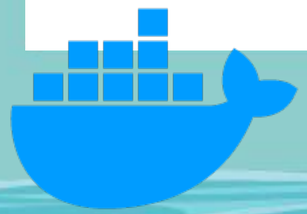
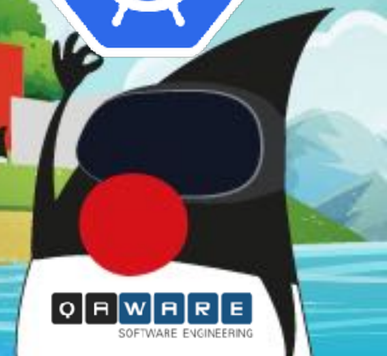
www.javaland.eu

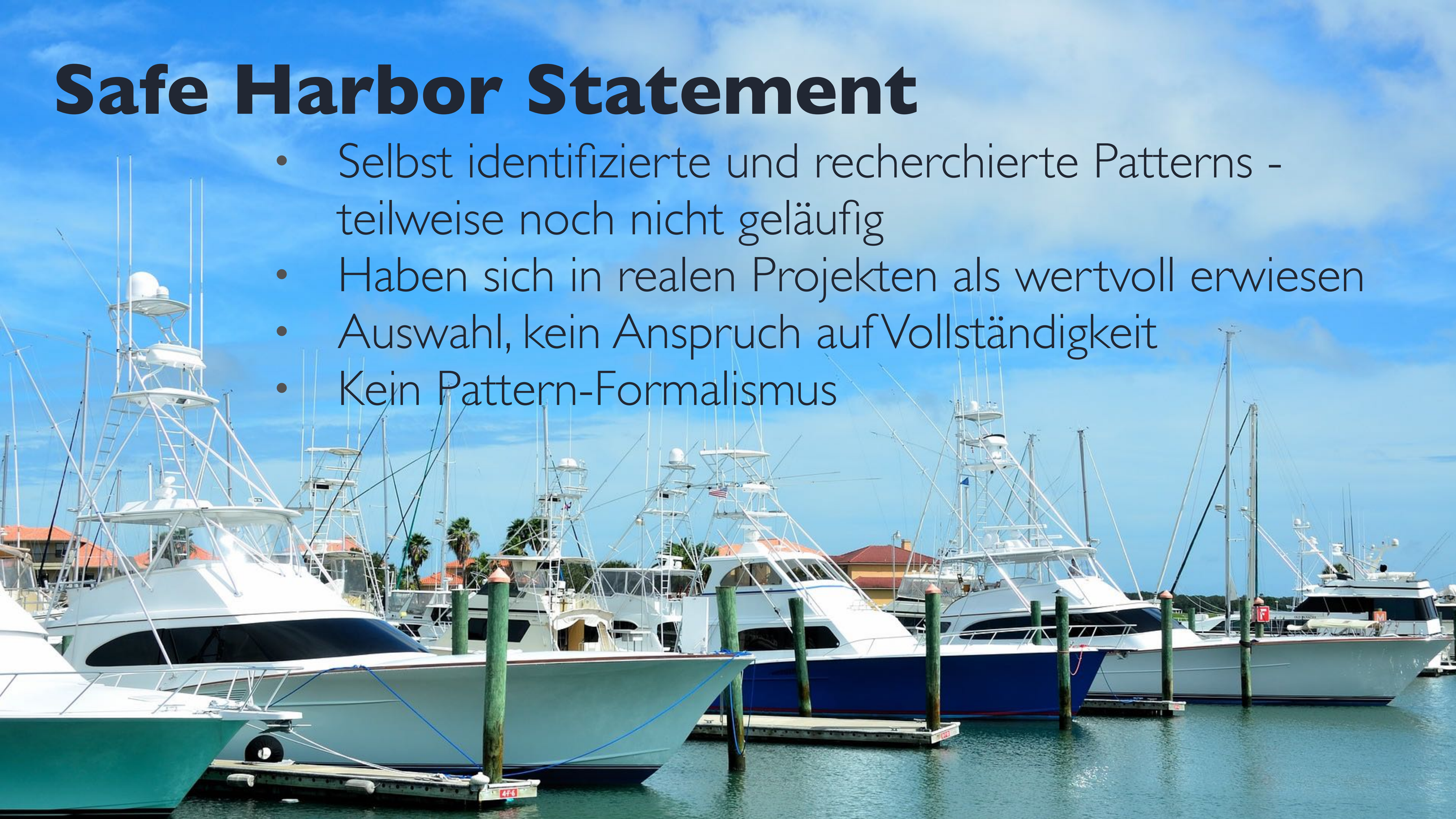
Docker und Kubernetes Patterns & Anti-Patterns





Safe Harbor Statement

- Selbst identifizierte und recherchierte Patterns - teilweise noch nicht geläufig
- Haben sich in realen Projekten als wertvoll erwiesen
- Auswahl, kein Anspruch auf Vollständigkeit
- Kein Pattern-Formalismus



HYPERSCALE

RESILIENT

SPEED

OPEX SAVINGS

BUILD AND COMPOSED

AS MICROSERVICES

CLOUD NATIVE APPLICATIONS

PACKAGED AND

DISTRIBUTED AS CONTAINERS

DYNAMICALLY

EXECUTED IN THE CLOUD

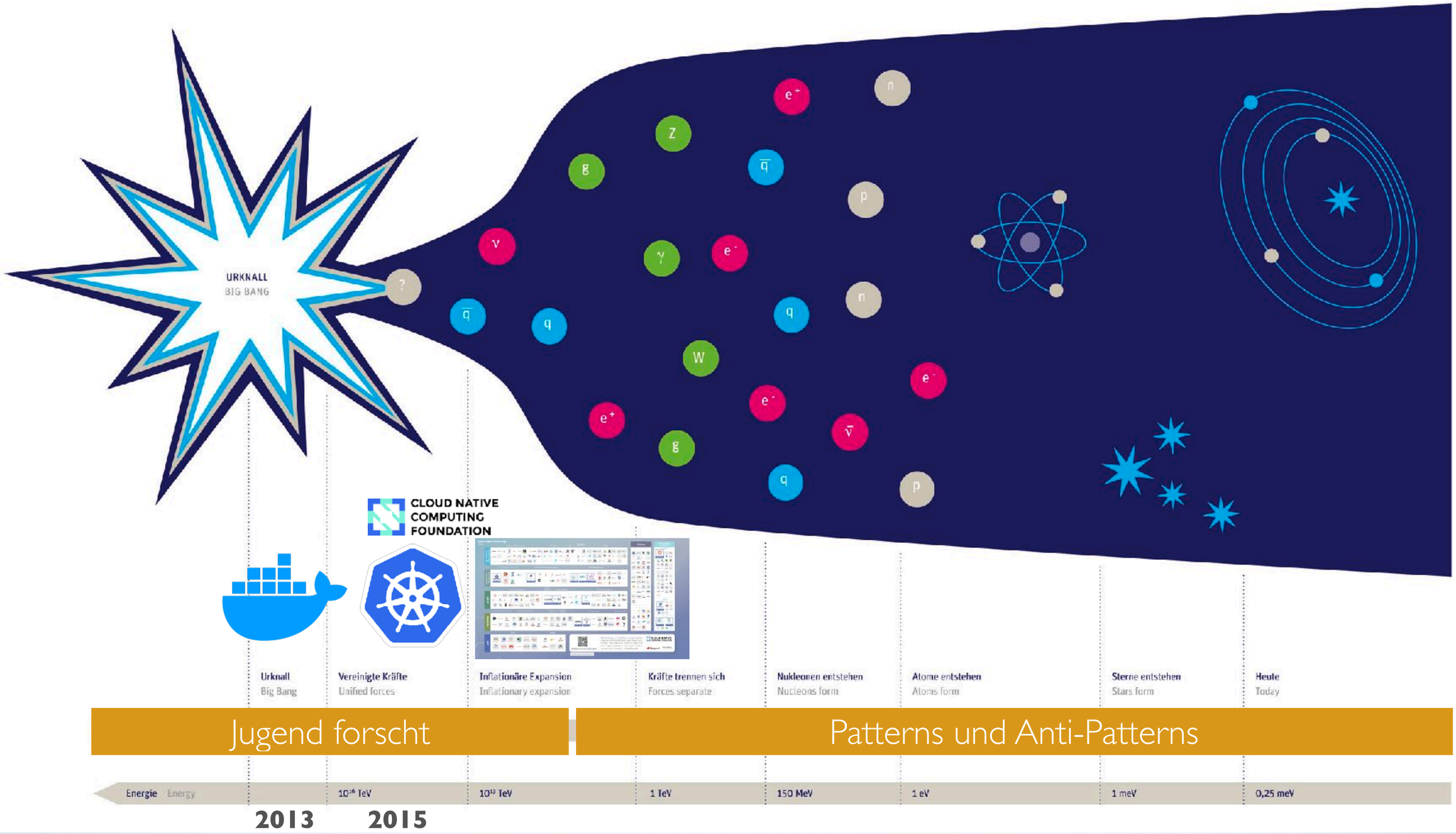

Lasst uns
Cloud Native
Anwendungen
bauen!

WIE ?????

… so dass es mir bei der Entwicklung und in Produktion nicht auf die Nase fällt.


CLOUD NATIVE HELL

Modus: Jugend forscht



Jugend forscht

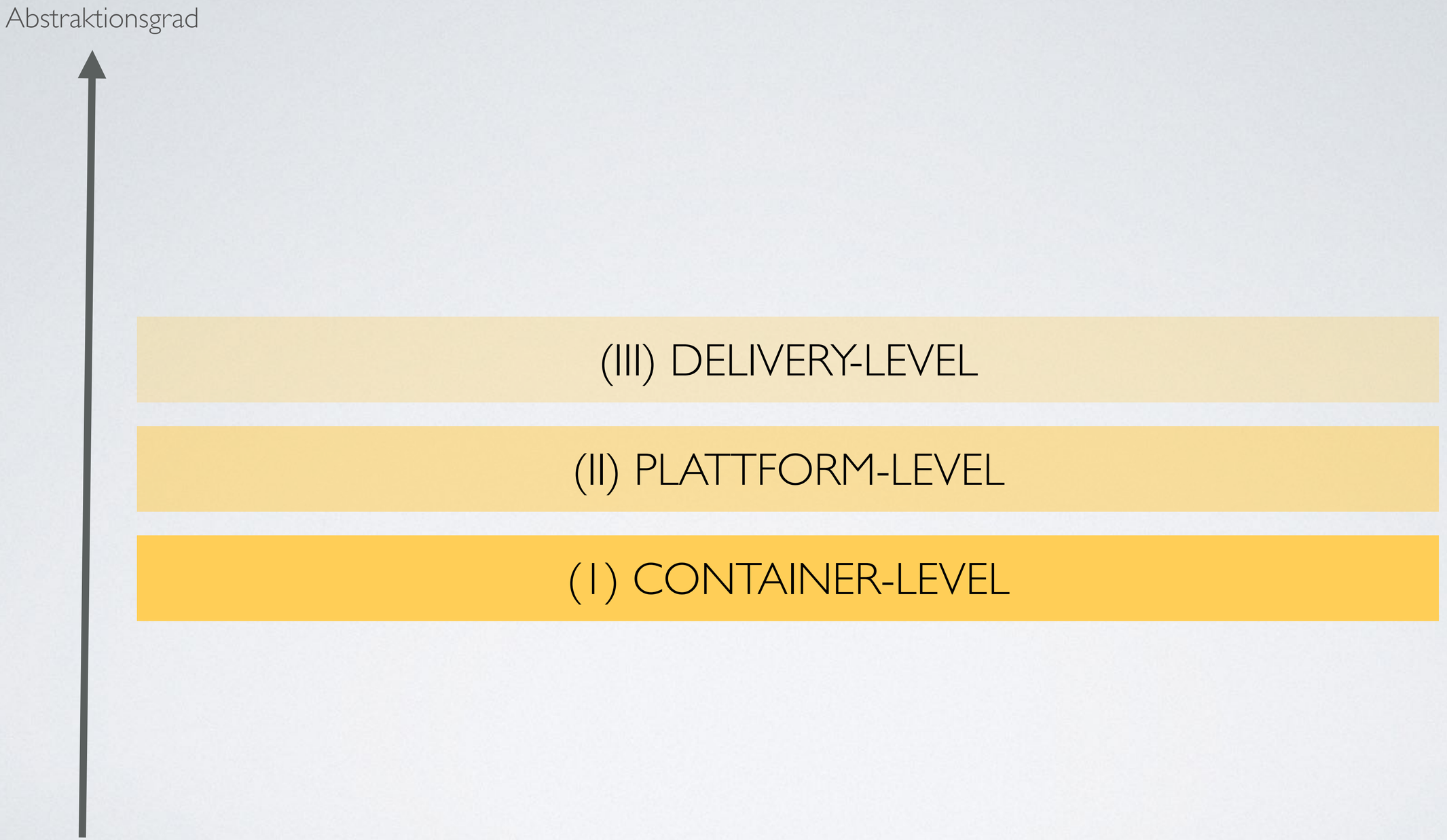
Patterns und Anti-Patterns

https://www.weltmaschine.de/physik/geschichte_des_universums

Sweetspot für Docker/k8s Patterns/Antipatterns

- Abstraktion stabil genug
- Konkrete Beispiele möglich

```
docker run -it -v /var/run/docker.sock:/var/run/docker.sock qaware/devbox
```


Abstraktionsgrad
(III) DELIVERY-LEVEL
(II) PLATTFORM-LEVEL
(1) CONTAINER-LEVEL


(III) DELIVERY-LEVEL
(II) PLATTFORM-LEVEL
(I) CONTAINER-LEVEL

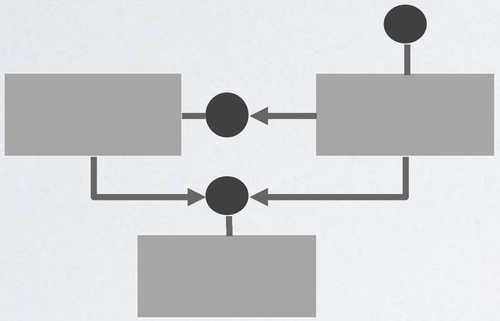
OPS COMPONENT

DESIGN

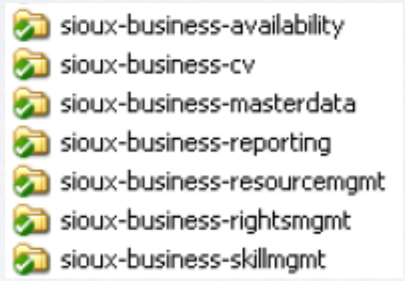
BUILD

RUN

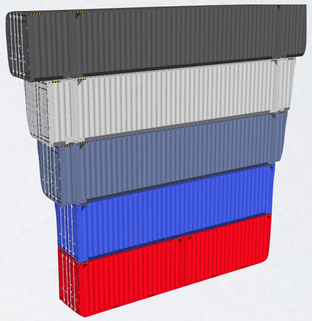
Design Components



Dev Components



Ops Components



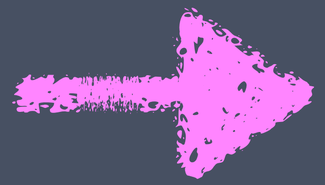
- Complexity unit
- Data integrity unit
- Cohesive feature unit
- Decoupled unit

+

- Planning unit
- Team assignment unit
- Development unit
- Integration unit

+

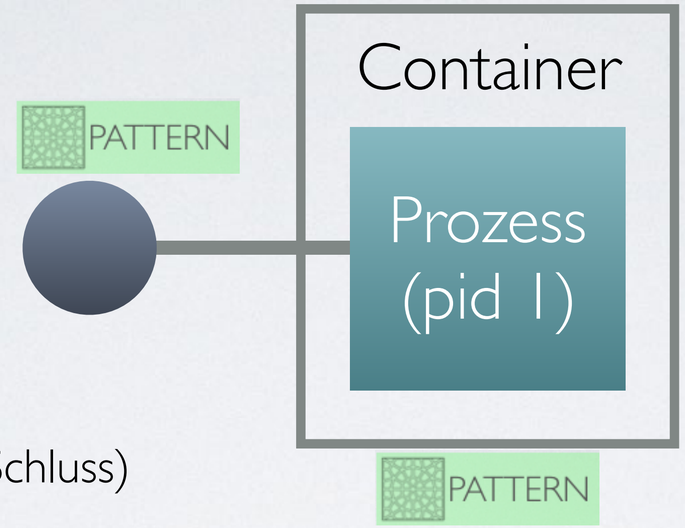
- Release unit
- Deployment unit
- Runtime unit
- Scaling unit



- **Container** = Verpackung für Ops Components
- **Standard-Schnittstellen** für Standard-Betriebsprozeduren
- **Einfach zu transportierende, schnell zu startende und mit wenig Overhead ausführbare** Software-Einheiten

Container Interface

- EXPOSE von allen
  von außen zugänglichen Ports
  `EXPOSE 80 443`
- Alle Umgebungsvariablen,
  die von außen gesetzt werden können
  per ENV definieren und möglichst mit
  sinnvollem Default-Wert besetzen
  `ENV NGINX_VERSION 1.9.9`
- Dockerfile als Interface Definition
  - Kommentare und `LABEL` [5]
  - `ENV`, `EXPOSE` und `VOLUME`
    (möglichst im Block und ganz am Schluss)
- ENTRYPOINT für Prozessstart und
  CMD für Default-Argumente
  `ENTRYPOINT ["/entrypoint.sh"]`
  `CMD ["--db", "localhost","--user", “root”]`
- Standard-Entrypoints
  (z.B. `docker run my-container /usr/bin/test`)
  - `run`: Container produktiv laufen lassen (default)
  - `run-dev`: im Dev-Modus laufen mit z.B. Verbose Log
  - `test`: Testfälle im Container durchlaufen lassen
  - `HEALTHCHECK`: einen Healthcheck durchführen
  - `debug`: eine passende interaktive Shell öffnen
  - `help`: einen Hilfe zur Verwendung anzeigen



Well-behaved Process

- reagiert auf SIGTERM [1] oder
  definiert ein `STOPSIGNAL` [2] für einen
  würdevollen Abgang
- gibt sinnvolle Exit Codes zurück [3]:
  0 = OK, 1 = allgemeiner Fehler, …
- schreibt Log-Ausgaben auf
  STDOUT/STDERR [4], damit sie per
  Docker Log Driver verschifft werden
  können
- Vordergrund-Prozess, kein
  Daemon- oder Hintergrund-Prozess
  `CMD ["nginx", "-g", "daemon off;"]`

[1] https://medium.com/@gchudnov/trapping-signals-in-docker-containers-7a57fdda7d86
[2] https://docs.docker.com/engine/reference/builder/#stopsignal
[3] http://tldp.org/LDP/abs/html/exitcodes.html
[4] https://success.docker.com/article/Docker_Reference_Architecture-_Docker_Logging_Design_and_Best_Practices
[5] http://label-schema.org/rc1
[6] https://alexei-led.github.io/post/docker_testing

BEISPIEL: LABELS

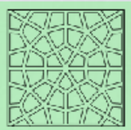
`docker.cmd`	`org.label-schema.docker.cmd=` `"docker run -d -p 5000:5000 -v config.json:/etc/config.json myapp"`	How to run a cont the image under th runtime.
`docker.cmd.devel`	`org.label-schema.docker.cmd.devel =` `"docker run -d -p 5050:5050 -e ENV=DEV myapp"`	How to run the co development mod Docker runtime e.g tooling or more ve
`docker.cmd.test`	`org.label-schema.docker.cmd.test =` `"docker run myapp runtests"`	How to run the bu suite for the image Docker runtime. M tests then exit, retu on stdout and exit zero exit code on f
`docker.cmd.debug`	`org.label-schema.docker.debug =` `"docker exec -it $CONTAINER /bin/redis-cli"`	How to get an app interactive shell fo the container unde
`docker.cmd.help`	`org.label-schema.docker.cmd.help =` `"docker exec -it $CONTAINER /bin/app --help"`	How to output help image under the d The container MUS information to stdo exit.

http://label-schema.org/rc1

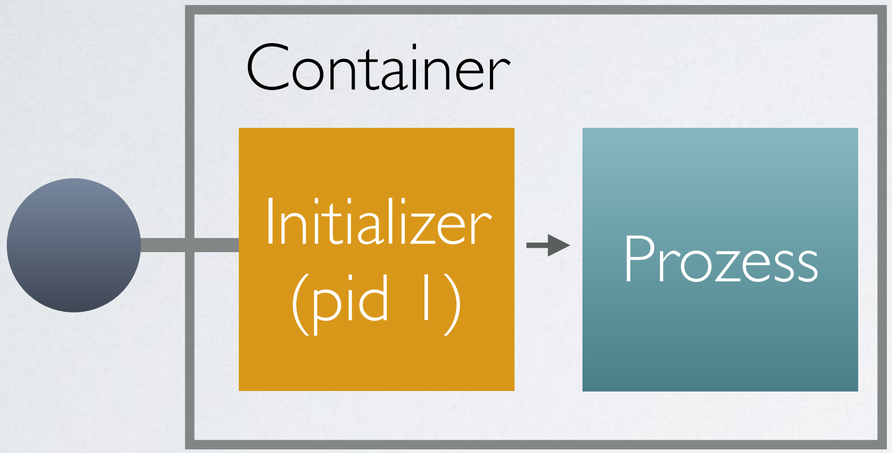
Tags	latest 0.9.1	
Created	January 18, 2017 at 10:49 AM	
ID	63271c99d6fb	
Maintainer	Ross Fairbanks "ross@[hidden]"	
Download Size	23.4 MB	
Git Commit	45b22cb	
License	Apache-2.0	
Labels	com.microscaling.docker.dockerfile	/Dockerfile
	com.microscaling.license	Apache-2.0
	org.label-schema.build-date	2017-01-18T09:49:02Z
	org.label-schema.description	Our Microscaling Engine provides automation, resilience and efficiency for microservice architectures. Experiment with microscaling at app.microscaling.com.
	org.label-schema.name	Microscaling Engine
	org.label-schema.schema-version	1.0
	org.label-schema.url	https://microscaling.com
	org.label-schema.vcs-ref	45b22cb
	org.label-schema.vcs-url	https://github.com/microscaling/microscaling.git
	org.label-schema.vendor	Microscaling Systems
	org.label-schema.version	0.9.1

https://microbadger.com/images/microscaling/microscaling

CONTAINER INITIALIZER



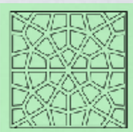
PATTERN

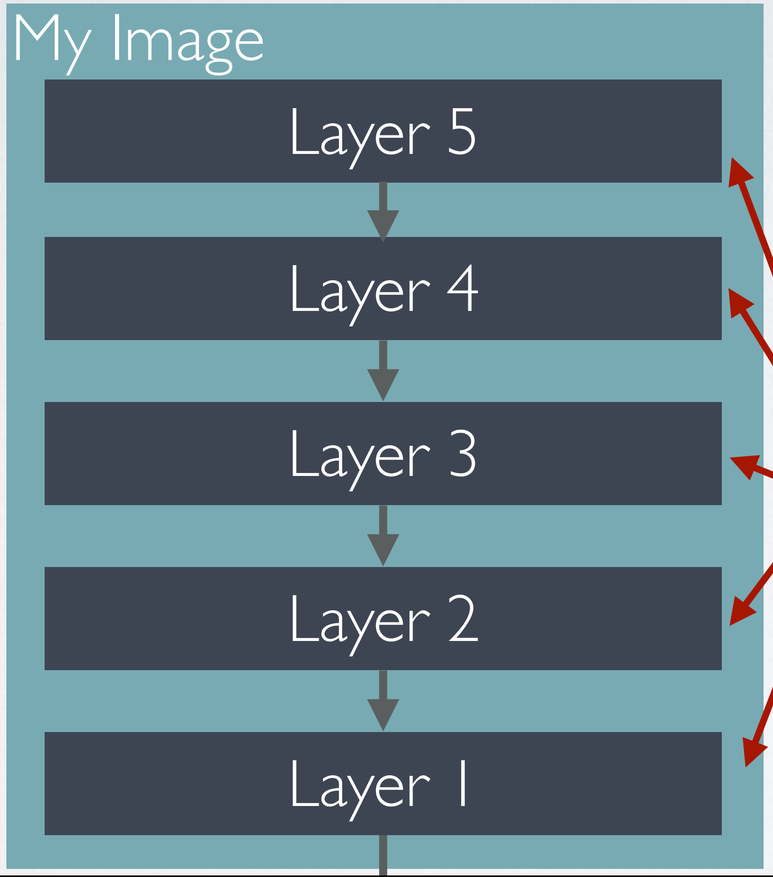


	Prozess-management (Exit, SIG, Zombies)	Log-Umleitung (syslog, files)	Config-Dateien schreiben	Cron	Warten auf TCP/HTTP Endpunkt
Chaperone (veraltet)	x	x	x	x	
Dockerize		x	x		x
Tini (in Docker > 1.13 enthalten)	x				
dumb-init	x				
pid1	x				
TrivialRC	x				
phusion baseimage	x	x		x	

… oder eigenes Shell-Skript oder Go-Programm (aber Achtung: Prozess stets mit `exec` starten!)

IMAGE LAYER ARCHITECTURE


PATTERN



Niemals `latest` Tag nutzen!

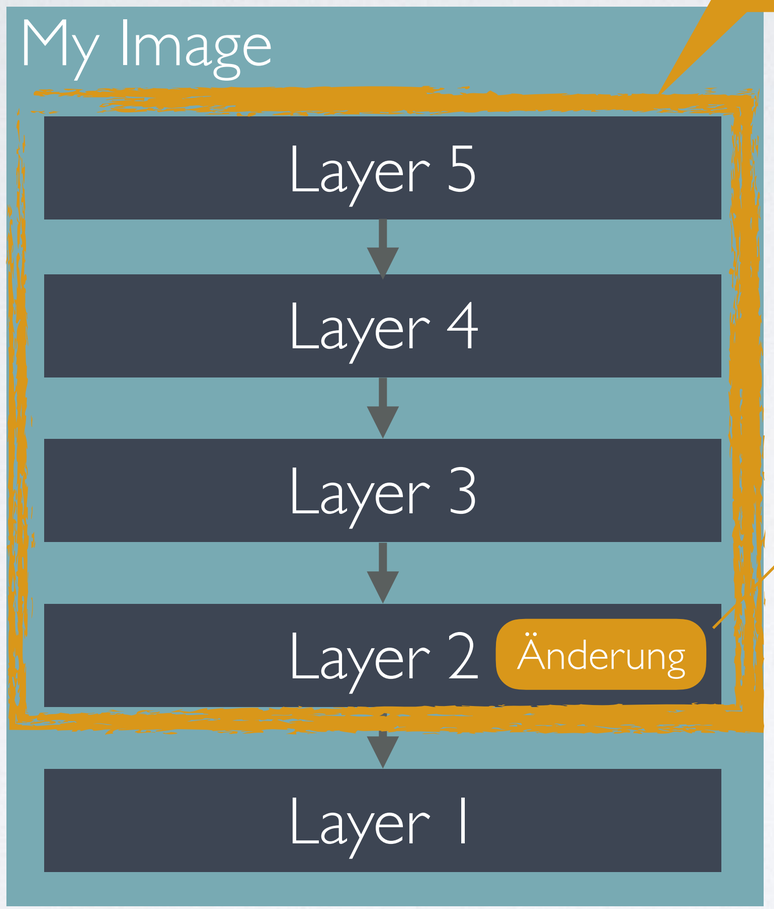

ANTIPATTERN

```
FROM debian:jessie

ENV NGINX_VERSION 1.12.2-1~jessie

RUN apt-get update && \
    apt-get install -y ca-certificates && \
    nginx=${NGINX_VERSION} && \
    rm -rf /var/lib/apt/lists/*

RUN ln -sf /dev/stderr /var/log/nginx/error.log

EXPOSE 80 443

CMD ["nginx", "-g", "daemon off;"]
```

jede Instruktion im Dockerfile erzeugt einen neuen Layer (manche einen mit 0 Bytes)

```
$ docker history --no-trunc c3e0e82fb8ec
IMAGE                                                                       CREATED              CREATED BY                                                                                                SIZE
sha256:c3e0e82fb8ec8175b34a412b47b41f642e29c72dee61f2338062aa436c50ef6d     About a minute ago   /bin/sh -c #(nop)  CMD ["nginx" "-g" "daemon off;"]                                                       0B
sha256:885bbf498551d8ef13a3d79eb2a8d658f7e931d79e070d843f1dbc3fe8248cf1     About a minute ago   /bin/sh -c #(nop)  EXPOSE 443 80                                                                          0B
sha256:86a1ceb9b25982eb599233cd1dee0b0d1616ea52498403aed100615cf66de402     About a minute ago   /bin/sh -c ln -sf /dev/stderr /var/log/nginx/error.log                                                    11B
sha256:e31a00b43629ebcc4893736f70deda5539863a06fc80657bec745078d34c75c8     About a minute ago   /bin/sh -c apt-get update &&     apt-get install -y ca-certificates nginx &&     rm -rf /var/lib/apt/lists/*   63.7MB
sha256:9fa8f57f57fa492fdceb20127649852a90360afa3440adc560507b44c468f68e     4 minutes ago        /bin/sh -c #(nop)  ENV NGINX_VERSION=1.12.2-1~jessie                                                      0B
sha256:ce40fb3adcc648d2e2c6bdb602cdbee35156bc2a41cb3e73b069f0b1bf1bcf97     3 weeks ago          /bin/sh -c #(nop)  CMD ["bash"]                                                                           0B
<missing>                                                                   3 weeks ago          /bin/sh -c #(nop) ADD file:f1509ab9c2cd3810736e26739fa0f78ee1ba942e14498ba5f266d8a78e664acc in /           123MB
```

Cache wird invalidiert [1]:
Muss neu gebaut und
transportiert werden!

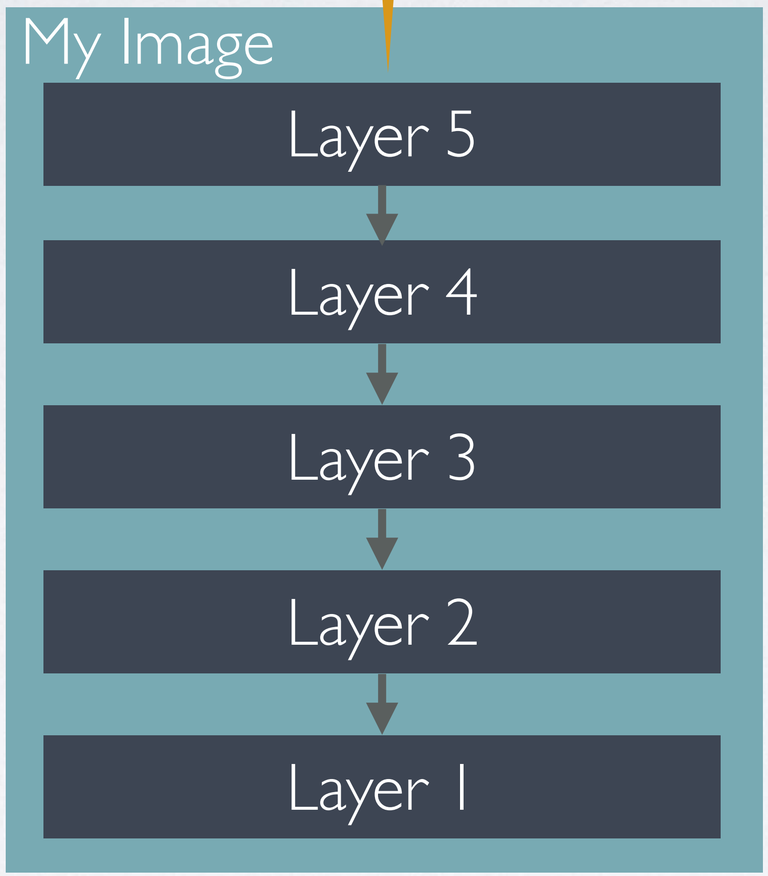


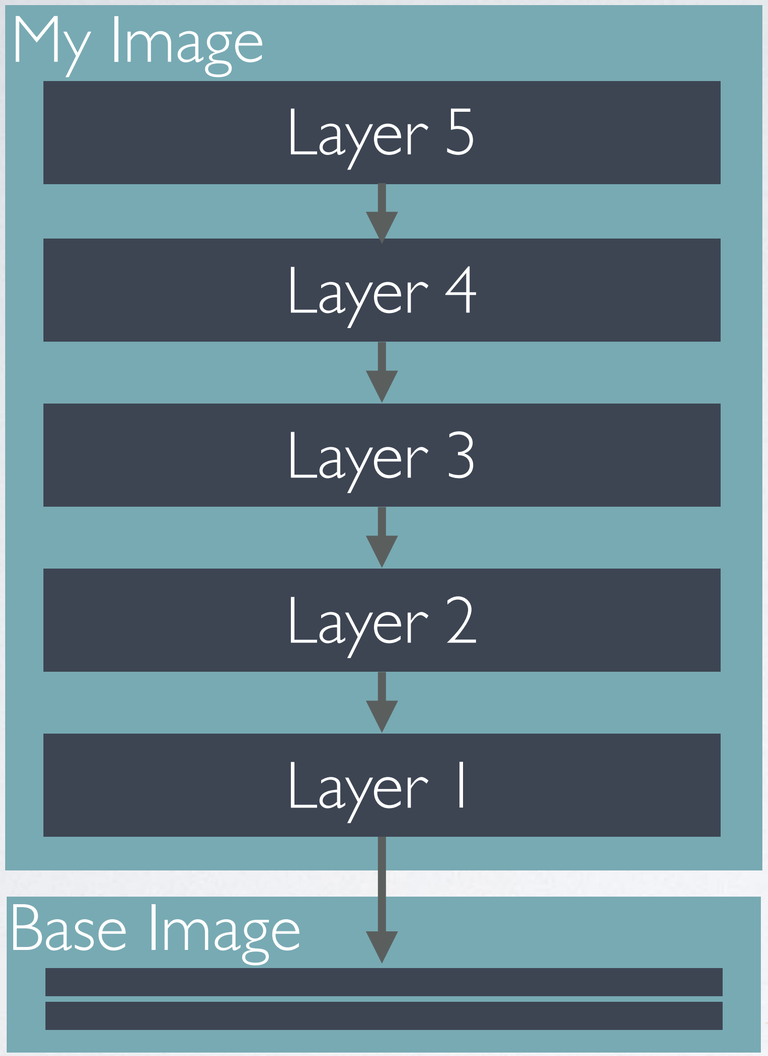

```
FROM debian:jessie

ENV NGINX_VERSION 1.12.2-1~jessie

RUN apt-get update && \
    apt-get install -y ca-certificates && \
    wget nginx=${NGINX_VERSION} && \
    rm -rf /var/lib/apt/lists/*

RUN ln -sf /dev/stderr /var/log/nginx/error.log

EXPOSE 80 443

CMD ["nginx", "-g", "daemon off;"]
```

[1] mehr zu Docker Caching: https://www.ctl.io/developers/blog/post/caching-docker-images

Änderungs- und Transporteinheit

- klein
- geringer Impact von Änderungen





volatil

stabil

klein

groß

1. Abhängigkeiten
2. Applikation
3. Konfiguration
4. Metadaten / Schnittstelle

klein: Alpine-*, Baseimage-Docker, Busybox, Scratch

- Minimale Docker Images mit Java 9: https://blog.dekstroza.io/building-minimal-docker-containers-with-java-9
- Security Checks von Docker Images: Clair & docker-bench-security & dockscan

IMAGE SHRINKING 101

- Unnötige Layer vermeiden
  - **RUN** Chaining:

    ```
    RUN apk add --update wget git && rm -rf /var/cache/apk/*
    ```
  - Alle Layer verschmelzen zu einem (nur bei Basis-Images empfehlenswert):

    docker export + **docker** import

- Platzschonende Installation von Paketen

  ```
  RUN apt-get update && apt-get install -y –no-install-recommends apache2 wget && apt-get clean && rm -rf /var/lib/apt/lists/*
  ```

IMAGE METAMORPHOSIS

ANTIPATTERN

DEV

INT

PROD

myimage-dev

myimage-int

myimage-prod

myimage

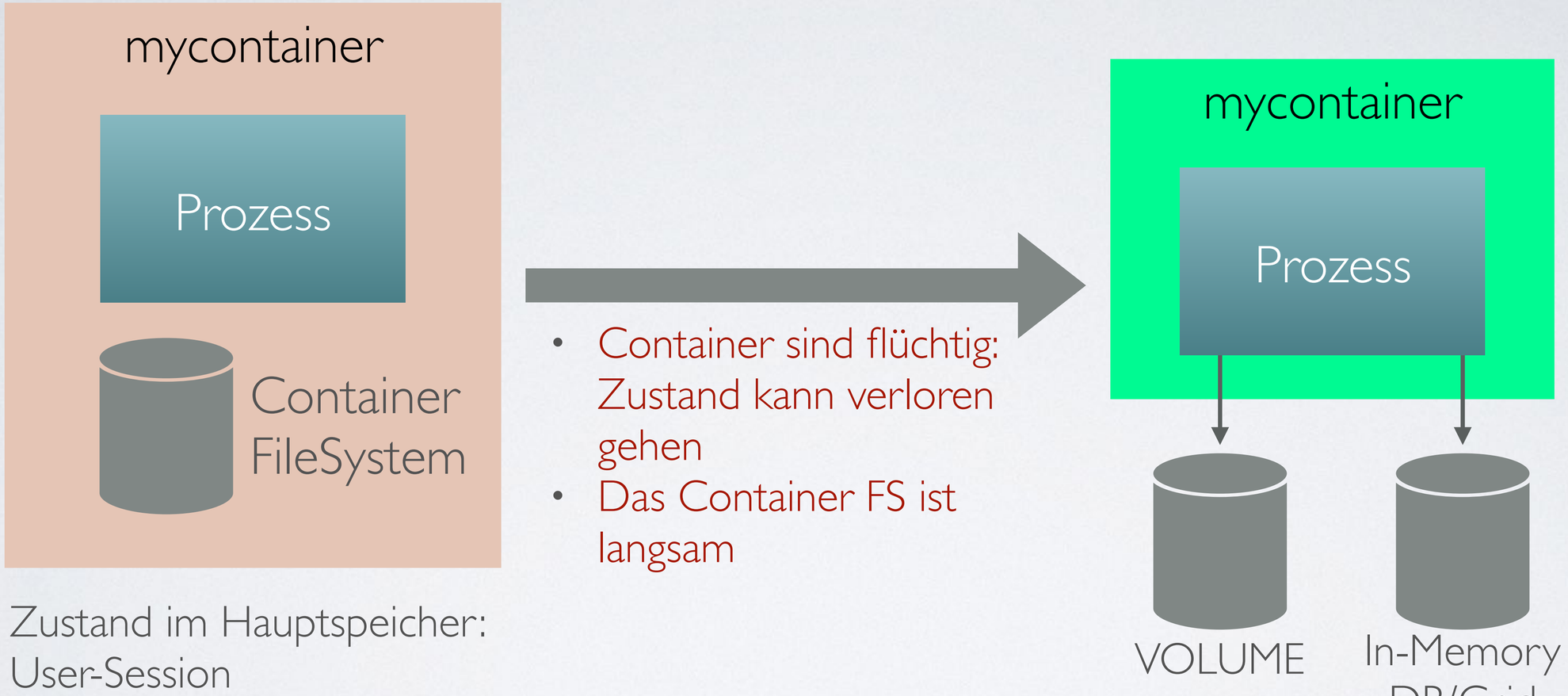
DEV.config

INT.config

PROD.config

STATEFUL CONTAINER

ANTIPATTERN



- Container sind flüchtig: Zustand kann verloren gehen
- Das Container FS ist langsam

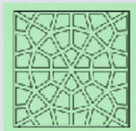
- Zustand im Hauptspeicher: User-Session
- Zustand auf Platte: Logs, Anwendungsdaten

(z.B. redis, Ignite, Hazelcast)

Für Log-Dateien:

```
RUN ln -sf /proc/1/fd/1 /var/log/test.log
```

CONTAINER UNIT TESTING



PATTERN

nginx-container-test.rb

```
describe package('nginx') do
  it { should be_installed }
end

describe port(80) do
  it { should be_listening }
end
```

```
$ inspec exec nginx-container-test.rb -t docker://f80443273223

Profile: tests from nginx-container-test.rb (tests from nginx-container-test.rb)
Version: (not specified)
Target:  docker://f804432732231fc24f696cf7527ce458d3799ec7769961b6fc72021893921945

  System Package nginx
     ✔  should be installed
  Port 80
     ×  should be listening
     expected `Port 80.listening?` to return true, got false

Test Summary: 1 successful, 1 failure, 0 skipped
```

INSPEC

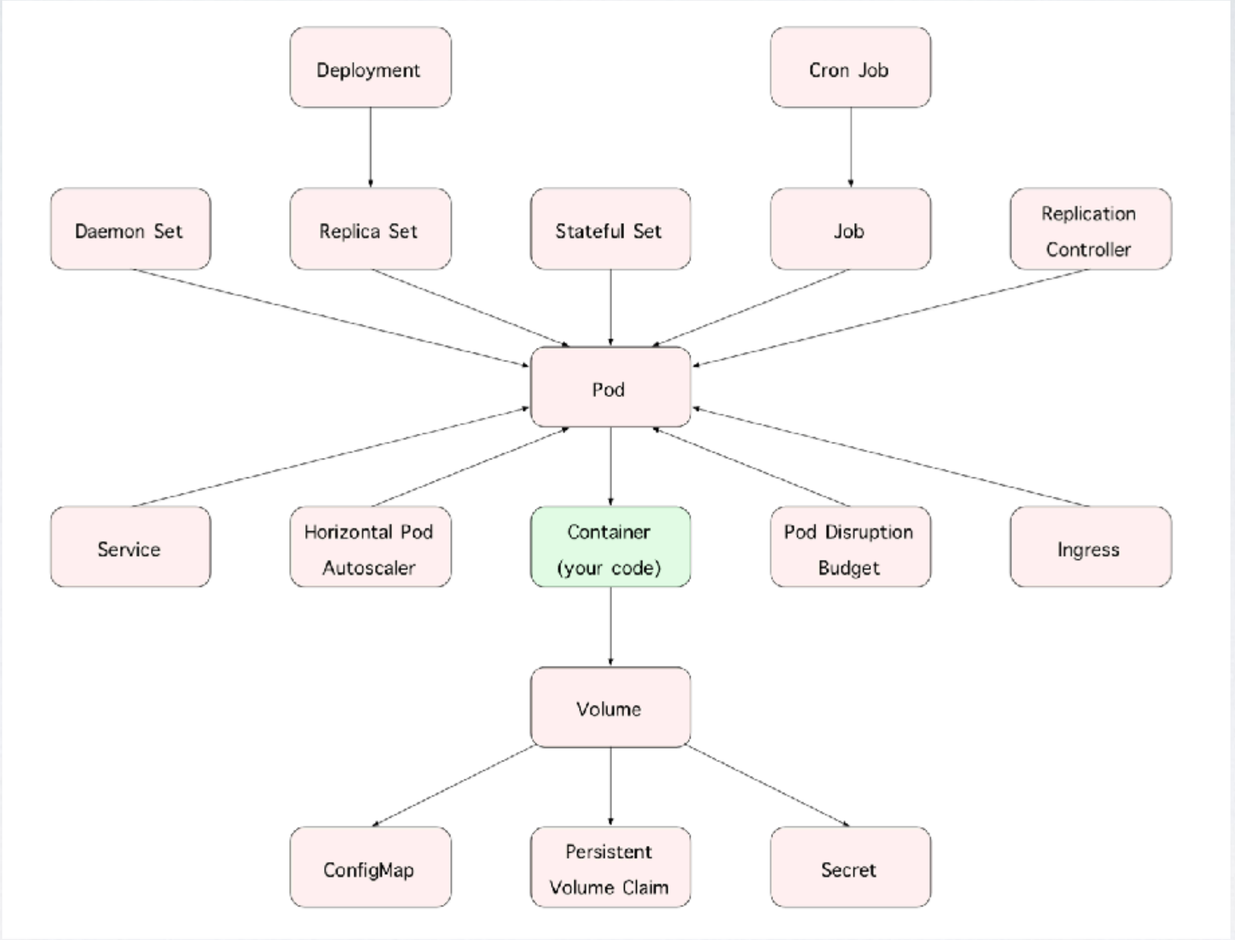
https://www.inspec.io

Alternativen: goss+dgoss, ServerSpec, Bats, Testinfra

(III) DELIVERY-LEVEL

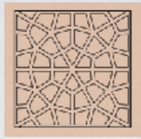
(II) PLATTFORM-LEVEL

(I) CONTAINER-LEVEL

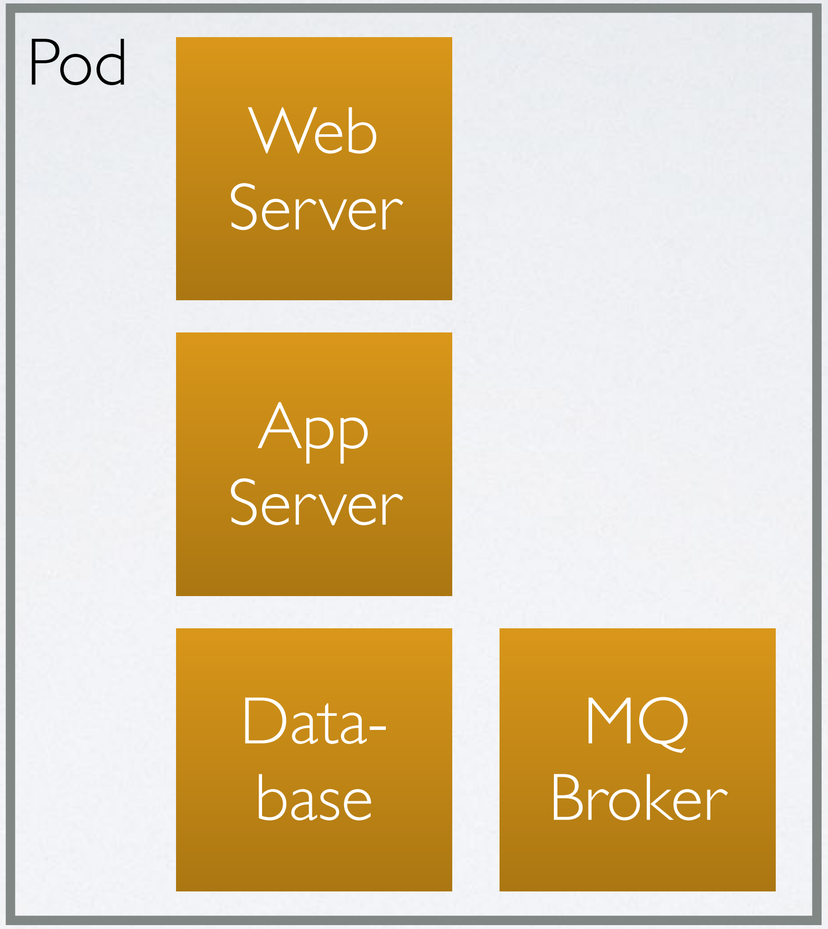


Kubernetes Patterns, Bilgin Ibryam und Roland Huß, https://leanpub.com/k8spatterns

ENTERPRISE POD



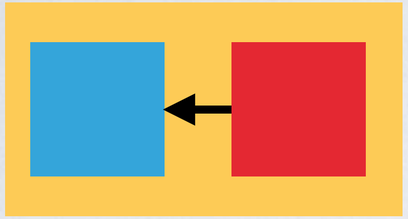
ANTIPATTERN



- verletzt das Single Responsibility Prinzip
- Scheduling schwierig durch hohe Anforderungen
- unzureichende Isolation (Netzwerk, Volumes, IPC, Hostname, PID)

STRUCTURAL CONTAINER PATTERNS

Sidecar (Sidekick): Enhance container behaviour 

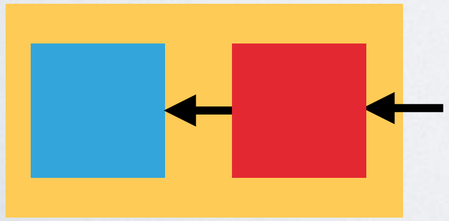


- Scheduling (Quartz)
- Failbot

Ambassador: Proxy communication 

- TLS tunnel (Stunnel, ghosttunnel, istio)
- Circuit Breaking (linkerd, istio)
- Request monitoring (linkerd, istio)

Adapter: Provide standardized interface 



- Configuration (ConfigMaps & Secrets to files)
- Log extraction / re-formatting / shipping (fluentd)

Pod

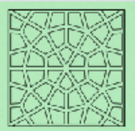

Application Container


Pattern Container


Other Container

“Design patterns for container-based distributed systems”. Brendan Burns, David Oppenheimer. 2016

EXTERNALIZED CONFIGURATION



PATTERN

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: april-aos-runtime-config
data:
  april-aos.properties: |
    com.foo.iap.april.jwt.secret=some-secret
    osmc.username=april
    osmc.url=https://www.magic.works
    ...
  april-feature-togglz.properties: |
    WRITE_TEXT5_TO_SOFAK_INVOICES=false
    ...
  log4j2.xml: |
    <?xml version="1.0" encoding="UTF-8"?>
    <Configuration monitorInterval="60">
      <Appenders> ... </Appenders>
      <Loggers> ... </Loggers>
    </Configuration>
```

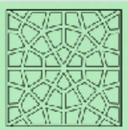
```yaml
spec:
  containers:
    - name: april-aos-runtime
      image: 'april-aos-runtime:1.5.0'
      imagePullPolicy: Always
      ports:
      - containerPort: 8080

      volumeMounts:
      - mountPath: /april/cfg
        name: april-aos-runtime-config-vol

  volumes:
  - name: april-aos-runtime-config-vol
    configMap:
      name: april-aos-runtime-config
```

Umgebungsspezifische Konfiguration: ConfigMap ENV

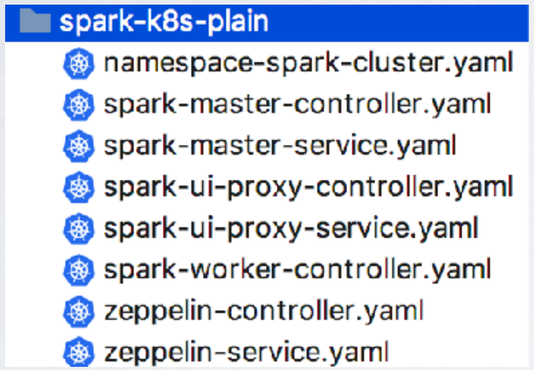
Dynamische Konfiguration: ConfigMap Files

Statische Konfiguration: Config Files im Container

PACKAGE MANAGEMENT

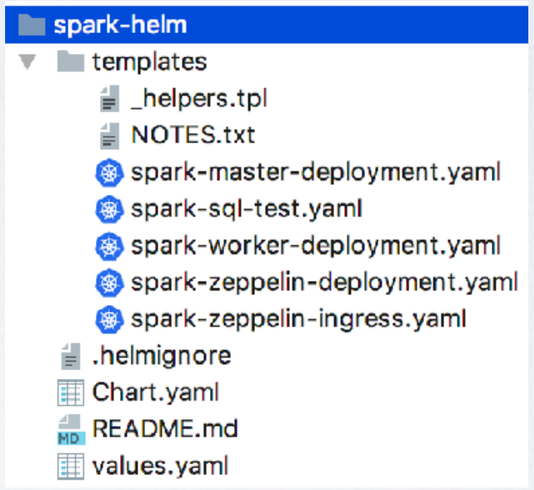


PATTERN



- `kubectl, kubectl, kubectl, …`
- Konfiguration?
- Endpunkte?

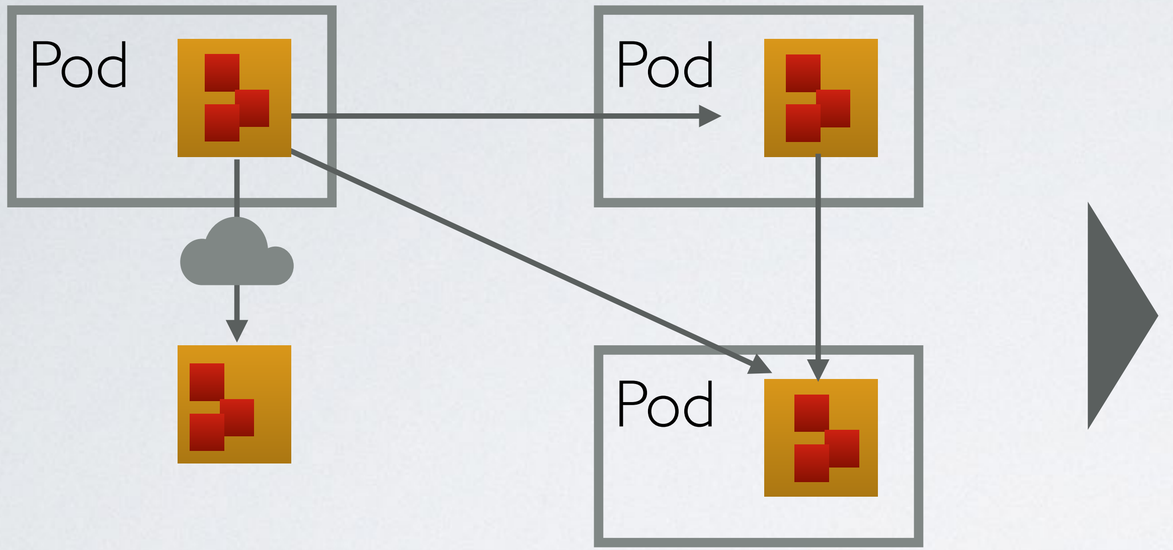
https://helm.sh

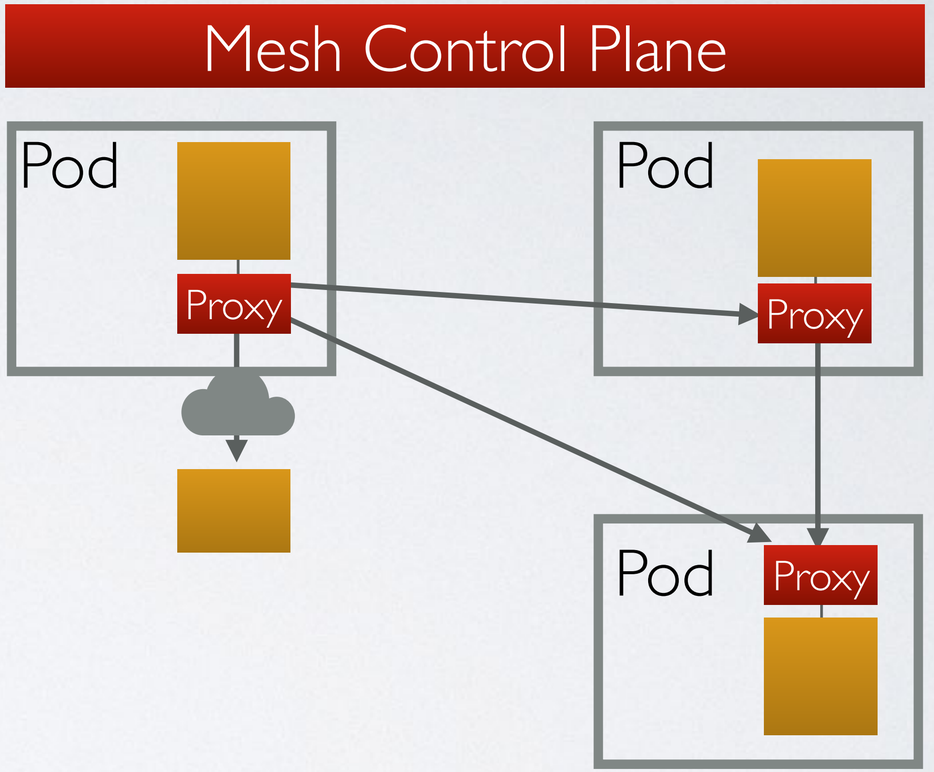


- Chart suchen auf https://hub.kubeapps.com
- Doku dort lesen (README.md)
- Konfigurationsparameter lesen:
  `helm inspect stable/spark`
- Chart starten mit überschriebener Konfiguration:

```
helm install --name my-release
     -f values.yaml stable/spark
```

SERVICE MESH



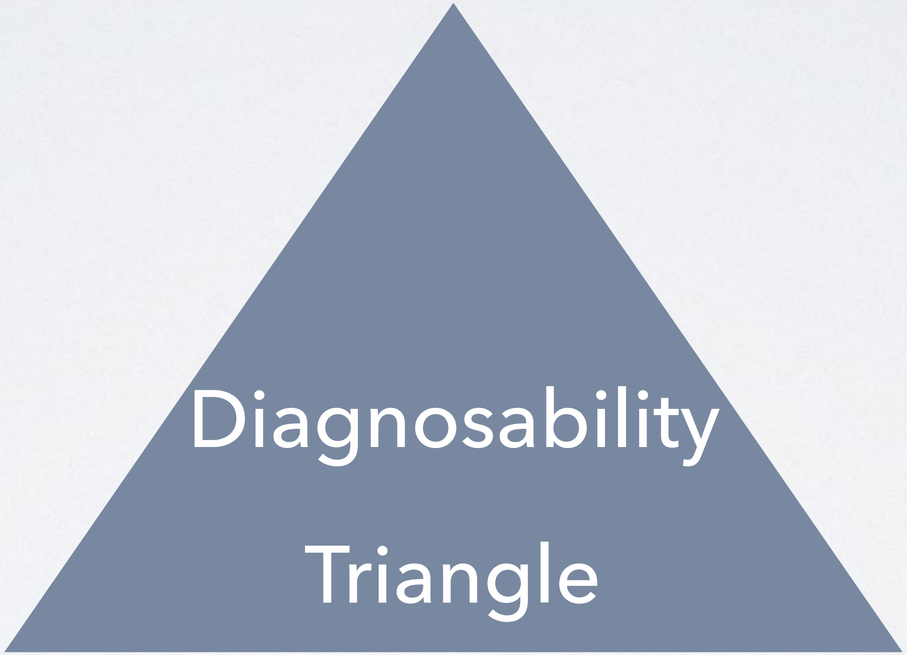


- Sichere Kommunikation (2-Way-SSL, Token Relay)
- Resilienz (Circuit Breaker)
- Policy Enforcement
- Diagnostizierbarkeit (Traces, Logs, Metriken)
- A/B Testing, Canary Releases
- Fault Injection, Latency Testing
- Location Transparency

- ohne Anpassung der Anwendungen!
- zentral gesteuert!

DIAGNOSABILITY TRIANGLE

Metrics

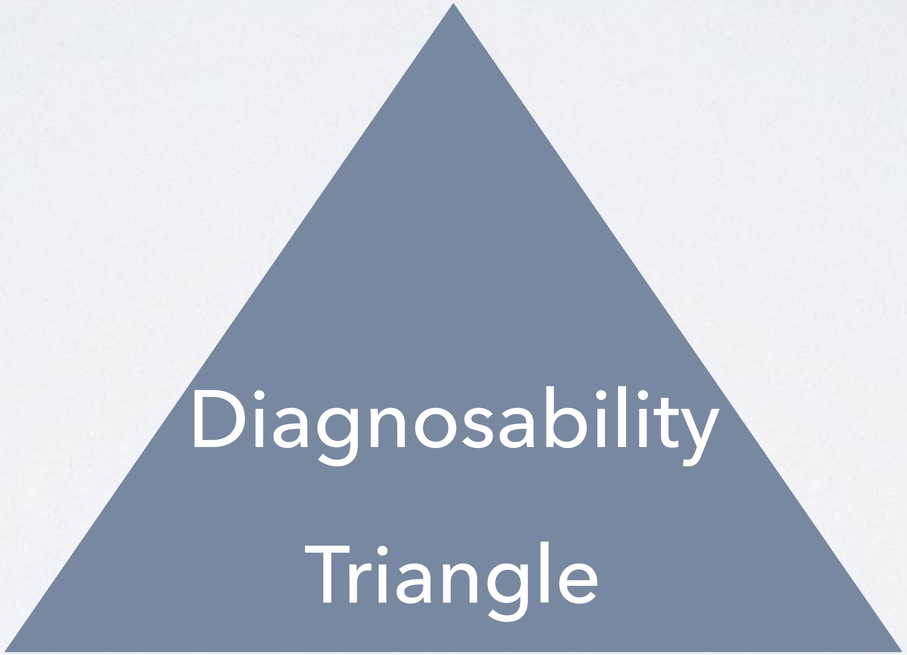


Traces

Events / Logs

DIAGNOSABILITY TRIANGLE

Metrics (RED = rate, errors, duration)



Distributed Traces

Events / Logs

OPERATORS

PATTERN

(Betriebsroboter)

Zustandsbehaftete Infrastruktur

(z.B. Datenbanken)

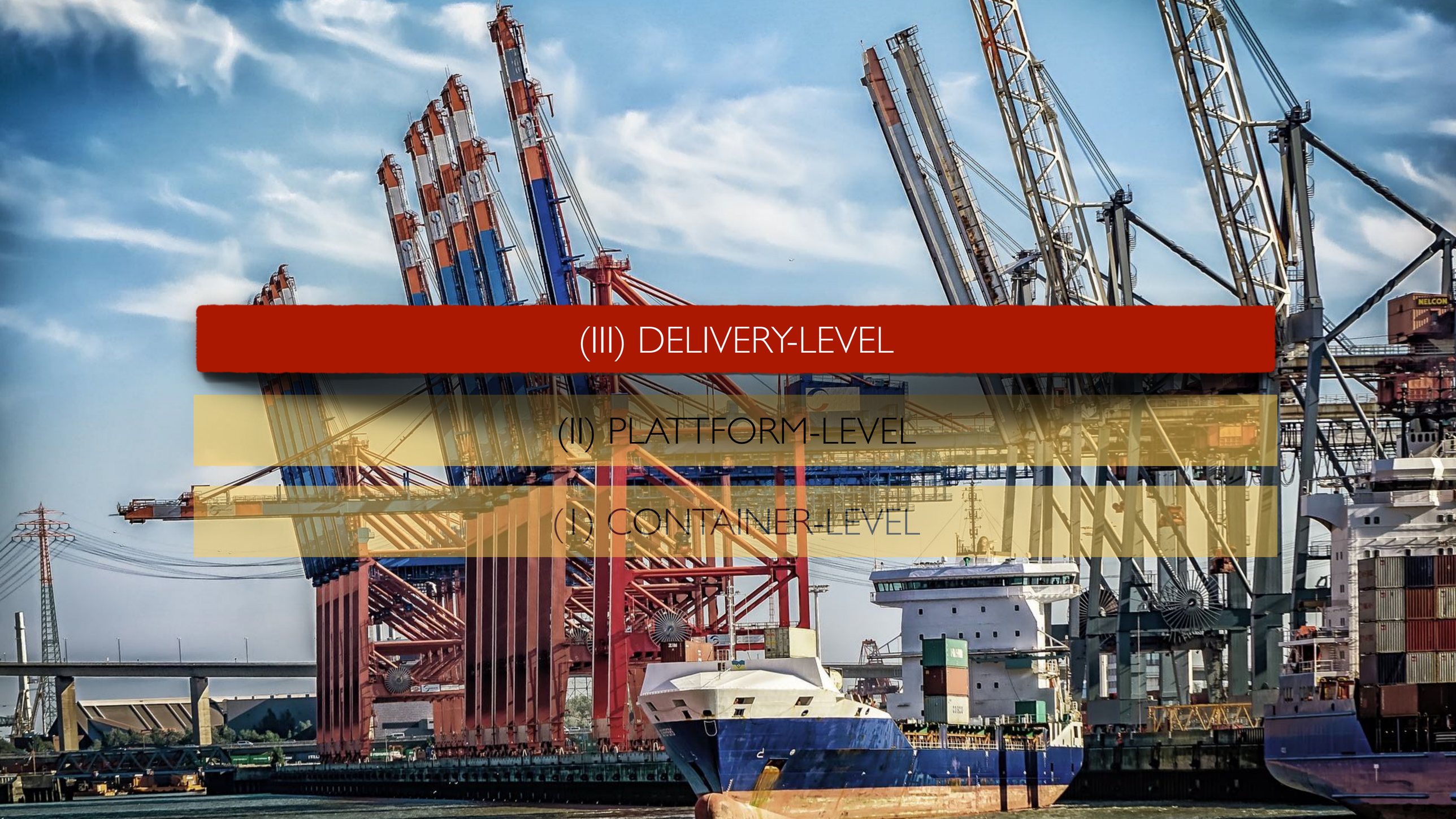
- Cluster Join / Leave
- Failover auslösen
- Fehlerzustände erkennen
- Migration von Daten

Standard-Betriebsprozeduren für Anwendungen

- Zertifikate einspielen
- Neue Konfiguration einspielen
- Komplexe Rollout-Szenarien
- Alerting

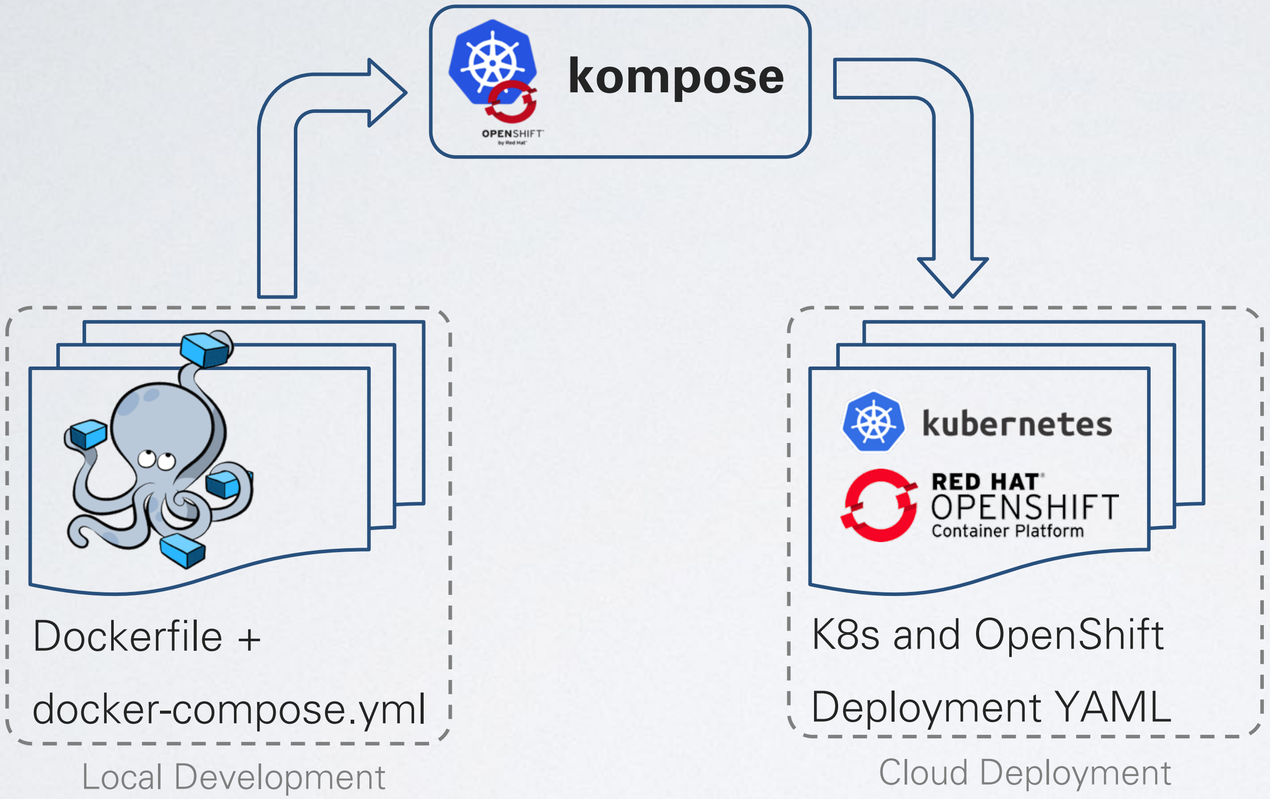
…

[1] https://coreos.com/operators
[2] https://kubernetes.io/docs/concepts/api-extension/custom-resources
[3] https://github.com/giantswarm/operatorkit
[4] https://github.com/sapcc/kubernetes-operators
[5] https://www.youtube.com/watch?v=cj5uk1uje_Y


(III) DELIVERY-LEVEL
(II) PLATTFORM-LEVEL
(I) CONTAINER-LEVEL

DEV ORCHESTRATION

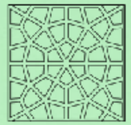
PATTERN



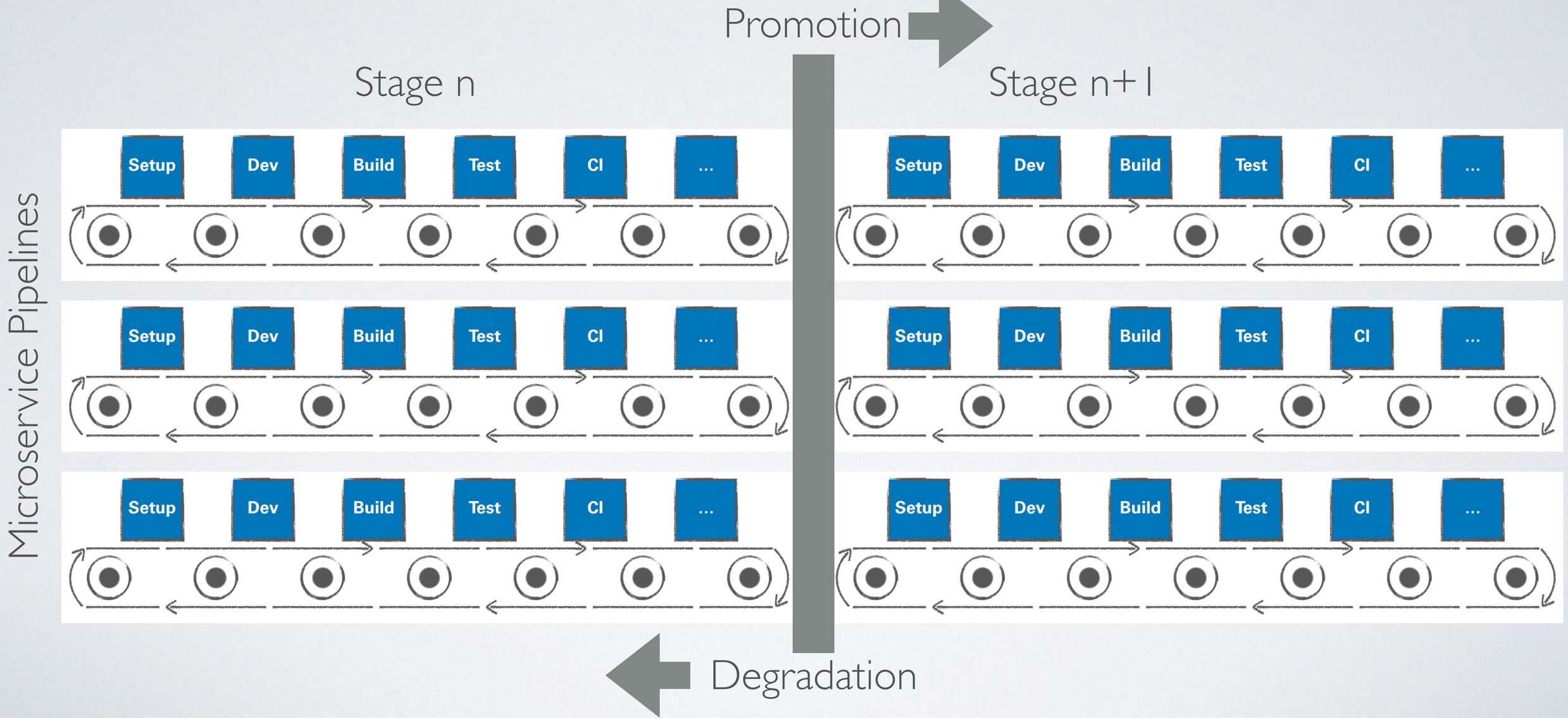
- schnelle Roundtrips
- geringe Komplexität
- lokales Troubleshooting

Bei einzelnen lokalen Prozessen, die in ein existierendes Remote-Cluster gehängt werden sollen: https://www.telepresence.io

PROMOTION



PATTERN



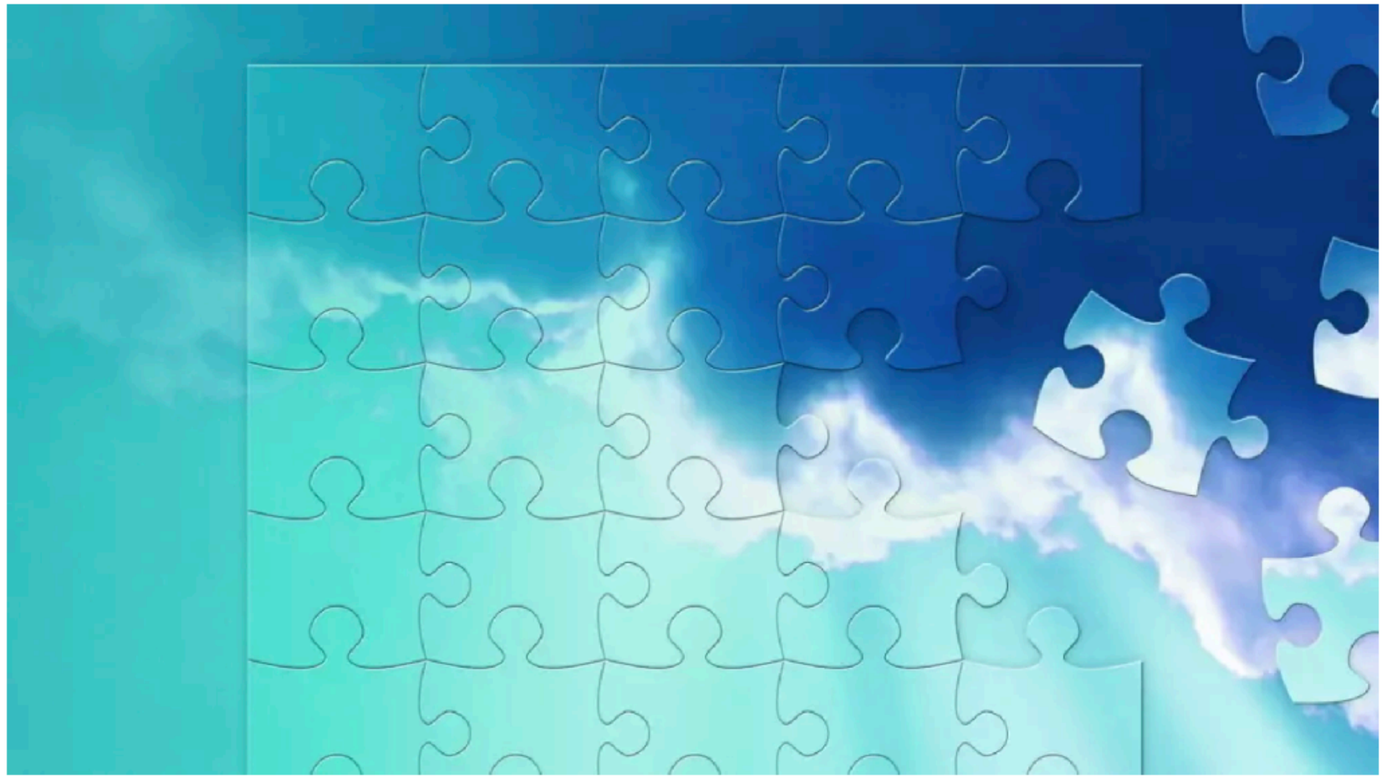
QUELLEN

- Generell
  - Container Patterns: https://l0rd.github.io/containerspatterns/#1
- Docker
  - DockerFile Best Practices: https://docs.docker.com/develop/develop-images/dockerfile_best-practices/#use-a-dockerignore-file
  - Docker Tools: https://github.com/veggiemonk/awesome-docker
  - OpenShift General Docker Guidelines: https://docs.openshift.com/enterprise/3.0/creating_images/guidelines.html
  - Common Docker Mistakes: https://runnable.com/blog/9-common-dockerfile-mistakes
- Kubernetes
  - Kubernetes Patterns: https://vimeo.com/233785743
  - Kubernetes Best Practices: https://www.youtube.com/watch?v=BznjDNxp4Hs
- Continuous Delivery
  - Continuous Delivery Patterns: https://continuousdelivery.com/implementing/patterns/

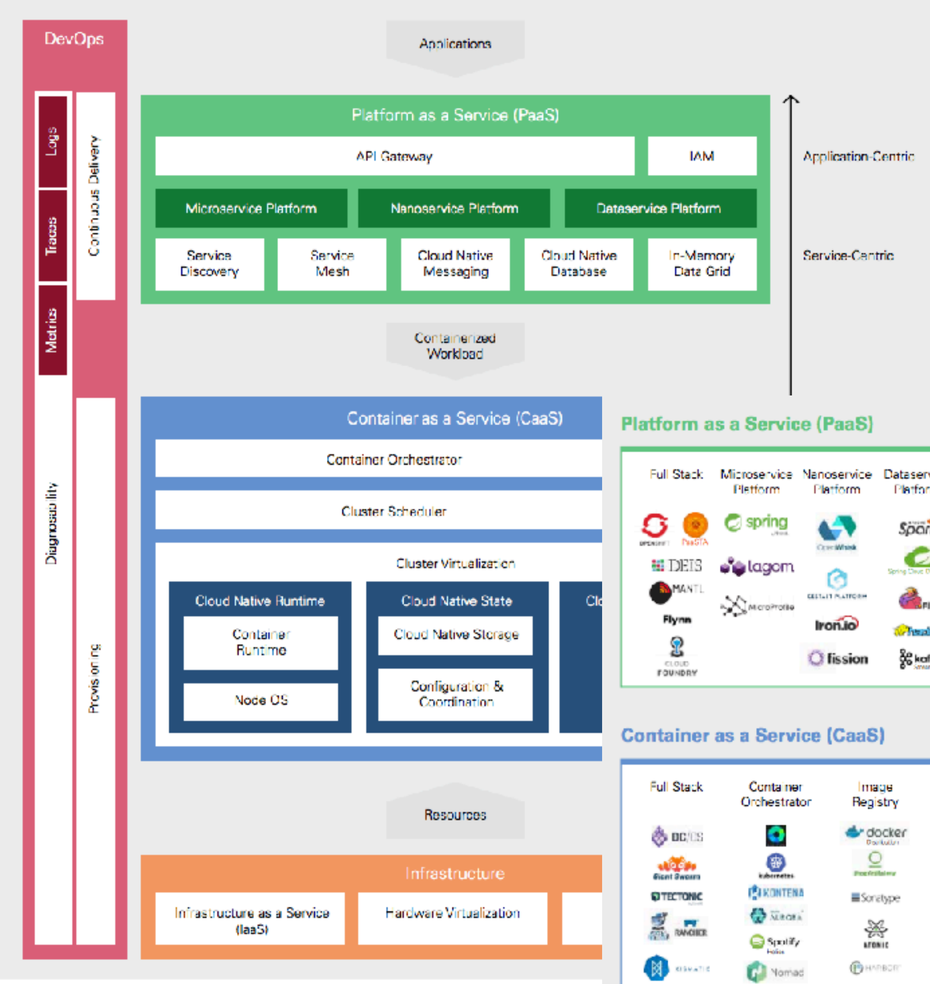
heise **Developer**

Ziel der Cloud Native News ist es, im regelmäßigen Abstand von wichtigen Ereignissen und Neuigkeiten im Cloud-Native-Ökosystem zu berichten und diese zu interpretieren, damit der Überblick gewahrt bleibt.

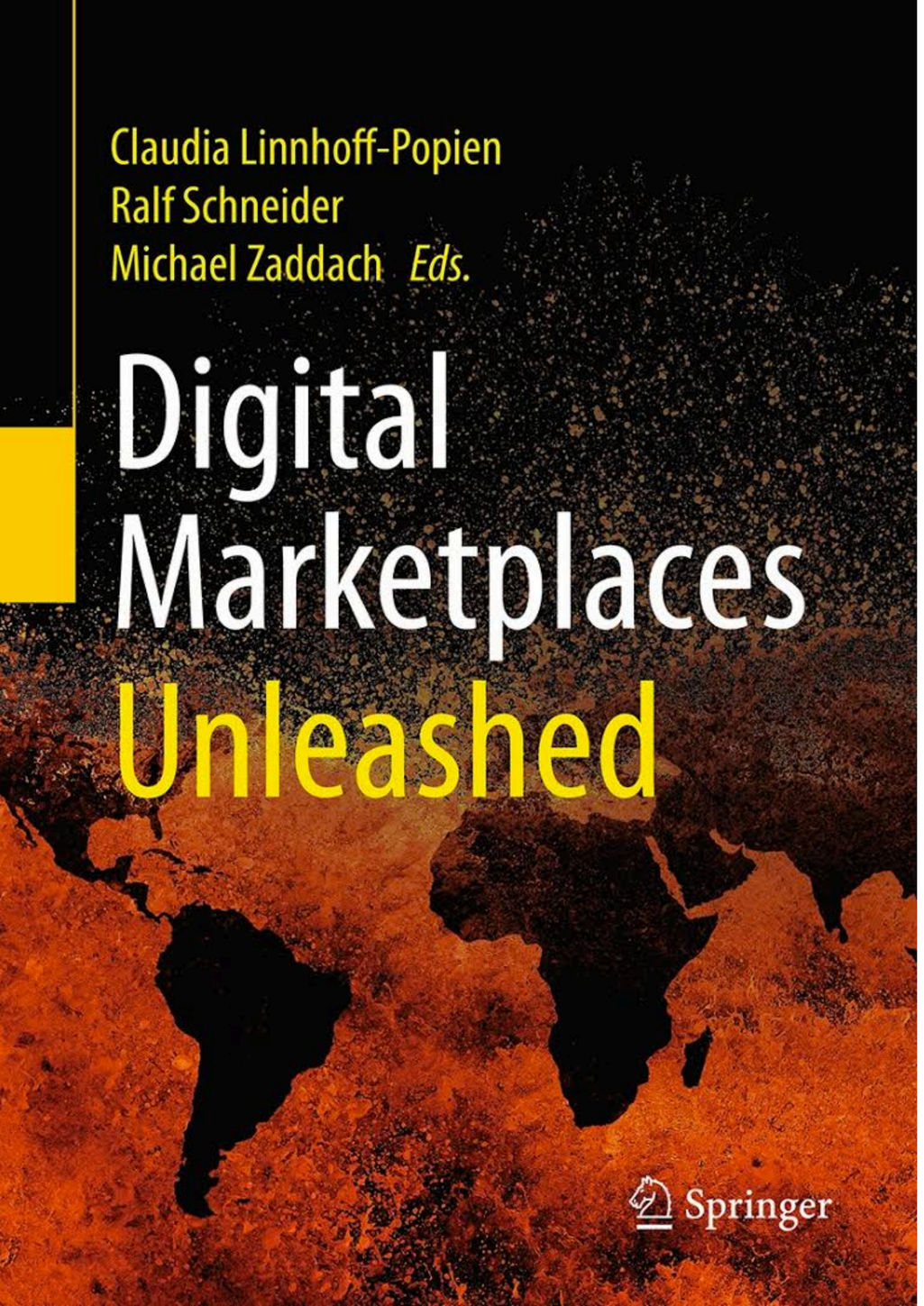
Know-how 26.01.2018 09:19 Uhr – Andreas Zitzelsberger (QAware), Josef Adersberger (QAware), Sebastian Scheele (Loodse) – 0 Kommentare



https://www.heise.de/developer/know-how-1033.html



http://www.qaware.de/news/java-magazin-cloud-native-universum



The Cloud Native Stack: Building Cloud Applications as Google Does

63

Josef Adersberger and Johannes Siedersleben

Abstract

Cloud giants like Google, Twitter or Netflix have released their core cloud technologies open source. The cloud pioneers' knowledge how to plan, build and run cloud applications are now accessible for free. Everyone can develop applications as scalable, as efficient and as resilient as Google's. This is called *GIFEE* (Google Infrastructure for Everyone Else), or more descriptively *Cloud Native Stack*. This stack is composed of cloud technologies open-sourced by cloud giants like *Kubernetes* from Google, *Mesos* from Twitter and the *Netflix OSS*. In this paper we describe the anatomy of the cloud native stack, map available technologies onto it and help decide when to move towards cloud native applications, gauging luring benefits and looming risks.

63.1 Cloud Native Disruption

The term *Digitalization* disguises the world's perplexity about the immense success and the hegemony of digital age companies – notably the *GAFA* gang (Google, Apple, Facebook, Amazon) which are often called *disruptors* for having disrupted classical industries such as retail, banking and travel. Other areas like insurances, logistics and mobility will be affected before long. The digital disruptors not only have had bright business ideas and good strategies to grow and monetize but have been clever at vastly improving non-

J. Adersberger (✉) · J. Siedersleben
QAware GmbH
Munich, Germany
e-mail: josef.adersberger@qaware.de

J. Siedersleben
e-mail: johannes.siedersleben@qaware.de

@adersberger

A Hitchhiker's Guide to Cloud Native Java EE

Mario-Leander Reimer

Dienstag, 13. März, 12:00 (40 min)

STOCK's 400 51

Enterprise & Microservices

https://www.qaware.de
https://slideshare.net/qaware
https://github.com/qaware

QAWARE
SOFTWARE ENGINEERING