edv-buchversand.de

GitOps

Das Praxisbuch für DevOps-Teams und Systemarchitekten

» Hier geht's direkt zum Buch

DAS VORWORT

Dieses Vorwort wird Ihnen von www.edv-buchversand.de zur Verfügung gestellt.

Vorwort

7.072.

Das ist nicht die Anzahl der Sekunden, die Sie brauchen werden, wenn Sie dieses Buch sorgfältig durcharbeiten möchten.

Sondern es ist die Anzahl der Seiten, die ich in den letzten 8 Jahren im Rahmen von diversen Publikationen zum Thema »Skalierbare Container-Infrastrukturen auf Kubernetes/OpenShift-Basis und GPU-beschleunigte KI/ML-Infrastrukturen im Kubernetes/OpenShift-Umfeld« zu Papier gebracht habe. Man könnte meinen, die Geschichte von Käpt'n Kube & Co wäre langsam auserzählt.

Ist sie aber nicht. Manchmal kommt es mir vor, als käme sie jetzt, dank eines immer größeren und effizienteren Ökosystems, erst richtig in Fahrt. Ein zunehmend wichtiger Teil davon sind CI/CD- und GitOps-Systeme, von denen vor allem der GitOps-Part im Hinblick auf Infrastructure as Code immer wichtiger wird. Der Grad an Automation in Operator-gestützten Self-Healing-Self-Provisioning-Systemen wächst, TTMs werden kürzer, die Systeme immer intelligenter und dank zentraler Schnittstellen sind wir von »Everything as Code« nicht mehr weit entfernt.

Ein Grund, sich des Themas anzunehmen.

Ich hoffe, dass ich Sie wie in meinen bisherigen Publikationen mit auf einen kurzweiligen, informativen und unterhaltsamen Ausflug in das spannende Thema nehmen kann. Und da Zeit eine immer kostbarere Ressource wird (wie jeder, der älter wird, feststellt), will ich Sie nicht mit langen Vorreden aufhalten, sondern nur wie üblich sagen:

Gehen wir ans Werk.

Kapitel 1
CI/CD und GitOps unter Kubernetes/OpenShift in Enterprise-Umgebungen

»Geschwindigkeit ist nur eine Frage des Geldes. Wie schnell willst du denn unterwegs sein?«
– aus »Mad Max«, 1979

Unternehmen stehen unter immer stärkerem Druck, Software schneller und zuverlässiger als je zuvor auszuliefern. Die Einführung von Containern und Cloud-nativen Plattformen wie Kubernetes hat Entwicklungs- und Betriebsabläufe grundlegend verändert.

Unternehmen, die die drei elementaren Säulen der Vollautomation unter Kubernetes/OpenShift für einen optimalen und kosteneffizienten Betrieb konsequent umsetzen, liegen im harten Wettbewerb klar vorne. Diese sind

- IaC: Infrastructure as Code,
- In-Cluster Automation mit Operatoren und
- Operator-gestützte CI/CD- und GitOps-Systeme.

In einer Welt, in der Software-Updates oder gar die Erstellung kompletter Infrastrukturen in Stunden statt in Wochen erwartet werden, stoßen traditionelle, manuelle Prozesse an ihre Grenzen. Lange Release-Zyklen, aufwendige manuelle Tests und fehleranfällige Deployments führen zu Frustration – bei den Teams ebenso wie bei den Nutzern. Gleichzeitig steigt der Wettbewerbsdruck: Wer Innovationen schneller und stabiler bereitstellen kann, schafft einen klaren Marktvorteil. Diese Realität hat dazu geführt, dass DevOps-Praktiken inzwischen zum De-facto-Standard avanciert sind. Die Branche hat erkannt, dass umfassende Automatisierung im Software-Lifecycle kein Luxus mehr ist, sondern zur Überlebensfrage werden kann.

Dabei sind die Pain Points allzu bekannt: Viele IT-Abteilungen kämpfen noch immer mit langsamen Lieferprozessen, in denen vom Code-Commit bis zum Rollout Wochen vergehen. Häufig liegt das an aufeinanderfolgenden manuellen Schritten – vom Testen bis zur Freigabe –, die jeden Ablauf verzögern. Die Folgen spürt das gesamte Unternehmen: Fachabteilungen warten frustriert auf neue Features, während Wett-

bewerber vielleicht schon das nächste Update ausrollen. Ein anderer Schmerzpunkt ist die hohe Fehleranfälligkeit manueller Deployments. Ein vergessenes Konfigurationsflag oder ein Tippfehler in einem Skript – und schon schlägt ein Deployment fehl oder, schlimmer, es kommt zu Produktionsausfällen. Menschliches Versagen wird so zur alltäglichen Risikoquelle. Hinzu kommen Intransparenz und mangelnde Nachvollziehbarkeit: Ohne automatisierte Pipelines und zentrales Git-Tracking ist oft unklar, welche Version einer Anwendung gerade in welcher Umgebung läuft. Changes gehen in E-Mail-Fluten unter, und im Ernstfall fehlt der Überblick: Welche Änderungen könnten den Fehler verursacht haben? Wo liegt der aktuelle Stand der Deployments? Dieses Gefühl des Blindflugs zehrt an den Nerven der Teams und erschwert zugleich die Compliance. Denn wenn nicht eindeutig nachvollziehbar ist, was wann von wem geändert wurde, leidet auch die Sicherheit – ein Albtraum für jede Organisation, die strenge regulatorische Vorgaben einhalten muss.

Moderne CI/CD- und GitOps-Systeme treten an, um genau diese Probleme zu lösen. Sie versprechen nicht nur technische Verbesserungen, sondern auch eine tiefgreifende Veränderung der Arbeitskultur: Weg von isoliert arbeitenden Teams und manuellen Firefights, hin zu einer kollaborativen, durch Automation unterstützten Lieferkette:

- *Continuous Integration (CI)* sorgt dafür, dass jeder Code-Änderungsschritt automatisch gebaut und getestet wird.
- *Continuous Delivery* und *Continuous Deployment (CD)* stellen sicher, dass diese Änderungen reibungslos und jederzeit bis in die Produktion fließen können.
- *GitOps* erweitert dieses Prinzip auf die Infrastruktur: Die gewünschte Systemkonfiguration wird in Git versioniert und automatisiert auf die laufenden Umgebungen angewendet. Und wenn eine Umgebung noch nicht vorhanden ist, wird sie per IaC automatisiert und mit minimalem Zeitaufwand jederzeit reproduzierbar erstellt.

Diese Ansätze emotionalisieren im positivsten Sinne – weil sie den Teams spürbar Last von den Schultern nehmen. Plötzlich wird Deployment vom Angstthema zur Routine. Eine Änderung am Freitagabend deployen? Mit einem zuverlässigen, automatisierten Pipeline-Prozess und vollautomatisierten Rollback-Verfahren im Fehlerfall verliert das seinen Schrecken. Die harte Grenze zwischen »Development« und »Operations« weicht auf; was bleibt, ist ein gemeinsames Verantwortungsgefühl für schnelle und stabile Auslieferung.

Der Impact einer solchen Transformation ist enorm: Zum einen beschleunigt sich die Liefergeschwindigkeit dramatisch. Durchgehende Automatisierung von Infrastruktur, Build, Test und Deployment ermöglicht es, Änderungen in Minuten statt Tagen bereitzustellen. Hochperformante Teams schaffen es heute, neue Code-Änderungen

im Schnitt binnen weniger Stunden bis zur Produktionsreife zu bringen – herkömmliche Prozesse benötigen dafür deutlich mehr Zeit.

Diese drastische Verkürzung der Deployment-Zeiten bedeutet: Features und Fixes erreichen den Endnutzer schneller, Feedback fließt früher zurück, neue Versionen werden besser optimiert. Zum anderen steigt die Effizienz, denn automatisierte Pipelines arbeiten rund um die Uhr, ohne Pausen und ohne die Ermüdungsfehler, denen Menschen unterliegen. Aufgaben, die früher ganze Nächte verschlangen – etwa das Bauen von Release-Artifacts, das manuelle Durchführen von Testskripten oder das Ausrollen auf zig Umgebungen –, laufen nun im Hintergrund ab. Das Team kann sich auf wertschöpfende Tätigkeiten konzentrieren, anstatt sich mit lästiger Release-Administration aufzuhalten. Damit sinken nicht zuletzt die Betriebskosten: Weniger manuelle Nacharbeit, weniger Ausfallzeiten und ein schlankerer Prozess bedeuten, dass personelle Ressourcen effektiver eingesetzt werden können. Jede Automatisierungsstufe spart Zeit und Geld.

Automatisierung erhöht gleichzeitig die Zuverlässigkeit und Sicherheit der Abläufe. Standardisierte Pipeline-Schritte stellen sicher, dass jeder Build und jedes Deployment nach bewährten Mustern abläuft – immer gleich, reproduzierbar und nachvollziehbar. Überraschungen werden zur Ausnahme. Durch automatische Tests, Code-Qualitätsprüfungen und Sicherheits-Scans in jeder Pipeline-Stufe werden Fehler oder Schwachstellen frühzeitig erkannt und gestoppt, bevor sie in Produktion gelangen. Manuelle Fehlerquellen werden so gut wie eliminiert. Die Folge: Deployments schlagen deutlich seltener fehl, und die Recovery im Problemfall erfolgt schneller. Aktuelle Untersuchungen unterstreichen diesen Effekt deutlich – Teams, die konsequent moderne CI/CD-Werkzeuge einsetzen, erzielen in allen zentralen Kennzahlen der Software-Auslieferung bessere Ergebnisse (State of CI/CD Report 2024 – CD Foundation; *https://cd.foundation/state-of-cicd-2024*). Effizienz, Sicherheit und Skalierbarkeit steigen messbar an, wenn sich Prozesse *end-to-end* automatisieren lassen.

Auch die Change-Mentalität wandelt sich: Kleine, frequente Updates sind weniger riskant als große, eher seltene *Big-Shifts*. Wenn doch einmal etwas schiefgeht, betrifft es nur ein kleines Inkrement, das sich leichter beheben oder zurückrollen lässt. Dieses »Fail Fast, Recover Fast«-Prinzip führt zu einer neuen Fehlerkultur – Probleme werden als Chancen zur Verbesserung angesehen, nicht als Katastrophen.

Ein weiterer entscheidender Aspekt moderner Pipeline-Systeme ist die Compliance-Automatisierung. In Branchen mit strengen Vorgaben (von Datenschutz bis Finanzregularien) war die Einhaltung von Compliance früher oft ein lähmender Prozess: händische Checklisten, manuelle Abnahmen, Dokumentationsmarathons. Mit dem Aufkommen von *Policy as Code* und eingebetteten Compliance-Checks in CI/CD-Pipelines lässt sich dieser Aufwand drastisch reduzieren. Jede Änderung durchläuft automatisierte Prüfungen – seien es Security-Scans auf Container-Images, statische Code-Analysen auf Schwachstellen oder Überprüfungen von Infrastrukturdefinitio-

nen gegen Unternehmensrichtlinien. Nichts gelangt mehr ungeprüft live. Die Pipeline fungiert als Gatekeeper, der nur konforme Artefakte durchwinkt. Das erhöht die Sicherheit enorm, ohne die Geschwindigkeit zu drosseln. Ganz im Gegenteil: Entwickler können schneller iterieren, weil sie wissen, dass die automatischen Kontrollen sie auffangen. Compliance wird vom Bremsklotz zum integralen, reibungslosen Teil des Workflows. Gleichzeitig entsteht eine lückenlose Audit-Historie – jedes Deployment und jede Konfigurationsänderung ist nachvollziehbar in Git dokumentiert. Für Entscheider bedeutet das: mehr Transparenz und Vertrauen, dass selbst bei hoher Release-Frequenz alle Vorgaben eingehalten werden.

Schließlich ermöglichen moderne DevOps- und GitOps-Ansätze den Sprung zu selbstheilenden Systemen. Besonders in containerisierten Umgebungen – allen voran Kubernetes – wird Infrastruktur zunehmend reaktiv und adaptiv. Fällt ein Dienst aus, startet die Orchestrierungsplattform ihn automatisch neu. Braucht eine Anwendung mehr Ressourcen, skaliert sie innerhalb von Sekunden hoch. Kombiniert man diese Fähigkeiten mit GitOps, entsteht ein System, das Abweichungen selbsttätig korrigiert. Ein Werkzeug wie *Argo CD* überwacht kontinuierlich den Soll-Zustand einer Applikation im Kubernetes-Cluster, der in Git hinterlegt ist. Stellt es fest, dass die laufende Umgebung vom definierten Zustand abweicht – sei es durch einen fehlgeschlagenen Deployment-Schritt oder eine manuelle Änderung außerhalb von Git –, greift es ein und steuert zurück auf Kurs. Die Infrastruktur »heilt« sich also selbst, indem sie sich stets am gewünschten Zielzustand ausrichtet. Für den operativen Betrieb bedeutet das: weniger nächtliche Einsätze, weniger hektische Fehlerbehebung von Hand. Probleme werden oft korrigiert, bevor sie überhaupt jemand bemerkt. Die Transparenz steigt dabei ebenfalls, da jederzeit klar ist, welcher Zustand als korrekt gilt und welche Änderungen vorgenommen wurden.

Um diese Vision Realität werden zu lassen, hat sich ein neues Ökosystem von Tools etabliert. Zwei prominente Vertreter, die in diesem Buch eine zentrale Rolle spielen, sind Tekton und Argo CD. Beide adressieren genau die genannten Herausforderungen, jedoch auf unterschiedlichen Ebenen der Pipeline:

- *Tekton* ist ein Framework für CI/CD-Pipelines, das Cloud-nativ für Kubernetes entwickelt wurde. Es ermöglicht, Build- und Release-Prozesse als Kubernetes-Objekte zu definieren und auszuführen. In einer OpenShift-Enterprise-Umgebung bildet Tekton die Grundlage von *OpenShift Pipelines* – das heißt, CI/CD-Jobs laufen direkt im Cluster, skalieren mit dessen Ressourcen und fügen sich nahtlos in die Container-Infrastruktur ein. Das Ergebnis: Die Pipeline selbst wird zu einem skalierbaren, hochverfügbaren Service, der genau wie die Applikationen versioniert und gemanagt werden kann.
- *Argo CD* hingegen fokussiert auf die CD-Seite mit GitOps: Es überwacht Git-Repositories auf Änderungen an Deployment-Deskriptoren (beispielsweise Kubernetes-Manifeste) und sorgt dafür, dass diese Änderungen automatisch und sicher

auf die entsprechenden Umgebungen ausgerollt werden. Für Kubernetes und OpenShift ist Argo CD damit ideal, weil es die Stärken des deklarativen Ansatzes dieser Plattformen nutzt – Deployments werden nachvollziehbar, rückrollbar und stimmen stets mit dem in Git definierten Zustand überein.

Beide Tools zusammen illustrieren den High-Level-Ansatz moderner Lieferketten: Tekton und Argo CD zeigen, wie Build, Test und Deployment in Container-Ökosystemen vollständig automatisiert und orchestriert ablaufen können, ohne dass Tiefe in technische Details verloren geht. Wichtig ist nicht, jede Featureschraube dieser Tools zu kennen, sondern zu verstehen, was sie ermöglichen: eine Welt, in der ein Commit in kürzester Zeit zum laufenden Service wird – überprüft, konform und stabil.

Kombiniert man – wie im Buch gezeigt – diese Tools mit weiteren Werkzeugen wie leistungsfähigen Policy-Engines und Secret Stores, der Cluster API und Crossplane als universeller IaC-Engine, ist der Schritt zu »Everything as Code« nur ein kleiner: Kubernetes und OpenShift werden zur zentralen und zuverlässigen Verwaltungsinstanz für komplette Infrastrukturen, auch außerhalb des eigenen Ökosystems.

Für erfahrene DevOps-Teams, Architekten und Entscheider liegt der Gewinn moderner CI/CD- und GitOps-Ansätze damit klar auf der Hand. Die realen Schmerzpunkte – träge Release-Zyklen, hohe Fehlerraten, unsichere Ad-hoc-Änderungen und unnötig hohe Kosten – können endlich systematisch adressiert werden. An ihre Stelle treten rasante Deployment-Zeiten, vertrauenswürdige und sichere Änderungen, transparente Prozesse und eine effizientere Ressourcen-Nutzung. Es geht nicht um Automatisierung um der Automatisierung willen, sondern um das Hebeln von Potenzialen: Teams können sich wieder auf Innovation konzentrieren statt auf Routineaufgaben, Führungskräfte gewinnen belastbare Kennzahlen und Kontrollmechanismen, und Kunden erhalten schneller bessere Produkte. Kurz: Effizienz, Flexibilität, Auditierbarkeit, Sicherheit und Skalierbarkeit steigen sprunghaft an – und mit ihnen die Zufriedenheit aller Beteiligten.

Diese Einleitung hat die groben Linien dieser Entwicklung skizziert. In den folgenden Kapiteln werden wir tiefer in die Materie eintauchen, Praxisbeispiele betrachten und zeigen, wie sich mit den benannten Werkzeugen die gerade beschriebenen Vorteile in Kubernetes- und OpenShift-Umgebungen tatsächlich realisieren lassen. Die Reise hin zu vollständig automatisierten, selbstheilenden und Compliance-sicheren Delivery-Plattformen hat begonnen – und sie verspricht, die Art und Weise, wie wir Software produzieren und ganze Systeme betreiben, nachhaltig zum Besseren zu verändern.

1.1 Vorbemerkungen

In diesem Buch finden sich vermehrt Anglizismen (die in der Regel auch erklärt werden), die sich im Rahmen aktueller und fachspezifisch fortgeschrittener Applikatio-

nen/Publikationen ohne kilometerlange und oft unpassende deutsche Um- und Beschreibungen oft nicht umgehen lassen.

1.1.1 Verwendete Formatierungen

Die in diesem Buch verwendeten Formatierungen schlüsseln sich auszugsweise wie folgt auf:

```
Kommandozeilenbefehl
```

```
Ausgabe (STDOUT/STDERR)
Listings von Konfigurationsdateien
Änderungen in Konfigurationsdateien
```

URLs

Dateien und Pfade

Screen-Element (wie z. B. Schaltflächen oder Elemente auf UIs)

Tasteneingabe: ↵

> **Wichtige Hinweise**
> Hinweistext

1.1.2 Weiterführende Hinweise

Ungeachtet der Tatsache, dass dieses Buch relativ umfangreich geworden ist, bieten viele der behandelten Themen noch mehr Stoff und Detailtiefe.

Um dieser Tatsache – wie auch der schnellen Evolution des Themas – gerecht zu werden, habe ich an den relevanten Stellen der jeweiligen Abschnitte wie üblich meist weiterführende Links zu den betreffenden Themen eingefügt.

Aufgrund der extrem schnellen Evolution kann leider nicht verbindlich sichergestellt werden, dass nach der Veröffentlichung des Buches noch jeder Link passt. Entsprechende korrespondierende Referenzen sollten dennoch jederzeit zu finden sein.

1.2 Kernziele und die rote Fäden

Das Buch orientiert sich an folgenden »roten Fäden« bzw. Kernzielen: Von den theoretischen Grundlagen geht es über strategische Betrachtungen hin zu CI/CD- und GitOps-Lösungen, die wir sukzessive von simplen Setups zu immer komplexeren Anwendungsfällen ausbauen, die realen Szenarien im Unternehmensumfeld entstammen. Der Schwerpunkt liegt dabei auf maximaler Vollautomation. Wie in meinen an-

deren Container-Publikationen konzentriert sich der Fokus auf die in produktiven Kubernetes-/OpenShift-Umgebungen unverzichtbaren drei Säulen der Vollautomation, die einen hocheffektiven, resilienten und kosteneffizienten Betrieb garantieren:

- IaC (Infrastructure as Code)
- In-Cluster-Automation mit Operatoren
- Operator-gestützte CI/CD- und GitOps-Systeme, die selbst IaC-Aufgaben wie den Rollout kompletter Cluster auf Basis der Cluster-API oder den automatischen Build und Rollout von komplexen Operatoren übernehmen können.

1.3 Zielgruppen und verwendete Systeme

Dieses Buch richtet sich als praxisorientierter Leitfaden an Admins, DevOps-Teams, Architekten und Entscheider und durchleuchtet neben praktischen Beispielen aus dem Bereich der CI/CD- und GitOps-Lösungen (mit Fokus auf Tekton und Argo CD) im Unternehmensumfeld auch die Hintergründe zu den behandelten Konzepten, Verfahren, Strategien und Tools. Betrachtungen aus der »8 Miles Above«- bzw. Vogelperspektive sollen Ihnen, ebenso wie einige Deep-Dives in bestimmte Techniken, ein besseres Verständnis ermöglichen.

Dieses Buch ist ausdrücklich kein Handbuch für Container-Neulinge oder ein Nachschlagewerk für Entwickler, das sich ausschließlich auf Dev-spezifische Belange fokussiert.

Achtung

Diese Auflage des Buches ist ausdrücklich nicht mehr für komplette Container-Neulinge geeignet.

1.3.1 Verwendete Systeme

Da dieses Buch auf den Unternehmenseinsatz fokussiert, kommt primär *OpenShift* (OCP) zum Einsatz, in der zum Zeitpunkt der Erstellung verfügbaren LTS-Version 4.18, mit OpenShift Pipelines 1.19.x (Tekton 1.0.x) und OpenShift GitOps 1.16.x/1.17.x (Argo CD 2.14.x / 3.0.x).

Daneben werden teilweise Kubernetes-basierte Umgebungen auf Basis von GKE 1.32 verwendet, mit Tekton in Version 1.x (Operator 0.76 LTS) und Argo CD 3.x.

Die CI/CD- und GitOps-Szenarien werden exemplarisch anhand der benannten Tool-Stacks *Tekton* (*OpenShift Pipelines*), *Argo CD* (*OpenShift GitOps*) sowie *Argo Rollouts* und anderen Tools, vorgestellt, da diese herstellerübergreifend mit allen Git-Plattformen zusammenarbeiten können.

Die Commandline Utilities wie **kubectl** oder **oc** werden in fast allen Beispielen per `alias` abgekürzt:

```
o = oc
k = kubectl
ow = "watch -d oc […]"
kw = "watch -d kubectl […]"
kns = "kubectl config set-context --current --namespace " (Namespace switch)
```

1.3.2 Erforderliche Vorkenntnisstände

Dieses Buch setzt solide Vorkenntnisse im Bereich Kubernetes/OpenShift voraus. Grundlagen zur Container-Technologie (»Was sind Pods, Deployments, Services, Ingress, PVs etc., wie funktioniert Kubernetes?«) werden als bekannt vorausgesetzt und in diesem Buch nicht erläutert. Kenntnisse in der Linux-Systemadministration und Sicherheit im Umgang mit der Linux-Commandline sind hilfreich.

1.3.3 Gescriptete Setups

Begleitend zu einigen Themen, insbesondere im Bereich CI/CD und GitOps unter Vanilla Kubernetes, finden Sie gescriptete Setups, die zum einen das Setup erleichtern und Ihnen zum anderen eine bessere Reproduzierbarkeit an die Hand geben.

1.3.4 Betrachtete Stände

Auch wenn es an manchen Stellen im Text nicht explizit gesagt wird, gilt für alle Betrachtungen bzw. Feststellungen implizit immer »im betrachteten Stand«, d. h., sie beziehen sich auf die Software-Pakete-/Versionen, Applikationen, Features etc., die während der Erstellung des Buches aktuell waren.