



# **Using Docker to deliver Java Enterprise Applications one year later...**

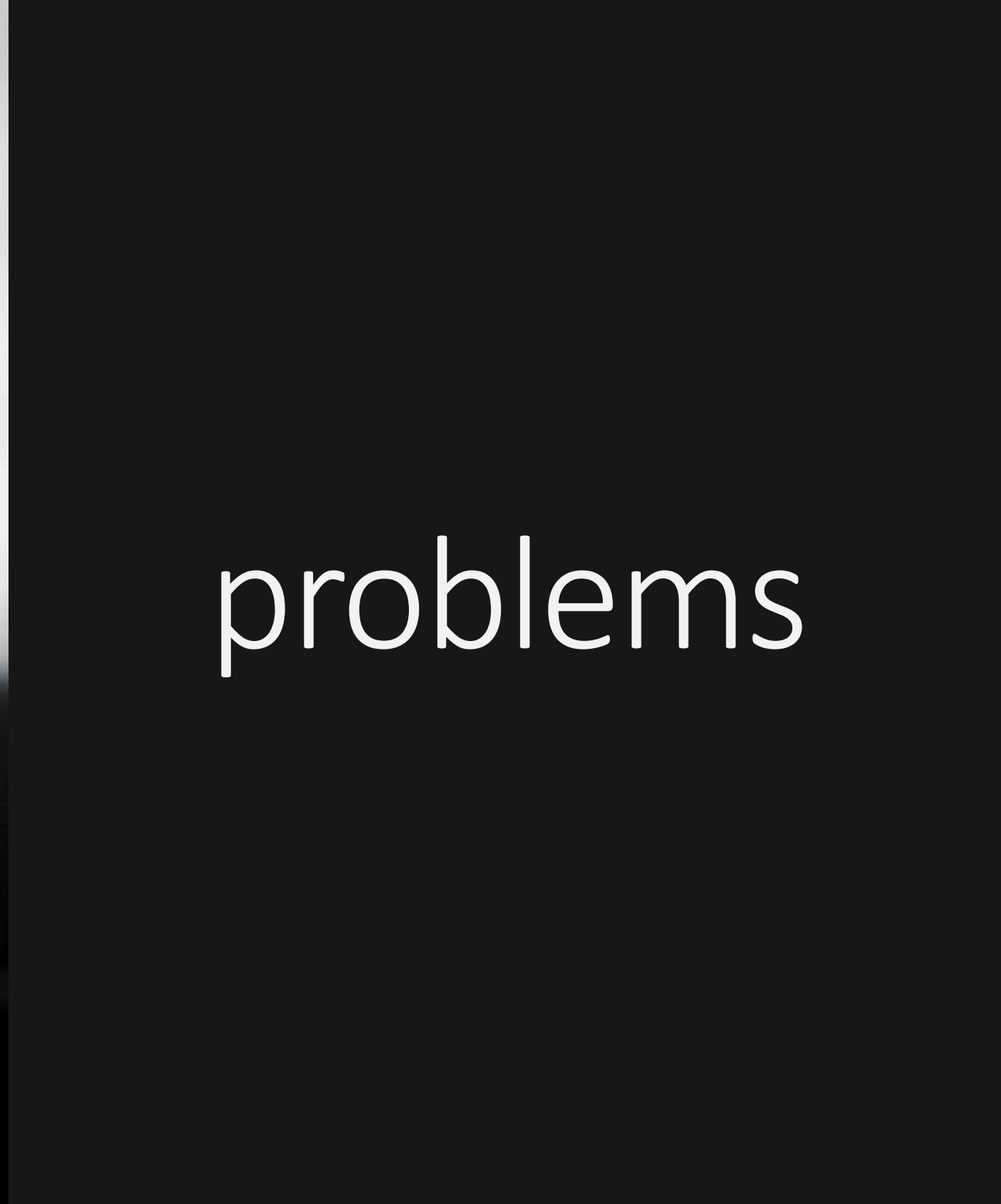
**Petyo Dimitrov**

**May 26-27 '16, Sofia**



# Agenda

- What problems we had?
- Docker introduction
- How to "dockerize" a sample application?
- Docker technologies
- What about VMs?
- Lessons learnt....





A black and white photograph showing a worker in a hard hat and light-colored shirt standing on a wooden cart. The cart is heavily overloaded with stacks of cardboard boxes, some of which are falling off the sides. The scene is set in a large industrial building with a high ceiling and structural beams. The floor is made of cobblestones. The text 'inadequate packaging of application artifacts' is overlaid in white on a semi-transparent dark band across the middle of the image.

# inadequate packaging of application artifacts





inconsistencies across *environments*





high cost  
supporting multiple static environments



lack of  
freedom  
experimenting with  
new languages,  
technologies and  
frameworks





# Sea-free analogy





# Developers

- care about apps
- put stuff in containers
  - code & data
  - libraries
  - applications



# Operations

- care about containers
- work with containers
  - logging & monitoring
  - networking
  - scaling



# High level view of a container

- **it is like a lightweight Virtual Machine**
- it provides:
  - own process space
  - own network interface
  - running stuff as root



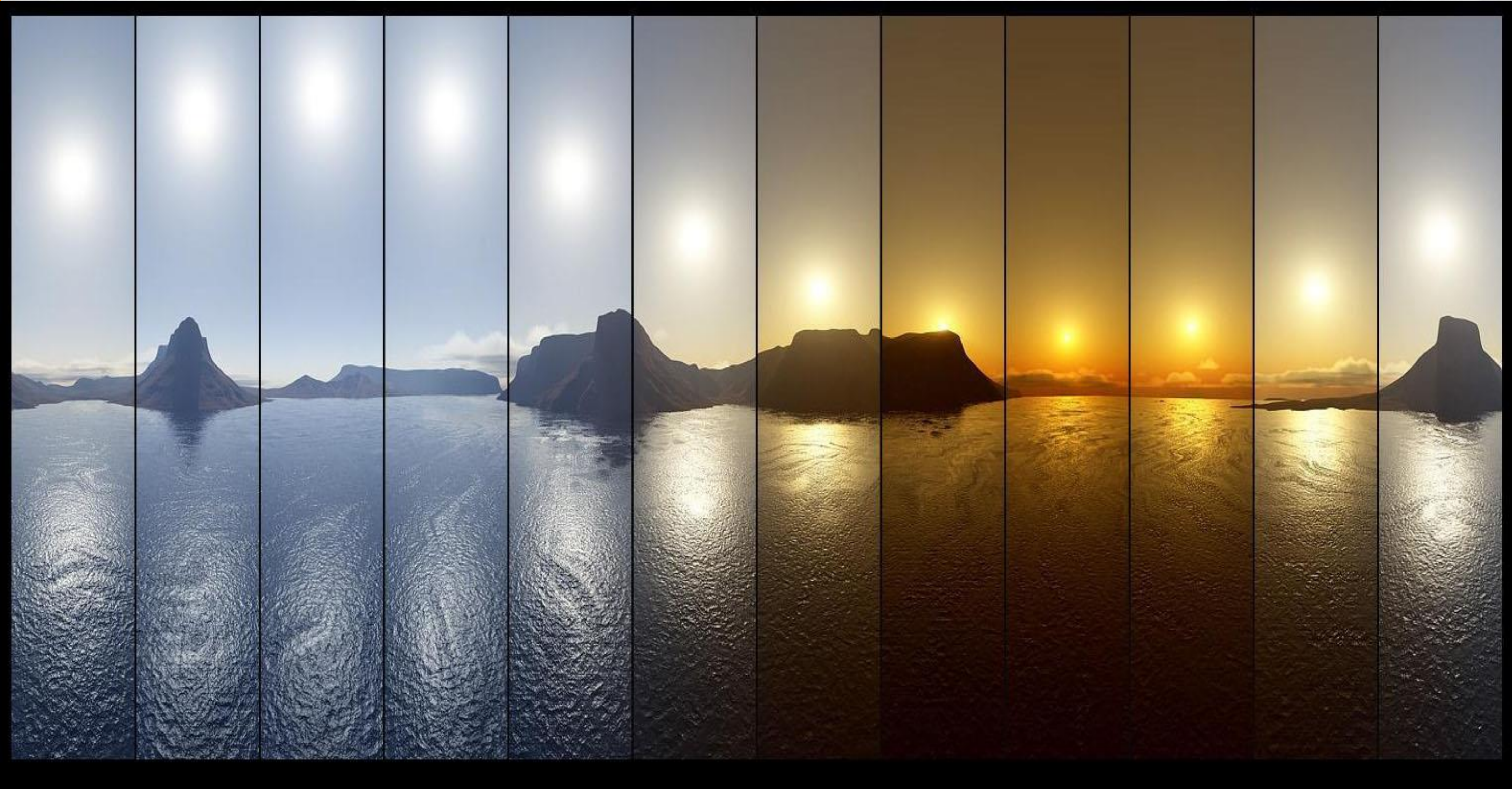
# Low level view of a container

- **container = "process in a box"**
- shares kernel with host → boots faster
- processes run directly on the host
- there is no device emulation
- none or little CPU, memory, network and I/O overhead





# Fast forward one year





# Demonstration

<https://github.com/petyodimitrov/spring-music.git>

<https://github.com/petyodimitrov/app-setup.git>

<https://github.com/petyodimitrov/ci-setup.git>





# Used technologies

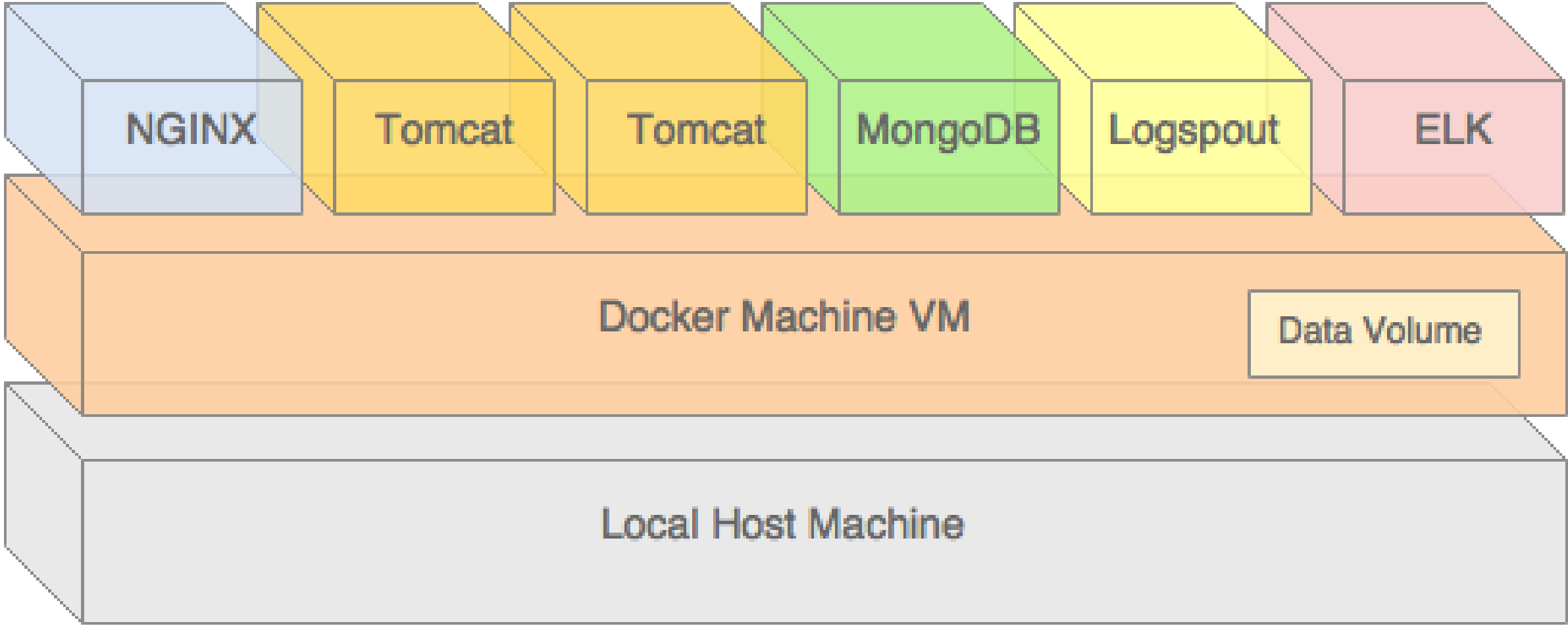


***maven***





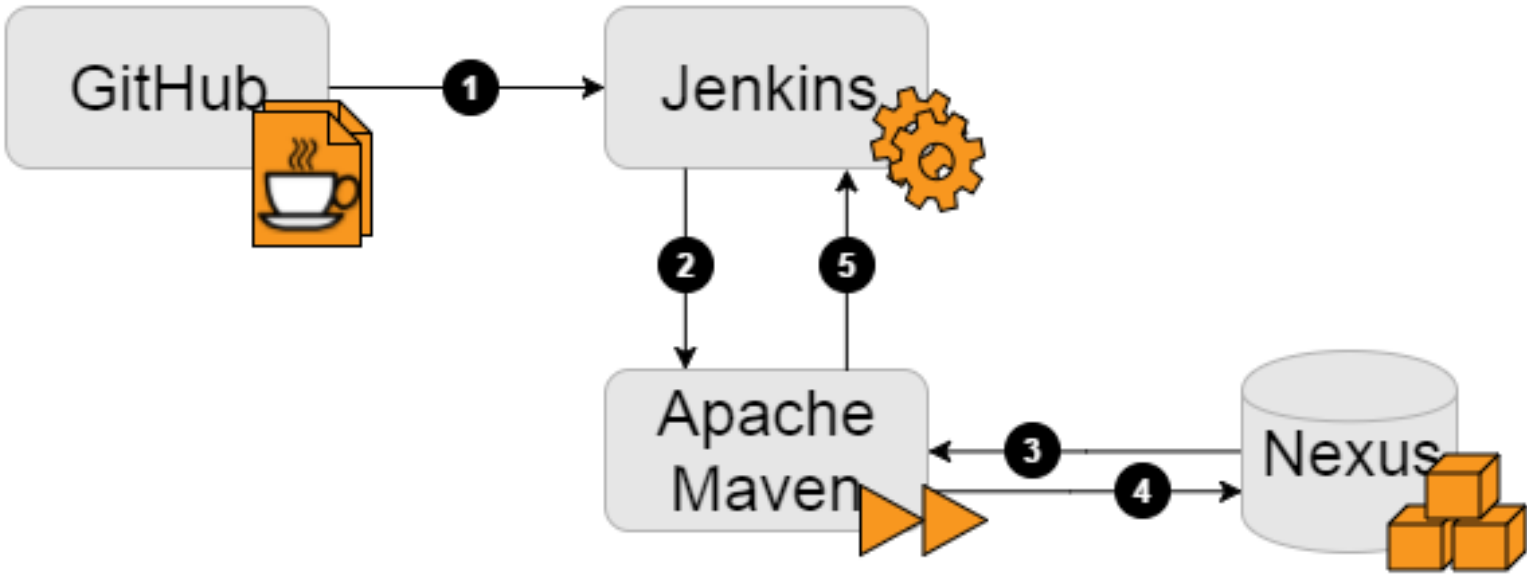
# Runtime view of containers







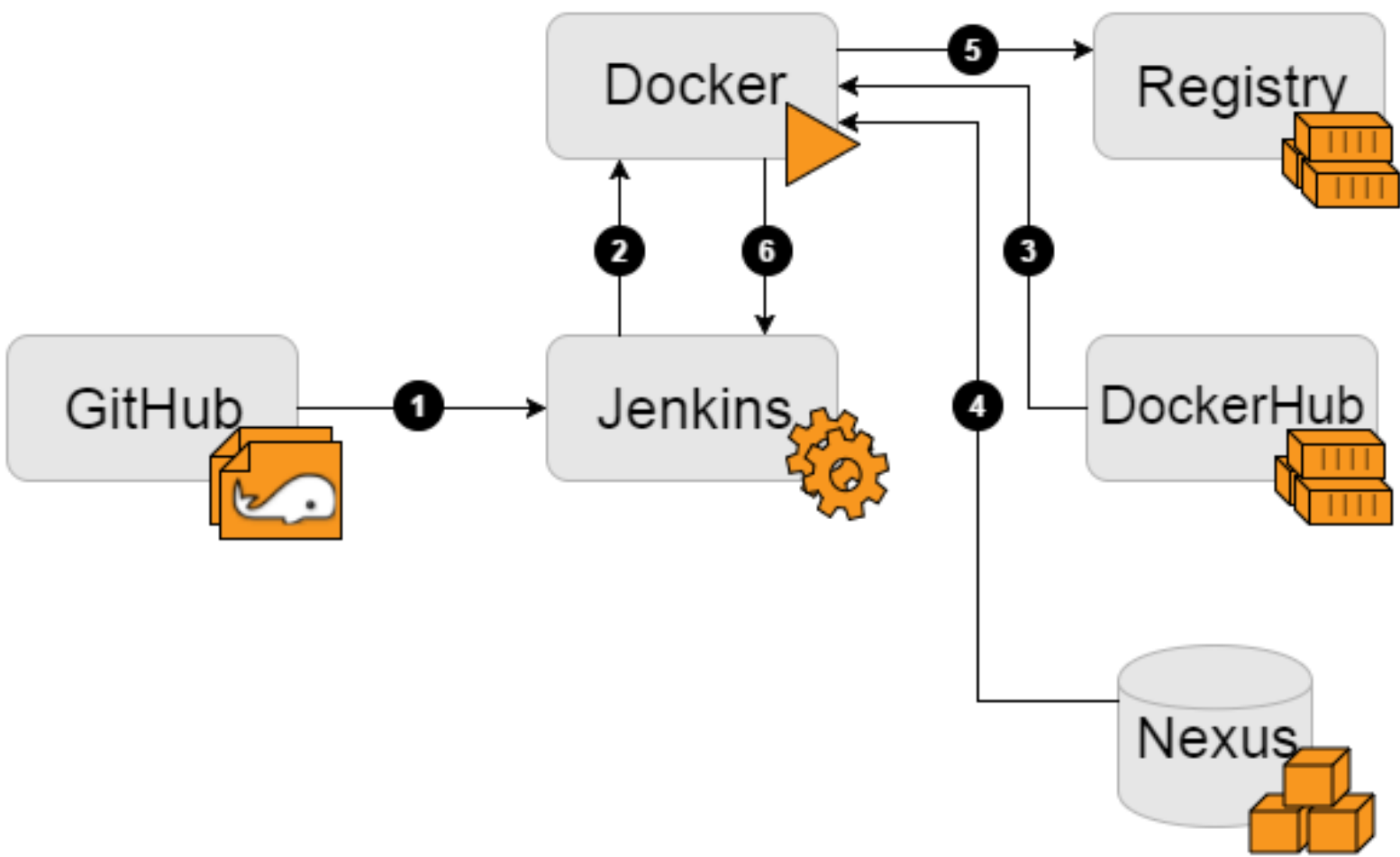
# Standard Java application build





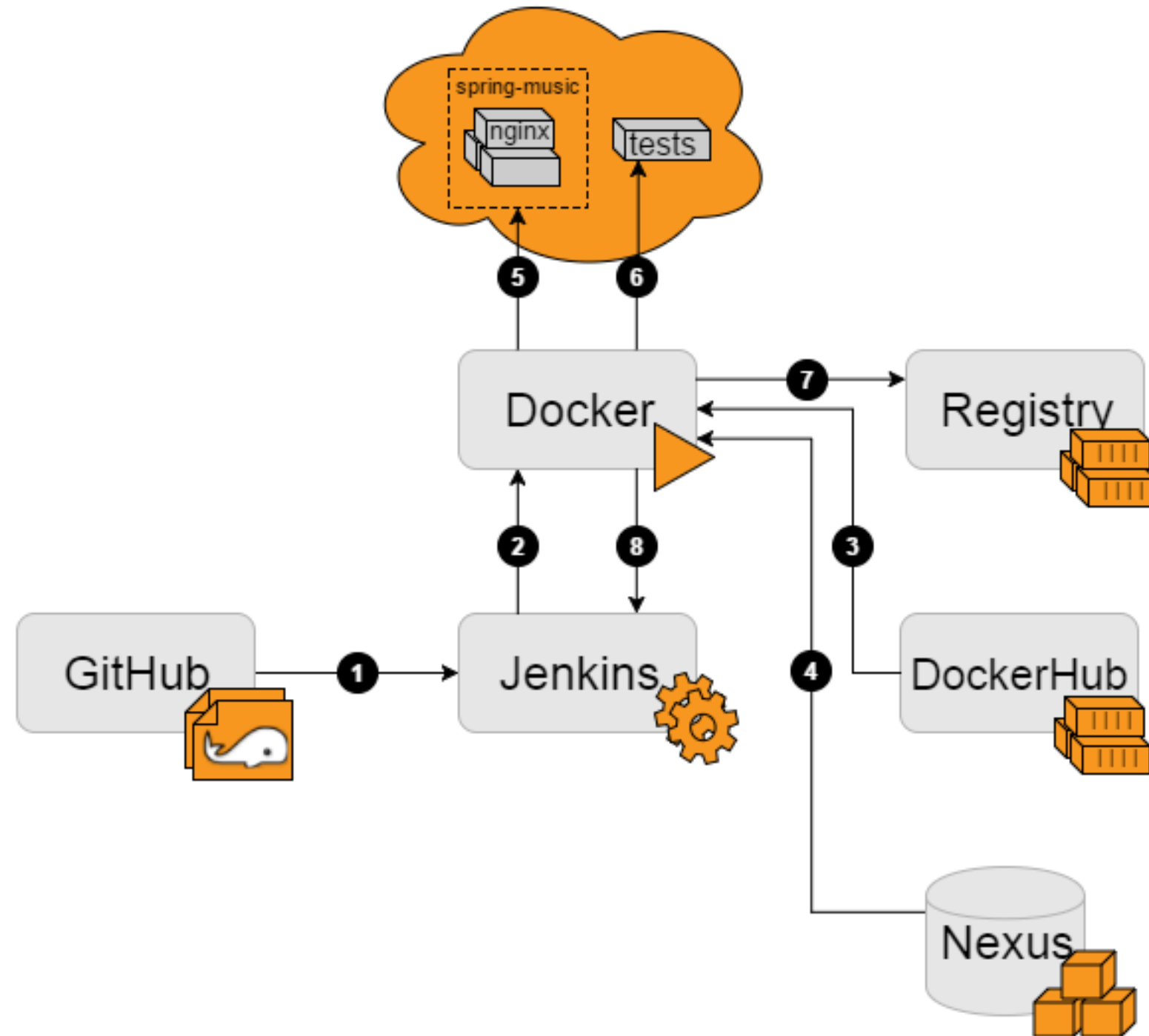


# Docker containers build





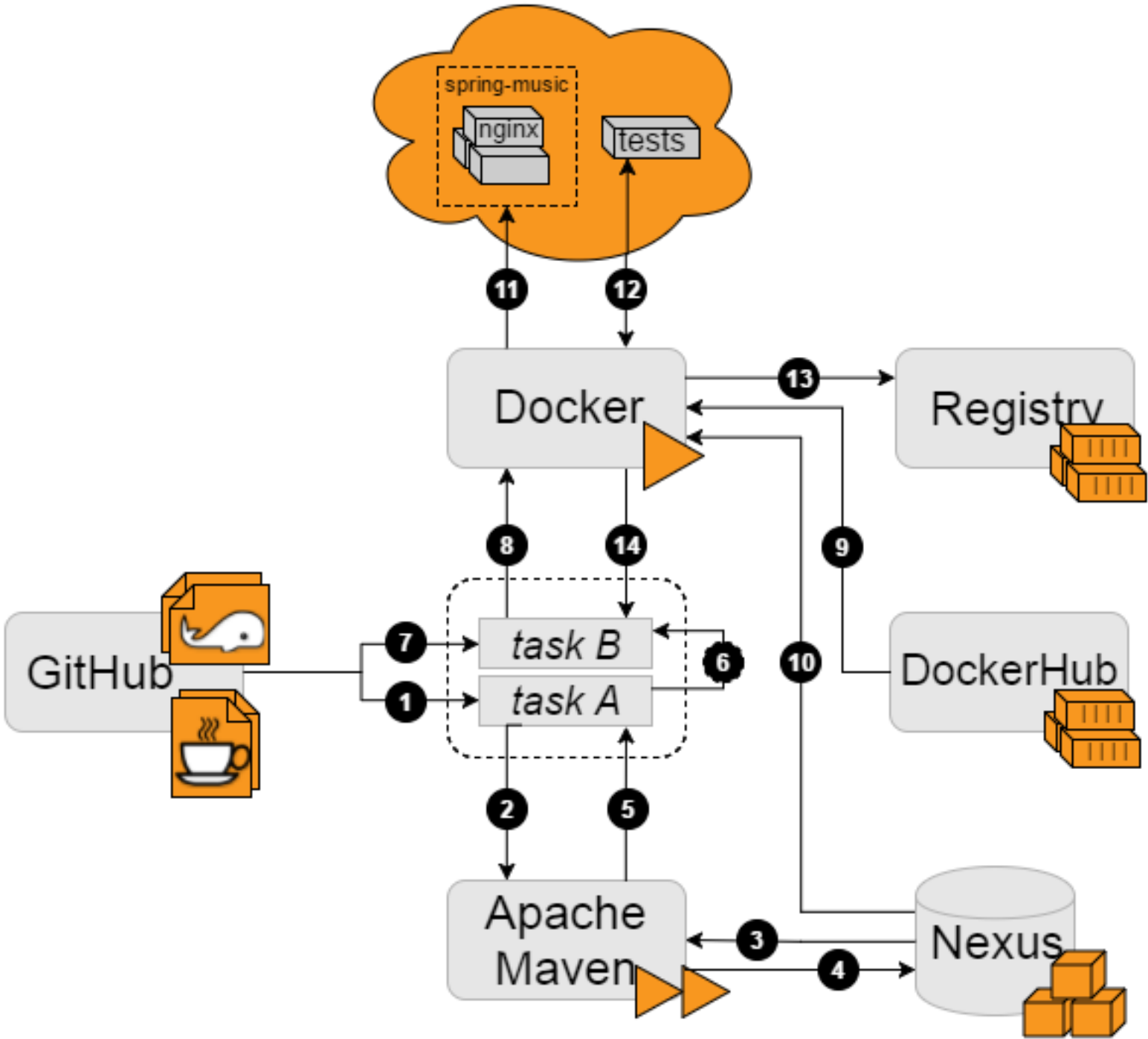
# Adding system testing







# Combining all the parts



















# Demonstration

Spring Music - Google Chrome

Spring Music

192.168.99.100

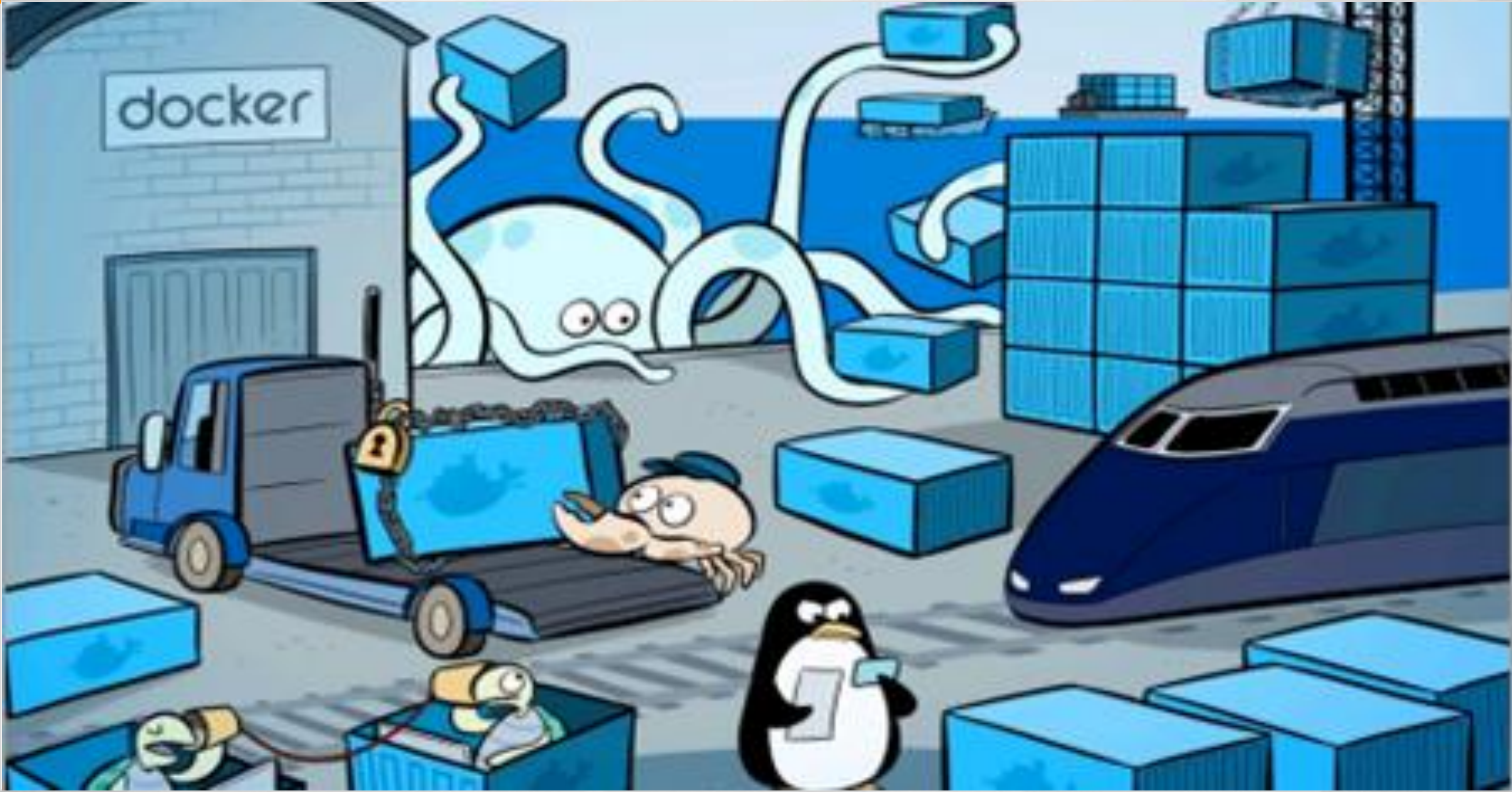
Spring Music  

Led Zeppelin Led Zeppelin 1969 Rock 	IV Led Zeppelin 1972 Rock 	Thriller Michael Jackson 1982 Pop 	Folk Singer Muddy Waters 1964 Blues 
Nevermind I Found It Nirvana 1993 Rock 	Synchronicity Police 1983 Rock 	A Night At The Opera Queen 1975 Rock 	King of the Delta Blues Robert Johnson 1961 Blues 
Couldn't Stand The Weather Stevie Ray Vaughan 1984 Blues 	Texas Flood Stevie Ray Vaughan 1983 Blues 	Pet Sounds The Beach Boys 1966 Rock 	Rubber Soul The Beatles 1965 Rock 



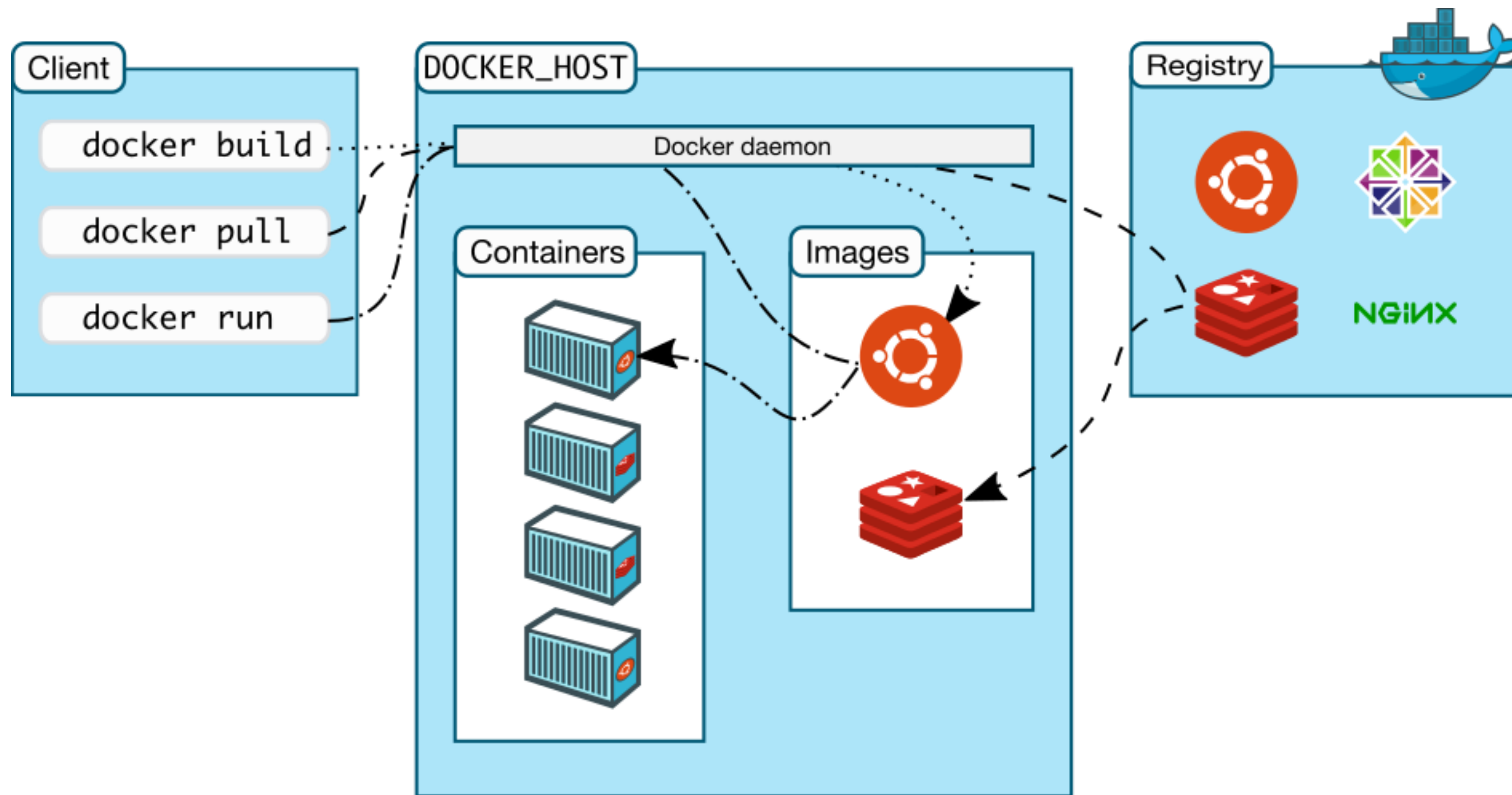


# Docker "ecosystem"





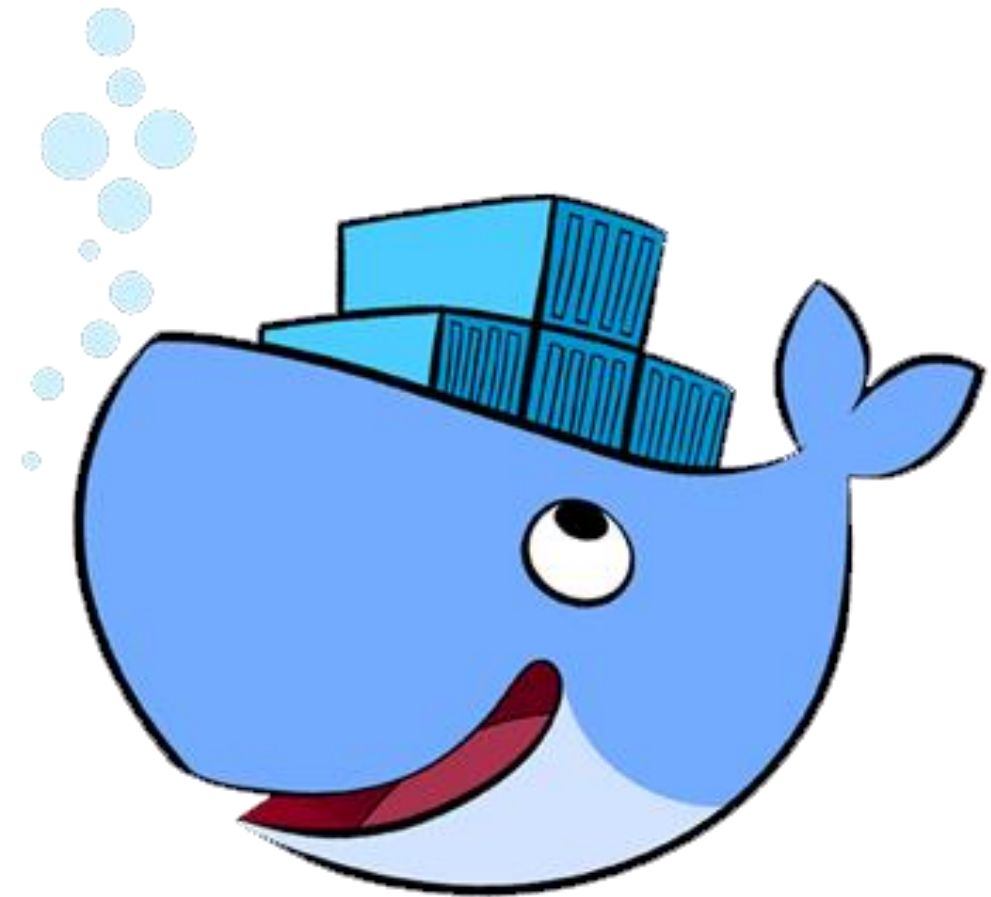
# Client-Server model





# Docker Engine

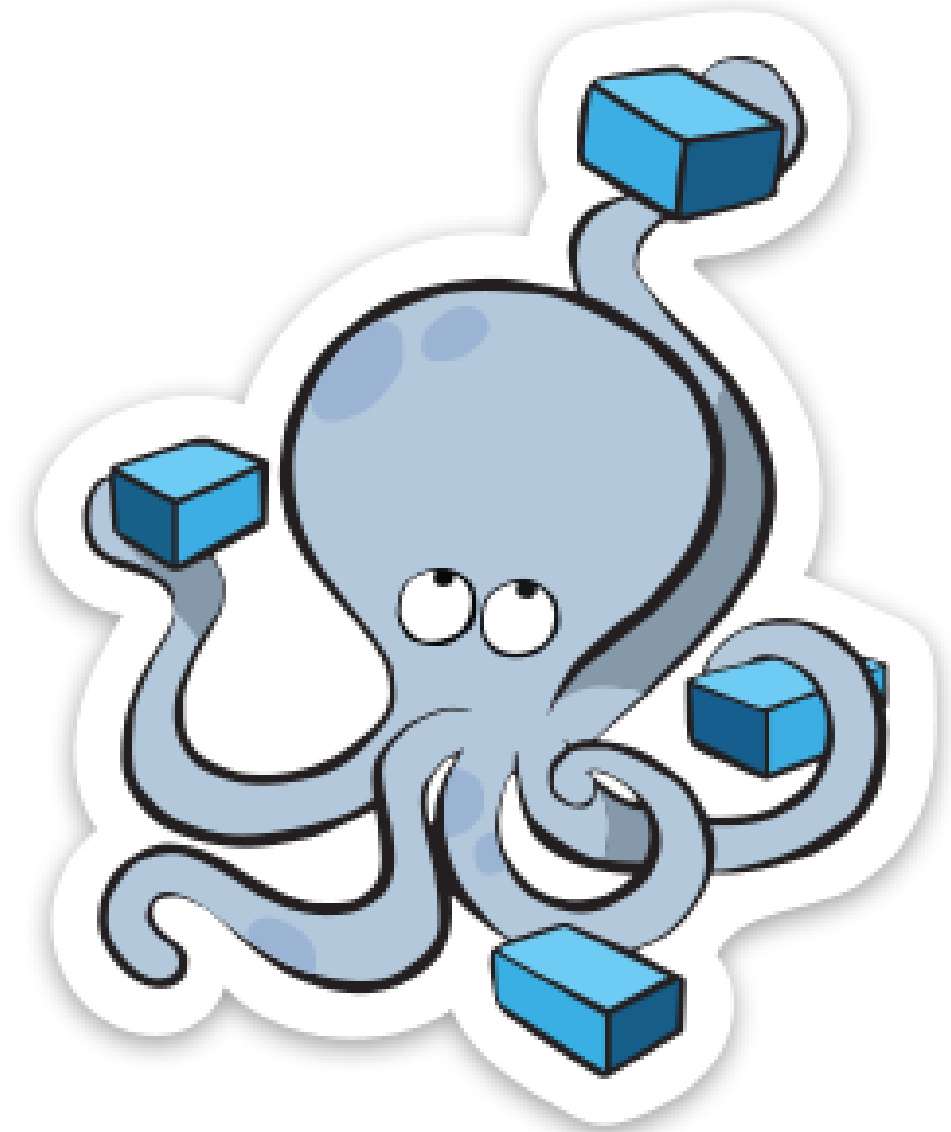
- runs and commoditizes Linux containers
- allows building and sharing images
- runs as daemon and has CLI
- functionality is exposed via REST API
- moves towards standardization  
(runC, containerd)



# Docker Compose

- manages a collection of containers
- fast, isolated development environments using Docker
- define environment via YAML file
- quick and easy to start

```
docker-compose up -d
```

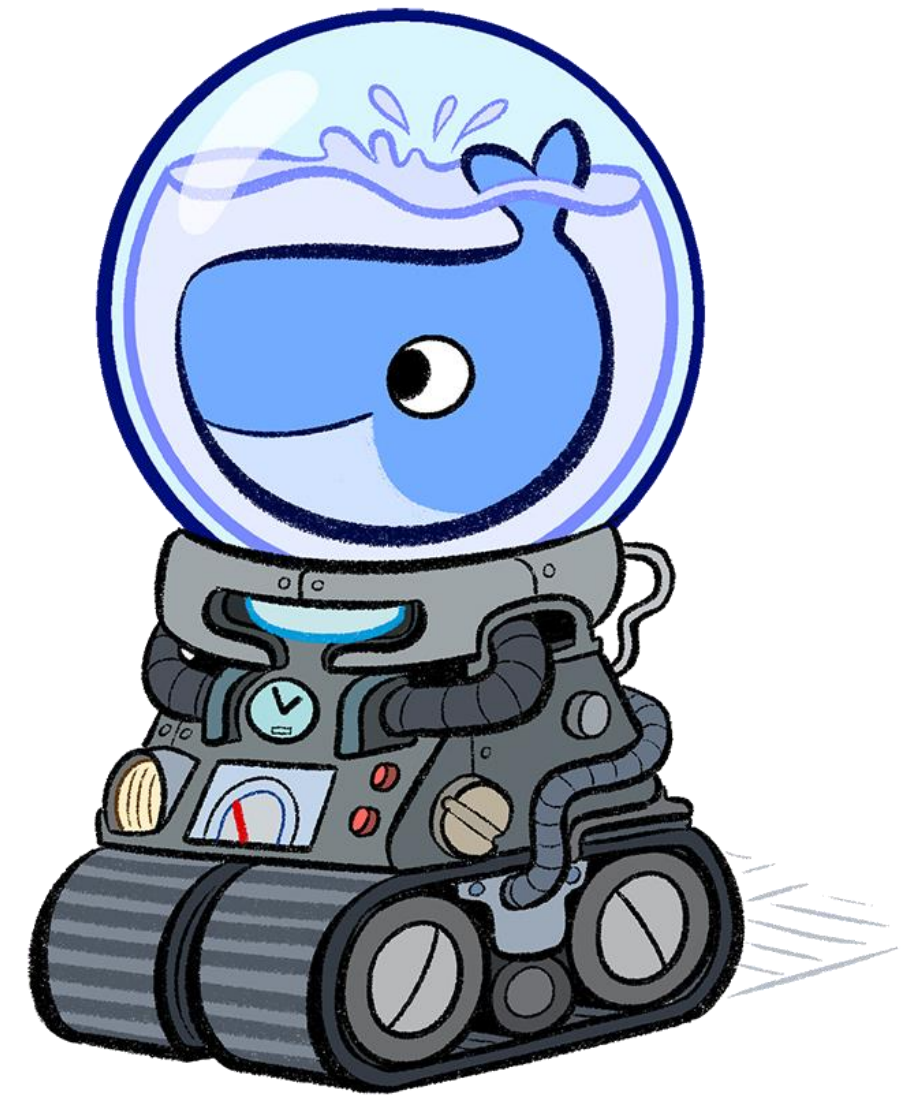




# Docker Machine

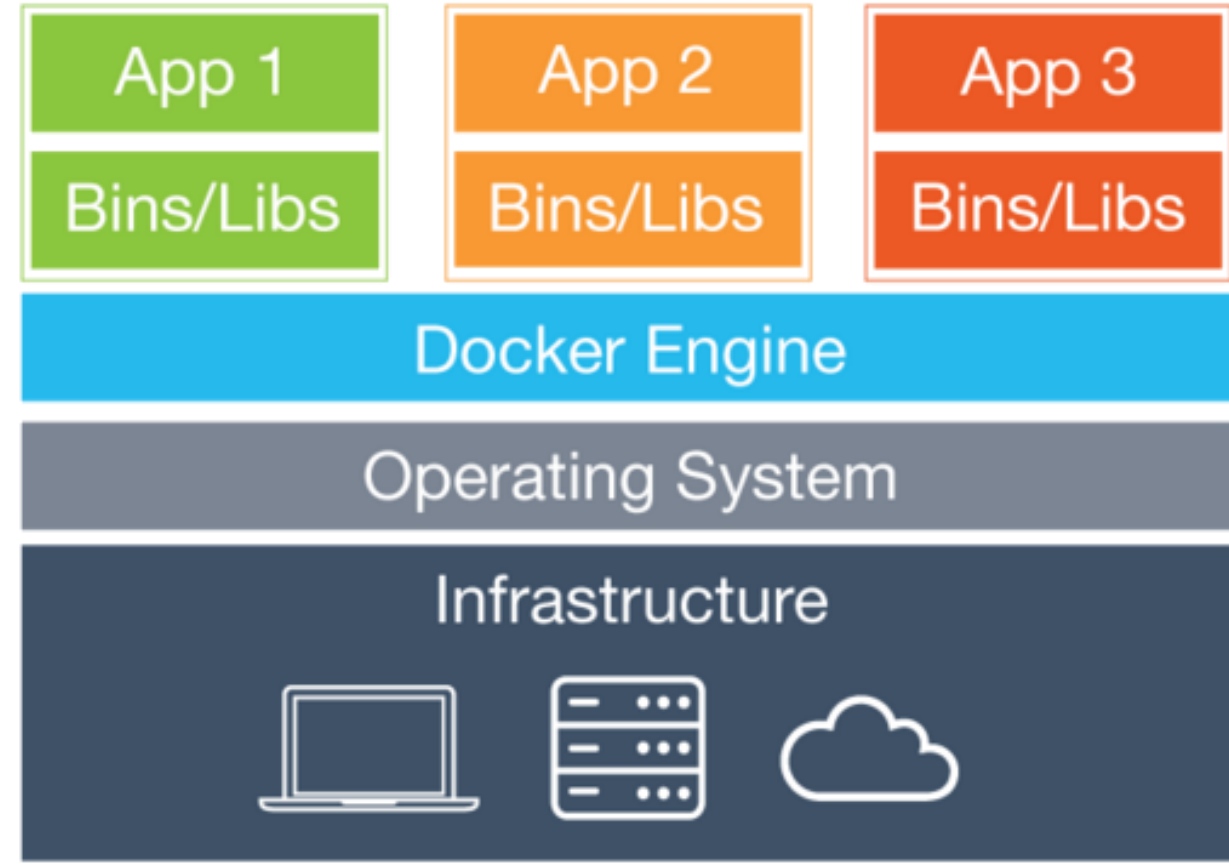
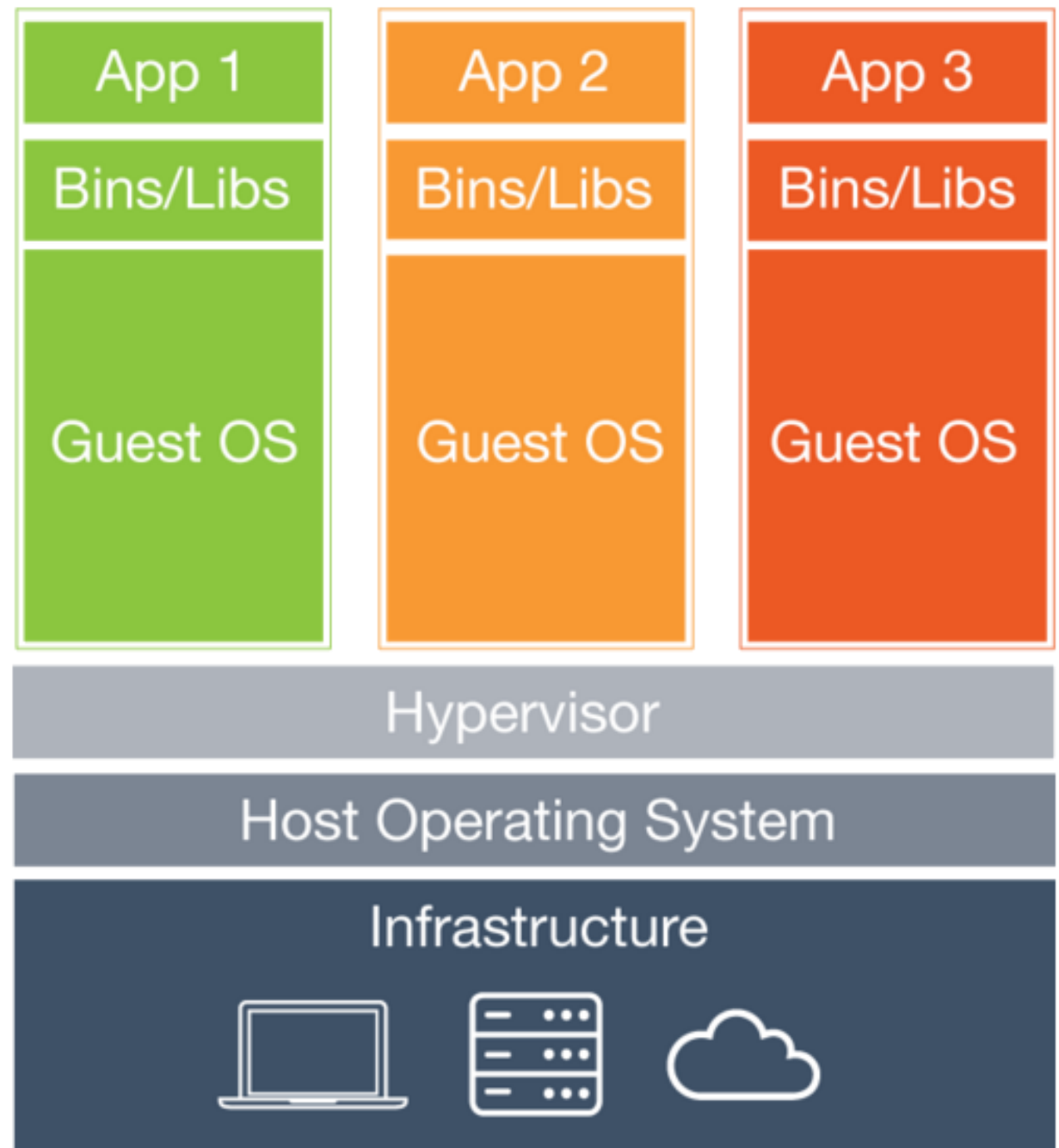
- allows creating Docker hosts on local computer or in cloud providers
- automatically creates host, installs Docker and configures the client
- offers commands to start, stop, restart and inspect a host

```
docker-machine create
--driver virtualbox default
```





# Virtual machines vs. Containers





# Comparison with VMs



VAGRANT



<https://github.com/petyodimitrov/ci-setup.git>

<https://github.com/petyodimitrov/vagrant-setup.git>

<https://github.com/petyodimitrov/vm-container-comparison.git>

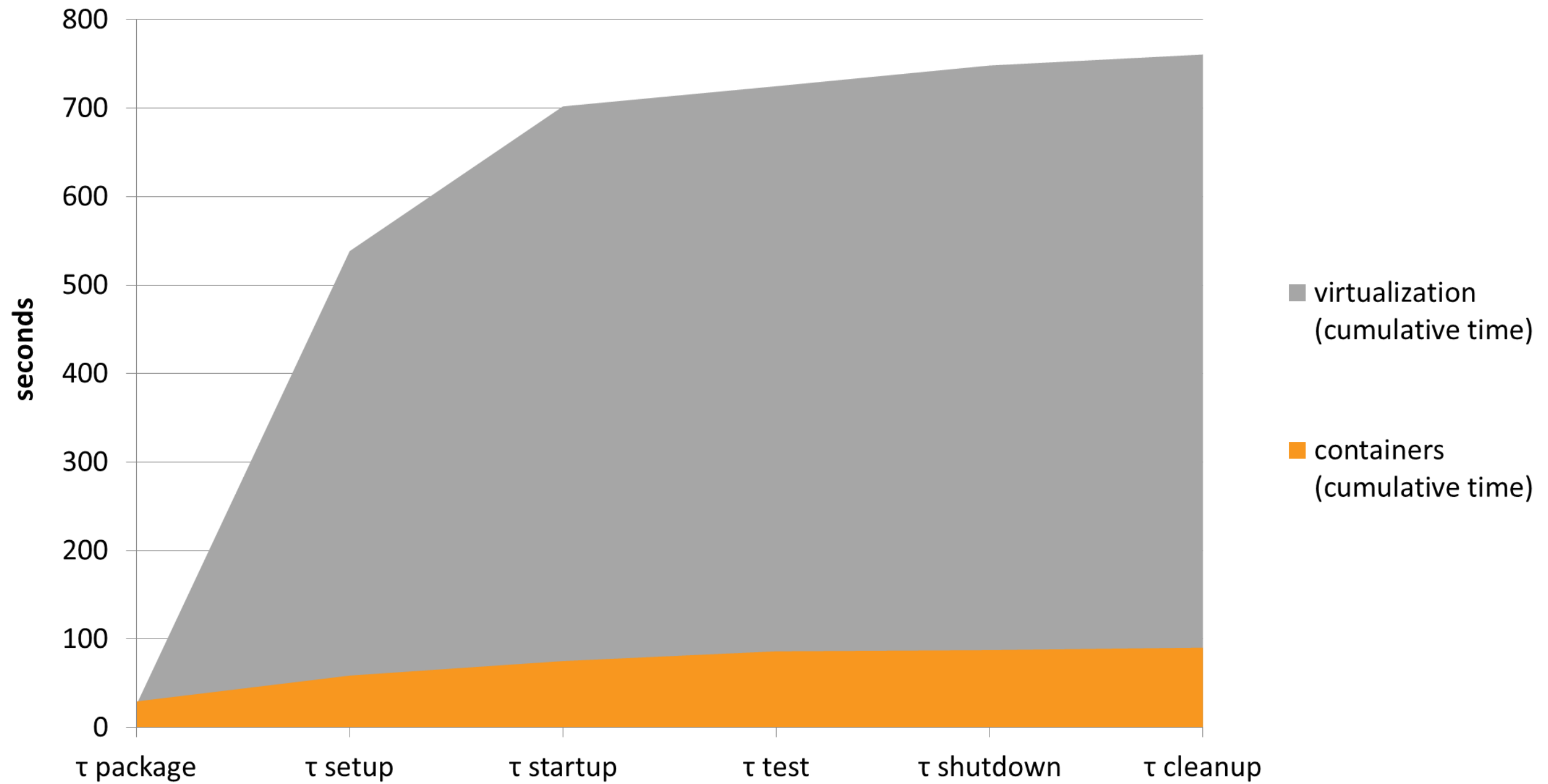
# Application delivery time

$\tau_{\text{delivery}}$ (сек.)	virtualization	containers
$\tau_{\text{package}}$	25,6	29,6
$\tau_{\text{setup}}$	<b>512,8</b>	<b>29,2</b>
$\tau_{\text{startup}}$	<b>163,6</b>	<b>16,4</b>
$\tau_{\text{test}}$	22,8	11
$\tau_{\text{shutdown}}$	<b>23,4</b>	<b>1,4</b>
$\tau_{\text{cleanup}}$	12,4	2,8
total	760,6	90,4





# Application delivery time (2)



# Performance benchmarks (cpu, memory, net)

- CPU performance [MFLOPS]: 2%
- Compression speed [MB/s]: 11%
- Memory bandwidth [MB/s]: 3%
- Serial/Parallel memory access speed [MB/s]: 19%
- Network efficiency [Mbps] & latency [ $\mu$ s]: ?
- Privisioning speed [s]: 60 times

average  
improvement  
with containers



# Lessons learnt (1)

- Decide on a strategy for introducing Docker in the project



- Keep an eye for improvements & changes (e.g. networks)
- Keep containers simple (i.e. only one/few processes)
- Automate, automate, automate ... (via docker or other CM tools)
- Use initialization & validation scripts (e.g. init, supervisord)



# Lessons learnt (2)

- Review Dockerfile-s as part of development process (e.g. peer reviews)

```
FROM alpine-java 14.04.3
```

```
ENV REPO=10.0.1.1:1236
```

```
RUN apt-get update -y && \  
apt-get install curl
```

```
# unnecessary
```

```
RUN curl -L https://${REPO}/something.jar
```

```
USER nobody
```

```
CMD java -jar something.jar
```

verify  
checksum

add  
volume



# Lessons learnt (3)

- Use internal Docker registry for image distribution (unless OSS)
- Invest time to setup container monitoring (e.g. logspout + ELK)
- Consider using JARs instead of other \*ARs
- Shared Docker pattern overall works better than DinD
- to be continued...



# THANK YOU :)

You can find me at:  
[petyo.dimitrov@gmail.com](mailto:petyo.dimitrov@gmail.com)

Petyo Dimitrov  
May 26-27 '16, Sofia

