

# Achieving Continuous Delivery with *Docker*

for Java Enterprise applications

# Agenda

- A typical Enterprise Java landscape
- Continuous Delivery pipeline
- Identified problems
- What are Docker containers?
- Docker "ecosystem"
- Demo
- Tips & Tricks

# Dev landscape for Java EE applications



**maven**



# Continuous Delivery pipeline

## CONTINUOUS DELIVERY



## CONTINUOUS DEPLOYMENT





# potential issues

A black and white photograph showing a worker on a truck bed. The truck is heavily loaded with large stacks of cardboard boxes. Some boxes are falling off the truck, and others are scattered on the ground. The scene is set in an urban environment with buildings in the background. The text 'inadequate packaging of application artifacts' is overlaid on the image in a large, white, sans-serif font.

# inadequate packaging

of application artifacts



inconsistencies across *environments*

A close-up photograph of a person in a dark blue suit and white shirt, holding a thick stack of US dollar bills. The bills are fanned out, showing the front side with the number '7' and the Treasury seal. The person's hands are visible, and a watch is seen on the left wrist. The background is a solid dark blue.

# high cost

supporting multiple static environments





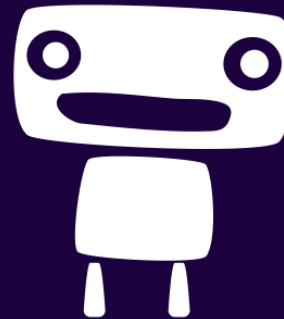
# lack of freedom

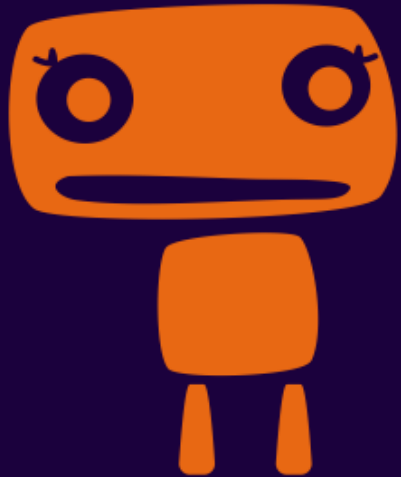
experimenting with new  
languages, technologies  
and frameworks

how can docker  
containers  
help?



wait...what...  
**containers?**







# sea-free analogy

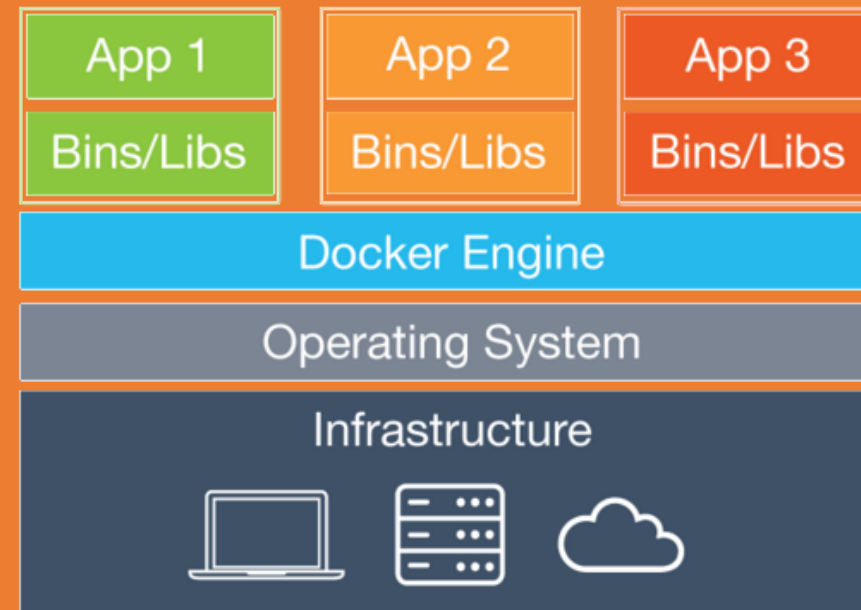
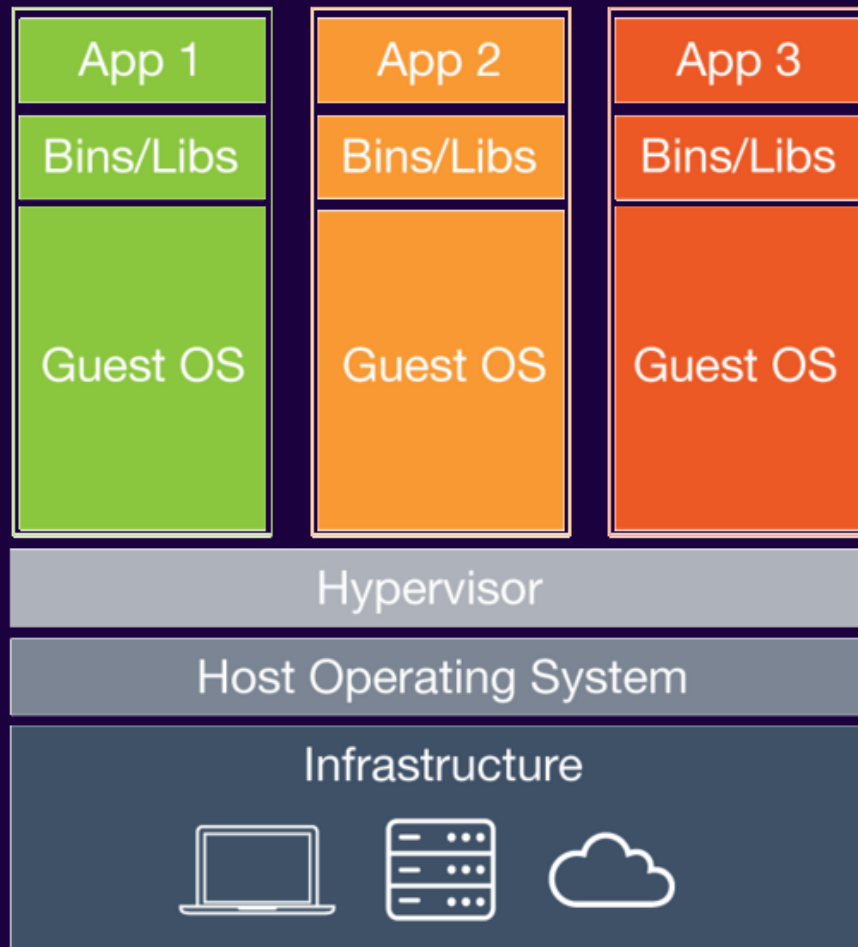
# High level view of a container

- **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

# Virtual machines vs Containers



# How does it work? – Isolation

- **isolation via namespaces**
  - pid, mnt, net, uts, user, ipc
- **isolation via control groups**
  - memory, cpu, blkio, devices





# How does it work? – Storage

- union file system (aufs, overlayfs)
- allows reusing common layers
- reduces traffic and storage
- allows tracking changes
- copy-on-write pattern used for speed

# Developers

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



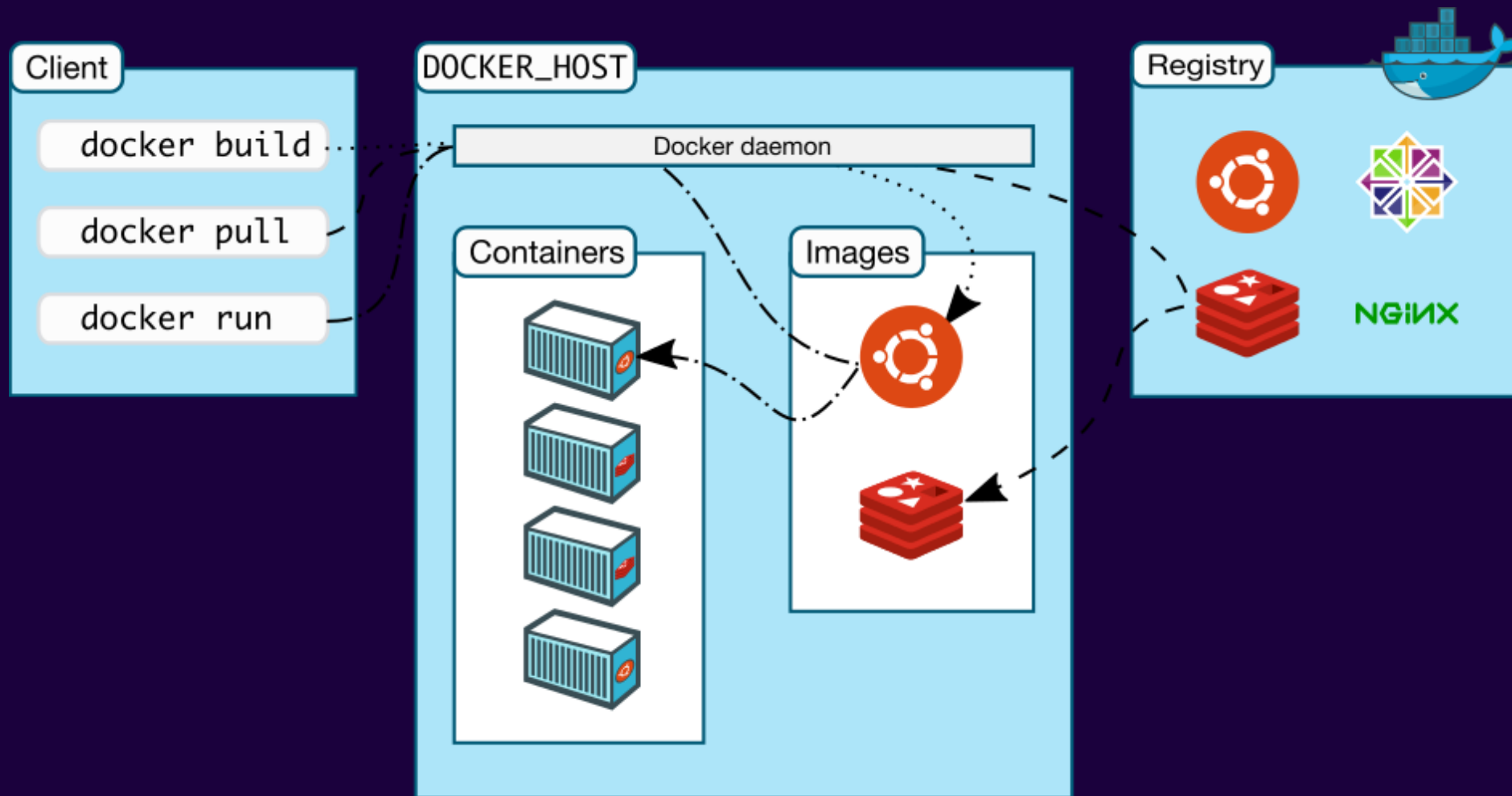
# Operations

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



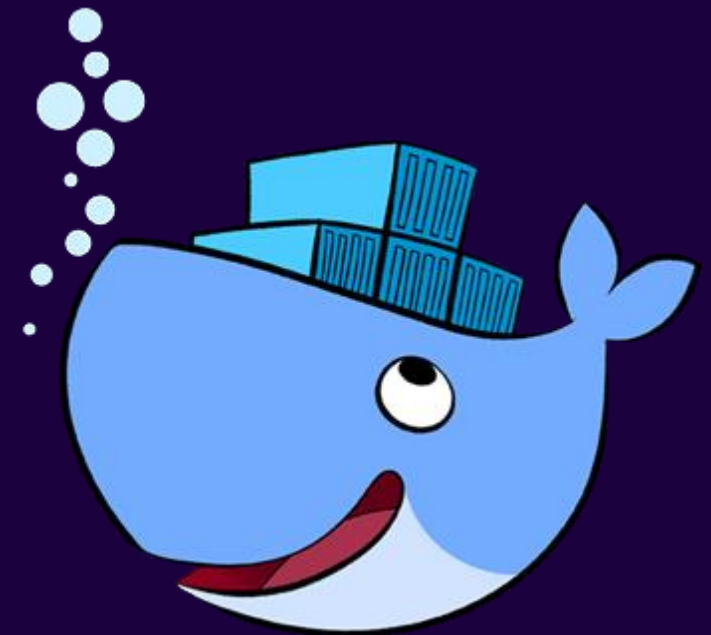
# docker "ecosystem"

# Client-Server model



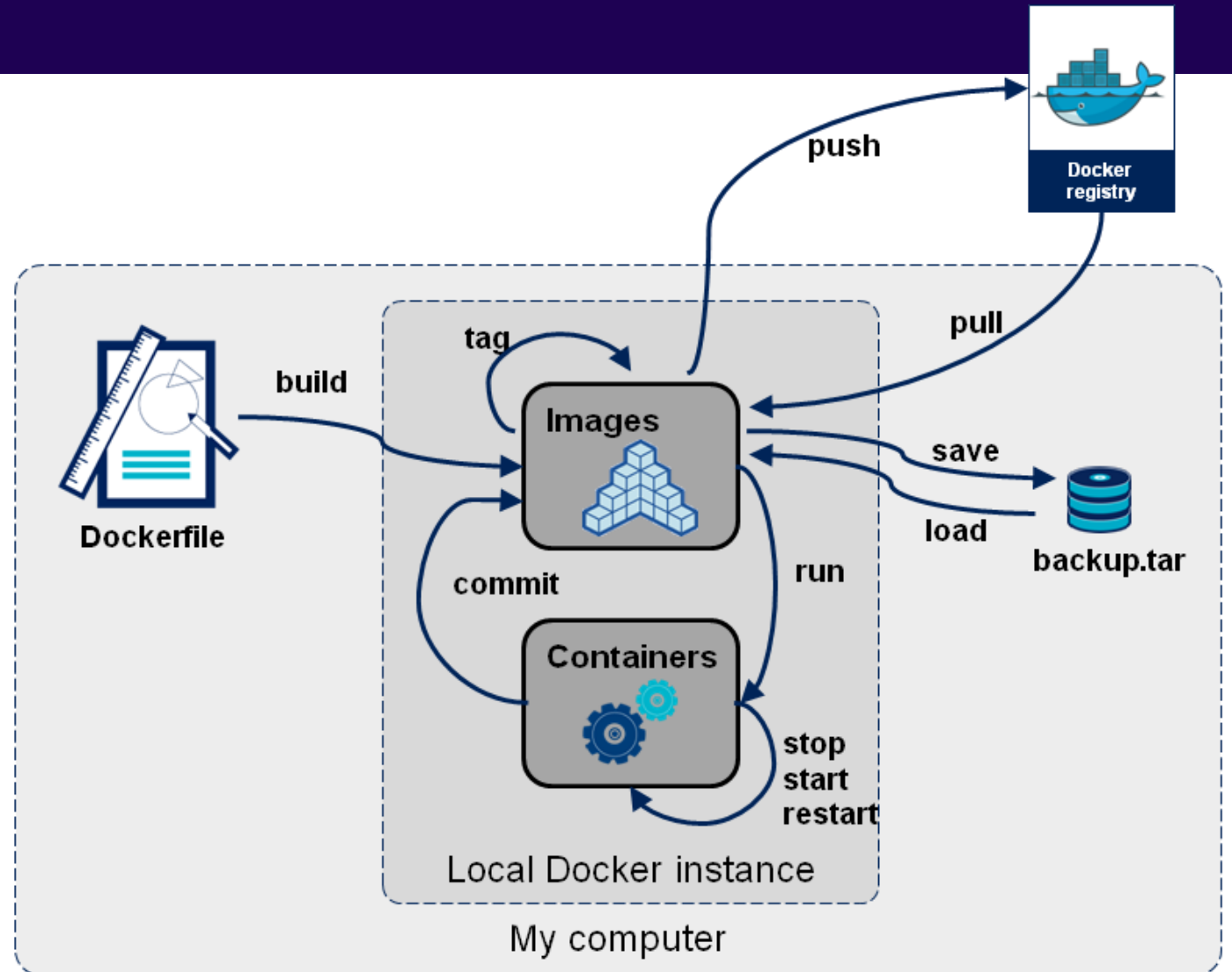
# Docker engine

- runs and **commoditizes** Linux containers
- uses copy-on-write for quick provisioning
- runs as daemon and has CLI
- allows building and sharing images
- functionality is exposed via REST API
- defines standard format for containers



# Docker commands

- ~40 commands
- for working with:
  - images
  - containers
  - registry



# Dockerfile-s

```
FROM jenkins:1.625.1
MAINTAINER Petyo Dimitrov
ENV REFRESHED_AT 2015-10-24

RUN curl -L https://github.com/... > /usr/local/bin/docker-compose
    && chmod +x /usr/local/bin/docker-compose
RUN mkdir /var/cache/jenkins
RUN docker build -t myjenkins .

# Set Defaults
ENV JAVA_OPTS="-Xmx6144m"
ENV JENKINS_OPTS="--handlerCountMax=300 --webroot=/var/cache/jenkins/war"

COPY plugins.txt /plugins.txt
RUN /usr/local/bin/plugins.sh /plugins.txt
```

# Docker Hub/Registry



- collection of ready-made images
- options:
  - public/private registry
  - local registry
- REST API for access



”

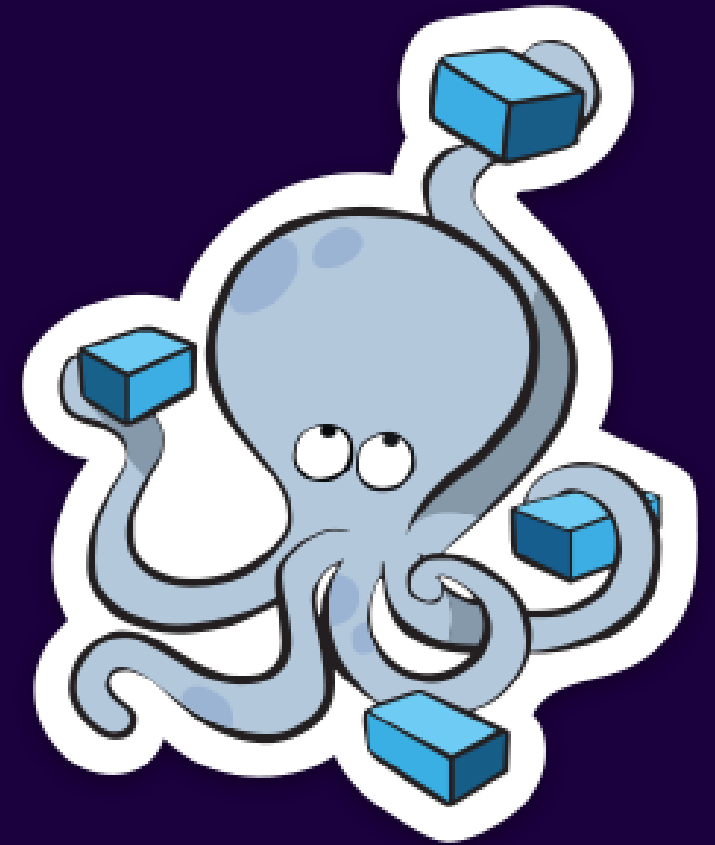
But how to build  
a stack of  
containers?

“

# 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
```



# YAML example

```
proxy:
  build: nginx/
  ports:
    - "80:80"
  links:
    - appinstance
  hostname: "proxy"

nosqlldb:
  build: mongo/
  hostname: "nosqlldb"
  volumes:
    - "/opt/mongodb:/data/db"

appinstance:
  build: tomcat/
  expose:
    - "8080"
  ports:
    - "8180:8080"
  links:
    - nosqlldb
  hostname: "appinstance"
```

”

How can I quickly  
provision a  
Docker host?

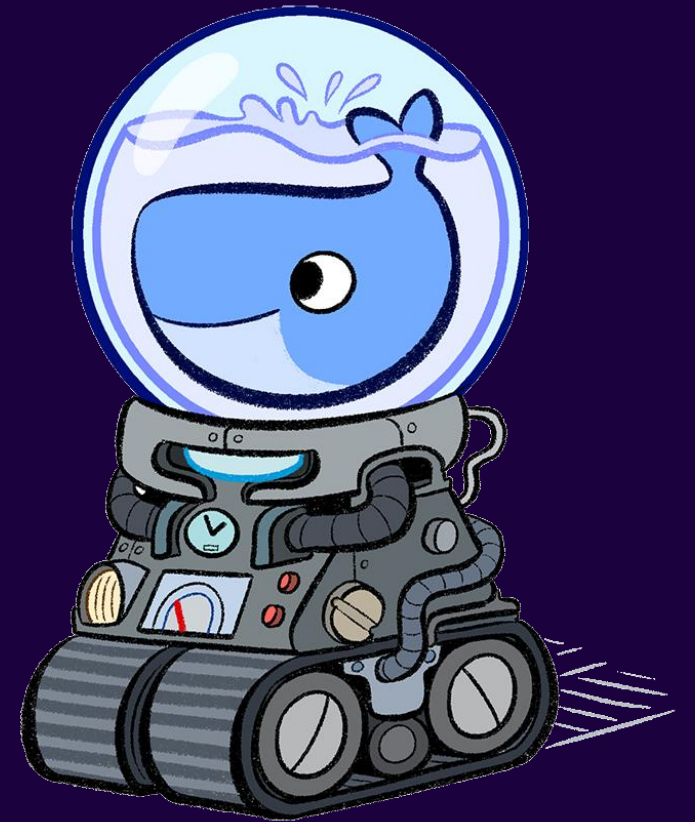
“

# 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

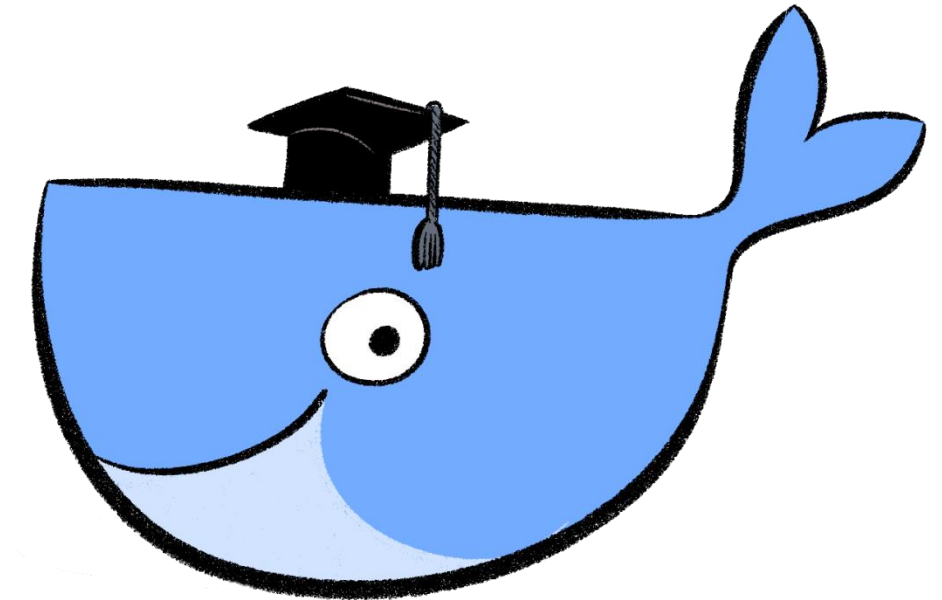
```
docker-machine create
```

```
--driver virtualbox default
```



# Docker Summary

- put my software in containers
- run those containers anywhere
- write recipes to automatically build containers



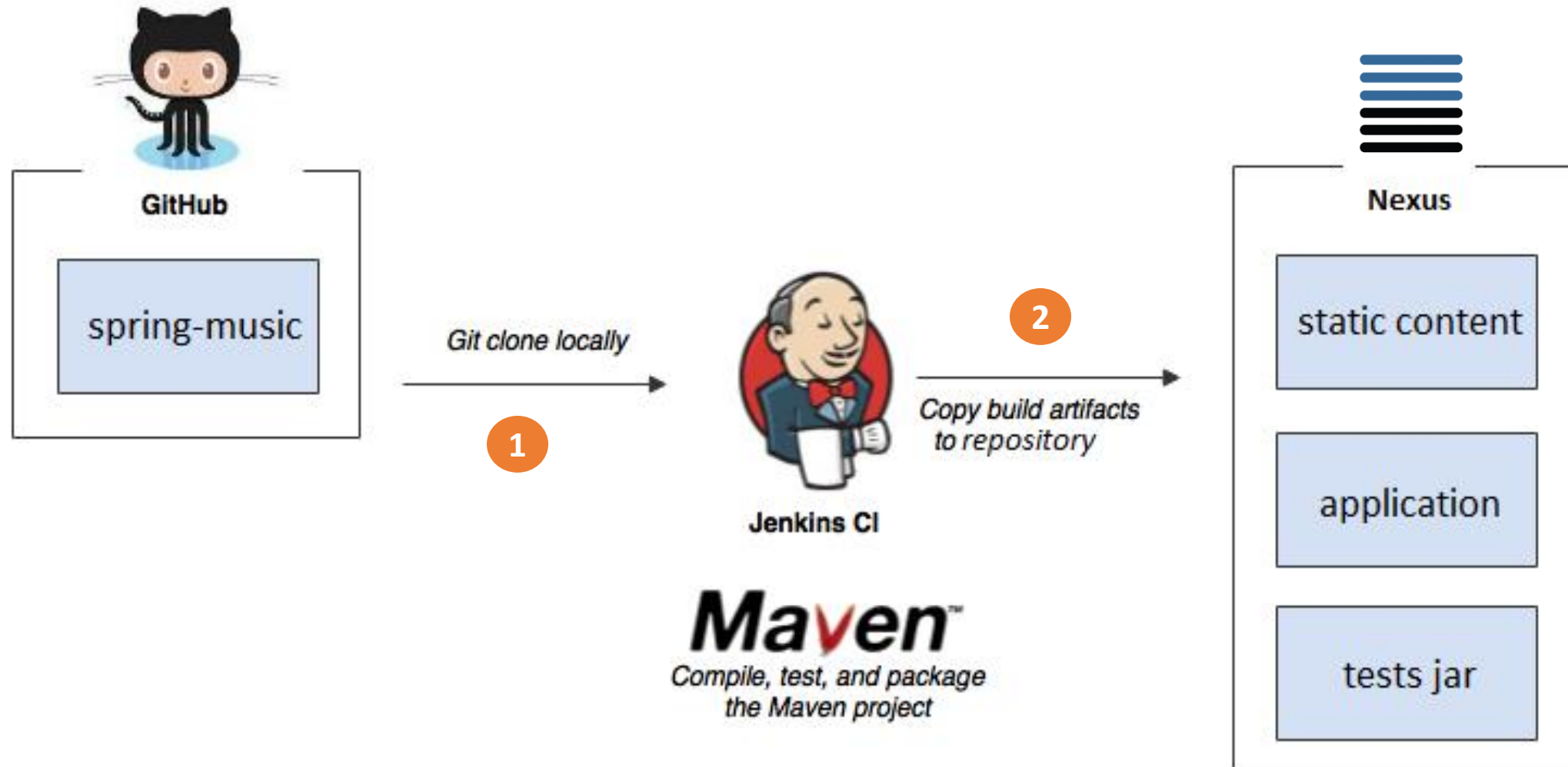
# Demonstration

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

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

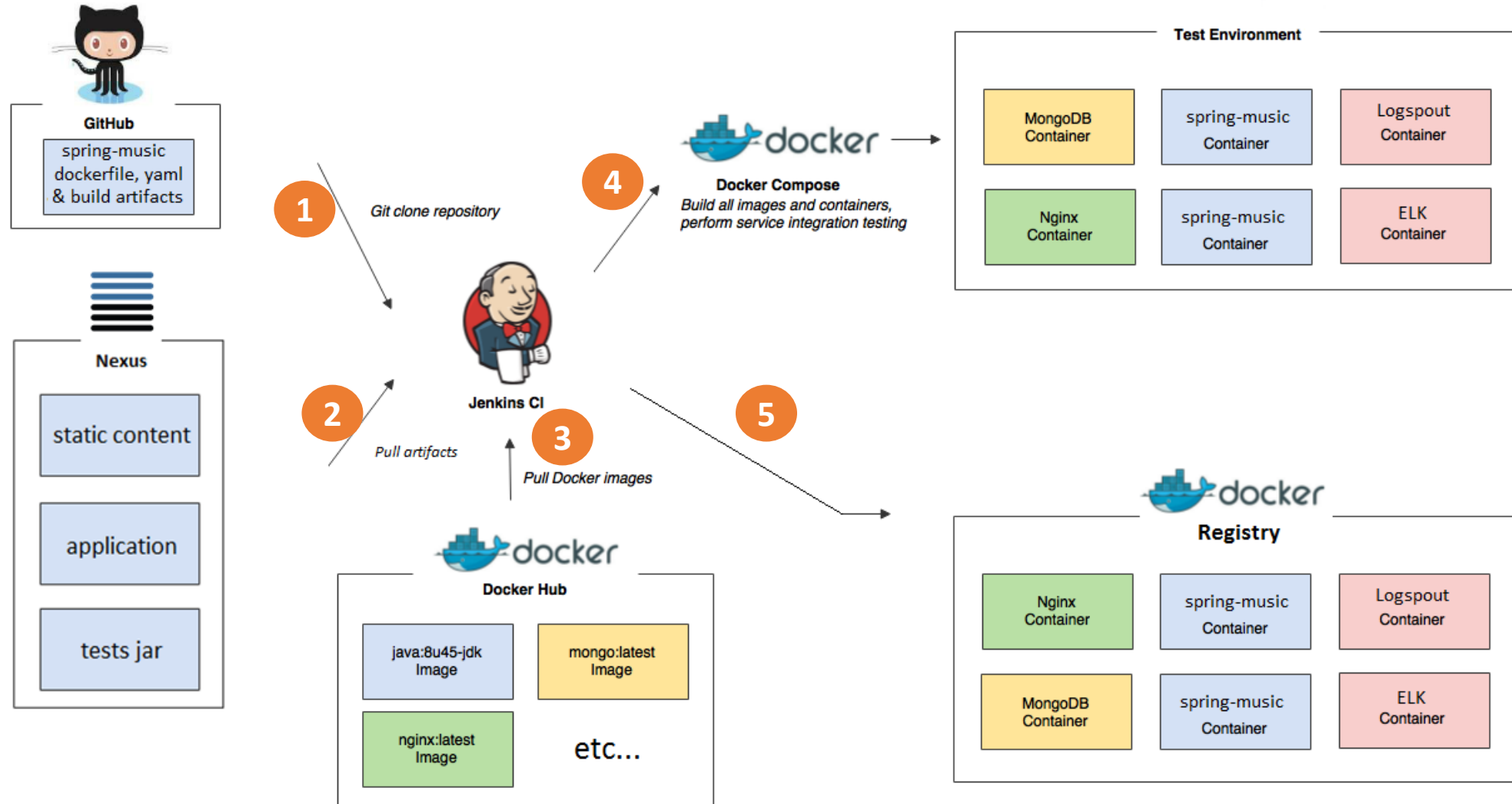
<https://github.com/petyodimitrov/ci-setup.git>

# Standard Java application build

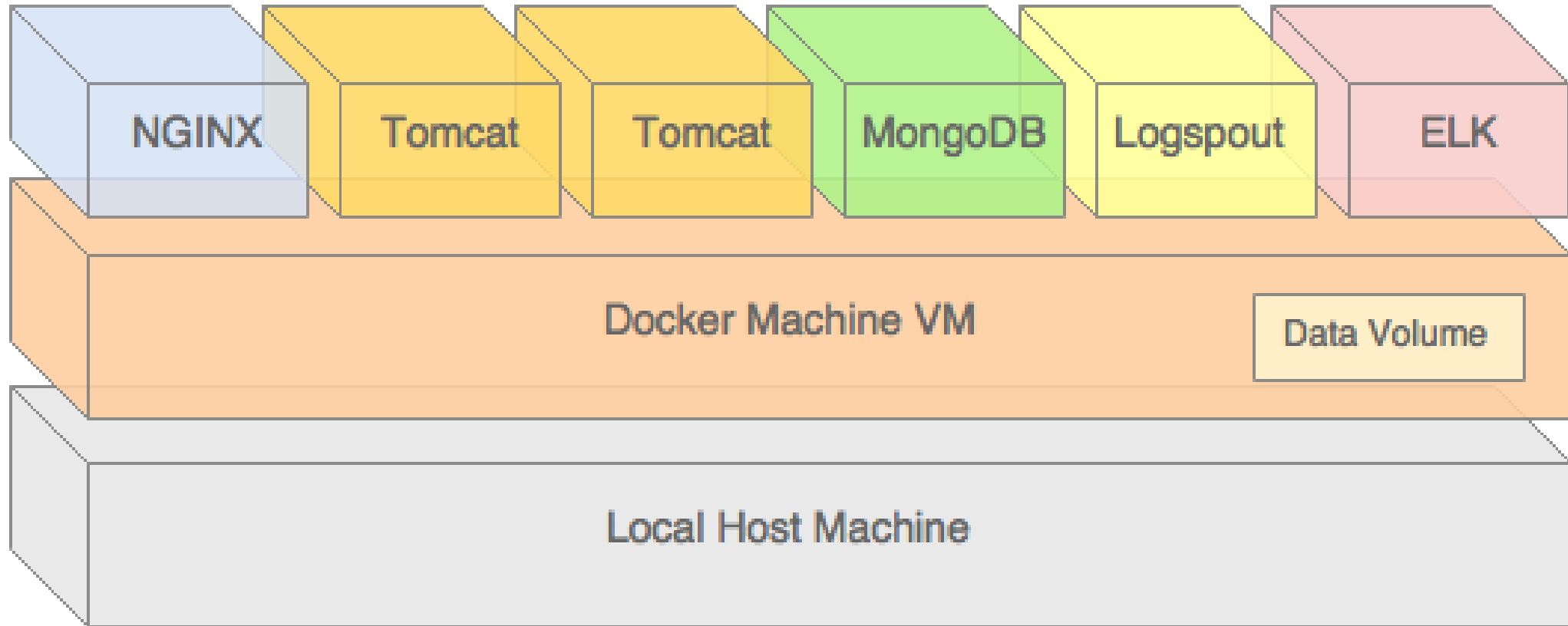






# Adding docker images build




# Runtime view of containers




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

# Demonstration



# Tips & Tricks

- single process per container
- security considerations
- use data containers for portability
- consider reducing image sizes
- etc...

# Thank you!



[petyo.dimitrov@gmail.com](mailto:petyo.dimitrov@gmail.com)