

Docker Introduction *

Jens Lechtenbörger

Data Science Winter School 2023

Contents

Introduction

Motivation (1/2)

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

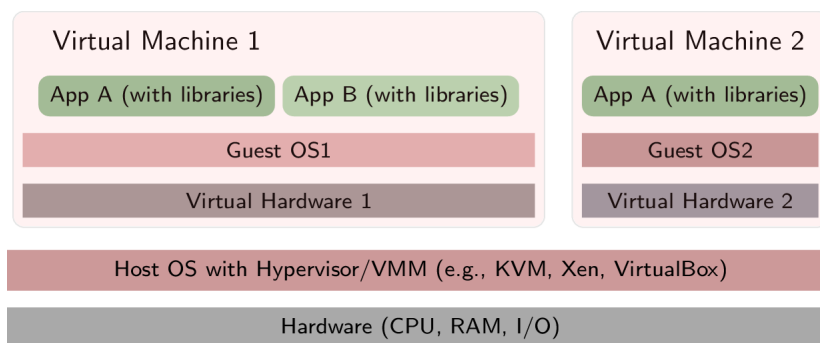


Figure 1: Layering with virtualization

- Interface implemented by Hypervisor/VMM
 - * VMM runs on (usual) host OS, manages real hardware
- Virtual hardware can have **arbitrary** features
 - * Largely independent of real hardware, say, ten network cards
- 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
 - * Use VM as **blueprint** to share reliable environment with others

*This PDF document is an inferior version of an [OER HTML page](#); [free/libre Org](#) mode source repository.

- Or to fire up lots of **identical** VMs for compute-intensive tasks with **cloud computing**

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.

Motivation (2/2)

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

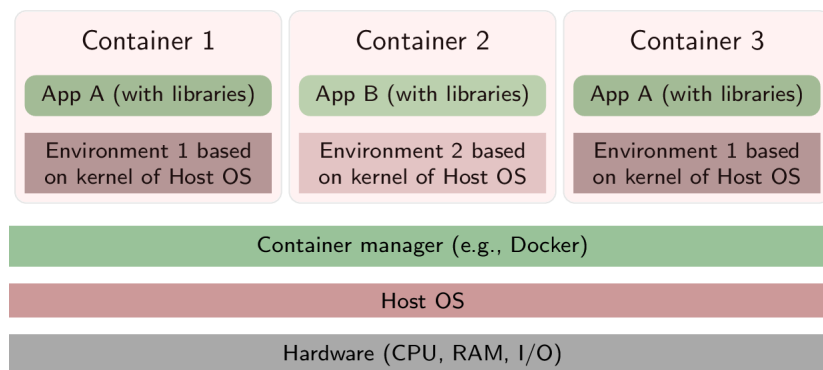


Figure 2: Layering with containerization

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

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/Solid server with static files
 - Interpret and modify simple docker files

Core Questions

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

Virtualization

History (1/2)

- Virtualization is an old concept
 - IBM mainframes, 1960s
 - Frequently cited survey article by Goldberg, 1974: [Gol74]
 - Original motivation
 - * Resources of **expensive** mainframes better utilized with multiple VMs
 - * Ability to run different OS versions in parallel, **backwards compatibility**
- 1980s, 1990s
 - Modern multitasking OSs on **cheap** hardware
 - * Cheap hardware did not offer virtualization support
 - * Little use of virtualization

History (2/2)

- Ca. 2005
 - PC success becomes **problematic**
 - * How to limit **energy usage** and **management overhead** of fleets of PCs in data centers?
 - * 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, [BH07])
 - Consolidated server with improved resource utilization
 - * Additional answer: Virtualization reduces management, testing, and deployment overhead, see [Vog08] for Amazon
 - Virtualization as enabler for **cloud computing**
- [SPF+07]: Containers for lightweight virtualization
- [CIM+19; KHA+23]: Serverless computing

Intuition and Examples

- Virtualization: Creation of virtual/abstract version of something
 - Virtual memory, recall OS concepts
 - * Not our focus
 - Network, e.g., overlay networks, software-defined networking
 - * Not our focus
 - Execution environment (e.g., Java, Dotnet)
 - Hardware/system: virtual machine (VM)
- Typical meaning: **virtual machine** (VM)
 - Virtual hardware
 - * Several OSs share same underlying hardware
 - VMs isolated from each other

Definitions

- Cited from [PG74] (bold face added)
 - “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.”
 - Essentially identical: Programs with same results (as long as they do not ask for hardware specifics), maybe different timing
 - Speed: Most instructions executed directly by CPU with no VMM intervention
 - Control: (1) Virtualized programs restricted to resources allocated by VMM, (2) VMM can regain control over allocated resources
 - * “A *virtual machine* is the environment created by the virtual machine monitor.”

TODO Explain this.

Isolation

- Isolation of VMs: Illusion of exclusive hardware use (despite sharing between VMs)
 - Related to “isolated duplicate” and “complete control” of [PG74]
- Sub-types (see [SPF+07; FFR+15])

- Resource isolation: Fair allocation and scheduling
 - * Reservation (e.g., number of CPU cores and amount of RAM) vs best-effort
- Fault isolation: Buggy component should not affect others
- Security isolation
 - * Configuration independence (global names/settings do not conflict)
 - Applications with conflicting requirements for system-wide configuration
 - E.g., port 80 for Web servers, each application with own version of shared libraries
 - * Safety (no access between VMs/containers)
 - * Beware! Lots of security issues in practice
 - E.g., **hypervisor privilege escalation** and **cross-VM side channel attacks**

Layering with Virtualization

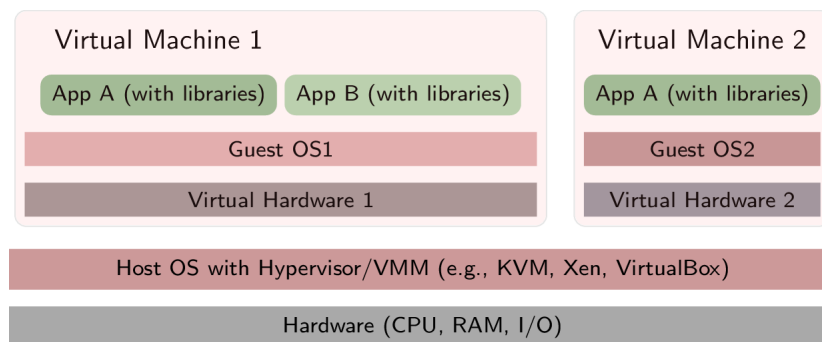


Figure 3: Layering with virtualization

Layering Explained

- Hypervisor or virtual machine manager (VMM) with full access to physical hardware
 - Most privileged code
 - * Details depend on CPU hardware
 - E.g., **kernel mode** (CPU ring 0) or additional “root mode” (e.g., ring -1) with more privileges than kernel mode
 - Create abstract versions of hardware, to be used by **guest OSs**
 - * VM = Guest OS running on abstract hardware
 - * Host = Environment in which the VMM runs
 - Host software may be full OS or specialized
- Guest OS is **de-privileged**

- No longer with full hardware access, e.g., CPU ring 1
- Privileged/sensitive instructions lead to hypervisor
 - * Executed, translated, or emulated accordingly
- Each VM can run different OS
- VM backups/snapshots **simplify** management, placement, parallelization
- Sharing among applications in different VMs **restricted**, requires networking
 - (Neither shared memory nor file nor pipes)
- Creation of more VMs with **high overhead**
 - Each with full OS, own portion of underlying hardware

Review Question

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

Containerization

Basics

- Motivation: Trade isolation for efficiency (see [SPF+07])
 - **Main idea** of containerization: **Share kernel** among containers
 - * (Instead of separate OS per VM)
- Mechanisms
 - Add container ID to each process, add new access control checks to system calls
 - In case of Linux kernel
 - * Kernel namespaces
 - Limit what is visible inside container
 - * Control groups (cgroups)
 - Limit resource usage
 - * Copy-on-write, e.g., UnionFS
 - New container without copying all files, localized changes

Layering with Containerization

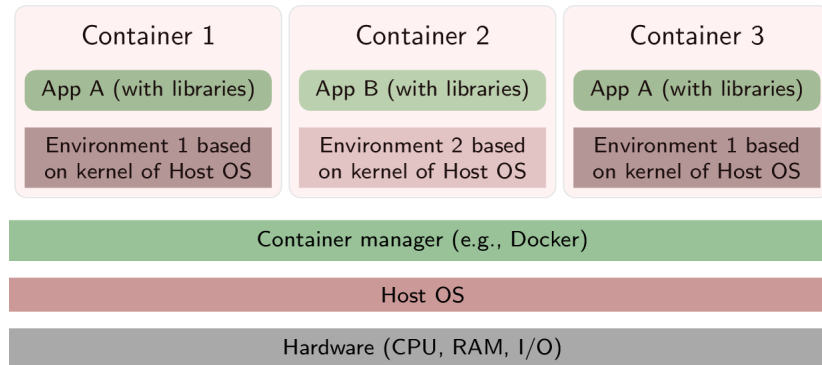


Figure 4: Layering with containerization

Selected Technologies

- Docker



Figure 5: “Docker logo” under Docker Brand Guidelines; from Docker

- **Image** describes OS/application environment: What software/configuration?
 - * **Registries** publish images
 - * **Dockerfiles** are build recipes for images in simple text format
- **Container** is process (set), created from image (image is template for container)

- Kubernetes



Figure 6: “Kubernetes logo” under Kubernetes Branding Guidelines; from GitHub

- Cluster manager for Docker
 - * Pod = group of containers sharing resources, unit of deployment
 - * Pods can be replicated (copied) for scalability
 - * Integrated load-balancer

On Images

- With VMs, you could install software as in any other OS
 - Getting messy over time
- With Docker, images are defined via Dockerfiles
 - Explicitly listing necessary pieces and dependencies
 - Enforcing order and reproducibility
 - [Sample dockerfile](#) (used in the past to generate reveal.js presentations and PDF from org files):

```
FROM ubuntu
LABEL maintainer="Jens Lechtenbörger"
RUN apt-get update && apt-get --no-install-recommends install -y \
    ca-certificates emacs git \
    texlive-bibtex-extra texlive-fonts-recommended texlive-generic-recommended \
    texlive-latex-base texlive-latex-extra texlive-latex-recommended
COPY manage-packages.el /tmp/
```

Review Question

- Which conditions for virtualization as defined in 1974 does Docker satisfy?

Docker

Docker Installation

- Community Edition of Docker available for different OSs
 - See [here for installation links](#)
- Install on one of your machines, ideally on one that you can bring to (or access in) class
 - Your installation may come with a graphical user interface (GUI), which you do **not** need
 - * Some students perceive the GUI to be confusing
 - * Use command line instead to enter commands shown subsequently (any terminal should work, maybe try [Bash](#))

First Steps

- Run hello-world, read output
 - `docker run hello-world`
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
[...]
- List your images and containers

- `docker image ls`
- `docker container ls -all`
 - * Help is available, e.g.:
 - `docker container --help`
 - `docker container ls --help`
- Maybe delete image and container
 - `docker rmi -f hello-world`

A Web Server

- Run `nginx`
 - `docker run -p 8080:80 nginx`
 - * `-p`: Web server listens on port 80 in container; bind to port 8080 on host
 - Visit `local server` (see subsequent slide for Docker Toolbox under Windows)
 - * Maybe add option `--name my-nginx`: Assign name to container for subsequent use
 - E.g., `docker stop/start/logs/rm my-nginx`
- Serve own HTML files
 - Add option `-v` in above `docker run ...` (**before** `nginx`)
 - * Mount (make available) directory from host in container
 - E.g.: `-v /host-directory/with/html-files:/usr/share/nginx/html`
 - `/usr/share/nginx/html` is where `nginx` expects HTML files, in particular `index.html`
 - Thus, your HTML files replace default ones of `nginx`

Selected Errors

- **Error** message: name in use already
 - You cannot use the same name multiple times with `docker run --name ...`
 - Instead: `docker start my-nginx`
- **Error** message: port is allocated already
 - You cannot use option `-p` with same port in several `docker run` invocations
 - * Other container still running, stop first
 - `docker ps`: Note ID or name
 - `docker stop <ID-or-name>`
 - `docker run ...`
 - * (Or some other process uses that port. Kill process or choose different port.)

On Option -v

- Say, you start `nginx` with option `-v` but your files do not appear
 - `docker inspect <name-or-id-of-container>`
 - * Check output for `binds`, telling you what is mapped to `/usr/share/nginx/html`
 - May not meet your expectations
 - Are you on Windows?
 - * Try `-v C:\Users\...` with Powershell
 - * Try `-v C:\\Users/...` with Bash
 - * Try `-v /mnt/c/Users/...` with WSL terminal

Docker Toolbox under Windows

- (I do not recommend this in any way. [Switch to GNU/Linux.](#))
- Docker Toolbox installs a virtual machine, in which Docker runs
 - Initial output informs about
 - * IP address of VM, e.g., `192.168.99.100`
 - Visit [port 8080 on 192.168.99.100](#)
 - * File system path
 - `/c/Program Files/Docker Toolbox`
 - Paths under `C:\Users` can be mounted by default
 - * E.g., `docker run -p 8080:80 -v /c/Users/<your-name>/<folder-with-index.html>:/usr/share/nginx/html nginx`
 - [Maybe you need double slashes](#)

Conclusions

Summary

- Virtual **virtual machines** are **efficient, isolated duplicates** of the real machine
- **Containers** are running processes, defined by **images**
 - Containers on one host share same OS kernel
- Virtual machines and containers
 - can be contrasted in terms of their layering approaches
 - allow to deploy software in well-defined environments

Outlook

- Containerization (in combination with version control such as offered by Git) is enabler of **DevOps**
 - DevOps = Combination of Development and Operations, see [JbA+16; WFW+19]
 - * Bridge gaps between teams and responsibilities
 - * Aiming for rapid software release cycles with high degree of automation and stability
 - Trend in software engineering
 - * Communication and collaboration, continuous integration (CI) and continuous deployment (CD)
 - * Approach based on Git also called GitOps, see [Lim18]
 - Self-service IT with proposals in pull requests (PRs)
 - Infrastructure as Code (IaC)

Bibliography

License Information

Source files are available on [GitLab](#) (check out embedded submodules) under free licenses. Icons of custom controls are by [@fontawesome](#), released under [CC BY 4.0](#).

Except where otherwise noted, the work “Docker Introduction”, © 2018-2021, 2023 [Jens Lechtenbörger](#), is published under the [Creative Commons](#) license [CC BY-SA 4.0](#).