# Building a Self-Sufficient System

Viktor Farcic

<http://vfarcic.github.io/devops22/workshop-short.html>

export AWS\_ACCES\_KEY\_ID=

export AWS\_SECRET\_ACCESS\_KEY=

export AWS\_DEFAULT\_REGION=

aws ec2 create-key-pair --key-name workshop |jg -r '.KeyMaterial' > workshop.pem

chmod 400 workshop.pem

curl -o exporters.yml <https://raw.githubusercontent.com/vfarcic/docker-flow-monitor/master/stacks/exporters-aws.yml>

docker stack deploy -c exporters.yml exporter

docker service create --name util --network monitor –mode global alpine sleep 100000000

## Bäh

* Nutzung von Swarm ohne Kenntnisse von Swarm
* Leinwand verdeckt
* Akzent teilweise unverständlich
* Workshop nicht geeignet zum Mitmachen
* Kalt

## Stack

Prometheus, Altermanager, docker-flow-monitor (GiteHub Projekt von vfarcic)

Docker Swarm

Jenkins

# Building a Modern CI/CD Pipeline (with Docker and Kubernetes)

Anton Weiss

ngrok for sharing

seit kubernetes 1.8 kann man mounten

# Building a Self-Sufficient System (Viktor Farcic)

* Immutable Servers are key 🡪 same environment and config on each system 🡪 images needed
* Als Sprecher ist er besser als als Workshopleiter
* Apache Mesos wird nicht mehr richtig unterstützt; viele Entwickler gegangen, tot gesagt
* Service Discoveries
  + Apache Zookeper 🡪 tot
  + etcd (kubernetes)
  + …
* ELK: wichtig vor allem für verteilte Anwendungen (bei denen nicht klar ist, in welchem Container auf welchem Server in welchem Pod der Fehler aufgetreten ist)
* Dashboards „sind von gestern“: 95% der Aktionen können automatisiert werden

## Eindrücke

### Architektur und Technologien

Die heutige Software-Entwicklung benötigt Architekturen, in denen Tools und Libraries leicht ausgetauscht werden können: Technologien ändern sich rasend schnell (nicht nur bei JS!); alte werden discontinued, und neue, bessere erfunden.

Problem: das ist bisher Fantasie, allein die DB kann nicht einfach ausgetauscht werden (und es gibt gute Gründe dafür!).

Diese Anforderung gleicht dem Wunsch nach einer Software-GUT…

Allerdings: durch das Aufsplitten der Software in kleinste Artefakte (loosely coupled microservices) wird das Upgraden einzelner Elemente erst möglich – im Gegensatz zu Monolithen.

### Slack und Informationen

(betrifft auch Dashboards, ELK / Logging, etc.)

Slack ist trending und im DevOps Rahmen sehr beliebt. Aber ich habe das Gefühl, es ist „nur“ eine inkrementelle Verbesserung gegenüber Email. Zu viel Detail, von dem wir noch abstrahieren müssen. Tools helfen, wenn sie die für mich unwichtigen Infos ausfiltern (aber nicht löschen), und nur eine Helikopter-Sicht zeigen.

# Stateful Applications on Kubernetes (Nicolas Byl)

Jeder Microservice sollte seine eigene Db verwenden (bei uns: Schema).

## Requirements

* Scalable
* Highly available
* Fault tolerance

## Kubernetes Architektur

* Cluster > Node > Pod > Container > Service
* Namespace (z.B. Prod, RC, …) bündelt Services (in ihren Containern in ihren Nodes)
* Node besteht aus Pods
* Pods enthalten Container

## Statefulness

* Pod bekommt eine ClaimRef
* PVClaim (PersistentVolumenClaim) ist das Brückenglied zwischen Pod und Speicher

### Kubernetes unterstützt

* GCEPersistentDisk
* CephFS
* AWSElasticBlockStore
* Cinder
* AzureFile
* Glusterfs
* AzureDisk
* VsphereVolume
* … (many others)

### Access Modes

* ReadWriteOnce (single node)
* ReadOnlyMany (many nodes)
* ReadWriteMany (many nodes)

## Resources

<https://kubernetes.io/docs/concepts/storage/persistent-volumes/>

<https://kubernetes.io/docs/tutorials/stateful-application/basic-stateful-set/>

## Sonstiges

<https://kubeapps.com/> (von <https://bitnami.com/>)

<https://helm.sh/> (Kubernetes Package Manager)

<https://www.cncf.io/> (Cloud Native Computing Foundation)

# Docker: Ops unleashed (Mario Kleinsasser)

System Operator bei Strabag

Drei Personen betreiben 1545 Services (davon er auf der DevOps, einer krank)

<https://labs.play-with-docker.com/>

<https://github.com/n0r1sk/bosnd>

<https://github.com/n0r1sk/bosnd/tree/master/examples/apache-httpd-tomcat>

## Erkenntnis

CD evaluieren & einsetzen

# Architecture Patterns for Microservices in Kubernetes (Thomas Fricke)

## Namespaces

|  |  |
| --- | --- |
| Cgroup | Cgroup root dir |
| IPC | System V IPC, POSIX message queues |
| Network | Network devices, stack, ports, etc. |
| Mount | mount points |
| PID | process ids |
| User | User and group IDs |
| UTS | Hostname and NIS domain name |

<https://www.usenix.org/system/files/conference/hotcloud16/hotcloud16_burns.pdf>

* Sidecar
  + OpenShift OAuth proxy
  + Logging
* Leader Election
  + <https://github.com/zalando/patroni> (Postgres High Availability)
  + Zookeeper
* Locomotive
* Separation of Control & Communication
* Linking Containers / Wormhole
  + Geeignet für Build Pipelines
  + Nicht geeignet für Produktion, Multi-Tenancy (Security risks)
* Distributed Patterns
  + Client-Server
  + Layers
  + Message Queue
  + Cattle-Pets
  + Replication

Social Antipattern

* Kubernetes on premise (Übertreibung)

# The Modern IT Stack (Christian Daschner, Patrick Gruhn)

Cloud Native Logging Stack (kann ELK ersetzen):

1. cAdvisor
2. Prometheus (Data store)
3. Grafana (Visualization)

# Kubernetes Patterns (Roland Huß)

<https://github.com/ro14nd-talks/kubernetes-patterns>

## Concepts

ReplicaSet

* Nanny for Pods
* Hat ein Template für Pods

Service

* Entrypoint for a set of Pods

Predictable Demands

Runtime Deps

* Persistent Volumes
* …

Resource Profiles

* Resource (CPU, Network: compressible; Memory: incompressible)
* App: declaration of resource **requests** and **limits**
* Platform

QOS Classes

* Best effort: No requests or limits
* Burstable: requests < limits
* Guaranteed: requests = limits

Deployment

* Declarative vs. Imperative
* Holds template for Pod
* Strategy
  + Rolling: nach und nach ersetzen (für stateful: funktioniert nur, wenn Schemas abwärtskompatibel sind)
  + Canary: einen ersetzen
  + Blue-Green (deploy all new ones, then remove all old ones 🡪 twice the resources)
  + Recreate: alte herunterfahren, neue hochfahren
* Init containers

Sidecar

* Ambassador/Proxy, e.g. circuit breaker
* Adapter, e.g. monitoring, logging

Configurational Patterns

* EnvVar
* Configuration Resource (ConfigMap and Secret)
* Immutable Configuration (K8s flexVolumes)

ServerSpec: testing der docker Infrastruktur

# Lightning Talks

DTrace / OpenSnoop: prevent dev env attacks

# Rancher (Jan Bruder)

* Serves AWS, Azure, GCP
* Runs on top of: Kubernetes, Mesos, Swarm, or self-running
* Integration mit Prometheus, ELK, fluentd, etc.
* Integration mit Active Directory, SAML, LDAP
* Bringt User/Gruppen/Rollen mit
* 1.x: built on Docker, stand up k8s clusters
* 2.0: built on k8s, accelerate container adoption (GA erscheint 2. Quartal 2018)
* Vereinfacht Verwaltung von k8s; man kann kubectrl neben Rancher verwenden
* 4-6 Wochen nach einem k8s Release erscheint die Bündelung von Rancher mit diesem Release (da Rancher ein angepasstes k8s ausliefert)

## Komponenten

* Cattle
* RKE: Rancher Kubernetes Engine

# Modern Monitoring for Dev and Ops (Björn Rabenstein)

Bei Soundcloud, eh. Google, Prometheus contributor

<https://github.com/beorn7/talks>

Blackbox Monitoring (Ops, e.g. using Nagios – classic) vs. Whitebox Monitoring (Dev, e.g. using Prometheus – modern)

Site Reliability Engineering: How Google runs production system (Buch von Niall Murphy et. al.)

* Symptom-based alerting: differentiate symptom and cause
  + Symptoms should create pages (e.g. slack message)
  + Causes should create tickets
  + Check symptoms with blackbox monitoring
  + Check causes with whitebox monitoring (predictive)
* Predictive monitoring
  + Loss of redundancy
  + Disk running full in 30 mins
* Blackbox monitoring is not sufficient
  + Probing a test target is not what real users see
  + Latenz über alle Anfragen ist wichtig 🡪 probes reichen nicht, sondern Prometheus data
  + You still need to investigate causes

More Resources:

* Brendan Greeg’s Use method: utilitization, saturation, and error count (rate)
* Tom Wilkie‘s RED method
* „The four golden signals of monitoring: …“ (see slides with red quotations)
* Blue Book

# Docker in AWS (Govinda Fichtner)

* VPC = Virtual Private Cloud is a Software defined network
* Auto Scaling Group
* CloudFormation: Infrastructure as Code
* IAM = Identity and Access Management; Policies werden in JSON formuliert
* Route 53: DNS von AWS
* Docker CE = Docker for AWS
* Docker EE = Docker CE + Unified Control Panel (Mgmt. Console), RBAC (User, Teams, Roles), LDAP, Security Scanning, Docker Trusted Registry, Image Management, Support
* Kops: einfaches Aufsetzen von Kubernetes
  + Automated provisioning of k8s clusters in AWS and GCE
  + export AWS\_PROFILE=
  + export KOPS\_STATE\_STORE=
  + export NAME=
  + kops create cluster
* kube-aws: (zukünftig) ähnlich wie kops
* ECS: Elastic Container Service
* Man kann eigene AMI (Amazon Machine Image) erzeugen
* EKS: (zukünftig) Elastic Kubernetes Service

# Cloud-native Microservices (Timo Derstappen)

## Controllers

* Consistent distributed storage (etcd)
* Resource definition (PODs, PV, NetworkPolicies, …)
* Resources define desired state in the cluster
* Controllers transform desired state into actual state

## Operator

* Custom controller
* Uses a CustomResourceDefinition (CRD)
* You can add CustomResources (CR) to your cluster
* The operator will watch the CRs and act on them

Ex.: Prometheus Operator, ElasticSearch Operator

## CD

1. Wenn quay.io gebaut hat: deployment-event an github
2. draughtsman überwacht das deployment-event
3. tiller zieht sich das Artefakt von quay.io
4. das Artefakt wird in kubernetes deployt

## versionbundle

* expose version info & changelog in each service
* apigateway automatically consolidates each possible combination of services that are supported

## Best practices

* Minimize usage of hand written libraries!
* Avoid a monoculture!
* Use circuit breakers and exponential backoff to sub-services

## Service Mesh

* istio.io (currently not for production – cannot be updated!)

# Tear down silos (Konstantin Diener)

* Cross functional teams (Ops, FE-Dev und BE-Dev im gleichen Team)
* T-shaped personalities (everybody is able to perform each job)
* Shippable product at end of sprint

<http://web.devopstopologies.com/>

Probleme

* „I’m alone“
* Missing knowledge sharing
* No collective learning
* Isolated technology decisions

🡪 kann gelöst werden durch teamübergreifendes Teilen der jeweiligen Rollen untereinander („Community of Practice“): DevOps, BE, FE, Support, Marketing, …

## BizDevOps

* Zusätzlich Product Owner, Support, Business Analysts, Marketing im Cross-functional Team

Je nachdem, ob die jeweilige Einheit (Marketing, …) am meisten mit dem Cross-functional Team oder mit Leuten der gleichen Funktion arbeitet, ist die Einheit im Cross-functional Team oder in ihrer Abteilung.

## #noprojects

Movement, um Produkte anstelle von Projekte zu stellen.

Offen u.a.: wie wird Budget gehandhabt?

# Fazit

* <https://www.usenix.org/system/files/conference/hotcloud16/hotcloud16_burns.pdf> lesen
* <https://github.com/zalando/patroni> (Postgres High Availability)
* Slack-Artikel lesen (Zeitschrift)
* Buch „The Phoenix Project“ lesen
* CD evaluieren (siehe auch Beispiel von Timo Derstappen)
* Kubernetes ansehen, üben – im Zusammenhang mit Rancher 2.0
* Prometheus testen
* Packer ansehen