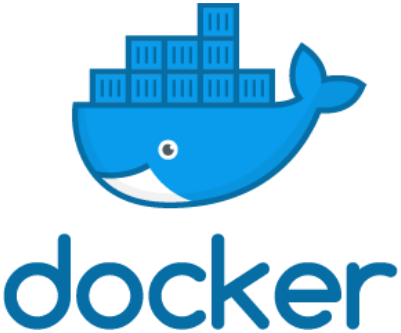
Docker
BWI Innovation Talk

Dr. Timm Heuss
March 2018



Docker is one of the most interesting and promising paradigm shifts of the last years. In this talk, I will introduce Docker shortly and present universal lessons learned after two years of practical use. It becomes clear that not only source code needs to be refactored, but also infrastructure, processes and people's mindsets. Once embraced, Docker allows for faster ramp-up times and is an enabler for modern software development.

These slides are available online:
https://github.com/heussd/docker-bwi-innovation-talk

Fraunhofer
FKIE

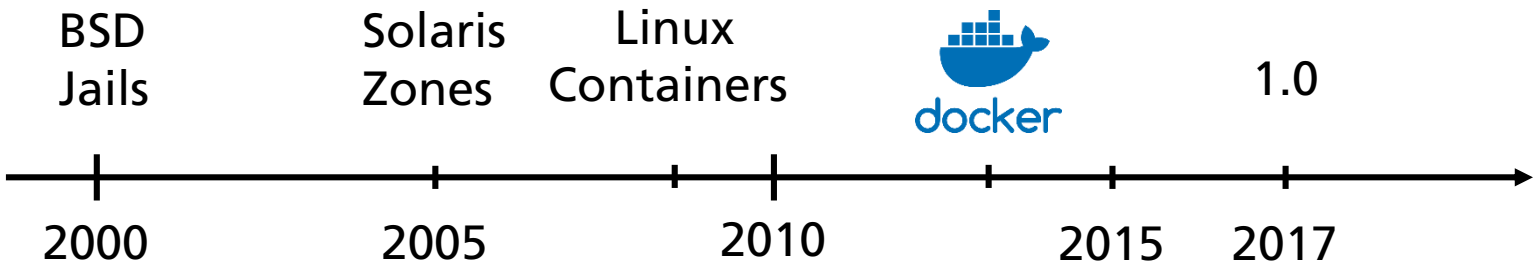
Agenda

- Introduction to Docker
- Why the Hype?
- Lessons learned from two years of practical use
- Interesting readings and tools
- What's next?

These slides are available online:

https://github.com/heussd/docker-bwi-innovation-talk

Introduction to Docker



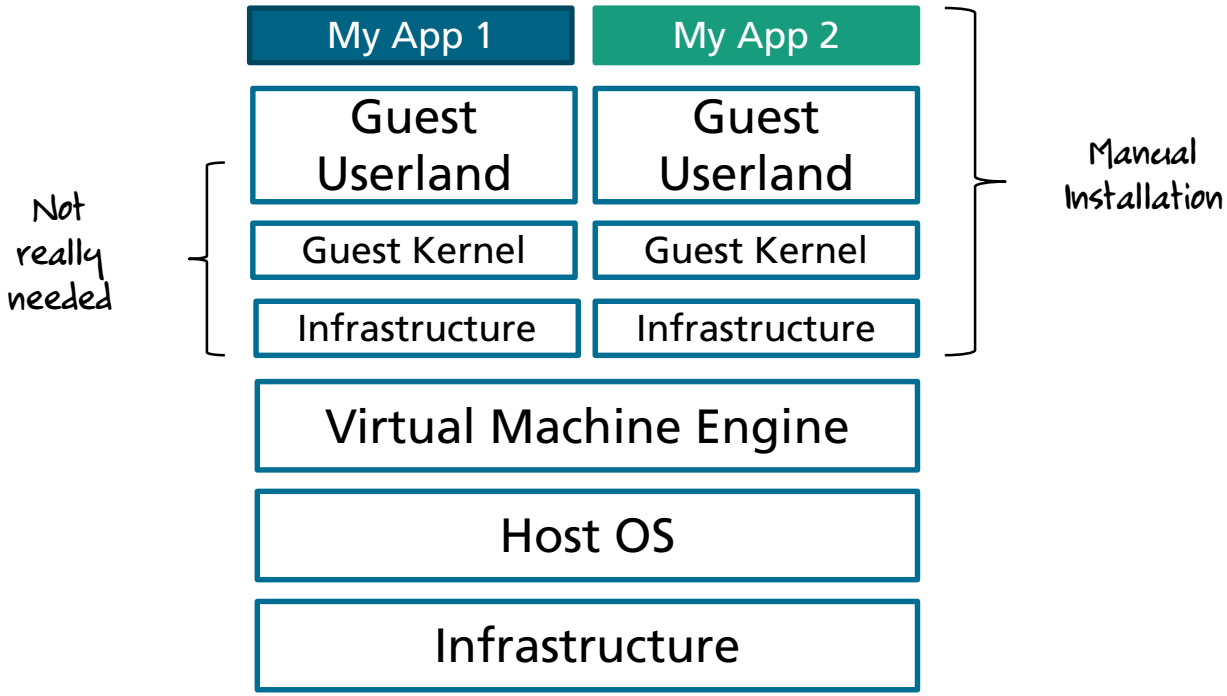
- Shipping software in virtual images with all runtime dependencies
- The idea is not really new
- What makes Docker special:
  - Docker focuses on streamlined developer workflows
  - Cross-platform runtime environment
  - Open Container Initiative (OCI) Standard
- Related approaches: rkt (“Rocket”), Facebook Tupperware, …

Introduction to Docker

VirtualBox?

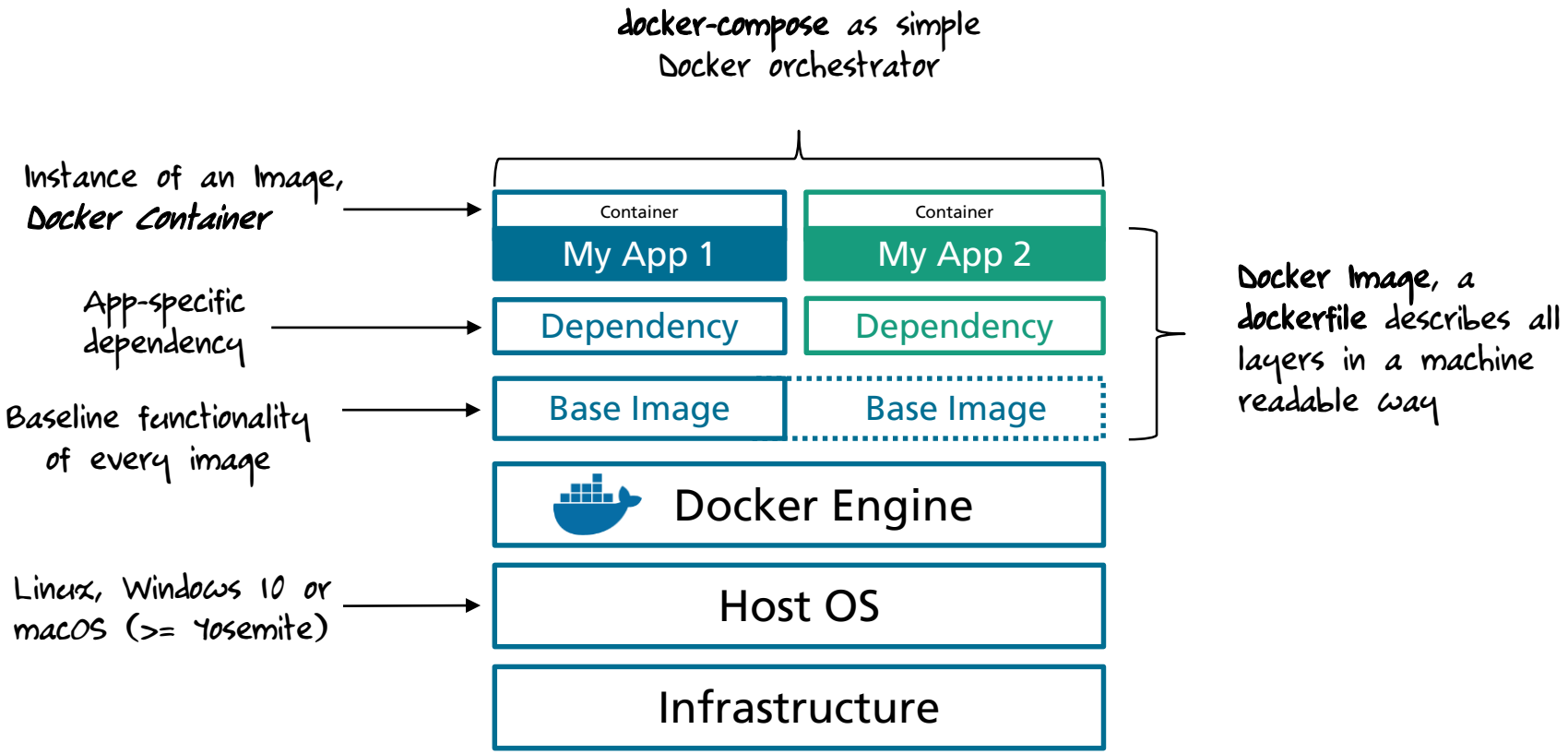
- Shipping software in virtual images with all runtime dependencies

VMware?



Introduction to Docker

- Shipping software in virtual images with all runtime dependencies



Dockerfile + docker-compose.yml

```
services:
  app1:
    build: myapp1/
    image: java-sth
    ports: 8080
  app2:
    build: myapp2/
    image: bash-tool
  importantdb:
    image: postgres:latest
```

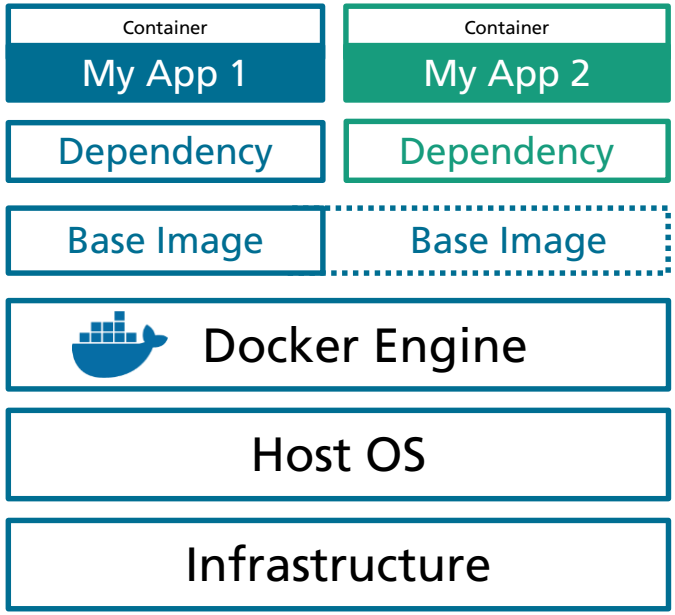
docker-compose

latest... what exactly?
We'll see later:
Lesson 2: Meanings of "latest"

```
ADD myapp1.war
/usr/local/tomcat/webapps

FROM tomcat:alpine
```

dockerfile

alpine is a special
size-reduced baseline

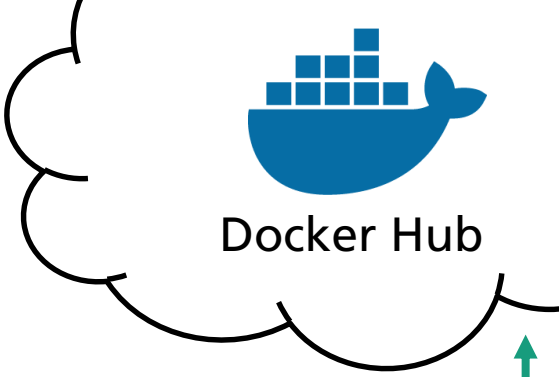



```
CMD /exec.sh
ADD myapp2.sh /exec.sh

RUN apt install -y curl

FROM ubuntu:17.10
```

dockerfile

Version

Dockerfile + docker-compose.yml




```
services:
  app1:
    build: myapp1/
    image: java-sth
    ports: 8080
  app2:
    build: myapp2/
    image: bash-tool
  importantdb:
    image: postgres:latest
```

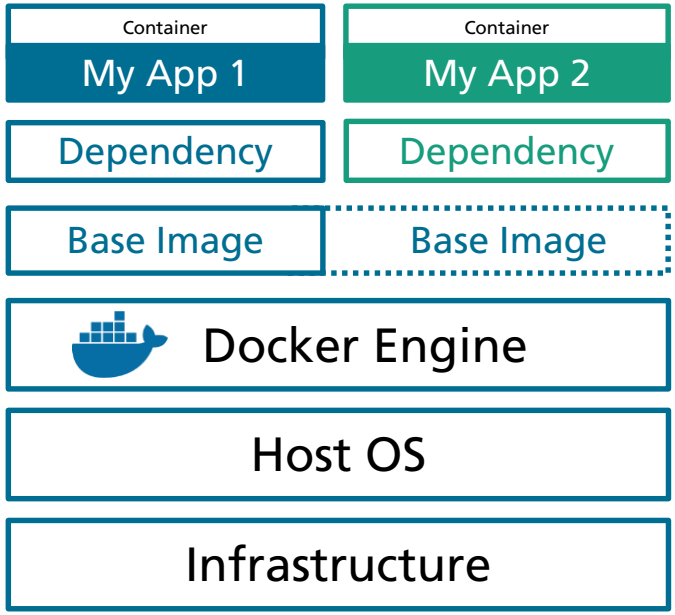
docker-compose

```
ADD myapp1.war
  /usr/local/tomcat/

FROM tomcat:alpine
```

Dockerfile

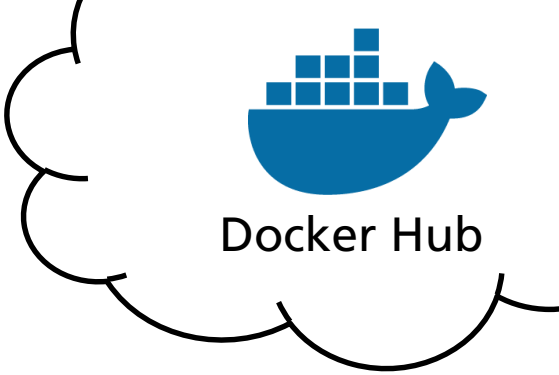
```
CMD /exec.sh
ADD myapp2.sh /exec.sh

RUN apt install -y curl

FROM ubuntu:17.10
```

Dockerfile



See also:
Lesson 3: Docker & Security - it's complicated

Dockerfile + docker-compose.yml

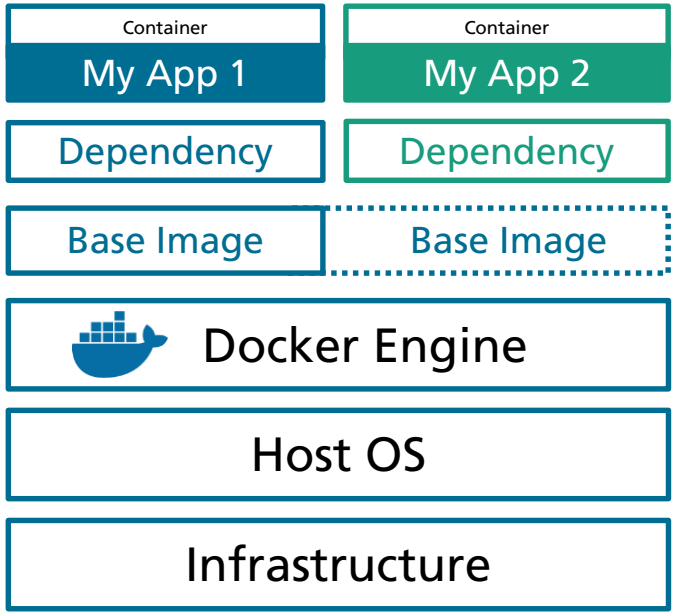



```
services:
  app1:
    build: myapp1/
    image: java-sth
    ports: 8080
  app2:
    build: myapp2/
    image: bash-tool
  importantdb:
    image: postgres:latest
```

docker-compose

```
ADD myapp1.war
  /usr/local/tomcat/

FROM tomcat:alpine
```

Dockerfile




```
CMD /exec.sh
ADD myapp2.sh /exec.sh

RUN apt install -y curl

FROM local:5000/ubuntu:17.10
```

Dockerfile

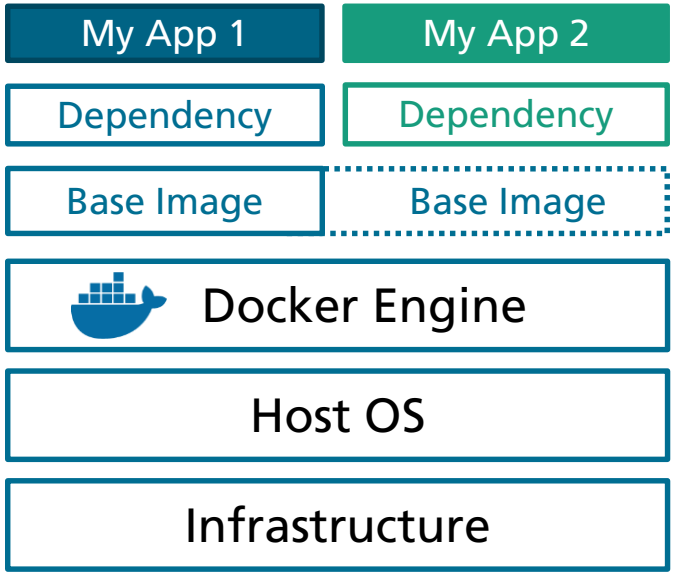
See also:
Lesson 3: Docker & Security - it's complicated

Use a local Docker registry by adding prefix to the image names

Docker in a Nutshell

- Shipping software in virtual images with all runtime dependencies

- **Layered**, caching filesystem that transparently reuses previous build steps / other images
- **Reproducible**, machine-readable instructions to produce images (dockerfile), ready for Continuous Integration and Continuous Delivery
- **Standardised** specification of networks, ports, external files and the interplay of several virtual images (docker-compose)



Why the Hype?

```
docker run -e MYSQL_ROOT_PASSWORD=my-secret-pw mysql:5.7.21
```

Downloads, installs,
configures and runs
MYSQL 5.7.21

Why the Hype?

Minimise Lead &
Ramp Up Times

Self-Contained-Services /
Microservices

„Cloud is the new normal“

Why the Hype?

Minimise Lead & Ramp Up Times	Developers define their own infrastructure (“as code”), seamless integration of third-party software and distribution channels, build automation can build, package, roll-out and execute the entire software stack
Self-Contained-Services / Microservices	Docker images are self-contained or clearly define their dependencies to external databases and resources
„Cloud is the new normal“	Docker is cloud native, Docker images are infrastructure agnostic

Why the Hype?

Developers are payed for:

- Developing software

Developers are not payed for:

- Re-inventing the wheel — Commodities
- Configure IDE
- Manage and download dependencies
- Package software
- Execute unit tests and integration tests — Build Management
- Distributing software packages
- Continuously monitor source control, build
- Determine the common build status ("Are we green?") — Continuous Integration
- Document operational environment
- Install, configure and start-up operational environment
- Install, configure and start-up runtime dependency software such as Databases — Containers
- Prepare, package, configure, distribute and install releases — Continuous Delivery

Docker helps significantly in automating this

Even if you don't use Docker - your competitor will!

Lessons learned from two years of Docker

Lesson 1: Understand the tools you use
Lesson 2: Meanings of “latest”
Lesson 3: Docker & Security – it’s complicated
Lesson 4: Docker is not done yet

Docker Basics

Lesson 5: Have a Docker-ready architecture
Lesson 6: Docker, the MVP enabler

Software-Architecture

Lesson 7: Use Build Infrastructure
Lesson 8: Streamline Developer’s Workflow

Processes

Lesson 9: Rethink Infrastructure

Infrastructure

Lesson 10: Refactor mind sets

Mindsets

Lesson 1: Understand the tools you use

- The art of writing a Dockerfile
  - Understand Docker
  - Follow Docker best practices [1]
  - dockerfile: Readability and duplication over fanciness
  - Don't use `docker commit`
- Question everything (including best practices)
- Do not solve future problems

Don't do this:

```
RUN APT UPDATE
RUN APT INSTALL -y openjdk, make

RUN cd /somepath
RUN make
```

↓ Cachable, readable

```
RUN APT UPDATE && \
    APT INSTALL -y openjdk, make

RUN cd /somepath && \
    make
```

↓ Cachable, readable, image size is minimised

```
ENV BUILD_DEPS make
ENV RUN_DEPS openjdk

RUN APT UPDATE && \
    DEBIAN_FRONTEND=noninteractive APT \
    INSTALL -y --no-install-recommends \
    $BUILD_DEPS, $RUN_DEPS

RUN cd /somepath && \
    make

RUN APT remove -y $BUILD_DEPS && \
    rm -rf /var/lib/apt/lists/
```

[1] https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/

Lesson 2: Meanings of “latest”

“latest LTS release“

`ubuntu:latest` vs. **`swaggerapi/swagger-ui:latest`**

“nightly“

The `ubuntu:latest` tag points to the "latest LTS", since that's the version recommended for general use. The `ubuntu:rolling` tag points to the latest release (regardless of LTS status).

https://hub.docker.com/_/ubuntu/

The v3.x tag series supports the Swagger / OpenAPI Specification 2.0 only. If you need 1.x support, please use the 2.x series.

- `latest` based on master
- `v3.0.5`
- `v2.2.9`

https://hub.docker.com/r/swaggerapi/swagger-ui/

- Docker images are **not** automatically tagged with “latest“
- “latest” has **no** common meaning across projects on Docker Hub
- “latest” is assumed if no explicit version is provided
- My personal preference: Use “latest“ as “stable release”
  - Use a $VERSION variable in Docker-Compose-files
  - Automatically produce docker-compose files with explicit versions
  - Keep Maven POM version == Docker version tag

Lesson 3: Docker & Security – it's complicated

munity images shows little difference from that of all community images. In addition, more than 80% of both types of images have at least one high severity level vulnerability.

About 50% of both community and official images have not been updated in 200 days, and about 30% of images have not been updated in 400 days. There is some difference in the percentage of more frequently

Shu, R., Gu, X., & Enck, W. (2017). A Study of Security Vulnerabilities on Docker Hub. In Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy - CODASPY '17 (pp. 269–280). https://doi.org/10.1145/3029806.3029832

„Docker is a completely new way to ship security flaws!"

- The Internet

Docker Hub is a central repository for Docker developers to pull and push container images. We performed a detailed study on Docker Hub images to understand how vulnerable they are to security threats. Surprisingly, we found that more than 30% of official repositories contain images that are highly susceptible to a variety of security attacks (e.g., shellshock, heartbleed, poodle, etc.). For general images (images pushed by docker users, but not explicitly verified by any authority) this number jumps up to ~40% with a sampling error bound of 3%. In order to perform this study, we pulled images from Docker Hub and inspected packages and their versions installed in them. We then used the information from the Mitre, NVD (National Vulnerability Database) and Linux distro-

Gummaraju, J., Desikan, T., & Turner, Y. (2015). Over 30% of Official Images in Docker Hub Contain High Priority Security Vulnerabilities. Https://Banyanops.Com, (May), 1–6. Retrieved from https://banyanops.com/blog/analyzing-docker-hub/

Lesson 3: Docker & Security – it's complicated

- With the base image concept, large amounts of binaries are pulled and executed automatically from the internet

Does this comply to BMWi's Merkblatt f. die Behandlung von VS-NfD?

- The Docker daemon has root-level permissions and communicates directly with the host OS kernel
- Attack vector syscalls – can SELinux, S-Virt, Seccomp, … help?

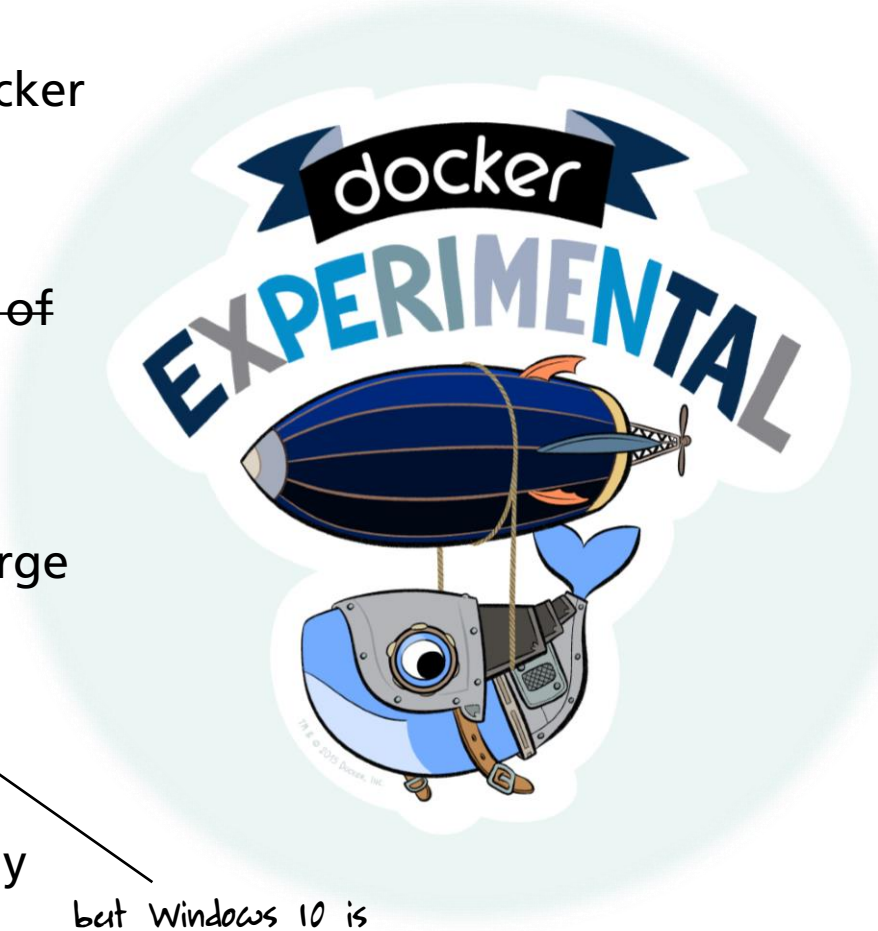
IBM, Xen, Huawei: Virtual Machines

- What we can do:
  - Have a local custom registry with approved base images
  - Install update for software within the Docker images
    -> regularly rebuild all layers without using the cache

More about this in:
Lesson 7: Use Build Infrastructure

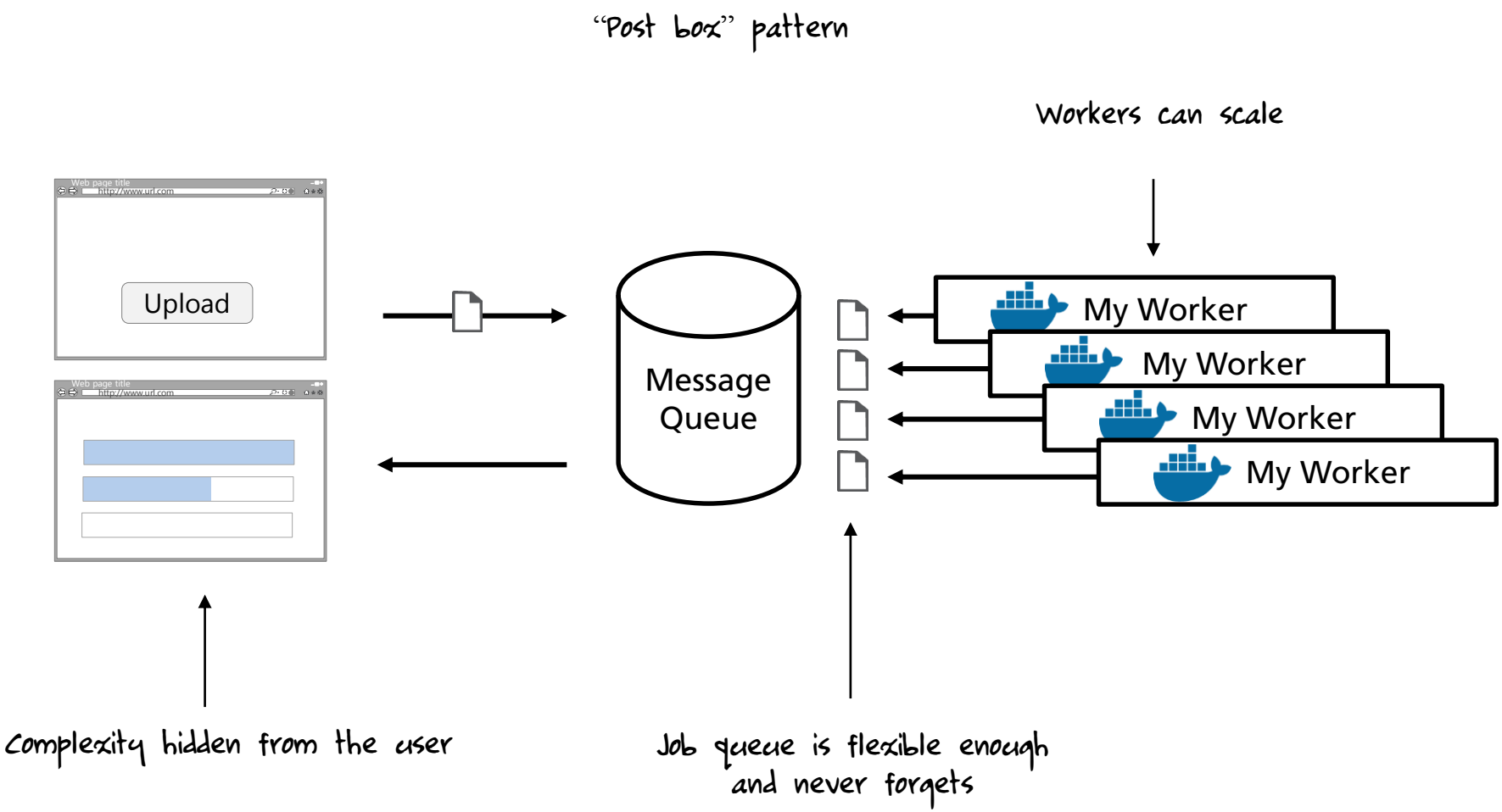
Lesson 4: Docker is not done yet

- Docker is under heavy development
- #1 information source: GitHub issue tracker
- Docker is not trivial, requires cross-technology knowledge
- Missing features (such as: ~~management of dangling volumes~~, no data volume management)
- Missing concepts (such as: runtime dependency between containers and large data assets)
- Painful use of Docker on Windows 7
  - Non-native Docker engine
  - Windows Shell / Linux Bash dichotomy



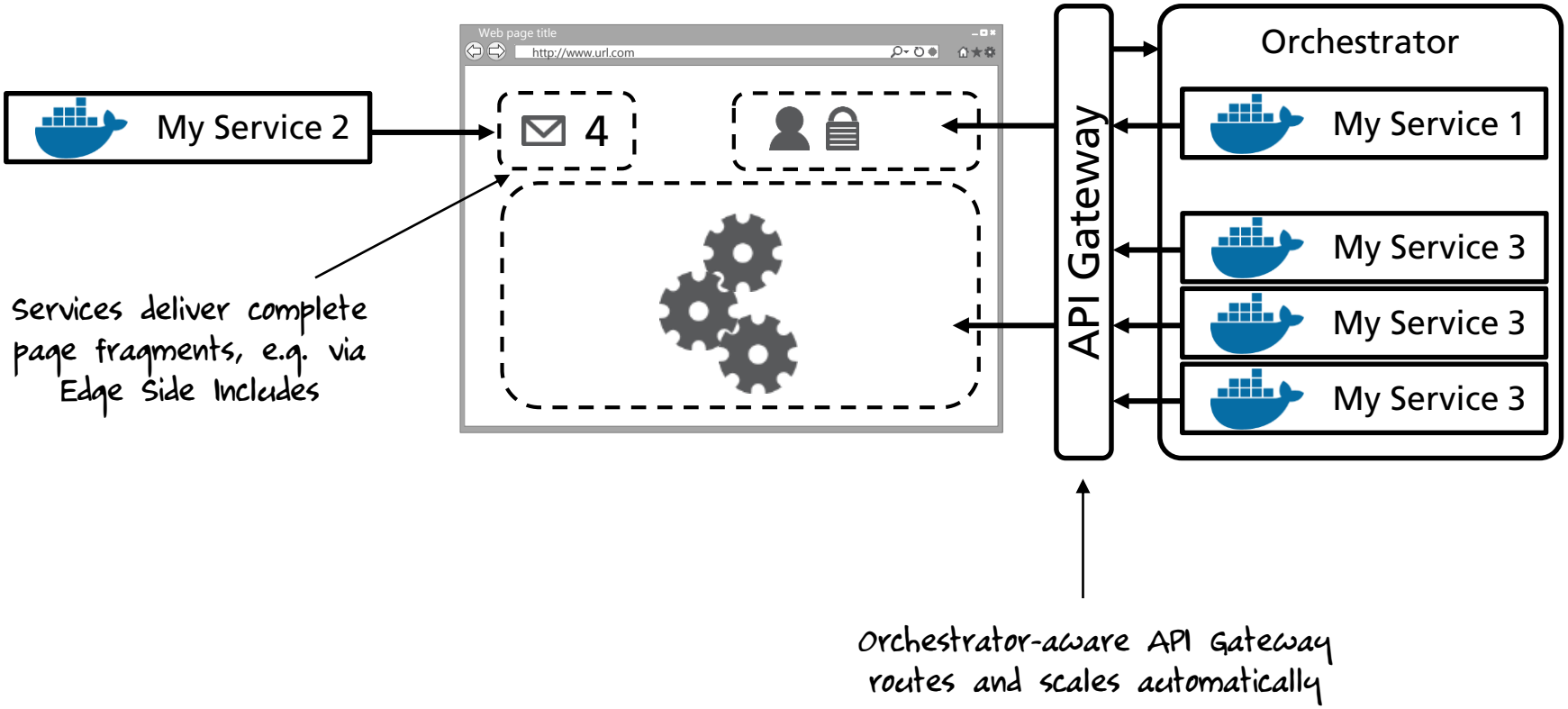
but Windows 10 is
not VS-NfD-ready yet...

Lesson 5: Have a Docker-ready software architecture



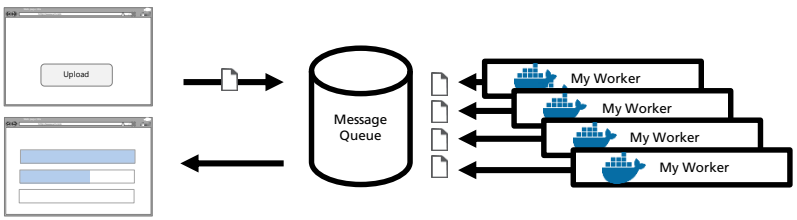
Lesson 5: Have a Docker-ready software architecture

Dynamic Web Content Assembly / „Chatty Architecture“

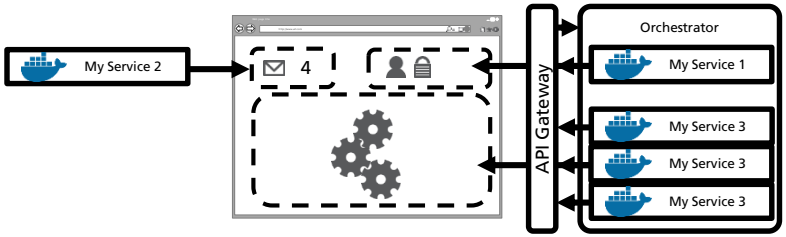


Lesson 5: Have a Docker-ready software architecture

"Post box" pattern



Dynamic Web Content Assembly / „Chatty Architecture"

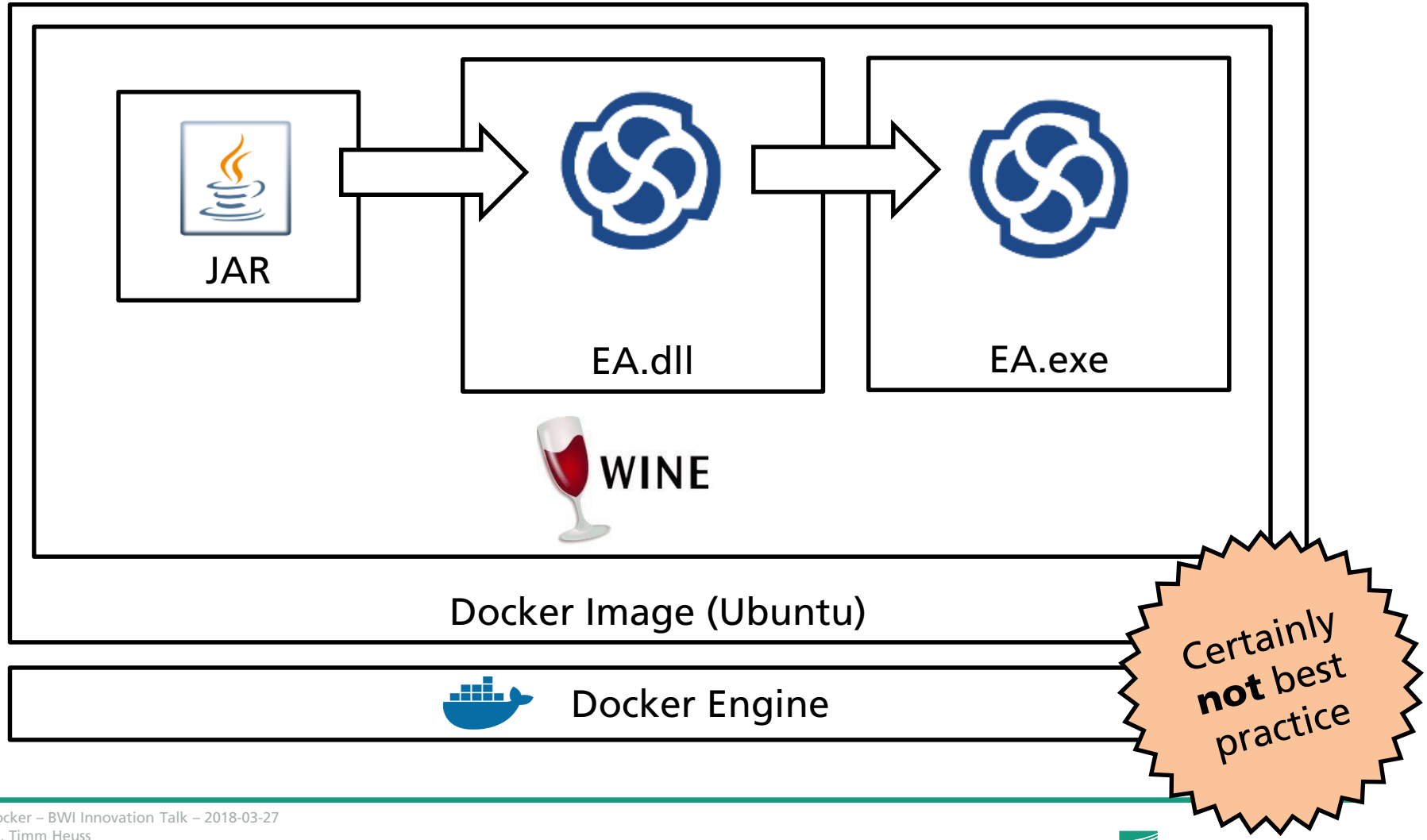


- Design the scalability of your application alongside Docker's capabilities
- Beware of a cargo cult: **Docker is not a "lift and shift" technology!**
- Porting monolithic / legacy / GUI applications might not be feasible
- However: Scalability is not the only reason to use Docker

A fully documented software stack as code
Being able to package, rollout and execute

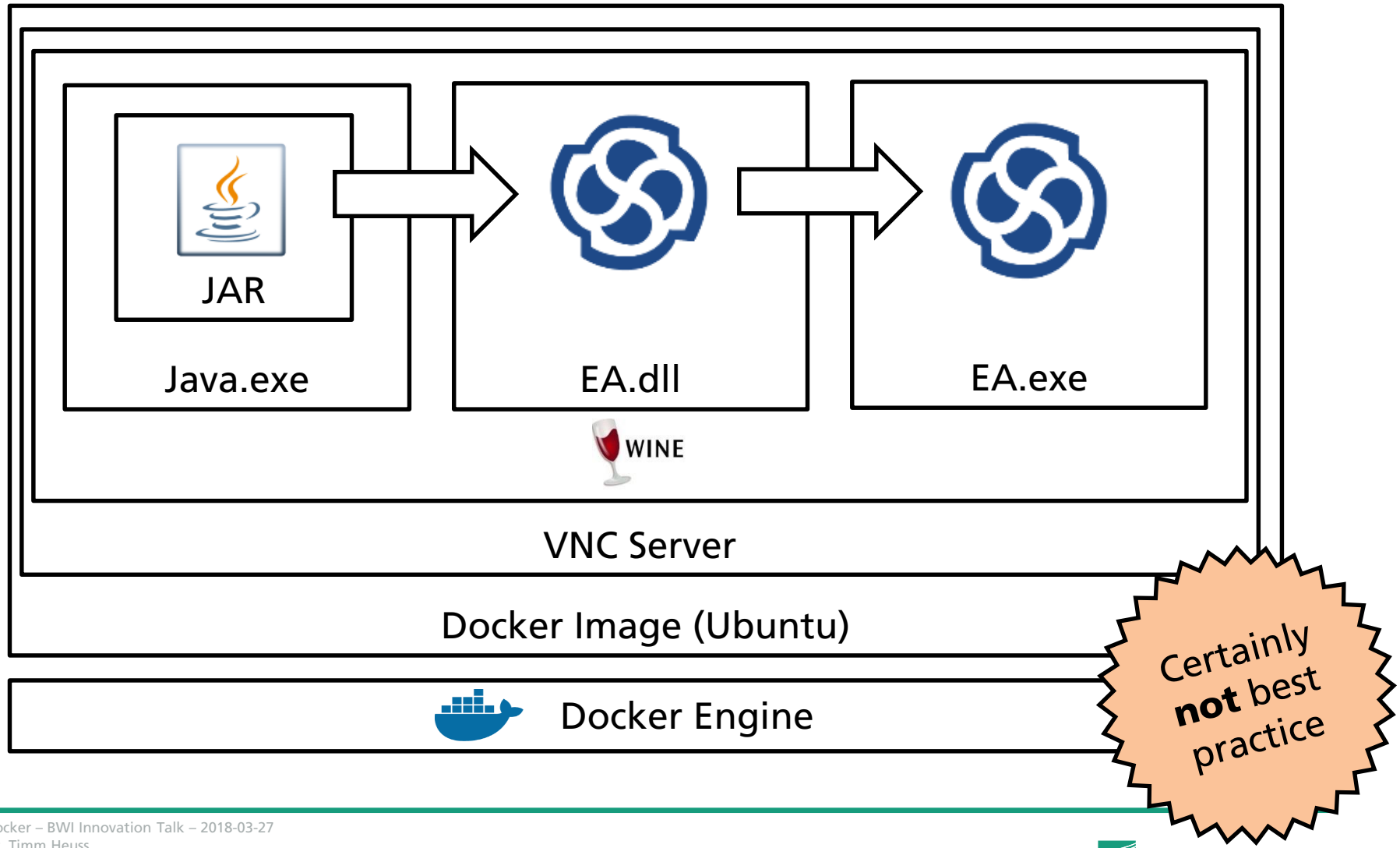
Well this is also possible…

Using a Windows-Java-native DLL and Windows application in Docker

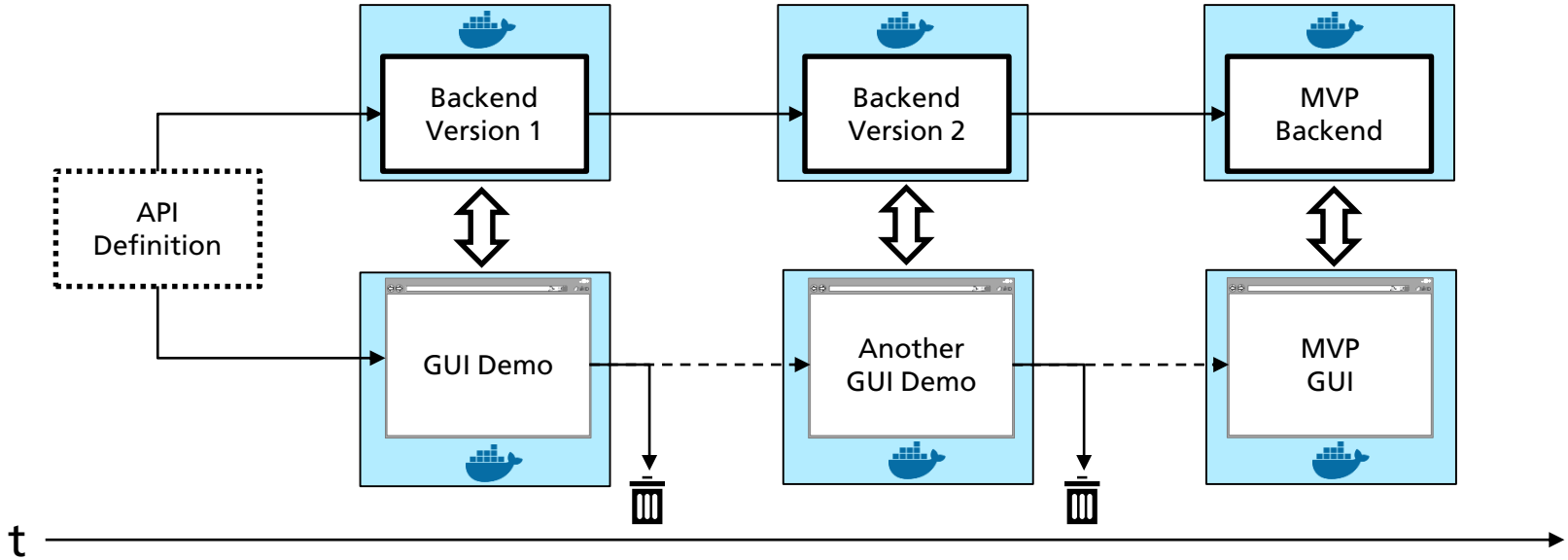


… or this…

Using a Windows-Java-native DLL and Windows GUI application in Docker



Lesson 6: Docker the MVP-Enabler



- Docker is a nice contribution to the ecosystem for MVPs and Rapid Prototyping
  - Contract first ← e.g. with tools like Swagger
  - Generate backend and frontend stubs ← Java Code Gen produces JAX-RS compliant stub
  - Package in Docker ← Maven plugin detects JAX-RS, builds Docker image
  - Let the backend evolve
  - Throw-away demos for customer feature demonstrations
- Thanks to Docker, all demos are kept available, infrastructure might change entirely

Lesson 7: Use Build Infrastructure

Things developers want to do:

- Developing software

Things developers don't want to do:

- Re-inventing the wheel
- Configure IDE
- Manage and download dependencies
- Package software
- Execute unit tests and integration tests
- Distributing software packages
- Continuously monitor source control, build
- Determine the common build status ("Are we green?")
- Document operational environment
- Install, configure and start-up operational environment
- Install, configure and start-up runtime dependency software such as Databases
- Prepare, package, configure, distribute and install releases

Commodities

Build Management

Continuous Integration

Containers

Continuous Delivery

Build Infrastructure helps automating this

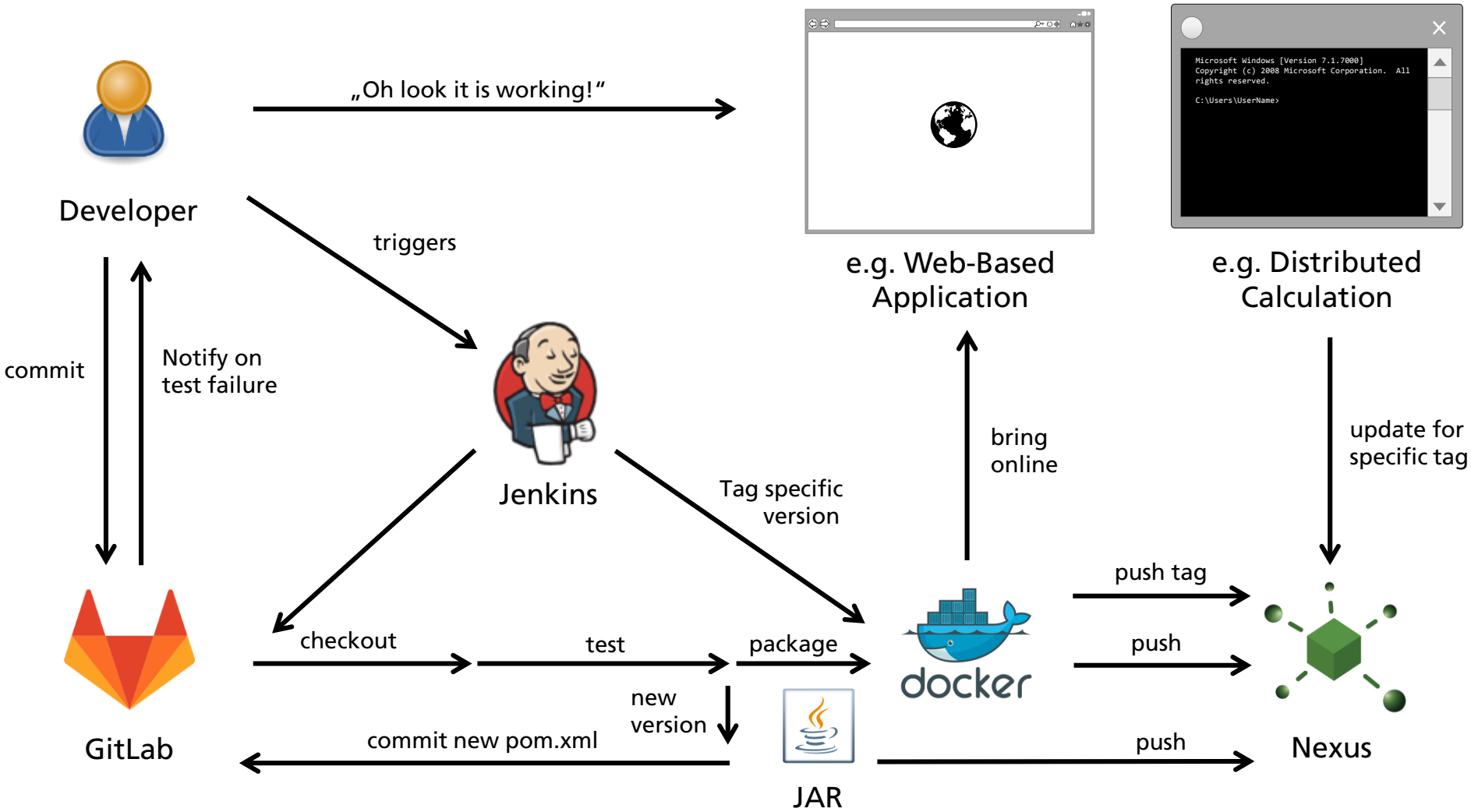
Lesson 7: Use Build Infrastructure

Jenkins, GitLab Pipelines, etc...

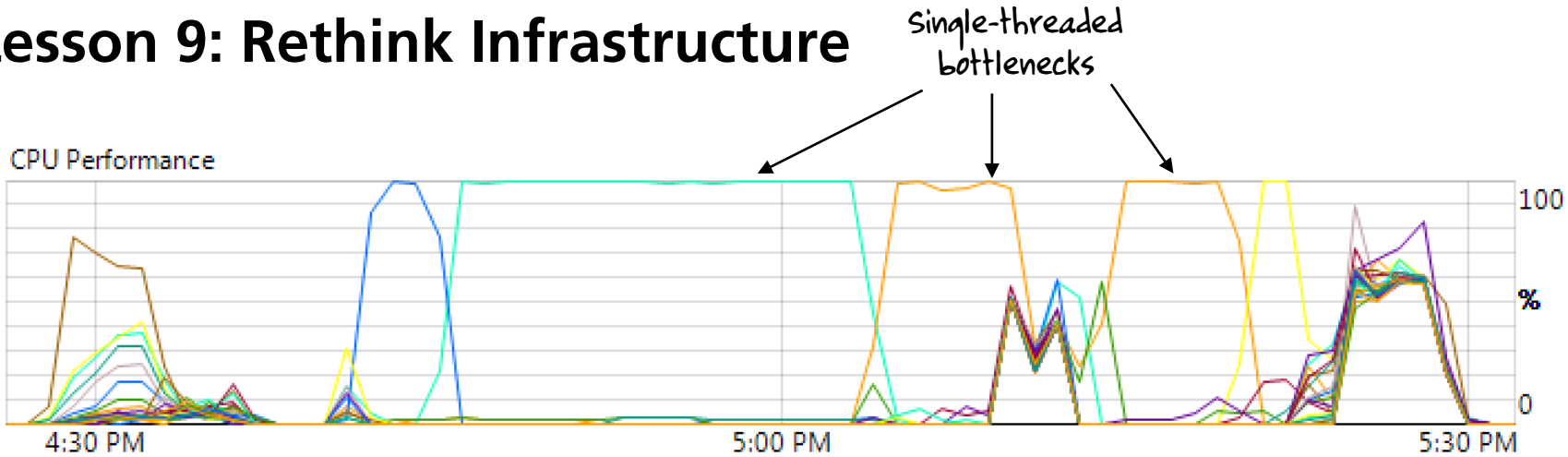
- Use Build infrastructure to
  - Continuously build Docker images
  - Do the testing, indicate results
  - Push images / roll-out
  - Launch complete software stack
  - Overnight complete rebuild of all layers

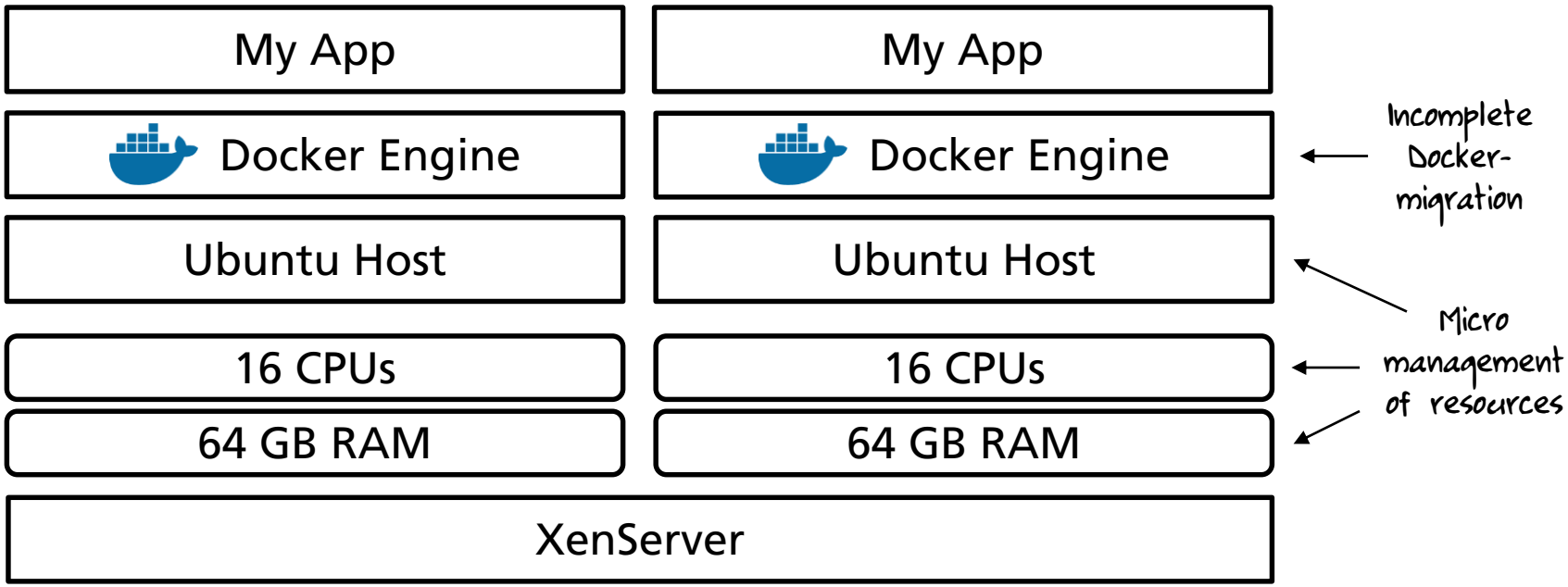
- Automation means:
  standardisation of build and deployment processes

Lesson 8: Streamline Developer's Workflow



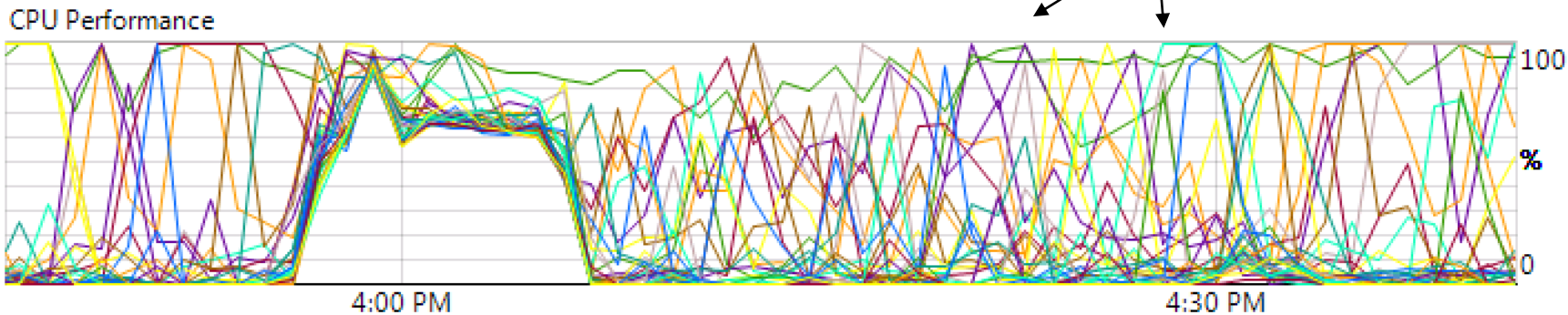
Lesson 9: Rethink Infrastructure

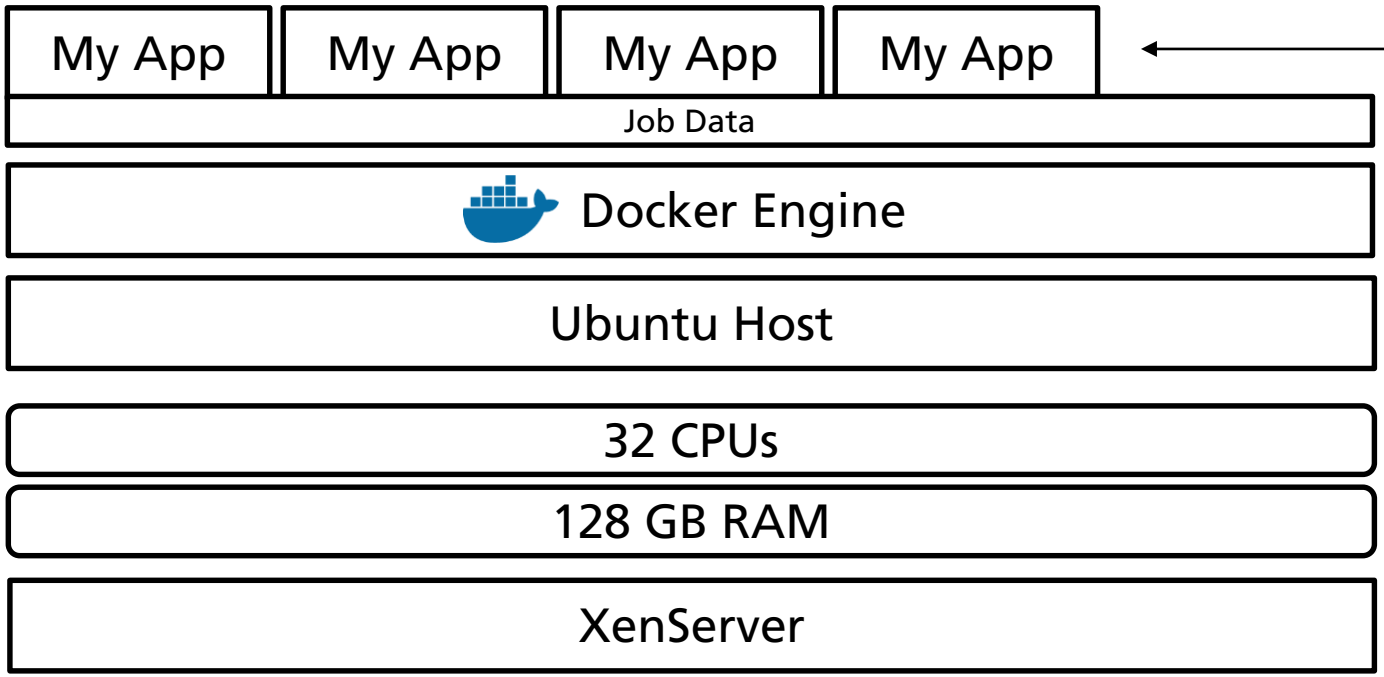




Lesson 9: Rethink Infrastructure

More efficient use of (rented) resources





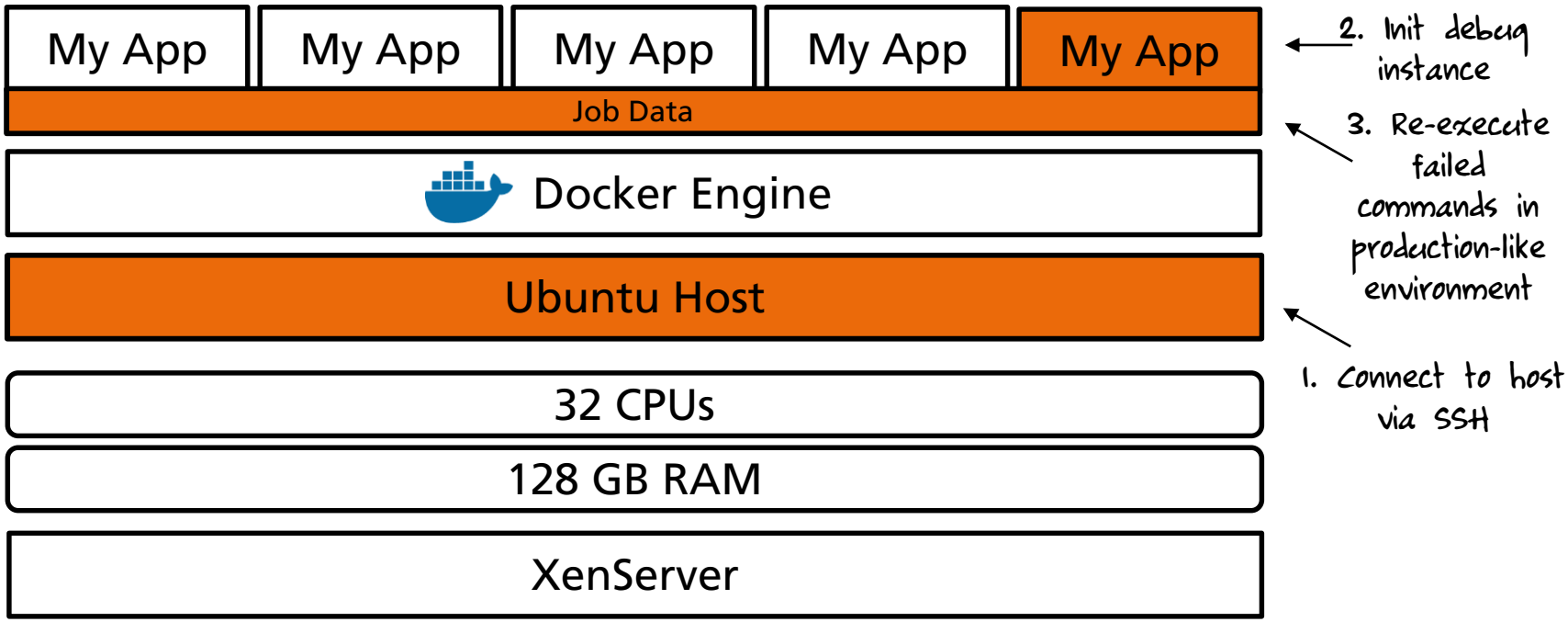
Scale on demand

Common persistent Job data storage

All resources are available to Docker

Lesson 9: Rethink Infrastructure

Debugging production in 3 simple steps



Lesson 10: Refactor mind sets

"Machines should work, Humans should think"

- Architectural changes, building and shipping is daily business
- Promote use of standard tools over custom solutions
- Promote a culture of documentation

Two main enemies:
fears & secrets

- The one who deploys (or tells systems to do so) defines the rules for collaboration
- Tech. project management / "Operations person" / everybody does DevOps?

New Role?

- "Cool technology" vs. young computer science graduates
  - Technological product roadmap is impacted by the level of knowledge of graduates in the field

Lessons learned from two years of Docker

Lesson 1: Understand the tools you use
Lesson 2: Meanings of “latest”
Lesson 3: Docker & Security – it’s complicated
Lesson 4: Docker is not done yet

Docker Basics

Lesson 5: Have a Docker-ready architecture
Lesson 6: Docker, the MVP enabler

Software-Architecture

Lesson 7: Use Build Infrastructure
Lesson 8: Streamline Developer’s Workflow

Processes

Lesson 9: Rethink Infrastructure

Infrastructure

Lesson 10: Refactor mind sets

Mindsets

Interesting readings and tools

Must-read: Docker Best Practices

https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/

Advanced: Good container practices

https://l0rd.github.io/containerspatterns/

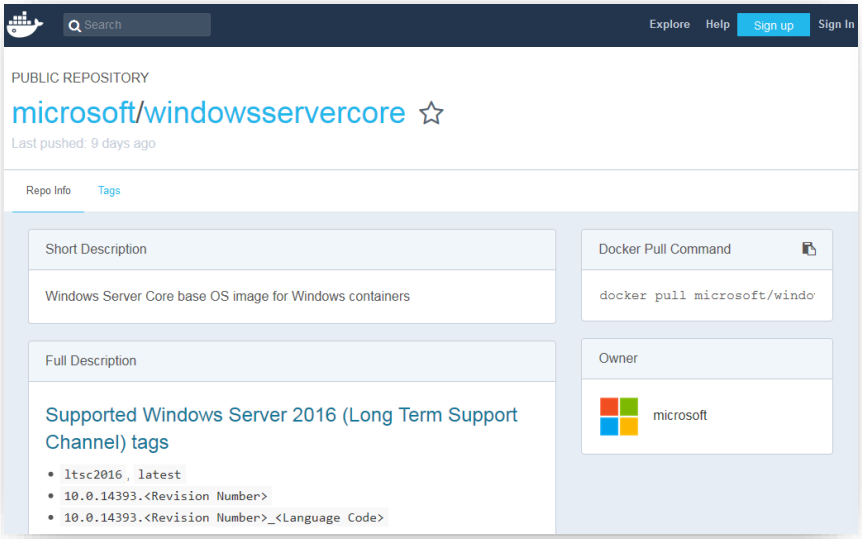
Graphical User Interfaces for Docker Engine Management

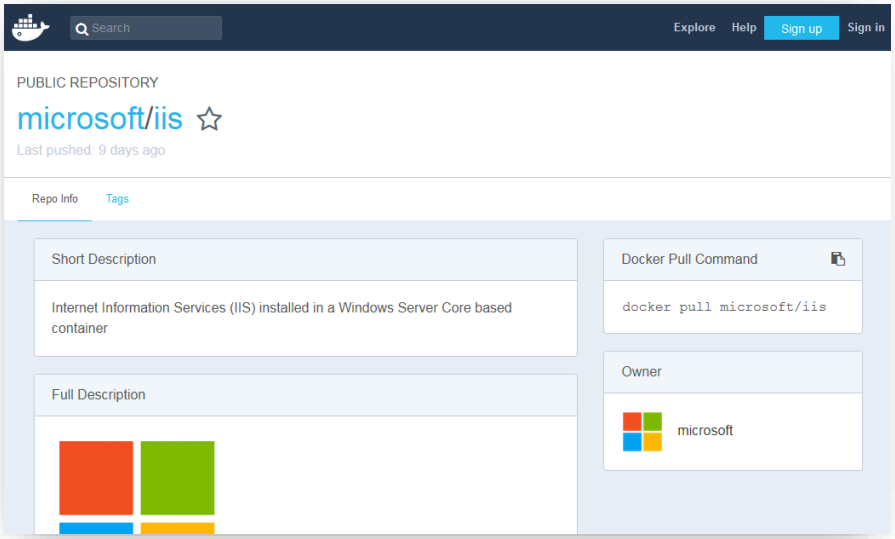
UI for Docker, Portainer

Docker-Container Auto-Update

Watchtower

What‘s next: Windows-native Software in Docker





- Future versions of Docker:
  Run Windows-native and Linux-native containers on one Windows host

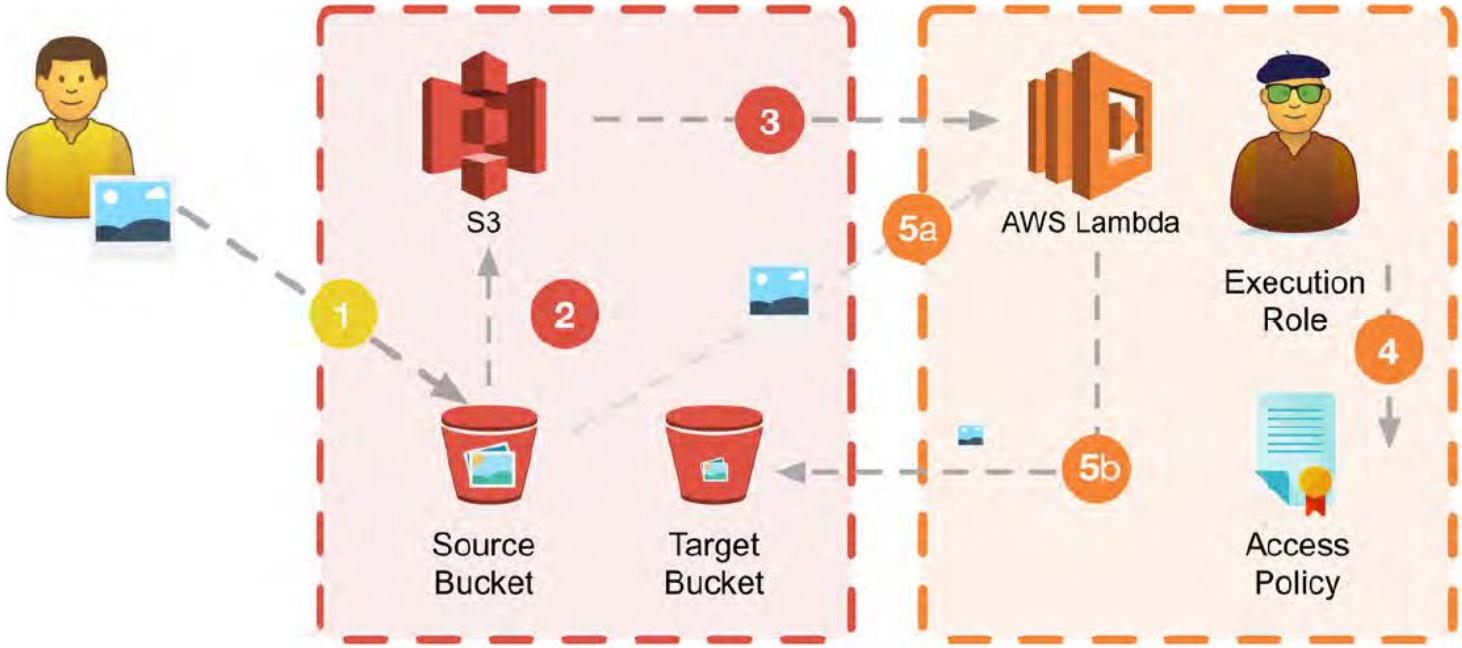
What's next: Streamlined Orchestrator Workflows

for example...

- Build ecosystems for the automatic creation of microservices and for debugging them (e.g. fabric8)
- Complete toolchain to cover the Plan-Code-Verify-Package-Release-Configure-Monitor lifecycle (e.g. GitLab)

What‘s next: Serverless

- Don’t pay idle – “run code, not a server!”
- Complex, event-based execution, hard to debug, often highly proprietary
- Especially useful when deployables must scale rapidly
- Scenarios: Thumbnail creation, Event Streaming, Resource Optimisation, …



Source: W-Jax 2017: Lars Röwekamp – Cloud-Workshop – Vom Cloud-Muffel zum Cloud-Native in sechs Stunden

Thank You

These slides are available online:

Dr. Timm Heuss
Fraunhofer Institute for Communication,
Information Processing and Ergonomics

Fraunhoferstraße 20
53343 Wachtberg
Germany

Timm.Heuss@fkie.fraunhofer.de
+49 228 9435-154

https://github.com/heussd/docker-bwi-innovation-talk