DOGR = DevOps-Gathering Ruhr

K8s = Kubernetes (K, 8 Buchstaben, s)

# Software-Varianten

## Orchestration

* Docker Swarm
* Kubernetes
* Apache Mesos

## Runtime Environments

* Java EE
* Spring Boot
* Wildfly Swarm
  + Ähnlich wie Spring Boot; noch nicht so ausgereift
  + Für Wechsler von Java EE geeignet
  + Erzeugt war als Runtime

## Container-Technologien

* docker
* Rocket
  + Hat keine Client-Server-Architektur (war größter Kritikpunkt von CoreOs)
* Google Container

## Reverse Proxy

* nginx
* HAProxy: <http://www.haproxy.org/>
* Traefik: <https://traefik.io/>

## Data visualization

* Kibana
* Grafana: <https://grafana.com/>

## Monitoring

* Nagios
* Prometheus: <https://github.com/prometheus>

## Deployment Management

* Puppet
* Chef
* Ansible
* SaltStack
* Fabric

## CoreOS

Produkte: CoreOS, quay.io, Rocket, Etcd

# Docker and Kubernetes for Java Developers

Workshop mit Roland Huss

## docker-maven-plugin

* 8-16 verschiedene Plugins verfügbar (gewesen)
* Spotify: seit 9 Monaten kein Fortschritt bei GitHub mehr; offene PRs werden nicht abgearbeitet; kann nur bauen
* Fabric8io: von Roland Huss; kann bauen + starten, docker-compose auswerten; hat wait conditions für hochfahren
* Doku: <dmp.fabric8.io>

## Java im Container

* Java ist nicht dafür vorbereitet, im Container zu laufen
* Java sieht den verfügbaren Gesamtspeicher, selbst wenn der Speicher für den Container reduziert wurde (mit docker run -m 25m)
* Gleiches gilt für Anzahl der Cores
* Man muss dann zusätzlich mit -Xmx die JVM beschränken 🡪 es müssen 2 Params gepflegt werden
* In Java 9 besser gelöst: -XX:+UnlockExperimentalVMOptions -XX:+UseCGroupMemoryLimitForHeap
* Keine Kopplung mehr für Memory (allerdings noch nicht für Cores möglich)
* Java-Docker-Images von fabric8io lösen dieses Problem:
  + <https://github.com/fabric8io-images/>
  + <https://hub.docker.com/r/fabric8/>
* Zur Optimierung kann Java 32 Bit Sinn machen (kleinere Pointer)

## Jolokia

* JMX-http-Bridge als Agend (von Roland Huß)

## Kubernetes

* In Go geschrieben (Golang ist von Google – wie Kubernetes)
* Populärstes Projekt auf GitHub
* Google startet damit 2 Mrd. Container pro Woche (z.B. jeder GMail-Login = 1 Container)
* Über Beschreibungssprache wird Zielzustand definiert
* Client 🡪 Server 🡨🡪 Nodes
  + Client: kubectrl
  + Server: master
  + Nodes: kubelets
* Gute Benutzung, (war) schwierig zu installieren („Software defined networks“)
* Guter Start: minikube (single node)
* Läuft derzeit nicht mit Docker 1.13
* Hat internen DNS-Server
* Kann auto-skalieren (abhängig von Last)

## POD

* Wird über yml deskriptiv beschrieben
* Menge von Docker
* Werden repliziert

## kubectrl-Befehle

* get nodes
* get pods
* create –f spec.yml
* describe pod <name>
* logs <name>
* exec -it <name> bash
* scale
* edit
* delete
* rolling-update

## Replication Controller

* managt PODs
* fährt hoch/runter gemäß Spec

## Service

* stellt feste IPs nach außen zur Verfügung
* ist Proxy für PODs (interner Loadbalancer)
* verbindet verschiedene PODs miteinander
* ermöglich Zero-Downtime-Updates (rolling-update)

## Volumes

* externer Speicher für zustandsbehaftete Dienste

## Ingress

* Sammlung von Regeln zum Routing 🡪 Ingress-Controller

## Kubernetes vs. Swarm

* nähern sich beide einander an
* Features etwas unterschiedlich
* Swarm: (noch) einfacher zu handhaben
* Kubernetes: (noch) mehr Features

## OpenShift

* Projekt von RedHat
* Seit V3: für Kubernetes; open source
* UI für Kubernetes
* Im Gegensatz zu Kubernetes nicht nur betreiben, sondern auch bauen
* Bringt Router, Service Registry und Security (OAuth2) mit
* Betriebsmodi: On Premise, Cloud, Dedicated, …

## MiniShift

* Single node OpenShift cluster
* Analog zu Minikube

## Fabric8

* Offene Toolsammlung von RedHat
* fabric8-maven-plugin: bringt Spring Boot nach Kubernetes
  + nutzt docker plugin intern, um images zu bauen
  + Doku: <https://maven.fabric8.io>
  + Löst Problem der Ressource-Deskriptoren von Kubernetes (lang, schwer lesbar) durch direkte Erzeugung

## Sonstiges

* helm: Package Manager für Kubernetes
  + „helm install gitlab“
  + „helm install mariadb“
* kubeadm
* <https://github.com/coreos/flannel>

## Ressourcen

* <https://ro14nd.de/kubernetes-on-raspberry-pi3>
* <https://leanpub.com/k8spatterns> (3€)
* <https://github.com/kubernetes/minikube/>
* <https://kubernetes.io/docs>

### Präsentation

* <https://hub.docker.com/r/rhuss/kubernetes-for-java-developers/>
* Sollte auf Linux laufen, damit die Demos genutzt werden können
* docker pull rhuss/kubernetes-for-java-developers
* docker run -d \

-v /var/run/docker.sock:/var/run/docker.sock \

-p 9000:9000 -p 57575:57575 -p 35729:35729 -p 8001:8001 \

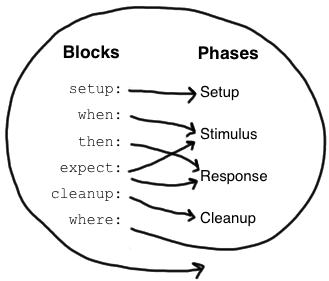
rhuss/kubernetes-for-java-developers:latest 1>&2

* Anschließend <http://localhost:9000> aufrufen

# "Bootiful" Containers with Spring-Boot and Docker

Workshop mit Kevin Wittek und Matthias Rottländer

* Spring Boot Definition: “**Opinionated view on Spring using sensible defaults**”
* Gradle: Java Build-Management auf Basis einer Groovy-DSL
* Spock: Test-Framework auf Basis einer Groovy-DSL
  + Java und Groovy-Code lassen sich mischen
  + Methodenname kann beliebig sein: def "lists all existing books"() {}
  + Block given: 🡪 Setup



## Ressourcen

<http://spockframework.org/spock/docs/1.1-rc-3/index.html>

<http://slides.com/kiview/deck-8>

<https://12factor.net/>

<https://github.com/devops-gathering>

### bootiful-hello-world

1. Install Gradle on Windows (using provided script in project <https://github.com/devops-gathering/bootiful-hello-world>)
2. Install buildship from eclipse marketplace
3. git clone https://github.com/devops-gathering/bootiful-hello-world
4. Turn on gradle nature (Project > right click > Configure > Gradle Nature)
5. Run project: ./gradlew bootrun
6. Run project with env vars: GREETER\_NAME=my-name ./gradlew bootrun
7. Build image: ./gradlew buildImage
8. Run docker-compose up

### bootiful-book-store

Execute Tests: ./gradlew check

# How we run Kubernetes in Kubernetes, aka Kubeception

Talk von Timo Derstappen

* Zertifikate werden alle 3 Tage automatisiert erneuert (Nutzung von HashiCorp Vault: Password rotation)
* Nutzen Software defined networks

# Docker Container Loading

Talk von Roland Huß

<https://hub.docker.com/r/rhuss/talk-docker-container-loading/> (Anleitung siehe Workshop von Roland Huß)

## Problems

* No composition (“includes”)
* Influence on layers limited (“&&”)
* Too simple parametrization
* …

## Solutions

* Templates
* fish-pepper (by Roland Huß, based on node.js)
* Simple
  + Create base container
  + Modify with ansible
  + docker commit (otherwise dirty!)
* HashiCorp Packer
* Rocker (Erweiterung zu Dockerfile)
  + multiple FROM
  + supports templates

# Docker Orchestration and Swarming

Talk von Peter Rossbach

* docker swarm: ca. 9 Monate alt
* manager + worker sind flexibel (manager kann worker werden und umgekehrt)
* Setup ist leichter als Kubernetes
* Stateful with mounting: noch ein Problem
* Organisations-Struktur muss dafür ausgelegt sein
* „Add value = Build ship, run“: nur, was beim Kunden ankommt, bietet Mehrwert
* Swarm macht mehr Sinn in kleineren Setups (~4 Systeme), nicht bei sehr großen Anwendungen
* <https://github.com/ManoMarks/docker-swarm-visualizer>

## Befehle

* docker swarm init
* docker service create
* docker node

## Infrakit

* create & manage declarative, self-healing infrastructure

## Golden Rules für Container & Images

1. One Purpose: 1 Image = 1 Purpose
2. Labeling: Add as much metadata as possible
3. Working: Must be working all the time
4. Fix deps: Dependencies must be fixed
5. Minimal: Add nothing more than necessary
6. Whitebox: Source must be available
7. Secure: Update regularly
8. Limited: Make sure to only consume minimal resources
9. Trust

# One year monitoring a small startup with Prometheus: lessons learned

Talk von Maykel Moya, Kuba

* Guter Vortrag, wenn wir Monitoring lernen wollen
* Many best practices
* PromQL: Prometheus Query Language

# Build and deploy multiarch Linux and Windows Container images

Talk von Stefan Scherer

* Windows Based Images
  + NanoServer (~ Alpine)
  + WindowsServerCore (~ Debian)
* Windows containers läuft auf Win 10 Pro (braucht Hyper-V) sowie auf Windows Server 2016
* Stefan nutzt
  + Travis CI: kann Matrix für Multiarch- / MultiOS-Builds
  + AppVeyor: Cloud CI für Windows
  + DockerHub: privates Docker Image-Storate

# Intel and ARM, let Kubernetes rule them all!

Talk von Lucas Käldström, 17-jähriger Schüler & Kubernetes-Maintainer

* Cluster aus 3 Raspberry Pi, 2 ARM, 2 AMD
* Auf diesem laufen per Kubernetes verteilte Container

# Create the Dashboard Deployment and Service

kubectl apply -f demos/dashboard/dashboard.yaml

# Create the Heapster Deployment and Service

kubectl apply -f demos/monitoring/heapster.yaml

# Deploy Traefik as the Ingress Controller and use Ngrok to

# expose the Traefik Service to the Internet

kubectl apply -f demos/loadbalancing/traefik-common.yaml

kubectl apply -f demos/loadbalancing/traefik-ngrok.yaml

# Expose the Dashboard to the world

kubectl apply -f demos/dashboard/ingress.yaml

# Get the public ngrok URL

curl -sSL $(kubectl -n kube-system get svc ngrok -o template --template \

"{{.spec.clusterIP}}")/api/tunnels | jq ".tunnels[].public\_url" | sed 's/"//g;/http:/d'

# Create InfluxDB and Grafana for the saving the Heapster data

kubectl apply -f demos/monitoring/influx-grafana.yaml

# Create the Prometheus Operator, a Prometheus instance and a sample metrics app

kubectl apply -f demos/monitoring/prometheus-operator.yaml

kubectl apply -f demos/monitoring/sample-prometheus-instance.yaml

# Create a Custom Metrics API server

kubectl apply -f demos/monitoring/custom-metrics.yaml

## Resources

* <https://luxaslabs.com/>
* <https://github.com/luxas>
* <https://slides.com/lucask/devops-gathering/>

# Make load-balancing great again with Traefik!

Talk von Emile Vauge (Frankreich, Lead Developer Traefik) und Manuel Laufenberg

* Another reverse proxy (vgl. nginx)
* In Go geschrieben
* Backends: Docker, Swarm, Kubernetes, Mesos, Consul, Etcd, Zookeeper, Rancher, …

# Getting the most out of your containerized database

Talk von Claus Matzinger, crate.io

* Machine generated data: 40% of all data (IoT (sensors, …), wearables, logs, vehicles, …) 🡪 main use case
* CrateDb: OpenSource distributed SQL Db
* Low Config, Partitioning, Sharding, Replication
* Not ACID, no Transactions
* Not working on multi nodes (“split brain scenario”)

# DevOps is not enough - towards a DevOps ecosystem

Keynote von Uwe Friedrichsen, CTO von codecentric

* The Phoenix project: „DevOps bible“
* DevOps – Definition

1. Systems thinking – how fast are cycles
2. Feedback loops
3. Continuous experimentation & learning

* IT-Role
  + (Business): Nervensystem (BSS)
  + (Business): Differentiator (Disruptive Tech): IT verändert Einkaufen, …
  + (Market): Digitalisierung (Produkte)
  + (Market): Continuous Conversation (Medium)
* Rethinking IT – a suggestions
  + IT needs to be fast (hours instead of months)
  + KPIs: cycle time, availability in production 🡪 more important than test coverage, standups, …
  + Effectiveness: outcome (e.g. customer buys), not output (e.g. lines of code)
  + …
* Building Blocks
  + Process
  + Organization
  + Technologies
  + People (Empathy first!)
  + Governance
  + Adaptation
* DevOps IT Organization
  + Many cross-functional product teams for each business capabilities
* Each team has all skills (from Business Analyst to Operations)
* Autonomy through Microservices
  + Reason for Microservices is not only tech, but also effective teams (“cross-functional product teams”)
* “Lean EAM” (Enterprise architect management)

Nur Grenzen klären zwischen MicroServices, z.B.

* + Message Bus
  + http/REST/JSON
  + Encryption
* Kanban (Feature flow) rather than Agile/Scrum
* T-Shaped people/skills: collaborate across disciplines, skill depth

# OpenSpace

* Regeln siehe <https://de.neuland.com/methodentools/openspace/>, Poster 1-9 sowie Mails von [info@devops-gathering.io](mailto:info@devops-gathering.io).
* „Butterflies“: stetige Leute, bieten Anlaufstelle
* „Bumblebees“: Wechsler, sind Informationsträger von einer zur nächsten Gruppe

## Scrum bei subito.it / G DATA / Zalando

* Team besteht aus 5-6 Personen
* In jedem Team FE, BE, Middleware, Admin / DevOps vertreten
* 1 Feature pro Sprint (nicht 10…)
* „Team is Devops, not Person is DevOps”
* Scrum of Scrums / Nexus Scrum als Alternative
* Tellerrand: Admin liest Code, Dev versteht Admin
* Support ist teamübergreifend: Umfrage im Unternehmen, wer 24/7 Bereitschaft machen möchte 🡪 Zusatzvertrag (Bereitschaft + Einsatz) 🡪 3-5 Leute rotieren; ggf. Neueinstellung
  + Mgl. Lösung im Fehlerfall: rollback zum letzten Container

## Hacker Room

* 5x 3er Raspberry Pi (bee42)
* 4er Raspberry Pi (Roland Huß)
* Lucas‘ System (s.o.: 3 Raspberry Pi, 2 ARM, 2 AMD)

# Groovy + testcontainers

Lightning Talk von Kevin Wittek

* testcontainers + posgres besser als h2!
* testcontainers + posgres in jedem IT
* G DATA nutzt für ITs tatsächlich Selenium!
* testcontainers + selenium nur nachts

# Eindrücke

## Tag I

* Roland Huß: Lead Developer Jolokia, sehr guter Speaker, großes Know-How
  + Werbung für eigene Software
* com.spotify:docker-maven-plugin seit ~9 Monaten inaktiv, PRs werden nicht bearbeitet
* Java im Container
  + entweder doppeltes Speicher-Management nötig (für docker und für JVM), oder Nutzung von io.fabric8:docker-maven-plugin
  + Anwendungen, die auf kleiner Maschine (wenig RAM) laufen, tun es auf AWS nicht unbedingt (falls Speichermanagement nicht sauber ist)
* Müssen uns Gedanken machen über Speichermanagement
* docker swarm vs. kubernetes: ähnlich; kubernetes: hottest Project on github, noch umfangreicher und etwas komplizierter
* Kevin Wittek: hat testcontainers für „Docker for Windows“-fähig gemacht
* Gradle/Groovy-Unterstützung in eclipse schlecht (oder „nur“ schlecht bei mir konfiguriert?)
  + Keine offizielle Groovy-Unterstützung von Spring mehr, nur über Hack
  + Für Groovy fehlt Syntax-Highlighting, Open Declaration, Quick Fix, Organize Imports, Autovervollständigung
  + Bei Gradle wird Dependency nur nach “Update Project” erkannt; kein Dependency-Tree gefunden
* IntelliJ bei Java-Developer weitaus beliebter
* G DATA nutzen testcontainers
  + Mit PostgresContainer statt mit jdbc:tc:postgresql://; allerdings ist letzterer Ansatz einfacher
  + Netter Ansatz, die Url vom docker Image in Spring bekannt zu geben:

static class Initializer implements ApplicationContextInitializer<ConfigurableApplicationContext> {

@Override

void initialize(ConfigurableApplicationContext configurableApplicationContext) {

EnvironmentTestUtils.addEnvironment("testcontainers", configurableApplicationContext.getEnvironment(),

"spring.datasource.url=" + staticContainerHandle.jdbcUrl,

"spring.datasource.username=" + staticContainerHandle.username,

"spring.datasource.password=" + staticContainerHandle.password,

)

}

}

## Tag 2

* Vortrag von Maykal über Prometheus – sehenswert.
* Vortrag von Lucas und Gespräch mit ihm: beeindruckend. Auch am Folgetag wird sichtbar, wie er eine Gruppe von Menschen „lehrt“. Er diskutiert auf Augenhöhe mit Roland.
* Vortrag von Uwe: gemeinsam reflektieren!
* Gespräche mit Roland, Lucas und Kevin Wittek
* Kevin W.: IntelliJ Community Edition probieren.
  + Lizenz gegen JRebel tauschen?
* Swarm vs. K8s
* Traefik: interessanter Reverse-Proxy
* Warum MultiOS Images nötig, wenn dieselben Images bei uns auf Linux und Windows laufen?

## Tag 3

* Open mindedness
* Knowledgeable people
* Scrum bei subito.it / G DATA / Zalando Dortmund
* Kevin Wittek über testing
* Jenkins Plugin docker-build-step: für uns interessant?

# Microservices mit Spring Boot und Spring Cloud

Eberhard Wolff

## Ressourcen

* <https://leanpub.com/microservices-ueberblick/>
* <https://www.dropbox.com/sh/md05e3tv7f8r0l6/AADp-wQaKRjsH7UZ1fgTm2VYa?dl=0>
* <https://github.com/ewolff/spring-boot-demos>
* <https://github.com/ewolff/microservices-vm>
* <http://lvm-las-roca.herokuapp.com/>: 4 Apps; Spring Boot, Node.js gemischt

### Kontakt

[eberhard.wolff@innoq.com](mailto:eberhard.wolff@innoq.com), +49 151 75004986, <http://twitter.com/ewolff>

<https://www.innoq.com/de/newsletter/>

## Definition „Microservices“ (Wolff)

„Independently deployable modules“

* Änderungen an einem Modul beeinflussen nicht die Installation eines anderen Moduls
* Dafür ist eine Technologie wie z.B. Docker nötig; Java allein ist nicht geeignet.

## Kommunikation zwischen Microservices

* HTML-Links
* Datenreplikation
* REST
* Messaging

## Anforderungen

* Jeder Microservice soll einheitlich betrieben werden können
* Leichtes Aufsetzen von Projekten

## Empfehlungen

*Microservices sollten keine gemeinsame parent.pom haben (widerspricht dem Microservice-Gedanken).*

*Self contained systems: Jeder Microservice sollte sein eigenes FE beinhalten (Widerspruch zu Single-Page-Anwendungen) 🡪 FE Monolith konterkariert Microservices*

## Spring Boot

* Optional: als Linux Service deployen (/etc/init.d)
* spring-boot-starter-remote-shell: ssh -p 2000 user@localhost (PW von Konsole)

## Spring Cloud

* Stream (Kafka, Redis, …)
* Sleuth (verteiltes Tracing: Zipkin, …)
* Cluster
* HashiCorp Consul (Service Discovery)
* Zookeeper (Configuration)
* Connectors
* Cloud for AWS
* Security
* Configuration
* Bus
* Netflix Zuul (Routing)
* Netflix Ribbon (Client side load balancing)
* Netflix Eureka (Service Discovery)
* Netflix Hystrix (Resilience)

## Eureka

* Nur für REST Kommunikation relevant; Messaging ist fundamental anders
* REST based
* Supports replication
* Caches on client
* Resilient (widerstandsfähig)
* Fast
* Non consisten (was Eureka1 weißt, weiß Eureka2 nicht unbedingt)

## Consul

* DNS based 🡪 DNS Caching möglich; Technologie unabhängig (Java, Go, JS, Groovy, …)
* SRV Records
* Redundant
* Can be consistent
* Can cover configuration
* Leichter zu betreiben
* consul-templates generates configuration e.g. for load balancing

## Zuul Proxy

* Eine Url nach außen
* Intern: viele Microservices
* Ähnlich wie ein Reverse Proxy; im Gegensatz dazu aber Filter
* Entspricht einem ApiGateway

## Spring Cloud Config

* Alternative zu Zuul
* Dynamische Neukonfiguration (lt. Wolff schlecht: besser Config Service mit neuer Konfiguration hochfahren und alten herunterfahren)

## Spring Cloud Security

* SSO via OAuth2
* Forward token e.g. via RestTemplate
* Support for Zuul

## Ribbon

* Client side load balancer 🡪 no bottle neck, resilient

## Hystrix

* Resilience
* Do call in other thread pool
* No blocking of request handlers
* Circuit breaker
* Offers fallback method

## Alternative: Kubernetes

* Transparent routing, service discovery per DNS, Load balancing
* Kein Ribbon, Eureka, Consul nötig