

architektur SPICKER

Übersichten für die konzeptionelle Seite der Softwareentwicklung

MEHR WISSEN IN KOMPAKTER FORM:

Weitere Architektur-Spicker gibt es als kostenfreies PDF unter www.architektur-spicker.de

NR. 11

IN DIESER AUSGABE

- Kleine Stillkunde für Anwendungen in Containern
- Images bauen und Container betreiben
- Kubernetes (k8s) und Co.
- Migrationshilfe für bestehende Anwendungen



Container-Anwendungen entwickeln

Der Einsatz von Containern verspricht die immer größer werdende Komplexität der Anwendungslandschaft besser zu beherrschen. Dieser Spicker erklärt, wie Sie und Ihr Team Applikationen in Containern bauen und was Sie beachten müssen, um diese zu betreiben.



Worum geht's?

- Sie wollen Anwendungen in Containern betreiben. Welche Vorteile bringen diese bei welchem Architekturstil mit?
- Alte Anwendungen verharren noch in monolithischen Deployments oder schwergewichtigen Applikationsservern. Wie kriegen Sie diese schmerzarm in die „schöne neue Welt“?
- Um den richtigen Containereinsatz wird schon lange viel diskutiert. Wie setzen Sie Container heute richtig ein?
- Der Markt stellt so viele Orchestrierungslösungen bereit. Welche passt auf Ihre Situation am besten?

Anwendungsentwurf, Container und Orchestrierung



Die Zusammenhänge. Fragen und Antworten

Anwendungen in Containern sind ohne Hilfsmittel schwierig zu betreiben. Wie hängen die zentralen Begriffe zusammen? Was motiviert den Einsatz von Docker, Kubernetes und Co.?



1.

Warum zerlegen wir Anwendungen?

Die Zerlegung einer komplexen Anwendung macht sie wartbar und beherrschbar.

Ohne Zerlegung ließe sich z. B. nicht mit mehreren Teams gleichzeitig daran arbeiten.

Die Zerlegung kann zu mehreren Prozessen und einer verteilten Anwendung führen (Alternative: Monolith).

2.

Was haben Microservices damit zu tun?

Microservices sind ein Architekturstil, also eine grundlegende Art Anwendungen zu bauen.

Die einzelne Anwendung besteht dabei aus relativ kleinen, lose gekoppelten Services.

Jeder Microservice ist ein Prozess und unabhängig installierbar.

- Zu Architekturstilen siehe Seite 2
- Zu Microservices siehe Spicker #3

3.

Warum sollten wir Anwendungen in verschiedenen Prozessen laufen lassen?

So lassen sich Anwendungsteile gezielter skalieren.

Es erhöht die technologische Freiheit; Teile können unterschiedlich gebaut sein (z.B. Programmiersprachen).

4.

Und was sind Container?

Container sind vergleichbar mit VMs, aber kleiner.

Auf einem Betriebssystem können viele Container parallel laufen.

Container beinhalten jeweils nur eine Anwendung und alles, was es dazu braucht.

Container teilen sich die Ressourcen und sind gleichzeitig voneinander isoliert.

Docker ist die am Markt verbreitetste Container-Technologie.

5.

Wie helfen Container bei verteilten Anwendungen?

Container können Anwendungsteile effizient isolieren.

Sie bieten ein einheitliches Deployment-Format (Images) für unterschiedliche Technologien.

6.

Wie kommen Orchestrierungslösungen ins Spiel?

Mit Orchestrierungslösungen lassen sich Anwendungen einfacher definieren, strukturieren und konfigurieren.

Sie überwachen den Zustand der Anwendung und heilen bei Bedarf.

Ein Betrieb auf mehreren Rechnerknoten erhöht die Verfügbarkeit und eröffnet eine bessere Lastverteilung.

Kubernetes ist die am Markt verbreitetste Orchestrierungslösung.

→ Mehr zu Containern ab Seite 3

→ Mehr zu Kubernetes ab Seite 4

Anwendungsentwurf



Kleine Architekturstillkunde für containerisierte Anwendungen

Ein Architekturstil drückt ein grundlegendes Organisationsschema für Softwaresysteme aus. Hier finden Sie vier prominente Vertreter in der Softwarearchitektur inklusive des typischen Ansatzes mit Containern.



Slogan des Stiles, Leistungsversprechen



synonyme oder nah verwandte Stile, „auch“



strukturierendes Element („Bauteil“)



Beispiel-anwendung(en) für den Stil, Architektur-Ikonen



Umsetzung mit Containern und mögliche Vorteile dabei

Modulith



„Strukturiert gebaut, in einem Stück geliefert.“



(Deployment-)Monolith, als Anti-Pattern: Big Ball of Mud (wenn unstrukturiert)

Klare Strukturierung der Anwendung in Module, geplante Abhängigkeiten, einzelne Deployment-Einheit.



Modul (auch: Komponente, Subsystem)



Atlassian Confluence



Container-Ansatz: Ein Image als Deployment-Format für die komplette Anwendung.
 + Einheitliches Format für Auslieferung
 + ggf. Vereinheitlichung im Entwicklungsprozess/ Betrieb

Mehrschichtarchitektur



„Leicht zu portieren durch Abstraktion.“



N-Tier, Client-Server

Gliederung der Anwendung in technische und/oder logische Schichten. Zugriff jeweils nur auf die nächste (tiefere) Schicht.



Schicht (im Englischen: Layer, Tier)



SAP R/3, WordPress



Container-Ansatz: Einzelne Schichten in separate Container (z. B. komplette Geschäftslogik)
 + Einheitliches Format für Auslieferung
 + ggf. Vereinheitlichung im Entwicklungsprozess/ Betrieb
 + Bei geeigneter Implementierung (grob) skalierbar und aktualisierbar

Microservices



„Flexibilität durch kleine, unabhängige Teile“



Self-contained Systems (SCS)

Anwendung besteht aus einzeln deploybaren, sehr lose gekoppelten Services. Typisch: technologische Vielfalt



Service (auch: Vertikale, SCS)



Netflix, Xing (für SCS)



Container-Ansatz: Services jeweils als Container.
 + Bei geeigneter Implementierung fein skalierbar und aktualisierbar

Serverless



„Der erste wahre Cloud-Native-Stil“



Function as a Services (FaaS)

Geschäftslogik deploy als kleine, zustandslose Funktionen. Ressourcen-Verbrauch nur bei Aufruf/Ausführung.



Funktion



Alexa Skills



Verschiedene Container-Lösungen für Serverless z.B. in Kubernetes verfügbar. Container werden davon weg-abstrahiert.
 - Wirken On-Premises schwergewichtig und gefährden teilweise Serverless-Vorteile.
 + Lastspitzen mit Containern auffangen



Checkliste für den Anwendungsentwurf mit Containern

Beim Bau von Anwendungen in Containern haben sich Good Practices herauskristallisiert, um sie beispielsweise leicht betreiben, gezielt skalieren und unprovisionieren zu können. Und damit sie wenig Angriffsfläche bieten.

Gezielt skalieren

- Nur ein Anwendungsprozess pro Container
- Zustandslose Anwendungen bevorzugen
- Container robust runterfahren können

Leicht betreiben

- Log-Nachrichten auf stdout ausgeben
- Anwendungsüberwachung einsetzen
- Container robust runterfahren können

Sicherheit im Auge behalten

- Ausführung als root vermeiden
- Privilegierte Container vermeiden
- Anforderungen bzgl. Security von Anfang an mitdenken



→ Siehe auch Cloud-Prinzipien in Spicker #5 „Cloud-Anwendungen“ (u.a. 12-Factor-App).

Container und Images



Die wichtigsten Begriffe rund um Container

Kleines Container-Wörterbuch

Container Image: Paketierungsmöglichkeit für die Laufzeitumgebung einer Anwendung und aller von ihr benötigten Bibliotheken und Dateien. Es besteht aus mehreren Layern.

Container: Instanz eines Container Images, die isoliert betrieben werden kann

Layer: unveränderliche Schicht eines Images, die wiederum auf anderen Layer aufbaut. Jede Schicht enthält Befehle oder Dateien.

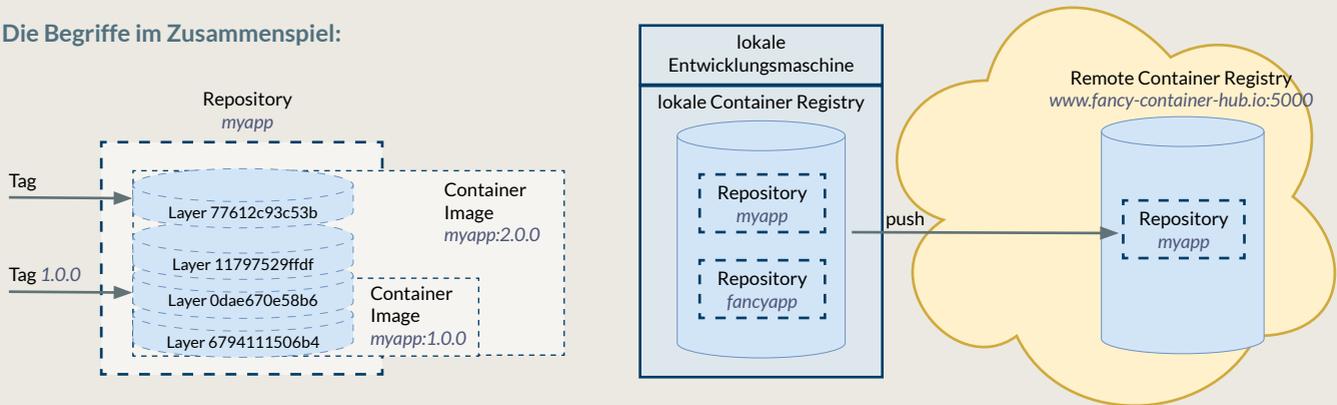
Repository: Speicherort für Images

Tags: Eine benannte Referenz auf ein bestimmtes Layer

Registry: Speicherort für Repositories, auf die auch andere zugreifen können

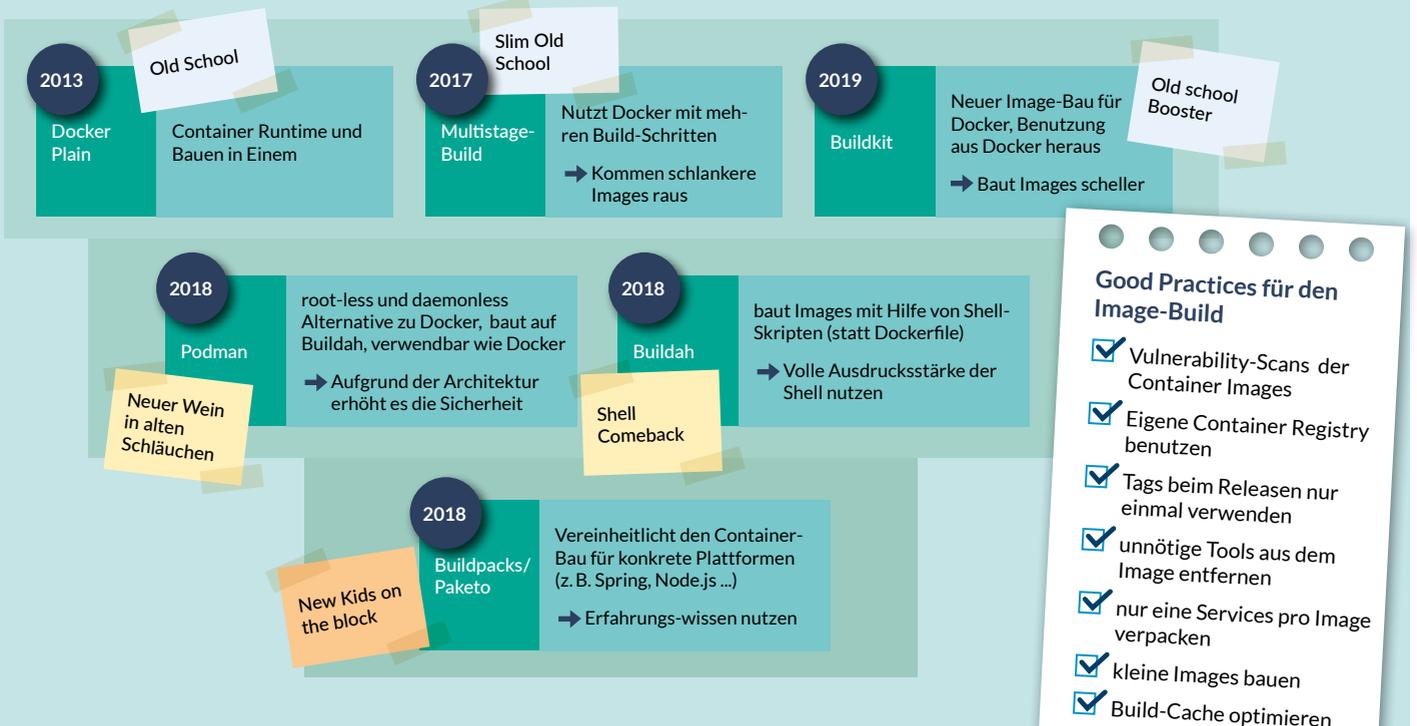
Namenschema für Container Images: ,Registry: Port/Repository:Tag', wobei nur der ,Repository'-Teil verpflichtend ist. Für die anderen Bestandteile greifen dann Default-Werte.

Die Begriffe im Zusammenspiel:



Wie baue ich heute Images?

Mit der Zeit sind verschiedene Werkzeuge zum Image-Bau entstanden. Hier die verbreitetsten im Überblick:

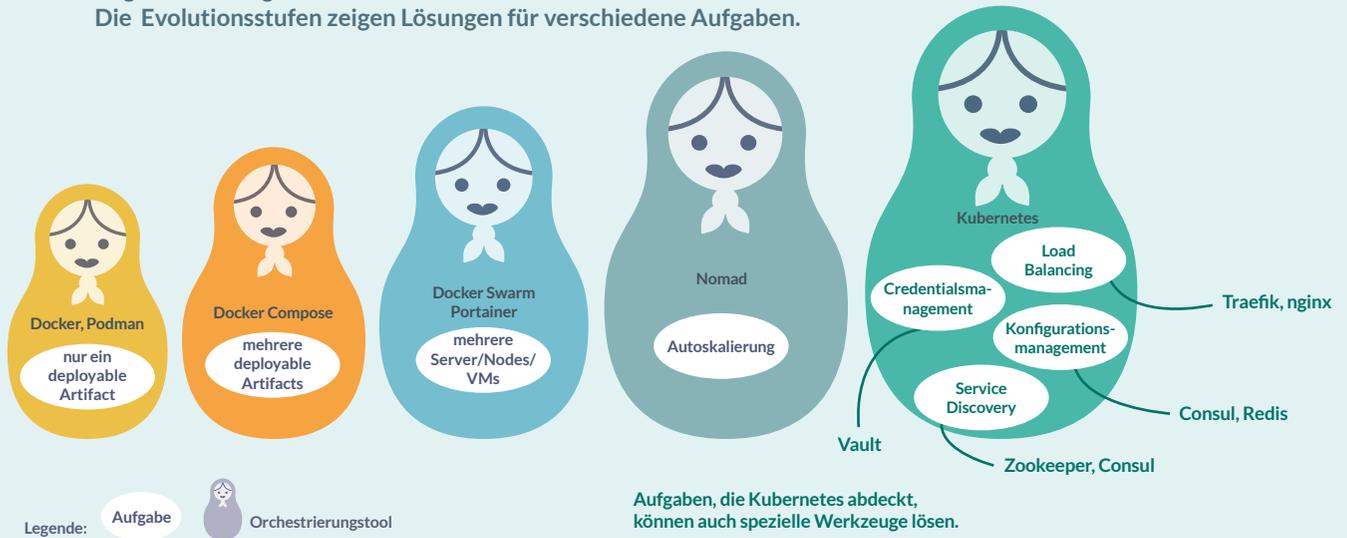


- ### Good Practices für den Image-Build
- Vulnerability-Scans der Container Images
 - Eigene Container Registry benutzen
 - Tags beim Release nur einmal verwenden
 - unnötige Tools aus dem Image entfernen
 - nur eine Services pro Image verpacken
 - kleine Images bauen
 - Build-Cache optimieren



Evolutionsstufen für Container-Orchestrierung

Es gibt viele Möglichkeiten Container laufen zu lassen. Die Evolutionsstufen zeigen Lösungen für verschiedene Aufgaben.



Orchestrierung mit Kubernetes



Kubernetes im Überblick

Kubernetes (kurz K8s) ist ein Open-Source-System zur Automatisierung der Verteilung

Einige Fakten

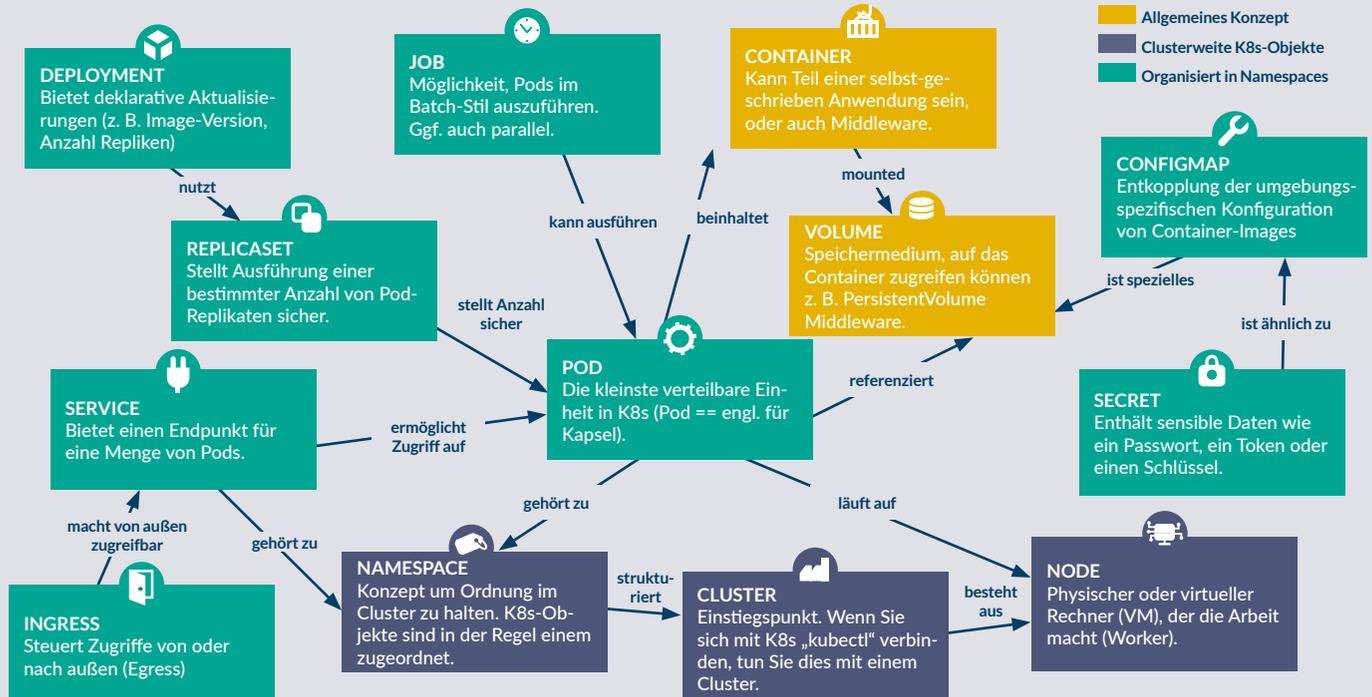
- Entwickelt in Go, verfügbar seit 2014
- K8s == K -ubernete- s (8 Buchstaben zwischen K und s)
- Initiiert von Google, das Erfahrung aus früheren, internen Projekten einfließen ließ
- Maintainer jetzt: CNCF (Cloud Native Computing Foundation)
- Sogenannte Distributionen „veredeln“ K8s, teilweise zu kommerziellen Produkten (Motto: „K8s für Unternehmen“)

Wesentliche Aufgaben

- containerisierte Anwendungen definieren, strukturieren und konfigurieren (Stichwort *.yaml)
- Anwendungen verteilen und aktualisieren (Deployment, Updates)
- Zustand der Anwendung überwachen und bei Bedarf heilen
- Lasten über die Infrastruktur verteilen (Skalierung)
- Kommunikation zwischen Anwendungsteilen leiten und absichern (Traffic Routing)

Ein Begriffsbild für Kubernetes

Das folgende Bild zeigt aus Sicht der Anwendungsentwicklung zentrale Begriffe von k8s im Zusammenspiel.



Wichtige k8s-Distributionen



Vanilla K8s

Die Basis für alles:



Was es von Hause aus bietet:

- Service discovery und Load Balancing
- Storage orchestration
- Automatische Rollouts und Rollbacks
- Automatic bin packing
- Selbstheilungs-Mechanismen
- Konfigurationsmanagement und Secret Management

Was fehlt und wo man sich selber drum kümmern muss:

- CI/CD Prozesse
- Keine Anwendungsdienste wie Datenbanken, Speichersysteme, Messaging-Dienste, Caching, etc
- Keine Logging, Monitoring und Alerting Lösungen
- Fortgeschrittene Systeme für Rechnerkonfigurationen, Wartungen Selbstheilung



On-Premises aka „Im eigenen RZ“

Wichtige Distributionen über Vanilla-k8s hinaus

Rancher

Erweitert Vanilla-k8s basierend auf CNCF-Tools. Open Source.

Enthält alle Vanilla-k8s Feature plus (Auswahl)

- Cross-Provider Cluster Deployment
- User management mit verschiedenen Auth-providern
- Web UI
- Integrierte CI/CD-Pipelines
- Projekt-Management Logging, Monitoring Alerting

Red Hat Openshift

Verwendet im Kern Kubernetes. Anpassungen/ Erweiterungen auch mit Non-CNCF-Tools. Enterprise Support gegen \$\$\$

Hauptunterschiede zu Vanilla-k8s

- Integriertes User-Management
- Integrierte Docker-Registry und CI-Pipelines
- Templates für Ressourcen
- Verwendung ähnlicher, leicht anderer Konzepte, z. B. Routers statt Ingresses, Projekte statt Namespaces, oc statt kubectl ...



Cloud aka Managed Kubernetes

Spezielle Distributionen von Public Cloud Anbietern

Amazon Elastic Kubernetes Service (Amazon EKS)

- basiert auf Vanilla-K8s
- eigene Implementierungen für z.B. Storage
- eigenes Tooling eksctl

Google Kubernetes Engine (GKE)

- basiert auf Vanilla-K8s
- built-in Security Scan
- Integration in die Google Cloud Platform (Logging, Monitoring)
- Auto-Pilot (Autoscaling der Nodes, Auto-Update etc.)

Azure Kubernetes Service (AKS)

- basiert auf Vanilla-K8s
- eigene Implementierungen für z.B. Storage
- Integration mit AD
- Windows-Container

Kubernetes für besondere Fälle

Für Anwendungen abseits des Mainstreams von Informationssystemen mit Weboberflächen finden sich spezialisierte k8s-Lösungen.

Hier eine Auswahl an Distributionen bzw. Frameworks

Serverless

- **Knative**, Serverless Container in Kubernetes <https://knative.dev>
- **Fission**, Serverless Framework für Kubernetes <https://fission.io>
- **OpenFaaS**, weiteres Serverless Framework für Kubernetes, auf Basis von Nats und Prometheus <https://www.openfaas.com>

Machine Learning

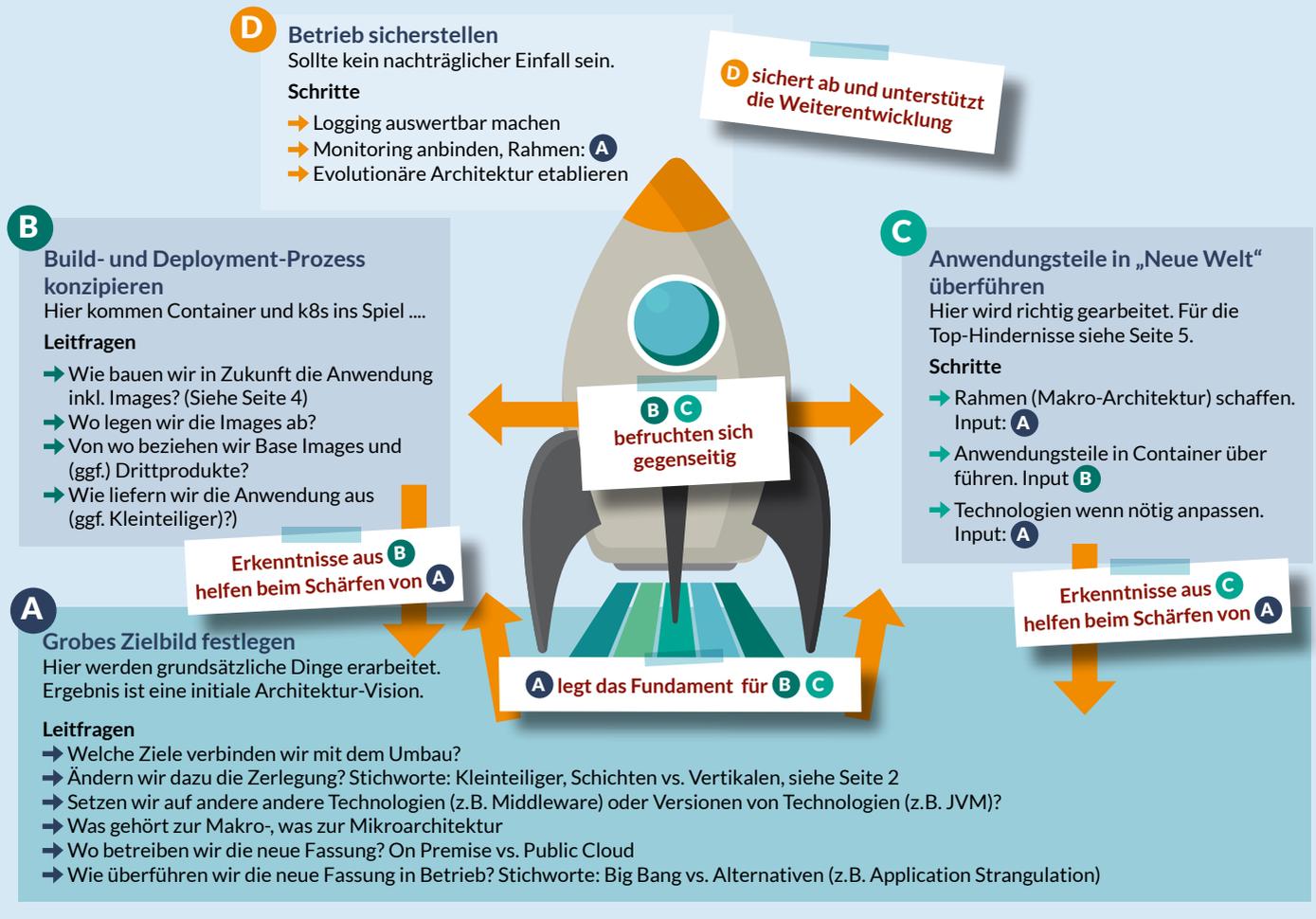
- **Kubeflow**, Machine Learning Toolkit für Kubernetes <https://www.kubeflow.org>

Mit kleinem Fußabdruck

- **minikube**, implementiert einen lokalen Kubernetes-Cluster auf macOS, Linux und Windows. Interessant vor allem für die Entwicklung. <https://minikube.sigs.k8s.io>
- **K3s**, sehr kleine Kubernetes-Distribution insbesondere für IoT und Edge-Computing <https://k3s.io/>

Migrations-“Plan“ Richtung Container und Orchestrierung

Sie beabsichtigen eine bestehende Anwendung zukünftig in Containern zu betreiben? Die folgenden vier Handlungsfelder geben Ihnen Orientierung. Sie durchlaufen diese iterativ, und schärfen dadurch das initiale Bild aus **A**



Weitere Informationen



Online-Artikel



Stefan Zörner: Architekturstile und -prinzipien in Zeiten von Containern und der Cloud, Informatik aktuell 2021



Sandra Parsick: Container-Images Deep Dive – 101 Wege zum Bauen und Bereitstellen, Informatik aktuell 2022



Bücher

→ **Bilgin Ibryam et al:** Kubernetes Patterns. Wiederverwendbare Muster zum Erstellen von Cloud-nativen Anwendungen, O'Reilly 2020

→ **Marko Luksa:** Kubernetes in Action, Manning Publications, 2. Auflage 2023



Die Autoren dieses Spickers

→ **Sandra Parsick** (info@sandra-parsick.de) berät Kunden rund um ihre Schwerpunkte Java-Enterprise-Anwendungen, Software Craftmanship und der Automatisierung von Softwareentwicklungsprozessen.

→ **Stefan Zörner** (stefan.zoerner@embarc.de) unterstützt Entwicklungsteams in Architektur- und Umsetzungsfragen mit dem Ziel, gute Lösungsansätze wirksam in der Software zu verankern.



Verwandte Architektur-Spicker

- Nr. 3: **Microservices**
- Nr. 5: **Cloud-Anwendungen**
- Nr. 7: **Continuous Delivery**

<https://www.architektur-spicker.de>

Wir freuen uns auf Ihr Feedback: spicker@embarc.de

<https://architektur-spicker.de>



<https://www.embarc.de>
info@embarc.de



<https://www.sigs-datacom.de>
info@sigs-datacom.de