Micro-CernVM: Slashing the Cost of Building and Deploying Virtual Machines

J Blomer, D Berzano, P Buncic, I Charalampidis, G Ganis, G Lestaris, R Meusel, V Nicolaou

CERN, CH-1211 Genève 23, Switzerland

E-mail: jblomer@cern.ch

Abstract. The traditional virtual machine building and and deployment process is centered around the virtual machine hard disk image. The packages comprising the VM operating system are carefully selected, hard disk images are built for a variety of different hypervisors, and images have to be distributed and decompressed in order to instantiate a virtual machine. Within the HEP community, the CernVM File System has been established in order to decouple the distribution from the experiment software from the building and distribution of the VM hard disk images.

We show how to get rid of such pre-built hard disk image altogether. Due to the high requirements on POSIX compliance imposed by HEP application software, CernVM-FS can also be used to host and boot a Linux operating system. This allows the use of a tiny bootable CD image that comprises only a Linux kernel while the rest of the operating system is provided on demand by CernVM-FS. This approach speeds up the initial instantiation time and reduces virtual machine image sizes by an order of magnitude. Furthermore, security updates can be distributed instantaneously through CernVM-FS. By leveraging the fact that CernVM-FS is a versioning file system, a historic analysis environment can be easily re-spawned by selecting the corresponding CernVM-FS file system snapshot.

1. Introduction

Virtual machines provide a uniform and portable environment for high energy physics applications [1]. The same virtual machine can be used for both development and for deployment on cloud infrastructures so that very same operating system libraries that are used to develop physics algorithms are also used to execute them. As such, virtual machines provide a way to escape the so called "dependency hell", i. e. the problem that applications might not work or behave differently when using slightly different (versions of) system libraries. Furthermore, virtual machines remove the need to port applications to different operating systems. This is one of the key benefits when it comes to working with volunteers that provide free CPU cycles on a variety of platforms (Windows, Linux, Mac) [2]. Properly archived, virtual machines are also an important ingredient for the long-term preservation of experiment data because they contain the complex historic software stack necessary to interpret data files [3].

The hard disk resembling a virtual machine is distributed in the form of a hard disk image file. There are two contradicting approaches to creating virtual machine images. The first approach creates a hard disk image from a standard installation of an operating system. Such images fit a large base of different applications and use cases, such as web servers, machines for interactive

Table 1. Comparison of the size of the μCernVM kernel and the Scientific Linux 6 kernel

	μCernVM	**Scientific Linux 6**
Number of modules	100	2000
Size (incl. modules)	8 MB	120 MB

login, or graphical workstations. The image size is rather large, typically a few gigabytes, and due to the many packages contained in an image, images quickly become outdated.

The second approach creates a *virtual appliance*. The virtual appliance comprises a "just enough operating system", that is a stripped down version of an operating system tailored to only one or few applications and use cases. An image size of a virtual appliance can be as small as a few hundred megabytes.

As we have seen maintaining the CernVM virtual appliance[1], even for a minimalistic image it can take hours to prepare package repositories, build and compress virtual machine images in various formats, and to test them. This delay between making a change and verifying the result of the change restricts the speed at which the virtual appliance can be developed. On the user's end, despite the fact that sophisticated tools have been developed to manage virtual machine images [4–8], the image distribution problem is best mitigated by small images.

With all its libraries, tools, and configuration data, an operating system is a complex software stack. The problem of distributing complex and frequently changing software stacks to virtual machines were previously faced for experiment applications. Experiment applications are now widely distributed by the CernVM File System (CernVM-FS) [9], a caching, read-only file system that downloads software bits and pieces on demand from an HTTP content delivery network. Here, we bring the use of CernVM-FS to the next level and use it to distribute as well the operating system to a virtual machine. The result is a tiny virtual machine image only consisting of a Linux kernel and the CernVM-FS client. The actual operating system is then booted from CernVM-FS. Instead of a "just enough operating system", this approach leads to an *operating system on demand*.

The rest of the paper is structured as follows. Section 2 describes the components of a μCernVM system. Section 3 describes the virtual machine's root file system stack and the connection between μCernVM and the operating system on CernVM-FS. Section 4 describes the maintenance of an operating system on CernVM-FS; in particular we focus on exploiting the internal versioning of CernVM-FS that allows the very same virtual machine image to instantiate any operating system ever released on CernVM-FS. Section 5 describes how μCernVM based systems are updated. Section 6 compares μCernVM to similar approaches.

2. Building blocks

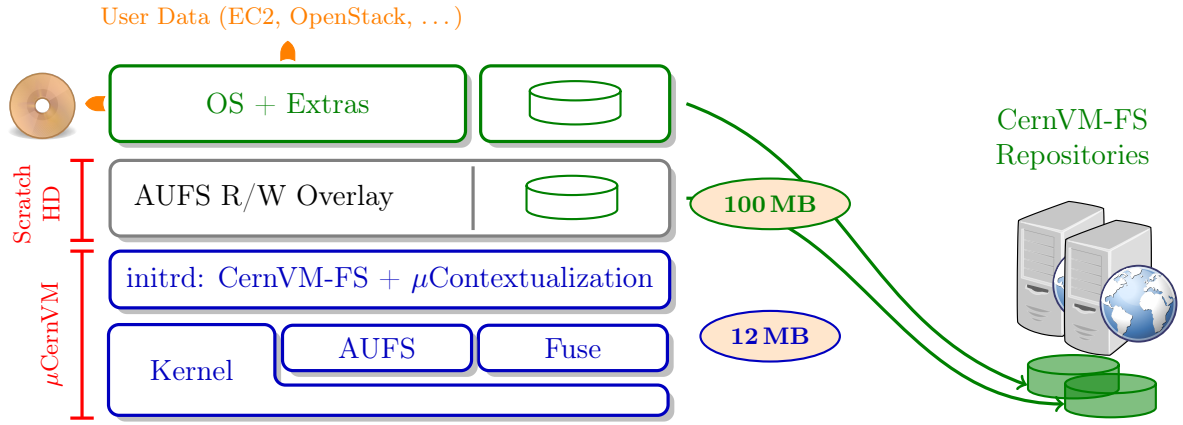
Figure 1 shows the key components of the μCernVM image. The image consists of a Linux kernel and an init ramdisk that includes the CernVM-FS client. The Linux kernel is "virtualization friendly". It is slim as it only requires device drivers for the handful of hypervisors around. Table 1 compares the size of the μCernVM kernel with an Scientific Linux 6 kernel. On the other hand, the μCernVM kernel has the latest para-virtualized drivers which are not necessarily found in Scientific Linux or other distribution kernels.

Furthermore, the μCernVM kernel contains the *union file system* [10] AUFS[2]. The union file

[1] http://cernvm.cern.ch

[2] http://aufs.sourceforge.net

Figure 1. A μCernVM based virtual machine is twofold. The μCernVM image contains a Linux kernel, the AUFS union file system, and a CernVM-FS client. The CernVM-FS client connects to a special repository containing the actual operating system. The two CernVM-FS repositories contain the operating system and the experiment software.



system is needed because CernVM-FS is a read-only file system. Local changes, such as of files in /etc or /var, cannot be written to CernVM-FS. Instead, the union file system transparently redirects such changes to a locally attached scratch hard disk. The local hard disk is also used to store the CernVM-FS cache. Note that the scratch hard disk does *not* need to be distributed. It can be created instantaneously when instantiating the virtual machine as an empty, sparse file.

The init ramdisk contains the CernVM-FS client and a steering script. The purpose of the steering script is to create the virtual machine's root file system stack that is constructed by unifying the CernVM-FS mount point with the writable scratch space. To do so, the steering script can process contextualization information (sometimes called "user data") from various sources, such as OpenStack, OpenNebula, or Amazon EC2. Based on the contextualization information, the CernVM-FS repository and the repository version is selected.

The amount of data that needs to be loaded in order to boot the virtual machine is very little. The image itself sums up to some 12 MB. In order to boot Scientific Linux 6 from CernVM-FS, the CernVM-FS client downloads additional 100 MB. The CernVM-FS infrastructure used to distribute experiment software can be reused. In comparison, the (already small) CernVM 2.6 virtual appliance sums up to 300 MB to 400 MB that needs to be fully loaded and decompressed upfront before the boot process can start. As a result, booting a μCernVM virtual machine starts practically instantaneously so that it can be, for instance, integrated with a web site that starts a virtual machine on the click of a button.[3]

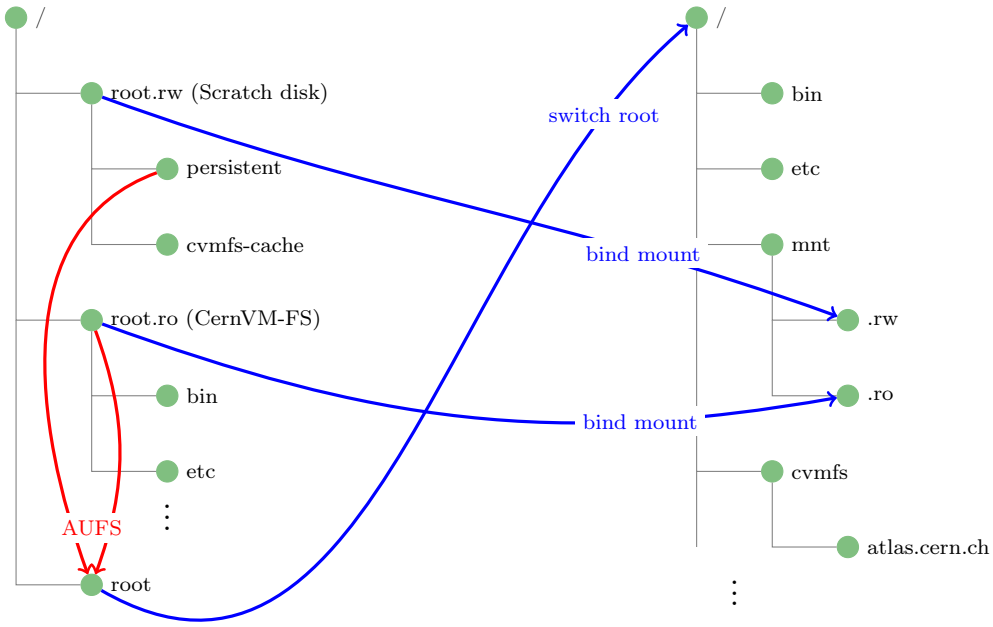
3. The μCernVM root file system stack

At the beginning of the Linux boot process, in the so called *early user space*, the Linux kernel uses a root file system in memory provided by the init ramdisk. The purpose of the early user space is to load the necessary storage device drivers to access the actual root file system. Once the actual root file system is available, the system switches its root file system to the new root mount point after which the previous root file system becomes useless and is removed from memory.

Figure 2 shows the transformation of the file system tree in the early user space in μCernVM.

[3] An example of such a web site is a volunteer computing project by the CERN theory group: http://crowdcrafting.org/app/cernvm

Figure 2. Transformation of the root file system stack. Left hand side: the root file system of the early user space. The /root directory is an AUFS mount point. Right hand side: the final root file system stack.



First, the scratch hard disk is mounted on /root.rw. μCernVM grabs the first empty hard disk or partition attached to the virtual machine, or remaining free space on the boot hard disk. It automatically partitions, formats, and labels the scratch space. Due to the file system label, μCernVM finds an already prepared scratch space on next boot. The scratch space is used as a persistent writable overlay for local changes to the root file system and as a cache for the CernVM-FS client that loads the operating system. Secondly, a CernVM-FS repository containing a template operating system installation is mounted on /root.ro. The file system unification of the CernVM-FS mount point and the directory containing the persistent overlay is mounted on /root. Finally, the /root directory is installed as new root file system. The /root.ro and /root.rw directories are projected into the final root file system via *bind mounts*. The rest of the init ramdisk is removed from memory.

Although μCernVM and the operating system loaded by μCernVM are independent from each other, there are a few possible points of interaction between the two components. These interaction points are implemented through special files in the root directory. The following files are created or recognized by μCernVM:

/.cvmfs_pids In this file, μCernVM stores the process IDs (PIDs) of the CernVM-FS process used to mount the operating system repository. The PIDs help operating system scripts to avoid accidental killing of these vital processes.

/.ucernvm_pinfiles This file is provided by the operating system repository and contains a list of files that should be pinned in the cache. The idea is to always keep a minimal set of files available in the cache that allows for recovery from a broken network connection.

/.ucernvm_bootstrap This is a shell script provided by the operating system repository. It is sourced just before the root file system is switched and allows for custom actions.

Special care has to be taken in the shutdown script of the operating system. Typically, the shutdown script does not expect an AUFS file system stack as root file system. Therefore, the script needs to be modified to first remount the underlying read-write layer (/mnt/.rw) in read-only mode before remounting the root file system itself in read-only mode. Only then the machine can be halted without risking any file system corruption.

4. Versioned operating system on CernVM-FS

An easy way to provide an operating system template installation on CernVM-FS is to use the operating system's package manager to install the desired packages in the CernVM-FS repository area.[4] New and updated packages can then be installed incrementally by the package manager on top of the existing installation. While being very fast, this results, however, in an ever-changing operating system directory tree. It can quickly become difficult to trace back which update introduced or fixed a certain problem and which state of the operating system directory tree is mounted by virtual machines.

We have tackled this problem for *CernVM 3*, a Scientific Linux 6 based operating system on CernVM-FS. CernVM 3 benefits from versioning on three levels that ensure traceability and a well-defined, predictable virtual machine (see Figure 3).

(i) The *cernvm-system* meta package provides the notion of a well defined operating system. The meta package has no payload but contains only package dependencies. In case of the cernvm-system package, the dependencies are fully versioned and no dependencies are missing, i. e. there is no degree of freedom for the dependency resolution of this package.

(ii) Three different CernVM-FS repositories are maintained, *development*, *testing*, and *production*, that reflect different levels of maturity of the cernvm-system meta package. Furthermore, separation of repositories allows for injection of a security hotfix in the production repository while continue to develop in the other repositories.

(iii) To avoid silent and unwanted updates of the virtual machine operating system, μCernVM exploits the fact that CernVM-FS is a versioning file system. CernVM-FS creates a snapshot whenever changes are published. These snapshots remain available and they can be named. The concept and the implementation is similar to a *tag* in the git versioning system. CernVM-FS clients can, through mount options, select a particular snapshot on mount.[5]
On first boot, μCernVM mounts the newest available snapshot of the given repository and it will stay on this snapshot during reboots unless instructed otherwise. Snapshots are named after the version number of the cernvm-system meta package. Contextualization can be used to select another snapshot to mount, letting μCernVM go back in time and instantiate a historic data processing environment.

The three levels of versioning allow the μCernVM image to always instantiate the newest available stable operating system version. At the same time, the very same image provides instant access to every other operating system version ever released in the given repository.

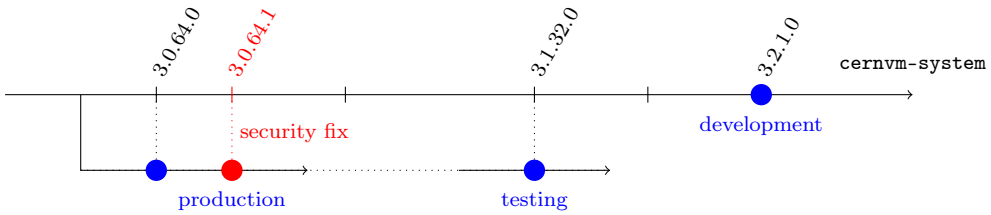
5. System updates

While some virtual machines are short-lived and only instantiated for a particular computing job, others can run for many weeks or months. A typical example of a long-lived virtual machine

[4] Like `yum -installroot /cvmfs/sl6.cern.ch install glibc emacs ...`

[5] This feature is so far untapped by experiments' usage of CernVM-FS. Instead experiments use different directories for different software releases, reflecting the fact that many versions of the experiment software need to be available on the same worker node at the same time.

Figure 3. CernVM 3 versioning scheme. Every released cernvm-system meta package is resembled as a versioned snapshot in CernVM-FS. The named branches *development*, *testing*, and *production* provide entry points to versioned snapshots at different levels of maturity.



is an interactive virtual machine on an end user's laptop. It is desirable to provide a long-lived virtual machine with bug fixes and security updates without the need to reconfigure the virtual machine or to replace the image file. This section describes how to update both kernel and init ramdisk on μCernVM and the operating system on CernVM-FS.

5.1. Updates of μCernVM

The μCernVM image can be used as a read-only CD-ROM image. Hence writing updated data to the image is not possible. However, updated versions of the Linux kernel and the init ramdisk can be dropped into a predefined location on the scratch hard disk. Upon boot, after mounting the scratch hard disk, μCernVM uses the *kexec* facility [11] in order to reboot on the fly into the updated kernel and init ramdisk.

5.2. Updates of an operating system on CernVM-FS

Once booted, a μCernVM based virtual machine can be kept up to date by standard means of the package manager provided by the operating system. This approach, however, accumulates more and more local changes on the writable overlay stored on the scratch hard disk. Over time, more and more disk space is spent on the writable overlay and the local operating system diverges from the one provided on CernVM-FS.

Instead it is preferable to pick up updates from CernVM-FS. To this end, μCernVM can be instructed to "unpin" the currently mounted CernVM-FS snapshot and to mount the newest available snapshot on next boot. Changing the snapshot can result in conflicts with the local changes on the writable overlay. For instance, a user might have replaced /usr/bin/gcc while at the same time the updated snapshot provides a newer version of /usr/bin/gcc as well. The conflict resolution in μCernVM is guided by the widely used file system hierarchy standard [12]: μCernVM resolves conflicts in /etc and /var by keeping the locally modified versions, whereas in all other directories the version from CernVM-FS has precedence. Furthermore, packages installed by the user need to be reinserted into the package database of the updated snapshot. Finally, the system account databases (/etc/passwd, /etc/group, ...) are merged record by record as follows

- Local passwords have precedence
- Group membership is merged
- Conflicting user ids or group ids from the new snapshot are mapped to the locally defined ids. If necessary, user ids or group ids from the new snapshot are remapped to previously unused ids. Remapping of user ids and group ids is supported by CernVM-FS, in the same way NFS clients support id mappings. [13]

This approach to updates assumes that operating system components comply with common standards (for instance, that components do not store configuration data in /usr/bin). While

first observations with CernVM 3 are positive, it requires more experience to verify the approach in practice.

6. Related Work

With live CDs and network booted Linux systems, μCernVM shares the fact that the root file system is read-only and local changes are written to an overlay area by means of a union file system. Live CDs need to be loaded as a whole before the boot process can start and the writable overlay is in memory. Typical network boot setups rely on NFS, which cannot be used for wide-area setups. Also, booting from the network relies on a network boot protocol (e. g. PXE) instead of using a minimal virtual machine image.

An idea very similar to μCernVM was implemented as *HTTP-FUSE Xenoppix* [14]. This system consists of a kernel and an init ramdisk that downloads the actual root file system on demand via HTTP. Important features of CernVM-FS are not implemented though, such as versioning and digitally signed network traffic. Unlike μCernVM, HTTP-FUSE Xenoppix runs only on the Xen hypervisor. There is no discussion of system updates in HTTP-FUSE Xenoppix.

The *vagrant* tool set greatly facilitates building images for virtual appliances, especially "virtual development environments" [15]. Despite the convenient user interface, hard disk images containing an full operating system must still be distributed.

Similarly, *CoreOS*[6] simplifies massive server deployment. It consists of a tiny Linux system just enough to run lightweight virtual machines ("containers") but it is not a virtual machine by itself. Instead of loading files on demand, CoreOS manages entire containers.

7. Conclusion

We have presented μCernVM, a 12 MB virtual machine image that loads the actual operating system files on demand through CernVM-FS. Updates to the virtual machine do not require changing the image but they are simply distributed through changes in the CernVM-FS repository. Moreover, the very same tiny image can boot any version of the operating system ever released on CernVM-FS, which facilitates the long-term preservation of data processing environments.

We have verified our approach by creating CernVM 3, the SL6 based successor of CernVM 2, on top of μCernVM.[7] CernVM 3 runs on VMware, VirtualBox, Xen, and KVM hypervisors as well as in OpenStack, OpenNebula, and Amazon EC2 clouds. CernVM 3 can build itself and it runs experiment software from the four LHC experiments. A systematic validation of experiment software remains to be done.

References

[1] Buncic P, Sanchez C A, Blomer J, Harutyunyan A and Mudrinic M 2011 *The European Physical Journal Plus* **126**

[2] González D L, Grey F, Blomer J, Buncic P, Harutyunyan A, Marquina M, Segal B and Skands P 2012 *PoS(ISGC 2012)036*

[3] The ICHFA DPHEP International Study Group 2012 Status report of the dphep study group: Towards a global effort for sustainable data preservation in high energy physics Tech. Rep. FERMILAB-PUB-12-878-PPD Fermilab

[4] VMware 2007 VMware virtual machine file system: Technical overview and best practices Tech. Rep. WP-022-PRD-01-01 VMware

[5] McLoughlin M 2008 The QCOW2 image format https://people.gnome.org/~markmc/qcow-image-format.html

[6] Morita K 2010 Sheepdog: Distributed storage system for QEMU/KVM talk at the LinuxCon Japan 2010

[7] Wartel R, Cass T, Moreira B, Roche E, Guijarro M, Goasguen S and Schwickerath U 2010 *Proc. of the 2nd IEEE Int. Conf. on Cloud Computing Technology and Science (CloudCom'10)* pp 112–117

[6] http://coreos.com

[7] The CernVM 3 build system is available under https://github.com/cernvm.

[8] SwiftStack 2012 The openstack object storage system Tech. rep. SwiftStack
[9] Blomer J, Aguado-Sanchez C, Buncic P and Harutyunyan A 2011 *Journal of Physics: Conference Series* **331**
[10] Wright C P, Dave J, Gupta P, Krishnan H, Zadok E and Zubair M N 2004 Versatility and unix semantics in a fan-out unification file system Tech. Rep. FSL-04-01b Stony Brook University
[11] Pfiffer A 2003 Reducing system reboot time with kexec Tech. rep. Open Source Development Labs
[12] Russell R, Quinlan D and Yeoh C 2004 Filesystem hierarchy standard Tech. rep. freestandards.org
[13] Callaghan B, Pawlowski B and Staubach P 1995 NFS Version 3 Protocol Specification RFC 1813 Internet Engineering Task Force URL http://www.rfc-editor.org/rfc/rfc1813.txt
[14] Suzaki K, Yagi T, Iijima K, Kitagawa K and Tashiro S 2006 *Proc. of the 2006 Linux Symposium* vol 2 pp 379–392
[15] Palat J 2012 *Linux Journal*