Virtualization [1][2]

IT Systems, Summer Term 2026

Dr. Matthes Elstermann

The topics of virtualization and containerization are addressed in two presentations, of which this is the first one. It focuses on virtualization.

1 Introduction

Let us look at essential questions of our topic, followed by a short overview.

1.1 Core Questions

- What do virtualization and containerization mean?
- How to deploy potentially complex software in a reproducible fashion?

These two presentations address the following questions:
What do virtualization and containerization mean?
How to deploy potentially complex software in a reproducible fashion?

1.2 Learning Objectives

- Explain definitions of virtual machine and virtual machine monitor
- Explain and contrast virtualization and containerization
  - Including isolation
  - Including layering
- Use Docker for simple tasks
  - E.g., start web server with static files
  - Interpret simple Docker files

Take some time to think about the learning objectives specified here.

1.3 Overview (1/2)

- Virtualization provides **(virtual) hardware interface**

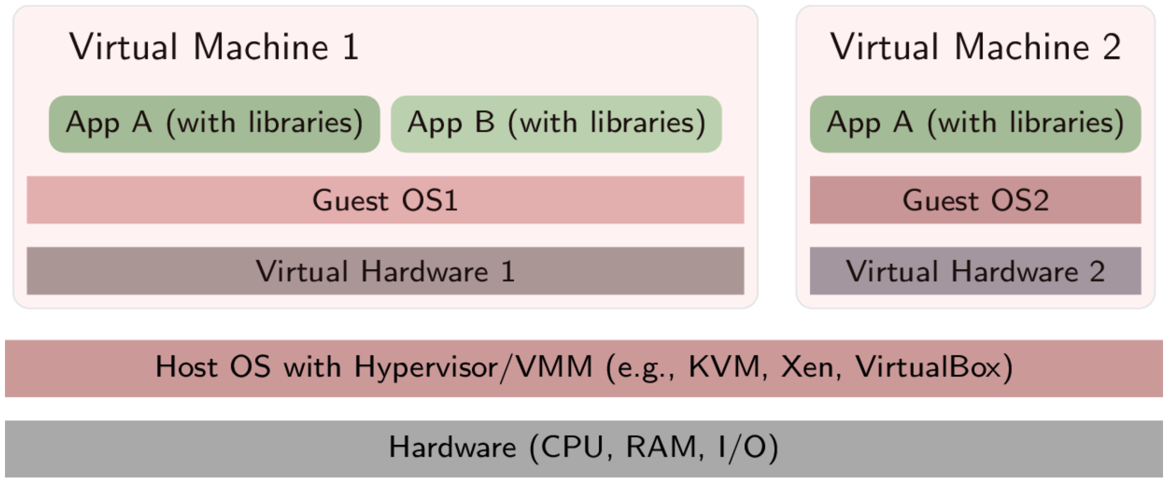

Figure 1: Layering with virtualization

[1]This PDF document is an inferior version of an OER in HTML format; free/libre Org mode source repository.
[2]Material created by Jens Lechtenbörger; see end of document for license information.

- Interface implemented by Hypervisor/VMM
  - VMM runs on (usual) host OS, manages real hardware

This and the subsequent slide are intended as quick overview for virtualization and containerization. Terms used here as well as the layered figure are revisited later on.

Different variants of virtualization exist, and a typical one is as follows: We consider computer hardware at the bottom, with an OS as usual. Now, the OS comes with virtualization software in the form of a so-called hypervisor or virtual machine monitor, VMM for short, which implements functionality to support virtualization. It does so by providing a virtual hardware interface, which can present the illusion of having arbitrary hardware available, to be used in higher layers. As the OS in combination with the VMM serves as “host” for virtualized “guest” OSs, it is also called host OS.

For efficiency reasons, modern CPUs come with virtualization features that can be used by the VMM. However, details are not important for us. See Wikipedia if you are interested.

- Virtual hardware can have **arbitrary** features
  - Largely independent of real hardware, say, ten network cards

Similarly to how virtual memory can have an arbitrary size, independently of the size of main memory, virtual hardware can have arbitrary features. For example, it is possible to experiment with a cluster of “virtual computers” inside a single machine that is not even connected to the Internet.

  - On top of virtual hardware, install operating systems (guests) and other software to create virtual machines (VMs)
  - **Share resources** of powerful server machine among several VMs
    - E.g., your “own” server as VM in a project seminar
  - Fire up lots of **identical** VMs for compute-intensive tasks with cloud computing

The OS running on the real hardware is called host OS, as it serves as host for the creation of virtual machines, VMs for short. Each virtual machine may “see” a different version of virtual hardware, provided by the VMM. Also, each VM can run a different OS, which is then called guest OS.

Thus, with virtualization, we can run several OSs with their applications at the same time on a single physical computer.

Clearly, all VMs and their guest OSs share resources of a single physical machine, which may be a powerful server in a datacenter or a laptop.

As prominent use case of virtualization, lots of identical VMs can be started for compute intensive tasks, e.g., with cloud computing.

1.4 Overview (2/2)

- Containerization (e.g., with Docker) as lightweight variant of virtualization

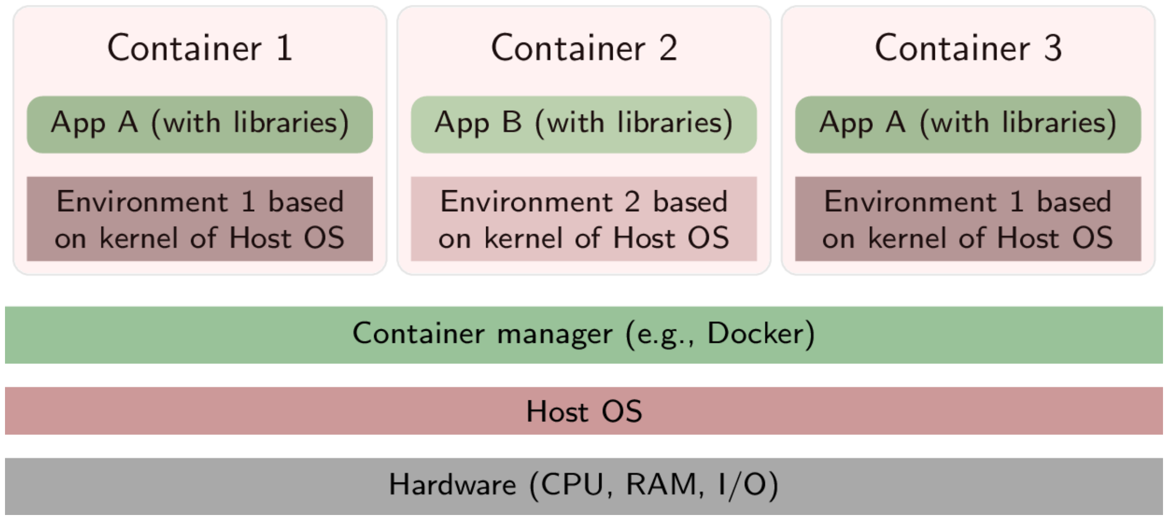
Container 1
App A (with libraries)
Environment 1 based on kernel of Host OS
Container 2
App B (with libraries)
Environment 2 based on kernel of Host OS
Container 3
App A (with libraries)
Environment 1 based on kernel of Host OS
Container manager (e.g., Docker)
Host OS
Hardware (CPU, RAM, I/O)

Figure 2: Layering with containerization

- Containerization provides **OS interface**
- No virtual hardware, but shared OS kernel

Containerization can be perceived as lightweight variant of virtualization. While virtualization provides a hardware interface, containerization provides an OS interface. In the layered architecture here you see that we still have a host OS running on top of real hardware. This time, however, no virtual hardware exists. Instead, a container manager, e.g., Docker, runs as process (or set of processes) in the host OS and allows creating containers, which are execution environments for software. Importantly, all containers share the same kernel of the host OS, but each container may be restricted differently regarding its access of underlying hardware.

- Use containers to execute software (versions) in controlled way
  - Think of larger application that uses external libraries
  - Libraries evolve, may introduce incompatible changes over time
    - Specific version of application depends on specific versions of libraries
    - Container bundles “correct” versions

  Containers are used to distribute and execute software in controlled ways. E.g., software may depend on external libraries in specific versions, and a container can bundle, or tie together, all the necessary pieces in a reproducible fashion.

Agenda

- Part 1
  - Introduction
  - History and Variants
  - Virtualization
- Part 2
  - Containerization
  - Docker
  - Conclusions

Fundamentals of virtualization are presented in two parts, of which the first one starts with a brief history and different variants of virtualization, followed by a precise definition for classical hardware virtualization.

In part 2, we look at containerization as lightweight virtualization. In particular, we take first steps with Docker.

2 History and Variants

Let us look at the history of virtualization and popular meanings of that term.

2.1 History (1/2)

- Virtualization is an old concept
  - IBM mainframes, 1960s
  - Frequently cited survey article: (Goldberg 1974)
  - Original motivation
    - Resources of **expensive** mainframes better utilized with multiple VMs
    - Ability to run different OS versions in parallel, **backwards compatibility**, **incremental updates**
- 1980s, 1990s
  - Multitasking OSs on relatively **cheap** hardware
    - Early PC hardware did not offer virtualization support
    - Little use of virtualization

Hardware virtualization is an old concept going back to mainframe computers. In fact, in 1974, a frequently cited overview article about virtualization was published, which introduced important terminology that is still relevant today.

The original motivation for virtualization was to utilize resources of expensive mainframes better with multiple VMs (than with just one OS). Also, it appeared attractive to run different OS versions in parallel, in particular for backwards compatibility. Also, different VMs may be updated and tested separately from each other, and updates can be rolled out incrementally to a fleet of machines in production.

Then, with the rise of multitasking OSs on cheap hardware, such as PCs, virtualization was not used much. Notably, early PC hardware lacked virtualization support that would have been important for efficient operation.

2.2 History (2/2)

- Ca. 2005
  - PC success becomes **problematic**
    - How to limit **energy usage** and **management overhead** of fleets of PCs?
    - One answer: Use virtualization for **server consolidation**
      - Turn independent servers into VMs, then allocate them to single server
      - Servers often with low resource utilization (e.g., CPU usage between 10% and 50% at Google in 2007, (Barroso and Hölzle 2007))
      - Consolidated server with improved resource utilization
    - Additional answer: Virtualization reduces management, testing, and deployment overhead, see (Vogels 2008) for Amazon
  - Virtualization as enabler for cloud computing
- Literature
  - (Soltesz et al. 2007): Containers for lightweight virtualization
  - (Castro et al. 2019; Kounev et al. 2023): Serverless computing

At the start of this millennium, cheap PCs became highly popular. Thanks to Moore's Law, they were now powerful enough to be used in datacenters of big tech companies. For the used fleets of machines, energy usage and management overhead became limiting challenges.

One answer to address these challenges was found in server consolidation based on virtualization: Here, previously independent servers were reconfigured into VMs, then allocated to single servers.

For example, a paper from Google cited here reports that servers often showed low resource utilization. After consolidating them as VMs on a single server, much higher utilization rates were achieved.

Additionally, a paper from Amazon cited here reports that virtualization reduced their management, testing, and deployment overhead.

Consequently, virtualization turned out to be an enabler for cloud computing.

Note that the benefits of virtualization mentioned here also apply to containerization. Indeed, for some time now, containers have been providing support for lightweight virtualization. This trend also enabled the rise of serverless computing, an evolution of cloud computing to be revisited in a later presentation.

2.3 Virtualization Variants

- Virtualization: Creation of virtual/abstract version of something
  - Hardware/system: virtual machine (VM)
  - Virtual memory in OSs
  - Network, e.g., overlay networks, software-defined networking
    - Beyond class
  - Execution environment (e.g., Java, Dotnet)

  In the context of IT Systems, "virtualization" may mean different things, but it always refers to the creation of a virtual or abstract version of something else.

  First, virtual machines may be execution environments where a guest OS runs on virtual hardware, as introduced previously and revisited next.

  Then, you saw virtual memory in the context of OSs.

  Also, virtualization exists for overlay networks or software-defined networking, which is beyond our topics.

  Finally, Java programs run inside "virtual machines". Note that such virtual machines are processes inside a host OS on real hardware.

- Typical meaning: **virtual machine** (VM)
  - Virtual hardware
    - Several OSs run concurrently, share (variants of) underlying hardware
  - VMs isolated from each other

In this presentation, our focus is on VMs as introduced previously: A VMM provides virtual hardware, on which guest OSs run. The guest OSs and VMs share the same underlying hardware, but are isolated from each other.

3 Virtualization

Let us see a precise definition of virtualization and its properties.

3.1 Definitions

- Cited from (Popek and Goldberg 1974)
  - "A **virtual machine** is taken to be an *efficient*, *isolated duplicate* of the real machine." (bold face added)
    The classical definition of a virtual machine is as follows: "A virtual machine is taken to be an efficient, isolated duplicate of the real machine."
  - Made precise with **Virtual Machine Monitor** (VMM)
    - "First, the VMM provides an **environment** for programs which is **essentially identical** with the original machine; second, programs run in this environment show at worst only **minor decreases in speed**; and last, the VMM is in **complete control** of system resources."
      This definition is made precise with the notion of the virtual machine monitor, VMM for short:
      "First, the VMM provides an environment for programs which is essentially identical with the original machine; second, programs run in this environment show at worst only minor decreases in speed; and last, the VMM is in complete control of system resources."
      Thus, a VM is an environment for programs, which is managed by a VMM, which in turn satisfies certain properties. Let us look at these properties in more detail.
      - Essentially identical: Programs with same results (as long as they do not ask for hardware specifics), maybe different timing
        The virtualized environment is considered to be essentially identical, if programs produce the same results as they would on real hardware (as long as they do not ask for hardware specifics). They may show differences in timing, though.
      - Speed: Most instructions executed directly by CPU with no VMM intervention
        The virtualized environment should not reduce the speed of program too much. More precisely, most instructions should be executed directly by the CPU.
        In fact, we expect CPU bound computations to run without VMM interventions at native speed. However, access to virtualized hardware requires VMM intervention, which comes with additional work and overhead.
      - Control: (1) Virtualized programs restricted to resources allocated by VMM, (2) VMM can regain control over allocated resources
        The VMM should be in control of resource allocations. Thus, virtualized programs must be restricted to resources allocated by the VMM, and the VMM must be able to regain control over allocated resources.
    - "A *virtual machine* is the environment created by the virtual machine monitor."
      Given this understanding of a VMM, the following alternative definition for a VM emerges: "A virtual machine is the environment created by the virtual machine monitor."

3.2 Isolation

- Isolation of VMs: Illusion of exclusive hardware use (despite sharing between VMs)
  - Related to "isolated duplicate" and "complete control" of (Popek and Goldberg 1974)

  The above definition requires a virtual machine to be an "isolated duplicate" of the real machine. Similarly to how processes and their virtual address spaces are isolated from each other by the OS, VMs are isolated from each other by the VMM: Thanks to the "complete control" of the VMM over hardware, it can restrict each VM to run under the illusion of exclusive hardware access.

- Sub-types (see (Soltesz et al. 2007; Felter et al. 2015))
  Different types of isolation are discussed in the literature.
  - Resource isolation: Fair allocation and scheduling
    - Reservation (e.g., number of CPU cores and amount of RAM) vs best-effort

    Each VM may receive certain resources, independently of the behavior of other VMs, which are running on the same computer. The VMM may offer strict reservation mechanisms towards that goal, or it may run with relaxed "best effort" promises.
  - Fault isolation: Buggy component should not affect others
    Faults in one VM should not affect other VMs.
  - Security isolation
    - Configuration independence (global names/settings do not conflict)
    - Safety (no access between VMs/containers)
    - Beware! Lots of security issues in practice
      - E.g., hypervisor privilege escalation and cross-VM side channel attacks

    Security isolation is probably the most complicated type of isolation, which subsumes different expectations: First, we may expect configurations for different VMs to be independent of each other. E.g., names of files chosen in one VM should not conflict with the same names in other VMs. Regarding safety, VMs must not be able to modify data or code of other VMs.
    Importantly, security isolation of VMs is a complex topic. Clearly, different VMs share the same CPUs with registers and caches. Time and again, this sharing can be exploited for so-called side-channel attacks that break isolation promises: Code running in one VM extracts secrets from another VM. E.g., see the general search terms on the slide or check out specific vulnerabilities of CPU architectures such as Meltdown and Spectre in 2017, Herzbleed in 2022, GhostRace in 2024, Training Solo in 2025.
    Be careful when you run code or manage potentially sensitive data "in the cloud".

3.3 Layering with Virtualization

- Hypervisor/VMM with full access to hardware

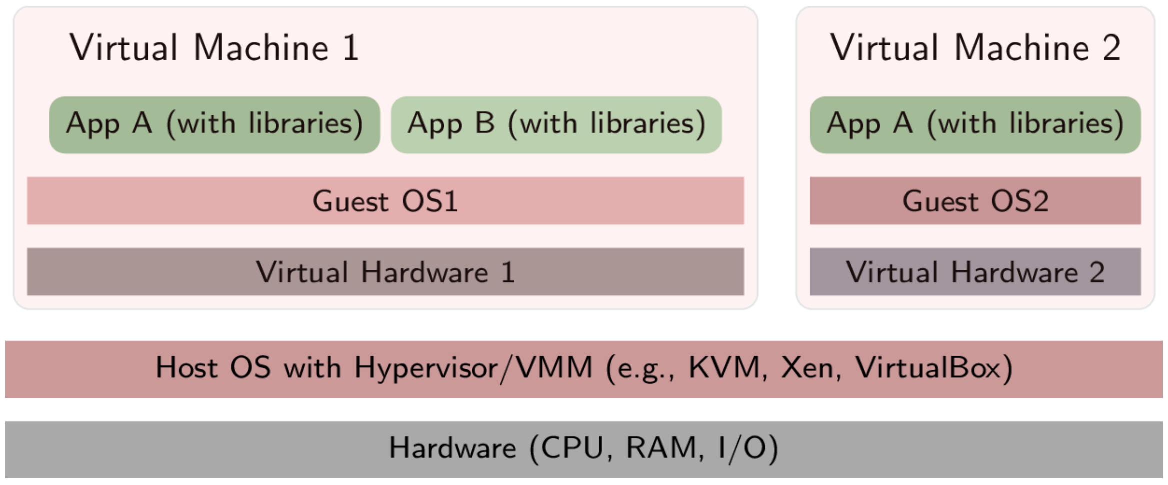

Figure 3: Layering with virtualization

- Most privileged code
  - Recall CPU Rings
- Creates abstract versions of hardware, to be used by **de-privileged guest OSs**

Let us revisit this layered architecture. Towards the bottom, the VMM or hypervisor is part of the host software, which may be a full OS or specialized software. It has full privileges on the physical hardware, replacing the role of a kernel for ordinary OSs. It creates abstract or virtual versions of hardware. This virtual hardware is what is visible to virtual machines and their guest OSs. These guest OSs now run with fewer privileges, and accesses to virtual hardware by their kernels is mediated by the VMM. More privileged, or sensitive, instructions are intercepted by the VMM, to be checked and executed, translated, or emulated accordingly.

Inside each VM, applications for the guest OS can be installed as usual. Importantly, applications do not need to be modified in any way for execution under virtualization.

- Notes
  - Each VM can run different OS, isolated from others
    - VM backups/snaphots **simplify** management, placement, parallelization
  - Creation of more VMs with **high overhead**
    - (Compared to containerization)

As explained earlier, virtualization isolates VMs from each other. VMs can be managed independently, including backup, updating, placement and deployment, potentially on multiple physical machines in parallel.

As VMs are isolated, sharing among applications in different VMs is restricted: OS mechanisms such as shared memory, shared files, or pipes are not available across VMs (but, of course, still work inside each VM); instead, networking is necessary for communication across VMs.

Finally, each VM requires some portion of the underlying hardware resources, which implies that VM creation comes with an overhead. Compared to containerization later on, that overhead is high.

3.4 Self-Study Question

- The Java VM was mentioned as variant of virtualization. Discuss whether it satisfies the conditions for virtualization as defined in 1974.

Take a break to work on the self-study task here.

Bibliography

Barroso, L. A., and U. Hölzle. 2007. "The Case for Energy-Proportional Computing." *Computer* 40 (12): 33–37. https://doi.org/10.1109/MC.2007.443.

Castro, Paul, Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. 2019. "The Rise of Serverless Computing." *Commun. Acm* 62 (12): 44–54. https://doi.org/10.1145/3368454.

Felter, Wes, Alexandre Ferreira, Ram Rajamony, and Juan Rubio. 2015. "An Updated Performance Comparison of Virtual Machines and Linux Containers." In *Performance Analysis of Systems and Software (Ispass), 2015 Ieee International Symposium on*, 171–72. IEEE.

Goldberg, Robert P. 1974. "Survey of Virtual Machine Research." *Computer* 7 (6): 34–45.

Kounev, Samuel, Nikolas Herbst, Cristina L. Abad, Alexandru Iosup, Ian Foster, Prashant Shenoy, Omer Rana, and Andrew A. Chien. 2023. "Serverless Computing: What It Is, and What It Is Not?" *Commun. Acm* 66 (9): 80–92. https://doi.org/10.1145/3587249.

Popek, Gerald J., and Robert P. Goldberg. 1974. "Formal Requirements for Virtualizable Third Generation Architectures." *Commun. Acm* 17 (7): 412–21. https://doi.org/10.1145/361011.361073.

Soltesz, Stephen, Herbert Pötzl, Marc E Fiuczynski, Andy Bavier, and Larry Peterson. 2007. "Container-Based Operating System Virtualization: A Scalable, High-Performance Alternative to Hypervisors." In *Acm Sigops Operating Systems Review*, 41:275–87. 3. ACM.

Vogels, Werner. 2008. "Beyond Server Consolidation: Server Consolidation Helps Companies Improve Resource Utilization, but Virtualization Can Help in Other Ways, Too." *Queue* 6 (1): 20–26. https://doi.org/10.1145/1348583.1348590.

The bibliography contains references used in this presentation.

License Information