

Containerization

BECM33MLE — Machine Learning Engineering

Dr. Tomáš Bába

Multi-Robot Systems group, Faculty of Electrical Engineering
Czech Technical University in Prague



Lecture 11: Containerization

2025-12-09

Containerization BECM33MLE — Machine Learning Engineering

Dr. Tomáš Bába

Multi-Robot Systems group, Faculty of Electrical Engineering
Czech Technical University in Prague



Why containers?

A solution to the dependency hell

When you need to run 2 apps with conflicting dependencies.

A solution to isolation

When you need to run two apps that might conflict on networking, file system or runtime level.

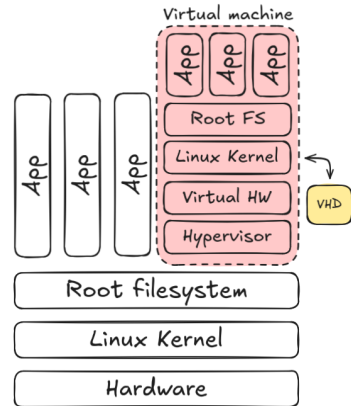
A solution to portability

Being able to share the whole system as well as a *prescription* for its installation.

Comparison to Virtual Machines (VMs)

The other extreme: complete isolation (almost hardware-like)

Virtual machines



Lecture 11: Containerization

Motivation

Why containers?

Why containers?

A solution to the dependency hell

When you need to run 2 apps with conflicting dependencies.

A solution to isolation

When you need to run two apps that might conflict on networking, file system or runtime level.

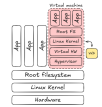
A solution to portability

Being able to share the whole system as well as a *prescription* for its installation.

Comparison to Virtual Machines (VMs)

The other extreme: complete isolation (almost hardware-like)

Virtual machines



Why containers?

Lecture 11: Containerization

Tomáš Bába

Motivation

Docker

Management tools

Lazydocker
Portainer

Orchestration

Apptainer

Nix

References

A solution to the dependency hell

When you need to run 2 apps with conflicting dependencies.

A solution to isolation

When you need to run two apps that might conflict on networking, file system or runtime level.

A solution to portability

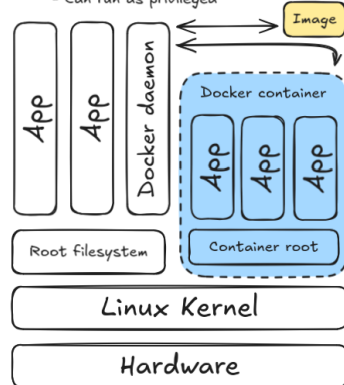
Being able to share the whole system as well as a *prescription* for its installation.

Comparison to Virtual Machines (VMs)

The other extreme: complete isolation (almost hardware-like)

Container system

- Complete namespace separation
- Can run as privileged



Tomáš Bába (CTU in Prague)

Lecture 11: Containerization

December 9th, 2025

2 / 17

Lecture 11: Containerization

Motivation

Why containers?

2025-12-09

Why containers?

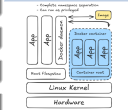
A solution to the dependency hell
When you need to run 2 apps with conflicting dependencies.

A solution to isolation
When you need to run two apps that might conflict on networking, file system or runtime level.

A solution to portability
Being able to share the whole system as well as a prescription for its installation.

Comparison to Virtual Machines (VMs)
The other extreme: complete isolation (almost hardware-like)

Container system



Core concepts

Lecture 11: Containerization

Tomáš Bába

Motivation

Docker

Management tools

Lazydocker
Portainer

Orchestration

Apptainer

Nix

References

Image

Layered file system with runtime, libraries, application code and configuration.

Container

A running instance of the software *spawned* from an image.

Networking

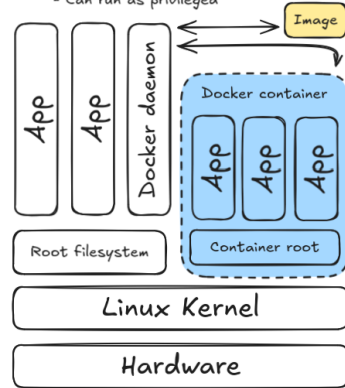
Containers connect by virtual network adapters.

Registries

Image are shared through registries where they are versioned and tagged.

Container system

- Complete namespace separation
- Can run as privileged



Tomáš Bába (CTU in Prague)

Lecture 11: Containerization

December 9th, 2025

3 / 17

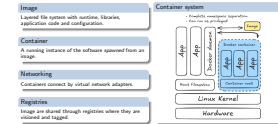
Lecture 11: Containerization

Motivation

Core concepts

2025-12-09

Core concepts



Docker — Architecture

Lecture 11: Containerization

Tomáš Bába

Motivation

Docker

Management tools

Lazydocker
Portainer

Orchestration

Apptainer

Nix

References

Docker client

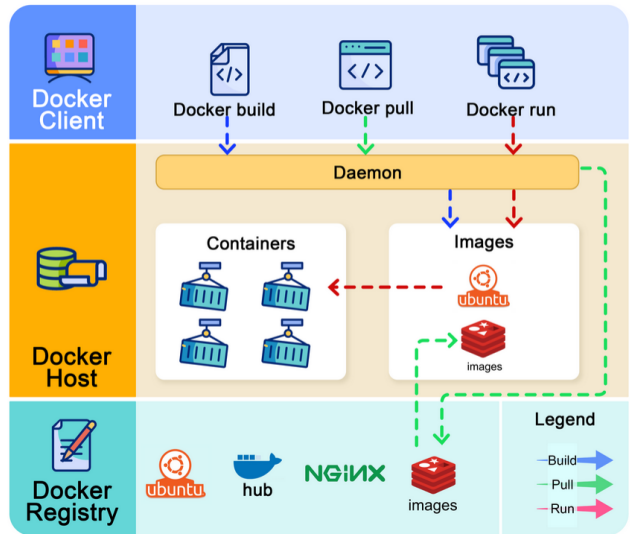
- user interaction
- control of the host

Docker daemon

- executes the image builds
- creates the runtime processes
- stores images locally

Docker registry

- analogue to git repository
- often hosted in cloud, can be local



Tomáš Bába (CTU in Prague)

Lecture 11: Containerization

December 9th, 2025

4 / 17

Lecture 11: Containerization

└─ Docker

└─ Docker — Architecture

2025-12-09

Docker — Architecture



- the Daemon runs as a system service on Linux, but as a as a light VM on Windows and MAC (they don't have the Linux Kernel).

1. Initial app development

- can happen natively (rare), or
- can already start with Docker in mind

2. Docker image build

- isolation of dependencies
- **recipe** for minimal image

3. Testing and development

- preparation of runtime instructions
- recipe for minimal image

4. Orchestration

- preparation of deployment

```
1 FROM alpine
2
3 ENV PYTHONUNBUFFERED=1
4
5 RUN apk add --update --no-cache python3 &&
6     ln -sf python3 /usr/bin/python
7
8 RUN ...
9
10 RUN ...
11
12 COPY my_script.py /
13
14 CMD ["python3", "-u", "/my_script.py"]
```

Figure 1: Illustration of a Dockerfile. See the lecture attachment for practical example.

Towards repeatability

- Containerizing your own work increases the chance that it will work not just **on your own machine**.

Minor points

- You can run linux-based docker containers on all major platforms, but Windows and Mac virtualize the Kernel
- the current host's USER can be emulated but he needs to be created during the image build phase
- GUI can be piped from the containers xserver to the host's xserver
- you can connect to remote docker host by exporting DOCKER_HOST
- docker can be used completely offline:
 - images can be transport as '.tar'
 - images can be transport through self-hosted registry.
- images can be built as multi-platform (or for specific platform only)
 - the cross-platform build runs in QEMU

Towards repeatability

- Containerizing your own work increases the chance that it will work not just **on your own machine**.

Minor points

- You can run linux-based docker containers on all major platforms, but Windows and Mac virtualize the Kernel
- the current host's USER can be emulated but he needs to be created during the image build phase
- GUI can be piped from the containers xserver to the host's xserver
- you can connect to remote docker host by exporting DOCKER_HOST
- docker can be used completely offline
 - images can be transport as '.tar'
 - images can be transport through self-hosted registry.
- images can be built as multi-platform (or for specific platform only)
 - the cross-platform build runs in QEMU

Native Docker CLI interface

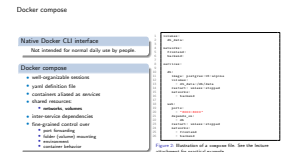
Not intended for normal daily use by people.

Docker compose

- well-organizable sessions
- yaml definition file
- containers aliased as *services*
- shared resources:
 - **networks, volumes**
- inter-service dependencies
- fine-grained control over
 - port forwarding
 - folder (volume) mounting
 - environment
 - container behavior

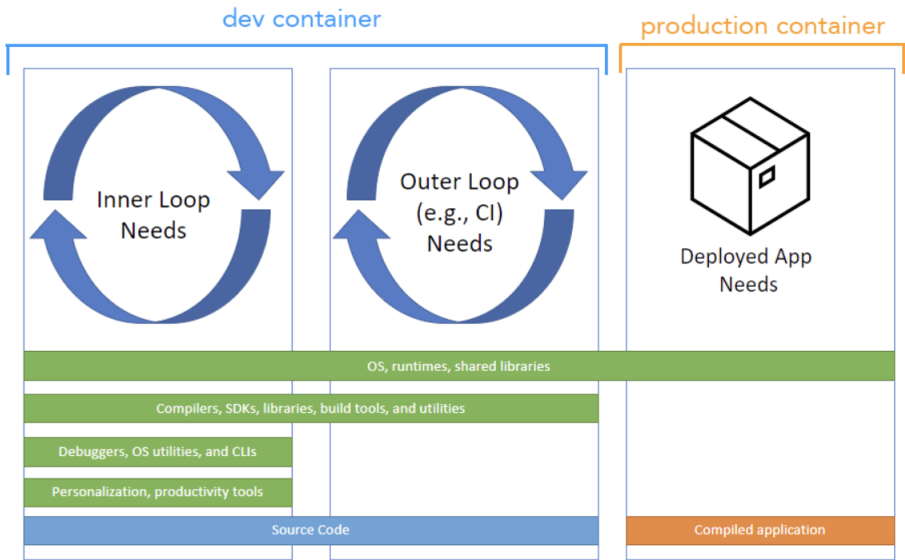
```
1 volumes:
2   db_data:
3
4 networks:
5   frontend:
6   backend:
7
8 services:
9
10  db:
11    image: postgres:16-alpine
12    volumes:
13      - db_data:/db/data
14    restart: unless-stopped
15    networks:
16      - backend
17
18  web:
19    ports:
20      - "8000:8000"
21    depends_on:
22      - db
23    restart: unless-stopped
24    networks:
25      - frontend
26      - backend
```

Figure 2: Illustration of a compose file. See the lecture attachment for practical example.



Docker — devcontainers [1]

Lecture 11: Containerization
Tomáš Báča
Motivation
Docker
Management tools
Lazydocker
Portainer
Orchestration
Apptainer
Nix
References



Tomáš Báča (CTU in Prague)

Lecture 11: Containerization

December 9th, 2025

8 / 17

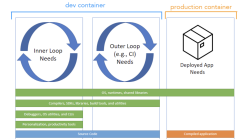
Lecture 11: Containerization

└─ Docker

└─ Docker — devcontainers [1]

2025-12-09

Docker — devcontainers [1]



Lazydocker — container management in TUI

Lecture 11: Containerization

Tomáš Bába

Motivation

Docker

Management tools

Lazydocker

Portainer

Orchestration

Apptainer

Nix

References

Lazydocker [2]

- TUI for local Docker management
 - (directly connecting to the Docker Host)
 - The *brother* of Lazygit
- image management
- container management
- volume management
- structured stdout visualization

Sidenote: use the right tool for the job

- Lazydocker, is not for everyone, but it can be the right tool for the job at a very specific time in your development, e.g., in the local testing phase.
- Aiming at the most ultimate tool for the job (unless you already know it) can be the wrong thing to do at the beginning of the project.

```
[1]-Project
k1axelk

~Config
Name: alpine
ID: sha256:9234e8fb04c47cfe0f49931e4ac7e
Tags: alpine:latest
Size: 8.31MB
Created: Tue, 15 Jul 2025 13:01:16 CEST

ID TAG SIZE COMMAND
<missing> 7.93MiB ADD alpine-m1

~[3]-Containers
running buildx_buildkit

~[4]-Images
alpine
cturns/kalibr
cturns/rvr_uv_system
cturns/ros_jazzy
cturns/ros_noetic
fy4future/robofly
fy4future/uv_custom
freqtradeorg/freqtrade

~[5]-Volumes
local buildx_buildkit

~[6]-Networks
bridge bridge
host host
null none

PgUp/PgDn: scroll, b: view bulk commands, q: quit, x: menu Donate 8.24.2
T214896 0 zsh 09:18 k1axelk
```

Lecture 11: Containerization

Management tools

Lazydocker

Lazydocker — container management in TUI

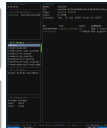
2025-12-09

Lazydocker [2]

- TUI for local Docker management
 - (directly connecting to the Docker Host)
 - The *brother* of Lazygit
- image management
- container management
- volume management
- structured stdout visualization

Sidenote: use the right tool for the job

- Lazydocker, is not for everyone, but it can be the right tool for the job at a very specific time in your development, e.g., in the local testing phase.
- Aiming at the most ultimate tool for the job (unless you already know it) can be the wrong thing to do at the beginning of the project.



Universal UI for containers

- handles local, remote, Docker and Kubernetes
- Fine-grained access control

Portainer Community Edition

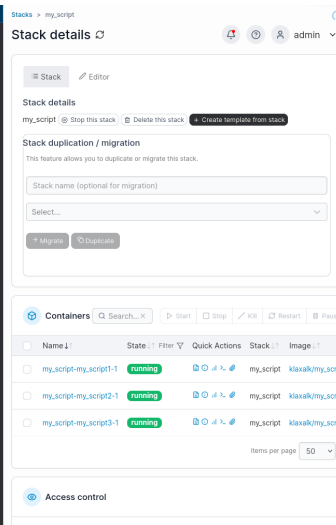
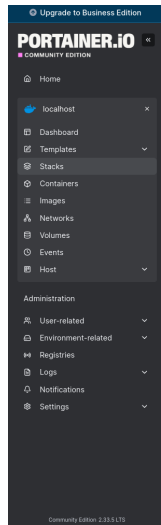
- Free, open source
- <https://docs.portainer.io/start/install-ce>

Portainer Agent (Docker API)

- Connects to Docker host
- Thin, no UI, stateless

Portainer Server (UI)

- Connects to agents (or Docker host)
- Stores stacks per agent

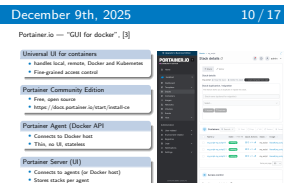


Lecture 11: Containerization

Management tools

Portainer

Portainer.io — “GUI for docker”, [3]



Orchestration

- health management of containers on a cluster
 - no program is 100% robust and reliable
 - downtime of containers is natural and it should be monitored and managed
- automatic load balancing using user-defined metric:
 - CPU load, memory usage, API request rate, etc.

Docker Swarm [4]

- orchestration built-in Docker
- automatic health monitoring
- automatic load-balancing
- automatic network management
 - *Nodes* get a dynamic hostname and ports
 - *Nodes* get the traffic routed automatically

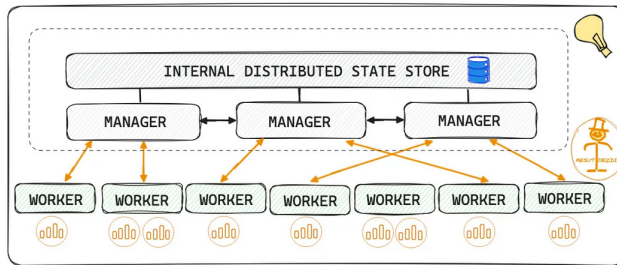


Figure 3: Source: [4]

December 9th, 2025 11 / 17

Orchestration

Orchestration

- health management of containers on a cluster
 - no program is 100% robust and reliable
 - downtime of containers is natural and it should be monitored and managed
- automatic load balancing using user-defined metric:
 - CPU load, memory usage, API request rate, etc.

Docker Swarm [4]

- orchestration built-in Docker
- automatic health monitoring
- automatic load-balancing
- automatic network management
 - *Nodes* get a dynamic hostname and ports
 - *Nodes* get the traffic routed automatically

Figure 3: Source [4]

Docker's problems

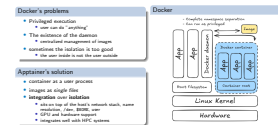
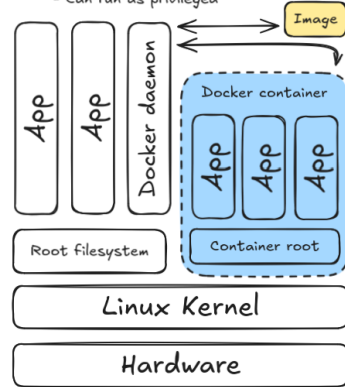
- Privileged execution
 - user can do "anything"
- The existence of the daemon
 - centralized management of images
- sometimes the isolation is too good
 - the user inside is not the user outside

Apptainer's solution

- container as a user process
- images as *single files*
- **integration over isolation**
 - *sits* on top of the host's network stack, name resolution, /dev, \$HOME, user
 - GPU and hardware support
 - integrates well with HPC systems

Docker

- Complete namespace separation
- Can run as privileged



Apptainer [5] — Motivation

Lecture 11: Containerization

Tomáš Bába

Motivation

Docker

Management tools

Lazydocker

Portainer

Orchestration

Apptainer

Nix

References

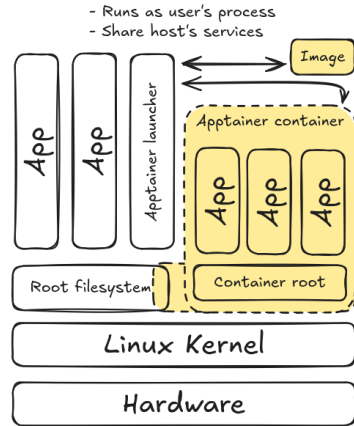
Docker's problems

- Privileged execution
 - user can do "anything"
- The existence of the daemon
 - centralized management of images
- sometimes the isolation is too good
 - the user inside is not the user outside

Apptainer's solution

- container as a user process
- images as *single files*
- **integration over isolation**
 - *sits* on top of the host's network stack, name resolution, /dev, \$HOME, user
 - GPU and hardware support
 - integrates well with HPC systems

Apptainer



Tomáš Bába (CTU in Prague)

Lecture 11: Containerization

December 9th, 2025

12 / 17

Lecture 11: Containerization

Apptainer

Apptainer [5] — Motivation

2025-12-09

Apptainer [5] — Motivation

This is a smaller version of the slide content, including the 'Docker's problems', 'Apptainer's solution', and 'Apptainer' diagram.

Image recipe

- Can bootstrap from Docker image
- Can bootstrap from another Apptainer image

Image file

- *.sif — Single Image File
- No local cache image cache
- Not layered images like Docker's
- Can be *sandboxed*
 - mutable by the user in runtime
 - can be packed back as an image
- Optional writable overlay

Runtime

- user spawns the process directly using an Apptainer's launcher system

Apptainer vs. Singularity

- Singularity used to be developed privately by Syllabs
 - SingularityCE (community Edition)
 - SingularityPro
- Apptainer developed by Linux Foundation
 - open source
 - backwards-compatible with the old Singularity

Practical comments

- similarly to Docker, the native CLI of Apptainer is not very pleasant
 - wrapping in custom scripts is usually beneficial

Concepts

- Image recipe**
 - Can bootstrap from Docker image
 - Can bootstrap from another Apptainer image
- Image file**
 - *.sif — Single Image File
 - No local cache image cache
 - Not layered images like Docker's
 - Can be *sandboxed*
 - mutable by the user in runtime
 - can be packed back as an image
 - Optional writable overlay
- Runtime**
 - user spawns the process directly using an Apptainer's launcher system
- Apptainer vs. Singularity**
 - Singularity used to be developed privately by Syllabs
 - SingularityCE (community Edition)
 - SingularityPro
 - Apptainer developed by Linux Foundation
 - open source
 - backwards-compatible with the old Singularity
- Practical comments**
 - similarly to Docker, the native CLI of Apptainer is not very pleasant
 - wrapping in custom scripts is usually beneficial

Usecase 1: sharing academic software

- preparing an image with a recipe or sandbox
- user is someone else than the creator

Usecase 2: “as a VM”

- long-term development of an obscure software
- starts by a recipe, ends with a sandbox

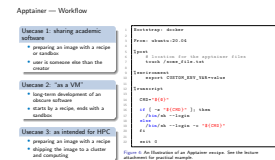
Usecase 3: as intended for HPC

- preparing an image with a recipe
- shipping the image to a cluster and computing

```

1 Bootstrap: docker
2
3 From: ubuntu:20.04
4
5 %post
6     # location for the apptainer files
7     touch /some_file.txt
8
9 %environment
10     export CUSTOM_ENV_VAR=value
11
12 %runscript
13
14     CMD="${@}"
15
16     if [ -z "${CMD}" ]; then
17         /bin/sh --login
18     else
19         /bin/sh --login -c "${CMD}"
20     fi
21
22     exit 0
    
```

Figure 4: An Illustration of an Apptainer recipe. See the lecture attachment for practical example.



- why would you use Apptainer over docker for sharing academic software?
 - with Apptainer, it is much easier to make it work over files (datasets, source codes) on your computer
 - with docker it is possible, but the work is gonna be done with root, or the user needs to build a new image and inject a matching user inside

Traditional dependency management (deb packaging system)

- you are expected to always update to the latest combination of stable versions
- **versions** are not encoded in dependencies
 - only version inequality constraints
 - only for a check.. the installer is not forced to install anything else than the latest version
- **installing older versions is not supported**
 - this leads to compilation from sources, but what if that needs older versions of dependencies?
 - To quote a famous Jedi: "Not updating is the path to the dark side, it leads to anger. Anger leads to hate. Hate leads to suffering".

Nix (<https://nixos.org/>)

- atomic and 100% replicable builds
 - the build should not rely on anything but **static sources** and Nix dependencies.
- build artifacts marked by hashes composed of:
 - source paths (and their hashes)
 - build dependencies (and their hashes)
 - system's hash
 - builder and build arguments
- **two libyaml-1.0.0 can have different hash and can be installed simultaneously**
- multiple instance of the library can be installed at once (even the same numeric version)
- **dependencies** can be encoded by hash

[6] E. Dolstra, A. Löh, and N. Pierron, "Nixos: A purely functional linux distribution," *Journal of Functional Programming*, vol. 20, no. 5-6, pp. 577–615, 2010

Lecture 11: Containerization

└─ Nix

└─ Nix — (maybe) future solution to dependency hell

Traditional dependency management (deb packaging system)	Nix (https://nixos.org/)
<ul style="list-style-type: none">• you are expected to always update to the latest combination of stable versions• versions are not encoded in dependencies• only version inequality constraints• only for a check.. the installer is not forced to install anything else than the latest version• installing older versions is not supported• this leads to compilation from sources, but what if that needs older versions of dependencies?• To quote a famous Jedi: "Not updating is the path to the dark side, it leads to anger. Anger leads to hate. Hate leads to suffering".	<ul style="list-style-type: none">• atomic and 100% replicable builds• the build should not rely on anything but static sources and Nix dependencies.• build artifacts marked by hashes composed of:<ul style="list-style-type: none">• source paths (and their hashes)• build dependencies (and their hashes)• system's hash• builder and build arguments• two libyaml-1.0.0 can have different hash and can be installed simultaneously• multiple instance of the library can be installed at once (even the same numeric version)• dependencies can be encoded by hash

[6] E. Dolstra, A. Löh, and N. Pierron, "Nixos: A purely functional linux distribution," *Journal of Functional Programming*, vol. 20, no. 5-6, pp. 577–615, 2010

Prerequisites

- Install Docker
 - <https://docs.docker.com/desktop/>
- Install Apptainer
 - <https://apptainer.org/docs/admin/main/installation.html>

How to run

Example scripts are located in lecture attachment: `scripts.zip`.

- Install Docker
 - <https://docs.docker.com/desktop/>
- Install Apptainer
 - <https://apptainer.org/docs/admin/main/installation.html>

Example scripts are located in lecture attachment: `scripts.zip`.

References I

Lecture 11: Containerization

Tomáš Bába

Motivation

Docker

Management tools

Lazydocker
Portainer

Orchestration

Apptainer

Nix

References

- [1] *Devcontainers*, <https://containers.dev/>, Accessed: 2025-12-09.
- [2] *Lazydocker*, <https://github.com/jesseduffield/lazydocker>, Accessed: 2025-12-09.
- [3] *Portainer*, <https://github.com/portainer/portainer>, Accessed: 2025-12-09.
- [4] *Docker swarm — an indepth introduction*, <https://mesutoezdil.medium.com/docker-swarm-6efa43f7b68d>, Accessed: 2025-12-09.
- [5] *Apptainer*, <https://apptainer.org/>, Accessed: 2025-12-09.
- [6] E. Dolstra, A. Löh, and N. Pierron, "Nixos: A purely functional linux distribution," *Journal of Functional Programming*, vol. 20, no. 5-6, pp. 577–615, 2010.

Tomáš Bába (CTU in Prague)

Lecture 11: Containerization

December 9th, 2025

17 / 17

Lecture 11: Containerization

References

References

2025-12-09

References I

- [1] *Devcontainers*, <https://containers.dev/>, Accessed: 2025-12-09.
- [2] *Lazydocker*, <https://github.com/jesseduffield/lazydocker>, Accessed: 2025-12-09.
- [3] *Portainer*, <https://github.com/portainer/portainer>, Accessed: 2025-12-09.
- [4] *Docker swarm — an indepth introduction*, <https://mesutoezdil.medium.com/docker-swarm-6efa43f7b68d>, Accessed: 2025-12-09.
- [5] *Apptainer*, <https://apptainer.org/>, Accessed: 2025-12-09.
- [6] E. Dolstra, A. Löh, and N. Pierron, "Nixos: A purely functional linux distribution," *Journal of Functional Programming*, vol. 20, no. 5-6, pp. 577–615, 2010.