Docker Introduction

Jens Lechtenbörger

VM OER 2019/2020

Contents

1 Introduction
2 Virtualization
3 Containerization
4 Docker
5 Conclusions

1 Introduction

1.1 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

1.2 Core Questions

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

2 Virtualization

2.1 History

- 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
    * Little use of virtualization, no hardware support any longer

- Since ca. 2005
  - PC success becomes problematic
    * Energy usage, management overhead
  - Virtualization for server consolidation, cloud computing

2.2 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

2.3 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, 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.”

2.4 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 [Sol+07; Fel+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/setting 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

2.5 Layering with Virtualization

![Layering with virtualization](image)

Figure 1: Layering with virtualization
2.5.1 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” 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

2.6 Review Question

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

3 Containerization

3.1 Basics

- Motivation: Trade isolation for efficiency (see [Sol+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

### 3.2 Layering with Containerization

![Layering with Containerization Diagram](image)

Figure 2: Layering with containerization

### 3.3 Selected Technologies

- **Docker**

![Docker Logo](image)

Figure 3: “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**

![Kubernetes Logo](image)

Figure 4: “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

3.3.1 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):

```bash
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/
```

4 Docker

4.1 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

4.2 First Steps

- Run hello-world as instructed in Get Started
  - In case of problems, please ask in the forum
- 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
4.3 A Web Server

- Run nginx

  - `docker run --name my-nginx -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)
    * `--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

4.3.1 Selected Errors

- Repeated use of `docker run --name ...` with same name

  - Error message: name in use already
  - Instead: `docker start my-nginx`

- Use of option `-p` with same port in several `docker run` invocations

  - Error message: port is allocated already
  - Other container still running, stop first
    * `docker ps`: Note ID or name
    * `docker stop <ID-or-name>`
    * `docker run` ...

4.3.2 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`
      * Maybe you need double slashes
5 Conclusions

5.1 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

Bibliography


License Information

This document is part of a larger course. Source code and source files are available on GitLab under free licenses.

Except where otherwise noted, this work, “Docker Introduction”, is © 2018, 2019 by Jens Lechtenbörger, published under the Creative Commons license CC BY-SA 4.0.

No warranties are given. The license may not give you all of the permissions necessary for your intended use.

In particular, trademark rights are not licensed under this license. Thus, rights concerning third party logos (e.g., on the title slide) and other (trade-) marks (e.g., “Creative Commons” itself) remain with their respective holders.