernVM-FS repositories.

## 4.9 Integration of a Graphical Interface

For users deploying the container image in a desktop environment, it is necessary to provide a graphical interface, e. g., to develop or compile analysis code using a scientific stack. Although it is possible, installing graphical applications into a container image is rather impractical for two reasons. First of all, installing the applications and their respective dependencies would significantly increase the image size for the sake of a rather rare use case of CernVM 5. While outsourcing the large packages to the system application area would solve this problem, actually running them comes with prerequisites regarding the host system, such as an X11 server [42] open for incoming connections. To avoid these problems, a web-based graphical interface was integrated and is presented in the following.

### 4.9.1 Web-Based Jupyter Notebooks

A web-based solution avoids further host requirements and also tedious configurations for users. Jupyter [43] was chosen because it is popular among scientists but also because it is already available on many scientific software stacks on CernVM-FS. Jupyter Notebooks provide an interactive development environment for over 40 programming languages, including C++, Python, R, Julia or the HEP analysis framework ROOT [44]. Jupyter adds additional value to CernVM 5 by providing a basic file browser and the option to launch a web-based shell in the container. Enabling a locally hosted Notebook only requires forwarding the default port 8888 of the container to the host using the `--port` option when starting it. In this case, the Jupyter installation provided by the LCG stack installed on the `sft.cern.ch` repository is used.

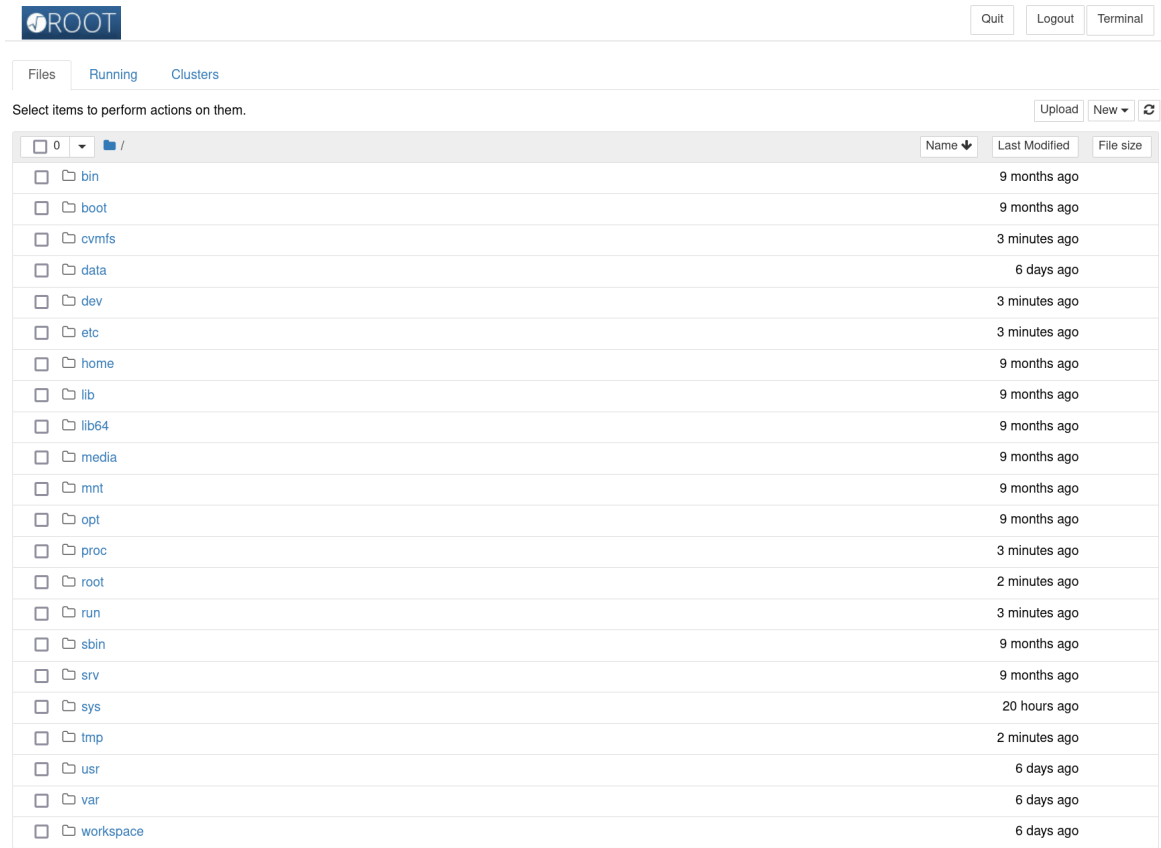


Figure 4.2: This Jupyter instance is hosted by a CernVM 5 container. The landing page allows users to browse the container’s file system.

The landing page shown in Figure 4.2 allows the user to browse the file system of the container and create new Notebooks, terminals and files. The Notebook is used to create ordered lists of input and output cells containing code, text, mathematics expressions, plots or other rich media contents. Since the environment of the underlying CernVM 5 container is set up to use the LCG stack, a new ROOT-specific C++ Notebook is started to showcase the UI and workflow. By adding and running cells, as can be seen in Figure 4.4, a plot is created and copied to the `workspace` directory which, e.g. can be bind mounted to the host system or a container volume.



## 4 Implementation of the Container Image

```
ROOT Notebook Last Checkpoint: a few seconds ago (autosaved)
File Edit View Insert Cell Kernel Help Trusted ROOT C++ O
In [1]: TCanvas *c = new TCanvas;
In [2]: TH1F *h = new TH1F("gaus", "gaus", 100, -5, 5);
In [3]: h->FillRandom("gaus", 10000);
In [4]: h->Draw();
In [5]: c->Update();
In [6]: gSystem->ProcessEvents();
In [7]: TImage *img = TImage::Create();
In [8]: img->FromPad(c);
In [9]: img->WriteImage("workspace/canvas.png");
In [10]: delete h;
In [11]: delete c;
In [12]: delete img;
```

Figure 4.3: The cells created in this Jupyter Notebook contain ROOT code and are run sequentially to create a plot.

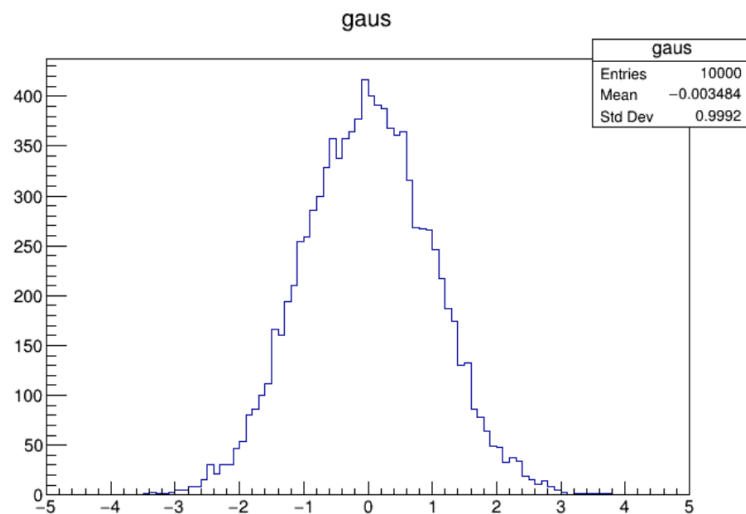


Figure 4.4: Jupyter Notebooks can also display rich media in web browsers, as seen in this figure. The previously created plot using a ROOT Notebook is written into the **workspace** directory, which can be bind mounted to the host.

### 4.10 Derived Images with pre-loaded CernVM-File System Cache

For interactive use cases, the start latency problem described in Section 3.4.3 has to be addressed. In this particular case, a proof-of-concept special-purpose image is built for the Future Circular Collider (FCC) [45] Starterkit tutorial [46]. The needed software for this tutorial is available on CernVM-FS and frequently updated by librarians. Therefore, it is important to populate the cache in an automatic and dynamic manner to keep it in sync with the latest available software. In the case of the FCC tutorial image, the user’s workflow is recreated by running the already available tests inside a CernVM 5 container with the latest available stack mounted. The thereby created cache covers the exact software needed to complete the tutorial. The root file system of the used container, including the loaded cache, is then exported, e. g., using the `docker export` command. The resulting tar file can then be imported as a standalone container image, e. g., using the `docker import` command. The final image is tagged, e. g., `cernvm-five:fcc-tutorial`, and pushed to a registry from where it can be downloaded by users.

Table 3: This table compares the execution time of commands included in the Starterkit FCC tutorial. The scenario-specific FCC image, distributed with a pre-loaded CernVM-FS cache, allows users to initially run commands consistently faster than with the base image. Note that these measurements were taken 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 network caches.

	CernVM 5 Base Image	CernVM 5 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
babayaga -f <sup>1</sup>	5.541s	5.289s
Import ROOT	6.843s	1.544s
fccanalysis run <sup>2</sup>	11.230s	9.319s

<sup>1</sup> babayaga -f 15. -t 165. -e 91.2 -n 10000 -o bbyg\_10000.LHE

<sup>2</sup> fccanalysis run analysis\_stage1.py --output p8\_ee\_ZH\_ecm240.root --files-list ./p8\_ee\_ZH\_ecm240\_edm4hep.root

### 4.11 Build Infrastructure

The automated build infrastructure enables the developers to build CernVM 5 artifacts. Not having to deal with the build system allows for quick testing of changes in the code. The continuous integration was realized with Jenkins [47] and can be used to build nightly and release builds on clean build machines with different architectures such as x86 or ARM. The build system of CernVM 5 is highly portable because all artifacts are built in unprivileged containers.

### 4.12 Implementation of Tests

Being able to run tests is a crucial tool for developers when it comes to making changes in the project or updating versions. Therefore, a solid and easily extensible test suite with good coverage had to be implemented along with the actual development of the project to enable a certain degree of stability. The test suite has a strong focus on the interaction between the base image, system application area and scientific software stacks. This is done by trying to trigger previous problems regarding the dependency linking in CernVM 5 components

when setting up a scientific software stack (see Section 3.4.1). Although the image format is compatible with each supported container engine (see Section 3.2.7), errors during runtime can still occur due to major or minor differences in the specific implementation.

A custom test runner is used to configure a test run. This includes which test cases are executed on which container engine with either CernVM-FS mounted inside the container or from the host. By default, all possible combinations are executed. Functions to create predefined container environments for each test are defined separately. These mainly include commands to start containers with different engines, capabilities or sources of CernVM-FS. Along with a file describing how to run it, a test case is mapped to a bash script which is then mounted into the respective container. After a test script finished, the result is evaluated and aggregated in a xUnit file. By running every test in a separate and clean container, potential problems can be narrowed down easily.

#### 4.12.1 CI Integration

Running automated tests is also an important part of continuous integration. Therefore, every successful build automatically triggers a test job. The new image is loaded into a clean test environment and the tests are executed. The results are displayed in the Jenkins web interface, as can be seen in Figure 4.5.

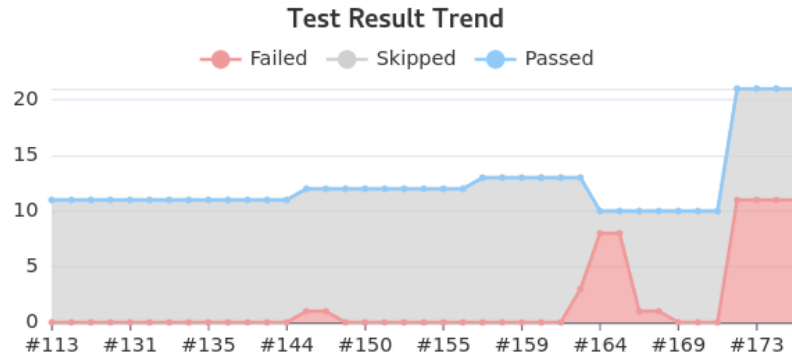


Figure 4.5: The CernVM 5 nightly builds test results are displayed in the Jenkins web interface.

#### 4.13 Distribution

Image releases, as well as nightly and special-purpose builds like tutorial images, can be pulled from the CERN Harbor registry [48]. The root filesystem of the base image is available on the CernVM-FS repository `unpacked.cern.ch` [49] for HPC runtimes such as Apptainer. The latest RPMs of the project are distributed via a yum repository [50]. This is especially useful when updating virtual machine instances of CernVM 5, which usually have longer lifecycles. All build products of the project are also publicly available on the EP-SFT webspace [51]. This area is also used to archive previous versions of images and RPMs.

#### 4.14 Container Deployment

In this section, an example CernVM 5 container is started using Docker to bootstrap scientific software from CernVM-FS.

```
$ docker run --interactive --tty \
  --cap-add SYS_ADMIN \
  --device /dev/fuse \
  cernvm bash
```

Listing 4: This command starts a CernVM 5 container, capable of mounting CernVM-FS inside using the FUSE interface.

To enable the mounting of CernVM-FS repositories inside a running container, the process tree representing it has to be provided with sufficient capabilities [52]. These can be granted in the `docker run` command with the `--cap-add` option. The relevant command `mount()` [53] requires the `SYS_ADMIN` capability set. The `--device` option adds a host device to the container, in this case, the FUSE interface located under the `dev` directory. The complete command is depicted in Listing 4 above.

```
$ mkdir -p /cvmfs/sft.cern.ch
$ mount -t cvmfs sft.cern.ch /cvmfs/sft.cern.ch
```

Listing 5: The `mount()` command attaches a file system. In this case, the `sft.cern.ch` CernVM-FS repository is mounted in user space using FUSE.

Repositories can now be mounted inside the container, as demonstrated in Listing 5 above. The `sft.cern.ch` repository provides the scientific software stack LCG [23] with over 450 packages, available for all relevant OS-compiler combinations.

```
$ source /cvmfs/sft.cern.ch/lcg/views/LCG_101/x86_64-centos8-gcc11-opt/setup.
  sh
$ which root
  /cvmfs/sft.cern.ch/lcg/views/LCG_101/x86_64-centos8-gcc11-opt/bin/root
```

Listing 6: The `source` command can be used to load variables into the current bash environment. The script provided by the maintainers includes the needed variables to use the scientific stack.

The software provided via CernVM-FS can be used, without further installations, by setting up the local bash environment. By sourcing the setup script as shown in Listing 6, environment variables like `PATH` or `LD_LIBRARY_PATH` are set to point to the executables and packages located on CernVM-FS. The listing above illustrates how for example, the analysis framework `ROOT` is now available.

## 5 System Application Area on CernVM-File System

The System Application Area extends the image by providing additional packages via CernVM-FS. This chapter regards the evaluation of suitable tooling and build methods to enable an easily maintainable System Application Area capable of working correctly in the presence of environment variables modified by scientific software stacks.

### 5.1 Experimental Installation using Spack

Spack [54] is an HPC-focused package manager that aims to be non-destructive. New or updated packages do not break older versions of a given software but rather coexist in a separate prefix. The executables and their needed shared libraries are installed into such a prefix and are linked using a runtime search path. Internally, Spack uses `patchelf` to add a `DT_RPATH` to each ELF file. An experimental installation on CernVM-FS showed that this measure is sufficient to enable the System Application Area to work properly regardless of environment variables used by scientific stacks.

Spack successfully validates the `DT_RPATH` technique. The goal of the CernVM 5 project, however, is to build all components from stock packages, installed with a standard package manager such as `dnf`.

### 5.2 System Application Area RPM

The `cernvm-system-cvmfs` RPM meta-package defines a superset of packages, including the contents of the base layer image and additional packages which are needed in interactive user sessions, such as editors. The RPM can be used to quickly build a versioned System Application Area on CernVM-FS publisher nodes.

```
Requires: iputils
Requires: nano
Requires: tree
Requires: vim
Requires: wget
```

Listing 7: This listing shows parts of the `cernvm-system-cvmfs.spec` file, which is used to build the System Application Area RPM. It defines common packages which provide utilities, e.g., editors such as `vim` or `nano`, common networking tools included in `iputils`, the `tree` command to visualize a file system tree, or `wget` to retrieve files via HTTP.

### 5.3 Stock RPMs with rpath

The content of the System Application Area is defined in the `cernvm-system-cvmfs` meta-RPM. Building a new release requires a `rootinstall` into an empty directory on a build machine. Similar to how the container's root file system is built (see Section 4.4), this creates a `chroot` environment. As a post-install step, a custom script determines all installed ELF files and adds a `DT_RPATH` to them using `patchelf`. The `DT_RPATH` points to the OS and architecture-specific prefix on CernVM-FS, as can be seen in Listing 9. The resulting directory is copied to the CernVM-FS publisher node.

## 5.4 Resulting CernVM 5 System Application Area

The System Application Area adds functionality to the minimal base image by providing common and interactive utilities via a CernVM-FS repository. Build from stock packages defined in a single RPM package, the System Application Area is easily extendable and maintainable. In an additional build step, the dynamically linked executables installed on it are automatically hard-coded to use their own dependencies. This allows CernVM 5 system applications to coexist with scientific software stacks, as demonstrated in Listing 8 below.

```
$ source /cvmfs/sft.cern.ch/lcg/views/LCG_101/x86_64-centos8-gcc11-opt/setup.sh
$ echo $LD_LIBRARY_PATH
/cvmfs/sft.cern.ch/lcg/releases/MCGenerators/thepeg/2.2.1-aa9c8/...
$ vim
Segmentation fault (core dumped)
```

Listing 8: In this listing, a scientific software stack installed on CernVM-FS is set up by sourcing a script, which includes the environment variable `LD_LIBRARY_PATH`. This environment variable overwrites local values with paths pointing to the scientific stack-specific shared libraries, ensuring that the scientific applications work properly. However, other applications which have not been supplemented with a `DT_RPATH` (e.g. the exemplary stock `vim` editor seen in this listing) break once they are linked with a mismatching version of a shared library.

```
$ patchelf --print-rpath /cvmfs/cernvm-five.cern.ch/x86_64-rocky8/bin/vim
/cvmfs/cernvm-five.cern.ch/x86_64-rocky8/lib:/cvmfs/cernvm-five.cern.ch/
x86_64-rocky8/lib64
$ echo $LD_LIBRARY_PATH
/cvmfs/sft.cern.ch/lcg/releases/MCGenerators/thepeg/2.2.1-aa9c8/...
$ vim && echo $?
0 # Successful
```

Listing 9: All dynamically linked executables installed on the System Application Area and in the CernVM 5 base image work regardless of the presence of `LD_LIBRARY_PATH`. This is achieved by adding the higher prioritized binary header `DT_RPATH` to each installed executable. The `DT_RPATH` of an executable can be displayed using `patchelf`, as seen in the listing. It includes directories on CernVM-FS where compatible shared libraries are located. Unlike the example shown in Listing 8, the editor `vim` can now be used along with scientific software loaded from CernVM-FS.

```
$ ls /cvmfs/cernvm-five.cern.ch/.cvmfs/snapshots/
x86_64-rocky8-1.0
x86_64-rocky8-1.1
```

Listing 10: Every current and future release of the CernVM 5 System Application Area is snapshotted and preserved by CernVM-FS. Each version is instantly available to users under the `.cvmfs` directory of the CernVM-FS repository.

## 6 Virtual Machine Images

This chapter describes how CernVM 5 container images are extended to work equally well as full virtual machines on desktop hypervisors such as VirtualBox and cloud infrastructure such as OpenStack or EC2.

### 6.1 Virtual Machine RPM

The `cernvm-kernel-default` RPM defines additionally needed packages as shown and described in Listing 11 below. It also includes a later-described default configuration for contextualization (see Section 6.2 below) and a boot loading configuration explained in Listing 14.

```
Requires: kernel-core
Requires: systemd
Requires: NetworkManager
Requires: openssh-server
Requires: dracut
Requires: e2fsprogs
Requires: cloud-init
Requires: sudo
```

Listing 11: The RPM depends on the packages needed to enable a fully virtualized CernVM 5. Besides the Linux kernel itself, this includes e.g. the system manager `systemd` [55], the `NetworkManager` package or an OpenSSH server [56]. The `dracut` [57] tool creates later described initial ramdisks and the package `e2fsprogs` [58] provides file system management utilities.

### 6.2 Contextualization of Cloud Instances

Virtual machine instances created on cloud infrastructure need initial configuration upon boot. This process is called contextualization. A small, usually human-readable file is provisioned to the VM, e.g., on a web server. This file is called context and can include user-specific configurations and arbitrary actions such as the automated generation and provision of an SSH key pair, creating a local user and the setup of ephemeral mount points. The contextualizer `cloud-init` [59] is used for CernVM 5 cloud images. Listing 12 below shows parts of the respective context file `cloud.cfg`, which is distributed via the `cernvm-kernel-default` RPM.

```
disable_root: 0
ssh_pwauth: 1
mount_default_fields: [~, ~, 'auto', 'defaults,nofail,x-systemd.requires=
    cloud-init.service', '0', '2']
resize_rootfs_tmp: /dev
ssh_deletekeys: 1
ssh_genkeytypes: ['rsa', 'ecdsa', 'ed25519']
```

Listing 12: This Listing shows parts of the default context of CernVM 5, which allows users to quickly get started with new instances, e.g., on OpenStack. This includes the provisioning of SSH private keys to its owners or setting up ephemeral mount points for virtual volumes.

### 6.3 Creating the Root File System

The root file system of CernVM 5 virtual machine images is prepared by extending the CernVM 5 container image or an arbitrary derivation. Listing 13 shows the Dockerfile `Dockerfile.kernel`, which includes instructions to install the `cernvm-kernel-default` RPM and to set up system services that are needed on startup. An initial ramdisk [60] contains a temporary root file system which is loaded into memory during the boot process of a Linux system to enable the mounting of the actual root file system. The tool `dracut` is used to create the initial ramdisk (`initrd`) for CernVM 5. Symbolic links are created in the `boot` directory to point to the kernel and the initial ramdisk image, which are installed in a fluctuating sub-directory of the `boot` directory. The links are automatically created using `find` to enable a stable `syslinux.cfg`, a bootloader configuration file included in the `cernvm-kernel-default` package and displayed in Listing 14 further below.

```
ARG BASE=registry.cern.ch/cernvm/five/cernvm-five:latest
FROM $BASE
USER root

### Installing cernvm-kernel-default RPM and enabling services ###
RUN dnf update && dnf clean all
RUN dnf install -y cernvm-kernel-default && \
    systemctl enable NetworkManager && \
    systemctl unmask systemd-remount-fs.service && \
    systemctl unmask getty.target

### Creating initial ramdisk ###
RUN dracut --no-hostonly --regenerate-all --force

### Creating links to kernel and initial ramdisk ###
WORKDIR /boot
RUN ln -s $(find /boot/ -name 'linux') kernel && \
    ln -s $(find /boot/ -name 'initrd') initrd.img
```

Listing 13: The root file system of a CernVM 5 container image is reused to prepare a full virtualization-enabled image by deriving from it. The needed steps for this are bundled in the Dockerfile `Dockerfile.kernel`.

```
DEFAULT linux
SAY Booting kernel from EXT LINUX...
LABEL linux
KERNEL /boot/linux
APPEND ro root=/dev/sda1 initrd=/boot/initrd.img net.ifnames=0 console=tty0
console=ttyS0,115200n8
```

Listing 14: The `syslinux.cfg` for CernVM 5 is part of the `cernvm-kernel-default` package and is installed into the `boot` directory of the root file system. The included paths point to the locations of the installed kernel and the `initrd`, or in this case the links to each of them.

The resulting container image has a volume of  $\sim 1.8GB$ . Since it consists of file system layers, a container is briefly started to unify the layers to a single root file system which is then exported to a `tar` file, as can be seen in Listing 15.



```
docker export -o rootfs.tar $(docker run -d $CONTAINER_IMAGE /bin/true)
```

Listing 15: The `docker run` command constructs the root file system of the previously built image and starts a container based on it. The container immediately stops after the specified command `true` (returns 0) finished. The `docker export` command is used to extract the root file system of the stopped container and save it to a `tar` file.

## 6.4 Creating and Mounting a Bootable Partition

A script was developed to automate the assembly of the VM images inside a built container. This container includes all needed applications and has access to the later-described host (pseudo-)devices `loop` [61] and `mapper` [62]. A buffered `raw` file is written using `fallocate` [63] and partitioned with `fdisk` [64], as can be seen in Listing 16 below.

```
(
echo n # New partition
echo p # As primary partition
echo 1 # Partition number
echo   # First sector (default 2048)
echo   # Last sector (default last possible)
echo a # Tag as bootable
echo w # Write out
) | fdisk cernvm.raw
```

Listing 16: The partition is configured and created by piping commands into `fdisk`. A new (**n**) primary (**p**) partition with the number one (**1**) is created starting from the first default sector until the last available one. The partition is then tagged as bootable (**a**) and written out (**w**).

The `loop` device makes a file accessible as a block device. The partitioned `raw` file is attached to an unused loop device with the command `losetup -f`. The new (pseudo-)block device can now be accessed under the directory `dev`. The tool `kpartx` [65] adds device mappings to the previously created partition, which makes it accessible via the `mapper` device. After formatting the partition to the `ext4` [66] format, it is mounted via its respective `mapper` device located under the `dev` directory.

```
$ cat /proc/mounts | grep 'mnt\|workspace'
/dev/mapper/cs_pcepsft100-home /workspace xfs rw,seclabel,relatime 0 0
/dev/mapper/loop3p1 /mnt ext4 rw,seclabel,relatime 0 0
```

Listing 17: The command seen in this listing outputs two mount points of the build container. The directory `workspace` is mounted to the host to import the root file system and export the image later. The second line of the output shows the mounted partition one (**p1**) of loop device three (`loop3`), which corresponds to the previously written `raw` file.

## 6.5 Installing the Root File System and Bootloader

The previously extracted container file system is copied to the partition. The needed configuration for mounting it during the boot can only be added now because it requires the UUID of the partition, as can be seen in Listing 18. The bootloader `extlinux` [67] can boot an image from an `ext4` file system and is therefore used by CernVM 5 images. The `extlinux` package also provides a Master Boot Record [68], which is explained in the following Section 6.6.

```
export UUID=$(blkid -s UUID -o value /dev/mapper/$PARTITION)
echo "UUID=$UUID / ext4 errors=remount-ro 0 1" > /mnt/etc/fstab
```

Listing 18: The CernVM 5 root file system is configured to remount on errors. The configuration requires the UUID of the partition, which can be read with `blkid` [69].

## 6.6 Master Boot Record

The first sectors of a partitioned drive are also called the boot section. It contains a Master Boot Record (MBR) which includes executable code used to start a computer system, also referred to as boot loading. The MBR provided by the `extlinux` package is copied to the image's first sectors, as seen in Listing 19 below.

```
SIZE=$(du -b /usr/share/syslinux/mbr.bin | awk '{ print $1 }')
dd if=/usr/share/syslinux/mbr.bin of=cernvm.raw bs=$SIZE count=1 conv=notrunc
```

Listing 19: The MBR is 440 bytes large and copied to the first sections of the image using `dd` [70]. Since the image has been detached from the host devices, the `notrunc` option is used to avoid truncation when writing into the `raw` image file.

## 6.7 Resulting CernVM 5 Virtual Machine Image

The resulting virtual machine image enables full virtualization of CernVM 5. Just as a CernVM 5 container, yet fully virtualized, a CernVM 5 virtual machine provides a minimal runtime environment for scientific software stacks. The image is automatically built using a custom script that reuses the processed root file system of a CernVM 5 container image and extends it with its own kernel and bootloader. Users can run CernVM 5 on various hypervisors such as VirtualBox, as seen in Figure 6.1 below, or KVM (see Figure 6.2). Images are available on the EP-SFT webspace [51] and can be converted to other formats such as VDI [71] or QCOW2 [72]. Cloud image templates are available on CERN OpenStack for users to manage and create custom contextualized CernVM 5 instances using the OpenStack CLI as demonstrated in Listing 20 or using the web interface.

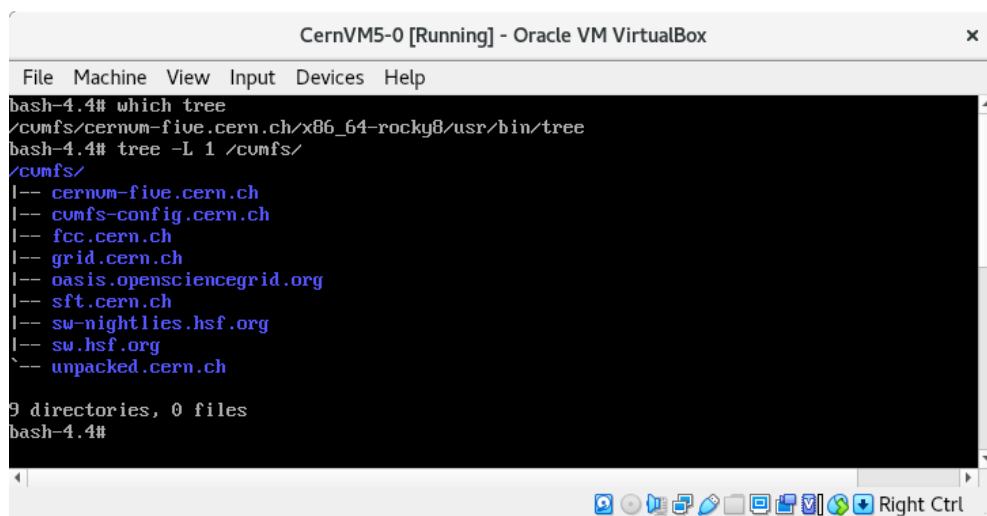


Figure 6.1: This figure shows a CernVM 5 instance running on VirtualBox. The `tree` command is installed on the CernVM 5 System Application Area. It is loaded on-demand, e.g., to display the mounted CernVM-FS repositories.

CernVM5-1 on QEMU/KVM								
File	Virtual Machine	View	Send Key					
It	Calls	Integral[fb]	Error[fb]	Err[%]	Acc	Eff[%]	Chi2 N[It]	
1	1000	1.5048458E+06	5.06E+04	3.36	1.06*	22.24		
2	999	1.5072490E+06	1.82E+04	1.21	0.38*	41.45		
3	998	1.5051831E+06	1.10E+04	0.73	0.23*	52.57		
3	2997	1.5057045E+06	9.23E+03	0.61	0.34	52.57	0.00	3
4	9999	1.5153790E+06	3.89E+03	0.26	0.26	12.13		
5	9999	1.5098341E+06	3.82E+03	0.25	0.25*	9.66		
6	9999	1.5103967E+06	3.63E+03	0.24	0.24*	9.66		
6	29997	1.5117815E+06	2.18E+03	0.14	0.25	9.66	0.63	3
Time estimate for generating 10000 events: 0d:00h:00m:02s								
n_events = 10000								

Figure 6.2: CernVM 5 can provide a fully virtualized environment for scientific software stacks distributed via CernVM-FS repositories. The figure shows a CernVM 5 virtual machine running on the KVM hypervisor. In this example, particle collision data is generated using the event generator `whizard` [73]. This scientific application is part of the Key4hep [74] software stack, which is distributed via the `sw.hsf.org` [75] repository.

```
$ openstack image list -c Name -c Status | grep 'CernVM 5'
CernVM 5.1.2a - x86_64 [2022-07-17 ] | active
$ openstack server create --image 'CernVM 5.1.2a - x86_64 [2022-07-17 ]' \
--user-data setup.sh \
--flavor m2.small \
--key-name key1 \
cernvm5-0
```

Listing 20: Users can create and manage CernVM 5 virtual machines on CERN’s OpenStack computing infrastructure. A new instance created from the available CernVM 5 cloud image template is contextualized by `cloud-init` (see Section 6.2). Users can customize the default `context` of their created VM by adding their specific user data, e.g., a setup script or SSH keys.

```
$ openstack server add volume cernvm5-0 cvm5_disk0
$ openstack volume list -c Name -c 'Attached to'
cvm5_disk0 | Attached to cernvm5-0 on /dev/vdb
```

Listing 21: The contextualization also includes setting up ephemeral mount points for virtual volumes, which users can attach to CernVM 5 machines.

## 7 Conclusion

In this thesis, a container-based Virtual Appliance for developing and running High Energy Physics applications was designed and implemented. In the process, challenges related to accessing and using large and frequently changing scientific software stacks were identified and addressed. The resulting CernVM 5 container image is a minimal yet complete platform for scientific software accessed via the integrated CernVM-FS client. Built with stock packages and standard container build tools, the image can serve as a base layer for custom images. Extended with web-based Jupyter Notebooks and utilities installed on CernVM-FS, CernVM 5 provides a complete and portable environment for developing analysis code. Additionally, the container-based CernVM 5 image can be extended to a full virtual machine image.

A special multi-stage build process was implemented, which includes the post-build processing of dynamically linked executables. Adding a runtime search path to ELF executables proved adequate for ensuring the coexistence of scientific software stacks using their own custom dependencies and standard system applications. This technique also enables the usage of standard packages in both the images and the System Application Area, making the CernVM 5 system easily maintainable by removing the need to build custom packages. Suitable build methods were implemented to create scenario images in an automated manner. These special-purpose images considerably reduce the startup latency by including a preloaded CernVM-FS cache. Finally, a custom script extends any given CernVM 5 container image to a full virtual machine by reusing its root file system and extending it with a kernel and bootloader. CernVM 5 is versatily applicable in both container and hardware virtualized environments. Container images can be deployed using various runtimes such as Docker, Podman or Apptainer. CernVM 5 can run fully virtualized on hypervisors such as VirtualBox or KVM. Since all images are distributed with a default configuration, users can quickly get to a basic deployment, especially regarding mounting CernVM-FS repositories and accessing the scientific software stacks. Detailed documentation is provided, comprising the usage and relevant technical aspects of CernVM 5. All images and other artifacts are built and tested in an automated pipeline. The test suite is easily extendable for future developments.

CernVM 5 images are publicly available on the EP-SFT webspace [51]. Internally, CernVM 5 cloud templates are available on CERN OpenStack. As of today, CernVM 5 is foreseen to be adopted by CERN's web-based analysis service SWAN [76] and the maintainers of the Key4hep stack, who will use CernVM 5 as a build environment for their software stack for future particle detector studies. As for future developments of CernVM 5, the dependency linking of interpreted languages such as Python should be automated and put in production. Furthermore, the image should be made available for more architectures such as ARM or PowerPC.

## Appendix

```
FROM docker.io/library/rockylinux:8 AS builder
ENV BUILD_DIR=/build
WORKDIR /

### Installing builder packages ###
COPY /system/install/repo.conf/ /etc/yum.repos.d/
COPY /system/install/dnf.conf /etc/dnf/dnf.conf
RUN dnf update && dnf clean all
RUN dnf install autoconf \
    epel-release \
    file \
    findutils \
    gcc \
    gcc-c++ \
    gdb \
    git \
    glibc-devel \
    grep \
    libtool \
    make \
    wget \
    which

### Installing patchelf ###
### https://github.com/NixOS/patchelf ###
RUN git clone https://github.com/NixOS/patchelf && \
    cd /patchelf && \
    ./bootstrap.sh && \
    ./configure && \
    make && \
    make install
WORKDIR /

### Installing cernum-system-default into /build ###
RUN mkdir /build
COPY /system/install/install.sh /
RUN bash install.sh

### Adding rpath to ELF executables ###
COPY /system/install/patch.sh /
RUN bash patch.sh $BUILD_DIR

### Creating mount points ###
RUN mkdir $BUILD_DIR/workspace
RUN mkdir $BUILD_DIR/data
```

```

### Copying /build to scratch ###
FROM scratch AS cernvm-five
COPY --from=builder /build /
LABEL MAINTAINER="Jakob Eberhardt jakob.karl.eberhardt@cern.ch, Jakob
    Blomer jblomer@cern.ch"
LABEL io.k8s.display-name="CernVM"
LABEL io.k8s.description="A container image providing CernVM-FS and
    HEP_OSlibs"
EXPOSE 8888
WORKDIR /
STOPSIGNAL SIGTERM
CMD [ "bash" ]

```

Listing 22: The listing shows the two-staged Dockerfile of the CernVM 5 project. The first stage includes instructions to prepare a build environment needed to construct and post-build process the root file system of CernVM 5 images. The second stage begins after the so-called **builder** stage finished building and processing the root file system in the **build** directory. Its contents are copied to the yet empty **scratch** image.

## References

- [1] The Worldwide LHC Computing Grid. <https://wlcg.web.cern.ch/using-wlcg/monitoring-visualisation>. [Online; accessed 25-June-2022].
- [2] Miron Livny, Rajesh Raman, Todd Tannenbaum, and Jim Basney. Mechanisms for high throughput computing. *SPEEDUP*, 11(1), 1997.
- [3] Ian Bird et al. LHC computing grid: Technical design report. Technical Report LCG-TDR-001, CERN, 2005.
- [4] Dan Kusnetzky. *Virtualization*. O'Reilly Media, Inc., 2011.
- [5] The VirtualBox Project. <https://www.virtualbox.org/>. [Online; accessed 21-June-2022].
- [6] The Kernel-based Virtual Machine Project. [https://www.linux-kvm.org/page/Main\\_Page](https://www.linux-kvm.org/page/Main_Page). [Online; accessed 26-June-2022].
- [7] The Linux Kernel Project. <https://kernel.org/>. [Online; accessed 26-June-2022].
- [8] KVM Guest Support Documentation. [https://www.linux-kvm.org/page/Guest\\_Support\\_Status](https://www.linux-kvm.org/page/Guest_Support_Status). [Online; accessed 26-June-2022].
- [9] The QEMU Project Page. <https://www.qemu.org/>. [Online; accessed 21-July-2022].
- [10] The OpenStack Project. <https://www.openstack.org/>. [Online; accessed 27-June-2022].
- [11] The OverlayFS Project. <https://www.kernel.org/doc/html/latest/filesystems/overlayfs.html>. [Online; accessed 5-July-2022].
- [12] The Linux Namespaces Documentation. <https://man7.org/linux/man-pages/man7/namespaces.7.html>. [Online; accessed 28-June-2022].
- [13] The Linux cgroups Documentation. <https://www.man7.org/linux/man-pages/man7/cgroups.7.html>. [Online; accessed 28-June-2022].
- [14] Arne Wiebalck and Sean Crosby Tim Bell Ulrich Schwickerath. Optimisations of the compute resources in the cern cloud service. <https://indico.cern.ch/event/384358/contributions/909247/>, 2015.
- [15] Docker Reference Documentation. <https://docs.docker.com/reference/>. [Online; accessed 17-June-2022].
- [16] The Docker Hub Container Image Registry. <https://hub.docker.com/>. [Online; accessed 17-June-2022].
- [17] The Podman Project. <https://podman.io/>. [Online; accessed 21-June-2022].
- [18] Gregory M et. al. Kurtzer. Singularity 2.5.2 - Linux application and environment containers for science, July 2018.
- [19] Dockerfile Documentation. <https://docs.docker.com/engine/reference/builder/>. [Online; accessed 29-June-2022].
- [20] Buildah Container Build Tool. <https://buildah.io/>. [Online; accessed 21-June-2022].

- [21] The CernVM-File System Project. <https://cvmfs.readthedocs.io>. [Online; accessed 29-June-2022].
- [22] Jakob Blomer, Predrag Buncic, René Meusel, Gerardo Ganis, Igor Sfiligoi, and Douglas Thain. The evolution of global scale filesystems for scientific software distribution. *Computing in Science and Engineering*, 17(6):61–71, 2015.
- [23] Cervantes Villanueva, Javier, Ganis, Gerardo, Konstantinov, Dmitri, Latyshev, Grigori, Mato Vila, Pere, Mendez Lorenzo, Patricia, Pacholek, Rafal, and Razumov, Ivan. Building, testing and distributing common software for the lhc experiments. *EPJ Web Conf.*, 214:05020, 2019.
- [24] The Parrot Toolkit. <http://ccl.cse.nd.edu/software/parrot/>. [Online; accessed 5-July-2022].
- [25] The FUSE Framework Project. <https://www.kernel.org/doc/html/latest/filesystems/fuse.html>. [Online; accessed 5-July-2022].
- [26] Predrag Buncic, Jakob Blomer, Pere Mato, Carlos Aguado Sanchez, Leandro Franco, and Steffen Klemer. Cernvm - a virtual appliance for lhc applications. In *Proceedings of the XII International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT08)*, 2008.
- [27] Jakob Blomer, Dario Berzano, Predrag Buncic, Ioannis Charalampidis, Gerardo Ganis, Georgios Lestaris, René Meusel, and V. Nicolaou. Micro-cernvm: Slashing the cost of building and deploying virtual machines. *CoRR*, abs/1311.2426, 2013.
- [28] The Executable and Linkable Format (ELF) Specification. <https://refspecs.linuxfoundation.org/elf/elf.pdf>. [Online; accessed 18-July-2022].
- [29] The GNU C Library (glibc). <https://www.gnu.org/software/libc/documentation.html>. [Online; accessed 18-July-2022].
- [30] The mold Linker Project on GitHub. <https://github.com/rui314/mold>. [Online; accessed 18-July-2022].
- [31] The LLVM lld Linker Project Documentation. <https://lld.llvm.org/>. [Online; accessed 18-July-2022].
- [32] The GNU ld Linker Manual Page. <https://man7.org/linux/man-pages/man8/ld.so.8.html>. [Online; accessed 18-July-2022].
- [33] The EC2 User’s Documentation. <https://docs.aws.amazon.com/ec2/index.html>. [Online; accessed 19-July-2022].
- [34] N. Garelli, M. Morar, and W. Vandelli. Balancing the resources of the high level trigger farm of the atlas experiment. *Journal of Physics Conference Series*, 396:2022–, 12 2012.
- [35] The HEP-OSlibs Meta-package. [https://gitlab.cern.ch/linuxsupport/rpms/HEP\\_OSlibs](https://gitlab.cern.ch/linuxsupport/rpms/HEP_OSlibs). [Online; accessed 29-June-2022].
- [36] The RPM Project. <http://rpm.org/>. [Online; accessed 11-July-2022].
- [37] The spec File Format Documentation. <https://rpm-software-management.github.io/rpm/manual/spec.html>. [Online; accessed 11-July-2022].



- [38] The GNU Bash Project Page. <https://www.gnu.org/software/bash/>. [Online; accessed 24-June-2022].
- [39] The dnf Package Manager Documentation. [https://dnf.readthedocs.io/en/latest/command\\_ref.html](https://dnf.readthedocs.io/en/latest/command_ref.html). [Online; accessed 18-July-2022].
- [40] Docker Mutli-Stage Builds. <https://docs.docker.com/develop/develop-images/multistage-build/>. [Online; accessed 9-July-2022].
- [41] The patchELF Project on GitHub. <https://github.com/NixOS/patchelf>. [Online; accessed 13-July-2022].
- [42] The X.Org Project. <https://x.org/>. [Online; accessed 16-June-2022].
- [43] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, and Carol Willing. Jupyter notebooks – a publishing format for reproducible computational workflows. In F. Loizides and B. Schmidt, editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87 – 90. IOS Press, 2016.
- [44] Fons Rademakers, Philippe Canal, Axel Naumann, Olivier Couet, Lorenzo Moneta, Vassil Vassilev, Sergey Linev, Danilo Piparo, Gerardo GANIS, Bertrand Bellenot, Enrico Guiraud, Guilherme Amadio, wverkerke, Pere Mato, TimurP, Matevž Tadel, wlv, Enric Tejedor, Jakob Blomer, Andrei Gheata, Stephan Hageboeck, Stefan Roiser, marsupial, Stefan Wunsch, Oksana Shadura, Anirudha Bose, CristinaCristescu, Xavier Valls, Raphael Isemann, and Kim Albertsson. root-project/root: v6.20/06, June 2020.
- [45] The Future Circular Collider (FCC). <https://home.cern/science/accelerators/future-circular-collider>. [Online; accessed 4-July-2022].
- [46] The FCC Starterkit Tutorial. <https://hep-fcc.github.io/fcc-tutorials/index.html>. [Online; accessed 4-July-2022].
- [47] The Jenkins Project. <https://www.jenkins.io/>. [Online; accessed 4-July-2022].
- [48] The CernVM 5 Project on the CERN Harbor Registry. <https://registry.cern.ch/harbor/projects/1962/repositories>. [Online; accessed 12-July-2022].
- [49] The unpacked.cern.ch Repository Documentation. <https://cvmfs.readthedocs.io/en/2.8/cpt-containers.html>. [Online; accessed 12-July-2022].
- [50] The CernVM 5 yum Repository for RPMs. <http://ecsft.cern.ch/dist/cernvm/five>. [Online; accessed 12-July-2022].
- [51] CernVM 5 Build Products on the EP-SFT Webspace. <http://ecsft.cern.ch/dist/cernvm/five/>. [Online; accessed 13-July-2022].
- [52] Michael Kerrisk. *The Linux Programming Interface*. No Starch Press, 2010.
- [53] The mount Linux Manual Page. <https://man7.org/linux/man-pages/man2/mount.2.html>. [Online; accessed 1-June-2022].
- [54] The Spack Package Manager Project. <https://spack.readthedocs.io/en/latest/>. [Online; accessed 18-July-2022].

- [55] The systemd Service Manager. <https://systemd.io/>. [Online; accessed 14-July-2022].
- [56] The OpenSSH Project. <https://www.openssh.com/>. [Online; accessed 19-July-2022].
- [57] The dracut Manual Page. <https://www.man7.org/linux/man-pages/man8/dracut.8.html>. [Online; accessed 19-July-2022].
- [58] The E2fsprogs Project. <http://e2fsprogs.sourceforge.net/>. [Online; accessed 19-July-2022].
- [59] The cloud-init Project. <https://cloud-init.io/>. [Online; accessed 19-July-2022].
- [60] Werner Almesberger and Hans Lermen. The Linux Kernel User's Documentation on Initial Ramdisks. <https://www.kernel.org/doc/html/latest/admin-guide/initrd.html>. [Online; accessed 19-July-2022].
- [61] The loop Device Linux Manual Page. <https://man7.org/linux/man-pages/man4/loop.4.html>. [Online; accessed 19-July-2022].
- [62] The Linux Kernel User's Documentation on the Mapper Device. <https://www.kernel.org/doc/html/latest/admin-guide/device-mapper/index.html>. [Online; accessed 19-July-2022].
- [63] The falldate Linux Manual Page. <https://www.man7.org/linux/man-pages/man2/fallocate.2.html>. [Online; accessed 19-July-2022].
- [64] The fdisk Linux Manual Page. <https://www.man7.org/linux/man-pages/man8/fdisk.8.html>. [Online; accessed 19-July-2022].
- [65] The kpartx Linux Manual Page. <https://linux.die.net/man/8/kpartx>. [Online; accessed 19-July-2022].
- [66] The ext4 File System Project Page. [https://ext4.wiki.kernel.org/index.php/Main\\_Page](https://ext4.wiki.kernel.org/index.php/Main_Page). [Online; accessed 19-July-2022].
- [67] The extlinux Bootloader Linux Manual Page. <https://linux.die.net/man/1/extlinux>. [Online; accessed 19-July-2022].
- [68] The syslinux Documentation on Master Boot Records. <https://wiki.syslinux.org/wiki/index.php?title=Mbr>. [Online; accessed 19-July-2022].
- [69] The blkid Linux Manual Page. <https://www.man7.org/linux/man-pages/man8/blkid.8.html>. [Online; accessed 19-July-2022].
- [70] The dd Linux Manual Page. <https://www.man7.org/linux/man-pages/man1/dd.1.html>. [Online; accessed 19-July-2022].
- [71] The VDI Disk Image Format. <https://docs.oracle.com/en/virtualization/virtualbox/6.0/user/vdidetails.html>. [Online; accessed 19-July-2022].
- [72] The QEMU QCOW2 Format Documentation. <https://www.qemu.org/docs/master/system/images.html>. [Online; accessed 19-July-2022].
- [73] The WHIZARD Event Generator Project. <https://whizard.hepforge.org/>. [Online; accessed 20-July-2022].

- [74] Gerardo Ganis, Clément Helsen, and Valentin Völkl. Key4hep, a framework for future HEP experiments and its use in FCC. *Eur. Phys. J. Plus*, 137:149. 8 p, Nov 2021.
- [75] The HEP Software Foundation. <https://hepsoftwarefoundation.org/>. [Online; accessed 20-July-2022].
- [76] CERN’s Service for Web-based ANalysis(SWAN) Project Page. [https://swan.docs.cern.ch/intro/what\\_is/](https://swan.docs.cern.ch/intro/what_is/). [Online; accessed 24-July-2022].